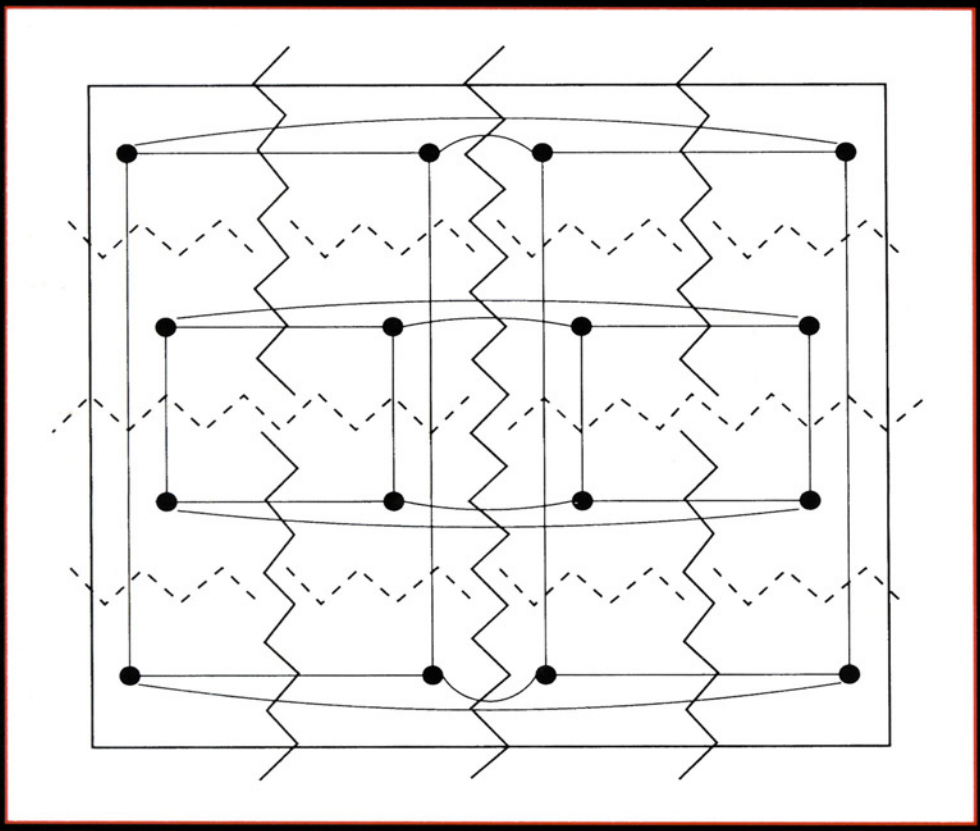
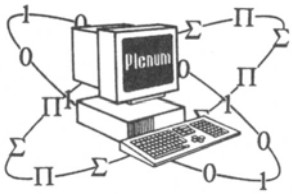
Frontiers of Computer Science

Graph Separators, with Applications



Arnold L. Rosenberg
and
Lenwood S. Heath

Graph Separators, with Applications



FRONTIERS OF COMPUTER SCIENCE

Series Editor: Arnold L. Rosenberg
University of Massachusetts
Amherst, Massachusetts

ASSOCIATIVE COMPUTING: A Programming Paradigm for Massively Parallel Computers
Jerry L. Potter

INTRODUCTION TO PARALLEL AND VECTOR SOLUTION OF LINEAR SYSTEMS
James M. Ortega

PARALLEL EVOLUTION OF PARALLEL PROCESSORS
(A book in the *Surveys in Computer Science* series, Edited by Larry Rudolph)
Gil Lerman and Larry Rudolph

GRAPH SEPARATORS, WITH APPLICATIONS
Arnold L. Rosenberg and Lenwood S. Heath

A Continuation Order Plan is available for this series. A continuation order will bring delivery of each new volume immediately upon publication. Volumes are billed only upon actual shipment. For further information please contact the publisher.

Graph Separators, with Applications

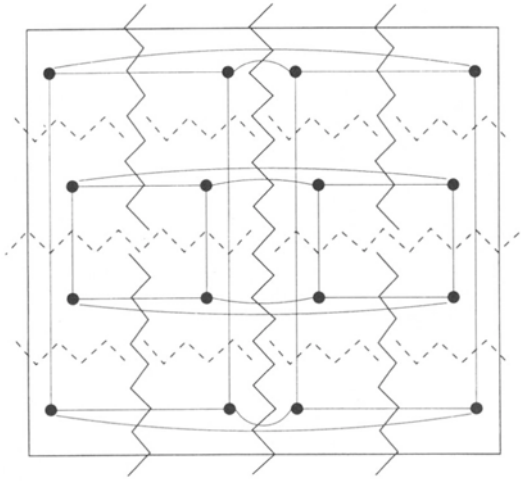
Arnold L. Rosenberg

University of Massachusetts
Amherst, Massachusetts

and

Lenwood S. Heath

Virginia Polytechnic Institute
Blacksburg, Virginia



KLUWER ACADEMIC PUBLISHERS
NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW

eBook ISBN: 0-306-46977-4
Print ISBN: 0-306-46464-0

©2002 Kluwer Academic Publishers
New York, Boston, Dordrecht, London, Moscow

Print ©1999 Kluwer Academic / Plenum Publishers
New York

All rights reserved

No part of this eBook may be reproduced or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without written consent from the Publisher

Created in the United States of America

Visit Kluwer Online at: http://kluweronline.com
and Kluwer's eBookstore at: http://ebooks.kluweronline.com

Preface

Theoretical computer science is a mathematical discipline that often abstracts its problems from the (hardware and software) technology of "real" computer science. When these problems are solved, the results obtained often appear in journals dedicated to the motivating technology rather than in a "general-purpose" Theory journal. Since the explosive growth of computer science makes it impossible for anyone to stay up to date in all areas of the field, many widely applicable theoretical results never get promulgated within the general Theory community, hence get re-proved (and republished) numerous times, in numerous guises. When a subject area develops a sufficiently rich, albeit scattered, mass of results, one can argue that the Theory community would be well served by a central, theory-oriented (rather than application-oriented) repository for the mass of results. The present book has been written in response to our perception of such need in the area of graph separators. This need is all the more acute given the multitude of notions of graph separators that have been developed and studied over the past (roughly) three decades. The need is absolutely critical in the area of lower-bound techniques for graph separator, since these techniques have virtually never appeared in articles having the word "separator" or any of its near synonyms in the title.

Graph-theoretic models naturally abstract a large variety of computational situations. Among the areas that give rise to such models are the problems of finding storage representations for data structures, finding efficient layouts of circuits on VLSI chips, finding efficient structured versions of programs, and organizing computations on networks of processors. In addition, numerous specific computational problems, say involving decomposition of problem domains, can fruitfully be formulated as problems of manipulating and/or partitioning graphs in various ways, including myriad problems that employ the well-known divide-and-conquer

paradigm. A striking feature of all of the cited areas is that they exploit the same major structural feature of their graph-theoretic models, namely the decomposition structure of the graphs as embodied in various notions of *graph separator*. All variations on the theme of graph separation involve removing either edges or nodes from the subject graphs in order to chop each graph into subgraphs—usually, but not always, disjoint—whose sizes must be within certain prespecified absolute or relative bounds. In all of the cited areas, the complexities of either procedures (e.g., algorithm timing) or structures (e.g., circuit areas) can be bounded by bounding the sizes of graph separators. Although we do not have the machinery to be formal, or even precise, at this stage of the exposition, we can describe at an intuitively evocative level a couple of scenarios that benefit from abstractions involving graph separators.

Consider first the problem of laying integrated circuits out on the chips that control our watches, calculators, computers, washing machines, cars, etc. The hallmark of integrated-circuit technology is that the world of integrated circuits is populated by only two types of objects, transistors and the wires that interconnect them. (The capacitors, resistors, etc., of the days of yore have all been replaced by transistors that can play multiple roles.) Thus, integrated circuits almost cry out to be viewed as graphs: transistors become nodes, and wires become edges.* Two problems that loom large in the layout of integrated circuits are the dearness of silicon real estate—chips are small—and the slowness of long wires—there are definite physical limitations on the speed of signal propagation. We shall see in Section 2.4 that one can obtain good upper and lower bounds on the amount of silicon needed to implement a given circuit design and on the length of the longest wire in the implementation by analyzing the separation characteristics of the graph that abstracts the circuit.

Our second example concerns programs in a procedural programming language. It has long been the practice in the design of compilers and other devices for mapping programs into computers (e.g., assemblers, schedulers) to represent a program awaiting mapping by a set of graphs that represent the flow of data and/or control and/or "communication" in the program. A typical control-flow graph, for instance, views each straight-line block of code in the program as a node in a graph and views deviations from straight-line flow of control as arcs that interconnect the nodes; for instance, a k-way branch would engender k arcs, each leading from the block that contains the branch to one of the blocks that branch might lead to. (One

* Our discussion is only a first-order approximation to reality, in that it ignores the "multipoint" nets" that are used in advanced circuit designs. However, the preponderance of "two-point" nets in circuits renders our approximation a valuable one.

might want to refine the blocks so that all arcs enter a block at its top.) A typical data-flow graph or communication graph might begin with a partition of the program into node-chunks (called tasks) and install arcs that originate at task-nodes in which a variable x is defined or modified and end at a task-node in which x is used with no intervening modification. We shall see several examples in Chapter 2 of how various mapping problems for programs can be solved efficiently if—and sometimes, only if—the graph(s) associated with the program can be recursively decomposed efficiently, i.e., the graph(s) have small separators. Section 2.2 uses an efficient recursive graph decomposition to craft an efficient divide-and-conquer implementation of an abstract program; Section 2.3 uses an efficient recursive graph decomposition to map the communication structure of a program efficiently onto the interprocessor communication network of a parallel computer; Section 2.6 uses the efficiency of a graph's decomposability to bound the number of memory registers that must be available in order to execute the program with maximum efficiency.

The current book is devoted to techniques for obtaining upper and lower bounds on the sizes of graph separators, upper bounds being obtained via decomposition algorithms. While we try to survey the main approaches to obtaining good graph separations, our main focus is on techniques for deriving *lower bounds* on the sizes of graph separators. This asymmetry in focus reflects our perception that the work on upper bounds, or algorithms for graph separation, is much better represented in the standard Theory literature than is the work on lower bounds, which we perceive as being much more scattered throughout the literature on application areas. A secondary motive is the first author's abiding personal interest in lower-bound techniques, which allows this book to slake a personal thirst.

The book is organized in four chapters and an appendix. Chapter 1 gives a technical overview of the graph theory that we need in order to study the lower-bound techniques of interest. We survey there the various types of graph separators that have been studied and their relationships. We introduce families of graphs that have proven important in many of the problem areas mentioned. We then introduce two technical topics that are needed to develop or appreciate the lower-bound techniques: we introduce the field of *graph embeddings,* which is at once a client of the techniques we develop and a facilitator of those techniques; and we introduce the notion of *quasi-isometry* of graphs, which is a formal notion of equivalence of graphs "for all practical purposes." Chapter 2 surveys a number of problem areas that have important abstractions to graph-theoretic problems that center on graph separation. This chapter should help motivate the reader for the highly technical development of the chapters on upper- and lower-bound techniques. Chapters 3 and 4 respectively, introduce and develop, the

upper- and lower-bound techniques that are our major focus. As we develop the techniques, we illustrate their application to the popular graph families of Chapter 1. Chapter 3, on upper bounds, can be viewed as an overview of the field with pointers to later and more specialized developments. Chapter 4, on lower bounds, covers that aspect of the field almost exhaustively, as of the date of the book's completion. Finally, Appendix A is somewhat a reprise of Chapters 2, 3, and 4, in that it illustrates how the separator-oriented techniques of Chapters 3 and 4 apply to the applications surveyed in Chapter 2. We hope that this sampler of applications of the abstract development will suffice to illustrate how the techniques can be brought to bear on a large range of the problem areas mentioned. Throughout, we have attempted to make the coverage adequate for the expert and the exposition careful enough for the novice. Thus, we hope that the book will prove useful as both a reference and text. Toward this end, we conclude each chapter with an annotated list of references to the literature. Most obviously, we cite the sources where the material we cover originated; in addition, though, we list a variety of sources whose material does not appear in the book; indeed, we list many sources that are only indirectly relevant to our subject, in the hope of fanning whatever flames of interest we have been able to kindle in the reader.

We share credit for whatever quality the reader perceives herein with many people. First, and foremost, no words suffice to express our debt to our collaborators, whose work—over a period spanning literally decades—is inextricably imbedded in the technical developments in this book. While we wish to avoid listing these numerous friends and colleagues explicitly, for fear of inadvertently omitting one, three stand out so prominently for the first author that they must be mentioned. My long-standing collaboration with Sandeep Bhatt, Fan Chung, and Tom Leighton, for well over 15 years, has so profoundly influenced my research that their influence touches virtually every word of this book. Next, we are grateful to the many colleagues (and their various publishers) who graciously permitted us to paraphrase excerpts from their technical papers. We owe special thanks to the first author's former students Fred Annexstein, Miranda Barrows, Bojana Obrenić, Vittorio Scarano, and Julia Stoyanovich for their careful reading of portions of various versions of this work; many improvements to the original presentation are due to them. Finally, we thank all of the (present and former) students at Duke University, the University of Massachusetts at Amherst, the University of North Carolina, Virginia Tech, and the Technion (Israel Institute of Technology) who suffered with patience and good will through seminars and courses in which the material herein was developed, sharing myriad helpful comments and suggestions. While we

thank all of these, we acknowledge sole responsibility for the errors that inevitably escape detection in large works.

We thank the companies and agencies that have supported both the research that enabled this project and the preparation of the book. We thank the International Business Machines Corporation, where much of the first author's early research was done; the National Science Foundation for continuing support for more than 18 years; the Lady Davis Foundation for support in spring 1994, when much of the first author's initial writing was done; and Tellcordia Technologies, which nurtured the multiyear Bhatt–Chung–Leighton–Rosenberg collaboration.

Finally, we thank our wives, Susan and Sheila, for their support throughout the period during which this book was written: especially for putting up with the mental absence that seems inevitably to accompany immersion in a large intellectual project.

Arnold L. Rosenberg
Amherst, Massachusetts

Lenwood S. Heath
Blacksburg, Virginia

This page intentionally left blank

Contents

1. A Technical Introduction **1**

1.1. Introduction 1
1.2. Basic Notions and Notation 2
1.3. Interesting Graph Families 4
1.4. Graph Separators 12
1.5. Graph Embeddings 27
1.6. Quasi-Isometric Graph Families 33
1.7. Sources 44

2. Applications of Graph Separators **47**

2.1. Introduction 47
2.2. Nonserial Dynamic Programming 49
2.3. Graph Embeddings via Separators 53
2.4. Laying Out VLSI Circuits 68
2.5. Strongly Universal Interval Hypergraphs 82
2.6. Pebbling Games: Register Allocation and Processor Scheduling 92
2.7. Sources 94

3. Upper-Bound Techniques **99**

3.1. Introduction 99
3.2. NP-Completeness 101
3.3. Topological Approaches to Graph Separation 109
3.4. Geometric Approaches to Graph Separation 121
3.5. Network Flow Approaches to Graph Separation 130

3.6. Heuristic Approaches to Graph Separation 147
3.7. Sources 156

4. Lower-Bound Techniques 159

4.1. Overview of Lower-Bound Techniques 159
4.2. Packing Arguments for Bounding Separation-Width 162
4.3. Congestion Arguments for Bounding Separation-Width 188
4.4. A Technique for Complete Trees 209
4.5. Information-Transfer Arguments 218
4.6. Sources 223

Appendix A. Applications of Graph Separators, Revisited 227

A.1. Introduction 227
A.2. Graph Embeddings via Separators 227
A.3. Laying Out VLSI Circuits 232
A.4. Strongly Universal Interval Hypergraphs 235
A.5. Pebbling Games 239
A.6. Sources 240

Bibliography 241

About the Authors 251

Index 253

1

A Technical Introduction

1.1. Introduction

The world of computing is heavily populated with graphs. Graph-theoretic models naturally abstract a large variety of computational situations. It is impossible to enumerate all of the areas that give rise to such models, but included among them are the problems of finding storage representations for data structures[1] (DeMillo *et al.* [1978], Lipton *et al.* [1976], Lipton and Tarjan [1980], Rosenberg [1978, 1981], Rosenberg and Snyder [1978]), of finding efficient layouts of circuits on VLSI chips (Aleliunas and Rosenberg [1982], Bhatt and Leighton [1984], Leighton [1983], Leighton and Rosenberg [1986], Leiserson [1983], Thompson [1980], Valiant [1981], Vuillemin [1983]), of finding efficient structured versions of programs (Lipton *et al.* [1976]), and of organizing computations on networks of processors (Kung and Stevenson [1977]). In addition, numerous specific computational problems—especially those that admit solutions involving decomposition of problem domains—can fruitfully be formulated as problems of manipulating and/or partitioning graphs in various ways (Gannon [1980], Lipton and Tarjan [1980]).

A striking feature of all of the cited problem areas—as well as of their kindred areas that we had no space to enumerate—is that they exploit the same major structural feature of their graph-theoretic models, namely the decomposition structure of the graphs, as embodied in various notions of *graph separator*. All variations on the theme of graph separation involve removing either edges or nodes from the subject graphs in order to partition each graph into disjoint subgraphs whose sizes must be within certain prespecified absolute or relative bounds. In all of the cited areas, the complexities of either procedures (e.g., algorithm timing) or structures (e.g., circuit areas) can be bounded via bounds on the sizes of graph separators.

In the Preface we reviewed a few detailed scenarios that benefit from abstraction into a graph-theoretic framework and, particularly, from a study of the separators of the graphs that the framework yields.

This book is devoted to surveying and unifying the work on upper- and lower-bound techniques for the sizes of graph separators. We survey the highlights of the world of upper bounds on graph separator-sizes, which manifest themselves almost always in algorithms for computing good recursive decompositions of graphs belonging to particular families, but we try to be almost exhaustive in our coverage of lower-bound techniques. This asymmetry in our coverage of the field is due to our perception that the work on upper bounds (i.e., on algorithms for separating graphs) is adequately located in the Theory literature for scholarly access, whereas the work on lower bounds tends to be scattered among the literature on the motivating applications. This scattering has led to many independent developments of similar techniques and of many weak lower bounds when better ones were readily available. We hope to ameliorate this situation by gathering a large battery of generally applicable techniques in one work whose title leaves little chance for misinterpretation or misclassification.

This book is partitioned into four chapters and an appendix. This chapter presents basic notions and notations and surveys a bunch of simple, yet useful, results that develop a formal analog, *quasi-isometry*, of the informal notion of two graph families' being "essentially" the same for all practical purposes. Chapter 2 presents a sampler of problems areas wherein graph separators play a major algorithmic role. Chapter 3 surveys the major avenues that have led to efficient algorithms for decomposing graphs. Chapter 4 is the longest and meatiest portion of the book. It is here that we develop, and illustrate, the lower-bound techniques that motivate the entire enterprise. The popular graph families enumerated in Section 1.3 yield a set of benchmarks for the techniques developed. Appendix A returns to some of the applications of Chapter 2 in order to illustrate the implications of the techniques and illustrations of Chapters 3 and 4 for the application areas that depend on the theory of graph separators. References to sources used in the technical development and pointers to further sources in the literature appear as the last section of each chapter.

1.2. Basic Notions and Notation

This section presents the basic notions that will accompany our journey through the world of graph separators and establishes the notation we shall use to talk about these notions.

1.2.1. Useful Combinatorial Notions

Given any set S, we denote by $|S|$ the *cardinality* of S. This is a well-defined notion since all of our sets will be finite. We introduce notation for some important specific families of sets.

- For each nonnegative integer n, we denote by Z_n the set

$$Z_n \stackrel{\text{def}}{=} \{0, 1, \ldots, n-1\}.$$

- For any set S, we denote by S^n the set of all $|S|^n$ length-n strings of elements of S. (Note, in particular the length-0 *null string* λ that is the sole resident of the set S^0.)
- For any set S, we denote by S^* the set of all finite-length strings of elements of S.

Given any string x of elements of a set S, we denote by $|x|$ the *length* of x. We observe the obvious tautology $x \in S^{|x|}$. For any *binary* string $x \in Z_2^n$, we call the number of 1s in x the *weight* of the string.

Finally, one simplifying notational convention: unless otherwise indicated, all logarithms will be to the base 2.

1.2.2. Graph-Theoretic Notions

A *graph* is a system $\mathcal{G}$ having a finite set $\mathbf{N}(\mathcal{G})$ of *nodes* and a set $\mathbf{E}(\mathcal{G})$ of doubleton subsets of $\mathbf{N}(\mathcal{G})$ called *edges*. We denote graphs by script letters, often embellished with personalizing parameters (that specify characteristics like "height" or "side" or "size"). On occasion, we shall want the edges of our graphs to have "directions." At these times, we endow our directed graphs (*digraphs*, for short) with *arcs*, which are elements of $\mathbf{N}(\mathcal{G}) \times \mathbf{N}(\mathcal{G})$, instead of edges. We often illustrate graphs as indicated in Figure 1.2-1. We establish special notation for the cardinalities of the node- and edge-sets of a graph $\mathcal{G}$: we let $N(\mathcal{G}) \stackrel{\text{def}}{=} |\mathbf{N}(\mathcal{G})|$ and $E(\mathcal{G}) \stackrel{\text{def}}{=} |\mathbf{E}(\mathcal{G})|$.

Let u and v be nodes of the graph $\mathcal{G}$. If the doubleton $\{u, v\}$ is an edge of $\mathcal{G}$ (i.e., is an element of $\mathbf{E}(\mathcal{G})$), then we say that nodes u and v are *adjacent*, or are *neighbors*, in $\mathcal{G}$. The *degree* of node v in $\mathcal{G}$, denoted $d_{\mathcal{G}}(v)$, is the number of nodes that are adjacent to v in $\mathcal{G}$ or, equivalently, the number of edges in $\mathbf{E}(\mathcal{G})$ that v belongs to or that *are incident to* v. The *maxdegree* of $\mathcal{G}$, denoted $d_{\mathcal{G}}$, is the maximum degree of any of its nodes. A graph $\mathcal{G}$ is *regular* if all its nodes have equal degrees. Dual to these notions for nodes, two edges $\{u_1, v_1\}$ and $\{u_2, v_2\}$ of $\mathcal{G}$ are *dependent* if they share a node: $\{u_1, v_1\} \cap \{u_2, v_2\} \neq \varnothing$; otherwise, the edges are *independent*.

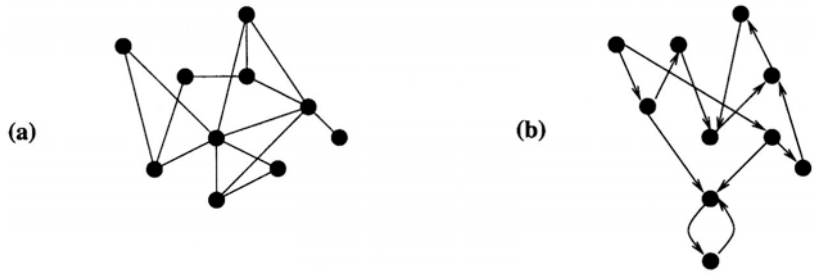


Figure 1.2-1. (a) A graph, (b) A digraph.

A *path* in $\mathcal{G}$ between nodes u and v is a sequence of nodes

$$u = w_0, w_1, w_2, \ldots, w_n = v \tag{1.2.1}$$

where each doubleton $\{w_i, w_{i+1}\}$, for $i \in \mathbf{Z}_n$, is an edge of $\mathcal{G}$. We say that each edge $\{w_i, w_{i+1}\}$ *occurs* in the path. We usually denote path (1.2.1) by the more perspicuous notation

$$w_0 \leftrightarrow w_1 \leftrightarrow w_2 \leftrightarrow \cdots \leftrightarrow w_n$$

We typically intend that a path is *simple*; that is, no edge occurs in the path twice. We say that the indicated path has *length* $n + 1$; i.e., we measure pathlength by the number of nodes. We say that graph $\mathcal{G}$ is *connected* if there is a path between every pair of nodes of $\mathcal{G}$. Finally, the *diameter* of a graph $\mathcal{G}$ is the length of the longest simple path in $\mathcal{G}$ or, equivalently, the maximum distance between any two nodes of $\mathcal{G}$.

A graph $\mathcal{H}$ is a *subgraph* of a graph $\mathcal{G}$ if $\mathbf{N}(\mathcal{H}) \subseteq \mathbf{N}(\mathcal{G})$ and $\mathbf{E}(\mathcal{H}) \subseteq \mathbf{E}(\mathcal{G})$. In addition, $\mathcal{H}$ is a *spanning subgraph* of $\mathcal{G}$ if $\mathbf{N}(\mathcal{H}) = \mathbf{N}(\mathcal{G})$, and an *induced subgraph* of $\mathcal{G}$ if $\mathbf{E}(\mathcal{H})$ contains *all* edges of $\mathcal{G}$ that connect nodes within $\mathbf{N}(\mathcal{H})$.

1.3. Interesting Graph Families

This section is devoted to defining a number of families of graphs that are interesting in the context of a variety of application areas, notably the study of data structures and of parallel architectures.

1.3.1. Clique-Related Graphs

Clique-related graphs are the epitome of point-to-point connections in graphs. Their denseness makes them unwieldy to implement directly, either in software or hardware, so they are more of interest as a goal to be approximated than as a structure to be implemented; cf. Section 4.3 and Aiello and Leighton [1991] and Leighton [1983].

The *n-node clique*, or *complete graph* $\mathcal{K}_n$ has node-set $\mathbf{N}(\mathcal{K}_n) = Z_n$; its edges connect every pair of distinct nodes. Because $\mathcal{K}_n$ has $N(\mathcal{K}_n) = n$ nodes, each of degree n – 1, it has

$$E(\mathcal{K}_n) = \binom{n}{2}$$

edges. See Figure 1.3-l(a).

The $m \times n$ *complete bipartite graph* $\mathcal{K}_{m,n}$ has node-set $V_m \cup V_n$, where V_m and V_n are disjoint sets (symbolically, $V_m \cap V_n = \varnothing$) having, respectively, m nodes and n nodes (symbolically, $|V_m| = m$ and $|V_n| = n$). The edges of $\mathcal{K}_{m,n}$ connect every pair of nodes u and v, where $u \in V_m$ and $v \in V_n$. The graph $\mathcal{K}_{m,n}$ has $N(\mathcal{K}_{m,n}) = m + n$ nodes and $E(\mathcal{K}_{m,n}) = mn$ edges. See Figure 1.3-l(b).

1.3.2. Paths and Cycles

The structural simplicity of paths and cycles makes these structures important in data structures (Rosenberg [1978]) and parallel architectures (Fellows and Langston [1988], Kung and Picard [1984]). The *length-n path* $\mathcal{P}_n$ has node-set $\mathbf{N}(\mathcal{P}_n) = Z_n$; its edges connect every pair of nodes x and $x + 1$ for every $x \in Z_n - \{n - 1\}$. $\mathcal{P}_n$ has $N(\mathcal{P}_n) = n$ nodes and $E(\mathcal{P}_n) = n - 1$ edges.

The *length-n cycle* $\mathcal{C}_n$ also has node-set $\mathbf{N}(\mathcal{C}_n) = Z_n$; its edges connect every pair of nodes x and $x + 1 \bmod n$. $\mathcal{C}_n$ has $N(\mathcal{C}_n) = n$ nodes. Because each node of $\mathcal{C}_n$ has degree 2, the graph has $E(\mathcal{C}_n) = n$ edges.



Figure 1.3-1. (a) The 6-node clique $\mathcal{K}_6$. (b) The 4×3 complete bipartite graph $\mathcal{K}_{4,3}$.

1.3.3. Products of Simple Graphs

Some of the most common and useful families of graphs can be defined as products of the simple graphs from the previous subsections. The *product* $\mathcal{G} \times \mathcal{H}$ of graphs $\mathcal{G}$ and $\mathcal{H}$ has node-set $\mathbf{N}(\mathcal{G}) \times \mathbf{N}(\mathcal{H})$ and has an edge between nodes $\langle u_1, v_1 \rangle$ and $\langle u_2, v_2 \rangle$ exactly when either $u_1 = u_2$ and $\{v_1, v_2\} \in \mathbf{E}(\mathcal{H})$ or $v_1 = v_2 \{u_1, u_2\} \in \mathbf{E}(\mathcal{G})$.

1.3.3.1. Products of Paths and Cycles

Meshes, both "flat" and toroidal, play a major role in data structures (Rosenberg [1975], Johnsson [1987]) and parallel architectures (Dally and Seitz [1986], Gannon [1980]).

The *two-dimensional* $m \times n$ *toroidal mesh* $\tilde{\mathcal{M}}_{m,n}$ is the product graph $\mathcal{C}_m \times \mathcal{C}_n$. Therefore, $\tilde{\mathcal{M}}_{m,n}$ has node-set $\mathbf{N}(\tilde{\mathcal{M}}_{m,n}) = Z_m \times Z_n$; its edges come in two classes: $\tilde{\mathcal{M}}_{m,n}$ has a *row-edge* between every pair of nodes $\langle u, v \rangle$ and $\langle u, v + 1 \bmod n \rangle$; it has a *column-edge* between every pair of nodes $\langle u, v \rangle$ and $\langle u + 1 \bmod m, v \rangle$. Because $\tilde{\mathcal{M}}_{m,n}$ has $N(\tilde{\mathcal{M}}_{m,n}) = mn$ nodes, each of degree 4, it has $E(\tilde{\mathcal{M}}_{m,n}) = 2mn$ edges. See Figure 1.3-2(a).

The *two-dimensional* $m \times n$ *rectangular mesh* $\mathcal{M}_{m,n}$ is the product graph $\mathcal{P}_m \times \mathcal{P}_n$. Therefore, $\mathcal{M}_{m,n}$ has node-set $\mathbf{N}(\mathcal{M}_{m,n}) = Z_m \times Z_n$. Its edges come in two classes: $\mathcal{M}_{m,n}$ has a *row-edge* between every pair of nodes $\langle u, v \rangle$ and $\langle u, w \rangle$, where $|v - w| = 1$, it has a *column-edge* between every pair of nodes $\langle u, v \rangle$ and $\langle w, v \rangle$, where $|u - w| = 1$, and it has $N(\mathcal{M}_{m,n}) = mn$ nodes and $E(\mathcal{M}_{m,n}) = 2mn - m - n$ edges. See Figure 1.3-2(b).

The *d-dimensional side-n mesh* $\mathcal{M}_n(d)$ is the d-fold product graph $\mathcal{P}_n^d \stackrel{\text{def}}{=} \mathcal{P}_n \times \mathcal{P}_n \times \cdots \times \mathcal{P}_n$. Therefore, $\mathcal{M}_n(d)$ has node-set $\mathbf{N}(\mathcal{M}_n(d)) = Z_n^d$; its edges connect every pair of nodes $\langle u_0, u_1, \ldots, u_{n-1} \rangle$ and $\langle v_0, v_1, \ldots, v_{n-1} \rangle$ for which $\Sigma_{i=0}^{n-1} |u_i - v_i| = 1$. The mesh $\mathcal{M}_n(d)$ has $N(\mathcal{M}_n(d)) = n^d$ nodes, and $E(\mathcal{M}_n(d)) = d(n^d - n^{d-1})$ edges. When $d = 2$, we elide the parameter d.

Other product combinations of paths and/or cycles can be defined, but the aforementioned are the ones that will recur in our study.

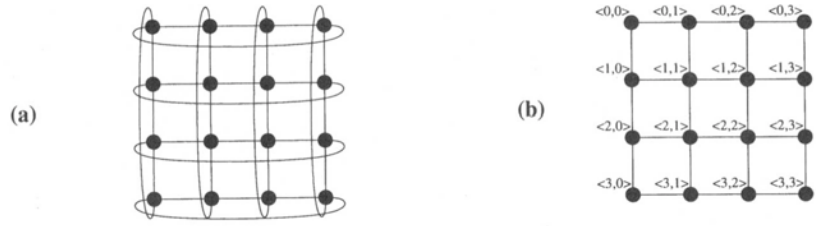


Figure 1.3-2. (a) The 4×4 toroidal mesh $\tilde{\mathcal{M}}_{4,4}$. (b) The 4×4 "flat" mesh $\mathcal{M}_{4,4}$.

1.3.3.2. Products of Cliques

Useful families have emerged also by taking products of cliques. (When considering the next two families, the reader should recall that the 2-node clique $\mathscr{K}_2$ and the 2-node path $\mathscr{P}_2$ are isomorphic.)

The mesh-of-cliques graph, as with other clique-based graphs, is useful more as an ideal to be approximated or a tool for analysis than as a graph to be "implemented."

The *two-dimensional m × n mesh-of-cliques graph* $\mathscr{MC}_{m,n}$ is the product $\mathscr{K}_m \times \mathscr{K}_n$. Therefore, $\mathscr{MC}_{m,n}$ has node-set $\mathbf{N}(\mathscr{MC}_{m,n}) = Z_m \times Z_n$; its edges connect every pair of nodes in each *row* $\{i\} \times Z_n$ $(i \in Z_m)$ and in each *column* $Z_m \times \{j\}$ $(j \in Z_n)$. Because $\mathscr{MC}_{m,n}$ has $N(\mathscr{MC}_{m,n}) = mn$ nodes, each of degree $(m - 1) + (n - 1) = m + n - 2$, it has

$$E(\mathscr{MC}_{m,n}) = m\binom{n}{2} + n\binom{m}{2}$$

edges.

The family of hypercubes embodies one of the most important graph structures in the world of parallel architectures (Aiello and Leighton [1991], Johnsson [1987], Seitz [1985], Stanfill [1987]) (for their use as interconnection networks) and in the world of coding theory (Harper [1964, 1966, 1967], Peterson and Weldon [1981]) (for their utility in constructing codes with various adjacency and nonadjacency properties among codewords[2]).

The *base-b n-dimensional hypercube* $\mathscr{Q}_n(b)$ is the n-fold product graph $\mathscr{K}_b^n \stackrel{\text{def}}{=} \mathscr{K}_b \times \mathscr{K}_b \times \cdots \times \mathscr{K}_b$. Therefore, $\mathscr{Q}_n(b)$ has node-set $\mathbf{N}(\mathscr{Q}_n(b)) = Z_b^n$, which set is usually interpreted as comprising all length-n strings over the alphabet Z_b. The hypercube $\mathscr{Q}_n(b)$ has an edge between every pair of nodes $x\beta y$ and $x\gamma y$ that differ in precisely one digit-position; i.e., $x \in Z_b^k$ is some length-k string, $y \in Z_b^{n-k-1}$ is some length-$(n - k - 1)$ string, and both β and $\gamma \neq \beta$ are in Z_b (so are digits, or length-1 strings). Because $\mathscr{Q}_n(b)$ has $N(\mathscr{Q}_n(b)) = b^n$ nodes, each of degree $(b - 1)n$, it has

$$E(\mathscr{Q}_n(b)) = \frac{b-1}{2} nb^n$$

edges. When $b = 2$, we call $\mathscr{Q}_n(b)$ the *boolean* hypercube and elide the parameter b. See Figure 1.3-3.

1.3.4. Trees and Related Graphs

Trees embody the structurally simplest connected graphs. Complete trees are important in the study of data structures (Rosenberg [1979]) and

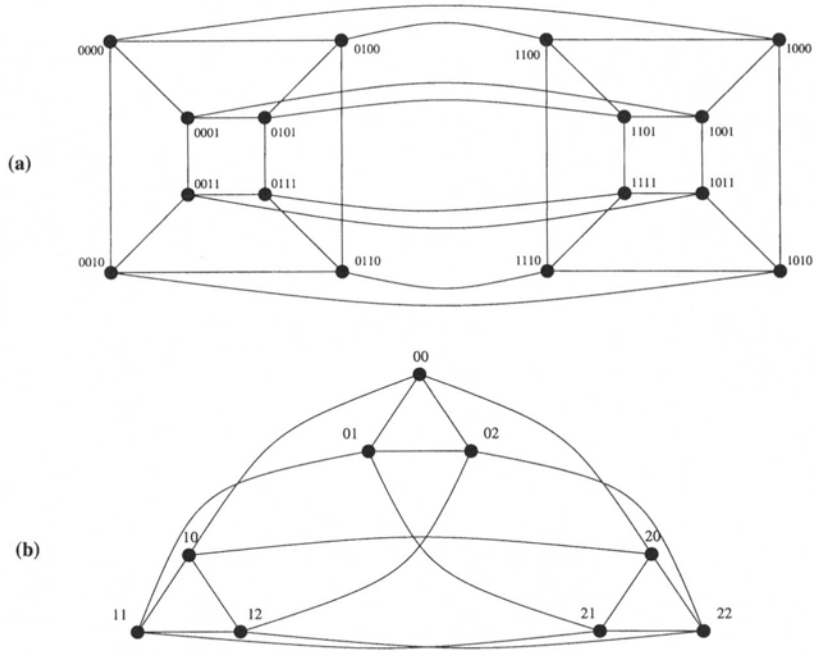


Figure 1.3-3. (a) The four-dimensional boolean hypercube $\mathcal{Q}_4$. (b) The two-dimensional ternary hypercube $\mathcal{Q}_2(3)$.

parallel architectures (Bentley and Kung [1979], Browning [1980]). Complete trees augmented to X-trees by the addition of "cross edges" have been studied as interconnection networks for parallel architectures (Despain and Patterson [1978]). Trees that are not necessarily complete arise in the study of data structures (Aho *et al.* [1977], Berkman and Vishkin [1993]) and in the control structures of algorithms and programs (Carlson [1984], Lipton *et al.* [1976]).

The *height-h complete b-ary tree* $\mathcal{T}_h(b)$ has node-set

$$\mathbf{N}(\mathcal{T}_h(b)) = \bigcup_{i=0}^{h} Z_b^i$$

i.e., the set of b-ary strings of length at most h; it has an edge between every pair of nodes x and $x\gamma$, where $x \in \bigcup_{i=0}^{h-1} Z_b^i$ is some b-ary string of length $< h$, and $\gamma \in Z_b$ (so it is a digit, or length-1 string). One conventionally partitions

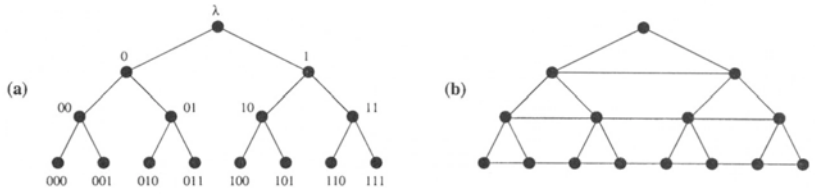


Figure 1.3-4. (a) The height-3 complete binary tree $\mathcal{T}_3$. (b) The height-3 *X*-tree $\mathcal{X}_3$

the nodes of $\mathcal{T}_h(b)$ into *levels* by their lengths: the *root* of $\mathcal{T}_h(b)$ is the unique node at level 0; the b^ℓ nodes of $\mathcal{T}_h(b)$ of length ℓ, where $0 \leqslant \ell \leqslant h$ reside at level ℓ; the *leaves* of $\mathcal{T}_h(b)$ are the b^h nodes at level *h*. Thus, if $b > 1$, $\mathcal{T}_h(b)$ has $N(\mathcal{T}_h(b)) = (b^{h+1} - 1)/(b - 1)$ nodes and (as with all trees) one fewer edge than nodes. When $b = 2$, we elide the parameter *b*. Note that the path $\mathcal{P}_h$ can be viewed as the height-$(h - 1)$ unary tree. See Figure 1.3-4(a).

NOTE. One usually talks about the *arity* of a nonleaf node in a rooted tree, which is its number of children, rather than the node's *degree*.

The *height-h X-tree* $\mathcal{X}_h$ is obtained from the height-*h* complete binary tree $\mathcal{T}_h$ by adding edges that create a path along each level of $\mathcal{T}_h$ with the nodes occurring *in lexicographic order*. See Figure 1.3-4(b).

Our next family has much looser structure than the other families we have defined, in that it has many more members of each size.

A *rooted b-ary tree* is any connected graph $\mathcal{T}$ satisfying the following:

- $\mathbf{N}(\mathcal{T})$ is a finite set of strings over Z_b, which contains the length-0 string: this (null) string is the *root* of $\mathcal{T}$
- Whenever a positive-length node $x\beta$ is in $\mathbf{N}(\mathcal{T})$, where $x \in Z_b^*$ and $\beta \in Z_b$, the node x is also in $\mathbf{N}(\mathcal{T})$;[3] moreover, there is an edge of $\mathcal{T}$ that connects nodes x and $x\beta$
- The edge connecting x and $x\beta$ is the *only* edge that connects $x\beta$ with a shorter node.

We call node x the *parent* of node $x\beta$, and we call node $x\beta$ a *child* of node x; a node that has no children is a *leaf* of $\mathcal{T}$.

1.3.5. Shuffle-Like Graphs

The graphs defined in this subsection are called *shuffle-like* because their nodes are most perspicuously denoted as strings and their edges



Figure 1.3.5. (a) The order-3 de Bruijn graph $\mathscr{D}_3$. (b) The order-3 shuffle-exchange graph $\mathscr{S}_3$.

specified in terms of the *shuffle* operator on strings; this operator cyclically shifts a string one position to the left, for instance producing the string 1100 from the string 0110. Shuffle-like graphs differ in one major respect from the other graph families we study in this book: for reasons that are both historical and technical,[4] one typically allows shuffle-like graphs to have both self-loops—edges that connect a node v with itself—and parallel edges—distinct edges connecting the same pair of nodes. These graphs have been studied extensively in the context of parallel architectures (Bermond and Peyrat [1989], Schwartz [1980], Stone [1971]) and have important applications in the world of codes (Peterson and Weldon [1981]). We define and study only the two most frequently encountered families of shuffle-like graphs; a third family, the *perfect shuffle* graphs, appear in Schwartz [1980].

The *base-b order-n de Bruijn graph* $\mathscr{D}_n(b)$ has node-set $\mathbf{N}(\mathscr{D}_n(b)) = Z_b^n$; its edges connect every pair of nodes βx and $x\gamma$, where $x \in Z_b^{n-1}$ is some length-$(n-1)$ string, and both β and γ are in Z_b (so are digits, or length-1 strings). Because $\mathscr{D}_n(b)$ has $N(\mathscr{D}_n(b)) = b^n$ nodes, each of degree $2b$, it has $E(\mathscr{D}_n(b)) = b^{n+1}$ edges. When $b = 2$, we elide the parameter b. See Figure 1.3-5(a).

The *order-n shuffle-exchange graph* $\mathscr{S}_n$ is a close relative of $\mathscr{D}_n$; in common with $\mathscr{D}_n$, $\mathscr{S}_n$ has node-set $\mathbf{N}(\mathscr{S}_n) = Z_2^n$. The edges of $\mathscr{S}_n$ connect every pair of nodes βx and $x\beta$, as well as every pair of nodes $x\gamma$ and $x\bar{\gamma}$, where $x \in Z_2^{n-1}$ is a length-$(n-1)$ string; β and γ are in Z_2 (so are digits, or length-1 strings); and $\bar{\gamma} = 1 - \gamma$. Because $\mathscr{S}_n$ has $N(\mathscr{S}_n) = 2^n$ nodes, each of degree 3, it has $E(\mathscr{S}_n) = 3 \cdot 2^{n-1}$ edges. See Figure 1.3-5(b). Although de Bruijn graphs are often studied in their general, base-b, versions, shuffle-exchange graphs appear almost exclusively in their binary, base-2, version.

1.3.6. The Butterfly and Related Graphs

Butterfly-oriented graphs are so named because they can be viewed as being composed of overlapping copies of the *butterfly*, or complete bipartite

graph $\mathcal{K}_{2,2}$. Butterfly-oriented graphs play an even more important role in the study of parallel architectures than do shuffle-oriented graphs, for several reasons. First, butterfly-oriented graphs can be seen directly to be bounded-degree approximations to the computationally efficient family of hypercubes; with shuffle-oriented graphs, there is a level of indirection involved. Second, butterfly-oriented graphs enjoy symmetries that facilitate the implementation of algorithms; shuffle-oriented graphs can simulate such symmetries but do not actually have them. Finally, numerous actual (even commercial) parallel architectures have appeared that are based on butterfly-oriented interconnection networks. The following sources offer a good start on the extensive literature about butterfly-oriented networks: Annexstein *et al.* [1990], Bhatt *et al.* [1996], Gottlieb [1986], Preparata and Vuillemin [1981], Rettberg [1986], and Schwabe [1993].

The *base-b order-n butterfly graph* $\mathcal{B}_n(b)$ has node-set $\mathbf{N}(\mathcal{B}_n(b)) = Z_n \times Z_b^n$; its edges connect every pair of nodes $\langle \ell, x\beta y\rangle$ and $\langle \ell + 1 \bmod n, x\gamma y\rangle$, where $x \in Z_b^\ell$ is some **length-ℓ** string, $y \in Z_b^{n-\ell-1}$ is some **length-$(n-\ell-1)$** string, and $\beta, \gamma \in Z_b$ (so are digits, or length-1 strings). For each node $\langle \ell, w\rangle \in \mathbf{N}(\mathcal{B}_n(b))$, we call ℓ the *level* of the node and w the *position-within-level string* (*PWL string*) *of the node. We call each edge*

$$\langle \ell, x\beta y\rangle \leftrightarrow \langle \ell + 1 \bmod n, x\gamma y\rangle \tag{1.3.1}$$

of $\mathcal{B}_n(b)$ a *straight* edge if $\beta = \gamma$ and a cross edge if $\beta \neq \gamma$. Sometimes $\mathcal{B}_n(b)$ is called the *butterfly graph with wraparound* (because of the "+1 mod n" proviso in (1.3.1). Because $\mathcal{B}_n(b)$ has $N(\mathcal{B}_n(b)) = nb^n$ nodes, each of degree $2b$, it has $E(\mathcal{B}_n(b)) = nb^{n+1}$ edges. When $b = 2$, we elide the parameter b. See Figure 1.3-6(a).

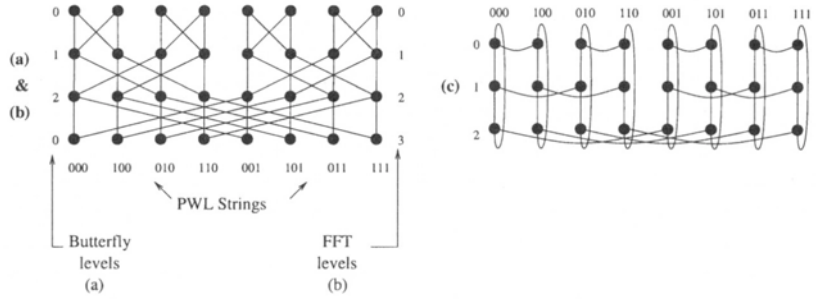


Figure 1.3-6. (a) The order-3 butterfly graph $\mathcal{B}_3$. (Note the wraparound.) (b) The order-3 *FFT* graph $\mathcal{F}_3$. (c) The order-3 cube-connected cycles graph $\mathcal{CCC}_3$.

The *order-n FFT graph* $\mathcal{F}_n$ (so named because its structure reflects the data dependencies of the Fast Fourier Transform algorithm (Aho *et al.* [1974])) has node-set $\mathbf{N}(\mathcal{F}_n) = Z_{n+1} \times Z_2^n$. It has an edge between every pair of nodes $\langle \ell, x\beta y \rangle$ and $\langle \ell + 1, x\gamma y \rangle$, where $\ell \in Z_n$, and where $x \in Z_2^\ell$ is some length-ℓ string, $y \in Z_2^{n-\ell-1}$ is some length-$(n - \ell - 1)$ string, and $\beta, \gamma \in Z_2$ (so are digits, or length-1 strings). The graph $\mathcal{F}_n$ inherits *levels*, *PWL strings*, and the notions of *straight* and *cross* edges from $\mathcal{B}_n$. Because of its structural kinship with $\mathcal{B}_n$, $\mathcal{F}_n$ is often called the *butterfly graph without wraparound.* In addition, $\mathcal{F}_n$ has $N(\mathcal{F}_n) = (n + 1)2^n$ nodes and $E(\mathcal{F}_n) = n2^{n+1}$ edges. See Figure 1.3-6(b).

The *order-n cube-connected cycles graph* $\mathcal{CCC}_n$ has node-set $\mathbf{N}(\mathcal{CCC}_n) = Z_n \times Z_2^n$. $\mathcal{CCC}_n$ has a *straight-edge* between every pair of nodes $\langle \ell, x \rangle$ and $\langle \ell + 1 \bmod n, x \rangle$, where $x \in Z_2^n$ is some length-n string; $\mathcal{CCC}_n$ has a *level* edge between every pair of nodes $\langle \ell, x\beta y \rangle$ and $\langle \ell, x\bar{\beta} y \rangle$, where $x \in Z_2^\ell$ is some length-ℓ string, $y \in Z_2^{n-\ell-1}$ is some length-$(n - \ell - 1)$ string, and $\beta \in Z_2$ (so is a bit, or a length-1 string). Levels and PWL strings are defined for $\mathcal{CCC}_n$ in the same manner as for $\mathcal{B}_n$. Because $\mathcal{CCC}_n$ has $N(\mathcal{CCC}_n) = n2^n$ nodes, each of degree 3, it has $E(\mathcal{CCC}_n) = 3n2^{n-1}$ edges. See Figure 1.3-6(c).

The reader should be able to define the generalized, base-b, version $\mathcal{CCC}_n(b)$ of the cube-connected cycles graph by perusing the relationship between $\mathcal{B}_n$ and $\mathcal{B}_n(b)$.

1.4. Graph Separators

We have already commented on the multitude of computational situations that can fruitfully be modeled within a graph-theoretic setting. The ubiquity of so-called divide-and-conquer strategies in computation (cf. Aho *et al.* [1974]) would lead one to expect, correctly, that in many of these situations it is important to understand (both conceptually and algorithmically) the decomposition structure of the graph-theoretic models.

Numerous variations on the theme of the decomposition structure of graphs have appeared in the literature, under a variety of names (even for the same variation). Thus, one reads about *graph separators*, which is the name we shall always use, as well as *graph bifurcators* and *graph boundaries. Loosely speaking, all of these terms refer to a set of nodes (in one flavor) or of edges (in another flavor) whose removal chops a given graph* $\mathcal{G}$ into (usually two, but sometimes more) disjoint subgraphs whose sizes stand in some sought relation. One is almost always interested in the *size* of $\mathcal{G}$'s graph separator, i.e., the cardinality of the removed set of nodes or edges (see Gilbert *et al.* [1984], Lipton and Tarjan [1979]) as two examples among

many), although a very few studies have been interested also in the graph-theoretic structure of the subgraph of $\mathcal{G}$ induced by the separator (in Miller [1986], for instance, one seeks a node-separator whose induced graph is hamiltonian). In some studies, the sizes of a graph's separators, as one chops the graph into pieces of various sizes, have been called the graph's *bisection-width* or the graph's *exposure function.* The study of the sizes of graphs' separators has often occurred under the rubric *isoperimetric inequalities*, in analogy with boundary-volume studies in continuous domains. A closely related area of study, mentioned briefly in Section 2.3, is the *expansion property* of a graph.

1.4.1. Variations on the Theme

In order to give the reader a reasonable introduction to the many variations on the theme of graph separator, we must employ something of a matrix organization, since notions often differ along orthogonal axes. We have already presented our first variation by distinguishing between node- and edge-separators.

1.4.1.1. Partitioning versus Decomposition

A major focus of this book is techniques for establishing *lower* bounds on the sizes of graph separators. In typical application areas that are modeled using graphs, one can obtain lower bounds in a situation modeled by a graph $\mathcal{G}$ by bounding from below the size of the smallest set of edges (or nodes) whose removal partitions (or cuts) $\mathcal{G}$ into subgraphs of appropriate (absolute or relative) sizes. Most of our deliberations in Chapter 4 involve a search for such lower bounds as the sought sizes of the surviving pieces of $\mathcal{G}$ vary. In deference to the existing notions of the edge- or node-*bisection-width* of a graph $\mathcal{G}$, which is the size of the smallest edge- or node-separator of $\mathcal{G}$ into two subgraphs of equal sizes (to within rounding), we shall henceforth term the size of the smallest edge-separator (resp., node-separator) that partitions $\mathcal{G}$ into two subgraphs of appropriate sizes the *edge-separation-width* (resp., *node-separation-width*) of $\mathcal{G}$. We always take care in the text to qualify every instance of the phrase "separation-width" in a way that indicates the operative notion of appropriateness and the choice of edge- or node-removal; these qualifications will often be by context.

Thus far, we have talked only about the most common situation, wherein one wants to chop the graph $\mathcal{G}$ into two pieces. There have been very few studies wherein one wants to *mince* $\mathcal{G}$, i.e., partition it into k

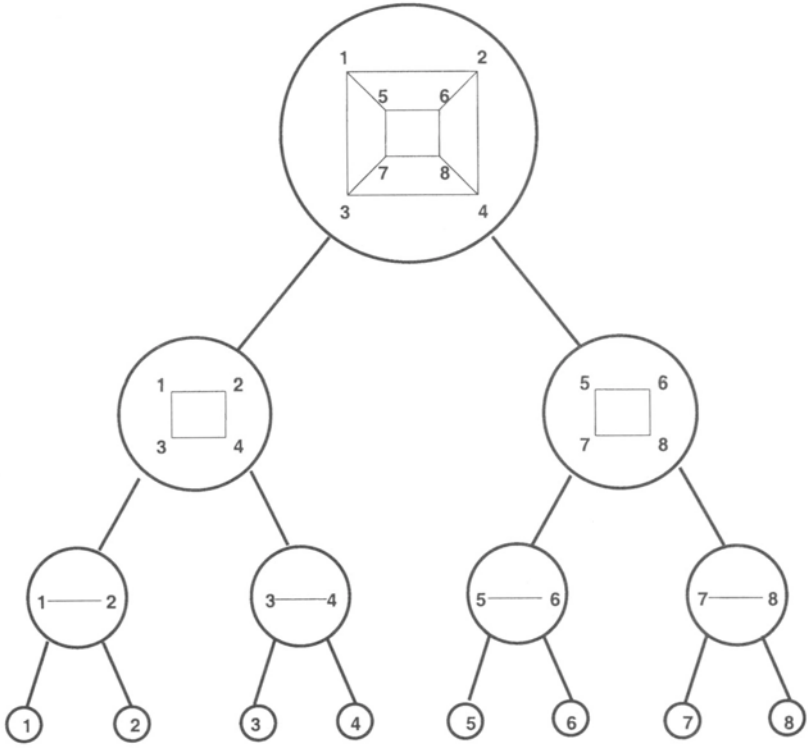


Figure 1.4-1. A (l/2)-decomposition tree for the three-dimensional hypercube.

equal-size subgraphs, for varying k, instead of into just two subgraphs; we briefly focus on this problem in Section 4.2.5.

In contrast to the *one-level* partitioning problem that characterizes the study of lower bounds, the study of *upper* bounds usually demands that one *recursively partition* the subject graph $\mathcal{G}$ until one reduces it to a collection of trivial (usually 1-node) subgraphs. In the world of upper bounds, therefore, one usually wants to find a *decomposition tree* for the graph $\mathcal{G}$. The nodes of this tree are subgraphs of $\mathcal{G}$: the root node is the entire graph $\mathcal{G}$, and, recursively, the children of a node-graph $\mathcal{H}$ are the subgraphs into which $\mathcal{H}$ is decomposed by the partitioning algorithm. See Figure 1.4-1 and Section 1.4.2.1. Obviously, the upper-bound-oriented graph-decomposition setting demands a somewhat more complicated notion of separator-size than does the lower-bound-oriented graph-partition setting.

It should be clear that when we discuss bounded-degree families of graphs, the distinction between the edge- and node-oriented versions of

graph-separation manifests itself only in constant factors. Hence, except in those rare situations where optimizing constant factors is a major issue, we lose little by allowing some ambiguity to slip into our discussion. When discussing graph families whose degrees are not bounded, however, such ambiguity cannot be countenanced. In order to afford us brevity at no cost in accuracy, we shall, therefore, henceforth discuss only edge-separators and edge-separation-width, unless otherwise stated. We shall see in Chapter 2 that certain applications naturally call for either the edge- or the node-oriented version of graph-separator.

1.4.1.2. Full Separation versus I/O Separation

Virtually all graph-separation problems in the literature fall into one of two categories. The first type of problem (the one we have been discussing) seeks what we call a *full separation* of a graph: a partition of the graph into two subgraphs whose sizes satisfy some condition. This is the "classical" notion of graph-separation that is studied in early sources such as Harper [1966], Leiserson [1983], Lipton *et al.* [1976], Lipton and Tarjan [1979], Rosenberg [1978, 1981], and Sheidvasser [1974]. The second type of separator problem deals with graphs that are endowed with designated disjoint sets of *input* nodes and *output* nodes; the problem seeks what we call an *I/O-separation* of a graph: a partition of the graph $\mathcal{G}$ into two subgraphs in such a way that the numbers of $\mathcal{G}$'s input nodes (and/or $\mathcal{G}$'s output nodes) that end up in the two subgraphs are in some prespecified proportion. This type of separation originated in Thompson [1980]; it is a standard device when one wants to study the complexity of a family of functions as determined by the amount of information flow necessary to compute the function; see, for instance, Cole and Siegel [1988] and Vuillemin [1983].

1.4.1.3. Absolute versus Relative Subgraph Sizes

We discuss here only *full edge-separation*, although what we say applies with only clerical modification to I/O-separation and to node-separation as well. The question at hand is: What is an "appropriate" decomposition of the subject graph $\mathcal{G}$?

1.4.1.3a. Absolute Sizes. The strongest notion of appropriateness that we could demand is an *absolute* one, which is feasible only in the study of lower bounds on separator sizes. This notion seeks the separation-width of an N-node graph $\mathcal{G}$ for arbitrary partitions of $\mathcal{G}$ into subgraphs having, respectively, M nodes and $N - M$ nodes. The resulting notion of the *M-separation-width* of $\mathcal{G}$, denoted $W_{\mathcal{G}}(M)$, is the quantity that we aim to

bound from below throughout Chapter 4. Of course, the notion of separation-width is symmetric in M and $N - M$, so, for instance, $W_{\mathcal{G}}(M) = W_{\mathcal{G}}(N - M)$.

1.4.1.3b. Relative Sizes. More typical than the preceding absolute notion—especially in the literature on upper bounds, i.e., on decomposition algorithms—is some variation of the following *relative* notion of appropriateness. We choose some rational number β in the range $0 < \beta \leqslant 1/2$. We then seek a device for measuring the decomposition complexity of $\mathcal{G}$. Two competing devices have each proved so successful in the literature that we present both here.

1. *Classical separation.* The graph $\mathcal{G}$ has a ***β*-separator** *of size* $S(n)$, where $S(n)$ is an integer function, just when the following holds.

> Either $N(\mathcal{G}) = 1$, or, by removing no more than $S(N(\mathcal{G}))$ edges, one can partition $\mathcal{G}$ into two subgraphs, each having at most $\lceil (1 - \beta)N(\mathcal{G}) \rceil$ nodes, and each having a **β-separator** of size $S(n)$.

A (1/2)-node-separator is often called a *recursive bisector.*[5]

2. *Bifurcation.* The graph $\mathcal{G}$ has a **(β, γ)-*bifurcator*** *of size* S, where $\gamma > 1$ and $S \geqslant 0$, just when the following holds:

> Either $N(\mathcal{G}) = 1$, or, by removing no more than S edges, one can partition $\mathcal{G}$ into two subgraphs, each having at most $\lceil (1 - \beta)N(\mathcal{G}) \rceil$ nodes, and each having a **(β, γ)-bifurcator** of size S/γ.

Most applications of bifurcators in the literature have $\beta = 1/2$.

1.4.1.4. Mincing Graphs

Most computational situations which admit graph-theoretic models involve algorithms that require one to partition the underlying graphs into two disjoint subgraphs. However, a number of such situations require a more stringent type of graph partitioning: one must partition the underlying graph into some number $c > 2$ disjoint subgraphs, usually of equal sizes (to within rounding). The scenario that mandates such *mincing* of a graph is typified by the problem of laying out a large electronic system (say, a parallel architecture) on integrated-circuit chips or printed circuit boards. Economic considerations often demand that the system be partitioned into subsystems that are as close as possible to identical in size and structure, where each subsystem resides on a single chip (or a single board).

The study of graph-mincing operates within the following scenario. Let

$\mathcal{G}$ be a graph, and let k be any integer in the set $\{2, 3, \ldots, N(\mathcal{G})\}$. Let $\mathbf{N}_0$, $\mathbf{N}_1, \ldots, \mathbf{N}_{k-1}$ be any *partition* of $\mathbf{N}(\mathcal{G})$ into k equal-size subsets (to within founding); i.e., letting N_i denote the cardinality of the set $\mathbf{N}_i$:

- $\mathbf{N}(\mathcal{G}) = \mathbf{N}_0 \cup \mathbf{N}_1 \cup \cdots \cup \mathbf{N}_{k-1}$.
- $\mathbf{N}_i \cap \mathbf{N}_j = \varnothing$ whenever $i \neq j$.
- $|N_i - N_j| \leqslant 1$ for all i, j.

For each $i \in Z_k$, let $\mathcal{G}_i$ be the induced subgraph of $\mathcal{G}$ on the node-set $\mathbf{N}_i$, and let $\mathcal{G}'$ be the *sum* of all of the $\mathcal{G}_i$, i.e., the (spanning) subgraph of $\mathcal{G}$ defined by

- $\mathbf{N}(\mathcal{G}') = \mathbf{N}(\mathcal{G})$ and
- $\mathbf{E}(\mathcal{G}') = \bigcup_{i=0}^{k-1} \mathbf{E}(\mathcal{G}_i)$.

We write $\mathcal{G}' = \mathcal{G}_0 + \mathcal{G}_1 + \cdots + \mathcal{G}_{k-1}$, and we call $\mathcal{G}'$ a *k-sum subgraph* of $\mathcal{G}$, with *constituents* $\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_{k-1}$

For any graph $\mathcal{G}$ and any integer $k \in \{2, 3, \ldots, N(\mathcal{G})\}$, the *mincing-width* $M_{\mathcal{G}}(k)$ is the smallest number of edges that one must remove from $\mathcal{G}$ in order to partition the graph into subgraphs that collectively form a k-sum subgraph.

1.4.2. Relating Some Themes

We end this section with three important, useful results which can be viewed as initial evidence that the world of separators does not consist of isolated islands.

1.4.2.1. Bifurcators and Balanced Decomposition Trees

In this section we present a result of Bhatt and Leighton [1984] that converts an arbitrary $\sqrt{2}$-**bifurcator** for a graph of degree 4 into a fully balanced $\sqrt{2}$-**bifurcator** for the graph. In fact, we modestly generalize the result to work on graphs of arbitrary degree. For the purposes of this section, we need a more detailed definition of a decomposition tree than we have needed thus far.

Recall that a decomposition tree results from a recursive partitioning of a graph $\mathcal{G}$. If each partitioning produces at most two subgraphs, then it is straightforward to view the decomposition tree as a binary tree. Each node of the tree represents a subgraph of $\mathcal{G}$. If we allow a node to represent the empty subgraph, then we may enforce the requirements that every internal node have precisely two children and that the tree be a complete binary tree.

For the purposes of this section, a *decomposition tree* $\mathcal{T}$ for a graph $\mathcal{G}$ is a complete binary tree satisfying the following properties:

1. Every node x of $\mathcal{T}$ represents a (possibly empty) subgraph $\mathcal{G}_x$ of $\mathcal{G}$.
2. The root of $\mathcal{T}$ represents $\mathcal{G}$; symbolically, $\mathcal{G}_\lambda = \mathcal{G}$.
3. Every leaf x of $\mathcal{T}$ represents either an empty subgraph or a single node.
4. The node-sets associated with the children, $x0$ and $x1$, of each internal node x of $\mathcal{T}$ partition $\mathbf{N}(\mathcal{G}_x)$, i.e., satisfy

$$\mathbf{N}(\mathcal{G}_{x0}) \cap \mathbf{N}(\mathcal{G}_{x1}) = \varnothing$$
$$\mathbf{N}(\mathcal{G}_{x0}) \cup \mathbf{N}(\mathcal{G}_{x1}) = \mathbf{N}(\mathcal{G}_x)$$

We need a series of definitions that isolate relevant features of decomposition trees. Fix an internal node x of the decomposition tree $\mathcal{T}$. The number of edges that are *cut* at x is

$$C(x) \stackrel{\text{def}}{=} |\{(u, v) : u \in \mathbf{N}(\mathcal{G}_{x0}),\ v \in \mathbf{N}(\mathcal{G}_{x1})\}|$$

The number of edges that *flow to the left* at x is

$$L(x) \stackrel{\text{def}}{=} |\{(u, v) : u \in \mathbf{N}(\mathcal{G}_{x0}),\ v \in \mathbf{N} - \mathbf{N}(\mathcal{G}_x)\}|$$

The number of edges that *flow to the right* at x is

$$R(x) \stackrel{\text{def}}{=} |\{(u, v) : u \in \mathbf{N}(\mathcal{G}_{x1}),\ v \in \mathbf{N} - \mathbf{N}(\mathcal{G}_x)\}|$$

Figure 1.4.2 illustrates the sets of edges of $\mathcal{T}$ that are measured by these three numbers. The figure depicts three nodes, internal node x and its two children $x0$ and $x1$, each containing the subgraph of $\mathcal{G}$ represented by the node. The $C(x)$ edges that are cut by the partitioning of $\mathcal{G}_x$ are drawn horizontally from $x0$ to $x1$; in the figure, $C(x) = 3$. The partitioning splits the edges that connect $\mathcal{G}_x$ to the remainder of $\mathcal{G}$ into two parts, the $L(x)$ edges that go left to $\mathcal{G}_{x0}$ and the $R(x)$ edges that go right to $\mathcal{G}_{x1}$; in the figure, $L(x) = 4$ and $R(x) = 6$.

The *node imbalance of* $\mathcal{T}$ *at node* x is

$$\mathbf{NImb}(x) \stackrel{\text{def}}{=} |N(\mathcal{G}_{x0}) - N(\mathcal{G}_{x1})|$$

The *node imbalance* $\mathbf{NImb}(\mathcal{T})$ of $\mathcal{T}$ is the maximum node imbalance at any internal node of $\mathcal{T}$. We say that $\mathcal{T}$ is *fully balanced* if $\mathbf{NImb}(\mathcal{T}) \leqslant 1$. The

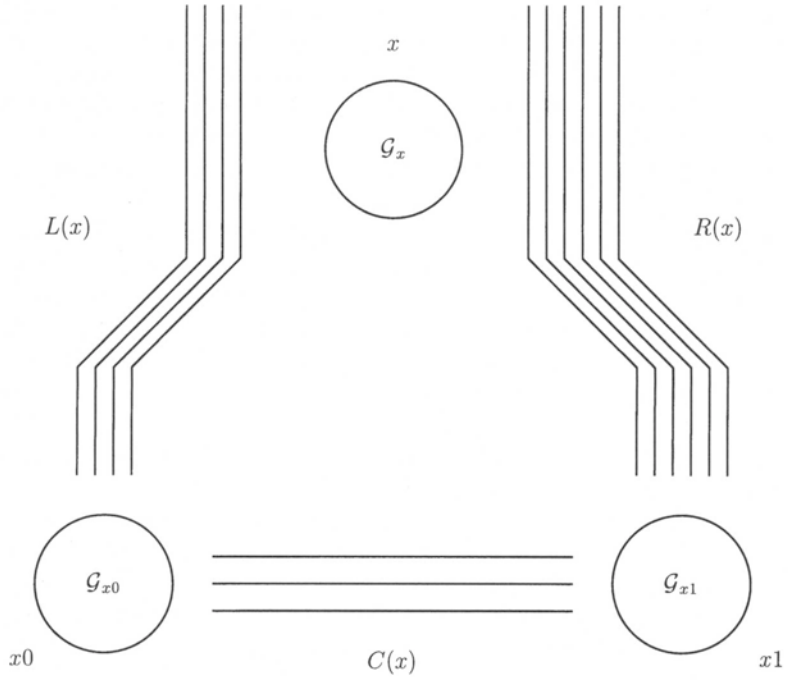


Figure 1.4-2. A typical internal node x and its children in a decomposition tree.

edge imbalance of $\mathcal{T}$ *at node* x is

$$\mathbf{EImb}(x) \stackrel{\text{def}}{=} |L(x) - R(x)|$$

The *edge imbalance* $\mathbf{EImb}(\mathcal{T})$ of $\mathcal{T}$ is the maximum edge imbalance at any internal node of $\mathcal{T}$.

An $(F_0, F_1, \ldots, F_r)$-decomposition tree for the graph $\mathcal{G}$ is a height-$(r + 1)$ complete binary tree $\mathcal{T}$ in which every internal node x at level i, where $0 \leqslant i \leqslant r$, has $C(x) \leqslant F_i$. Observe that for any decomposition tree we can choose a sequence of integers $F_0, F_1, \ldots, F_r$ for which the tree is an $(F_0, F_1, \ldots, F_r)$-decomposition tree. A moment's contemplation should convince the reader that a $\sqrt{2}$-bifurcator of size F for the graph $\mathcal{G}$ is equivalent to an $(F, F/2^{1/2}, F/2, F/2^{3/2}, \ldots, 1)$-decomposition tree for $\mathcal{G}$, in the sense that either the bifurcator or the decomposition tree can easily be transformed into the other. Our goal here is to show that an arbitrary decomposition tree for $\mathcal{G}$ can be transformed into a fully balanced one that has small edge imbalance. Before doing so, it is convenient to digress into the somewhat unexpected realm of strings of pearls.

1.4.2. 1a. Bisecting a String of Pearls. A *string of pearls* is a set of nodes ordered along a line; the nodes represent the pearls in the metaphor, while the ordering yields the string that interconnects the pearls. Say that each node in the string is colored with one of k colors. A beautiful result of Goldberg and West [1985] states that, with very few snips of the string, we can simultaneously bisect the entire string and the nodes of each color.

THEOREM 1.4.1. *Let each node of a string of M pearls be colored with one of k colors from* $\{1,2,\ldots, k\}$ *in such a way that* M_i *nodes have color i. One can snip the string in at most k places and partition the resulting* (*at most* $k + 1$) *substrings into two parts in such a way that* (a) *each part contains either* $\lfloor M/2 \rfloor$ *or* $\lceil M/2 \rceil$ *nodes, and* (b) *for each color* $i \in \{1,2,\ldots, k\}$, *each part contains either* $\lfloor M_i/2 \rfloor$ *or* $\lceil M_i/2 \rceil$ *nodes of color i.*

A moment's reflection should yield the proof for the case $k = 2$. We direct the reader to Goldberg and West [1985] for the quite sophisticated proof of the general case. We content ourselves here with an example.

Figure 1.4-3(a) exhibits a three-colored string of 16 pearls, with three nodes of color *A*, seven of color *B*, and six of color *C*. Figure 1.4-3(b) exhibits three snips that demonstrate the conclusion of the theorem. If one unites the first and third substrings to make part 1 and one unites the second and fourth substrings to make part 2, then each part contains precisely half (i.e., eight) of the nodes; part 1 contains one node of color *A*, four of color *B*, and three of color *C*; part 2 contains two nodes of color *A*, three of color *B*, and three of color *C*. Thus, the set of nodes is bisected, and so also is the set of nodes of each color, as nearly as possible.

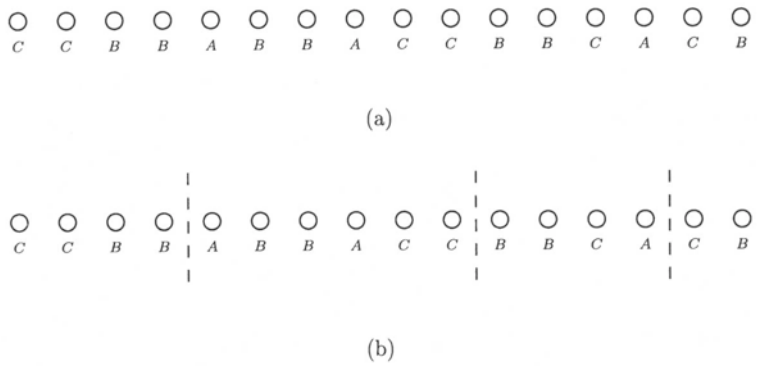


Figure 1.4-3. (a) String of pearls. (b) Bisecting the string.

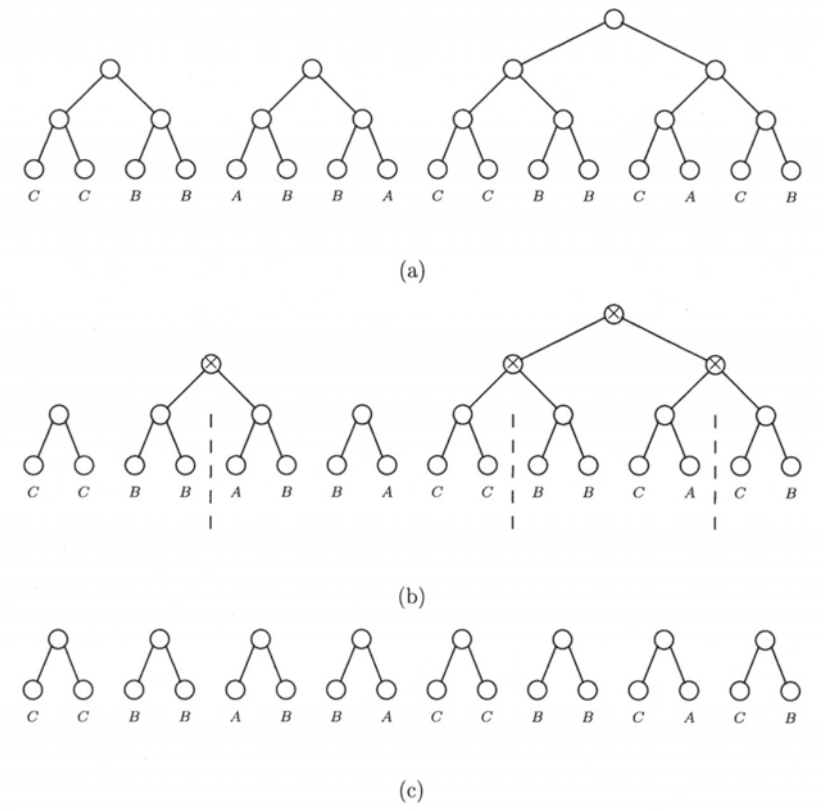


Figure 1.4-4. (a) Forest of complete binary trees. (b) Bisecting the string. (c) Resulting forest.

We now apply Theorem 1.4.1 to the problem of recursively splitting a decomposition tree (or, equivalently, a bifurcator) so as to achieve full balance. A typical splitting takes a forest of complete binary trees (obtained from the original decomposition tree) and produces two such forests, each having half the leaves of the original. At each step in the recursion, the trees in a forest appear in some linear order, with the leaves appearing on a common line, in the order induced by the order on the trees. The leaves of such an ordered arrangement of complete binary trees constitute a string of pearls, whose nodes may be "colored" as before. Figure 1.4-4(a) illustrates a forest whose leaves are exactly the string of pearls of Figure 1.4-3(a). Using this metaphor of a forest as a string of pearls, and letting the *grade* of a tree-node in the forest be its height in its tree, we obtain the following consequence of Theorem 1.4.1.

LEMMA 1.4.2. *Let $\mathcal{F}$ be a forest of complete binary trees, having a total of M leaves, each leaf being colored with one of k colors from $\{1,2,\ldots,k\}$. By deleting no more than k nodes of each grade ≥ 1 from $\mathcal{F}$, one can obtain a new forest that can be partitioned into two forests, $\mathcal{F}_0$ and $\mathcal{F}_1$, in such a way that* (a) *each of $\mathcal{F}_0$ and $\mathcal{F}_1$ contains either $\lfloor M/2 \rfloor$ or $\lceil M/2 \rceil$ leaves, and* (b) *for each color $i \in \{1,2,\ldots,k\}$, each of $\mathcal{F}_0$ and $\mathcal{F}_1$ contains either $\lfloor M_i/2 \rfloor$ or $\lceil M_i/2 \rceil$ leaves of color i.*

PROOF. Start with the previously described arrangement of the forest $\mathcal{F}$, with the leaves viewed as a string of pearls. Choose the (at most) k snips that constitute a k-color bisector for the string, as promised by Theorem 1.4.1. (See Figure 1.4-4(b) for an example that combines Figures 1.4-3(b) and 1.4-4(a).) We claim that any two leaves of $\mathcal{F}$ that have a snip between them will be in different trees of the new forest. This is guaranteed as follows. Focus on a snip that is immediately between two leaves u and v that are adjacent in the string of pearls. If u and v are in different trees of forest $\mathcal{F}$, then they automatically reside in different trees of the new forest. Focus, therefore, on the case when u and v are leaves of the same tree $\mathcal{T}$ of $\mathcal{F}$. These leaves have at most one common ancestor of each grade ≥ 1. We translate the string-snip to tree $\mathcal{T}$ by deleting each of these ancestors, obtaining thereby a collection of complete binary subtrees of $\mathcal{T}$. Once all k string-snips have been translated to the trees of $\mathcal{F}$ in this way, forest $\mathcal{F}$ will have been transformed into a new forest of complete binary trees (see Figure 1.4-4(c) for the final result), which indeed has resulted from the deletion of at most k nodes of each grade ≥ 1 from $\mathcal{F}$. The fact that this new forest can be partitioned into two appropriate subforests $\mathcal{F}_0$ and $\mathcal{F}_1$, as in the statement of the lemma, follows from the conclusions of Theorem 1.4.1. □

1.4.2.1b. On Balancing a Decomposition Tree. Finally we state and prove our main result on balancing a decomposition tree.

THEOREM 1.4.3. *Let $\mathcal{G}$ be a graph with degree $d_{\mathcal{G}}$ and let $\mathcal{T}$ be an $(F_0, F_1, \ldots, F_r)$-decomposition tree for $\mathcal{G}$. Then $\mathcal{G}$ has a fully balanced $(J_0, J_1, \ldots, J_{\lceil \log N \rceil})$-decomposition tree $\mathcal{T}'$ with*

$$\text{EImb}(\mathcal{T}') \leq \binom{d_{\mathcal{G}} + 1}{2}$$

where each J_i satisfies

$$J_i = (d_{\mathcal{G}} + 2) \sum_{a=i}^{r} F_a$$

PROOF. We construct $\mathcal{T}'$ from $\mathcal{T}$ recursively, beginning at the root λ. To distinguish the subgraphs in $\mathcal{T}$ from those in $\mathcal{T}'$, we use $\mathcal{G}_y$ in $\mathcal{T}$ but $\mathcal{G}'_x$ in $\mathcal{T}'$. The base case of the construction is $\mathcal{G}'_\lambda = \mathcal{G}$. Along with the portion of $\mathcal{T}'$ constructed so far, we also maintain a forest $\mathcal{F}$ of complete binary trees that is a subgraph of $\mathcal{T}$. We think of a subset of the complete binary trees as *containing* the set of nodes of $\mathcal{G}$ represented by the leaves of all the trees. This forest satisfies the following property: If x is a node of $\mathcal{T}'$, then there is some subset of the complete binary trees in $\mathcal{F}$ that contains exactly the node-set $\mathbf{N}(\mathcal{G}_x)$. Initially, $\mathcal{F} = \mathcal{T}$ and its single tree contains $\mathbf{N}(\mathcal{G})$.

In the general recursive step, we split the graph $\mathcal{G}'_x$ that corresponds to a nonleaf node x of $\mathcal{T}'$ to obtain the graphs $\mathcal{G}'_{x0}$ and G'_{x1} that correspond to the children of node x. From the current forest $\mathcal{F}$, we identify the subset of the complete binary trees that contain the node set $\mathbf{N}(\mathcal{G}'_x)$. This subset defines a subforest $\mathcal{F}_x$ of $\mathcal{F}$. Assume that the height of every tree in $\mathcal{F}_x$ is at most $r + 1 - |x|$, where $|x|$ is the length of x as a binary string. We justify this assumption at the end of the proof. As in Section 1.4.2.la, the forest $\mathcal{F}_x$ is assumed to be arranged on a line.

Careful application of Lemma 1.4.2 results in the desired split of $\mathcal{G}'_x$. Splitting $\mathbf{N}(\mathcal{G}'_x)$ in a balanced fashion is easy to accomplish with a single snip; it is the bound on edge-imbalance that requires additional snips. Color each leaf z of $\mathcal{F}_x$ as follows:

- If $\mathcal{G}_x$ is the empty graph, then assign color 1 to z.
- Otherwise, let e_z, the *external degree* of z, be the number of edges that connect z to $\mathbf{N} - \mathbf{N}(\mathcal{G}'_x)$. Clearly, $0 \leqslant e_x \leqslant d_{\mathcal{G}}$. Assign color $e_x + 2$ to leaf z.

The described coloring uses at most $d_{\mathcal{G}} + 2$ colors. Apply Lemma 1.4.2 to the forest $\mathcal{F}_x$ to obtain two subforests: $\mathcal{F}_{x0}$, which specifies subgraph $\mathcal{G}'_{x0}$, and $\mathcal{F}_{x1}$, which specifies subgraph $\mathcal{G}'_{x1}$. Because the numbers of leaves of color 1 in each subforest are within 1 of being the same, so also are the numbers of nodes in $\mathcal{G}'_{x0}$ and $\mathcal{G}'_{x1}$. For each d, where $0 \leqslant d \leqslant d_{\mathcal{G}}$, the numbers of nodes of external degree d in $\mathcal{G}'_{x0}$ and $\mathcal{G}'_{x1}$ are within 1 of being the same. An imbalance of 1 contributes at most d to the edge imbalance of x. We arrive, therefore, at the bound

$$\mathrm{EImb}(x) \leqslant \sum_{d=0}^{d_{\mathcal{G}}} d = \binom{d_{\mathcal{G}} + 1}{2}$$

as required. The forest $\mathcal{F}$ is updated by replacing the subforest $\mathcal{F}_x$ with the two subforests $\mathcal{F}_{x0}$ and $\mathcal{F}_{x1}$.

The level of node x in tree $\mathcal{T}'$ is $i = |x|$. The deletion of an internal node y of the forest corresponding to x represents the cutting of $C(y)$ edges. If y is at grade j, then $C(y) \leqslant F_{r-j+1}$. As no tree in the forest has height greater than $r + 1 - |x|$, the grades of the internal nodes of the forest are between 1 and $r + 1 - |x|$. As at most $d_{\mathcal{G}} + 2$ nodes at each grade are deleted, we have the following bound on the number of edges cut at x:

$$\begin{aligned} C(x) &\leqslant (d_{\mathcal{G}} + 2) \sum_{j=1}^{r+1-|x|} F_{r-j+1} \\ &= (d_{\mathcal{G}} + 2) \sum_{j=1}^{r} F_j \\ &= J_i \end{aligned}$$

as required. It remains to observe that the bound on the maximum height of a forest decreases by 1 at each successive level of $\mathcal{T}'$. This follows because the number of nodes of $\mathcal{T}$ assigned to each internal node $x0$ or $x1$ is exactly half the number of nodes assigned to x. Finally, this implies that the height of $\mathcal{T}'$ is $\lceil \log N \rceil + 1$

The following immediate corollary is the most useful for applications in VLSI theory.

COROLLARY 1.4.4. *Let $\mathcal{G}$ be a* **degree-$d_{\mathcal{G}}$** *graph that has a $\sqrt{2}$-bifurcator of size F. Then $\mathcal{G}$ has a fully balanced $(J, J/2^{1/2}, J/2, J/2^{3/2}, \ldots, 1)$-decomposition tree with edge imbalance at most*

$$\binom{d_{\mathcal{G}} + 1}{2}$$

where $J = (d_{\mathcal{G}} + 2)(2 + \sqrt{2})F$.

1.4.2.2. Separators and Bisectors

Say that the graph $\mathcal{G}$ has a *hereditary* **β-separator** of size $S(n)$ if $\mathcal{G}$ *and all of its subgraphs* have **β-separators** of size $S(n)$. One of the earliest results about graph separators was the proof in Lipton and Tarjan [1979] that every planar graph has a hereditary (1/3)-*node*-separation of size $O(\sqrt{n})$. It follows that every *bounded-degree* planar graph has a hereditary (1/3)-*edge*-separator of size $O(\sqrt{n})$. We shall see in Chapter 4 that many other interesting families of graphs enjoy small hereditary separators.

THEOREM 1.4.5. *If the graph $\mathcal{G}$ has a hereditary* ***β-edge-separator*** *of size $S(n)$, where $0 < \beta \leqslant 1/2$, and $S(n)$ is an integer function, then it has a recursive edge-bisector of size $O(S(n) \log n)$. If, moreover, $S(n) = \Theta(n^\delta)$ for some $0 < \delta \leqslant 1$, then $\mathcal{G}$ has a recursive edge-bisector of size $\Theta(n^\delta)$.*

PROOF SKETCH. We establish the following claim by induction on $N(\mathcal{G})$. The theorem will follow from the claim by direct calculation.

Claim. The graph $\mathcal{G}$ described in the statement of the theorem has a recursive bisector of size[6]

$$S(N) + S((1 - \beta)N) + S((1 - \beta)^2 N) + \cdots + S(1)$$

where $N = N(\mathcal{G})$.

We focus on a specific graph $\mathcal{G}$ and assume, for induction, that the claim holds for all graphs having fewer than $N(\mathcal{G})$ nodes. We extend the induction by laying the nodes of $\mathcal{G}$ out on a line in a way that allows us to bisect $\mathcal{G}$ recursively within the bounds of the claim.

We achieve the desired linearization of $\mathcal{G}$ by separating $\mathcal{G}$ recursively using its hypothesized hereditary β-edge-separator of size $S(n)$. We thereby obtain a β-decomposition tree whose leaves implicitly order the nodes of $\mathcal{G}$ linearly; this is the sought linearization. Importantly for the proof, *this ordering clusters the nodes of each decomposition-subgraph $\mathcal{G}_x$ of $\mathcal{G}$ disjointly from the nodes of any other decomposition-subgraph at the same level in the decomposition tree.* Now we "route" the edges of $\mathcal{G}$ "above" the linear layout of the modes. Because of the clustering induced by the decomposition tree, we note that, for each integer $i \geqslant 1$, the line of nodes of $\mathcal{G}$ consists of 2^i consecutive blocks, where

- Each block contains at most $(1 - \beta)^{i-1} N$ nodes (since this is the size of the largest subgraph after the level-i separations).
- Each block consists precisely of the leaves of some subtree of the decomposition tree, that is rooted at an ith-level tree-node (because of the just-noted clustering).
- We need to "route" no more than $S((1 - \beta)^{i-1} N)$ edges of $\mathcal{G}$ between each odd-numbered block $2j + 1$ of nodes (where the leftmost block is block 1) and its next higher block $2j + 2$ (since this is the maximum number of edges cut in any level-i separation).

See Figure 1.4-5.

Now, note that we can bisect $\mathcal{G}$ by means of a vertical line which bisects the line of nodes we have created and which cuts all edges of $\mathcal{G}$ that connect

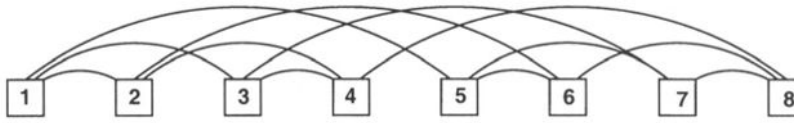


Figure 1.4-5. The linearization of the three-dimensional Boolean hypercube obtained from the decomposition tree of Figure 1.4-1.

nodes $\{1, 2, \ldots, \lfloor N/2 \rfloor\}$ to nodes $\{\lfloor N/2 \rfloor + 1, \ldots, N\}$. It is clear from the preceding analysis of our linearization-plus-routing that this vertical line cuts no more than $S(N) + S((1-\beta)N) + S((1-\beta)^2N) + \cdots + S(1)$ edges of $\mathcal{G}$. Moreover, this bisection yields two disjoint induced subgraphs of $\mathcal{G}$, each of which has fewer nodes than $\mathcal{G}$ and, hence, by induction, has a recursive bisector of size $S(n) + S((1-\beta)n) + S((1-\beta)^2n) + \cdots + S(1)$.

This verifies the claim. The result now follows via calculation

1.4.2.3. Separation-Width and Mincing-Width

The problem of mincing a graph $\mathcal{G}$ is closely related to the problem of partitioning $\mathcal{G}$ into two pieces, in that one can derive good bounds on the k-mincing-width of $\mathcal{G}$ from analogous bounds on the $(N(\mathcal{G})/k)$-separation-width of $\mathcal{G}$. The next theorem formalizes and validates this assertion.

At the present level of generality, we must make the simplifying assumption that the number of subgraphs we are mincing the subject graph $\mathcal{G}$ into divides $N(\mathcal{G})$. In general, of course, one need not encounter such exact divisibility, so the bound of the following theorem holds only up to some error term. Because the M-separation-width of a graph can vary wildly with the value of M (it is certainly *not* monotonic!), there is no way to predict the form of the error term without information about the structure of $\mathcal{G}$. However, the bound of the theorem at least lends one intuition about where to look for a true bound in any specific situation. In Section 4.2.5 we derive detailed bounds on the mincing-width of complete binary trees.

THEOREM 1.4.6. *For any graph $\mathcal{G}$ and for any integer k, the k-mincing-width of $\mathcal{G}$ can be no smaller than*

$$M_{\mathcal{G}}(k) \geq \frac{1}{2}\left((N(\mathcal{G}) \bmod k) W_{\mathcal{G}}\left(\left\lceil \frac{N(\mathcal{G})}{k} \right\rceil\right) + (k - (N(\mathcal{G}) \bmod k)) W_{\mathcal{G}}\left(\left\lfloor \frac{N(\mathcal{G})}{k} \right\rfloor\right)\right)$$

PROOF. Assume for simplicity that k divides $N(\mathcal{G})$, so the sought bound simplifies to

$$M_{\mathcal{G}}(k) \geq \frac{k}{2} W_{\mathcal{G}}\left(\frac{N(\mathcal{G})}{k}\right)$$

The reader should easily be able to supply the clerical details necessary to deal with general k.

Let us begin by mincing $\mathcal{G}$ in any optimal way into a k-sum subgraph with constituents $\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_{k-1}$, i.e., in any way that cuts exactly $M_{\mathcal{G}}(k)$ edges. For $i \in Z_k$, let $\mathbf{E}_i \subset \mathbf{E}(\mathcal{G})$ be the set of edges that connect $\mathcal{G}_i$ to the rest of $\mathcal{G}$. Because cutting the edges in $\mathbf{E}_i$ isolates the k-node graph $\mathcal{G}_i$ from the rest of $\mathcal{G}$, we know that, for each i,

$$|\mathbf{E}_i| \geq W_{\mathcal{G}}\left(\frac{N(\mathcal{G})}{k}\right) \tag{1.4.1}$$

Because cutting the edges in all of the $\mathbf{E}_i$ minces $\mathcal{G}$ into k pieces while cutting the smallest number of edges, we know that

$$M_{\mathcal{G}}(k) = \frac{1}{2} \sum_{i=1}^{k} |\mathbf{E}_i| \tag{1.4.2}$$

The factor 1/2 here accounts for the fact that the summation counts each edge twice, once for each of its endpoints. The theorem now follows by combining (1.4.1) and (1.4.2). □

1.5. Graph Embeddings

The notion of graph embedding has proven important in large variety of applications of graph-theoretic models to the study of computation. Among the many computational areas that have fruitfully been studied within the formal framework of graph embeddings are the

- Mapping one program control structure on another (Lipton *et al.* [1976])
- Mapping "logical" data structures onto "physical" storage structures (DeMillo *et al.* [1978a], Rosenberg (1978])
- Laying out electronic circuits on integrated-circuit chips (Bhatt and Leighton [1984], Leighton and Rosenberg [1986], Leiserson [1983], Valiant [1981]

- Mapping parallel algorithms onto parallel architectures (Bhatt *et al.* [1992], Bokhari [1981], Heath *et al.* [1988])
- Mapping one interconnection network on another (Bhatt *et al.* [1996], Heath [1997], Koch *et al.* [1997], Kosaraju and Atallah [1988]

We shall find the notion quite useful also in the study of graph separators. In particular, we shall observe an interesting synergy between the notions of graph separator and graph embedding, in that the use of embeddings enables a very powerful technique of bounding the size of a graph separator from below (Section 4.2), while the presence of good separators enables one to find efficient embeddings in a large variety of graph families (Section 2.3).

1.5.1. Graph Embeddings and Their Costs

For our purposes, a simple notion of graph embedding, as delimited in Rosenberg [1981], suffices. More elaborate variations on this theme can be found in the previously cited sources (and in Chapter 2).

An *embedding* of the graph $\mathcal{G}$ (the *guest*, or *source*, graph) into the graph $\mathcal{H}$ (the *host*, or *target*, graph) comprises two injective (one-to-one) mappings. The *node-assignment function* α maps $\mathbf{N}(\mathcal{G})$ one-to-one into $\mathbf{N}(\mathcal{H})$. The *edge-routing function* ρ assigns to each edge $\{u, v\} \in \mathbf{E}(\mathcal{G})$ a path in $\mathcal{H}$ that connects nodes $\alpha(u)$ and $\alpha(v)$.

Rather than present a contrived sample embedding at this point, we shall await a series of interesting examples in Section 1.6. Other interesting examples appear in Chapter 2 and Appendix A.

Four fundamental measures of the quality of a graph embedding have proven important in the many applications of embeddings. We focus primarily on these measures throughout the book, although we do present here also a fifth measure that has more limited application. The reader should be aware that a variety of other, special-purpose, measures are useful also in particular studies, as we shall see in Chapter 2.

In order to define the cost measures of interest, let us focus on an embedding $\langle \alpha, \rho \rangle$ of the graph $\mathcal{G}$ into the graph $\mathcal{H}$. We begin with the four primary cost measures. The *dilation* of embedding $\langle \alpha, \rho \rangle$ is the maximum amount that any edge of $\mathcal{G}$ is "stretched" as it is replaced by a path in $\mathcal{H}$. Formally,[7]

$$\mathbf{dilation}(\alpha, \rho) = \max_{(u,v) \in \mathbf{E}(\mathcal{G})} \mathbf{Length}(\rho(u, v))$$

In the special case when the host graph $\mathcal{H}$ is a path, the dilation of $\langle \alpha, \rho \rangle$

is called the *bandwidth* of the embedding. The *bandwidth of a graph* $\mathcal{G}$ is the smallest bandwidth of any embedding of $\mathcal{G}$ into a path.

> The term "bandwidth" originates in the field of numerical analysis. There, one attempts to simplify the solution of large sparse systems of linear equations by performing simultaneous row and column permutations of the matrix M of system coefficients in order to transform M into an equivalent matrix M' all of whose nonzero entries reside in some small number of diagonal "bands" clustered around the main diagonal. The *bandwidth* of matrix M is the smallest number of diagonal "bands" in any matrix M that is equivalent to M. When the matrix M is the adjacency matrix of a graph $\mathcal{G}$, the bandwidth of M is just the dilation of the best embedding of $\mathcal{G}$ into a path.

The *edge-congestion* of embedding $\langle \alpha, \rho \rangle$ is the maximum number of routing paths of the embedding that "cross" any one edge of $\mathcal{H}$. Formally,

$$\text{edge-congestion}(\alpha, \rho) = \max_{e \in \mathbf{E}(\mathcal{H})} |\{e' \in \mathbf{E}(\mathcal{G}) : \text{edge } e \text{ occurs in path } \rho(e')\}|$$

In the special case when the host graph $\mathcal{H}$ is a path, the edge-congestion of $\langle \alpha, \rho \rangle$ is called the *cutwidth* of the embedding. The *cutwidth of a graph* $\mathcal{G}$ is the smallest cutwidth of any embedding of $\mathcal{G}$ into a path.

The *node-congestion* of embedding $\langle \alpha, \rho \rangle$ is the maximum number of routing paths of the embedding that "pass through" any one node of $\mathcal{H}$. Formally,

$$\text{node-congestion}(\alpha, \rho) = \max_{v \in \mathbf{N}(\mathcal{H})} |\{e \in \mathbf{E}(\mathcal{G}) : \text{node } v \text{ occurs in path } \rho(e)\}|$$

The *expansion* of embedding $\langle \alpha, \rho \rangle$ is the ratio of the sizes of $\mathcal{H}$ and $\mathcal{G}$. Formally,

$$\text{expansion}(\alpha, \rho) = \frac{N(\mathcal{H})}{N(\mathcal{G})}$$

The final cost measure that we consider here has attracted much attention in the study of certain applications of graph embeddings (DeMillo [1978a,b], Harper [1964], Iordansk'ii [1976], Lipton and Tarjan [1980], Rosenberg [1978, 1979], Rosenberg and Snyder [1978]), yet has not achieved the popularity of the four primary measures. The *cumulative cost*

of embedding $\langle \alpha, \rho \rangle$ is the cumulative dilation of the edges of $\mathcal{G}$. Formally,

$$\text{cum-cost}(\alpha, \rho) = \sum_{(u,v) \in \mathbf{E}(\mathcal{G})} \text{Length}\,(\rho(u, v))$$

We call this measure "cumulative cost," rather than something like "cumulative-dilation," because it relates as naturally to the edge-congestion measure as to the dilation measure, as we see now.

FACT 1.5.1. *For any embedding* $\langle \alpha, \rho \rangle$ *of graph* $\mathcal{G}$ *into graph* $\mathcal{H}$,

$$\text{average-dilation}(\alpha, \rho) = \frac{1}{E(\mathcal{G})} \text{cum-cost}(\alpha, \rho) \tag{1.5.1}$$

and

$$\text{average-edge-congestion}(\alpha, \rho) = \frac{1}{E(\mathcal{H})} \text{cum-cost}(\alpha, \rho) \tag{1.5.2}$$

PROOF. Equation (1.5.1) being obvious by definition, we focus on verifying (1.5.2). By direct translation from its definition, $\text{cum-cost}(\alpha, \rho)$ can be calculated via the following procedure.

1. Initialize $\text{cum-cost}(\alpha, \rho)$ to 0.
2. For each edge $e \in \mathbf{E}(\mathcal{G})$:

   for each edge $e' \in \mathbf{E}(\mathcal{H})$ that occurs in path $\rho(e)$, add $+1$ to $\text{cum-cost}(\alpha, \rho)$.

By rearranging the order in which edges are encountered within the preceding procedure, one obtains the following equivalent method of calculating $\text{cum-cost}(\alpha, \rho)$:

1. Initialize $\text{cum-cost}(\alpha, \rho)$ to 0.
2. For each edge $e' \in \mathbf{E}(\mathcal{H})$:

   for each edge $e \in \mathbf{E}(\mathcal{G})$ that is routed over edge e' (so that e' occurs in path $\rho(e)$), add $+1$ to $\text{cum-cost}(\alpha, \rho)$.

It is clear that this second procedure, hence the first procedure also, calculates the cumulative congestion of the edges of $\mathcal{H}$ under embedding $\langle \alpha, \rho \rangle$. Hence, (1.5.2) presents a valid expression for the average congestion of the edges of $\mathcal{H}$ under embedding $\langle \alpha, \rho \rangle$. □

1.5.2. Interrelations among the Cost Measures

Our four primary measures of the cost of a graph embedding are interrelated in several ways, some obvious, some rather subtle. In this section we expose just a few of these relationships. Throughout this section we focus on an embedding $\langle \alpha, \rho \rangle$ of a **degree-$d_{\mathcal{G}}$** graph $\mathcal{G}$ into a **degree-$d_{\mathcal{H}}$** graph $\mathcal{H}$.

1.5.2.1. The Influence of Node-Degrees on Dilation

While the node-degrees of $\mathcal{G}$ and $\mathcal{H}$ do not literally "interrelate" with the primary cost measures of embedding $\langle \alpha, \rho \rangle$, they certainly do influence the dilation of the embedding.

Let us focus just on host-graphs $\mathcal{H}$ with $d_{\mathcal{H}} > 2$, for embeddings into paths and cycles require specialized analyses. Pick any maximum-degree node $v \in \mathbf{N}(\mathcal{G})$, and focus on node $\alpha(v) \in \mathbf{N}(\mathcal{H})$. By definition, $d_{\mathcal{G}}$ routing-paths emanate from $\alpha(v)$, and all of these must have distinct terminal nodes, since α is injective. Note that node $\alpha(v)$ has at most $d_{\mathcal{H}}$ neighbors; each of these neighbors can branch out to no more than $d_{\mathcal{H}} - 1$ "new" nodes; each of these neighbors' neighbors can also branch out to no more than $d_{\mathcal{H}} - 1$ "new" nodes; and so on. A simple calculation thus verifies that at least one of the routing-paths that emanates from $\alpha(v)$ must have length no smaller than $\lceil \log_{d_{\mathcal{H}}-1}(d_{\mathcal{G}} - 1) \rceil$. Stated formally,

PROPOSITION 1.5.2. *Any embedding of a* **degree-$d_{\mathcal{G}}$** *graph $\mathcal{G}$ into a* **degree-$d_{\mathcal{H}}$** *graph $\mathcal{H}$ must have dilation D, where*

$$D \geqslant \left\lceil \frac{\log(d_{\mathcal{G}} - 1)}{\log(d_{\mathcal{H}} - 1)} \right\rceil$$

A simple application of Proposition 1.5.2 shows that any embedding of the boolean hypercube $\mathcal{Q}_n$ into a ternary tree, or into a two-dimensional (toroidal or flat) mesh, or even into a more highly interconnected graph such as a de Bruijn graph or a butterfly graph, must have dilation no smaller than $\lceil \log_3 n \rceil$. The limiting fact is that all of the suggested host graphs have node-degree 4.

1.5.2.2. Edge-Congestion and Node-Congestion

Say that in embedding $\langle \alpha, \rho \rangle$, the routing-paths for c edges of $\mathcal{G}$ contain node v of $\mathcal{H}$ (so that c is the congestion at the node). Even if the c routing-paths that contain node v are allocated to the edges incident to v

evenly, some edge incident to v must be crossed by at least $c/d_{\mathcal{H}}$ routing-paths. This elementary reasoning yields the following observation.

PROPOSITION 1.5.3. *Given any embedding of a graph $\mathcal{G}$ into a graph $\mathcal{H}$ of degree $d_{\mathcal{H}}$, the node-congestion C_N and edge-congestion C_E of the embedding must satisfy*

$$C_E \geq \frac{1}{d_{\mathcal{H}}} C_N$$

The reader should be able to instantiate Proposition 1.5.3 by proving that any embedding of the clique $\mathcal{K}_{2^n}$ into the boolean hypercube $\mathcal{Q}_n$ must have edge-congestion $C_E \geq \sqrt{n}\,2^n$. (As a hint, the reader should focus on the middle level of $\mathcal{Q}_n$ to show that $C_N \geq n^{3/2}2^n$.)

1.5.2.3. Congestion and Dilation

Say again that in embedding $\langle \alpha, \rho \rangle$, the routing-paths for c edges of $\mathcal{G}$ contain node v of $\mathcal{H}$. Now, we know that at least $c/d_{\mathcal{G}}$ of these routing-paths originate at distinct nodes of $\mathcal{H}$, since the node-assignment α is injective. Using the same reasoning as in the proof of Proposition 1.5.2, the bound on $\mathcal{H}$'s node-degrees implies that at least one of these distinct nodes must lie at distance no smaller than $\log(c/d_{\mathcal{G}})/\log(d_{\mathcal{H}} - 1)$ from node v. This reasoning yields the following observation.

PROPOSITION 1.5.4. *Given any embedding of a degree-$d_{\mathcal{G}}$ graph $\mathcal{G}$ into a degree-$d_{\mathcal{H}}$ graph $\mathcal{H}$, the dilation D and the node-congestion C_N of the embedding must satisfy*

$$D \geq \frac{1}{\log(d_{\mathcal{H}} - 1)} \log\left(\frac{C_N}{d_{\mathcal{G}}}\right)$$

The message of Proposition 1.5.4 is seen most easily by considering an embedding of a "star" $\mathcal{T}_1(n)$ into a complete binary tree $\mathcal{T}$. The high node-degree of the root of the star manifests itself immediately as congestion near the root's image node in $\mathcal{T}$ and eventually as high dilation for many edges of the star.

1.5.2.4. Expansion and Dilation

Perhaps the subtlest interrelationship among the measures involves the notion of expansion. Indeed, at first blush, expansion seems to be a measure of wasted resources. (When would one ever want to use a bigger host than

necessary?) This intuition is accurate when considering embeddings into paths: the (minimum) bandwidth and cutwidth of any graph $\mathcal{G}$ are achieved by embedding $\mathcal{G}$ into $\mathcal{P}_{N(\mathcal{G})}$. However, at least two important instances have been found wherein other measures of the quality of embeddings can be decreased dramatically only at the cost of significantly increasing the expansion of the embedding (Blum [1985], Hong *et al.* [1983]). The simpler (but earlier) such result appears in Hong *et al.* [1983].

PROPOSITION 1.5.5. (a) *One can embed the complete ternary tree* $\mathcal{T}_h(3)$ *into the complete binary tree* $\mathcal{T}_{2h}(2)$ *with dilation* 2; *this embedding has expansion exponential in h.*

(b) *Any embedding of* $\mathcal{T}_h(3)$ *into the* ***height***-$\lceil \log \frac{1}{2}(3^{h+1} - 1) \rceil$ *complete binary tree—which is the smallest complete binary tree that is big enough to hold it—must have dilation proportional to* log *h.*

Part (a) of Proposition 1.5.5 is a simple exercise that is left to the reader. The proof of part (b) is a straightforward consequence of the development in Section 4.4, wherein we talk about cutting up trees in various ways.

1.6. Quasi-Isometric Graph Families

The literature on graph-theoretic computational models abounds with assertions that two families of graphs are "equivalent for all practical purposes," i.e., are technically indistinguishable within the context of the then-current discussion. Such technical indistinguishability occurs also within the domains we study here, namely, the ease of separating a graph into subgraphs. For instance, in Section 4.2 we establish lower bounds on the separation-widths of "flat" meshes and infer corresponding bounds for toroidal meshes, based on their "technical indistinguishability" from "flat" meshes; in Section 4.3 we proceed along the same street, but in precisely the opposite direction, bounding the bisection-widths of toroidal meshes directly and inferring corresponding bounds for "flat" meshes. The present section is devoted to an interesting mathematical notion of technical indistinguishability among graph families, which is strong enough to apply to virtually any study of graph-theoretic models.

We say that graphs $\mathcal{G}$ and $\mathcal{H}$ are *c-isometric*, where c is a positive integer, if each of $\mathcal{G}$ and $\mathcal{H}$ can be embedded into the other with dilation $\leqslant c$. Of course, 1-isometric graphs are isomorphic. By extension, two indexed families of graphs $\Phi = \{\mathcal{G}_0, \mathcal{G}_1, \ldots\}$ and $\Psi = \{\mathcal{H}_0, \mathcal{H}_1, \ldots\}$ are *quasi-isometric* if there is a constant c such that, for each i, the graphs $\mathcal{G}_i$

and $\mathscr{H}_i$ are c-isometric. One finds in the literature (e.g., Rosenberg and Snyder [1978], among other sources) numerous general structural properties of graph families that preclude quasi-isometry (cf. Proposition 1.5.2), but proofs that establish quasi-isometry tend to be quite specific to the graph families in question.

Most results about graph separators and graph embeddings in the literature hold only up to constant factors, hence do not distinguish between quasi-isometric families of graphs. Within such a constant-forgiving framework, quasi-isometry (which is obviously an equivalence relation) is usually an acceptable formal notion of technical *equivalence* or indistinguishability. We complete this chapter's technical introduction to graph-theoretic notions by establishing the quasi-isometry of a number of pairs of familiar indexed families of graphs.

1.6.1. Paths and Cycles

It is well known in a variety of computational contexts that the families of paths and cycles are quasi-isometric, when graphs are indexed by their sizes. In our context, as in others, one can embed a cycle into a like-sized path, with dilation 2, by carefully "interleaving" the nodes of the cycle. This is the simplest instance of the important operation of node-interleaving, which allows us to embed graphs that have wraparound efficiently into their "flat" analogues.

PROPOSITION 1.6.1 (Quasi-isometry of Paths and Cycles), (a) *For all n, the n-node path $\mathscr{P}_n$ is a subgraph of the n-node cycle $\mathscr{C}_n$; hence, $\mathscr{P}_n$ can be embedded into $\mathscr{C}_n$ with unit dilation.*

(b) *For all n, the n-node cycle $\mathscr{C}_n$ can be embedded into the n-node path $\mathscr{P}_n$ with dilation* 2.

PROOF. Part (a) being obvious, we concentrate on part (b). We note first that cycles of lengths 1 and 2 are degenerate and, hence, cannot appear in graphs that lack loops and parallel edges.

The proof of part (b) is given most elegantly via an algorithm for effecting the desired embedding. To embed the n-node cycle $\mathscr{C}_n$ into $\mathscr{P}_n$, we take an n-step "walk" along $\mathscr{P}_n$, depositing nodes of $\mathscr{C}_n$ as we go. During step i of the "walk", where $i \in \mathbb{Z}_n$, we visit node i of $\mathscr{P}_n$. When i is even, we deposit node $i/2$ of $\mathscr{C}_n$ at node i of $\mathscr{P}_n$; when i is odd, we deposit node $n - \lceil i/2 \rceil$ of $\mathscr{C}_n$ at node i of $\mathscr{P}_n$. See Figure 1.6-1.

Two simple observations suffice to verify that this embedding has dilation 2.

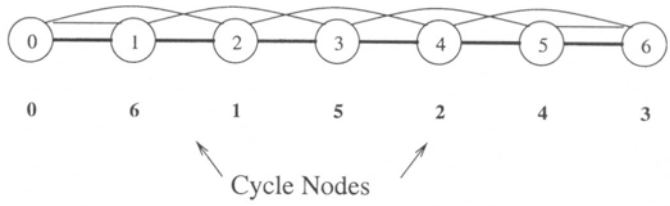


Figure 1.6-1. Illustrating the embedding of $\mathscr{C}_n$ into $\mathscr{P}_n$.

1. Odd and even steps of the "walk" alternate.
2. The following two equations hold:

$$(i + 2)/2 = i/2 + 1.$$

$$n - \lceil (i + 2)/2 \rceil = n - \lceil i/2 \rceil - 1.$$

1.6.2. Mesh-Like Graphs

It is easy to use Proposition 1.6.1 to establish the quasi-isometry of the families of meshes and toroidal meshes, when the graphs in the families are indexed by their side-lengths. Details are left as an exercise.

PROPOSITION 1.6.2 (Quasi-isometry of Meshes and Toroidal Meshes). (a) *For all m and n, them m×n mesh* $\mathscr{M}_{m,n}$ *is a subgraph of the m×n toroidal mesh* $\tilde{\mathscr{M}}_{m,n}$; *hence,* $\mathscr{M}_{m,n}$ *can be embedded into* $\tilde{\mathscr{M}}_{m,n}$ *with unit dilation.*

(b) *For all m and n, the m × n toroidal mesh* $\tilde{\mathscr{M}}_{m,n}$ *can be embedded into the m ×n mesh* $\mathscr{M}_{m,n}$ *with dilation 2.*

PROOF SKETCH. The result is immediate from Proposition 1.6.1 together with the fact that $\mathscr{M}_{m,n} = \mathscr{P}_m \times \mathscr{P}_n$ while $\tilde{\mathscr{M}}_{m,n} = \mathscr{C}_m \times \mathscr{C}_n$. □

1.6.3. Shuffle-Like Graphs

Somewhat less obvious than the preceding two results, but still quite intuitive when one looks at the graphs "in the right way," is the quasi-isometry of the families of de Bruijn and shuffle-exchange graphs, when the graphs in the families are indexed by their orders.

PROPOSITION 1.6.3 (Quasi-isometry of de Bruijn and Shuffle-Exchange Graphs), (a) *For all n, one can embed the order-n de Bruijn graph* $\mathscr{D}_n$ *into the order-n shuffle-exchange graph* $\mathscr{S}_n$, *with dilation 2.*

(b) *For all n, the undirected order-n shuffle-exchange graph* $\mathcal{S}_n$ *is a* (*spanning*) *subgraph of the undirected order-n de Bruijn graph* $\mathcal{D}_n$; *hence,* $\mathcal{S}_n$ *can be embedded into* $\mathcal{D}_n$ *with unit dilation.*

PROOF. The following straightforward pair of embeddings demonstrates that each of $\mathcal{D}_n$ and $\mathcal{S}_n$ is embeddable into the other with dilation 2. This establishes part (a) of the proposition, as well as a weakened, dilation-2 version of part (b). This proof is, therefore, sufficient to verify the quasi-isometry of these two shuffle-oriented graph families.

The straightforward embedding in either direction employs the *identity* node-assignment; that is, node x of the guest graph is assigned to node x of the host graph. Ignoring the shuffle-edges that are common to both guest and host graphs, hence are routed via the identity routing, one verifies the claimed dilation by routing

- Edge $(\beta x, x\bar{\beta})$ of $\mathcal{D}_n$ along the following length-2 path in $\mathcal{S}_n$:

$$\beta x \leftrightarrow x\beta \leftrightarrow x\bar{\beta}$$

- Edge $(x\beta, x\bar{\beta})$ of $\mathcal{S}_n$ along the following length-2 path in $\mathcal{D}_n$:

$$x\beta \leftrightarrow \beta x \leftrightarrow x\bar{\beta}$$

The stronger, unit-dilation, assertion of part (b) of the proposition requires a somewhat more sophisticated embedding of $\mathcal{S}_n$ into $\mathcal{D}_n$. The reader should peruse Figure 1.6-2 while reading the textual description of the embedding. Let x be an arbitrary node of $\mathcal{S}_n$. If x, viewed as a (binary) string, has *even* weight, then assign node x to node x of $\mathcal{D}_n$; if x has *odd* weight, then assign node x to the node x' of $\mathcal{D}_n$ that is obtained by cyclically shifting string x one place to the right. We claim that this node-assignment witnesses the claimed subgraph relation. Note first that, because this assignment is single-valued and onto, it must also be one-to-one. Now consider how the assignment affects the node-adjacencies of $\mathcal{S}_n$.

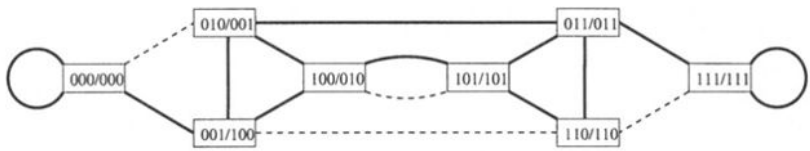


Figure 1.6-2. Illustrating the shuffle-exchange graph as a subgraph of the (undirected) de Bruijn graph; nodes are depicted by (name in shuffle-exchange)/(name in de Bruijn).

Table 1.6-1. Routing the Exchange Adjacencies of $\mathcal{S}_n$ within $\mathcal{D}_n$

x has...	Edge of $\mathcal{S}_n$	Edge of $\mathcal{D}_n$
even weight	$x0 \leftrightarrow x1$	$x0 \leftrightarrow 1x$
odd weight	$x0 \leftrightarrow x1$	$0x \leftrightarrow x1$

Shuffle adjacencies. Each shuffle-edge of $\mathcal{S}_n$ connects nodes of equal weights; therefore, the indicated node-assignment preserves shuffle adjacencies in $\mathcal{S}_n$.

Exchange adjacencies. The nodes connected by an exchange-edge of $\mathcal{S}_n$ have weights of different parities, hence are assigned to nodes of $\mathcal{D}_n$ via different rules. Table 1.6-1 illustrates that the mixed-rule mode-assignment guarantees that each exchange-edge $(x0, x1)$ of $\mathcal{S}_n$ maps to a shuffle-exchange edge of $\mathcal{D}_n$.

These cases complete the proof. □

1.6.4. Butterfly-Like Graphs

One typically finds three families of butterfly-like graphs discussed in the theoretical literature: the butterfly graphs $\mathcal{B}_n$, the FFT graphs $\mathcal{F}_n$, and the cube-connected cycles graphs $\mathcal{CCC}_n$. The literature on parallel architectures is full of numerous other kindred families and of alternative names for all of these families. (Regarding names, for instance, many people refer to $\mathcal{F}_n$ as a "butterfly graph" and to $\mathcal{B}_n$ as a "butterfly graph with wraparound.") We restrict attention to these three families here—of course, using our chosen names for the families—since their detailed structures illustrate the salient distinctions among the more general class of families of related networks. Therefore, the three different modes of reasoning in this section should prepare one to deal with the other related families.

1.6.4.1. Butterfly and Cube-Connected Cycles Graphs

We begin by establishing the quasi-isometry of the families of butterfly and cube-connected cycles graphs when the graphs in the families are indexed by their orders.

PROPOSITION 1.6.4 (Quasi-isometry of Butterfly and Cube-Connected Cycles Graphs). (a) *For all n, one can embed the order-n butterfly graph* $\mathcal{B}_n$ *into the order-n cube-connected cycles graph* $\mathcal{CCC}_n$, *with dilation* 2.

(b) *For all n, the undirected order-n cube-connected cycles graph* $\mathscr{CCC}_n$ *is a (spanning) subgraph of the undirected order-n butterfly graph* $\mathscr{B}_n$*; hence,* $\mathscr{CCC}_n$ *can be embedded into* $\mathscr{B}_n$ *with unit dilation.*

PROOF. We present just a sketch of the straightforward proof of part (a). As was the case in Proposition 1.6.3, this proof can also be adapted to establish a weakened, dilation-2, version of part (b). We employ the *identity* node-assignment. Ignoring straight edges, which are common to $\mathscr{B}_n$ and $\mathscr{CCC}_n$, hence can be routed using the identity routing, we route edge $(\langle \ell, x\rangle, \langle (\ell + 1) \bmod n, x'\rangle)$ of $\mathscr{B}_n$ along the following length-2 path in $\mathscr{CCC}_n$:

$$\langle \ell, x\rangle \leftrightarrow \langle \ell, x'\rangle \leftrightarrow \langle (\ell + 1) \bmod n, x'\rangle$$

For the converse embedding, we route edge $(\langle \ell, x\rangle, \langle \ell, x'\rangle)$ of $\mathscr{CCC}_n$ along the following length-2 path in $\mathscr{B}_n$:

$$\langle \ell, x\rangle \leftrightarrow \langle (\ell + 1) \bmod n, x'\rangle \leftrightarrow \langle \ell, x'\rangle$$

The sketched embeddings clearly have dilation 2. We leave the details as exercises for the reader.

We turn now to the stronger, unit-dilation, version of part (b). Consider the following assignment of nodes of $\mathscr{CCC}_n$ to nodes of $\mathscr{B}_n$. If the PWL string $x \in \mathbb{Z}_2^n$ of node $v = \langle \ell, x\rangle$ of $\mathscr{CCC}_n$ has *even* weight, then assign node v to node v of $\mathscr{B}_n$; if the PWL string has *odd* weight, then assign node v to node $\langle (\ell + 1) \bmod n, x\rangle$ of $\mathscr{B}_n$. We now verify that this assignment witnesses the claimed subgraph relation.

Interlevel adjacencies. Every interlevel edge of $\mathscr{CCC}_n$ maps onto a straight edge of $\mathscr{B}_n$. This is true because each node v of $\mathscr{CCC}_n$ is assigned to the "column" of $\mathscr{B}_n$ that is defined by the same PWL string as v's; either all nodes of $\mathscr{CCC}_n$ assigned to that "column" of $\mathscr{B}_n$ remain in the same level they had in $\mathscr{CCC}_n$, or they all "shift down" one level. In either case, interlevel adjacencies are preserved.

Bijectiveness of assignment. As a corollary of the preservation of interlevel adjacencies, our assignment of nodes is both one-to-one and onto.

Intralevel adjacencies. Each *intra*level edge of $\mathscr{CCC}_n$ connects a node u whose PWL string has even weight with a node v whose PWL string has odd weight. The same is true for each cross edge of $\mathscr{B}_n$, but these latter edges also "shift down" one level. To be more explicit, let us focus on length-n binary strings $x, x' \in \mathbb{Z}_2^n$ that differ in bit-position $\ell \in \mathbb{Z}_n$. On the one hand, there is an edge in $\mathscr{CCC}_n$ that connects nodes $\langle \ell, x\rangle$ and $\langle \ell, x'\rangle$; on the other hand, there is an edge in $\mathscr{B}_n$ that connects nodes $\langle \ell, x\rangle$ and $\langle (\ell + 1) \bmod n, x'\rangle$, as well as an edge that connects nodes $\langle \ell, x'\rangle$ and

$\langle (\ell + 1) \bmod n, x \rangle$. Since one of x and x' must have even weight while the other has odd weight, one of these edges in $\mathcal{B}_n$ must be the image under our node-assignment of the edge in $\mathcal{CCC}_n$.

Since our node-assignment is both one-to-one and onto, and since each edge of $\mathcal{CCC}_n$ is mapped via the assignment to an edge of $\mathcal{B}_n$, it follows that $\mathcal{CCC}_n$ is a spanning subgraph of $\mathcal{B}_n$, as claimed. This completes the proof of part (b). □

1.6.4.2. Butterfly and FFT Graphs

While the families of butterfly and FFT graphs are not literally quasi-isometric, their structures are intimately related in a way that also allows one to argue that the families are almost equivalent in separability and embeddability properties—but, of course, the "almost" here is weaker than it is in the case of truly quasi-isometric families.

PROPOSITION 1.6.5 (Mutual Embeddability of Butterfly and FFT Graphs). (a) *The order-n FFT graph $\mathcal{F}_n$ is a subgraph of the order-$(n + 1)$ butterfly graph $\mathcal{B}_{n+1}$; hence, $\mathcal{F}_n$ can be embedded into $\mathcal{B}_{n+1}$ with unit dilation.*

(b) *The order-n butterfly graph $\mathcal{B}_n$ can be embedded into the order-n FFT graph $\mathcal{F}_n$ with dilation 2.*

PROOF. Part (a) is immediate by definition of the two graph families.

Part (b) requires the most sophisticated embedding of this section. It behooves us, therefore, to approach its proof in two stages. First, we prove that $\mathcal{B}_n$ can be embedded into $\mathcal{F}_n$ with dilation 3. Then we indicate how to adapt the resulting embedding to one that has dilation 2.

Stage A. Embedding $\mathcal{B}_n$ into $\mathcal{F}_n$ with dilation 3. The dilation-3 embedding of $\mathcal{B}_n$ into $\mathcal{F}_n$ employs a rather sophisticated variation on the theme of "node-interleaving," which we touted in Propositions 1.6.1 and 1.6.2 as a useful tool for efficiently embedding a cyclic graph into a structurally related "flat" one. In fact, our embedding of $\mathcal{B}_n$ into $\mathcal{F}_n$ requires a two-phase interleaving. The first phase *macroscopically* interleaves the levels of $\mathcal{B}_n$ into the levels of $\mathcal{F}_n$, thereby emulating the wraparound in $\mathcal{B}_n$ so that we can route the straight edges of $\mathcal{B}_n$ with dilation 2. The second phase *microscopically* interleaves the bit-positions within the PWL strings of the nodes of $\mathcal{B}_n$, in order to allow the correct bit-positions to get flipped, so that we can route the cross edges of $\mathcal{B}_n$ with dilation 3. It is convenient to describe the assignment of nodes of $\mathcal{B}_n$ to nodes of $\mathcal{F}_n$ in two steps, corresponding to the two interleaving phases.

The macroscopic interleaving. Nodes of $\mathcal{B}_n$ that reside at level $k < \lceil n/2 \rceil$ are assigned to nodes at level $2k$ of $\mathcal{F}_n$; nodes of $\mathcal{B}_n$ that reside at level

Table 1.6-2. Level Assignments from Macroscopic Interleaving

Level in $\mathcal{B}_n$	Level in $\mathcal{F}_n$
0	0
$n-1$	1
1	2
$n-2$	3
⋮	⋮
$\lfloor n/2 \rfloor$	$n-1$

$k \geq \lceil n/2 \rceil$ are assigned to nodes at level $2(n-k)-1$ of $\mathcal{F}_n$. Table 1.6-2 illustrates schematically how the macroscopic interleaving assigns levels of $\mathcal{B}_n$ to levels of $\mathcal{F}_n$. Note first that this assignment of levels does not place any nodes of $\mathcal{B}_n$ at level n of $\mathcal{F}_n$. We shall later have to amend this feature in order to achieve the dilation-2 embedding of $\mathcal{B}_n$ into $\mathcal{F}_n$. Note next that this assignment of levels ensures that every straight edge of $\mathcal{B}_n$ can be routed within $\mathcal{F}_n$ with dilation ≤ 2, provided that nodes having the same PWL string in $\mathcal{B}_n$ are assigned to nodes having the same PWL string in $\mathcal{F}_n$. The microscopic interleaving will ensure this condition.

The microscopic interleaving. The macroscopic interleaving just described creates the following problem with the cross edges of $\mathcal{B}_n$. In $\mathcal{B}_n$ a cross edge between levels k and $k+1 \bmod n$ engenders a flip in bit-position k of the PWL strings of the endpoints of the edge; however, when we assign a level-k node of $\mathcal{B}_n$ to a level $\ell \neq k$ of $\mathcal{F}_n$, we inevitably thereby change the association of bit-positions with the flipping mechanism (which now lies in $\mathcal{F}_n$). In order to compensate for this, we must accompany the *macroscopic* assignment of new levels to the nodes of $\mathcal{B}_n$ with a rearrangement of the PWL strings of the nodes. Fortunately, a simple *microscopic* interleaving of $\mathcal{B}_n$, which uniformly rearranges the PWL strings of the nodes, reestablishes the desired association, thereby allowing us to regain the efficient routing of cross edges while retaining the efficient routing of straight edges that was enabled by the macroscopic interleaving. Specifically, we uniformly assign each bit-position in $\mathcal{B}_n$ to a bit-position in $\mathcal{F}_n$ according to the regimen illustrated in Table 1.6-3. We thus see that nodes of $\mathcal{B}_n$ that have the PWL string $\beta_0\beta_1 \cdots \beta_{n-1}$ are assigned to nodes of $\mathcal{F}_n$ that have the PWL string $\beta_{n-1}\beta_0\beta_{n-2}\beta_1 \cdots \beta_{n/2}\beta_{n/2-1}$ when n is even, and to nodes of $\mathcal{F}_n$ that have the PWL string $\beta_{n-1}\beta_0\beta_{n-2}\beta_1 \cdots \beta_{(n-1)/2}$ when n is odd.

Our descriptions of the macroscopic and microscopic interleavings implicitly specify the assignment of nodes of $\mathcal{B}_n$ to nodes of $\mathcal{F}_n$, by specifying both the level and PWL string of the node of $\mathcal{F}_n$ to which each node of $\mathcal{B}_n$ is assigned.

Table 1.6-3. Microscopic Interleaving of Bit-Positions

Bit-position in $\mathscr{B}_n$	Bit-position in $\mathscr{F}_n$
$n-1$	0
0	1
$n-2$	2
1	3
⋮	⋮
k	$\begin{cases} 2(n-k)-2 & \text{if } k \geq \lfloor n/2 \rfloor \\ 2k+1 & \text{if } k < \lfloor n/2 \rfloor \end{cases}$

As we noted earlier, straight edges of $\mathscr{B}_n$ are routed via paths of straight edges of $\mathscr{F}_n$ of lengths 1 or 2. The routing of cross edges is more complicated and depends on the levels in $\mathscr{B}_n$ of the endpoints of the edge. We analyze this latter routing by focusing on how we route within $\mathscr{F}_n$ a cross edge that connects a node from level k of $\mathscr{B}_n$ with a node from level $k + 1 \bmod n$ of $\mathscr{B}_n$. We distinguish four cases, depending on the value of k.

Case 1. $0 \leqslant k < \lceil n/2 \rceil - 1$ (top half of the butterfly). In this case, the macroscopic interleaving requires us to route along a path from level $2k$ to level $2k + 2$ in $\mathscr{F}_n$. The path we choose comprises a straight edge between levels $2k$ and $2k + 1$, followed by a cross edge between levels $2k + 1$ and $2k + 2$; symbolically,[8]

$$(2k) \downarrow [2k+1] \searrow (2k+2)$$

This path flips bit-position $2k + 1$ of the host $\mathscr{F}_n$-node; the microscopic interleaving has associated this host bit-position with bit-position k of the guest $\mathscr{B}_n$-**node,** so this is precisely the bit we want to flip. It follows that the guest edge incurs dilation 2 in the embedding. See Figure 1.6-3.

Case 2. $k = \lceil n/2 \rceil - 1$ (middle level of the butterfly). When n is even, the macroscopic interleaving causes us to route this edge within $\mathscr{F}_n$ via a path from level $n-2$ to level $n-1$; when n is odd, the macroscopic interleaving causes us to route this edge within $\mathscr{F}_n$ via a path from level $n-1$ to level $n-2$. When n is even, we want to flip bit-position $n/2 - 1$ in $\mathscr{B}_n$; when n is odd, we want to flip bit-position $\lceil n/2 \rceil - 1 = \lfloor n/2 \rfloor$ in $\mathscr{B}_n$. Since both possible parities of n mandate flipping bit-positions in $\mathscr{B}_n$ that correspond to bit-position $n-1$ in $\mathscr{F}_n$, we cannot take the shortest path in $\mathscr{F}_n$ between levels $n-2$ and $n-1$, for that path would flip bit-position $n-2$ in $\mathscr{F}_n$. Accordingly, we route the guest edge along a length-3 path in $\mathscr{F}_n$: When n is even, the routing path goes from level $n-2$ to level $n-1$,

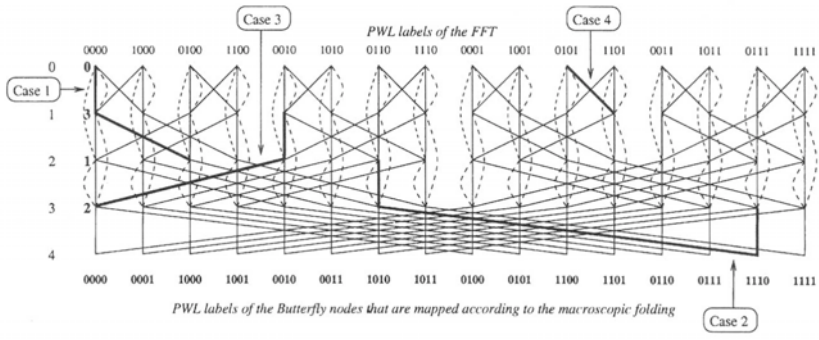


Figure 1.6-3. The four cases for routing $\mathcal{B}_n$'s cross edges in $\mathcal{F}_n$ with dilation 3. Butterfly nodes are labeled in boldface.

by following a straightedge, then a cross edge to level n, then a straight edge from level n to level $n-1$; symbolically,

$$(n-2)\downarrow[n-1]\searrow(n)\uparrow(n-1)$$

When n is odd, the routing path goes from level $n-1$ to level $n-2$ by following the illustrated (even-n) path in the reverse of the indicated order. In either case, the guest edge incurs dilation 3 in the embedding.

Case 3. $\lceil n/2\rceil \leqslant k < n-1$ (bottom half of the butterfly). In this case, the macroscopic interleaving requires us to route along a path from level $2(n-k)-1$ to level $2(n-(k+1))-1 = 2(n-k)-3$ in $\mathcal{F}_n$. The path we choose comprises a cross edge between levels $2(n-k)-1$ and $2(n-k)-2$, followed by a straight edge between levels $2(n-k)-2$ and $2(n-k)-3$; symbolically,

$$(2(n-k)-1)\nearrow[2(n-k)-2]\uparrow(2(n-k)-3)$$

This path flips bit-position $2(n-k)-2$ of the host $\mathcal{F}_n$-node; the microscopic interleaving has associated this host bit-position with bit-position k of the guest $\mathcal{B}_n$-node, so this is precisely the bit we want to flip. It follows that the guest edge incurs dilation 2 in the embedding.

Case 4. $k = n-1$ (last level of the butterfly). In this case, the macroscopic interleaving requires us to route from level 0 to level 1 in $\mathcal{F}_n$. We do so via a cross edge between the levels; symbolically,

$$[0]\searrow(1)$$

Table 1.6-4. Level Assignments from the Two Macroscopic Subinterleavings

Level in $\mathscr{B}_n$	First assignment in $\mathscr{F}_n$	Second assignment in $\mathscr{F}_n$
0	0	1
$n-1$	1	2
1	2	3
$n-2$	3	4
⋮	⋮	⋮
$\lfloor n/2 \rfloor$	$n-1$	n

This path flips bit-position 0 of the host $\mathscr{F}_n$-node; the microscopic interleaving has associated this host bit-position with bit-position $n-1$ of the guest $\mathscr{B}_n$-node, so this is precisely the bit we want to flip. It follows that the guest edge incurs dilation 1 in the embedding.

This routing completes the description and analysis of a dilation-3 embedding of $\mathscr{B}_n$ into $\mathscr{F}_n$.

Stage ***B****. Embedding* $\mathscr{B}_n$ *into* $\mathscr{F}_n$ *with dilation* 2. A bit of reflection will convince the reader that the preceding embedding has dilation 3 largely because it does not make efficient use of level n of $\mathscr{F}_n$. We can remedy this situation in a way that will yield the sought dilation-2 embedding, as follows. We begin by decomposing the macroscopic interleaving depicted in Table 1.6-2 into two subinterleavings, as illustrated in Table 1.6-4.

We begin by partitioning the nodes of $\mathscr{B}_n$ according to their PWL strings $\beta_0\beta_1 \cdots \beta_{n-1}$. Every node of $\mathscr{B}_n$ that has $\beta_{\lceil n/2 \rceil - 1} = \beta_{n-1}$ (or, equivalently, $\beta_{\lceil n/2 \rceil - 1}\beta_{n-1} \in \{00, 11\}$) is assigned its level in $\mathscr{F}_n$ via the *first* assignment in Table 1.6-4; this is the same as its assignment in the dilation-3 embedding. Every node of $\mathscr{B}_n$ that has $\beta_{\lceil n/2 \rceil - 1} \neq \beta_{n-1}$ (or, equivalently, $\beta_{\lceil n/2 \rceil - 1}\beta_{n-1} \in \{01, 10\}$) is assigned its level in $\mathscr{F}_n$ via the *second* assignment in Table 1.6-4. We do not alter the microscopic interleaving.

We claim that this new assignment of the nodes of $\mathscr{B}_n$ to nodes of $\mathscr{F}_n$ allows us to route the edges of $\mathscr{B}_n$ within $\mathscr{F}_n$ via paths of length $\leqslant 2$, thereby yielding the embedding that proves part (b). To verify this claim, we need reconsider only Cases 2 and 4 in our analysis of the dilation-3 embedding, because the new node-assignment does not alter the lengths of the image-paths of any edges of $\mathscr{B}_n$ whose endpoints stay within the same block of the partition that defines the new assignment. One sees easily that the new node-assignment merely cyclically shifts the straight edge cycles of $\mathscr{B}_n$ within $\mathscr{F}_n$, hence does not have these edges cross the partition boundary. Similarly, the only cross edges of $\mathscr{B}_n$ that the new node-assignment forces to cross the partition boundary are those that flip either bit-position $n-1$ or bit-

position $\lceil n/2 \rceil - 1$; these are precisely the cross edges treated in Cases 2 and 4. Let us, therefore, reconsider these cases. (In both cases, we leave the details of the actual routing, which depends on a case analysis, to the reader.)

Case 2′. $k = \lceil n/2 \rceil - 1$. The macroscopic interleaving causes us to route this edge within $\mathcal{F}_n$ either via a path between levels $n-2$ and n or via a path both of whose endpoints stay within level $n-1$; the choice depends on which side of the partition the initial node lies on. When n is even, we want to flip bit-position $n/2-1$ in $\mathcal{B}_n$; when n is odd, we want to flip bit-position $\lceil n/2 \rceil - 1 = \lfloor n/2 \rfloor$ in $\mathcal{B}_n$. Since both possible parities of n mandate flipping bit-positions in $\mathcal{B}_n$ that correspond to bit-position $n-1$ in $\mathcal{F}_n$, we can clearly route the appropriate path via a length-2 path in $\mathcal{F}_n$ that comprises one straight edge and one cross edge. In all cases, the guest edge incurs dilation 2 in the embedding.

Case 4′. $k = n-1$. In this case, the macroscopic subinterleavings require us to route either along a path from level 2 to level 0 in $\mathcal{F}_n$ or along a path that starts and ends within level 1 of $\mathcal{F}_n$; the choice depends on which side of the partition the initial node lies on. In both cases, we employ a length-2 path composed of one straight edge and one cross edge. The cross edge of the path is used to flip bit-position 0 of the host $\mathcal{F}_n$-node; the microscopic interleaving has associated this host bit-position with bit-position $n-1$ of the guest $\mathcal{B}_n$-node, so this is precisely the bit-position we want to flip. Moreover, since flipping this bit-position crosses the partition of the new node-assignment, the path ends up at the correct image-node in $\mathcal{F}_n$. It follows that the guest edge incurs dilation 2 in the embedding. See Figure 1.6-4.

This routing completes the description and analysis of a dilation-2 embedding of $\mathcal{B}_n$ into $\mathcal{F}_n$, hence, completes the proof. □

1.7. Sources

The family of de Bruijn graphs, which is named for its inventor N. G. de Bruijn [1946], is one of the most studied families of graphs among the "interesting" families we consider here. The family plays an important role in coding theory, where its simple structure combines with its properties as an efficient generator of (pseudorandom) sequences (Lempel [1970]). It is of considerable interest in the theory of interconnection networks because of its genealogy as a bounded-degree "approximation" to the family of hypercube interconnection networks, a family extolled for its computational efficiency (Annexstein *et al.* [1990], Leighton [1992], Ullman [1984]) but marred by the structural inefficiency of its unbounded mode-degrees. One of

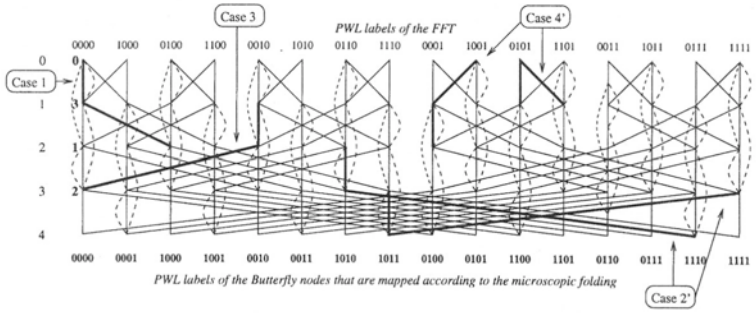


Figure 1.6-4. The modifications to the routing of Case 2 and Case 4 cross edges in order to achieve the dilation-2 embedding of $\mathscr{B}_n$ into $\mathscr{F}_n$.

the truly marvelous properties of de Bruijn networks is their *pancyclicity*: the N-node de Bruijn network, viewed as a digraph with loops and parallel edges, contains, as subgraphs, directed cycles of every length from 1 through N (Yoeli [1962]).

The classical notion of separation seems to have originated in Lipton and Tarjan [1979]; the notion was generalized to (roughly) its present form in Leiserson [1983]. The notion of bisection-width that forms the basis of our more general notion of M-separation-width seems to have originated in Thompson [1980]. The notion of bifurcator originated in Leighton [1982], the conference paper that evolved into Bhatt and Leighton [1984]. Theorem 1.4.5 seems to be part of the folklore of the field; Theorem 1.4.6 derives from Chung and Rosenberg [1986].

Proposition 1.5.5 appears in Hong *et al.* [1983]; it is the first published instance of a cost trade-off within the world of graph embeddings. The remainder of the results in Section 1.5 are part of the folklore (although some are reviewed in Rosenberg and Snyder [1978]).

Propositions 1.6.3(a) and 1.6.4(a) originate in Feldmann and Unger [1992]. The remaining results concerning quasi-isometric families seem to be part of the folklore of the field, with the exception of the dilation-2 portion of the proof of Proposition 1.6.5(b), which is unpublished work of R. Blumofe and S. Toledo [1992].

Notes

1. For each cited area we give only a small list of seminal works, to give the reader a starting place to explore the topic. The various lists of references in the last section of each chapter supplement these lists.

2. Adjacencies can be exploited to devise schemes that generate codewords efficiently; nonadjacencies can be exploited in constructing codes that allow error detection and/or correction.
3. Technically, we say that the set of strings $\mathbf{N}(\mathcal{T})$ is *prefix-closed*.
4. The technical reasons center largely around a quest for regularity.
5. The influential early paper of Lipton and Tarjan [1979] concentrates on classical node-separators.
6. For notational simplicity we assume that each of the indicated arguments to S is an integer. Removing this assumption is conceptually simple but notationally complex.
7. For legibility, we write $p(u, v)$ for $p(\{u, v\})$ and dilation (α, ρ) for dilation $(\langle \alpha, \rho \rangle)$.
8. In the symbolic depictions of routing paths, the bit-position that is flipped has its level-number in square brackets. The arrow $\uparrow$ (resp., $\downarrow$) indicates that a straight edge is taken from level i to level $i - 1 \bmod n$ (resp., to level $i + 1 \bmod n$); the arrow $\nearrow$ (resp., $\searrow$) indicates that a cross edge is taken between these levels.

2

Applications of Graph Separators

2.1. Introduction

This chapter is devoted to motivating the study of graph separators by illustrating a few of their many applications. We have attempted to present applications that use separators in somewhat different ways. Four of our five sample applications benefit in a direct, nontrivial way from the lower-bound techniques that appear in Chapter 4; the illustrations of these benefits appear in the Appendix.

Section 2.2 illustrates the use of graph separators in an algorithmic setting that does not obviously benefit from the divide-and-conquer paradigm, namely, nonserial dynamic programming. The main message of this application is that a serious search for independent subproblems, even in an application where such subproblems do not obviously occur, can reap significant computational advantages. A corollary message is that graph separators can be a valuable tool for identifying independent subproblems in the manifold application areas that admit faithful graph-structured models.

Section 2.3 is devoted to the problem of embedding one graph in another, as defined in Section 1.5 and illustrated in Section 1.6. In contrast to a concrete application of graph separators, such as nonserial dynamic programming, the notion of graph embedding is itself an abstract technique with manifold applications. The development in Section 2.3 illustrates how one can exploit the decomposition structure of a graph $\mathcal{G}$, as embodied in a small recursive bisector for the graph, in order to obtain small-dilation embeddings of $\mathcal{G}$ into a variety of host graphs. We shall see in the Appendix that the embeddings one obtains using the described strategy very often have dilations that are within small logarithmic factors — indeed, often within constant factors — of optimal.

Section 2.4 presents one of the theoretically most satisfying and, one might argue, practically most significant applications of the theory of graph separators. The section first describes a notion of graph embedding which, when coupled with graph-separation notions, leads to an algorithmically simple strategy for laying out circuits within an idealized version of VLSI (Very Large Scale Integrated circuit) technology (Weste and Eshraghian [1988]). It then uses the strategy to explain rigorously a phenomenon that had long been observed in practice, namely, the fact that VLSI layouts of many important classes of circuits look like seas of wires only sparsely populated by transistors. The explanation of (the inevitability of) the preponderance of wires in such layouts builds upon a demonstration that layouts produced via the section's strategy are provably close to optimal with respect to both area and the length of the longest run of wire that is uninterrupted by a transistor. The area of a layout is related to its actual economic cost (monocrystalline silicon is expensive; manufacturing defects increase with area); the wire length of a layout affects the speed of the circuit. The qualifier "close to" signals that optimality is established only to within logarithmic factors. The hitherto unexplained "wire heaviness" of certain circuits then follows from lower bounds of the following sort. Any layout of (a) an N-bit single-pass permuter, (b) an N-bit single-pass circular shifter, (c) a single-pass multiplier of N-bit integers must occupy area proportional to N^2, even though none of these circuits needs contain more than $N \log N$ unit-size transistors.

Section 2.5 focuses on an abstract circuit-layout problem that uses buses to establish arbitrary point-to-point connections. The mathematical framework here is rather different from that of Section 2.3 or 2.4, in that the mathematical objects that host the graph embeddings are *hypergraphs*—graph-like objects each of whose "hyperedges" may connect many nodes, not just two. (In addition to modeling bus-like communication structures, hypergraphs can be used to model circuits that have multipoint nets.) We show how to use graph separators to craft area-efficient layouts of host hypergraphs, which are highly tolerant to "faults" in the hypergraph's nodes. In our abstract framework a "fault" is merely a prohibition to use a given host-node to hold any guest-node. We close the section with a strategy for obtaining lower bounds on the areas of fault-tolerant hypergraphs. In the Appendix we combine the lower-bound results of Chapter 4 with the lower-bound strategies of this section to show that the constructions of Section 2.4 are within constant factors of optimal for a variety of graph families, including the mathematically challenging setting wherein the guest graphs are trees.

Finally, in Section 2.6, we discuss a combinatorial game, called the *pebble game*, which has been used to study (at least) two major computa-

tional problems: register allocation by a compiler and client-server task allocation in a multiprocessor. The game uses tokens (called "pebbles") on a *dag* (short for *directed acyclic graph*) that represents the data dependencies in a computation. The pebbles represent, alternatively, registers that hold data needed as inputs for upcoming tasks or tasks that are eligible for execution. The game proceeds by removing pebbles corresponding to already used data or already executed tasks, and placing new pebbles corresponding to newly available data or newly eligible tasks. One possible goal of the game, which has been studied extensively, is to minimize the allocation of resources during a play of the game, by using as few pebbles as possible. It turns out that the separation-widths of the dag being executed (maximized over all partition ratios) yield lower bounds on the number of necessary pebbles.

2.2. Nonserial Dynamic Programming

This section is devoted to a problem called *nonserial dynamic programming* (Bertele and Brioschi [1972] and Rosenthal [1982]).[1] We present a divide-and-conquer strategy for solving the problem, which allows one to use graph separators to exploit whatever independence exists among the subproblems produced by a dynamic programming solution to the full problem. The strategy we present here is an obvious generalization to graphs of arbitrary structure of the work of Lipton and Tarjan [1980] on planar graphs.

Let us denote by $\langle y_1, y_2, \ldots, y_n \rangle$ a list (or, sequence) of n variables. When the length of the list is implicit or immaterial, we denote the list by $\vec{y}$. Finally, we denote by $\sigma(\vec{y})$ the set of variables in the list $\vec{y}$.

The nonserial dynamic programming problem focuses on a function

$$F(\vec{x}) = F_1(\vec{x}_1) + F_2(\vec{x}_2) + \cdots + F_k(\vec{x}_k)$$

where

- Each F_i is a computable real function.
- The variables in the argument-list $\vec{x}$ of F and in each argument-list $\vec{x}_i$ of F_i range over a finite domain Δ of size δ:
- For each list $\vec{x}_i$, $\sigma(\vec{x}_i) \subseteq \sigma(\vec{x})$.

The problem is to maximize function F over domain Δ. For brevity, we henceforth refer to this problem as "the NSD problem $\mathscr{F} = (F; F_1, F_2, \ldots, F_k)$," or simply "the NSD problem $\mathscr{F}$."

The naive algorithm for solving the NSD problem

$$\mathcal{F} = (F; F_1, F_2, \ldots, F_k)$$

would instantiate the variables in $\sigma(\vec{x})$ with values from Δ in all possible ways, to discover an instantiation that maximizes F. If $\sigma(\vec{x})$ contains n variables, this naive algorithm requires δ^n evaluations of F. The following observation lends hope that we can sometimes find a solution that requires fewer function evaluations, hence less computation time. Say that for some pair of indices i, j, the set $\sigma(\vec{x}_i)$ is disjoint from the set $\sigma(\vec{x}_j)$. Then, letting n_i denote the size of the former set and n_j the size of the latter, one can maximize the sum $F_i(\vec{x}_i) + F_j(\vec{x}_j)$ by maximizing each F_i independently, thereby performing only $\delta^{n_i} + \delta^{n_j}$ function evaluations, in contrast to the $\delta^{n_i + n_j}$ evaluations required by the naive algorithm. The strategy we describe now attempts to identify and exploit subtler instances of the kind of "independence" among variables which is obvious in our example.

The central device in the strategy is the *variable-interaction* (*VI*) *graph* $\mathcal{G}_{\mathcal{F}}$ of the NSD problem $\mathcal{F} = (F; F_1, F_2, \ldots, F_k)$. The node-set of $\mathcal{G}_{\mathcal{F}}$ is the set $\sigma(\vec{x})$. Two node-variables of $\mathcal{G}_{\mathcal{F}}$ are connected by an edge just when they are both needed to evaluate some subfunction F_i. This condition is equivalent to the two variables' coresiding in some $\sigma(\vec{x}_i)$, hence in the corresponding argument-list $\vec{x}_i$.

We need one additional technical definition: for any NSD problem $\mathcal{F} = (F; F_1, F_2, \ldots, F_k)$, the *NSD subproblem of* $\mathcal{F}$ *induced on the set of variables* $A \subseteq \sigma(\vec{x})$ is the NSD problem obtained from $\mathcal{F}$ by deleting all subfunctions F_i such that $A \cap \sigma(\vec{x}_i) = \varnothing$.

In Figure 2.2-1, we schematically specify Algorithm DYNPROG, which exploits the decomposition structure of the VI graph $\mathcal{G}_{\mathcal{F}}$ to accelerate the computation that solves the NSD problem $\mathcal{F}$. Our specification assumes that we have chosen a fraction $0 < \beta < 1$ for which $\mathcal{G}_{\mathcal{F}}$ has a small β-node-separator. One solves the entire NSD problem $\mathcal{F}$ by invoking DYNPROG$(\mathcal{F}; \mathcal{G}_{\mathcal{F}}; \varnothing)$.

It is not hard to analyze the time required to evaluate DYNPROG$(\mathcal{F}; \mathcal{G}_{\mathcal{F}}; \varnothing)$ using Algorithm DYNPROG once one knows how to find small β-node-separators for the VI graphs used in the successive invocations of the algorithm. For simplicity, and to emphasize the benefits gleaned from using separators, we assess unit time for each evaluation of F.

Say that the VI graph $\mathcal{G}_{\mathcal{F}}$ belongs to a family of graphs that have β-node-separators of size $S(n)$ for some nondecreasing integer function $S(n)$. Say, moreover, that one can find these separator sets in time $T(n)$. To be explicit, let us say that in time $T(N(\mathcal{H}))$ one can find a β-node-separator J of size $S(N(\mathcal{H}))$ for the graph $\mathcal{H}$ in the invocation DYNPROG$(\mathcal{F}; \mathcal{H}; A)$ of

Algorithm DYNPROG ($\mathcal{F}$; $\mathcal{H}$; A)

{*The input* $\mathcal{H}$ *to* DYNPROG is *the VI graph of an NSD subproblem of* $\mathcal{F}$; *the input A is a subset of the variables of the NSD problem* $\mathcal{F}$ *that are fixed for this invocation of* DYNPROG.}

Step 1: If the number of un-fixed variables is "small"

then maximize the function F by the naive (exhaustive) algorithm.
{*The value of "small" is chosen later, in a way that balances the recursive algorithm.*}

Step 2: If the number of un-fixed variables is not "small"

then β-separate the graph $\mathcal{H}$ into subgraphs $\mathcal{I}$ and $\mathcal{K}$, each of size $\leqslant(1-\beta)|\mathcal{H}|$, by removing the node-separator J.

Foreach assignment of values in Δ to the variables in $J - A$ do

{*We assign values only to the variables in* $J - A$ *because the variables in A have already been assigned values by the preceding invocation of* DYNPROG.}

1. Invoke DYNPROG($\mathcal{F}$; $\mathcal{I}'$; $J \cup A$), where $\mathcal{I}'$ is the induced subgraph of $\mathcal{H}$ on the variables of $\mathcal{I}$ and of J.
2. Invoke DYNPROG($\mathcal{F}$; $\mathcal{K}'$; $J \cup A$), where $\mathcal{K}'$ is the induced subgraph of $\mathcal{H}$ on the variables of $\mathcal{K}$ and of J.
3. Combine the results of the preceding two invocations to maximize F for the fixed values of the variables in $J \cup A$.

Select that assignment of values to the variables in $J - A$ that maximizes F; return the maximizing assignment, plus the associated value of F.

Figure 2.2-1. An algorithm that uses Vl-graph decomposition to solve the NSD problem $\mathcal{F}$.

Algorithm DYNPROG. Letting $h = N(\mathscr{H})$, it follows that Step 2 of the algorithm cycles through no more than $\delta^{S(h)}$ values for the variables in the separator set J, and that each recursive invocation of the algorithm involves a subgraph of $\mathscr{G}_{\mathscr{F}}$ of size no greater than $(1 - \beta)h + S(h)$. By judicious choice of the value of the quantity "small" in Step 1 of the algorithm, we can ensure the existence of a fraction $0 < \gamma < 1$ for which the following inequality holds for all n:

$$(1 - \beta)n + S(n) \leqslant \gamma n$$

Having found such a γ, we can now assert that the time $\mathscr{T}(\mathscr{F})$ required for Algorithm DYNPROG satisfies the following recurrence:

$$\begin{aligned} \mathscr{T}(n) &\leqslant T(n) + \delta^{S(n)}\mathscr{T}(\gamma n) && \text{if } n \text{ is not "small"} \\ &= O(1) && \text{if } n \text{ is "small"} \end{aligned} \tag{2.2.1}$$

To aid the reader in recognizing how much better Algorithm DYNPROG does than the naive algorithm for solving the NSD problem $\mathscr{F}$, let us hypothesize certain values for the major parameters in recurrence (2.2.1) and solve the resulting recurrence. The reader will be able to follow our lead in instantiating other sample values of the parameters.

Let us assume that $T(n) \leqslant n^a$ for some integer a and that $S(n) \leqslant n^\varepsilon$ for some fraction $0 < \varepsilon < 1$. (These hypothesized functional forms are not unreasonable ones, as attested to by the empirical experience of practitioners who use the Kernighan–Lin heuristic for graph bisection (Kernighan and Lin [1970]; cf. Burstein [1981]). Under these assumptions, recurrence (2.2.1) becomes

$$\begin{aligned} \mathscr{T}(n) &\leqslant n^a + \delta^{n^\varepsilon}\mathscr{T}(\gamma n) && \text{if } n \text{ is not "small"} \\ &= O(1) && \text{if } n \text{ is "small"} \end{aligned} \tag{2.2.2}$$

By expanding recurrence (2.2.2), one verifies easily that

$$\mathscr{T}(n) \leqslant n^a + \delta^{n^\varepsilon}(\gamma n)^a + \delta^{n^\varepsilon + (\gamma n)^\varepsilon}(\gamma^2 n)^a + \cdots$$

Now, the exponents of δ in this expansion form a geometric series, hence are bounded above by δ^{cn^ε} for some constant $c > 0$. This fact allows us to rewrite and refine the inequality as follows:

$$\mathscr{T}(n) \leqslant \delta^{cn^\varepsilon}(n^a + (\gamma n)^a + (\gamma^2 n)^a + \cdots)$$

$$\leqslant \delta^{cn^{\varepsilon}} n^{a} \sum_{i=0}^{\infty} (\gamma^{a})^{i}$$

$$\leqslant d n^{a} \delta^{cn^{\varepsilon}}$$

for some constant $d > 0$. Clearly, this time for solving the NSD problem $\mathscr{F}$ is asymptotically less than the time δ^{n} of the naive algorithm.

The reader will easily find analogous savings for other forms of the functions S and T in recurrence (2.2.1).

2.3. Graph Embeddings via Separators

Experience suggests that if one seeks to find an embedding of a graph $\mathscr{G}$ into a graph $\mathscr{H}$, which optimizes any of the main cost measures of a graph embedding (cf. Section 1.5) then one will have to make use of detailed knowledge of the structure of $\mathscr{G}$ and $\mathscr{H}$. The embeddings of Section 1.6 back up this statement, as do the optimal embeddings of meshes and butterfly-like graphs into hypercubes in Chan [1991] and Greenberg *et al.* [1990], to cite just two relevant sources. Since many applications of graph embeddings model computational situations in which one is not likely to have access to such detailed knowledge, particularly about the guest graph $\mathscr{G}$ (cf. Antonelli and Pelagatti [1992], Berman and Snyder [1987], Bokhari [1981], Snyder [1986]), it is important to understand how far we can get with just limited knowledge of $\mathscr{G}$'s structure. We present in Section 2.3.1 a rather sophisticated (algorithmic) embedding strategy which shows that, if one has a "good" decomposition tree for $\mathscr{G}$ and a "good" embedding of a large complete binary tree into $\mathscr{H}$, then one can produce an embedding of $\mathscr{G}$ into $\mathscr{H}$ with rather "good" dilation. We then demonstrate in Section 2.3.2 that knowledge of the bisection-widths of $\mathscr{G}$ and $\mathscr{H}$ suffices to infer lower bounds on both the dilation and congestion of embeddings of $\mathscr{G}$ into $\mathscr{H}$. One is left with the (accurate) picture that graph separators are a powerful tool in the study of graph embeddings.

2.3.1. Dilation-Efficient Graph Embeddings

The question of which measure of the cost of a graph embedding is the most important has no unique answer, being dependent on the application at hand. However, it does appear that in most applications of embeddings to the study of computational problems, it is desirable to keep the dilation of the embedding—perhaps among other critical cost measures—small. It is significant, therefore, that having access to a good recursive bisector of a

guest graph $\mathcal{G}$ will allow one to embed $\mathcal{G}$ with low dilation into any graph $\mathcal{H}$ that admits small-dilation embeddings of large binary trees. This section presents a strategy that produces these low-dilation embeddings for a broad variety of guest and host pairings. Since the generality of our strategy will force us to ignore constant factors in many places, our actual concern will be with embeddings of the graphs of a given guest family **G** into the (appropriate-size) graphs of a given host family **H.** We now delimit the detailed characteristics of guests and hosts that the strategy requires.

2.3.1.1. Appropriate Guest Graphs

Our embedding strategy starts with a family **G** of guest graphs all of whom have recursive node-bisectors of size $R_{\mathbf{G}}(n) = O(n^{\varepsilon})$ for some absolute constant[2] $\varepsilon \in [0, 1)$. Hence, the strategy is not intended for guest graphs with really large bisection-widths. (In Chapter 4 we identify several such large-width graphs.) Using the techniques of Section 1.4, we could start with any β-node-separator of size $O(n^{\varepsilon})$ for the graphs in **G**, for arbitrary $\beta \in (0, 1/2]$, and produce from it the desired bisector.

2.3.1.2. Appropriate Host Graphs

The strategy we describe embeds a guest graph $\mathcal{G}$ into a host graph $\mathcal{H}$ using a complete binary tree as an intermediary graph. That is, the strategy embeds $\mathcal{G}$ into a tree $\mathcal{T}_h$, where $\lceil \log(N(\mathcal{G}) + 1) \rceil - 1 \leqslant h \leqslant \lfloor \log(N(\mathcal{H}) + 1) \rfloor - 1$, and then embeds $\mathcal{T}_h$ into $\mathcal{H}$. These bounds on h ensure that $\mathcal{T}_h$ is big enough to "hold" $\mathcal{G}$ and that $\mathcal{H}$ is big enough to "hold" $\mathcal{T}_h$. The separator-based portion of the strategy—which is what we describe here—focuses only on the first of the two embeddings. We refer only tangentially to the second embedding, via the following notion.

Let us focus on the host graph $\mathcal{H}$ in our embedding. Let us say that the complete binary tree $\mathcal{T}_h$, where $h \leqslant \lfloor \log(N(\mathcal{H}) + 1) \rfloor - 1$, can be embedded into $\mathcal{H}$ with dilation D_h. Then we say that $\mathcal{H}$ has β_h-*balance* D_h, where

$$\beta_h = \frac{2^{h+1} - 1}{2^{\lfloor \log(N(\mathcal{H}) + 1) \rfloor} - 1}$$

Note that, to within low-order terms, this is essentially saying that $\mathcal{T}_h$ can be embedded into $\mathcal{H}$ with dilation D_h and expansion $1/\beta_h$. Perforce, D_h is nondecreasing as h increases. In order to lend the reader some intuition about the range of applications of the strategy we are about to describe, let us remark on the balance of a few important families of graphs.

de Bruijn Graphs. The de Bruijn network $\mathcal{D}_n$ contains the complete binary tree $\mathcal{T}_{n-1}$, which is the largest complete binary tree that $\mathcal{D}_n$ is big enough to "hold," as a subgraph; hence $\mathcal{D}_n$ has 1-balance 1. One verifies this easily by mapping each node $x \in \mathbf{N}(\mathcal{T}_{n-1})$ to node $0^{n-|x|-1}1x \in \mathbf{N}(\mathcal{D}_n)$.

Boolean Hypercubes. The boolean hypercube $\mathcal{Q}_n$ does not contain $\mathcal{T}_{n-1}$, which is the largest complete binary tree that $\mathcal{Q}_n$ is big enough to "hold," as a subgraph; hence, $\mathcal{Q}_n$ does not have 1-balance 1.

> It is an interesting exercise for the reader to verify that $\mathcal{T}_{n-1}$ is not a subgraph of $\mathcal{Q}_n$. An easy verification begins with the observation that both graphs are bipartite but that they have different ratios of "red" and "green" nodes.

However, one can embed $\mathcal{T}_{n-1}$ into $\mathcal{Q}_n$ with dilation 2; therefore, $\mathcal{Q}_n$ has 1-balance 2. This efficient embeddability is verified by the following embedding (among others), which appears in Bhatt *et al.* [1992]. The embedding assigns nodes of $\mathcal{T}_{n-1}$ to nodes of $\mathcal{Q}_n$ by performing an in-order traversal of $\mathcal{T}_{n-1}$, starting from its leftmost leaf, 0^{n-1}. During the traversal, one labels each tree-node with its (ordinal) position in the traversal. If one starts counting with 0, then the length-n binary representation of each tree-node's label is its assignment to a node of $\mathcal{Q}_n$. One verifies easily (try it!) that each left edge of $\mathcal{T}_{n-1}$ gets mapped onto an edge of $\mathcal{Q}_n$ via this node-assignment, while each right edge gets mapped onto a path of length 2. A "better," but more complicated, embedding, which dilates precisely one edge of $\mathcal{T}_{n-1}$ (to length 2) while mapping all other edges to edges of $\mathcal{Q}_n$ appears in Wu [1985]. This better embedding demonstrates that $\mathcal{T}_{n-2}$ is a subgraph of $\mathcal{Q}_n$, whence $\mathcal{Q}_n$ has (1/2)-balance 1 (ignoring the low-order term in the balance fraction).

Butterfly Networks. A rather complex embedding of complete binary trees into butterfly networks in Bhatt *et al.* [1996a] proves that $\mathcal{B}_n$ has (1/8)-balance $\leqslant 6$. (Again, we ignore the low-order term in the balance fraction.)

The notion of balance extends to families of graphs as follows. We say that the family of host graphs $\mathbf{H}$ *has balance* $B_{\mathbf{H}}$ if there exists a constant $\varphi \in (0, 1]$ such that every graph $\mathcal{H} \in \mathbf{H}$ has φ-balance $B_{\mathbf{H}}$. Thus, the families of boolean hypercubes, de Bruijn networks, and butterfly networks all have balance $O(1)$.

2.3.1.3. The Embedding Strategy

We now indicate precisely what the strategy of this section achieves, phrased in terms of the notion of balance.[3]

THEOREM 2.3.1. *Let* $\mathbf{G}$ *be a family of maxdegree-*$d_{\mathbf{G}}$ *graphs that has a recursive bisector of size* $R_{\mathbf{G}}(n) = O(n^{\varepsilon})$ *for some absolute constant* $\varepsilon \in (0, 1)$, *and let* $\mathbf{H}$ *be a graph family having balance* $B_{\mathbf{H}}$. *Any graph* $\mathcal{G} \in \mathbf{G}$ *can be embedded into some graph* $\mathcal{H} \in \mathbf{H}$, *with simultaneous dilation*

$$O(\log d_{\mathbf{G}} + \log R_{\mathbf{G}}(N(\mathcal{G})))B_{\mathbf{H}}$$

and expansion $O(1)$.

There are numerous examples where the bound of Theorem 2.3.1 cannot be improved. Chapter 4 will give us the wherewithal to discover such examples. We turn now to a proof of the theorem.

PROOF. As we have indicated, we exhibit here only the first step of the two-step embedding that proves the theorem. Specifically, we illustrate how to embed each graph in $\mathbf{G}$ into some complete binary tree efficiently (in the sense of the theorem); we then rely on the balance of the family $\mathbf{H}$ to complete the proof (noting that dilations and expansions of composed embeddings combine multiplicatively). In more detail, we embed any given $\mathcal{G} \in \mathbf{G}$ into the complete binary tree $\mathcal{T}_{\lg N(\mathcal{G})}$, where $\lg n \stackrel{\text{def}}{=} \lceil \log n \rceil$, with dilation $O(\log(d_{\mathbf{G}} \Sigma_i R_{\mathbf{G}}(2^{-i}N(\mathcal{G}))))$; perforce, the embedding has expansion $O(1)$. We turn now to the details of the proof.

Our detailed proof employs the following refinement of the notion of bisector; cf. Bhatt and Leighton [1984]. Let k be a positive integer, and let $R(n)$ be a nondecreasing integer function. The graph $\mathcal{G}$ has a *k-color recursive node-bisector of size* $R(n)$ if $N(\mathcal{G}) < k$ or if the following holds for every way of labeling the nodes of $\mathcal{G}$ (independently) with one of k possible labels: By removing $\leqslant R(N(\mathcal{G}))$ nodes from $\mathcal{G}$, one can partition $\mathcal{G}$ into subgraphs $\mathcal{G}_0$ and $\mathcal{G}_1$ such that

1. $|N(\mathcal{G}_0) - N(\mathcal{G}_1)| \leqslant 1$; that is, graphs $\mathcal{G}_0$ and $\mathcal{G}_1$ are equal in size, to within one node.
2. Let ℓ be one of the k labels and letting $N_\ell(\mathcal{G}_i)$ denote the number of nodes of graph $\mathcal{G}_i$ that have label ℓ, for each label ℓ, $|N_\ell(\mathcal{G}_0) - N_\ell(\mathcal{G}_1)| \leqslant 1$.
3. Each of $\mathcal{G}_0$ and $\mathcal{G}_1$ has a k-color recursive bisector of size $R(n)$.

Note that a 1-color recursive node-bisector is just the standard notion of a recursive node-bisector.

Using techniques from Section 4 of Bhatt and Leighton [1984], the reader can prove the following crucial technical lemma, which states that

k-color bisectors need not be very much bigger than "ordinary" 1-color bisectors. (See also a similar result developed in Section 1.4.2.1.)

LEMMA 2.3.2. *For any integer k and graph $\mathcal{G}$, one can convert a recursive node-bisector of size $R(n)$ for $\mathcal{G}$ into a k-color recursive node-bisector of size $R_k(n) \leqslant k\Sigma_i R(2^{-i}n)$ for $\mathcal{G}$. Hence, when $R(n) = n^{\Theta(1)}$, then $R_k(n) = O(kR(n))$.*

RETURN TO THE PROOF. Our embedding of $\mathcal{G}$ into $\mathcal{T}_{\lg N(\mathcal{G})}$ uses the following auxiliary structure, which appears (in slightly different form) in Bhatt *et al.* [1992]. A *bucket tree for* $\mathcal{G}$ is a complete binary tree, each of whose level-ℓ nodes where $\ell \in \mathbf{Z}_{1+\lg N(\mathcal{G})}$, is a *bucket* that is capable of holding

$$C(\ell) = c_G R(2^{-\ell} N(\mathcal{G})) \log d_G$$

nodes of $\mathcal{G}$, for some fixed constant c_G to be chosen later (in the proof of Lemma 2.3.3). We embed $\mathcal{G}$ into $\mathcal{T}_{\lg N(\mathcal{G})}$ in two stages: First, we "embed" $\mathcal{G}$ into a bucket tree via a many-to-one node-assignment[4] that "respects" bucket capacities (always placing exactly $C(\ell)$ nodes of $\mathcal{G}$ into each level-ℓ node of the bucket tree) and has "dilation" $O(\log d_G)$. Then we "spread" the contents of the bucket tree's buckets within $\mathcal{T}_{\lg N(\mathcal{G})}$ to achieve an embedding of $\mathcal{G}$ into the tree, with the claimed dilation. The first stage of this embedding process is described in the following section.

"Embedding" & into a Bucket Tree

LEMMA 2.3.3. *The graph $\mathcal{G}$ can be "embedded" into a bucket tree in such a way that* (a) *exactly $C(\ell)$ nodes of $\mathcal{G}$ are assigned to each level-ℓ node of the bucket tree*; (b) *nodes that are adjacent in $\mathcal{G}$ are assigned to buckets that are at most distance $O(\log d_G)$ apart in the bucket tree.*

PROOF. Our goal is to make the bucket tree mimic a decomposition tree for $\mathcal{G}$ that is formed using an $O(\log d_G)$-color recursive node-bisector of size $R(n)$, by populating the buckets with the removed bisector nodes. (An appropriate constant of proportionality, hidden at this moment in the preceding big O, will be chosen during the course of analyzing our "embedding" algorithm.) The strength of this strategy is its automatically ensuring that successively smaller sets of bisector nodes get deposited in successively lower-level buckets of the bucket tree. The weakness of this strategy is that it may not fill the buckets at each level of the bucket tree uniformly. To remedy this weakness, we place nodes other than bisector nodes into the buckets, in order to fill all buckets to capacity. We use the

colors of the multicolor node-bisector to select the nodes we place in each bucket, thereby controlling the "dilation" of the "embedding."

Our procedure for mapping $\mathcal{G}$ into a bucket tree is described in Algorithm BUCKET in Figure 2.3-1. The algorithm uses the following notation.

- Bucket B_λ is the root of the bucket tree.[5]
- Inductively, for $x \in \{0, 1\}^*$, buckets B_{x0} and B_{x1} are the children in the bucket tree of bucket B_x. For example, buckets B_0 and B_1 are the children of bucket B_λ, B_{00} and B_{01} are the left grandchildren of B_λ, B_{10} and B_{11} are the right grandchildren of B_λ, and so on.
- For integers a and b, define

$$a(\text{MOD } b) = \begin{cases} a \bmod b & \text{if } a \bmod b \neq 0 \\ b & \text{if } a \bmod b = 0 \end{cases}$$

Verification and Analysis of Algorithm BUCKET. We claim that the algorithm's allocation of nodes of $\mathcal{G}$ to buckets satisfies both the bucket-capacity condition (a) and the "dilation" condition (b) of Lemma 2.3.3. Once we specify our choice of the parameter r (quite soon!), the reader will see that the latter condition is transparently enforced when certain colored nodes are automatically placed in buckets (in Step t.**0**). We demonstrate that the former condition is also enforced, by proving that the recursive bisection of $\mathcal{G}$ and the concerns about "dilation" in the bucket tree never force us to place more than $C(\ell)$ nodes in any level-ℓ bucket. This demonstration takes the form of an analysis of the described assignment, simplified by omitting all the substeps that mandate adding "enough extra nodes to [the] bucket . . . to fill the bucket to capacity" (specifically, Steps s.**3** and t.**3**). To the end of the analysis, let $G(k)$ denote the number of nodes of $\mathcal{G}$ that are assigned to a bucket at level $k - 1$ of the bucket tree. We claim that $G(k)$ obeys the recurrence

$$G(k) \leqslant 2^{-r} \cdot d_G \cdot G(k - r) + O(rR(2^{-k}N(\mathcal{G}))) \tag{2.3.1}$$

with initial conditions

$$\begin{aligned} G(1) &\leqslant R(N(\mathcal{G})) \\ G(2) &\leqslant 2R(2^{-1}N(\mathcal{G})) \\ G(3) &\leqslant 3R(2^{-2}N(\mathcal{G})) \\ &\vdots \\ G(r) &\leqslant rR(2^{-r+1}N(\mathcal{G})) \end{aligned}$$

Algorithm BUCKET: Mapping $\mathcal{G}$ into a bucket tree

{The value of the "color" parameter r will be chosen later.}

Step 0. {Initial coloring.}

0.1. Let $\mathcal{G}_\lambda = \mathcal{G}$.

0.2. Initialize every node of $\mathcal{G}_\lambda$ to color 0.

Step s. ($s = 1, 2, \ldots, r$) {Initial r bisections.}

For each subgraph $\mathcal{G}_y$ ($y \in \{0,1\}^{s-1}$) of $\mathcal{G}$ created in Step $s - 1$:

s.1. Bisect the graph $\mathcal{G}_y$ using an s-color recursive bisector, thereby creating the graphs $\mathcal{G}_{y0}$ and $\mathcal{G}_{y1}$.

s.2. Place the removed bisector nodes into bucket B_y of the bucket tree.

s.3. If necessary, add enough extra nodes to bucket B_y, taken equally from $\mathcal{G}_{y0}$ and $\mathcal{G}_{y1}$, to fill the bucket to capacity.

s.4. Recolor every 0-colored nodes of $\mathcal{G}$ that is adjacent to a node in bucket B_y with color s.

Step t. ($t = r + 1, r + 2, \ldots, \lg N(\mathcal{G})$) {All remaining bisections.}

For each subgraph $\mathcal{G}_y$ ($y \in \{0, 1\}^{t-1}$) of $\mathcal{G}$ created in Step $t - 1$:

t.0. Place every node of color t (MOD r) into bucket B_y.

t.1. Bisect the graph $\mathcal{G}_y$ using an $(r + 1)$-color recursive bisector, thereby creating the graphs $\mathcal{G}_{y0}$ and $\mathcal{G}_{y1}$.

t.2. Place the removed bisector nodes into bucket B_y of the bucket tree.

t.3. If necessary, add enough extra nodes to bucket B_y, taken equally from $\mathcal{G}_{y0}$ and $\mathcal{G}_{y1}$, to fill the bucket to capacity.

t.4. Recolor every 0-colored node of $\mathcal{G}$ that is adjacent to a node in bucket B_y with color t (MOD r).

Figure 2.3-1. An algorithm for embedding $\mathcal{G}$ into a bucket tree.

The recurrence and its initial conditions are justified as follows.

- The initial conditions reflect the sizes of the appropriately colored recursive node-bisectors of $\mathcal{G}$: at each step $s \in \{1, 2, \ldots, r\}$, one uses an s-color recursive node-bisector; at all subsequent steps, one uses an $(r + 1)$-color recursive node-bisector.
- At levels $\ell > r - 1$, the buckets contain not only bisector nodes, which are proportional to $rR(2^{-k}N(\mathcal{G}))$ in number; they contain also the nodes of $\mathcal{G}$ that are placed in the bucket to satisfy the "dilation" requirements. The former nodes account for the term $O(rR(2^{-k}N(\mathcal{G})))$ in recurrence (2.3.1); cf. Lemma 2.3.2. The latter nodes comprise all neighbors of the $G(k - r)$ occupants of the distance-r ancestor bucket that have not yet been placed in any other bucket. Since nodes of $\mathcal{G}$ can have no more than d_G neighbors, and since our $(r + 1)$-color node-bisections allocate these neighbors equally among the descendants of a given bucket, these "dilation"-generated nodes can be no more than

$$\lceil 2^{-r} \cdot d_G \cdot G(k - r) \rceil \leqslant 2^{-r} \cdot d_G \cdot G(k - r) + 1$$

  in number.

Thus, the bisector-nodes produced by the recursive node-bisectors, together with the "dilation"-generated neighbors of these nodes (in $\mathcal{G}$), account for the occupants of the buckets and for the recurrence counting them.

Now, one shows by induction that the term $O(rR(2^{-k}N(\mathcal{G})))$ dominates in recurrence (2.3.1) as long as the inequality

$$2^{-r} \cdot d_G \cdot R(2^{r-k}N(\mathcal{G})) < R(2^{-k}N(\mathcal{G})) \tag{2.3.2}$$

holds at each step of the recurrence. Given that $R(n) = O(n^\varepsilon)$ for some absolute constant $\varepsilon \in [0, 1)$, we can ensure the persistence of inequality (2.3.2) by choosing

$$r > \frac{1}{1 - \varepsilon} \log d_G$$

(So r is specified at last!) In other words, if we choose r to be an appropriate fixed-constant multiple of $\log d_G$, then we have

$$G(k) \leqslant C(k - 1) = O(R(2^{-k}N(\mathcal{G})) \log d_G)$$

(Bounding the big O here specifies the constant c_G.) This completes the proof of Lemma 2.3.3. □

Emptying the Buckets into the Host Tree

Our final task is to refine the $O(\log d_G)$-"dilation" assignment of Lemma 2.3.3 to a bona fide embedding of $\mathcal{G}$ into $\mathcal{T}_{\lg N(\mathcal{G})}$, with dilation

$$O(\log d_G + \log R(N(\mathcal{G})))$$

We proceed inductively, emptying buckets into $\mathcal{T}_{\lg N(\mathcal{G})}$ in such a way that each node of $\mathcal{G}$ is assigned to a unique node of the tree. Let c^* be a constant to be specified later. For each $x \in \{0,1\}^*$, let $\mathcal{T}^{(x)}$ be the complete binary tree of height $c^* \log(d_G R(N(\mathcal{G})))$ rooted at node x of $\mathcal{T}_{\lg N(\mathcal{G})}$. Our goal is to deposit the contents of the buckets in such a way that all nodes in each bucket B_x get placed within tree $\mathcal{T}^{(x)}$.

- Place the $C(0) = O(R(N(\mathcal{G})) \log d_G)$ elements of bucket B_λ, in any order, but as densely as possible, in the topmost levels of $\mathcal{T}_{\lg N(\mathcal{G})}$. Easily, there is a constant c_0 such that

$$h_\lambda \stackrel{\text{def}}{=} \lceil \log(R(N(\mathcal{G})) \log d_G) \rceil + c_0$$

  levels suffice for this task. Let all of our trees $\mathcal{T}^{(x)}$ start with h_λ levels; this is our first step in determining the constant c^*. If the bucket elements fill only m nodes of levels h_λ of $\mathcal{T}^{(\lambda)}$, then partition those m bucket-elements into two sets that are within 1 of each other in size. Place the larger of these sets in the leftmost nodes of the level, i.e., in nodes $0, 1, \ldots, \lceil m/2 \rceil - 1$; place the other set in nodes $2^{h_\lambda - 1}, 2^{h_\lambda - 1} + 1, \ldots, 2^{h_\lambda - 1} + \lfloor m/2 \rfloor - 1$ of the level. This redistribution of nodes assigned to level h_λ is an instance of a process we term *evening out* the bucket being unloaded. (We describe this process more fully imminently.)
- Because we evened out bucket B_λ, there are unoccupied nodes at level $h_\lambda - 1$ of both $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$. Place the contents of bucket B_0 into $\mathcal{T}^{(0)}$, starting immediately where we stopped placing the elements of bucket B_λ. Place the contents of bucket B_1 analogously into $\mathcal{T}^{(1)}$, again starting immediately where we stopped placing the elements of bucket B_λ. Then, even out both buckets within these trees, in just the way that we evened out bucket B_λ. By inspection of bucket capacities (Lemma 2.3.3), we conclude that only c_1 new levels are required to empty the new buckets, for some constant c_1. Let us "expand" all

trees $\mathcal{T}^{(x)}$ where x has length ≥ 1 to height $h_\lambda + c_1$.

We continue to empty buckets, level by level, into $\mathcal{T}_{\lg N(\mathcal{G})}$ in much the manner just described (evening out each bucket load), possibly increasing the heights of the subtrees $\mathcal{T}^{(x)}$ by some constant amount at each level. One verifies easily that after some constant number of levels, we need use only (part of) one more level of $\mathcal{T}_{\lg N(\mathcal{G})}$ in order to empty the next level of buckets. (This is, of course, because the levels of the tree are doubling in size.) At this point the heights of the subtrees $\mathcal{T}^{(x)}$ need never be increased further. Because these heights have been increased by (additive) constants only constantly many times, the constant c^* posited earlier is sure to exist.

The general procedure for *evening out* a bucket proceeds as follows: To *even out* bucket B_x, do the following.

- If B_x has *more* nodes than are available at the first partially empty level of $\mathcal{T}^{(x)}$, then proceed as in the case of $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$: Fill up this level of $\mathcal{T}^{(x)}$, and continue into the next level. Allocate the nodes of B_x that reach the lowest partially filled level of $\mathcal{T}^{(x)}$ equally (to within one) between the left and the right half of the level.
- If B_x has *fewer* nodes than are available at the first partially empty level of $\mathcal{T}^{(x)}$, then *merge* the nodes of B_x with the nodes already assigned to the level (in any order) and allocate the composite set equally (to within one) between the left and the right half of the level.

We now verify that we have achieved our goals.

1. The described procedure produces an embedding of $\mathcal{G}$ into $\mathcal{T}_{\lg N(\mathcal{G})}$, since each node of $\mathcal{G}$ is assigned to a unique node of the tree.
2. The embedding has expansion $O(1)$. To wit, $\mathcal{T}_{\lg N(\mathcal{G})}$ has at most twice as many nodes as does $\mathcal{G}$; the number of tree-nodes left unoccupied by our placement procedure is no greater than the number of buckets in the bucket tree; finally, all buckets at each level ℓ of the bucket tree have the same population $C(\ell)$, so after unloading all buckets at each level of the bucket tree, all subtrees $\mathcal{T}^{(x)}$ have the identical pattern of occupany.
3. The embedding has the desired dilation, namely,

$$O(\log d_G + \log R(N(\mathcal{G})))$$

This follows from our procedure's method of spreading bucket contents throughout $\mathcal{T}_{\lg N(\mathcal{G})}$. Specifically:

- Each of the subtrees $\mathcal{T}^{(x)}$ has height $O(\log(R(N(\mathcal{G}))\ \log d_G))$: $\mathcal{T}^{(\lambda)}$ starts with such a height. Subsequent subtrees $\mathcal{T}^{(x)}$ with short index-strings x may have slightly larger height, but only by an additive constant.
- All subtrees $\mathcal{T}^{(x)}$ whose index-strings x exceed some fixed constant in length have the same height, because the roots of such trees descend in $\mathcal{T}_{\lg N(\mathcal{G})}$ at the same rate (or faster) than the levels of $\mathcal{T}_{\lg N(\mathcal{G})}$ which we use to house bucket contents.
- Since each bucket B_x is emptied completely into subtree $\mathcal{T}^{(x)}$, the least common ancestor in $\mathcal{T}_{\lg N(\mathcal{G})}$ of the set comprising the contents of any bucket plus the nodes in buckets at most $O(\log d_G)$ buckets up (which lie in adjacent levels $k, k+1, \ldots, k + O(\log d_G)$ of the bucket tree) are always within a subtree of height

$$O(\log d_G + \log(R(N(\mathcal{G}))\ \log d_G))$$

of $\mathcal{T}_{\lg N(\mathcal{G})}$.

To summarize: Consider the path in $\mathcal{T}_{\lg N(\mathcal{G})}$ between a node v that resides in bucket y and the root of the subtree for the bucket that is $O(\log d_G)$ levels above y. All but (possibly) a constant number of the $O(\log d_G)$ subtrees that correspond to buckets encountered on the way from y to its $O(\log d_G)$ th ancestor have the same height; therefore, each contributes at most a single edge to the path. The subtrees for the remaining buckets between y and its $O(\log d_G)$th ancestor are each of height at most $O(R(N(\mathcal{G}))\ \log d_G)$, so that their collective contribution to the pathlength is at most $O(R(N(\mathcal{G}))\ \log d_G)$. The desired bound, namely,

$$O(\log d_G + \log(R(N(\mathcal{G}))\ \log d_G)) = O(\log d_G + \log R(N(\mathcal{G})))$$

follows. This completes the proof of Theorem 2.3.1. □

We close this section with two remarks which place Theorem 2.3.1 in technical and historical perspective. Our proof of the theorem builds on the availability of a "balanced" decomposition tree for the guest graph $\mathcal{G}$; it does not exploit in any way the particular mechanism used to produce that tree. For definiteness we have used (colored versions of) recursive node-bisectors to produce the trees, because settling on a particular decomposition mechanism allows us to adduce quantitative information about the embedding process. Translating our embedding scheme to another decomposition mechanism, e.g., the bifurcators of Bhatt and Leighton [1984], is a purely clerical procedure.

There have recently appeared two sophisticated embedding strategies that can sometimes control *congestion* as well as dilation in embeddings. The first strategy modifies the embedding produced by our proof of Theorem 2.3.1; it is introduced in Bhatt *et al.* [1996a] (which is the source of the theorem) where embeddings into butterfly graphs are studied. The second strategy replaces bucket trees with an alternative intermediate host graph; it is introduced in Obreni [1994], where it is exemplified with embeddings into hypercubes and de Bruijn graphs.

2.3.2. Lower Bounds on Efficiency

In this section we survey some simple results that suggest why differences in the separation characteristics of a graph $\mathcal{G}$ and a graph $\mathcal{H}$ influence the efficiency of embeddings of $\mathcal{G}$ into $\mathcal{H}$. We quantify this efficiency in terms of the congestions (Section 2.3.2.1), dilations (Section 2.3.2.2), and cumulative costs (Section 2.3.2.3) of the embeddings. In order to convey our message in the simplest possible setting, we restrict attention to the scenario in which the guest graph $\mathcal{G}$ and the host graph $\mathcal{H}$ are like-sized, i.e., $N(\mathcal{G}) = N(\mathcal{H})$, and, with the inevitable exception of Section 2.3.2.3, we restrict attention to the *bisection* characteristics of $\mathcal{G}$ and $\mathcal{H}$ rather than more general separation ratios. The reader will easily recognize ways to relax these restrictions; it is particularly easy to extend our arguments to allow $N(\mathcal{H})$ to be additively larger than $N(\mathcal{G})$.

> It is a simple exercise to verify that the first of these restrictions loses no generality when the host-graph is a path: one can never decrease bandwidth or cutwidth by increasing the size of the path one embeds one's guest graph into.

Throughout this section let us focus on an arbitrary embedding $\langle \alpha, \rho \rangle$ of the guest graph $\mathcal{G}$ into the like-sized host graph $\mathcal{H}$. Say throughout that $\mathcal{G}$ has bisection-width $W_{\mathcal{G}}$ and that $\mathcal{H}$ has a recursive edge-bisector of size $R_{\mathcal{H}}(n)$. All of the results in the first two subsections of this section follow from simple variations on the following chain of reasoning.

> Let us choose a bisection of $\mathcal{H}$ that removes no more than $R_{\mathcal{H}}(N(\mathcal{H}))$ edges; call these the *host bisection edges.* Since any bisection of $\mathcal{H}$ automatically bisects $\mathcal{G}$ also (by dint of the embedding, because $N(\mathcal{G}) = N(\mathcal{H})$), we know that at least $W_{\mathcal{G}}$ edges of $\mathcal{G}$ must be routed across the host bisection edges by the edge-routing function ρ. Since the maximum edge-congestion on the host bisection edges is no smaller than the average edge-congestion on these edges, we know that (at least) one host

bisection edge, call it e, must have no fewer than $w = W_{\mathcal{G}}/R_{\mathcal{H}}(N)$ edges of $\mathcal{G}$ routed across it by embedding $\langle \alpha, \rho \rangle$.

2.3.2.1. Bounds on Congestion

Our first lower bound, which focuses on the edge-congestion of the (arbitrary) embedding $\langle \alpha, \rho \rangle$, follows directly from the argument just presented.

PROPOSITION 2.3.4. *If the N-node graph $\mathcal{G}$ has bisection-width $W_{\mathcal{G}}$, and the N-node graph $\mathcal{H}$* has a recursive edge-bisector *of size $R_{\mathcal{H}}(n)$, then any embedding of $\mathcal{G}$ into $\mathcal{H}$ must have edge-congestion $C_{\mathrm{E}} \geq W_{\mathcal{G}}/R_{\mathcal{H}}(N)$.*

The bound of Proposition 2.3.4 is often quite close to being tight, especially when the guest graph $\mathcal{G}$ has a recursive edge-bisector of size $R_{\mathcal{G}}(n)$, where $R_{\mathcal{G}}(N)$ is very close to $W_{\mathcal{G}}$. Also, the argument that proves the proposition extends in a transparent way to embeddings with nonunit expansion: one just replaces the edge-bisectors of the argument with appropriate edge-separators.

When the host graph $\mathcal{H}$ is a path, the bound of Proposition 2.3.4 can be strengthened by removing the restriction on the relative sizes of $\mathcal{G}$ and $\mathcal{H}$.

PROPOSITION 2.3.5. *The cutwidth of a graph $\mathcal{G}$ can be no smaller than its bisection-width $W_{\mathcal{G}}$.*

PROOF SKETCH. For any embedding of $\mathcal{G}$ into a path, one need only consider the congestion on the edge of the path that has images of half of the nodes of $\mathcal{G}$ (to within rounding) on either side of it. □

2.3.2.2. Bounds on Dilation

Continuing our discussion of the embedding $\langle \alpha, \rho \rangle$ of $\mathcal{G}$ into $\mathcal{H}$, let us focus now on the respective node-degrees, $d_{\mathcal{G}}$ and $d_{\mathcal{H}}$, of $\mathcal{G}$ and $\mathcal{H}$. Since no node of $\mathcal{G}$ has degree exceeding $d_{\mathcal{G}}$, we can invoke the maximum-versus-average principle to show that the $w = W_{\mathcal{G}}/R_{\mathcal{H}}(N)$ congesting edges of $\mathcal{G}$ emanate from at least $w/d_{\mathcal{G}}$ distinct nodes of $\mathcal{G}$. Since no node of $\mathcal{H}$ has degree exceeding $d_{\mathcal{H}}$, at least one of these $w/d_{\mathcal{G}}$ "source" nodes of $\mathcal{G}$ must be placed by the embedding no closer than distance

$$\left(\frac{1}{\log d_{\mathcal{H}}}\right) \log \frac{w}{d_{\mathcal{G}}} - 1 = \left(\frac{1}{\log d_{\mathcal{H}}}\right) \log \frac{W_{\mathcal{G}}}{d_{\mathcal{G}} R_{\mathcal{H}}(N)} - 1$$

from the endpoint of the congested edge e. Since we have been discussing an arbitrary embedding of $\mathcal{G}$ in $\mathcal{H}$, we have proved the following.

PROPOSITION 2.3.6. *Let the N-node graph $\mathcal{G}$ have bisection-width $W_{\mathcal{G}}$, and let the N-node graph $\mathcal{H}$ have a recursive edge-bisector of size $R_{\mathcal{H}}(n)$. If $\mathcal{G}$ has maximum node-degree $d_{\mathcal{G}}$ and $\mathcal{H}$ has maximum node-degree $d_{\mathcal{H}}$, then the dilation D of any embedding of $\mathcal{G}$ into $\mathcal{H}$ must satisfy*

$$D \geq \left(\frac{1}{\log d_{\mathcal{H}}}\right) \log \frac{W_{\mathcal{G}}}{d_{\mathcal{G}} R_{\mathcal{H}}(N)} - 1$$

When the host graph $\mathcal{H}$ is a path, we get a strengthened version of Proposition 2.3.6.

PROPOSITION 2.3.7. *If the graph $\mathcal{G}$ has bisection-width $W_{\mathcal{G}}$ and maximum node-degree $d_{\mathcal{G}}$, then it has bandwidth $B \geq W_{\mathcal{G}}/d_{\mathcal{G}}$.*

PROOF. The bound follows from the same reasoning as does Proposition 2.3.6; the conclusions of the two results differ because any path $\mathcal{H}$ has $R_{\mathcal{H}}(N) \equiv 1$. Therefore, the $\mathcal{H}$-edge e that is highly congested under embedding $\langle \alpha, \rho \rangle$ must be "carrying" edges of $\mathcal{G}$ that have at least $W_{\mathcal{G}}/d_{\mathcal{G}}$ distinct source nodes. One of these source nodes must be placed by the embedding at distance $\geq W_{\mathcal{G}}/d_{\mathcal{G}}$ from an endpoint of edge e. □

Note that no restriction on the size of the host path is needed for the bound of Proposition 2.3.7.

2.3.2.3. A Bound on Cumulative-Cost

An *edge-separator set* of graph $\mathcal{H}$ is a subset of $\mathbf{E}(\mathcal{H})$ whose removal partitions $\mathcal{H}$ into two disjoint subgraphs. The *yield* of an edge-separator set is the number of nodes in the smaller of the resulting subgraphs. The reader will, of course, recognize that edge-separator sets and their yields underlie the entire study in this book.

PROPOSITION 2.3.8. *For each integer $M \leq N(\mathcal{G})$, let graph $\mathcal{G}$ have M-separation-width $W_{\mathcal{G}}(M)$. Let the graph $\mathcal{H}$ have pairwise disjoint edge-separator sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$. Then the cumulative cost of any embedding of*

$\mathcal{G}$ into $\mathcal{H}$ can be no smaller than

$$\sum_{i=1}^{k} W_{\mathcal{G}}(\text{yield}(\mathcal{B}_i))$$

PROOF. Each edge-separator set $\mathcal{B}_i$ clearly effects a yield$(\mathcal{B}_i)$-edge-separation of graph $\mathcal{H}$. Because graphs $\mathcal{G}$ and $\mathcal{H}$ have equal-size node-sets, each $\mathcal{B}_i$ effects a yield$(\mathcal{B}_i)$-edge-separation of graph $\mathcal{G}$ also. By definition, this latter edge-separation must cut at least $W_{\mathcal{G}}(\text{yield}(\mathcal{B}_i))$ edges of $\mathcal{G}$; hence, it must incur congestion at least this great on the edges in $\mathcal{B}_i$. Since the edge-separator sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$ are pairwise disjoint, we have, for any embedding $\langle \alpha, \rho \rangle$ of $\mathcal{G}$ into $\mathcal{H}$:

$$\begin{aligned}\text{cum-cost}(\alpha, \rho) &= \sum_{e \in \mathbf{E}(\mathcal{H})} \text{congestion on edge } e \\ &\geqslant \sum_{i=1}^{k} \sum_{e \in \mathcal{B}_i} \text{congestion on edge } e \\ &\geqslant \sum_{i=1}^{k} W_{\mathcal{G}}(\text{yield}(\mathcal{B}_i))\end{aligned}$$

□

To illustrate the use of Proposition 2.3.8, we present three of its immediate corollaries, with proofs left to the reader.

2.3.2.3a. Paths. Let us consider first the N-node path $\mathcal{P}_N$ and its exhaustive collection of singleton edge-separator sets, i.e., the $N - 1$ sets $\mathcal{B}_i \stackrel{\text{def}}{=} \{(i-1, i)\}$, for $i \in \{1,2,\ldots,N-1\}$. Using this collection, we infer immediately from Proposition 2.3.8 the following bound on the cumulative costs of embeddings into paths.

COROLLARY 2.3.9. *For any N-node graph $\mathcal{G}$, the cumulative cost of any embedding of $\mathcal{G}$ into $\mathcal{P}_N$ is no smaller than $\Sigma_{i=1}^{N-1} W_{\mathcal{G}}(i)$.*

2.3.2.3b. Trees. Next, let us consider the height-h complete binary tree $\mathcal{T}_h$ and its exhaustive collection of singleton edge-separator sets, i.e., the $N - 1$ sets $\mathcal{B}_i \stackrel{\text{def}}{=} \{(\lfloor i/2 \rfloor, i)\}$, for $i \in \{1,2,\ldots,N-1\}$. Using this collection, we infer immediately from Proposition 2.3.8 the following bound on the cumulative costs of embeddings into complete binary trees.

COROLLARY 2.3.10. For any $(2^{h+1} - 1)$*-node graph $\mathcal{G}$, the cumulative cost of any embedding of $\mathcal{G}$ into $\mathcal{T}_h$ is no less than $\Sigma_{i=1}^{h} 2^i W_{\mathcal{G}}(2^{h+1-i} - 1)$.*

2.3.2.3c. Meshes. Finally, let us consider the side-n d-dimensional mesh $\mathcal{M}_n(d)$. In this case, too, we consider an exhaustive collection of edge-separator sets, namely, the collection $\{\mathcal{B}_{i,j}\}_{1 \leq i < N, 0 \leq j < d}$, where each set $\mathcal{B}_{i,j}$ comprises precisely those edges $(\langle u_0, u_1, \ldots, u_{d-1} \rangle, \langle v_0, v_1, \ldots, v_{d-1} \rangle)$ of $\mathcal{M}_n(d)$ for which

$$u_j = i - 1; \quad v_j = i; \quad u_k = v_k \text{ for } k \neq j$$

Using this collection, we infer immediately from Proposition 2.3.8 the following bound on the cumulative costs of embeddings into meshes.

COROLLARY 2.3.11. *For any n^d-node graph $\mathcal{G}$, the cumulative cost of any embedding of $\mathcal{G}$ into $\mathcal{M}_n(d)$ is no less than $\Sigma_{i=1}^{n-1} i^{d-1}$.*

2.4. Laying Out VLSI Circuits

Our notion of the layout of a circuit on a VLSI "chip" follows the framework originated in Thompson [1980], refined and developed in Bhatt and Leighton [1984], Leiserson [1983], and Valiant [1981], and studied extensively in myriad subsequent sources. Within this abstract framework, circuits are viewed as undirected graphs whose nodes correspond to active devices (transistors, gates, etc.) and whose edges correspond to wires connecting these devices. The media in which the circuits are to be realized—be they chips or wafers or printed circuit boards (cf. Weste and Eshraghian [1988])—are viewed as two-dimensional rectangular meshes. A circuit layout is a restricted type of embedding of the circuit graph into the mesh, the restrictions being enumerated below. This model is generalized in a variety of interesting ways to three-dimensional meshes, representing three-dimensional chips, wafers, and circuit boards (cf. Etchells *et al.* [1981]), in Greenberg and Leiserson [1988], Leighton and Rosenberg [1983, 1986], Preparata [1983], and Rosenberg [1983]. We restrict attention to the two-dimensional version of the layout problem in this chapter, because it already exposes all of the underlying conceptual ideas. Thus, motivating scenarios aside, the topic of this section is a restricted class of embeddings of undirected graphs into the family of rectangular meshes $\{\mathcal{M}_{m,n} \mid m, n \geq 1\}$.

A *layout of the graph $\mathcal{G}$ in the mesh $\mathcal{M}_{m,n}$* comprises an embedding $\langle \alpha, \rho \rangle$ of $\mathcal{G}$ into $\mathcal{M}_{m,n}$, in which the routing-map ρ, which associates each edge (u, v) of $\mathcal{G}$ with a unique path $\rho(u, v)$ in $\mathcal{M}_{m,n}$ that connects node $\alpha(u)$ with node $\alpha(v)$, satisfies the following two restrictions.

1. All of the paths $\rho(u, v)$ are mutually *edge-disjoint*, i.e., do not share any edge.
2. No path $\rho(u, v)$ passes through (i.e., contains) any node-image $\alpha(w)$ other than $\alpha(u)$ and $\alpha(v)$.

The *area of a layout* of the graph $\mathcal{G}$ in the mesh $\mathcal{M}_{m,n}$ is the product mn of the dimensions of $\mathcal{M}_{m,n}$. The *area of the graph* $\mathcal{G}$, denoted AREA($\mathcal{G}$), is the minimum area of any layout of $\mathcal{G}$ in a mesh.

2.4.1. A Provably Efficient Layout Strategy

This section is devoted to showing that one can use any "efficient" decomposition tree for a graph $\mathcal{G}$, obtained through some sort of graph separation, to generate an area-"efficient" layout of $\mathcal{G}$ in a mesh.[6] We place the word "efficient in quotation marks here because, with most genres of graph separator, the layouts obtained can range in quality from area-optimal to area-awful, even if one uses the best possible separator of that type (Leighton [1982]). However, if one uses a decomposition tree for $\mathcal{G}$ that comes from a small-size $\sqrt{2}$-bifurcator for $\mathcal{G}$,[7] then one is guaranteed to get a layout that is within a predictable, small deviation from optimality. Since our interest here is in illustrating the usefulness of separators rather than in developing a theory of graph layouts, we present a somewhat simpler layout strategy than appears in Bhatt and Leighton [1984]; therefore, our layouts suffer a rather larger possible deviation from optimality than do the layouts in that source. Specifically, our layouts can be roughly two logarithmic factors from optimal in area.

THEOREM 2.4.1. *Let S be the size of the smallest $\sqrt{2}$-bifurcator of the graph $\mathcal{G}$. Then there is a constant* $c > 0$ *such that*

$$\max(N(\mathcal{G}), cS^2) \leqslant \text{AREA}(\mathcal{G}) \leqslant (4 + \tfrac{8}{2}\sqrt{2}) \max(N(\mathcal{G}), S^2 \log^2 N(\mathcal{G}))$$

PROOF. We simplify the proof by exploiting the robustness of the notion of bifurcator as illustrated in Bhatt and Leighton [1984]. Specifically, for the lower-bound portion of the proof, we employ the least demanding notion of bifurcator, which does not require any particular balance in the sizes of the subgraphs produced by each partition in the recursive decomposition of $\mathcal{G}$; for the upper-bound portion of the proof, we employ the most demanding notion of bifurcator, which insists that all partitions are, in fact, bisections. The constant c in the statement of the theorem is the square of the constant-factor difference in sizes of these two genres of bifurcator. As

an aside: it is far from intuitive that these two notions of bifurcator should differ in size by only a constant factor; however, such is the case (Bhatt and Leighton [1984]).

2.4.1.1. The Lower Bound

We remark first the lower bound, $\mathbf{AREA}(\mathcal{G}) \geq N(\mathcal{G})$, that follows trivially from the fact that each node of $\mathcal{G}$ occupies a node of $\mathcal{M}_{m,n}$. Let us concentrate, therefore, on the area accounted for by the edges of $\mathcal{G}$.

Assume that we start with a *minimum-area* layout of the graph $\mathcal{G}$ in the $m \times n$ mesh $\mathcal{M}_{m,n}$ (so $\mathbf{AREA}(\mathcal{G}) = mn$). We shall inductively decompose $\mathcal{G}$ implicitly by inductively decomposing $\mathcal{M}_{m,n}$ explicitly. For the sake of the induction, let $\mathcal{G} = \mathcal{G}_{00}$, $m = m_0$, and $n = n_0$; for the sake of clerical simplicity, which sacrifices no conceptual aspects of the proof, let m_0 and n_0 be powers of 2.

At each stage $i \geq 0$ of the induction, we assume that we have layouts of 2^i graphs $\{\mathcal{G}_{ij} \mid 0 \leq j < 2^i\}$. While some of these graphs can be degenerate, in the sense of having no nodes, each is laid out in a distinct copy of the $m_i \times n_i$ mesh, whence each layout has area $A_i = m_i n_i$; say, with no loss of generality, that $m_i \leq n_i$. Let us bisect each of these 2^i meshes by cutting it along the *longer* of its dimensions, i.e., the one of length n_i; see Figure 2.4-1. Implicitly, these bisections partition each graph $\mathcal{G}_{ij}$ into two graphs while cutting no more than $m_i \leq \sqrt{A_i}$ edges of the graph; the bound on the number of cut edges comes from the edge-disjointness of the edge-routings in layouts, coupled with the fact that we have cut $m_i \leq \sqrt{m_i n_i}$ edges of the level-i mesh. After this round of mesh-bisections, we are left with layouts of 2^{i+1} graphs, each in a mesh whose sides are powers of 2 and each having area $\frac{1}{2}A_i$. We can continue this partitioning process recursively, always cutting meshes along their longer dimensions, for no more than $\log A$ steps, for after that many steps each mesh has unit area, so each graph has at most one node. If we view the process of recursively partitioning the original graph $\mathcal{G}$ as creating a decomposition tree for $\mathcal{G}$, then we note that

- Each partition at level i of the tree cuts at most $\sqrt{A_i}$ edges of $\mathcal{G}$.
- For all i, $A_{i+1} = \frac{1}{2}A_i$.

Simple calculation verifies that these conditions imply that the graph $\mathcal{G}$ has a $\sqrt{2}$-bifurcator of size $\sqrt{A_0}$. The size, call it F, of the *smallest* $\sqrt{2}$-bifurcator of $\mathcal{G}$ can clearly have no larger size, whence, by squaring.

$$F^2 \leq \mathbf{AREA}(\mathcal{G})$$

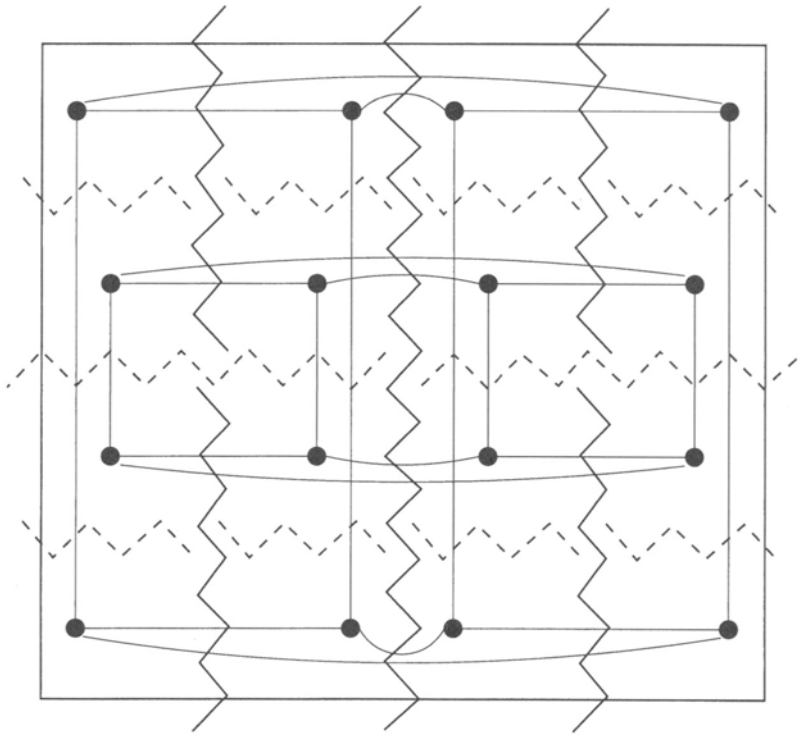


Figure 2.4-1. Recursively partitioning the four-dimensional hypercube by recursively bisecting the mesh.

The proof of the lower bound is now completed by appealing to the proof in Bhatt and Leighton [1984] that F is only a constant factor smaller than the size S of the smallest fully balanced $\sqrt{2}$-bifurcator of $\mathcal{G}$ (which are the bifurcators we now employ to obtain good constructions, i.e., good upper bounds). (See also Theorem 1.4.3.)

2.4.1.2. The Upper Bound

Once again, we concentrate on bounding the area accounted for by the edges of $\mathcal{G}$. Let us be given a decomposition tree for $\mathcal{G}$ that arises from a *fully balanced* $\sqrt{2}$-bifurcator of size S. The tree has two properties that are essential for our layout algorithm.

- The graphs residing at the children of node v of the tree have half as many nodes (to within ± 1) as the graph residing at node v.
- Each graph at level i of the tree has a fully balanced $\sqrt{2}$-bifurcator of size $S/(\sqrt{2})^i$.

The layout procedure works in stages that correspond to the (logarithmically many) levels in the decomposition tree. We construct a layout for $\mathcal{G}$ by proceeding up the tree, starting at the leaves, constructing layouts for the graphs at each level i by combining pairs of layouts of the graphs at level $i + 1$.

Assume, for induction, that at stage i of the procedure, we have laid out each graph $\mathcal{G}_{i,j}$ that resides at level i of the decomposition tree in a mesh of height H_i and width W_i. Readily, this is achievable with $H = W = 1$ for each graph that resides at a leaf of the decomposition tree.

At stage $i - 1$, we take the layouts of all pairs of sibling graphs that reside at level i of the decomposition tree and produce therefrom layouts for the graphs that reside at level $i - 1$ of the decomposition tree. This stage is best understood by focusing on a single pair of sibling graphs, call them $\mathcal{G}_{i,a}$ and $\mathcal{G}_{i,a+1}$, and their layouts in $H_i \times W_i$ meshes $\mathcal{M}_{i,a}$ and $\mathcal{M}_{i,a+1}$, respectively. We now describe a procedure that produces from these layouts a layout of the graph $\mathcal{G}_{i-1,\lfloor a/2 \rfloor}$, which is the parent of $\mathcal{G}_{i,a}$ and $\mathcal{G}_{i,a+1}$ in the decomposition tree, in the $H_{i-1} \times W_{i-1}$ mesh $\mathcal{M}_{i-1,\lfloor a/2 \rfloor}$, where

$$H_{i-1} = W_i + \frac{2S}{(\sqrt{2})^{i-1}} \quad \text{and} \quad W_{i-1} = 2H_i + \frac{S}{(\sqrt{2})^{i-1}}$$

Recall that $\mathcal{G}_{i-1,\lfloor a/2 \rfloor}$ is composed of $\mathcal{G}_{i,a}$ and $\mathcal{G}_{i,a+1}$, connected by some set of $\leqslant S/(\sqrt{2})^{i-1}$ edges. The procedure that creates the new layout involves the following steps.

1. *Rotation.* Rotate each of $\mathcal{M}_{i,a}$ and $\mathcal{M}_{i,a+1}$ so that they become $W_i \times H_i$ meshes; call the rotated meshes, respectively, $\mathcal{M}'_{i,a}$ and $\mathcal{M}'_{i,a+1}$. See Figure 2.4-2.
2. "*Opening up*" *the composite layout.*
   a. Column allocation
      i. Embed $\mathcal{M}'_{i,a}$ in columns $0, 1, \ldots, H_i - 1$ of $\mathcal{M}_{i-1,\lfloor a/2 \rfloor}$.
      ii. Embed $\mathcal{M}'_{i,a+1}$ in columns $H_i + S/(\sqrt{2})^{i-1}$, $H_i + S/(\sqrt{2})^{i-1} + 1, \ldots, 2H_i + S/(\sqrt{2})^{i-1} - 1$ of $\mathcal{M}_{i-1,\lfloor a/2 \rfloor}$.
      iii. Leave the "center" columns $H_i, H_i + 1, \ldots, H_i + S/(\sqrt{2})^{i-1} - 1$ of $\mathcal{M}_{i-1,\lfloor a/2 \rfloor}$ for routing the edges that connect $\mathcal{G}_{i,a}$ with $\mathcal{G}_{i,a+1}$.

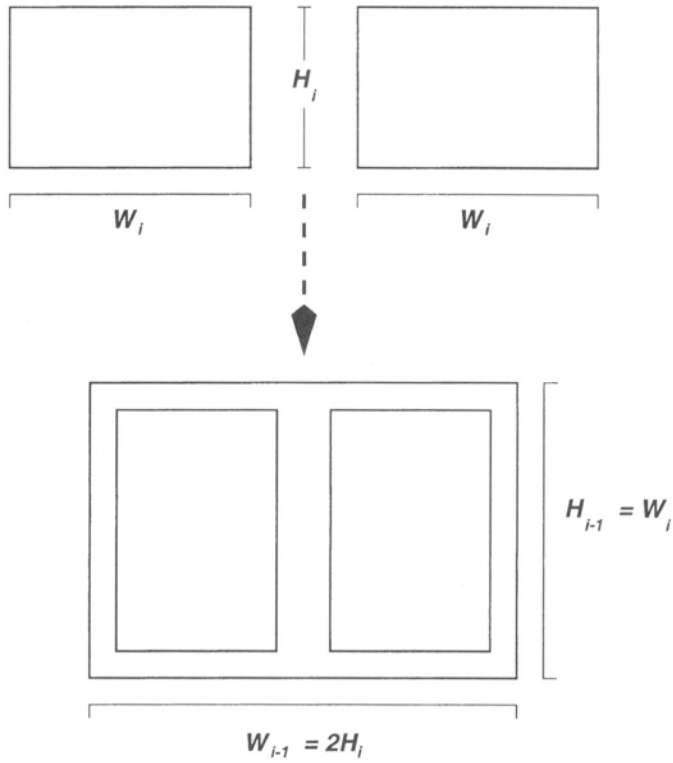


Figure 2.4-2. Step 1 in laying out $\mathcal{G}_{i-1,\lfloor a/2 \rfloor}$: rotating the constituent sublayouts.

b. Row allocation. Embed $\mathcal{M}'_{i,a}$ and $\mathcal{M}'_{i,a+1}$ in $\mathcal{M}_{i-1,\lfloor a/2 \rfloor}$ simultaneously, row by row.
   i. Embed row 0 of each of the small meshes in row 0 of the big mesh in the "natural" way:
      A. Embed row 0 of $\mathcal{M}'_{i,a}$ identically in columns $0, \ldots, H_i - 1$ of row 0 of $\mathcal{M}_{i-1,\lfloor a/2 \rfloor}$.
      B. Embed row 0 of $\mathcal{M}'_{i,a+1}$ identically in columns

$$H_i + S/(\sqrt{2})^{i-1}, \ldots, 2H_i + S/(\sqrt{2})^{i-1} - 1$$

      of row 0 of $\mathcal{M}_{i-1,\lfloor a/2 \rfloor}$.
   ii. Say that row k of each of the small meshes has been embedded in row r_k of the big mesh. Denote by $E_{i,a}$ the *multiset*[8] of

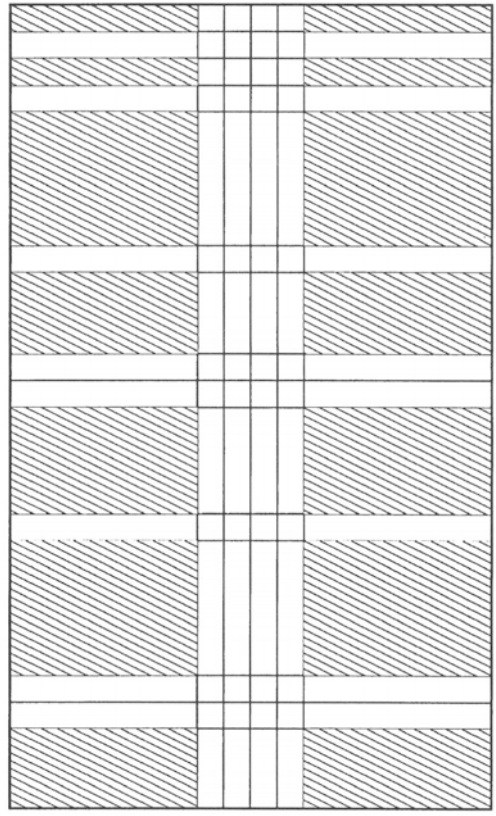


Figure 2.4-3. Step 2 in laying out $\mathcal{G}_{i-1,\lfloor a/2 \rfloor}$ "opening up" the rotated constituted sublayouts. Shaded areas represent "old" portions; clear areas represent new routing channels.

endpoints of the $(\leqslant S/(\sqrt{2})^{i-1})$ edges of $\mathcal{G}$ that connect $\mathcal{G}_{i,a}$ with $\mathcal{G}_{i,a+1}$. Assume that p nodes from $E_{i,a}$ reside in row r_k of $\mathcal{M}_{i-1,\lfloor a/2 \rfloor}$. Then embed row $k+1$ of each of the small meshes in row $r_k + p + 1$ of the big mesh, using the same strategy as with the embedding of row 0 (Step 3a). The p rows of the big mesh that are thereby skipped are used for routing the edges that are incident to these p nodes in $\mathcal{G}_{i,a}$ and $\mathcal{G}_{i,a+1}$.

See Figure 2.4-3.

3. *Edge routing.* We have skipped enough rows and columns in the node placement to dedicate two rows and one column of $\mathcal{M}_{i-1,\lfloor a/2 \rfloor}$ to each edge of $\mathcal{G}$ that connects a node of $\mathcal{G}_{i,a}$ with a node of $\mathcal{G}_{i,a+1}$. Now route each such edge along a zigzag path that connects each

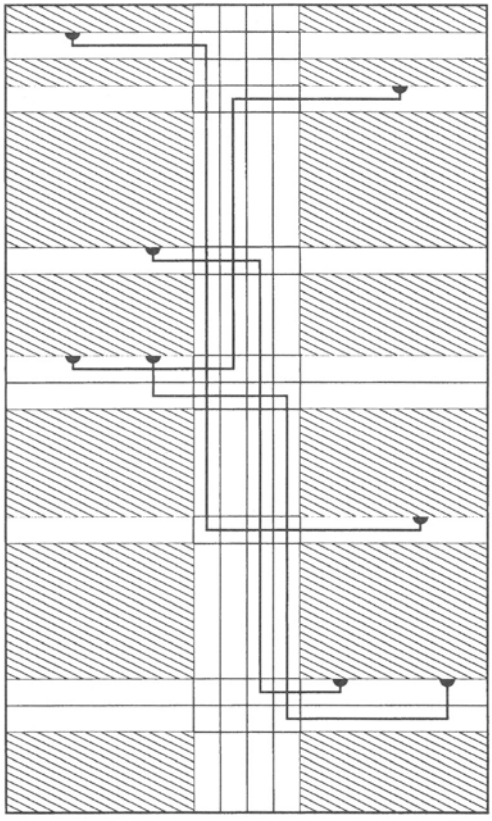


Figure 2.4-4. Step 3 in laying out $\mathcal{G}_{i-1,\lfloor a/2 \rfloor}$; running the new routing paths.

endpoint of the edge with the dedicated row just "below" it (from Step 3b) and proceeds thence along the dedicated center column. Clearly, this path uses no mesh-edges used by any other routing-path. See Figure 2.4-4.

It remains to estimate the area of the layout produced by the foregoing algorithm. The layout of each of the 2^i level-i graphs $\mathcal{G}_{i,a}$ that results from the layout algorithm places each $\mathcal{G}_{i,a}$ within a mesh of height H_i and width W_i, where

$$H_i = W_{i+1} + \frac{2S}{(\sqrt{2})^i} = 2H_{i+2} + \frac{(2 + \sqrt{2}/2)S}{(\sqrt{2})^i}$$

$$W_i = 2H_{i+1} + \frac{S}{(\sqrt{2})^i} = 2W_{i+2} + \frac{(1 + 2\sqrt{2})S}{(\sqrt{2})^i}$$

with initial conditions $H_\ell = W_\ell = 1$ for $\ell = \lceil \log N \rceil$. One now proves easily that these recurrences imply that

$$H_0 \leqslant (2 + \sqrt{2}/2)S \log N$$
$$W_0 \leqslant (1 + 2\sqrt{2})S \log N$$

so that the area occupied by the wires of $\mathscr{G}$ satisfies

$$\text{EDGE-AREA}(\mathscr{G}) = A_0 = H_0 W_0 \leqslant (4 + \tfrac{9}{2}\sqrt{2})S^2 \log^2 N$$

It follows that the total area occupied by $\mathscr{G}$ satisfies

$$\text{AREA}(\mathscr{G}) \leqslant (4 + \tfrac{9}{2}\sqrt{2}) \max(N, S^2 \log^2 N)$$

as was claimed.

Using a similar analysis, one can obtain bounds on the maximum length of any routing-path in the layout as a function of the bifurcator size S. By using a more sophisticated layout technique, one can improve the area bounds of Theorem 2.4.1 by lowering the constant factor and, even more important, by decreasing the argument of the logarithmic factor to N/S; the sophisticated layout engenders similar improvements in the bounds on the lengths of routing-paths. The reader is referred to Bhatt and Leighton [1984] for details.

We close this section by noting that one can often approximate the quality of the layouts produced using bifurcators by using other genres of recursive edge-bisector, providing that the genre used accurately reflects the difficulty of recursively bisecting the graph being laid out. Indeed, the major contribution of the notion of bifurcator is that it is guaranteed to reflect this difficulty.

2.4.2. A Simple Lower-Bound Technique

Although Theorem 2.4.1 affords one a provably good way to obtain lower bounds on the areas of graph layouts, the technique is hard to apply because it requires one to have information about a recursive decomposition of one's graph. It turns out that one can often get good lower bounds just by knowing about a graph's bisection-width. Techniques from Chapter 4 can help one get that information.

THEOREM 2.4.2. *If the graph $\mathcal{G}$ has minimum bisection-width $W_{\mathcal{G}}$, then*

$$\text{Area}(\mathcal{G}) \geq (W_{\mathcal{G}} - 1)^2$$

PROOF. Let us be given an area-minimal layout of the graph $\mathcal{G}$ in the mesh, $\mathcal{M}_{m,n}$. Say, with no loss of generality, that $m \leq n$. Let C_k, where $k \in Z_n$, denote the kth *column* of $\mathcal{M}_{m,n}$; i.e., $C_k \stackrel{\text{def}}{=} Z_m \times \{k\}$.

Let k_0 be the column-index in $\mathcal{M}_{m,n}$ that roughly bisects the layout of $\mathcal{G}$, in the sense that there are roughly equally many images of nodes of $\mathcal{G}$ to the left of, and including, column C_{k_0} as to the right of, and including, column C_{k_0}. (We include column C_{k_0} in both counts in order to defer allocating the image-nodes in the column to the "left" or the "right"; our method of compensating for the double counting will become clear imminently.) Precisely, let $k_0 \in Z_n$ be the smallest (i.e., "leftmost") column-index in $\mathcal{M}_{m,n}$ such that

$$\left|\alpha(\mathbf{N}(\mathcal{G})) \cap \bigcup_{l=0}^{k_0} C_l\right| \geq \left|\alpha(\mathbf{N}(\mathcal{G})) \cap \bigcup_{l=k_0}^{n-1} C_l\right|$$

Easily, k_0 exists and is unique (as one can verify via a discrete analogue of a "continuity argument").

Now—here is the compensation that we promised—let ℓ_0 be the row-index within column C_{k_0} that precisely bisects the layout of $\mathcal{G}$, to within one node-image. Precisely, choose $\ell_0 \in Z_m$ so that the difference

$$\left|\alpha(\mathbf{N}(\mathcal{G})) \cap \left(\{\langle i, k_0\rangle \mid 0 \leq i \leq \ell_0\} \cup \bigcup_{l=0}^{k_0-1} C_l\right)\right| - \left|\alpha(\mathbf{N}(\mathcal{G})) \cap \left(\{\langle i, k_0\rangle \mid \ell_0 < i < m\} \cup \bigcup_{l=k_0+1}^{n-1} C_l\right)\right|$$

is at most 1.

The preceding procedure partitions the layout of $\mathcal{G}$ into two pieces, each of which contains half of the images of $\mathcal{G}$'s nodes, by partitioning $\mathcal{M}_{m,n}$ into the two disjoint subgraphs which are the induced subgraphs of $\mathcal{M}_{m,n}$ on the node-sets

$$\{\langle i, k_0\rangle \mid 0 \leq i \leq \ell_0\} \cup \bigcup_{l=0}^{k_0-1} C_l \quad \text{and} \quad \{\langle i, k_0\rangle \mid \ell_0 < i < m\} \cup \bigcup_{l=k_0+1}^{n-1} C_l$$

(which are two possibly "ragged" meshes). See Figure 2.4-5. The two

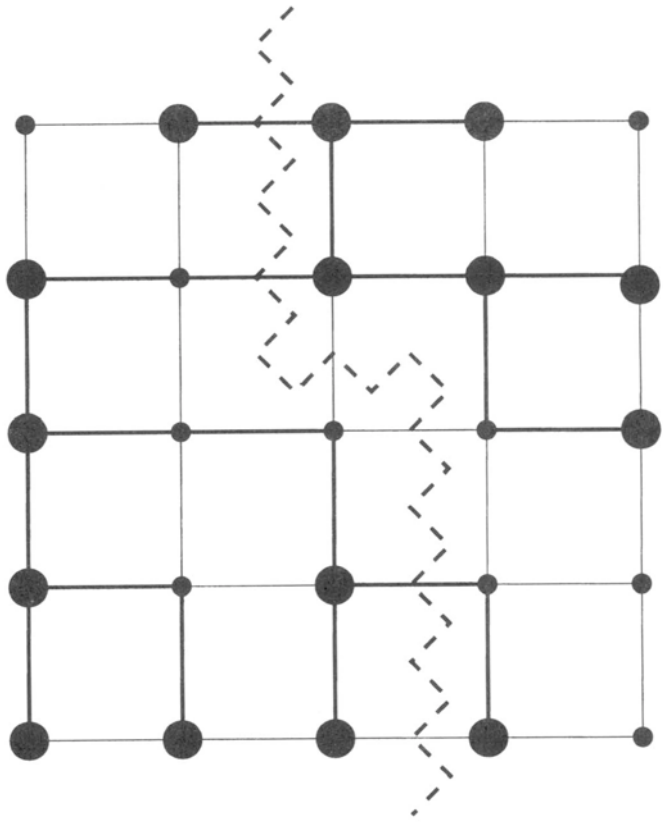


Figure 2.4-5. Bisecting the graph $\mathcal{G}$ (in this case, a complete binary tree) by partitioning the mesh $\mathcal{M}_{m,n}$.

important observations relative to this bisection and partition are the following.

- Because of the edge-disjointness of routing-paths in graph layouts, we can (edge-) bisect the graph $\mathcal{G}$ by cutting no more edges than are needed to partition $\mathcal{M}_{m,n}$ in this way.
- We achieve the partition of $\mathcal{M}_{m,n}$ while cutting no more than $m + 1$ mesh-edges. (If ℓ_0 is either 0 or $m - 1$, then we "save" one edge.)

These two observations combine to show that $m + 1$ can be no smaller than the bisection-width $W_{\mathcal{G}}$ of $\mathcal{G}$. Putting this fact together with the area-

minimality of $\mathcal{M}_{m,n}$, we infer that

$$\text{AREA}(\mathcal{G}) = mn \geq m^2 \geq (W_{\mathcal{G}} - 1)^2$$

which is precisely what we set out to establish. □

2.4.3. A Semantical Lower-Bound Technique

Thus far in this section we have demonstrated that the separation properties of the graph underlying a circuit expose enough of the structure of the circuit to obtain close upper and lower bounds on the minimum area of a VLSI layout of the circuit. The present subsection continues that theme, but with a significant variation. Here, we focus on inferring lower bounds on the complexity of realizing a circuit via a VLSI layout, based on the information-transfer requirements of the function the circuit computes. (Recall, for contrast, that until now we have never asked what the circuit was computing.)

> Historically, the theme pursued in this subsection predated that of the previous subsections (cf. Thompson [1980]), but the framework of VLSI layout is a bit easier to describe in a purely structural setup, whence our ordering of the presentation.

We focus here on two functions that most easily illustrate ideas involved in information-transfer arguments, namely, the computation of permutations of tuples of numbers and the computation of cyclic shifts of such tuples. Arguments building on those we present here lead to lower bounds on the complexity of VLSI layouts of a large variety of other functions (Vuillemin [1983]); arguments that are similar in spirit expand the repertoire of boundable functions even further (see, e.g., Abelson and Andreae [1980], Bilardi [1985], and Siegel [1986]).

Our study focuses on *combinational* (i.e., memoryless) circuits; hence, the layout of the N-variable version of the circuit must contain N sites, called *pins*, where the input values are made available to the circuit, and N pins where the circuit makes its output available. The restriction to functions that have equally many inputs and outputs can be overcome in a variety of ways; cf. Vuillemin [1983]. In order to simplify our setting without jeopardizing our main goal of demonstrating the use of separators in studying circuit efficiency, we assume that the input pins and the output pins are separate entities; this assumption, too, can be avoided; cf. Lipton and Sedgewick [1981] and Savage [1984]. Finally, we assume that the tuple of inputs to the circuit travels just once from the N input pins of the layout to

the N output pins; for obvious reasons, we call this *a one-pass* layout of the circuit. It is not hard to allow the resources in the VLSI layout, including the input and output pins, to be *multiplexed*, allowing each element of an input tuple to pass through the circuit several times in its journey from the appropriate (initial) input pin to the appropriate (final) output pin. Choosing between a one-pass layout of a circuit and a multipass layout usually involves trading computation time for circuit area. The analysis technique that underlies the development in this subsection is easily adapted to allow one to bound the size of the (area) × (time^2) product of multipass layouts for functions. This adaptation is beyond the scope of the current treatment, but it is treated in many of the cited sources (including the original source of such bounds, Thompson [1980]).

The first family of circuits we study here are called *permutation networks.*[9] An N-input permutation network $\mathcal{G}_N$ has N input nodes, N output nodes, and some number of other nodes, often called *switches.* The defining characteristic of such a network is that given any permutation π of Z_N, viewed as a permutation of $\mathcal{G}_N$'s input nodes, there are N edge-disjoint paths in $\mathcal{G}_N$ that simultaneously connect all input nodes to the appropriate output nodes; i.e., each input node i is routed to output node $\pi(i)$. We say that $\mathcal{G}_N$ *computes* the permutation π in this sense.

The second family of circuits we study is called *cyclic shifters.*[10] An N-input cyclic shifter has N input nodes and N output nodes along with some number of other ("switch") nodes that allow it to compute every permutation that is a cyclic shift of Z_N, in the same sense that a permutation network computes arbitrary permutations of Z_N.

This presents enough background for us to turn to our bounds. The main results of this subsection are embodied in the following.

THEOREM 2.4.3. (a) *The smallest one-pass VLSI layout of an N-input permutation network has area*

$$A \geq \left(\frac{N}{2} - 1\right)^2$$

(b) *The smallest one-pass VLSI layout of an N-input cyclic shifter has area*

$$A \geq \left(\frac{N}{4} - 1\right)^2$$

For perspective, one can easily lay out the N-input versions of permutation networks such as the **Beneš** networks (**Beneš** [1964]) in area N^2; this

is not difficult to accomplish directly, but the techniques of Section 2.4.1 can also be enlisted, since the network has $O(N \log N)$ nodes and a recursive bisector of size $S(n) = n$. *A fortiori*, one can easily lay out the N-input versions of cyclic shifters in area N^2.

PROOF. To avoid the distraction of unilluminating floors and ceilings in mathematical expressions, let us focus on permutation networks and cyclic shifters which have even numbers of inputs; clerical modifications suffice to remove this restriction. For both families of graphs, let us assume that we start with a VLSI layout of an arbitrary such graph in the $m \times n$ mesh $\mathcal{M}_{m,n}$, where, with no loss of generality, $m \leqslant n$.

(a) *Permutation networks.* We employ the scan-line argument that appears in the proof of Theorem 2.4.2.

Say that we are given a one-pass VLSI layout of an N-input permutation network $\mathcal{G}_N$. We begin our analysis of the layout by remarking that there is a path of length $\leqslant m + 1$ which bisects $\mathcal{M}_{m,n}$ into two subgraphs, call them $\mathcal{A}$ and $\mathcal{B}$, each of which contains the images of $N/2$ input pins of $\mathcal{G}_N$; cf. the proof of Theorem 2.4.2. The important fact for us is that this bipartition of $\mathcal{M}_{m,n}$ must segregate some set S of $\geqslant N/2$ *output* pins of $\mathcal{G}_N$ from some set T of $N/2$ *input* pins; this is because at least one of $\mathcal{A}$ and $\mathcal{B}$ must contain the images of at least $N/2$ output pins of $\mathcal{G}_N$, while each subgraph contains images of precisely $N/2$ input pins. Now consider any permutation π of the set $\mathcal{Z}_N$ that maps the input pins in T into the set of output pins S. Since $\mathcal{G}_N$ can realize the permutation π, there must be a set of (at least) $N/2$ edge-disjoint paths in $\mathcal{M}_{m,n}$ which connect the images of the pins in T to the images of equally many pins in S. Since these $N/2$ edge-disjoint paths connect $\mathcal{A}$ with $\mathcal{B}$, we conclude that $m + 1 \geqslant N/2$. It follows that the area $A = mn$ of $\mathcal{M}_{m,n}$ must satisfy the inequality in the theorem.

(b) *Cyclic shifters.* If we try to apply the argument in the preceding paragraph to the layout of an N-input cyclic shifter, call it $\mathcal{H}_N$, rather than a permutation network $\mathcal{G}_N$, we encounter an impenetrable barrier in the sentence, "Now consider any permutation π of the set $\mathcal{Z}_N$ that maps the input pins in T into the set of output pins S." If the permutation π must be a cyclic shift of the set $\mathcal{Z}_N$ then there is no reason to believe that such a π exists. We get around this barrier by resorting to the following subtler argument.

For every input pin $i \in T$ and every output pin $j \in S$, there is a cyclic shift $\pi_{i,j}$ that maps i to j. When the given shifter network $\mathcal{H}_N$ is used to realize cyclic shift $\pi_{i,j}$, it must supply a path from input pin i to output pin j that shares an edge with no other input-to-output path used to realize $\pi_{i,j}$. If we add up the number of such paths over all possible values of i and j, we see that there are $|S||T| \geqslant \frac{1}{4}N^2$ input-to-output paths that the circuit must

supply "over its lifetime." Since there are only N cyclic shifts in all, some one shift, call if $\mathscr{P}_0$, must account for at least $\frac{1}{4}N$ edge-disjoint paths crossing our scan line to connect inputs in T to outputs in S. Now we employ the reasoning in the proof of part (a) to conclude that the smaller dimension, m, of $\mathscr{M}_{m,n}$ can be no smaller than $\frac{1}{4}N - 1$, thus yielding the claimed bound. □

2.5. Strongly Universal Interval Hypergraphs

The application we study in this section combines the themes of several genres of investigations that have appeared in the literature in recent years.[11] The first genre is motivated by the usefulness of *multipoint* nets in present-day microelectronics, i.e., wires that interconnect several devices (e.g., transistors) in a circuit rather than just two. These studies attempt to extend the VLSI layout theory outlined in Section 2.4 so that the guest graphs can be *hypergraphs*, i.e., graphs in which each edge can connect many nodes (Bhatt and Leiserson [1984]). The second genre of investigation is motivated by the potential of "bus-oriented" parallel computer architectures that are enabled by VLSI technology; these studies attempt to expand the study of graph embeddings to allow the hosts to be hypergraphs (Peterson and Ting [1982], Stout [1986]). The third genre of investigation is motivated by a particular approach to the issue of fault tolerance in interconnection networks; these investigations seek, for a given finite family of graphs $\mathbf{G}$, a graph $\mathscr{G}(\mathbf{G})$ that is *strongly universal* for $\mathbf{G}$ in the sense of containing each graph in $\mathbf{G}$ as a subgraph, even if some positive fraction of the nodes of $\mathscr{G}(\mathbf{G})$ are killed," i.e., rendered unavailable (Alon and Chung [1988], Beck [1983, 1990], Bruck *et al.* [1993], Friedman and Pippenger [1987]).

The formal vehicle for this section, *interval hypergraphs* (*I-hypergraphs*, for short), was introduced in Rosenberg [1989] as a formal analog of multipoint or bus-oriented systems, to complement the use of graphs as a formal analog of point-to-point systems. I-hypergraphs are used in Rosenberg [1989] to study a bus-oriented approach to the design of fault-tolerant arrays of identical processors in an environment of VLSI circuitry. In the study one achieves tolerance to faults in the nodes of a given finite family of graphs $\mathbf{G}$ by designing a (small) I-hypergraph that is strongly universal for $\mathbf{G}$, in the sense just described; the study is, therefore, a hypergraph-based analog of graph-based studies such as Alon and Chung [1988], Beck [1983, 1990], Bruck *et al.* [1993], and Friedman and Pippenger [1987]. The result from Rosenberg [1989] that is relevant to this chapter is an algorithm that produces such small strongly universal I-hypergraphs from knowledge of the

separation characteristics of the graphs in family **G**. After presenting the construction of small strongly universal I-hypergraphs from Rosenberg [1989] in Section 2.5.2, we extract from Chung and Rosenberg [1986] a strategy for proving, in Section 2.5.3, that the construction's I-hypergraphs are almost optimal in size. In Appendix A we combine this strategy with the lower-bound results on separation-widths from Chapter 4 to prove the near optimality of the construction for a variety of important graph families.

The design algorithm from Rosenberg [1989] takes as input a finite family of graphs **G** and the knowledge that each graph $\mathcal{G} \in \mathbf{G}$ has a β-separator of size $S(n)$, for some given rational $0 < \beta \leqslant 1/2$ and some given integer function $S(n)$. The algorithm produces an I-hypergraph $\mathcal{I}(\mathbf{G})$ that is strongly universal for **G**, of SIZE (measured by the sum of the cardinalities of its hyperedges)

$$\mathrm{SIZE}(\mathcal{I}(\mathbf{G})) = N \cdot \left(\sum_{k=1}^{\log N} \sum_{i=0}^{\ell(2^k)} S((1-\beta)^i 2^k) \right) \tag{2.5.1}$$

where N is the number of nodes in the largest graph in **G**, and

$$\ell(n) \stackrel{\text{def}}{=} \log_{1/(1-\beta)} n$$

For many families **G**, including binary trees and any family for which $S(n) = \Theta(n^\varepsilon)$ for some rational $0 < \varepsilon \leqslant 1$, the I-hypergraphs $\mathcal{I}(\mathbf{G})$ are optimal in SIZE to within a constant factor. Moreover, when $S(n) = \Theta(n^\varepsilon)$, the SIZE of $\mathcal{I}(\mathbf{G})$, which can be viewed as measuring the area required to lay $\mathcal{I}(\mathbf{G})$ out in the plane, in the sense of Section 2.4, is just a small constant factor greater than the area of any collinear[12] layout in the plane of the largest graph in **G**.

2.5.1. The Formal Framework

Before we consider the design algorithm, we must make the notions we have been discussing formal and precise.

2.5.1.1. Hypergraphs and Embedding

A *hypergraph* $\mathcal{H}$ comprises a set $\mathbf{N}(\mathcal{H})$ of *nodes* and a *multiset* $\mathbf{H}(\mathcal{H})$ of subsets of V, called *hyperedges*. An N-node *interval hypergraph* (*I-hypergraph*, for short) $\mathcal{I}$ is a hypergraph whose nodes comprise the set $\mathcal{Z}_N$ and whose hyperedges all have the form $\{k, k+1, \ldots, k+r\}$ for some $k \geqslant 0$ and $1 \leqslant r < N - k$. As with graphs, we denote by $N(\mathcal{I})$ the number of

nodes of the hypergraph $\mathcal{I}$; we denote by $\mathbf{SIZE}(\mathcal{I})$ the sum of the cardinalities of $\mathcal{I}$'s hyperedges.

An *embedding* of the graph $\mathcal{G}$ into the I-hypergraph $\mathcal{I}$ is a pair $\langle \alpha, \rho_h \rangle$ of one-to-one mappings:

- $\alpha : \mathbf{N}(\mathcal{G}) \rightarrow \mathbf{N}(\mathcal{I})$
- $\rho_h : \mathbf{E}(\mathcal{G}) \rightarrow \mathbf{H}(\mathcal{I})$

such that, for each edge $(u, v) \in \mathbf{E}(\mathcal{G})$, $\{\alpha(u), \alpha(v)\} \subseteq \rho_h(u, v)$; i.e., the image nodes $\alpha(u)$ and $\alpha(v)$ are both elements of the image hyperedge $\rho_h(u, v)$. We say that an I-hypergraph *contains* any graph that is embeddable in it.

2.5.1.2. Strong Universality and Strong Separation

Let $\mathbf{G}$ be a finite family of graphs. The I-hypergraph $\mathcal{I}$ is *strongly universal* for $\mathbf{G}$ if the following is true for any set $W \subseteq \mathbf{N}(\mathcal{I})$: For every graph $\mathcal{G} \in \mathbf{G}$ for which $N(\mathcal{G}) \leqslant |W|$, there is an embedding of $\mathcal{G}$ into $\mathcal{I}$ such that $\alpha(\mathbf{N}(\mathcal{G})) \subseteq W$.

Let $\mathcal{G}$ be a graph, let β be a rational in the range $0 < \beta \leqslant 1/2$, and let l be any integer $\geqslant \ell(N(\mathcal{G}))$. The graph $\mathcal{G}$ has a *β-separation profile* (β-SP) $\langle s_l, s_{l-1}, \ldots, s_1 \rangle$, where each s_i is a nonnegative integer, precisely if: by removing at most s_l edges from $\mathcal{G}$, one can partition the graph into subgraphs $\mathcal{G}_0$ and $\mathcal{G}_1$, each of size $\leqslant (1 - \beta)N(\mathcal{G})$, and each having a β-SP $\langle s_{l-1}, s_{l-2}, \ldots, s_1 \rangle$. Another view of separation profiles is given by the notion of a $\langle s_l, s_{l-1}, \ldots, s_1 \rangle$-decomposition tree for $\mathcal{G}$: If one has a graph $\mathcal{G}$ with a β-SP $\langle s_l, s_{l-1}, \ldots, s_1 \rangle$, then one can construct a depth-l binary tree whose root is $\mathcal{G}$ and whose left and right subtrees are, respectively, the $\langle s_{l-1}, s_{l-2}, \ldots, s_1 \rangle$-decomposition trees of the graphs $\mathcal{G}_0$ and $\mathcal{G}_1$ already mentioned.

The notions "separator" and "separation profile" converge in the fact that every graph $\mathcal{G}$ having a β-separator of size $S(n)$ admits a β-SP $\langle s_l, s_{l-1}, \ldots, s_1 \rangle$, where each $s_i = S((1 - \beta)^{l-i} N(\mathcal{G}))$. We leave to the reader the exercise of translating this correspondence into a decomposition tree for $\mathcal{G}$ and verifying that it yields the same $(\beta, S(n))$-decomposition tree that we used in Section 2.3.

2.5.2. The Construction

We turn now to the main result of the section, the construction algorithm for strongly universal I-hypergraphs. Say that we are given the finite family of graphs $\mathbf{G}$, where the largest graph in $\mathbf{G}$ has N^* nodes. For

convenience, say that N^* is a power of 2 and that $r = \log N^*$. Let **G** have a β-separator of size $S(n)$ for some $0 < \beta \leqslant 1/2$.

THEOREM 2.5.1. *The family of graphs* **G**, *as previously described, admits a strongly universal I-hypergraph* $\mathscr{I}(\mathbf{G})$ *of SIZE*[13]

$$SIZE(\mathscr{I}(\mathbf{G})) = N^* \cdot \left(\sum_{k=1}^{r} \sum_{i=0}^{-k/\log(1-\beta)} S((1-\beta)^i 2^k) \right) \tag{2.5.2}$$

We prove Theorem 2.5.1 by describing the I-hypergraph $\mathscr{I}(\mathbf{G})$ and verifying that it is indeed strongly universal for the family **G**.

2.5.2.1. Constructing $\mathscr{I}(\mathbf{G})$

Let the nodes of $\mathscr{I}(\mathbf{G})$ be the set Z_{N^*}. We give $\mathscr{I}(\mathbf{G})$ the following hyperedges: for all positive $k \in Z_{r+1}$ and all $a \in Z_{2^{r-k}-1}$, we create

$$\sum_{i=0}^{-k/\log(1-\beta)} S((1-\beta)^i 2^k)$$

copies of the hyperedge

$$\{a2^k, a2^k + 1, \ldots, (a+1)2^k - 1\}$$

It is clear that the I-hypergraph $\mathscr{I}(\mathbf{G})$ so constructed has

$$\mathrm{SIZE}(\mathscr{I}(\mathbf{G})) = N^* \cdot \left(\sum_{k=1}^{r} \sum_{i=0}^{-k/\log(1-\beta)} S((1-\beta)^i 2^k) \right)$$

as claimed in the theorem. We need, therefore, only verify that $\mathscr{I}(\mathbf{G})$ is strongly universal for the family G. While we delay this verification until the next subsection, we indicate informally how the graphs in **G** are embedded into $\mathscr{I}(\mathbf{G})$, allocating the nodes of the graph to arbitrary node-subsets of the I-hypergraphs.

Say that we are told that some specific p nodes of $\mathscr{I}(\mathbf{G})$ are the only ones available for embeddings and that we are to embed the $(\leqslant p)$-node graph $\mathscr{G} \in \mathbf{G}$ into $\mathscr{I}(\mathbf{G})$ (perforce, using only these nodes). We begin the embedding process by constructing a $(\beta, S(n))$-decomposition tree for $\mathscr{G}$. We then lay out the nodes of $\mathscr{G}$ on the available nodes of $\mathscr{I}(\mathbf{G})$, in the order in which the nodes occur as leaves of the decomposition tree. (If $\mathscr{G}$ has fewer than p nodes, then we arbitrarily choose $N(\mathscr{G})$ of the available nodes of

$\mathcal{I}(\mathbf{G})$ as homes for $\mathcal{G}$'s nodes.) Thus we have the node-injection α. In order to specify the edge-injection ρ_h, we associate with each edge (u, v) of $\mathcal{G}$ any as-yet unused *smallest* hyperedge of $\mathcal{I}(\mathbf{G})$ that contains both $\alpha(u)$ and $\alpha(v)$.

2.5.2.2. Validating the Construction

We now validate the construction and embedding process of the previous subsection. Our validation uses a nonstandard graph-theoretic notion motivated by the stringent demands of strong universality.

Our I-hypergraph $\mathcal{I}(\mathbf{G})$ decomposes naturally by bisection. Removing the largest hyperedges decomposes $\mathcal{I}(\mathbf{G})$ into two copies of the I-hypergraph that we would construct if all graphs of size exceeding $\frac{1}{2}N_{\mathbf{G}}$ were removed from $\mathbf{G}$, and so on for the sets of hyperedges of progressively smaller sizes. When a graph $\mathcal{G}$ is embedded into $\mathcal{I}(\mathbf{G})$, it is not clear how this bisection will dissect $\mathcal{G}$, for that depends on which nodes of $\mathcal{I}(\mathbf{G})$ are declared available for the embedding. Our guarantee that $\mathcal{G}$ can be embedded no matter which nodes of $\mathcal{I}(\mathbf{G})$ are available thus leads naturally to the following unusually demanding notion of graph decomposition.

Focus on any power of 2, $N^* = 2^r$, let $\mathcal{G}$ be a graph having N^* or fewer nodes, and let l be any integer $\geqslant \log N^*$. The l-tuple of nonnegative integers $\langle e_l, e_{l-1}, \ldots, e_1 \rangle$ is a *strong separation profile* (SSP) for $\mathcal{G}$ if the following property holds.

> **THE SSP Property.** Given any integer N_0 such that both N_0 and $N(\mathcal{G}) - N_0$ are $\leqslant \frac{1}{2}N^*$: By removing at most e_l edges from $\mathcal{G}$, one can partition $\mathcal{G}$ into subgraphs $\mathcal{G}_0$ having N_0 nodes and $\mathcal{G}_1$ having $N(\mathcal{G}) - N_0$ nodes, each of which has $\langle e_{l-1}, e_{l-2}, \ldots, e_1 \rangle$ as an SSP. This recursive decomposition of $\mathcal{G}$ continues until we get down to single-node subgraphs of $\mathcal{G}$.

Note that one can view each candidate decomposition of $\mathcal{G}$ (corresponding to the different choices for N_0) in terms of an $\langle e_l, e_{l-1}, \ldots, e_1 \rangle$-decomposition tree for $\mathcal{G}$: the root of the tree is $\mathcal{G}$; the children of the root are $\mathcal{G}_0$ and $\mathcal{G}_1$ and so on, just as with $(\beta, \mathrm{S(n)})$-decomposition trees.

The qualifier "strong" in the term "strong separation profile" is intended to contrast SSPs with the notion of β-SP, wherein one seeks a "small cut" partition just for the case $\lceil \beta N(\mathcal{G}) \rceil \leqslant N_0 \leqslant \lfloor (1 - \beta) N(\mathcal{G}) \rfloor$ rather than for all values of $N_0 \in \{1, 2, \ldots, \frac{1}{2}N^*\}$. The relevance of the notion of SSP resides in the following result.

LEMMA 2.5.2. *Given any l-tuple of nonnegative integers*

$$\tau = \langle e_l, e_{l-1}, \ldots, e_1 \rangle$$

one can construct an $(N = 2^l)$*-node I-hypergraph* $\mathcal{I}_N$ *of SIZE*

$$\text{SIZE}(\mathcal{I}_N) = N \cdot \sum_{i=1}^{l} e_i$$

which is strongly universal for the family $\mathbf{G}(\tau)$ *comprising all graphs that have the tuple* τ *as an SSP.*

PROOF. We indicate how to construct $\mathcal{I}_N$ and then how to embed the graphs in $\mathbf{G}(\tau)$ into it.

The I-hypergraph $\mathcal{I}_N$. To construct $\mathcal{I}_N$, we create the following hyperedges from the node-set Z_N. For all positive $k \in Z_{l+1}$ and all $a \in Z_{2^{l-k}-1}$, we create e_k copies of the hyperedge $\{a2^k, a2^k + 1, \ldots, (a+1)2^k - 1\}$. It is clear that $\mathcal{I}_N$, so constructed, has the claimed SIZE.

The embedding procedure. Say that we are told that some specific set $\{v_1, v_2, \ldots, v_p\}$ of nodes of $\mathcal{I}_N$ is available for embeddings and that we are to embed the $(\leqslant p)$-node graph $\mathcal{G} \in \mathbf{G}(\tau)$ into $\mathcal{I}_N$ (perforce, using these nodes). The essence of the embedding process is the construction of an $\langle e_l, e_{l-1}, \ldots, e_1 \rangle$-decomposition tree for $\mathcal{G}$. We begin by choosing, in any way whatsoever, some $N(\mathcal{G})$ of the available nodes of $\mathcal{I}_N$ as homes for the nodes of $\mathcal{G}$. This choice then determines the parameter N_0, which is the size of one of the two graphs into which we partition $\mathcal{G}$. Specifically,

$$N_0 \stackrel{\text{def}}{=} |\{v_j : 0 \leqslant v_j \leqslant 2^{l-1} - 1\}|$$

that is, N_0 is the number of selected available nodes that reside "to the left" of the midpoint (i.e., node $\frac{1}{2}N$) of $\mathcal{I}_N$. By definition of SSP, $\mathcal{G}$ can be partitioned into a subgraph $\mathcal{G}_0$ of N_0 nodes and a subgraph $\mathcal{G}_1$ of $N(\mathcal{G}) - N_0$ nodes by removing no more than e_l edges from $\mathcal{G}$. These edges can thus be embedded in the e_l size-N hyperedges of $\mathcal{I}_N$, no matter to which nodes of $\mathcal{I}_N$ the edges' endpoints are assigned. By definition of SSP, we may assume that each of the two resulting subgraphs, $\mathcal{G}_0$ and $\mathcal{G}_1$, has an SSP $\langle e_{l-1}, e_{l-2}, \ldots, e_1 \rangle$. We thus find ourselves with two half-size versions of our original problem: By removing the e_l large hyperedges from $\mathcal{I}_N$, we are left with two copies of $\mathcal{I}_{N/2}$ in which to embed the two subgraphs of $\mathcal{G}$, each by definition having no more than 2^{l-1} nodes. We leave to the reader the easy details of inductively validating this recursive embedding process (which can be viewed as building an $\langle e_l, e_{l-1}, \ldots, e_1 \rangle$-decomposition tree for $\mathcal{G}$). □

Determining SSPs for arbitrary graphs is not a trivial pursuit. However, one can, with little difficulty, discover profiles for certain familiar graphs. For instance, every $(\leqslant N)$-node binary tree has an SSP of the form[14]

$\langle \log N, \log(N/2), \ldots, 1 \rangle$, so $e_{k-1} = e_k - 1$; similarly, every $(\leqslant N)$-node rectangular mesh has an SSP of the form $\langle \sqrt{N}, \sqrt{N}/\sqrt{2}, \sqrt{N}/2, \ldots, 1 \rangle$, so $e_{k-1} = e_k/\sqrt{2}$. The following lemma helps one discover SSPs, and it combines with Lemma 2.5.2 to complete the proof of Theorem 2.5.1.

LEMMA 2.5.3. *Let* $\mathbf{G}$ *be a finite family of graphs having a* β*-separator of size* $S(n)$*. For every integer* r*, every graph* $\mathcal{G} \in \mathbf{G}$ *with* $N(\mathcal{G}) \leqslant 2^r$ *has an SSP* $\langle e_r, e_{r-1}, \ldots, e_1 \rangle$*, where each*

$$e_k = \sum_{i=0}^{-k/\log(1-\beta)} S((1-\beta)^i 2^k)$$

PROOF. The proof builds on a device that appears in Rosenberg [1981b] for embedding any given graph $\mathcal{G} \in \mathbf{G}$ into a path. Note that this embedding problem is purely a technical device and should not be construed as an embedding of $\mathcal{G}$ into an I-hypergraph, despite the formal similarity between the two procedures. Note also the similarity of this proof with that of Theorem 1.4.5.

The embedding can be described most easily using the terminology of collinear VLSI layouts. Construct a $(\beta, S(n))$-decomposition tree for $\mathcal{G}$, and place the nodes of $\mathcal{G}$ in a row in the order they occur as leaves of the decomposition tree. Run $S(N(\mathcal{G}))$ unit-width horizontal routing tracks above the nodes,[15] in which to route the edges that interconnect the two subgraphs $\mathcal{G}_0$ and $\mathcal{G}_1$ of $\mathcal{G}$ at level 1 of the decomposition tree. These routing tracks can be viewed as rows in the plane that are reserved for "drawing" edges of $\mathcal{G}$; thus every edge of $\mathcal{G}$ ends up being drawn as two vertical line segments from its terminal nodes to the associated routing track, joined by a horizontal line segment within the routing track. Next, run $S((1-\beta)N(\mathcal{G}))$ unit-width horizontal routing tracks over the nodes of $\mathcal{G}_0$ and the same number of routing tracks over the nodes of $\mathcal{G}_1$. Continue in the indicated fashion to run unit-width horizontal routing tracks for routing the edges among the subgraphs of $\mathcal{G}$ in the decomposition tree, using $S((1-\beta)^k N(\mathcal{G}))$ routing tracks for the 2^{k-1} pairs of subgraphs at level-k of the tree. The reader will note that we have constructed a layout of $\mathcal{G}$ that uniformly has

$$W = \sum_{i=0}^{\ell(N(\mathcal{G}))} S((1-\beta)^i N(\mathcal{G}))$$

routing tracks above every node. It follows that, given any integer $N \leqslant N(\mathcal{G})$, one can partition $\mathcal{G}$ into a subgraph of size N and one of size $N(\mathcal{G}) - N$ by removing (or "cutting") at most W edges. In particular, such a

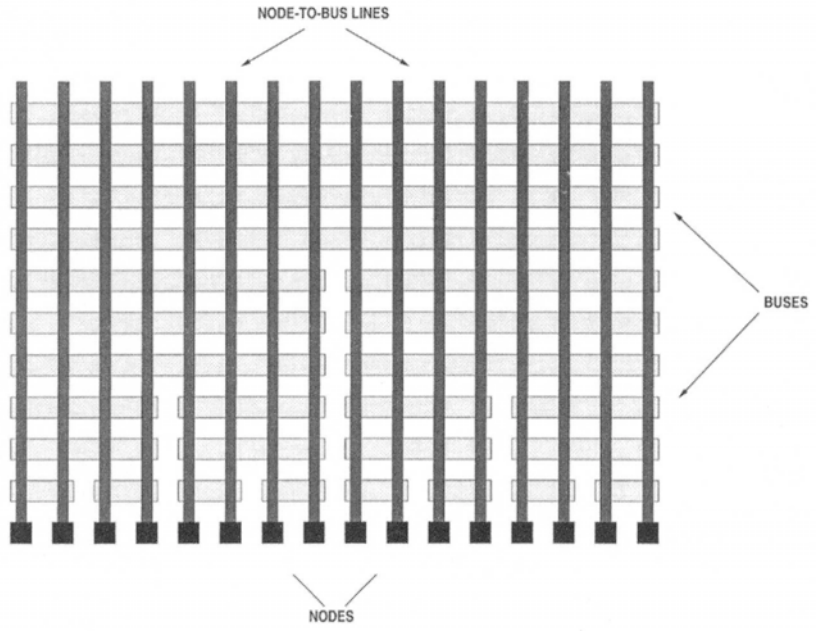


Figure 2.5-1. An interval hypergraph that is strongly universal for binary trees containing 15 or fewer nodes.

partition is possible for any N such that both N and $N(\mathcal{G}) - N$ are $\leqslant 2^{r-1}$. □

Lemmas 2.5.2 and 2.5.3 combine to establish Theorem 2.5.1.

We close this section with Figure 2.5-1, which depicts an I-hypergraph that is strongly universal for the family of binary trees having no more than 15 nodes. The construction of this I-hypergraph appears in Rosenberg [1985]; its SIZE-optimality is proved in Chung and Rosenberg [1986] (using techniques that we present in Chapter 4).

2.5.3. Gauging the Quality of the Construction

Recall that, for any graph $\mathcal{G}$, the k-mincing-width of $\mathcal{G}$, denoted $M_{\mathcal{G}}(k)$, is the smallest number of edges of $\mathcal{G}$ that must be removed in order to mince $\mathcal{G}$ into a k-sum subgraph; cf. Section 1.4. We can bound from below the SIZE of any I-hypergraph that is strongly universal for a graph family $\mathbf{G}$ in terms of the k-mincing-width of any graph $\mathcal{G} \in \mathbf{G}$ such that $N(\mathcal{G})$ is smaller than the number of nodes of the largest graph in $\mathbf{G}$.

THEOREM 2.5.4. *Let* $\mathbf{G}$ *be a finite family of graphs whose largest graph is* $\mathcal{G}_{\mathbf{G}}$, *and let* $\mathcal{G}$ *be any graph in* $\mathbf{G} - \{\mathcal{G}_{\mathbf{G}}\}$. *Say that there is a sequence of integers,* $c_1 < c_2 < \cdots < c_l$, *such that*

$$M_{\mathcal{G}}(c_1) < M_{\mathcal{G}}(c_2) < \cdots < M_{\mathcal{G}}(c_l)$$

Then any I-hypergraph $\mathcal{I}$ *that is strongly universal for* $\mathbf{G}$ *must have* SIZE *no smaller than*

$$\text{SIZE}(\mathcal{I}) \geq (N(\mathcal{G}_{\mathbf{G}}) - N(\mathcal{G})) \left[\frac{1}{c_l - 1} M_{\mathcal{G}}(c_l) + \sum_{i=1}^{l-1} \left(\frac{1}{c_i - 1} - \frac{1}{c_{i+1} - 1} \right) M_{\mathcal{G}}(c_i) \right]$$

PROOF. Let us be given an arbitrary I-hypergraph $\mathcal{I}$ that is strongly universal for the family $\mathbf{G}$, and let us focus on an arbitrary graph $\mathcal{G} \in \mathbf{G} - \{\mathcal{G}_{\mathbf{G}}\}$ as described in the theorem. We perform a succession of l *gedanken* experiments in which we "kill" different $(N(\mathcal{G}_{\mathbf{G}}) - N(\mathcal{G}))$-node subsets of $\mathcal{I}$'s nodes and insist that the graph $\mathcal{G}$ be embedded into the surviving nodes. By judiciously choosing the nodes to kill in each experiment, we show that the cumulative length of $\mathcal{I}$'s hyperedges must satisfy the bound of the theorem.

Our experiments will be parameterized by the theorem's sequence of positive integers $c_1 < c_2 < \cdots < c_l$. Specifically, in the kth experiment, we select as the *surviving* nodes of $\mathcal{I}$ the c_k (roughly) equal-size blocks of nodes with cumulative population $N(\mathcal{G})$, which are spaced (roughly) equally along the row of $\mathcal{I}$'s nodes. (Rounding, where necessary, can be done in any way without affecting the bound.) For instance, if one of the $c_i = 3$, then for that experiment we would select as the surviving nodes the "leftmost" $\lfloor \frac{1}{3} N(\mathcal{G}) \rfloor$ nodes of $\mathcal{I}$, i.e., nodes

$$0, 1, \ldots, \lfloor \tfrac{1}{3} N(\mathcal{G}) \rfloor - 1$$

the "middle" $\lfloor \frac{1}{3} N(\mathcal{G}) \rfloor$ nodes of $\mathcal{I}$, i.e., nodes

$$\lfloor \tfrac{1}{3} N(\mathcal{G}) \rfloor + \lfloor \tfrac{1}{2} (N(\mathcal{G}_{\mathbf{G}}) - N(\mathcal{G})) \rfloor, \ldots, 2 \lfloor \tfrac{1}{3} N(\mathcal{G}) \rfloor + \lfloor \tfrac{1}{2} (N(\mathcal{G}_{\mathbf{G}}) - N(\mathcal{G})) \rfloor - 1$$

and the "rightmost" $\lceil \frac{1}{3} N(\mathcal{G}) \rceil$ nodes of $\mathcal{I}$, i.e., nodes

$$N(\mathcal{G}_{\mathbf{G}}) - \lceil \tfrac{1}{3} N(\mathcal{G}) \rceil, \ldots, N(\mathcal{G}_{\mathbf{G}}) - 1$$

thereby "killing" the remaining $N(\mathcal{G}_{\mathbf{G}}) - N(\mathcal{G})$ nodes of $\mathcal{I}$. The goal of these experiments is to show that there must be many hyperedges "passing

between" adjacent blocks of surviving nodes. Since the blocks are spaced rather far apart in the linear arrangement of $\mathcal{I}$'s nodes, these "interblock" hyperedges build up substantially to a positive fraction of the SIZE of $\mathcal{I}$.

Now, let us assess the cumulative size of the interblock hyperedges from our experiments. Let us concentrate first on a single experiment, with integer parameter c. How might we show that for this experiment there must be many hyperedges passing between adjacent blocks? We exploit the following reduction of the problem. Any solution to the problem of embedding $\mathcal{G}$ into $\mathcal{I}$ using just the selected (surviving) nodes of $\mathcal{I}$ can be viewed as a way of mincing $\mathcal{G}$ into c "equal-size" pieces: each piece resides (under the embedding of $\mathcal{G}$ into $\mathcal{I}$) in one of the blocks of surviving nodes. By definition of mincing-width, no fewer than $M_{\mathcal{G}}(c)$ edges of $\mathcal{G}$ must be cut in order to effect this mincing. Moreover, when one embeds $\mathcal{G}$ into the surviving nodes of $\mathcal{I}$, each of these cut edges connects nodes in distinct blocks of selected nodes; hence, each must be mapped onto a hyperedge of $\mathcal{I}$ whose size is sufficient to span the gap between adjacent blocks of surviving nodes. Since there are $c - 1$ interblock gaps, we have the following.

FACT 2.5.5. *Each of the $M_{\mathcal{G}}(c)$ cut edges of $\mathcal{G}$ requires a distinct hyperedge of $\mathcal{I}$ of size at least*

$$\frac{N(\mathcal{G}_G) - N(\mathcal{G})}{c - 1}$$

in the embedding of $\mathcal{G}$ into $\mathcal{I}$.

The analysis of the previous paragraph focuses on one individual, isolated experiment. We must now take into account the fact that we are performing a sequence of experiments, dealing with a sequence of values of c, not just a single one. This fact manifests itself in our assessment of the total hyperedge-size requirements of I-hypergraph $\mathcal{I}$. We cannot merely add up the wire lengths computed in Fact 2.5.5, since a clever construction of $\mathcal{I}$ would reuse hyperedges that were introduced for one experiment to minimize the number of new hyperedges that are needed for the next experiment. Since the *numbers* of new hyperedges in successive experiments, namely, the sequence of integers $M_{\mathcal{G}}(c_1), M_{\mathcal{G}}(c_2), \ldots, M_{\mathcal{G}}(c_l)$, *increases* with subsequent experiments (by hypothesis), while the *sizes* of interblock gaps, namely, the sequence of integers

$$\frac{N(\mathcal{G}_G) - N(\mathcal{G})}{c_1 - 1}, \frac{N(\mathcal{G}_G) - N(\mathcal{G})}{c_2 - 1}, \ldots, \frac{N(\mathcal{G}_G) - N(\mathcal{G})}{c_l - 1}$$

decreases with subsequent experiments (by simple arithmetic), a smart construction would attempt to reuse the relatively large hyperedges that are needed for the early experiments to satisfy part of the hyperedge demand of the later experiments. Let us see how this works out.

For the first experiment, with parameter c_1, we have no leeway: we must give I-hypergraph $\mathcal{I}$ $M_{\mathcal{G}}(c_1)$ hyperedges each of size

$$\frac{(N(\mathcal{G}_G) - N(\mathcal{G}))}{(c_1 - 1)}$$

For the second experiment, with parameter c_2, we already begin to see the interaction. Instead of giving I-hypergraph $\mathcal{I}$ $M_{\mathcal{G}}(c_2)$ hyperedges each of size $(N(\mathcal{G}_G) - N(\mathcal{G}))/(c_2 - 1)$, we instead give it only $M_{\mathcal{G}}(c_2) - M_{\mathcal{G}}(c_1)$ such new hyperedges. To this point, therefore, we have contributed only

$$(N(\mathcal{G}_G) - N(\mathcal{G}))\left[\left(\frac{1}{c_1 - 1} - \frac{1}{c_2 - 1}\right) M_{\mathcal{G}}(c_1) + \frac{1}{c_2 - 1} M_{\mathcal{G}}(c_2)\right]$$

units to SIZE($\mathcal{I}$) rather than the naive bound of

$$(N(\mathcal{G}_G) - N(\mathcal{G}))\left[\frac{1}{c_1 - 1} M_{\mathcal{G}}(c_1) + \frac{1}{c_2 - 1} M_{\mathcal{G}}(c_2)\right]$$

Continuing in this way, we add, at each experiment, only as few new hyperedges as possible. It is not hard to verify that this strategy

1. Minimizes the cumulative hyperedge-size attributable to the sequence of experiments
2. Adds precisely $M_{\mathcal{G}}(c_k) - M_{\mathcal{G}}(c_{k-1})$ new hyperedges of sizes

$$\frac{(N(\mathcal{G}_G) - N(\mathcal{G}))}{(c_k - 1)}$$

each at the kth experiment.

The theorem now follows by summing the sizes of the hyperedges added throughout the l experiments. □

2.6. Pebbling Games: Register Allocation and Processor Scheduling

The application we study in this section, as with that of Section 2.3, is an abstract mechanism, called (*graph*) *pebble games*, for studying a variety

of real computational problems that involve the allocation of computational resources. Notable among the problems that can be abstracted to pebble games is the allocation of registers for an interdependent sequence of (say, arithmetic) operations (Cook [1974], Paterson and Hewitt [1970]) and the scheduling of processes on a multiprocessor, using a client-server scheduling regimen (Bhatt *et al.* [1996b]).

The medium for the pebble games studied in this section is the class of *directed acyclic graphs* (*dags*). In the register-allocation (RA) scenario, the nodes of a dag represent operations and its arcs represent data dependencies: an arc from node u to node v indicates that the operation at node v requires data that is produced by the operation at node u. In the processor-scheduling (PS) scenario, the nodes of a dag represent processes and its arcs represent data dependencies: an arc from node u to node v indicates that the process at node v requires input data that is produced by the process at node u. Clearly, these two scenarios are almost identical.

The process of allocating registers to data in an RA-dag or of managing the processes eligible for execution in a PS-dag is represented formally by a *pebble game*. We present the formalities of the game, assuming that the reader can easily map the game's features to the features of the two motivating computational scenarios (as well as others). We present a version of the pebble game that is somewhat nonstandard but is equivalent to the standard version (Cook [1974], Paterson and Hewitt [1970]) when one wants to measure the required number of pebbles (as opposed to, say, the required number of steps) in a play of the game.

The pebble game. We are given a finite dag $\mathscr{G}$ and endless supplies of two types of tokens, respectively called *enabling pebbles* and *execution pebbles.* The rules of a single step of the game are as follows.

1. One places an *execution pebble* on any single node of $\mathscr{G}$ all of whose incoming arcs contain *enabling pebbles.* Note that, at the beginning of the game, only the *source* nodes of $\mathscr{G}$ (i.e., those having no incoming arcs, hence satisfying this condition vacuously) are eligible for pebbling.
2. One removes the *enabling pebbles* from all arcs that enter the just-executed node.
3. One places *enabling pebbles* on all arcs that leave the just-executed node.

The game ends when every node of $\mathscr{G}$ contains an execution pebble.

Of course, when the dag $\mathscr{G}$ has nontrivial structure, one has many choices at each step of the pebble game on $\mathscr{G}$, as several nodes will typically be eligible for execution. Indeed, different plays of the game will often

require different numbers of "active" enabling pebbles. The goal is to find a play of the game that minimizes this number.

The *cost of a play of the pebble game* on a dag $\mathcal{G}$ is the maximum number of *enabling pebbles* that ever reside on the arcs of $\mathcal{G}$ during a step in the play of the game.

Not surprisingly (we hope, by this point in the book), the separation characteristics of a dag $\mathcal{G}$ can induce a nontrivial lower bound on the cost of playing the pebble game on $\mathcal{G}$.

PROPOSITION 2.6.1. *Any play of the pebble game on a dag $\mathcal{G}$ must use a number of enabling pebbles no smaller than the maximum M-edge-separation-width of $\mathcal{G}$; i.e.,*

$$\max_{1 \leq M < N(\mathcal{G})} W_{\mathcal{G}}(M)$$

PROOF SKETCH. For simplicity, we use the terminology of the PS game. At every moment t in an execution of the pebble game on $\mathcal{G}$, the arcs that contain enabling pebbles separate the set of nodes of $\mathcal{G}$ that have already been executed from the set of nodes that are yet to be executed. Moreover, the number of executed nodes increases by precisely one at each step of the game. □

The fact that the bound of Proposition 2.6.1 involves a maximization over all possible partition sizes is especially important for dags such as trees whose separation-widths are dramatically nonmonotonic as a function of the partition size. For instance, one can bisect the N-node complete binary tree by cutting just one edge (adjacent to the root of the tree), whereas cutting the tree into, say, a 1/3-2/3 partition requires cutting roughly $\log N$ edges, as we shall see in Section 4.4.

2.7. Sources

The first wide-ranging catalogue of computational problems that yield to graph separation techniques appears in Lipton and Tarjan [1980], which is devoted to situations that can be modeled using planar graphs. The strategy presented in Section 2.2 for solving nonserial dynamic programming problems by recursively separating the problems' variable-interaction graphs is a straightforward generalization of the technique presented in Lipton and Tarjan [1980] for planar Variable-Interaction graphs.

Theorem 2.3.1 derives from Bhatt *et al.* [1996a]. The use of bucket trees as intermediate host graphs in embeddings seems to originate (under

another name) with Bhatt *et al.* [1992]. The strategy of using intermediate host graphs in embeddings seems to originate with Leiserson [1985] and its nonarchival precursors. The lower bounds of Section 2.3.2 are harder to trace than the upper bounds. Proposition 2.3.4 may well be original in its stated form, although the reasoning leading to it appears at least implicitly in Rosenberg and Snyder [1978]. Proposition 2.3.6 originates in various versions in Rosenberg and Snyder [1978] and Hong and Rosenberg [1982]. Proposition 2.3.7 is implicit from its use with hypercube-guests in Harper [1966]. Proposition 2.3.8 is implicit from its use with mesh-guests in Sheidvasser [1974]. Corollary 2.3.9 seems to have been rediscovered numerous times, appearing (for mesh- and tree-guest graphs, respectively) in DeMillo *et al.* [1978a] and Iordanskíi [1976]. Corollary 2.3.10 seems to be original. Corollary 2.3.11 originates in Sheidvasser [1974].

The first formalization of VLSI layout as a graph-embedding problem appeared in Thompson [1980], wherein the layouts of specific families of circuits (defined by the function the circuit computed) were studied. Soon thereafter, the framework of Thompson [1980] was adapted, in Leiserson [1983] and Valiant [1981], to yield strategies for laying out arbitrary circuits, based only on their separation properties. The layout strategy we present in Section 2.4, which culminates in Theorem 2.4.1, adapts the strategy presented in Leiserson [1983] to the framework of *graph bifurcators* developed in Bhatt and Leighton [1984] (whereas Leiserson [1983] uses β-separators). This adaptation, which is only part of the contribution of Bhatt and Leighton [1984], is quite important, as the original, separator-based strategy does not yield the universally quantified lower bounds on area that the bifurcator-based strategy does. A more sophisticated layout strategy than the one we use in Section 2.4 appeared in Leiserson [1985]; this sophisticated strategy, which allows one to solve many more problems than just simple circuit layout, culminated in the definitive treatment of layout problems in Bhatt and Leighton [1984]. Building on the case study of the potential added efficiency of *three-dimensional* circuit layouts in Rosenberg [1983], which was refined in [180], the studies in Leighton and Rosenberg [1983, 1986] extended the general layout paradigm of Leiserson [1983] to three-dimensional circuit layouts. The more sophisticated strategy of Leiserson [1985] was generalized to three-dimensional layouts in Greenberg and Leiserson [1988]. Finally, the simple lower-bound technique of Theorem 2.4.2 is a straightforward adaptation of ideas in Thompson [1980].

The development in Section 2.5 comes from Rosenberg [1989], which generalizes the case studies in Rosenberg [1985].

Finally, the study of register allocation via pebbling games, as described in Section 2.6, originates in Paterson and Hewitt [1970] and is studied further in Cook [1974], the study of multiprocessor scheduling via pebbling

games seems to originate in Bhatt *et al.* [1996b]. We believe that Proposition 2.6.1 is original here, but the result builds on insights in the cited sources, especially Bhatt *et al.* [1996b]. We shall see in Chapter 4 that the proposition yields, via a very different proof technique from that found in the literature, most of the known lower bounds on pebble number. Additional sources that expose the relevance of pebbling games to the study of graph-theoretic problems are Lengauer [1981], which relates a family of pebbling games on graphs to the separation-widths of the graphs, and Rosenberg and Sudborough [1983] which relates a family of pebbling games on graphs to the bandwidths of the graphs.

In addition to the cited sources, we list in the bibliography a variety of sources not included in this chapter, which use graphs and their separators to study a variety of computational situations.

Notes

1. We are grateful to the authors and publisher of Lipton and Tarjan [1980] for permission to paraphrase from that source as the starting point of this section.
2. We use the phrase "absolute constant" to emphasize that the value of ε is fixed for the entire family $\mathbf{G}$ and does not change for different graphs in the family.
3. We are grateful to the authors and publisher of Bhatt *et al.* [1996a] for permission to paraphrase from that source, especially in Theorem 2.3.1.
4. We place the word "embed" in quotes and stress the many-to-one nature of the node-assignment in order to emphasize the departure here from our usual insistence that embeddings be one-to-one.
5. Recall that λ denotes the null string.
6. We are grateful to the authors and publisher of Bhatt and Leighton [1984] for permission to paraphrase from that source.
7. For three-dimensional layouts, one uses small-size $2^{2/3}$-bifurcators (Leighton and Rosenberg [1986]).
8. That is, we count the number of endpoints of edges, even though some nodes may be the endpoints of more than one edge.
9. They are also called *rearrangeable networks*.
10. They are also called *barrel shifters*.
11. We are grateful to the publisher of Rosenberg [1989] for permission to paraphrase from that source.
12. By a "collinear" layout, we mean one in which the graph's nodes lie along a line, with the graph's edges running above the line.

13. Recalling that $-k/\log(1-\beta) = \ell(2^k)$ [cf. (2.5.1)] may lend the reader some intuition in understanding (2.5.2).
14. The cited SSPs for trees and meshes can be derived by considering the sizes of "perimeters" of regions within the graphs, using the techniques of Chapter 4.
15. The metaphor of unit-width routing tracks running among (in this case, above) devices is an alternative to the mesh-based model for VLSI layouts that we used in Section 2.4; in fact, it is the original model from Thompson [1980].

This page intentionally left blank

3

Upper-Bound Techniques

3.1. Introduction

This chapter is devoted to developing techniques for deriving upper bounds on the size of a graph's smallest separators, specifically on its smallest β-edge- and β-node-separators, in the sense of Section 1.4. We begin with the computational difficulty of this task.

Each of the notions of graph separator that we have discussed in Section 1.4 and Chapter 2 suggests a corresponding optimization problem. The following two problems are typical of the genre.

1: MINIMUM EDGE-BISECTION. Graph $\mathcal{G}$ and produce a partition of the node-set $\mathbf{N}(\mathcal{G})$ into sets $\mathbf{N}_1$ and $\mathbf{N}_2$ such that $|N(\mathbf{N}_1) - N(\mathbf{N}_2)| \leqslant 1$, and the number of edges that connect $\mathbf{N}_1$ and $\mathbf{N}_2$ is minimum.

2: MINIMUM NODE-SEPARATION. Graph $\mathcal{G}$ and produce a partition of $\mathbf{N}(\mathcal{G})$ into sets A, B, and C such that $|A| \leqslant (2/3)N(\mathcal{G})$, $|B| \leqslant (2/3)N(\mathcal{G})$, there are no edges between A and B, and C is as small as possible.

One of the earliest complexity results was an NP-completeness proof (see Cormen *et al.* [1990] for definitions) for (the decision-problem version of) MINIMUM EDGE-BISECTION; Section 3.2 presents this proof. Subsequent research has shown that essentially any nontrivial notion of a graph separation decision problem is also NP-complete. As a consequence of this putative computational intractability of graph separation, in the face of the problem's myriad applications, there has been considerable research, in

many different directions, aimed at discovering tractable approaches to the problem. One well-studied direction is to seek algorithms that discover provably good separators for specific families of graphs rather than for general graphs. The classic families for this direction include planar graphs and graphs that can be embedded in an orientable surface of genus g. Section 3.3 presents separation algorithms for these families using a topological approach. A related approach uses geometric rather than topological information about a graph. One embeds a graph into d-dimensional Euclidean space and uses geometric properties to obtain bounds on the sizes of separators and to devise algorithms which find separators that achieve those bounds. We explore this approach in Section 3.4.

The classical max-flow min-cut theorem from the theory of flows in networks suggests that one might be able to use a maximum flow in a network to find a good edge-separator for the graph that underlies the network. The main drawback one must overcome in this approach is that a min-cut carries no guarantee of size balance in the partition it defines. Section 3.5 discusses some algorithms that adapt network flow ideas to finding good edge-separators.

Finally, in Section 3.6, we consider heuristic approaches to finding graph bisections. While these heuristics provide no guarantees on the number of edges cut by the bisection, several of them have been found to be very efficient in practice. We present and discuss two simple such heuristics.

One significant approach to efficient graph separation that we do not cover in this chapter is the algebraic approach based on eigenvalues (or spectra) of graphs. A typical formulation of this approach represents a graph via a matrix, called the *Laplacian*, and computes one (or more) eigenvector(s) of the matrix, each eigenvector having one entry for each graph-node. One sorts the entries of the eigenvector into increasing order, placing the nodes corresponding to the $N/2$ smallest entries into one part of the partition and the remaining nodes into the other part. Several variations of this typical formulation have been explored. Motivating the approach is the fact that the second eigenvalue of the Laplacian appears in a formula that gives a lower bound on the number of edges cut by any partition of the graph. Eigenvalue techniques also play an important role in the construction of expander graphs (a problem with close ties to graph separation). A variety of recent results are referenced in Section 3.7.

The pseudocode style we use in this chapter is adapted from the style in the now-standard algorithms text (Cormen *et al.* [1990]). Note, in particular, that we use $\triangleright$ as a comment indicator.

3.2. NP-Completeness

This section is devoted to presenting and proving a few basic NP-completeness results for edge-separation problems. As noted earlier, most variants of edge- and node-separation lead to NP-complete problems, so we try here to indicate by example the sources of the NP-completeness rather than to be exhaustive in our coverage. The reader interested in a large catalogue of such problems should consult Garey and Johnson [1979] as a starting place.

Throughout this section, let $\mathcal{G}$ be a graph, and let the pair of sets $(\mathbf{N}_1, \mathbf{N}_2)$ be a partition of $\mathbf{N}(\mathcal{G})$. The (*edge*)-*cut* $\text{cut}(\mathbf{N}_1, \mathbf{N}_2)$ is that subset of $\mathbf{E}$ comprising edges that have one endpoint in $\mathbf{N}_1$ and the other in $\mathbf{N}_2$. The partition is a *bisecting partition* (a *bisection*, for short) if $N(\mathbf{N}_1) = \lceil N/2 \rceil$ and $N(\mathbf{N}_2) = \lfloor N/2 \rfloor$. The key problem shown NP-complete in this section is

3: MINIMUM BISECTION-WIDTH (MinBW). Graph $\mathcal{G}$ and an integer K, where $0 \leqslant K \leqslant \lceil N/2 \rceil \lfloor N/2 \rfloor$. Is there a bisection $(\mathbf{N}_1, \mathbf{N}_2)$ of $\mathbf{N}(\mathcal{G})$ such that $|\mathbf{cut}(\mathbf{N}_1, \mathbf{N}_2)| \leqslant K$?

Our proof of the NP-completeness of MinBW builds on the intractability of three other decision problems, which we now describe.

Recall that a *literal* in a boolean variable x is either x itself (the uncomplemented variable) or $\bar{x}$ (the complemented variable). Let $U = \{x_1, x_2, ..., x_n\}$ be a set of boolean variables. A *truth assignment* for U is a function that assigns to each variable in U a (boolean) value TRUE or FALSE. A *clause* C over U is a set of literals in variables from U. We say that a truth assignment *satisfies* clause C if, as a consequence of the assignment, at least one literal in C is made TRUE. The first decision problem is the classical NP-complete problem 3SAT.

4: 3-SATISFIABILITY (3SAT). Set $U = \{x_1, x_2, \ldots, x_n\}$ of variables, clauses $C_1, C_2, \ldots, C_m$ over U, each of cardinality exactly 3. Is there a truth assignment for the variables in U that satisfies every C_i?

Since our key problem, MinBW, is derived from an optimization problem, we consider next an optimization-oriented variant of 3-SAT.

5: MAXIMUM 2-SATISFIABILITY (MAX 2SAT). Set $U = \{x_1 x_2, \ldots, x_n\}$ of variables, clauses $C_1, C_2, \ldots, C_m$ over U, each of cardinality either 1 or 2, a positive integer K, where $1 \leqslant K \leqslant m$. Is there a truth assignment for every variable in U that satisfies at least K of the clauses C_i?

The *maximization* analogue of MinBW turns out to be easier to reason about than MinBW itself, so our third problem is

6: MAXIMUM BISECTION-WIDTH (MaxBW). Graph $\mathcal{G}$ and an integer K, where $0 \leqslant K \leqslant \lceil N/2 \rceil \lfloor N/2 \rfloor$. Is there a bisection $(\mathbf{N}_1, \mathbf{N}_2)$ of $\mathbf{N}$ such that $|\mathbf{cut}(\mathbf{N}_1, \mathbf{N}_2)| \geqslant K$?

Since the membership of our three decision problems in the class NP is obvious, we concentrate only on the proofs of their NP-hardness. (Again, see Cormen *et al.* [1990] for definitions.) We begin with the proof for MAX 2SAT.

THEOREM 3.2.1. MAX 2SAT *is NP-hard.*

PROOF. We build on the well-known NP-hardness of 3SAT and reduce that problem to MAX 2SAT.

Let the set $U = \{x_1, x_2, \ldots, x_n\}$ of boolean variables and the clauses $C_1, C_2, \ldots, C_m$ over U constitute an instance of 3SAT. We describe how to construct a corresponding instance of MAX 2SAT, leaving to the reader the easy verification that the construction can be performed in polynomial time.

Say that each clause C_i consists of the three literals a_i, b_i, c_i. For each $i \in \{1, 2, \ldots, m\}$, introduce a new variable d_i, and define the following 10 clauses, each having cardinality at most 2:

$$D_i^1 = \{a_i\} \qquad D_i^2 = \{b_i\}$$
$$D_i^3 = \{c_i\} \qquad D_i^4 = \{d_i\}$$
$$D_i^5 = \{\bar{a}_i, \bar{b}_i\} \qquad D_i^6 = \{\bar{a}_i, \bar{c}_i\}$$
$$D_i^7 = \{\bar{b}_i, \bar{c}_i\} \qquad D_i^8 = \{a_i, \bar{d}_i\}$$
$$D_i^9 = \{b_i, \bar{d}_i\} \qquad D_i^{10} = \{c_i, \bar{d}_i\}$$

Fix $i \in \{1, 2, \ldots, m\}$, and fix truth assignments for a_i, b_i, and c_i.

Claim. (a) Any truth assignment for d_i will satisfy no more than 7 of the 10 clauses D_i^j. (b) There is a truth assignment for d_i which satisfies exactly seven clauses if, and only if, the truth assignments for a_i, b_i and c_i satisfy C_i.

To prove this claim, we determine how many clauses are satisfied for each truth assignment. We tabulate the possible counts of satisfied clauses in Figure 3.2-1. Due to the symmetry in the roles of a_i, b_i, and c_i in the 10 clauses, it suffices to count the number of TRUES among the assignments to those three variables (the second column of Figure 3.2-1). The reader can readily verify the count of TRUE clauses and hence check the validity of the claim.

d_i	Count of TRUE in $\{a_i, b_i, c_i\}$	Count of TRUE clauses
FALSE	0	6
FALSE	1	7
FALSE	2	7
FALSE	3	6
TRUE	0	4
TRUE	1	6
TRUE	2	7
TRUE	3	7

Figure 3.2-1. Summary of truth assignments for 2SAT instance.

Now let the corresponding instance of MAX 2SAT be

- $U' = U \cup \{d_1, d_2, \ldots, d_m\}$.
- The clauses D_i^j, where $i \in \{1,2,\ldots,m\}$ and $j \in \{1, 2, \ldots, 10\}$.
- $K = 7m$.

Pick any truth assignment for U'. By the claim, if this truth assignment results in $7m$ of the clauses D_i^j being satisfied, then the truth assignment results in every C, being satisfied. Conversely, also by the claim, if there is a satisfying truth assignment for the instance of 3SAT, then we can choose a truth assignment for 2SAT having exactly $7m$ of the clauses D_i^j satisfied. It follows that there is a satisfying truth assignment for the instance of 3SAT if, and only if, there is a truth assignment for the instance of MAX 2SAT which satisfies at least K clauses.

We have thus presented a polynomial-time reduction from 3SAT to MAX 2SAT, whence the latter problem is NP-hard.

We turn next to MaxBW.

THEOREM 3.2.2. Max BW is *NP-hard.*

PROOF. We reduce MAX 2SAT to MaxBW. Let the set

$$U = \{x_1, x_2, \ldots, x_m\}$$

of variables, the clauses $C_1, C_2, \ldots, C_m$ over U, where each C_i has either one or two literals, and the integer K constitute an instance of MAX 2SAT. We describe how to construct a corresponding instance $\mathcal{G}, K'$ of MaxBW, leaving to the reader the easy verification that the construction can be performed in polynomial time.

The node-set $\mathbf{N}(\mathcal{G})$ is the union of the following $2n + 2$ mutually disjoint sets:

$$\mathbf{T} = \{T_j : 1 \leqslant j \leqslant 4m\}$$
$$\mathbf{F} = \{F_j : 1 \leqslant j \leqslant 4m\}$$
$$\mathbf{X}_i = \{x_i^j : 1 \leqslant j \leqslant 4m\} \qquad \text{for } i = 1,2,\ldots,n$$
$$\bar{\mathbf{X}}_i = \{\bar{x}_i^j : 1 \leqslant j \leqslant 4m\} \qquad \text{for } i = 1,2,\ldots,n$$

We shall force the sets $\mathbf{T}$ and $\mathbf{F}$ to reside in opposite parts of any bisecting partition of $\mathcal{G}$, thereby identifying the TRUE and FALSE sides of the partition. We shall also force each set $\mathbf{X}_i$ to reside in the opposite part of the partition from $\bar{\mathbf{X}}_i$, thereby causing the sets to act as "complements" of each other. If the set $\mathbf{X}_i$ appears in the same part as the set $\mathbf{T}$ (resp., the set $\mathbf{F}$), this will be interpreted as the variable x_i being assigned TRUE (resp., FALSE).

The edge-set $\mathbf{E}(\mathcal{G})$ is also best described in parts. The first installment, $\mathbf{E}_1$, on $\mathbf{E}(\mathcal{G})$ emerges from making one complete bipartite graph on the sets $\mathbf{T}$ and $\mathbf{F}$ (i.e., these sets are the "parts" of the graph) and one complete bipartite graph on each pair of sets $\mathbf{X}_i$ and $\bar{\mathbf{X}}_i$; formally,

$$\mathbf{E}_1 = \{(T_p, F_q) : 1 \leqslant p, q \leqslant 4m\} \cup \{(x_i^p, \bar{x}_i^q) : 1 \leqslant p, q \leqslant 4m,\ 1 \leqslant i \leqslant n\}$$

Note that $|\mathbf{E}_1| = 16m^2 + 16nm^2$, that every node of $\mathcal{G}$ resides in exactly one of the complete bipartite graphs, and that every node has degree $4m$ in its complete bipartite graph.

The remaining edges in $\mathbf{E}(\mathcal{G})$ represent the clauses. Fix a clause C_j. For each literal $z \in C_j$, there is a *corresponding node*, corr(z), in $\mathcal{G}$, selected as

follows. If z is the uncomplemented variable x_i, then corr(z) = x_i^j; if z is the complemented variable $\bar{x}_i$, then corr(z) = $\bar{x}_i^j$. The edges that represent clause C_j join two nodes in $\mathbf{F}$ to the clause's one or two corresponding nodes:

$$\mathbf{E}_2 = \{(\text{corr}(u), \text{corr}(v)) : u \neq v \text{ and some clause } C_j = \{u, v\}\}$$
$$\cup \{(\text{corr}(u), F_{2j-1}), (\text{corr}(v), F_{2j}) : 1 \leqslant j \leqslant m, C_j = \{u, v\}\}$$

We thus add short paths within $\mathcal{G}$: a 3-path for each clause of size 2 and a 2-path for each clause of size 1. Note that $|\mathbf{E}_2| \leqslant 3m$, which is less than the degree of each node in its complete bipartite graph.

This completes the construction of $\mathcal{G}$: $\mathcal{G} = (\mathbf{N}, \mathbf{E})$, where $\mathbf{E} = \mathbf{E}_1 \cup \mathbf{E}_2$, and the bound for the instance of MaxBW is $K' = |\mathbf{E}_1| + 2K = 16m^2 + 16nm^2 + 2K$. The construction guarantees that $N(\mathbf{E}) - K' < 4m$.

Consider now any bisection ($\mathbf{N}_1$, $\mathbf{N}_2$) of $\mathbf{N}$ for which $|\text{cut}(\mathbf{N}_1, \mathbf{N}_2)| \geqslant K'$. We claim that (as planned) the set $\mathbf{T}$ is wholly contained in one of the two parts, and $\mathbf{F}$ is wholly contained in the other. Were this not the case, no more than $16m^2 - 4m$ of the edges in the complete bipartite subgraph on $\mathbf{T}$ and $\mathbf{F}$ would be in cut($\mathbf{N}_1$, $\mathbf{N}_2$), so that |cut($\mathbf{N}_1$, $\mathbf{N}_2$)| would be no larger than $\mathbf{N}(\mathbf{E}) - 4m < K'$. Similarly, each set $\mathbf{X}_i$ is wholly contained in one of the parts, and its complementary set $\bar{\mathbf{X}}_i$ is wholly contained in the other. Without loss of generality, say that $\mathbf{T} \subset \mathbf{N}_1$ and $\mathbf{F} \subset \mathbf{N}_2$. For each $i \in \{1, 2, \ldots, n\}$, assign variable x_i the value TRUE if $\mathbf{X}_i \subset \mathbf{N}_1$; otherwise, assign x_i FALSE. Since $|\text{cut}(\mathbf{N}_1, \mathbf{N}_2)| \geqslant K'$, at least $2K$ edges from E_2 reside in the cut. Consider a specific (but arbitrary) clause $C_j = \{u, v\}$. Either zero or two of the edges representing C_j reside in cut($\mathbf{N}_1$, $\mathbf{N}_2$), depending on whether the two nodes corr(u) and corr(v) are both in $\mathbf{N}_2$ or not. Hence, clause C_j contributes two edges to cut($\mathbf{N}_1$, $\mathbf{N}_2$) exactly when one or both of u and v are assigned TRUE—that is, exactly when the clause is satisfied—and contributes no edges otherwise. We conclude that at least K clauses are satisfied by the truth assignment.

Conversely, say that we have a truth assignment for the variables in U that satisfies at least K clauses. Choose the unique bisecting partition $\mathbf{N}_1$ and $\mathbf{N}_2$ for $\mathbf{N}$ that satisfies the following subset relations:

$$\mathbf{T} \subset \mathbf{N}_1$$
$$\mathbf{F} \subset \mathbf{N}_2$$
$$\mathbf{X}_i \subset \begin{cases} \mathbf{N}_1 & \text{if } x_i = \text{TRUE} \\ \mathbf{N}_2 & \text{otherwise} \end{cases}$$
$$\bar{\mathbf{X}}_i \subset \begin{cases} \mathbf{N}_1 & \text{if } x_i = \text{FALSE} \\ \mathbf{N}_2 & \text{otherwise} \end{cases}$$

It is straightforward to observe that $\mathbf{cut}(\mathbf{N}_1, \mathbf{N}_2) \geq K'$. Hence $\mathbf{N}_2$ and $\mathbf{N}_2$ is the sought bisecting partition for $\mathcal{G}$.

We conclude that the given instance of MAX 2SAT has a truth assignment satisfying at least K clauses if, and only if, the constructed instance of MaxBW has a bisecting partition with cut at least K'. We thus have a polynomial-time reduction of MAX 2SAT to MaxBW, whence the latter problem is NP-hard.

The assumption in Theorem 3.2.2 that the partition $(\mathbf{N}_1, \mathbf{N}_2)$ is a *bisection* is really not essential for the NP-hardness of the separation problem. The following more general separation problem is still NP-complete.

7: MAXIMUM BOUNDED-RATIO SEPARATION WIDTH (MaxBRSW). Graph $\mathcal{G}$, integers p and q, each in $\{1,2,\ldots,N\}$, and an integer $K \in Z_{N^2}$. Is there a partition $(\mathbf{N}_1, \mathbf{N}_2)$ of $\mathbf{N}(\mathcal{G})$ such that $|\mathbf{cut}(\mathbf{N}_1, \mathbf{N}_2)| \geq K$ and

$$\frac{N(\mathbf{N}_1)}{N(\mathbf{N}_2)} = \frac{p}{q}?$$

THEOREM 3.2.3. MaxBRSW *is NP-hard.*

PROOF. One need only modify the proof of Theorem 3.2.2 as follows. Choose the cardinality of $\mathbf{T}$ so that $\mathbf{T} \geq 4m$ and $4nm + |\mathbf{T}|$ is divisible by p. Choose the cardinality of $\mathbf{F}$ so that

$$|\mathbf{F}| = \frac{q(4nm + |\mathbf{T}|)}{p} - 4nm$$

Note that the cardinalities of $\mathbf{T}$ and $\mathbf{F}$ are polynomially-bounded functions of N. Set $K' = |\mathbf{T}||\mathbf{F}| + 16nm^2 + 2K$. The remainder of the proof is as before, *mutatis mutandis.* □

Finally, we are ready for the main result of the section.

THEOREM 3.2.4. MinBW *is NP-hard.*

PROOF. We now have the machinery to reduce MaxBW to MinBW via a simple mapping. Let graph $\mathcal{G}$ and integer K constitute an instance of MaxBW. The corresponding instance of MinBW is the complementary

graph $\bar{\mathcal{G}}$ and the integer $K' = \lceil N/2 \rceil \lfloor N/2 \rfloor - K$. Easily, any bisecting partition of $\mathcal{G}$ which cuts at least K edges is also a bisecting partition of $\bar{\mathcal{G}}$ which cuts at most K' edges, and vice versa. Hence, $\mathcal{G}$ has a bisecting partition of cardinality $\geqslant K$ if, and only if, $\bar{\mathcal{G}}$ has a bisecting partition of cardinality $\leqslant K'$. The problem MaxBW thus reduces in polynomial time to MinBW, whence the latter problem is NP-hard. □

In Theorem 3.2.4, as in Theorem 3.2.2, the assumption that the graph partition be a *bisection* is not essential for the NP-hardness of the separation problem. The following more general separation problem is also easily shown to be NP-complete.

8: MINIMUM BOUNDED-RATIO SEPARATION WIDTH (MinBRSW). Graph $\mathcal{G}$, integers p and q, each in $\{1,2,\ldots,N\}$, and an integer $K \in Z_{N^2}$. Is there a partition $(\mathbf{N}_1, \mathbf{N}_2)$ of $\mathbf{N}$ such that $|\mathbf{cut}(\mathbf{N}_1, \mathbf{N}_2)| \leqslant K$ and

$$\frac{N(\mathbf{N}_1)}{N(\mathbf{N}_2)} = \frac{p}{q}?$$

THEOREM 3.2.5. MinBRSW *is NP-complete.*

Of course, this problem remains NP-complete if p and q are fixed integers with $1 \leqslant p \leqslant q$.

It is difficult to obtain even an *approximation* (in the sense of Cormen *et al.* [1990]): to the minimal edge separator for a graph. For a graph $\mathcal{G}$, let $\mathbf{MC}(\mathcal{G})$ be the minimal cardinality of any bisection of $\mathcal{G}$. Coming close to $\mathbf{MC}(\mathcal{G})$ is as good as actually reaching the minimum exactly, in the sense made precise by the following theorem.

THEOREM 3.2.6. *If there were a polynomial-time algorithm that produced, for any graph* $\mathcal{G}$, *a bisection* $(\mathbf{N}_1, \mathbf{N}_2)$ *of* $\mathcal{G}$ *for which* $|\mathbf{cut}(\mathbf{N}_1, \mathbf{N}_2)| \leqslant \mathbf{MC}(\mathcal{G}) + \sqrt{N}$, *then there would be a polynomial-time algorithm that solves* MinBW.

PROOF. We show how to construct an algorithm, call it B, that solves MinBW from the presumed approximation algorithm, call it A.

Let us be given an instance of MinBW, consisting of a graph $\mathcal{G}$ and an integer K. We lose no generality by assuming that $N(\mathcal{G})$ is even, for if it were odd, we could merely augment $\mathcal{G}$ with a new node that is adjacent to all other nodes, add $(N + 1)/2$ to K, and proceed.

The putative Algorithm B begins by constructing from $(\mathcal{G}, K)$ a polynomially larger graph $\mathcal{G}'$ as follows. The nodes of $\mathcal{G}'$ are obtained by placing a clique on N^7 nodes into $\mathcal{G}'$ for each node of $\mathcal{G}$; hence, $N' = N^8$. In addition to the edges that come along with these cliques, B also places $2N^4$ edges into $\mathbf{E}(\mathcal{G}')$ for each edge $(u, v) \in \mathbf{E}(\mathcal{G})$; these edges somehow connect nodes in the clique corresponding to u to nodes in the clique corresponding to v. One can choose endpoints for these interclique edges arbitrarily, for the endpoints are immaterial. Thus, in all, $\mathbf{E}(\mathcal{G}')$ contains $2N^4$ "copies" of each edge from $E(\mathcal{G})$, together with all of the clique-edges, for a total of

$$E' = 2N^4E + N\binom{N^7}{2}$$

edges.

Algorithm B next passes the graph $\mathcal{G}'$ as an input to Algorithm A, in response to which the latter returns a bisecting partition $(\mathbf{N}_1, \mathbf{N}_2)$ with $|\mathbf{cut}(\mathbf{N}_1, \mathbf{N}_2)| \leqslant \mathrm{MC}(\mathcal{G}') + N'$. From this partition, Algorithm B derives an optimal bisection $(\mathbf{N}_1^*, \mathbf{N}_2^*)$ for $\mathcal{G}$, via a technique explained at the end of the proof.

Since Algorithm A is a polynomial-time algorithm, and the construction of $\mathcal{G}'$ requires polynomial time, Algorithm B is a polynomial-time algorithm.

Now to the quality of the partition.

Claim. $\mathrm{MC}(\mathcal{G}') = 2N^4 \cdot \mathrm{MC}(\mathcal{G})$.

To see this, note first that

$$\mathrm{MC}(\mathcal{G}) \leqslant \binom{N/2}{2} < N^2 - 1.$$

Clearly

$$\mathrm{MC}(\mathcal{G}') \leqslant 2N^4 \cdot \mathrm{MC}(\mathcal{G}) < 2N^6 - 2N^4,$$

because any bisection of $\mathcal{G}$ can be trivially converted to a bisection of $\mathcal{G}'$ that has a cut of cardinality a factor $2N^4$ greater. Finally, note that if any of the cliques that make up $\mathcal{G}'$ is not entirely contained in either $\mathbf{N}_1$ or $\mathbf{N}_2$, then $|\mathbf{cut}(\mathbf{N}_1, \mathbf{N}_2)| \geqslant N^7 - 1$, which contradicts the assumed behavior of Algorithm A. The claim is established.

The partition $(\mathbf{N}_1, \mathbf{N}_2)$ returned by Algorithm A satisfies

$$|\mathbf{cut}(\mathbf{N}_1, \mathbf{N}_2)| \leqslant \mathrm{MC}(\mathcal{G}') + \sqrt{N'} = 2N^4 \cdot \mathrm{MC}(\mathcal{G}) + N^4$$

and, as just argued, every clique is entirely contained in either $\mathbf{N}_1$ or $\mathbf{N}_2$. It follows that $\mathbf{N}_1$ and $\mathbf{N}_2$ uniquely determines a bisecting partition $\mathbf{N}_1^*$ and $\mathbf{N}_2^*$

of $\mathcal{G}$ with

$$|\mathrm{cut}(\mathbf{N}_1^*, \mathbf{N}_2^*)| = \frac{|\mathrm{cut}(\mathbf{N}_1, \mathbf{N}_2)|}{2N^4}$$

$$\leqslant \frac{2N^4 \mathrm{MC}(\mathcal{G}) + N^4}{2N^4}$$

$$= \mathrm{MC}(\mathcal{G}) + 1/2$$

Hence, $|\mathrm{cut}(\mathbf{N}_1^*, \mathbf{N}_2^*)| = \mathrm{MC}(\mathcal{G})$. The partition $(\mathbf{N}_1, \mathbf{N}_2)$ is the optimal partition returned by Algorithm *A*. □

As an immediate consequence, we observe the following.

COROLLARY 3.2.7. *If* $P \neq NP$, *then there is no polynomial-time algorithm that takes a graph* $\mathcal{G}$ *and returns a bisecting partition of* $\mathcal{G}$ *of size* $\leqslant \mathrm{MC}(\mathcal{G}) + \sqrt{N}$.

Stronger results are known; see the source in Section 3.7.

3.3. Topological Approaches to Graph Separation

The literature on graph separators contains a large number of separator theorems for graphs that are embeddable topologically into a variety of surfaces. Happily, these theorems usually provide not only upper bounds on the separation widths of the graphs in question, but also efficient algorithms that produce separators of these sizes, given the promised embeddings. Less happily, the bounds one finds in these theorems are usually of the *minimax* variety, deviating from optimality only by small amounts—often only a small constant factor—for the largest-separator graphs in the subject family, but providing no information about separator-sizes of individual graphs. For instance, the many known algorithms for (roughly) bisecting planar graphs promise node-separators of size $O(\sqrt{n})$ for any n-node planar graph but make no tighter promises when, for instance, the input planar graph is actually outerplanar[1] (so that a separator of size $O(\log n)$ actually exists (Diks *et al.* [1993])).

In this section we present a separator theorem for graphs that are embeddable (topologically) into an oriented surface of arbitrary genus. We start with the rudiments of topological graph theory (in Section 3.3.1),

continue with the now-classical separator theorem for planar graphs (in Section 3.3.2), and conclude with the extension of that theorem to graphs that are embeddable onto surfaces of arbitrary genus (in Section 3.3.3).

3.3.1. Topological Warmup

We now introduce the basic terminology and notions from topological graph theory, the subject of graphs embedded into surfaces. We avoid precise definitions of basic topology in favor of intuition, a decision justified by the fact that we are able to proceed purely combinatorially, with the topology merely giving us inspirational images.

For our purposes a *surface* is a subset of three-dimensional Euclidean space that is locally homeomorphic to a disk. We consider only compact (bounded and closed) surfaces (also called 2-manifolds). Since our surfaces are subsets of three-dimensional space, they are *orientable*, in the sense of having well-defined insides and outsides. A sphere, or a surface homeomorphic to a sphere, provides the simplest example of a compact surface. The classification theorem for compact surfaces states that each surface has a single nonnegative integer parameter, called its *genus*, that completely characterizes it topologically. Informally, the genus of the surface is the number of handles (or tubes) that one must add to a sphere in order to obtain (a surface that is homeomorphic to) the desired surface. One may also think of a genus-g surface as being a sphere that is punctured by g holes.

Our main interest here is in drawings of connected graphs on surfaces in which no two graph edges cross (except at a shared node). Easily, such a surface exists for every connected graph as long as one is allowed to endow the surface with sufficiently many handles. Topologically, each such drawing of a graph $\mathcal{G}$ is a continuous, one-to-one function mapping $\mathcal{G}$ into the surface; we call the drawing an *embedding* of $\mathcal{G}$ into the surface. The minimum genus of a surface that $\mathcal{G}$ can be embedded into is called the *genus* of $\mathcal{G}$. The complement of (the image of) $\mathcal{G}$ in the surface consists of a finite number F of connected, two-dimensional sets called the *faces* of the embedding. If each face is homeomorphic to the unit disk, then the drawing is a *2-cell embedding*.

Euler's formula for a 2-cell embedding of a graph into a compact surface relates four combinatorial quantities.

THEOREM 3.3.1 (Euler's Formula). *For any F-face 2-cell embedding of an N-node, E-edge graph into a compact genus-g surface,*

$$N - E + F = 2 - 2g$$

Assume that $N \geq 3$, so that $E \geq 2$ (because the graph is connected). Since every edge is twice incident to some face (perhaps the same face), and since every face is incident to at least three edges (though some may appear twice on the same face), we find that $3F \leq 2E$. For such graphs, therefore, Euler's formula implies that

$$E \leq 3N - 6 + 6g \tag{3.3.1}$$

A 2-cell embedding determines, for each $v \in \mathbf{N}$, a cyclic ordering (say, clockwise) of the edges incident to v; the ensemble of these cyclic orderings is called the *rotation* of the embedding. Conversely, any cyclic ordering of the edges incident to each $v \in \mathbf{N}$ determines (combinatorially) a 2-cell embedding of $\mathcal{G}$ in a surface of some genus. The rotation of a 2-cell embedding is a complete combinatorial representation of the embedding. To wit: each face of the embedding (more precisely, the sequence of edges bounding each face) is easily recovered from the rotation, and the genus of the surface of the embedding is recovered using Euler's formula. Importantly from our perspective, rotations form the ideal basis for data structures to represent both $\mathcal{G}$ and its embedding; simply represent $\mathcal{G}$ via adjacency lists, organizing the list for each node in the cyclic order of the rotation.

Any 2-cell embedding of a graph $\mathcal{G}$ into a compact 2-manifold determines a *dual graph* $\mathcal{G}^D$ for $\mathcal{G}$ as follows. The nodes of $\mathcal{G}^D$ are the faces of the embedding; for every edge $(x, y) \in \mathbf{E}$, $\mathcal{G}^D$ has a *dual edge* $(x, y)^D = \langle f_1, f_2 \rangle$ that connects the two faces f_1 and f_2 incident on (x, y) in the embedding. The graph $\mathcal{G}^D$ may not be simple if, in the embedding, some edge of $\mathcal{G}$ is incident on the same face twice or two edges of $\mathcal{G}$ are incident on the same two faces. (The computational difficulty arising with nonsimple dual graphs resides primarily in the need to extend the notion of a rotation appropriately.) If $\mathcal{G}$ and $\mathcal{G}^D$ are both simple graphs, then the dual of $\mathcal{G}^D$ is again $\mathcal{G}$. Clearly, $\mathcal{G}^D$ has an obvious embedding in the same surface as $\mathcal{G}$, and when $\mathcal{G}^D$ is a simple graph, this embedding is a 2-cell embedding. One verifies easily that one can use the previously mentioned representation of $\mathcal{G}$'s embedding into a surface $\mathcal{S}$ via ordered adjacency lists to construct both the dual graph $\mathcal{G}^D$ and its embedding into $\mathcal{S}$ in linear time.[2]

The most important special case of the notions discussed here resides in the family of *planar graphs*—graphs that are embeddable into the surface of genus 0, the sphere—and their *planar embeddings*. Every planar embedding of a connected graph is a 2-cell embedding, and Euler's bound on edge numbers (3.3.1) simplifies in the case of planar graphs to $E \leq 3N - 6$. By the *Jordan curve theorem*, any simple cycle in a planar graph determines a non-self-intersecting curve in the planar embedding, whose removal from the sphere leaves two surfaces, each homeomorphic to a disk. The two

surfaces may be thought of (arbitrarily) as the *inside* and the *outside* of the cycle. The cycle partitions the set of faces into *inside* and *outside faces* and partitions the set of those nodes that are not on the cycle into *inside nodes* and *outside nodes.*

3.3.2. Small Node-Separators for Planar Graphs

In this section, we present an algorithm that produces a small (1/3)-node-separator for a planar graph $\mathcal{G}$ from a planar embedding of $\mathcal{G}$.[3] To gain some intuition for the algorithm, consider the $m \times n$ rectangular mesh $\mathcal{M}_{m,n}$. Using the standard embedding of $\mathcal{M}_{m,n}$ [cf. Figure 1.3-2(b)], one can easily produce a (l/3)-node-separator of size $\min\{m, n\}$. We leave to the reader the not-so-easy exercise of showing that this size is best possible. (The proof appears in Section 4.2.4.) For the square grid $\mathcal{M}_{m,m}$, we easily find a (smallest) (l/3)-node-separator of size $m = \sqrt{N}$. The remainder of this section is devoted to showing that, to within a constant factor, we can do as well for *any* planar graph.

THEOREM 3.3.2. *Every N-node planar graph $\mathcal{G}$ has a* (l/3)*-node-separator of size* $2\sqrt{2}\sqrt{N} + l.o.t.$ *Moreover, one can find a node-separator of this size for $\mathcal{G}$ in linear time.*

We construct the separation algorithm that proves the theorem via a series of subsidiary algorithms. We begin by invoking a standard algorithmic device for planar graphs: we *triangulate* $\mathcal{G}$ by taking a planar embedding of $\mathcal{G}$ and adding edges that make every face a triangle while keeping the embedding planar. One sees that this is always possible as follows. Focus on any face in a planar embedding of $\mathcal{G}$ that is not a triangle. There must be distinct nodes u and v on the boundary of the face that are not adjacent in $\mathcal{G}$. If we add the edge (u, v) to the embedding, drawing the edge in the interior of the face, then we obtain a planar embedding of a supergraph of $\mathcal{G}$ that is "closer" to being triangulated than the embedding we started with. We can obviously repeat this edge-augmentation until we arrive at a triangulated embedding of a spanning supergraph of $\mathcal{G}$. This process takes only a linear number of augmentations because $\mathcal{G}$, being connected, must start out with at least $N - 1$ edges, and we know that a planar graph on N nodes can have no more than $3N - 6$ edges. Moreover, by replacing $\mathcal{G}$ with a triangulated supergraph, we can only increase the number of edges that must be cut when node-separating the graph. Therefore, we only strengthen the theorem if we assume henceforth that the graph $\mathcal{G}$ to be separated is a connected planar graph with a triangulated planar embedding.

If the triangulated planar embedding we start with has F faces, each face a triangle, then $3F = 2E$. This fact combines with Euler's formula to show that $F = 2N - 4$ for any triangulated planar graph.

Now, fix a node w of $\mathcal{G}$, and (in linear time) construct a breadth-first (hence, shortest-path) spanning tree $\mathcal{T}$ rooted at w; denote by L_k the set of *level-k* nodes of the tree, i.e., those nodes that are at distance k from w. Since $\mathcal{T}$ is a spanning tree of $\mathcal{G}$, any nontree edge $(x, y) \in \mathbf{E}(\mathcal{G})$ determines a unique cycle $C(x, y)$ consisting of (x, y) together with two or more tree edges. The length $|C(x, y)|$ of the cycle is at most $2t + 1$, where t is the height of $\mathcal{T}$. Any such cycle $C(x, y)$ separates the nodes not on the cycle into *inside nodes*, which are $\mathrm{In}(C(x, y))$ in number, and *outside nodes,* which are $\mathrm{Out}(C(x, y))$ in number. If both $\mathrm{In}(C(x, y))$ and $\mathrm{Out}(C(x, y))$ are at most $2N/3$, then the nodes of $C(x, y)$ constitute a (1/3)-node-separator of size at most $2t + 1$. We show now that such a (l/3)-node-separating cycle can always be found.

LEMMA 3.3.3. *Let $\mathcal{G}$ be a triangulated planar graph, and let $\mathcal{T}$ be a height-t breadth-first spanning tree of $\mathcal{G}$. In linear time one can find a nontree edge $(x, y) \in \mathbf{E}(\mathcal{G})$ such that the nodes of $C(x, y)$ constitute a (1/3)-node-separator of $\mathcal{G}$ size at most $2t + 1$.*

PROOF. Let $\mathcal{G}^D$ be the dual graph of the given planar embedding of $\mathcal{G}$. Since the lemma holds trivially when $N \leqslant 3$, we focus only on die case $N \geqslant 4$. This bound, coupled with the triangulation of the embedding, means that $\mathcal{G}^D$ is a simple graph that is regular of degree 3 and has $F = 2N - 4$ nodes. Let $\hat{\mathcal{G}}^D$ be obtained from $\mathcal{G}^D$ by deleting the edges that are dual to the edges of $\mathcal{T}$. Observe that $\hat{\mathcal{G}}^D$ is connected and acyclic and that it has maximum degree $\leqslant 3$. It follows that $\hat{\mathcal{G}}^D$ is a binary tree. Now convert $\hat{\mathcal{G}}^D$ to a *rooted* binary tree by choosing some (arbitrary) leaf f_r as the root. When we consider a cycle $C(x, y)$ that is formed by adding an edge to the tree, let us fix the designation of the inside and outside of the cycle by positing that f_r is an *outside* node. Focus now on any internal node f of the tree $\hat{\mathcal{G}}^D$ for which there exist nodes $x, y, z \in \mathbf{N}(\mathcal{G})$ such that (consult Figure 3.3-1 while reading this prescription)

- (x, y) is a nontree edge of $\mathcal{G}$.
- The edge that connects node f to its parent in $\hat{\mathcal{G}}^D$ is dual to edge $(x, y) \in \mathbf{E}(\mathcal{G})$.
- The edge that connects node f to one of its children in $\hat{\mathcal{G}}^D$ is dual to edge $(x, z) \in \mathbf{E}(\mathcal{G})$.
- If f has a second child in $\hat{\mathcal{G}}^D$, then the edge that connects node f to this child is dual to edge $(y, z) \in \mathbf{E}(\mathcal{G})$.

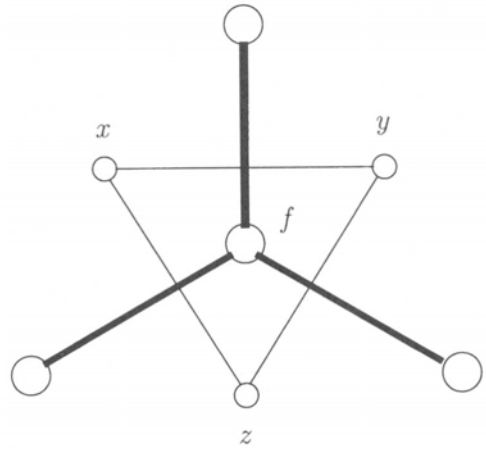


Figure 3.3-1. Internal node f of the dual of a spanning tree of a planar graph; edges of the dual tree are bold, while the nontree edges bounding f are thin. (This is the case wherein f has two children.)

We now assign the weight $\text{In}(C(x, y))$ to node f The reader can readily verify the following:

1. For the node f as described,
$$\text{Out}(C(x, y)) = N - (\text{In}(C(x, y)) + |C(x, y)|)$$
2. The weight of f satisfies the bound:
$$\text{In}(C(x, z)) + \text{In}(C(y, z)) \leqslant \text{In}(C(x, y))$$
(If edge (y, z) does not exist, then just set $|C(y, z)| = 0$.)
3. Either the weight of f_r is less than N/3 or there exists an internal node $f' \in \mathbf{N}(\hat{\mathcal{G}}^D)$ whose weight $\text{In}(C(x', y'))$ is no smaller than $N/3$, while neither of its children satisfies this bound. Such an f' can be found in linear time.
4. The weights of all nodes in $\hat{\mathcal{G}}^D$ can be computed in linear time.

If the weight of node f_r is less than $N/3$, then the single nontree edge[4] incident on f_r defines a cycle that can be taken as the required separator. Otherwise, let node f' be the node of $\hat{\mathcal{G}}^D$ whose weight $\text{In}(C(x', y'))$ is no smaller than $N/3$, while the weights $\text{In}(C(x', z'))$ and $\text{In}(C(y', z'))$ of its children are strictly less than $N/3$. If $\text{In}(C(x', y')) \leqslant 2N/3$, then $C(x', y')$ is the desired separator. Otherwise, we must have $\text{Out}(C(x', y')) < N/3 - |C(x', y')|$.

Now, the nodes inside cycle $C(x', y')$ come from the following three disjoint sets:

1. The nodes inside cycle $C(x', z')$
2. The nodes inside cycle $C(y', z')$
3. The nodes on cycle $C(x', z')$ that are not on cycle $C(x', y')$ (or, equivalently, the nodes on $C(y', z')$ that are not on $C(x', y')$)

Similarly, the set of nodes outside $C(x', z')$ come from the following three disjoint sets:

1. The nodes outside cycle $C(x', y')$
2. The nodes inside cycle $C(y', z')$
3. The nodes on cycle $C(x', y')$ that are not on cycle $C(x', z')$

Easily, the cardinality of this latter set satisfies

$$\begin{aligned} \text{Out}(C(x', z')) &\leqslant \text{Out}(C(x', y')) + \text{In}(C(y', z')) + |C(x', y')| \\ &< N/3 - |C(x' y')| + N/3 + |C(x', y')| \\ &< 2/N/3 \end{aligned}$$

We see immediately that cycle $C(x', z')$ is the desired separator. The lemma follows. □

If the depth t of the breadth-first tree rooted at w is no larger than $\sqrt{2}\sqrt{N} - 1$, then Theorem 3.3.2 is an immediate consequence of Lemma 3.3.3. If the depth t is too large, then we must look at the tree's $t + 1$ levels to find the needed separator. In particular, we seek level-indices i_1 and $i_2 > i_1$ whose "spanned" levels, $L_{i_1}, L_{i_1+1}, \ldots, L_{i_2}$, contain an appropriate fraction of $\mathcal{G}$'s nodes. We will then be able to apply Lemma 3.3.3 to these levels to obtain the desired separator. The precise construction follows.

Choose $k \in Z_{t+1}$ such that

$$|L_1 \cup L_2 \cup \cdots \cup L_{k-1}| < \tfrac{1}{2}N$$

and

$$R \stackrel{\text{def}}{=} |L_1 \cup L_2 \cup \cdots \cup L_k| \geqslant \tfrac{1}{2}N$$

(It should be obvious that such a k exists.) If $|L_k| \leqslant 2\sqrt{2}\sqrt{N}$, then the theorem holds with L_k serving as the desired (1/3)-node separator, and we are done. Assume, therefore, that $|L_k| > 2\sqrt{2}\sqrt{N}$.

Choose a level-index $i_1 \in Z_{k+1}$ for which $|L_{i_1}| \leqslant 2\sqrt{R} - 2(k - i_1)$. We know that such an i_1 exists by the following reasoning. Say, for contradiction, that $|L_j| > 2\sqrt{R} - 2(k - j)$ for all $j \in Z_{k+1}$. Since $|L_0| = 1$, it follows that $2\sqrt{R} - 2k < 1$, so that $\lfloor \sqrt{R} \rfloor \leqslant k$. Therefore, if we sum the sizes of the levels of interest, we obtain

$$\begin{aligned} R &= \sum_{j=0}^{k} |L_j| \\ &> \sum_{j=k-\lfloor \sqrt{R} \rfloor}^{k} 2\sqrt{R} - 2(k-j) \\ &= (2\sqrt{R} - \lfloor \sqrt{R} \rfloor)(\lfloor \sqrt{R} \rfloor + 1) \\ &\geqslant \sqrt{R}(\lfloor \sqrt{R} \rfloor + 1) \end{aligned}$$

Since the last quantity in this chain exceeds R, we reach a contradiction that allows us to conclude that a suitable i_1 exists.

Now, for notational convenience, let us add a new, empty, level $L_{t+1} = \varnothing$ to the tree.

Let $i_2 \in \{k+1, k+2, \ldots, t+1\}$ be a level-index for which $|L_{i_2}| \leqslant 2\sqrt{N - R} - 2(i_2 - k)$. By an argument analogous to the one for i_1, such an i_2 must exist.

We now try using $L_{i_1} \cup L_{i_2}$ as a separator. By the bounds defining i_1 an i_2, we have

$$\begin{aligned} |L_{i_1} \cup L_{i_2}| &\leqslant 2\sqrt{R} - 2(k - i_1) + 2\sqrt{N - R} - 2(i_2 - k) \\ &= 2(\sqrt{R} + \sqrt{N - R}) + 2(i_1 - i_2) \qquad (3.3.2) \\ &\leqslant 2\sqrt{2}\sqrt{N} + 2(i_1 - i_2) \end{aligned}$$

the last inequality following by maximizing $\sqrt{R} + \sqrt{N - R}$ over the range $R \geqslant N/2$.

In view of (3.3.2), if $L_{i_2} \cup L_{i_2}$ is actually a (1/3)-node-separator of $\mathcal{G}$, then it satisfies the theorem and we are done. Assume, therefore, that a connected component of size $>2N/3$ remains when we remove $L_{i_1} \cup L_{i_2}$ from $\mathcal{G}$. By the choice of i_1 and i_2, this large component must lie completely between levels i_1 and i_2; i.e., we must have

$$|L_{i_1+1} \cup \cdots \cup L_{i_2-1}| > 2N/3$$

In this case we apply Lemma 3.3.3 to the breadth-first tree to find a cycle $C(x, y)$ whose nodes constitute a (l/3)-node-separator of $\mathcal{G}$. Let M be the set of nodes in $C(x, y)$ that occur strictly between levels i_1 and i_2 of the tree. Since $C(x, y)$ (obviously) contains at most two nodes in any level of the tree, we must have $|M| < 2(i_2 - i_1)$. Now, the set of nodes $L_{i_1} \cup L_{i_2} \cup M$ clearly constitutes a (l/3)-node-separator of $\mathcal{G}$ that contains

$$\begin{aligned} |L_{i_1} \cup L_{i_2} \cup M| &\leqslant 2\sqrt{2}\sqrt{R} + 2(i_1 - i_2) + |M| \\ &< 2\sqrt{2}\sqrt{R} \\ &\leqslant 2\sqrt{2}\sqrt{N} \end{aligned}$$

nodes. We finally have established the existence of the desired separator.

To complete the proof of Theorem 3.3.2, we need only consider how much time it takes to find the desired separator. To this end, we collect the steps of the separator algorithm in Figure 3.3-2 and note that each can be accomplished in linear time. □

3.3.3. Small Node-Separators for Genus-g Graphs

As is presaged by the fact that every graph $\mathcal{G}$ has a 2-cell embedding in the surface of its genus g, we can generalize the construction of the previous section to obtain a separator theorem for graphs of any positive genus g.

THEOREM 3.3.4. *For any fixed $g > 0$, every N-node graph $\mathcal{G}$ of genus g admits a (1/3)-node-separator of size $6\sqrt{gN} + 2\sqrt{N/g}$ + l.o.t. Moreover, such a node-separator can be found in linear time.*

The major work in proving Theorem 3.3.4 is to generalize Lemma 3.3.3 to include the nonplanar case. As before, we assume that every graph is given with an embedding in its genus surface and that the embedding has been triangulated, in the sense that edges have been added to make every face a triangle.

LEMMA 3.3.5. *Let $\mathcal{G}$ be a triangulated graph of genus $g \geqslant 0$, and let $\mathcal{T}$ be a depth-t spanning tree of $\mathcal{G}$. Then there are $2g + 1$ nontree edges (x_j, y_j), where $j \in \{1,2,\ldots, 2g + 1\}$, such that the combined nodes of all the associated cycles $C(x_j, y_j)$ constitute a (1/3)-node-separator of $\mathcal{G}$ of size $\leqslant (2g + 1)2t + 1)$. These cycles can be found in time $O(E + gt)$.*

Algorithm PLANAR-SEPARATOR($\mathcal{G}$)

1. Embed $\mathcal{G}$ in the plane using any linear-time algorithm.
2. Choose a node w, and construct a breadth-first spanning tree $\mathcal{T}$ rooted at w. If $\mathcal{T}$ has levels $L_0, L_1, \ldots, L_t$, then add an empty "dummy" level L_{t+1}.

▷ The levels of $\mathcal{T}$ partition $\mathbf{N}(\mathcal{G})$ according to distance from w.

3. Find k such that $\left|\bigcup_{i=0}^{k-1} L_i\right| < \frac{N}{2} < \left|\bigcup_{i=0}^{k} L_i\right|$.

   Let $R = \left|\bigcup_{i=0}^{k} L_i\right|$.

4. Find i_1 and i_2 in the range $0 \leqslant i_1 \leqslant k < i_2 \leqslant t$ such that

   (a) $|L_{i_1}| + 2(k - i_1) \leqslant 2\sqrt{R}$

   (b) $|L_{i_2}| + 2(i_2 - k) \leqslant 2\sqrt{N - R}$.

▷ Use Lemma 3.3.3 to find a separator.

5. Find a cycle $C(x, y)$ consisting of edges of $\mathcal{T}$ and one nontree edge, (x,y), whose nodes constitute a (l/3)-node separator of $\mathcal{G}$.
6. Let M be the nodes in $C(x, y)$ in levels i_1 through i_2.
7. Output "$L_{i_1} \cup L_{i_2} \cup M$ is the desired separator."

Figure 3.3-2. The planar separator algorithm.

PROOF. The proof directly generalizes that of Lemma 3.3.3. Let $\mathcal{G}^D$ be the dual graph of the embedding of $\mathcal{G}$ in a surface of genus g. Since the lemma clearly holds if $N \leqslant 3$, we may concentrate on the case $N > 4$. In this case $\mathcal{G}^D$ is a simple graph[5] that is regular of degree 3 and has $F = 2N + 4g - 4$ nodes. Let $\hat{\mathcal{G}}^D$ be $\mathcal{G}^D$ with the edges dual to the edges of $\mathcal{T}$ deleted. Observe that $\hat{\mathcal{G}}^D$ is connected and has maximum degree $\leqslant 3$ but that it is not acyclic if $g > 0$. Using Euler's formula and the fact that the embedding of $\mathcal{G}$ is triangulated, we find that $E = 3N + 6g - 6$. Since $\hat{\mathcal{G}}^D$ has $N - 1$ fewer edges than $\mathcal{G}$, namely, $\hat{E} = 2N + 2g - 5 = F + 2g - 1$ edges, there are $2g$ edges of $\hat{\mathcal{G}}^D$ whose removal will convert the graph into a forest of binary trees; we denote the $2g$ dual edges of these nontree edges (x_j, y_j), for $j \in \{1, 2, \ldots, 2g\}$. These dual edges are nontree edges in $\mathcal{G}$.

Now, we wish to select these $2g$ edges so that the deletion of the $2g$ cycles $C(x_j, y_j)$ from $\mathcal{G}$ leaves a planar graph. We sketch the ideas needed for this selection. A cycle in $\mathcal{G}$ corresponds to a circle drawn on the surface that $\mathcal{G}$ is embedded into. Removing the cycle from $\mathcal{G}$ corresponds to cutting the surface along the circle. The cut can be "repaired" by taking two disks and patching the two holes; in addition, $\mathcal{G}$ may be retriangulated by adding some edges within the two disks. One of two results is obtained after this cutting-*cum*-patching. In one scenario, the cut will separate the surface into two surfaces the sum of whose genera (the plural of "genus") equals the genus of the original. In this case, therefore, removing the cycle partitions $\mathcal{G}$ into disjoint subgraphs of smaller genera. In the other scenario, which will always occur when we remove a cycle corresponding to a nonplanar edge, the cut will eliminate a "handle" from the surface, thus reducing the genus of the surface by 1. In this case, therefore, removing the cycle reduces the genus of $\mathcal{G}$. Now, on the one hand, one can cut the surface into two smaller surfaces, each of positive genus, at most $g - 1$ times; on the other hand, since the surface had genus g to start with, one can cut no more than g handles from the surface. It follows that removing the $2g$ cycles will leave us with a planar graph, as long as we can avoid cutting off a surface of genus 0. One finds in Heath and Istrail [1992] a technique for detecting *nonplanar edges*, which, as noted, will allow us to avoid the latter contingency.

If our removal of the $2g$ cycles leaves no component having more than $2N/3$ nodes, then we can choose the $(2g + 1)$th edge, (x_{2g+1}, y_{2g+1}), of the lemma at will. Otherwise, we apply the remainder of the proof of Lemma 3.3.3 to the component having more than 2N/3 nodes, thereby obtaining one more edge (x_{2g+1}, y_{2g+1}). The collection of cycles $C(x_j, y_j)$, where $j \in \{1, 2, \ldots, 2g + 1\}$, is now a (l/3)-node-separator of $\mathcal{G}$; moreover, as each cycle contains at most $2t + 1$ nodes, the stated bound on separator-size follows.

To demonstrate the stated time complexity, it suffices to show how to find the $O(t)$ nodes for each cycle $C(x_j, y_j)$ in time $O(t)$. To this end, focus on a fixed j between 1 and $2g + 1$. Let Z_j be the least common ancestor of x_j and y_j in $\mathcal{T}$, and let d_x (resp., d_y, d_z) be the depth of x_j (resp., y_j, z_j) in $\mathcal{T}$. Assume, with no loss of generality, that $d_z < d_x \leqslant d_y$. Starting at z_j, cycle $C(x_j, y_j)$ proceeds down the tree $d_x - d_z$ edges to x_j, crosses edge (x_j, y_j) to y_j, and then proceeds up the tree $d_x - d_z$, edges back to z_j. A bit of reflection on Figure 3.3-3 reveals that the invocation CYCLE-FINDING $(\mathcal{G}, \mathcal{T}, x_j, y_j)$ locates z_j and returns the nodes of $C(x_j, y_j)$. The $O(t)$ time complexity follows from the observation that we may assume that the depth and parent of each node in $\mathcal{T}$ are part of the representation of $\mathcal{T}$. □

Algorithm CYCLE-FINDING($\mathcal{G}$, $\mathcal{T}$, x, y)

▷ Find the cycle $C(x, y)$ in graph $\mathcal{G}$, via spanning tree $\mathcal{T}$.

▷ Let the representation of $\mathcal{T}$ include arrays d and P:

▷ $d(z)$ gives the depth of node z;

▷ $P(z)$ gives the parent of node z.

1. $C \leftarrow \{x, y\}$
2. if $d(y) < d(x)$
   then Switch the roles of x and y.
3. $u \leftarrow P(x)$; $v \leftarrow P(y)$
4. $C \leftarrow C \cup \{u, v\}$

▷ Proceed up $\mathcal{T}$ from y to its ancestor at depth $d(x) - 1$.

5. **for** $d \leftarrow d(y) - 1$ **downto** $d(x)$
   do $v \leftarrow P(v)$; $C \leftarrow C \cup \{u, v\}$

▷ Proceed up $\mathcal{T}$ from u and v to the least common ancestor of x and y.

6. **while** $u \neq v$
   do $u \leftarrow P(u)$; $v \leftarrow P(v)$; $C \leftarrow C \cup \{u, v\}$

▷ C contains the nodes in $C(x, y)$.

7. Return C.

Figure 3.3-3. Cycle-finding algorithm.

Continuing with the proof of Theorem 3.3.4, we once again construct a breadth-first tree $\mathcal{T}$ rooted at some node w of $\mathcal{G}$. Note that $\mathcal{T}$'s levels, $L_0, L_1, \ldots, L_t$ partition $\mathbf{N}$ by distance from w, and add an extra level $L_{t+1} = \varnothing$ for convenience. We then choose a level-index $k \in Z_{t+1}$ such that

$$|L_1 \cup L_2 \cup \cdots \cup L_{k-1}| < \tfrac{1}{2}N$$

and

$$|L_1 \cup L_2 \cup \cdots \cup L_k| \geq \tfrac{1}{2}N$$

If $|L_k| \leqslant \sqrt{gN}$, then L_k can serve as the (1/3)-node separator of the theorem. Otherwise, we choose i_1 and i_2 as follows. Choose $i_1 \in Z_k$ to be the largest level-index such that $|L_{i_1}| \leqslant \sqrt{gN}$. Similarly, choose $i_2 \in Z_{t+2} - Z_{k+1}$ to be the smallest level-index such that $|L_{i_2}| \leqslant \sqrt{gN}$. Note that $i_2 - i_1 \leqslant \sqrt{N/g}$. Now, if $L_{i_1} \cup L_{i_2}$ is a (1/3)-node separator of $\mathcal{G}$, then it is the separator we are seeking. Otherwise, there must be more than $N/3$ nodes in the levels between i_1 and i_2. In this case we invoke Lemma 3.3.5 to obtain $2g + 1$ cycles $C(x_j, y_j)$, for $j \in \{1, 2, \ldots, 2g + 1\}$, whose nodes constitute a (1/3)-node separator of $\mathcal{G}$. Let M be the set of those nodes in levels strictly between i_1 and i_2. By construction,

$$|M| \leqslant (2g + 1)(2\sqrt{N/g}) = 4\sqrt{gN} + 2\sqrt{N/g}$$

We then see that the set $L_1 \cup L_2 \cup M$ is a (1/3)-node separator of $\mathcal{G}$, of size no greater than

$$2\sqrt{gN} + 4\sqrt{gN} + 2\sqrt{N/g} = 6\sqrt{gN} + 2\sqrt{N/g}$$

which is the bound stated in the theorem.

We leave the necessary adaptations of Figure 3.3-2 to the genus-g case to the reader.

It remains to establish the linear time-complexity of the described algorithm. The only step in the algorithm that is not obviously linear time is the invocation of Lemma 3.3.5, which has time-complexity $O(E + T)$, where T is the time we may have to expend in order to find the $2g + 1$ cycles that eventually yield the separator. To assess the magnitude of T, recall that we find these cycles via a single up-down sweep of the depth-$O(t)$ spanning tree $\mathcal{T}$. Moreover, to obtain the separator, we need find only the portions of the cycles that lie between levels i_1 and i_2 of $\mathcal{T}$. Since $i_2 - i_1 \leqslant \sqrt{N/g}$, the time T to find all of the cycles is proportional to $g\sqrt{N/g} = \sqrt{Ng}$. We are almost done. We now invoke the fact—whose verification we leave to the reader—that adding a single edge to any noncomplete graph can increase the graph's genus by at most 1, to infer that $g = O(E)$. This tells us that $T = O(E)$; i.e., our algorithm operates in linear time.

3.4. Geometric Approaches to Graph Separation

For any fixed $d \geqslant 2$, let R^d denote d-dimensional Euclidean space endowed with a Cartesian coordinate system. We can embed a graph $\mathcal{G}$ into R^d by mapping each node $v \in \mathbf{N}(\mathcal{G})$ to a point $f(v) \in R^d$ and each edge

$(u, v) \in \mathbf{E}(\mathcal{G})$ to the line segment between $f(u)$ and $f(v)$. If $d \geq 3$, then we can always embed $\mathcal{G}$ *properly*, in the sense that the node-mapping f is one-to-one and no two line segments (edges) interesect except perhaps at a shared endpoint. In fact, with probability 1, a *random* embedding, in which the node-mapping f is selected according to any reasonable continuous distribution, is proper. Henceforth, we assume that we are presented the graph $\mathcal{G}$ via a proper embedding of $\mathcal{G}$ into R^d, where $d \geq 3$, and we consider the N image-points of $\mathbf{N}(\mathcal{G})$, call them $p_1, p_2, \ldots, p_N$, as the nodes of $\mathcal{G}$. This mode of presentation endows $\mathcal{G}$ with geometric properties to accompany its combinatorial properties. As we already have a separator algorithm for planar graphs (the case $d = 2$) that builds on proper embeddings (also known as drawings) of such graphs into R^2, the results of this section supplement, rather than displace, the results of Section 3.3.

The motivation for looking at a graph $\mathcal{G}$ via an embedding into R^d is the hope of exploiting some geometric property of $\mathcal{G}$ (actually, of its embedding) to obtain a good separator algorithm. Various authors have accomplished just this, by restricting attention to classes of graphs that enjoy some nice geometric property. In this section we derive a separator algorithm for one particular such class, based on the *density* of a graph's embedding into R^d. This focus notwithstanding, the outline of the development here applies to all the other known results for geometric separators. In Section 3.4.1 we define the density of (an embedding of) a graph and present the geometric preliminaries that our development builds on. The remaining three subsections present the three general steps one uses to derive a geometric separator for a graph: Section 3.4.2 constructs density functions derived from the given embedding of $\mathcal{G}$; Section 3.4.3 explains how to find a hyperplane in R^d that separates $\mathbf{N}(\mathcal{G})$ into balanced parts whose density functions have a small average value; Section 3.4.4 shows that this separating hyperplane leads to a small node-separator for $\mathcal{G}$.

3.4.1. Geometric Preliminaries

Let $\|p\|$ denote the Euclidean norm, $(p_1^2 + p_2^2 + \cdots + p_d^2)^{1/2}$, of point $p = (p_1, p_2, \ldots, p_d) \in R^d$. For any point $p \in R^d$ and positive integer r, the *radius-r d-dimensional ball* $B^d(p, r)$ *centered at* p is the locus of all points $q \in R^d$ such that $\|p - q\| \leq r$. The *boundary* of $B^d(p, r)$ is the *radius-r* $(d - 1)$*-dimensional sphere* $S^{d-1}(p, r)$ *centered at* p comprising all points $q \in R^d$ such that $\|p - q\| = r$. The boundary-sphere $S^{d-1}(p, r)$ partitions R^d into three subsets:

1. The boundary-sphere $S^{d-1}(p, r) = \{q \in R^d : \|p - q\| = r\}$ itself
2. The *interior* $\{q \in R^d : \|p - q\| < r\}$ of $S^{d-1}(p, r)$

3. The *exterior* $\{q \in R^d : \|p - q\| > r\}$ of $S^{d-1}(p, r)$

The *volume* of the ball $B^d(p, r)$ is given by the formula

$$\text{Vol}(B^d(p, r)) = \frac{\pi^{d/2} r^d}{\Gamma(d/2 + 1)}$$

where Γ is the classic Gamma function. The *surface area* of the boundary-sphere $S^{d-1}(p, r)$ is given by the formula

$$\text{SA}(S^{d-1}(p, r)) = \frac{2\pi^{d/2} r^{d-1}}{\Gamma(d/2)}$$

A notational aside. While there is a sharp, and obvious, distinction between a *point* $x = (x_1, x_2, \ldots, x_d) \in R^d$ and the associated *vector* $\vec{x}$, which one can view as the line segment directed from the origin $\vec{0}$ of R^d to point x, it is customary to refer to both entities via the ambiguous notation x, allowing the text and the context to steer the reader toward the intended entity.

A *hyperplane* H in R^d is a $(d - 1)$-dimensional (affine) subspace. For *vectors* $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$ in R^d, let the notation $x \perp y$ denote the fact that x and y are *perpendicular*, i.e., that $\sum_{i=1}^{d} x_i y_i = 0$. The *oriented* hyperplane $H(p, x)$ is determined by any point p in the hyperplane, together with a vector $x \in R^d - \{\vec{0}\}$ that is *normal* to the hyperplane; in symbols,

$$H(p, x) = \{y \in R^d : x \perp (y - p)\}$$

Every oriented hyperplane $H(p, x)$ partitions R^d into three subsets:

1. The hyperplane $H(p, x)$ itself
2. The *open halfspace* $H_+(p, x)$ on the side of $H(p, x)$ that contains point $p + x$
3. The *open halfspace* $H_-(p, x)$ on the side of $H(p, x)$ that contains point $p - x$

A *centerpoint* p for the (embedded) graph $\mathcal{G}$ is defined by the property that, for every $x \in R^d - \{\vec{0}\}$, each of the open halfspaces, $H_+(p, x)$ and $H_-(p, x)$, contains at most the fraction $dN/(d + 1)$ of the points of $\mathbf{N}(\mathcal{G})$. Using Helly's theorem (cf. Edelsbrunner [1987]), one can show the following.

LEMMA 3.4.1. *Every finite set of points in R^d has a centerpoint.*

The fact that centerpoints always exist gives us hope that we can always find a $(1/(d+1))$-node-separator of a graph that is properly embedded into R^d. We show in the rest of this section that this hope can be realized. We begin by defining the geometric concept that will enable us to compute the centerpoints that will yield our separators.

The *eccentricity* $ECC(p_i)$ of a point $p_i \in \mathbf{N}(\mathcal{G})$ is the ratio of p_i's distance from its *furthest neighbor* in $\mathbf{N}(\mathcal{G})$ to its distance from the *nearest node* in $\mathbf{N}(\mathcal{G}) - \{p_i\}$; symbolically,

$$\mathrm{ECC}(p_i) = \frac{\max_{(p_i,p_j)\in \mathbf{E}(\mathcal{G})} \|p_j - p_i\|}{\min_{k\neq i} \|p_k - p_i\|}$$

Clearly, $\mathrm{ECC}(p_i) \geq 1$ unless p_i is an isolated node. The *density of the graph* $\mathcal{G}$ is the maximum eccentricity of any of its nodes. More specifically, $\mathcal{G}$ is an *α-density graph*, where $\alpha \geq 1$, if its density is *no greater than* α.

> Clearly, the density of a graph $\mathcal{G}$ depends on the embedding used to present $\mathcal{G}$. Since this embedding is fixed throughout this section, we never explicitly acknowledge this dependence, but the reader should keep it in mind when applying the techniques of the section.

3.4.2. Density Functions

We now define some integrable functions on R^d that, informally, measure the "local" density of $\mathcal{G}$ in neighborhoods of its nodes. The analogy to probability density functions is apt, as will become clear. Let D_i be the distance between $p_i \in \mathbf{N}(\mathcal{G})$ and its furthest neighbor-node:

$$D_i = \max_{(p_i,p_j)\in \mathbf{E}(\mathcal{G})} \|p_j - p_i\|$$

Our first density functions involve just individual nodes; for each $p_i \in \mathbf{N}(\mathcal{G})$ define

$$f_i(x) = \begin{cases} 1/D_i & \text{if } x \in B^d(p_i, D_i) \\ 0 & \text{otherwise} \end{cases}$$

Note that the volume integral[6]

$$\int_{R^d} f_i^d(x)(\mathrm{d}x)^d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$$

is the volume of a d-dimensional ball of radius 1.

Of particular interest here are the maximum and average local densities; therefore, the remainder of this section is devoted to proving bounds on the following natural combinations of local density functions:

$$f(x) = \max_{1 \leq i \leq N} f_i(x) \qquad \text{and} \qquad g(x) = \sum_{i=1}^{N} f_i(x) \tag{3.4.1}$$

LEMMA 3.4.2. *If $\mathcal{G}$ is an α-density graph, then, letting f and g be as in* (3.4.1), *the following bounds hold*:

(a) $\int_{x \in R^d} f^d(x)(\mathrm{d}x)^d \leq N \cdot \mathrm{Vol}(B^d(0, 1))$.
(b) $f(x) \leq g(x) \leq 2(6\alpha)^d f(x)$, *for any* $x \in R^d$.
(c) $\int_{x \in R^d} g^d(x)(\mathrm{d}x)^d = O(\alpha^{d^2} N)$.

PROOF, (a) By direct calculation,

$$\begin{aligned}\int_{x \in R^d} f^d(x)(\mathrm{d}x)^d &\leq \sum_{i=1}^{N} \int_{x \in R^d} f_i^d(x)(\mathrm{d}x)^d \\ &= \sum_{i=1}^{N} \int_{x \in B^d(p_i, D_i)} (1/D_i)^d (\mathrm{d}x)^d \\ &= N \cdot \mathrm{Vol}(B^d(0, 1))\end{aligned}$$

(b) From their definitions, it is clear that $f(x) \leq g(x)$. To prove the upper bound on $g(x)$, focus on any x such that $g(x) > 0$ (note that $f(x) > 0$ also). Choose a point p_k for which $f_k(x) = f(x)$. Invoking the bound on eccentricity, for any p_i, we have

$$\alpha \geq \mathrm{ECC}(p_i) = \frac{D_i}{\min_{k \neq i} \|p_k - p_i\|}$$

We conclude from this inequality that the distance between any distinct points p_s and p_t satisfies

$$\|p_s - p_t\| \geq \frac{\max\{D_s, D_t\}}{\alpha}$$

This suggests that there is an upper bound on the number of points with a particular D value that are close enough to x to influence the value of $g(x)$.

More concretely, for $j \geq 1$, let N_j be the number of points p_s satisfying

$$2^{j-1} D_k \leq D_s < 2^j D_k \qquad \text{and} \qquad \|p_s - x\| \leq D_s$$

Any two such points, p_s and p_t, are within distance $2^j D_k$ of x and satisfy $\|p_s - p_t\| \geq (2^{j-1} D_k)/\alpha$. Therefore, the ball $B^d(x, (2^j + 2^{j-1})D_k)$ contains all these points; moreover, within this ball are N_j balls of radius $(2^{j-2} D_k)/\alpha$ centered at the N_j points that share no volume. This implies the inequalities

$$N_j \cdot \text{Vol}\left(B^d\left(0, \frac{2^{j-2} D_k}{\alpha}\right)\right) \leq \text{Vol}(B(x, (2^j + 2^{j-1})D_k))$$

and

$$N_j \leq \left(\frac{\alpha(2^j + 2^{j-1})}{2^{j-2}}\right)^d = (6\alpha)^d$$

Thus, the contribution of these N_j points to $g(x)$ is no greater than $(6\alpha)^d/(2^{j-1} D_k)$. Since every point that contributes to $g(x)$ is counted in exactly one of the N_j, the final bound on $g(x)$ is

$$g(x) \leq \sum_{j=1}^{\infty} \frac{(6\alpha)^d}{2^{j-1} D_k} = 2(6\alpha)^d f(x)$$

as was claimed.

(c) The bound of part (c) is an immediate consequence of parts (a) and (b). □

3.4.3. Finding a Separating Hyperplane

In this section we show how to find a separating hyperplane that contains little of the density of $\mathcal{G}$.

THEOREM 3.4.3. *If $\mathcal{G}$ is an* **α-density** *graph in R^d, then there exists a hyperplane $H(p, z)$ such that*

(a) *Each of $H_+(p, z)$ and $H_-(p, z)$ contains no more than $dN/(d + 1)$ points of $\mathbf{N}(\mathcal{G})$.*
(b) $\int_{w \in H(p,z)} g^{d-1}(w)(dw)^{d-1} = O(\alpha^{d(d-1)} N^{(d-1)/d})$.

PROOF. Invoking Lemma 3.4.1, we select p to be a centerpoint of $\mathbf{N}(\mathcal{G})$. By definition of centerpoint, any hyperplane that contains p satisfies (a). It thus remains only to show how to select z so that (b) holds.

Because the statement of the theorem is unchanged by a translation of every point in R^d by $-p$, we may assume, with no loss of generality, that $p = 0$. We define the uniform probability distribution for the points $x \in S^{d-1}(0, 1)$ and, hence, for the oriented hyperplanes $H(Q, x)$ through the

origin, by the following constant probability density function:

$$q(x) = \frac{1}{\mathrm{SA}(S^{d-1}(0,1))}$$

Let F be any nonnegative integrable function defined on R^d. The expected value of the integral of F over a random oriented hyperplane $H(0, x)$ is

$$\mathrm{E}(F) = \int_{x \in S^{d-1}(0,1)} \left(\int_{w \perp x} F(w)(\mathrm{d}w)^{d-1} \right) q(x)(\mathrm{d}x)^{d-1}$$

provided that this value is finite. We compute

$$\begin{aligned} \mathrm{E}(F) &= \int_{w \in R^d} F(w) \int_{\substack{x \in S^{d-1}(0,1) \\ x \perp w}} q(x)(\mathrm{d}x)^{d-2}(\mathrm{d}w)^d \\ &= \frac{\mathrm{SA}(S^{d-2}(0,1))}{\mathrm{SA}(S^{d-1}(0,1))} \int_{w \in R^d} F(w)(\mathrm{d}w)^d \end{aligned}$$

We used here the fact that if $w \in R^d - \{\vec{0}\}$, then the set of points $x \in S^{d-1}(0, 1)$ satisfying $x \perp w$ is a $(d - 2)$-dimensional sphere. In particular, when F is g^{d-1}, we have

$$\mathrm{E}(g^{d-1}) = \frac{\mathrm{SA}(S^{d-2}(0,1))}{\mathrm{SA}(S^{d-1}(0,1))} \int_{w \in R^d} g^{d-1}(w)(\mathrm{d}w)^d$$

Since $g(x) = 0$ for points x that are sufficiently far from the origin, we can select a radius $r > 0$ such that $g(x) > 0$ only when $\|x\| \leqslant r$. We then finally find that

$$\mathrm{E}(g^{d-1}) = \frac{\mathrm{SA}(S^{d-2}(0,1))}{\mathrm{SA}(S^{d-1}(0,1))} \int_{w \in B^d(0,r)} g^{d-1}(w)(\mathrm{d}w)^d \tag{3.4.2}$$

A special case of *Hölder's inequality for integrals* (cf. Hardy *et al.* [1952]) is the following:

For *any* $0 < \beta < 1$, *if* h_1 *and* h_2 *are integrable functions, then*

$$\int h_1^{\beta} h_2^{1-\beta} \leqslant \left(\int h_1 \right)^{\beta} \left(\int h_2 \right)^{1-\beta}$$

If we now apply Hölder's inequality to the expression (3.4.2) for $E(g^{d-1})$, with $\beta=(d-1)/d$, $h_1(x) = g^d(x)$, and $h_2(x) = 1$, we find that

$$E(g^{d-1}) \leqslant \frac{SA(S^{d-2}(0,1))}{SA(S^{d-1}(0,1))}\left(\int_{w\in B^d(0,r)} g^d(w)(dw)^d\right)^{(d-1)/d}\left(\int_{w\in B^d(0,r)} 1(dw)^d\right)^{1/d}$$
$$= \frac{SA(S^{d-2}(0,1))}{SA(S^{d-1}(0,1))}\left(\int_{w\in B^d(0,r)} g^d(w)(dw)^d\right)^{(d-1)/d} \cdot Vol(B^d(0,1))^{1/d}$$

From the bound of Lemma 3.4.2(c), we then obtain

$$E(g^{d-1}) = O((\alpha^{d^2}N)^{(d-1)/d})$$
$$= O(\alpha^{d(d-1)}N^{(d-1)/d})$$

If we now choose $z\in S(0,1)$ so that $\int_{w\in H(0,z)} g^{d-1}(dw)^{d-1} \leqslant E(g^{d-1})$, then, by our last bound on $E(g^{d-1})$, we obtain

$$\int_{w\in H(p,z)} g^{d-1}(w)(dw)^{d-1} = O(\alpha^{d(d-1)}N^{(d-1)/d})$$

as required. □

It is straightforward to devise a randomized algorithm that finds the separating hyperplane promised by Theorem 3.4.3 rather efficiently. First, find a centerpoint p for $\mathbf{N}(\mathcal{G})$; we provide pointers in Section 3.7 to algorithms that accomplish this. Next, select a vector $z\in S(0,1)$ at random. Since the probability that the hyperplane $H(p, z)$ satisfies the bound in the theorem is at least 1/2, the probability that one has not found a suitable hyperplane after a linear number of random selections is exponentially small.

3.4.4. Separating an α-Density Graph

We now have the machinery to state and prove the separation theorem for α-density graphs.

THEOREM 3.4.4. *Let $\alpha \geqslant 1$ and $d \geqslant 3$ be fixed. Every N-node graph that admits an α-density embedding into R^d has a $(1/(d+1))$-node-separator of size $O(\alpha^{d(d-1)}N^{(d-1)/d})$. Moreover, one can find such a separator in randomized polynomial time.*

PROOF. Let $p \in R^d$ and $z \in S(0, 1)$ be any points for which the oriented hyperplane $H(p, z)$ satisfies the conclusion of Theorem 3.4.3. Without loss of generality, we may assume that no point of $\mathbf{N}(\mathcal{G})$ lies on $H(p, z)$, as the probability of this (non)event is 0. Consider now the set $M \subset \mathbf{N}(\mathcal{G})$ constructed via the following rule.

> *For every edge* $(p_i, p_j) \in \mathbf{E}(\mathcal{G})$ *that crosses* $H(p, z)$, *place whichever of* p_i *and* P_j *is closer to* $H(p, z)$ *into* M.

(Since the probability that p_i and p_j are equidistant from $H(p, z)$ is 0, we just assume that such a tie cannot happen.) Now, by Theorem 3.4.3, the set M is a $(1/(d + 1))$-node-separator. We complete the proof by showing that the size of M is $O(\alpha^{d(d-1)} N^{(d-1)/d})$.

Let node p_i be placed into M by dint of edge (p_i, p_j), which crosses $H(p, z)$. Then $\|p_i - p_j\| \leq D_i$, and (by our rule for constructing M) the distance from p_i to $H(p, z)$ is $< D_i/2$. Figure 3.4-1 illustrates the situation, depicting the plane determined by point p_i, point p_j, and point y, which is the closest point to p_i in $H(p, z)$; note that $\|y - p_i\| < D_i/2$. In the figure, points x_1, x_2, which reside both in the hyperplane $H(p, z)$ and the plane of the figure, are chosen because $\|x_1 - p_i\| = \|x_2 - p_i\| = D_i$. The reader may verify that $\|x_1 - x_2\| < D_i$, so that the set $K = H(p, z) \cap B(y, D_i/2)$ is contained in $B(p_i, D_i)$; moreover, the density function f_i has the value $1/D_i$ everywhere on K. Since

$$\int_{w \in K} f_i^{d-1}(w)(\mathrm{d}w)^{d-1} = \frac{\mathrm{Vol}(B^{d-1}(0, D_i/2))}{D_i^{d-1}}$$

$$= \frac{\pi^{(d-1)/2}}{2^{d-1}\Gamma(d/2 + 1/2)}$$

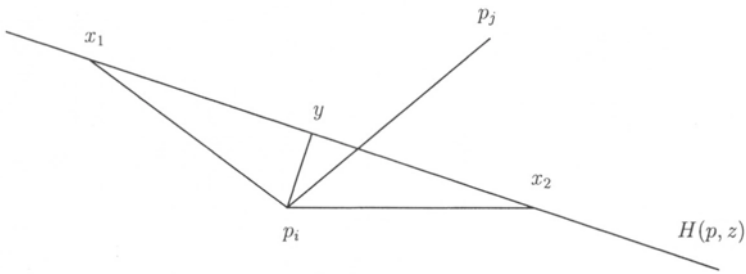


Figure 3.4-1. Edge (p_i, p_j) crossing the hyperplane $H(p, z)$.

is a constant—call it γ—we have

$$\int_{w \in H(p,z)} f_i^{d-1}(w)(\mathrm{d}w)^{d-1} \geqslant \gamma$$

We can now obtain a lower bound for the integral in Theorem 3.4.3:

$$\begin{aligned}
\int_{w \in H(p,z)} g^{d-1}(w)(\mathrm{d}w)^{d-1} &= \int_{w \in H(p,z)} \left(\sum_{i=1}^{N} f_i(w) \right)^{d-1} (\mathrm{d}w)^{d-1} \\
&\geqslant \int_{w \in H(p,z)} \sum_{i=1}^{N} f_i^{d-1}(w)(\mathrm{d}w)^{d-1} \\
&= \sum_{i=1}^{N} \int_{w \in H(p,z)} f_i^{d-1}(w)(\mathrm{d}w)^{d-1} \\
&\geqslant \sum_{p_i \in M} \int_{w \in H(p,z)} f_i^{d-1}(w)(\mathrm{d}w)^{d-1} \\
&\geqslant \gamma |M|
\end{aligned}$$

From the upper bound on the integral in Theorem 3.4.3, we conclude that $\gamma|M| = O(\alpha^{d(d-1)} N^{(d-1)/d})$. Since the constant γ depends only on d, we finally have $|M| = O(\alpha^{d(d-1)} N^{(d-1)/d})$, as desired.

The existence of a polynomial-time, randomized algorithm to find the separator follows from the previous discussion □

3.5. Network Flow Approaches to Graph Separation

In this section we present two classes of algorithms for finding (1/2)-edge-separators—or, for short, *bisectors*—for graphs, based on network flow algorithms. We start with the basics of the network flow problem in Section 3.5.1. We turn in Section 3.5.2 to an algorithm of Bui *et al.* [1987] for efficiently finding an *exactly minimum* bisection in a random regular graph that is known to have a small bisection. (This algorithm was developed to explain the observed, but unexplained, fact that certain heuristic algorithms seemed to find small graph bisections in practice.) We conclude in Section 3.5.3 with an approximation algorithm due to Leighton and Rao [1988] for finding nearly optimal edge-separators in arbitrary graphs.

3.5.1. The Basics of Network Flow

The *networkflow problem* is properly stated in the context of a network with edge capacities. A *network* $\mathcal{G} = (\mathbf{N}, \mathbf{E})$ is a directed graph having two distinguished nodes: a *source* s, and a *sink* t. A *capacity function*, cap, for $\mathcal{G}$ is a function mapping $\mathbf{E}(\mathcal{G})$ to the nonnegative reals. More conveniently, we require

$$\text{cap}: \mathbf{N} \times \mathbf{N} \to R^{\geq 0}$$

with the convention that $\text{cap}(u, v) = 0$ whenever $(u, v) \notin \mathbf{E}(\mathcal{G})$. A *flow* in $\mathcal{G}$ is a function $f : \mathbf{N} \times \mathbf{N} \to R$ that gives a *flow value* $f(u, v)$ from u to v and that satisfies three properties:

1. *Antisymmetry*: For all $u, v \in \mathbf{N}(\mathcal{G})$, $f(u, v) = -f(v, u)$. In particular, $f(u, u) = 0$ for all $u \in \mathbf{N}(\mathcal{G})$.
2. *Capacity constraint*: For all $u, v \in \mathbf{N}(\mathcal{G})$, $f(u, v) \leq \text{cap}(u, v)$.
3. *Flow conservation*: For every $u \in \mathbf{N}(\mathcal{G}) - \{s, t\}$, the flow from u satisfies

$$\sum_{v \in \mathbf{N}(\mathcal{G})} f(u, v) = 0$$

Because of antisymmetry, the flow conservation property can be interpreted as asserting that the net positive flow out of u equals the net positive flow into v:

$$\sum_{v \in \mathbf{N}(\mathcal{G})} \max\{0, f(v, u)\} = \sum_{v \in \mathbf{N}(\mathcal{G})} \max\{0, f(u, v)\}$$

The *value* $|f|$ *of flow* f is the net flow from the source s:

$$|f| = \sum_{v \in \mathbf{N}(\mathcal{G})} f(s, v)$$

The archetypical network flow problem is to find a flow of maximum value, called a *max-flow*. The classical mechanism for finding a max-flow is a *cut* of the network $\mathcal{G}$.

A *cut* (S, T) of $\mathcal{G}$ is any partition of $\mathbf{N}(\mathcal{G})$ such that $s \in S$ and $t \in T$. An edge that has one endpoint in each of S and T *crosses* the cut. The *capacity* $\text{cap}(S, T)$ of the cut (S, T) is the sum of the capacities from S to T:

$$\text{cap}(S, T) = \sum_{u \in S, v \in T} \text{cap}(u, v)$$

Of particular importance is any cut of minimum capacity, a *min-cut* (S, T). Given a flow f, the *flow* $f(S, T)$ *across the cut* (S, T) is

$$f(S, T) = \sum_{u \in S, v \in T} f(u, v)$$

By the capacity constraint, we always have $f(S, T) \leqslant \mathbf{cap}(S, T)$. It is easy to show that $f(S, T) = |f|$ for any cut (S, T).

The most basic theorem of network flow theory is the max-flow/min-cut theorem, which has given rise to a number of polynomial-time algorithms for finding max-flows and min-cuts in networks.

THEOREM 3.5.1. *For any network* $\mathcal{G}$ *with capacities* cap, *the value of a max-flow* f *equals the capacity of a min-cut* (S, T).

For our purposes we want to explore whether Theorem 3.5.1 can be used to identify a small set of edges that effect a cut (S, T) in which sets S and T are "large." Before we can explore this promise, though, we must adapt the network flow problem to the undirected graphs that are the subject of our separator theory.

We convert an undirected graph $\mathcal{G} = (\mathbf{N}, \mathbf{E})$ to a network by

1. Converting $\mathcal{G}$ into a digraph by replacing each edge $(u, v) \in \mathbf{E}(\mathcal{G})$ by a mated pair of opposing arcs, (u, v) and (v, u)
2. Choosing a source node s and a sink node t in $\mathbf{N}(\mathcal{G})$
3. Endowing the digraph with *symmetric* edge capacities: for $u, v \in \mathbf{N}(\mathcal{G})$, $\text{cap}(u, v) = \text{cap}(v, u)$

As we endow $\mathcal{G}$ with edge capacities, the simplest—indeed, our default—choice is to give each edge unit capacity. With that assignment, the max-flow/min-cut theorem does indeed identify a smallest set of edges, specifically of cardinality $\text{cap}(S, T)$, which separates s from t. Unfortunately, we are unlikely to be so lucky that the two parts of the identified min-cut (S, T) are roughly balanced in cardinality. Indeed, if we focus on graphs having some fixed constant maximum degree d, the capacity of the min-cut will never exceed d, and $\min(|S|, T|)$ is also likely to be bounded by a constant. Thus, as we shall see in the subsequent two subsections, more creativity is required to adapt network flow algorithms to the graph bisection problem.

Henceforth, we do not distinguish between an undirected graph and its derived networks: the previous construction will implicitly always have been used to derive a network from a graph whenever needed.

3.5.2. Regular Graphs with Small Bisectors

In this section we consider only regular graphs of fixed degree $d \geq 3$. Think of selecting, uniformly at random, a d-regular graph $\mathcal{G}$ having N nodes. Since $\mathcal{G}$ has $dN/2$ edges, the expected number of edges in a random bisector for $\mathcal{G}$ is $dN/4$, independent of the choice of $\mathcal{G}$. In fact, for almost all d-regular graphs having N nodes, the minimum bisection-width is $\Theta(dN)$ (cf. Bui *et al.* [1987]). This means that, over the ensemble of all random d-regular graphs, no bisection algorithm is going to do appreciably better than a random bisection algorithm. This is a frustrating fact since, as one might expect, "real" graphs—those encountered in practice—never have such large bisection width. Hence, if we are striving to prove that an algorithm efficiently finds small bisectors for "real" graphs, then we must test our algorithms' performance on graphs whose bisection widths are $o(dN)$.[7] We now construct such a class of graphs.

Let $\mathbf{G}(N, d, b)$ be the class of N-node, d-regular graphs that have unique bisectors of size b and no smaller bisector; hence, each $\mathcal{G} \in \mathbf{G}(N, d, b)$ has minimum bisection-width b. Bui *et al.* [1987] use the theory of network flow to craft an algorithm that, with high probability, returns the unique minimum bisector for any $\mathcal{G} \in \mathbf{G}(N, d, b)$, for sufficiently small b.

THEOREM 3.5.2. *Let $d \geq 3$, and let $b(N)$ be an integer function such that $b(N) = o(N^{1-1/\lfloor (d+1)/2 \rfloor})$. There is a deterministic polynomial-time algorithm that returns the unique minimum bisector for almost all graphs in $\mathbf{G}(N, d, b)$; it fails to return an answer on the remainder. Moreover, if the algorithm returns a bisector for a given d-regular graph, then it is guaranteed to be a minimum bisector.*

We now present an algorithm that demonstrates the theorem in the special case when $b(N) < \sqrt{N}$.

Let $\mathcal{G}$ be a d-regular graph, where $d \geq 3$. Focus on a node $v \in \mathbf{N}(\mathcal{G})$ and on its *p-neighborhood* $D(v, p)$, where $p \geq 0$, which is the set of nodes at distance $\leq p$ from v. Observe that $|D(v, p)| < d^{p+1}$ and that the number of edges connecting $D(v, p)$ with $\mathbf{N}(\mathcal{G}) - D(v, p)$ is $\leq d^{p+1}$. For a fixed $p \geq 0$, we construct a network from $\mathcal{G}$ as follows. Let s and t be two nodes that are distance $\geq 2p + 1$ apart in $\mathcal{G}$. Take the subgraph of $\mathcal{G}$ induced by the set $D(s, p)$, and contract it into a single node s' which has all of the adjacencies of the nodes in $D(s, p)$; similarly, take the subgraph induced by $D(t, p)$, and contract it into a single node t' which has all of the adjacencies of the nodes in $D(t, p)$. Denote by $\mathcal{G}(s', t')$ the resulting graph, which we call the *consolidated graph determined by s, t, and p.* We now begin to convert $\mathcal{G}(s', t')$ into a network in the manner described earlier, by replacing its

edges by mated opposing arcs and by designating node s' as the source and node t' as the sink. We complete the conversion of $\mathcal{G}(s', t')$ into a network by endowing its arcs with capacities. Let v be any node in $\mathbf{N}(\mathcal{G}) - D(s, p) - D(t, p)$.

- Assign arc (s', v) the capacity

$$\mathbf{cap}(s', v) = \text{number of edges between } v \text{ and } D(s, p)$$

- Assign arc, (v, t') the capacity

$$\mathbf{cap}(v, t') = \text{number of edges between } v \text{ and } D(t, p)$$

- Assign unit capacity to all other arcs incident on v.
- Assign arc (s', t') the capacity cap (s', t') = number of edges between nodes of $D(s, p)$ and of $D(t, p)$

To illustrate this procedure, consider the graph $\mathcal{G}$ in Figure 3.5-1, where $d = 3$ and $N = 12$. (This $\mathcal{G}$ is clearly far from random, but it will serve for illustration.) Taking $s = 0$, $t = 7$, and $p = 1$, we obtain $s' = D(s, p) = D(0,1) = \{0,1,2,3\}$, and $t' = D(t, p) = D(7, 1) = \{4,5,6,7\}$. After applying the described conversion procedure to the consolidated graph $\mathcal{G}(s', t')$, we obtain the (edge-capacitated) network depicted in Figure 3.5-2.

The intuition motivating the development in this section is that a min-cut (S, T) of a consolidated graph $\mathcal{G}(s', t')$ that is based on an appropriate neighborhood size p will yield a good bisection of graph $\mathcal{G}$ when the source and target nodes, s' and t', are "reconstituted" to their respective originating sets, $D(s, p)$ and $D(t, p)$. To illustrate the source of this intuition, note that the network of Figure 3.5-2 clearly has a max-flow of value 4 and a unique min-cut $(\{s', 8, 9\}, \{t', 10, 11\})$. When we translate this min-cut into a partition of $\mathcal{G}$ by "reconstituting" nodes s' and t', we obtain a cut $(\{0,1,2,3,8,9\}, \{4,5,6,7,10,11\})$ of $\mathcal{G}$, which happens to be a minimum bisection. Of course, it is impossible to extrapolate from a single example, and, indeed, the reader can certainly construct examples for which either a bisection of $\mathcal{G}$ does not result from this procedure or the resulting bisection is not a minimum one. However, we now show that the procedure is a valuable tool for finding minimum bisections. We begin by formalizing the described procedure into the algorithm BISECT-REGULAR depicted in Figure 3.5-3.

Algorithm BISECT-REGULAR is deterministic, and it is allowed to fail. However, when it does return a bisection of $\mathcal{G}$, it claims that this is a *minimum* bisection. The correctness of this claim is established in the following theorem.

THEOREM 3.5.3. *For any d-regular graph* $\mathcal{G}$, *if the call* BISECT-REGULAR $(\mathcal{G})$ *returns a bisection* (X, Y) *of* $\mathcal{G}$, *then* (X, Y) *is a minimum bisection.*

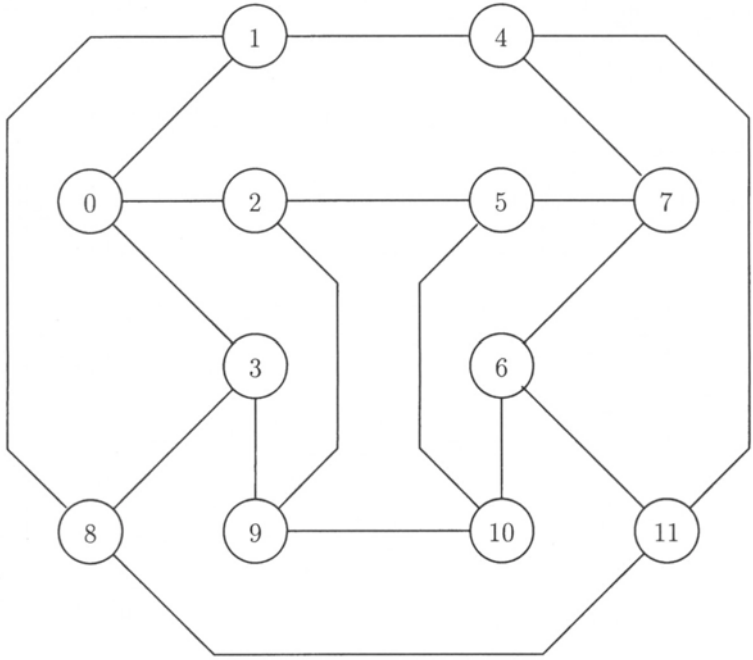


Figure 3.5-1. A 3-regular graph $\mathcal{G}$.

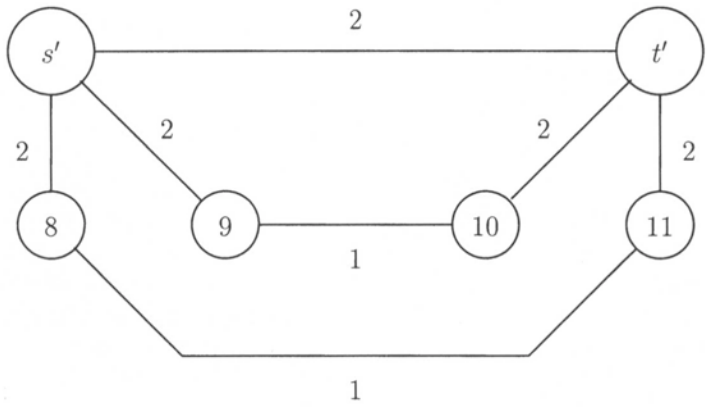


Figure 3.5-2. The consolidated graph $\mathcal{G}(s', t')$, when $s = 0$, $t = 7$, and $p = 1$.

Algorithm BISECT-REGULAR($\mathcal{G}$)

▷ Return a minimum bisection of $\mathcal{G}$ if possible.

▷ $\mathcal{G}$ must be d-regular.

1. $b \leftarrow$ "undefined"
2. $p \leftarrow (\log, \sqrt{N/2})$ -1
3. **for** all node-pairs s, $t \in \mathbf{N}(\mathcal{G})$ **with** $s \neq t$ and s, t distance $\geq 2p + 1$ apart

   do

   (a) Construct $\mathcal{G}(D(s, p), D(t, p)) = \mathcal{G}(s', t')$.

   ▷ Use your favorite max-flow algorithm here.

   (b) Find a min-cut (S', T') of $\mathcal{G}(s', t')$.

   ▷ Recover a cut of $\mathcal{G}$ from this min-cut.

   (c) $(S, T) \leftarrow S' - \{s'\} \cup D(s, p), T' - \{t'\} \cup D(t, p))$

   ▷ Remember the best cut found.

   (d) **if** b = "undefined" **or** cap$(S', T') < b$

   then $b \leftarrow$ cap(S', T'); $(X, Y) \leftarrow (S, T)$
4. **if** (X, Y) is a bisection of $\mathcal{G}$

   then return "(X, Y) is a minimum bisection".

   else return "Unable to find a minimum bisection".

Figure 3.5-3. Bisection algorithm for d-regular graphs.

PROOF. Let s, t be any pair of nodes of $\mathcal{G}$ that leads to the computation of a min-cut (S', T') of the consolidated graph produced by Algorithm BISECT-REGULAR. Since no more than $d^{p+1} = \sqrt{N/2}$ edges connect D(s, p) with $\mathbf{N}(g) - D(s, p)$, the smallest-capacity cut that the algorithm found must have capacity $\leq \sqrt{N/2}$. In particular, we know that $b = \text{cap}(X, Y) \leq \sqrt{N/2}$.

Assume now, for contradiction, that $\mathcal{G}$ admits a bisection $(\hat{X}, \hat{Y})$ of capacity $\hat{b} < b$. Let $U \subset \hat{X}$ and $V \subset \hat{Y}$ be the nodes of $\mathcal{G}$ that are incident on the $\hat{b}$ edges that actually cross the bisection. Clearly, both $|U|$ and $|V|$ are no larger than $\hat{b}$. Now, the number of nodes in $\hat{X}$ that are at distance $<p$ to some node in U is strictly smaller than $\hat{b}d^p < \hat{b}\sqrt{N/2}$; a similar bound

holds for the number of nodes in $\hat{Y}$ that are at distance $<p$ to some node in V. Since $\hat{b} < b \leqslant \sqrt{N/2}$, it follows that fewer than $\hat{b}\sqrt{N/2} < N/2$ of the $N/2$ nodes in X are at distance $<p$ from the nodes in U; we can, therefore, choose a node 5 that is at distance $\geqslant p$ from U. By symmetric reasoning, we can choose a node t that is at distance $\geqslant p$ from V. Since nodes s, t are distance $\geqslant 2p + 1$ apart in $\mathcal{G}$, this pair will lead to a computation (by the algorithm) of some min-cut (S', T'). By construction, however, $\mathbf{cap}(S', T') \leqslant \hat{b}$, contrary to the fact that the smallest min-cut found by Algorithm BISECT-REGULAR has capacity $b > \hat{b}$. This contradiction establishes the theorem. □

Since Algorithm BISECT-REGULAR solves $O(N^2)$ max-flow (min-cut) problems on a network all of whose edge capacities are integers of size $O(N)$, the algorithm clearly operates within polynomial time, though it is far from efficient in practice. It is shown in Bui *et al.* [1987] that, when presented with a graph $\mathcal{G}$ that is chosen uniformly at random from the class $\mathbf{G}(N, d, b)$, the algorithm almost certainly succeeds in finding the unique minimum bisection of $\mathcal{G}$ whenever $b = o(\sqrt{N})$. The algorithm fails to find the minimum bisection whenever some min-cut is found which has capacity $<b$ but is not a bisection of $\mathcal{G}$. Intuition suggests that adaptively increasing p so that $|D(s, p)|$ and $|D(t, p)|$ are just a bit less than $N/2$ might avoid at least some of these unwanted min-cuts. We know of no work on this or other possible enhancements of Algorithm BISECT-REGULAR.

3.5.3. Approximately Optimal Separation

In this section we develop a deterministic algorithm that produces an *approximately* optimal (1/3)-edge-separator for an arbitrary connected graph $\mathcal{G} = (\mathbf{N}, \mathbf{E})$. The algorithm is based on a generalization of the network flow problem called the *uniform multicommodity flow problem* (UMFP). We begin by defining the restricted version of UMFP that is germane to the algorithm.

The UMFP views a graph $\mathcal{G}$ as an *undirected* network with all edge capacities 1. In contrast to the standard network flow problem, wherein we focused on the flow of a single commodity from a single source to a single target, the UMFP has a distinct commodity, denoted $\langle u, v \rangle$, for every distinct pair of nodes $u, v \in \mathbf{N}(\mathcal{G})$. For each commodity $\langle u, v \rangle$, the network carries a flow $f_{\langle u,v \rangle}$ from the $\langle u, v \rangle$-**source** u to the $\langle u, v \rangle$-sink v; each flow $f_{\langle u,v \rangle}$ satisfies the antisymmetry and flow conservation properties we defined for the standard network flow problem. The capacity constraint between a pair of nodes, $x, y \in \mathbf{N}(\mathcal{G})$, restrains the aggregate absolute values of the flows

from x to y:

$$\sum_{\substack{u,v \in \mathbf{N}(\mathcal{G}) \\ u \neq v}} |f_{\langle u,v \rangle}(x, y)| \leqslant \begin{cases} 1 & \text{if } (x, y) \in \mathbf{E}(\mathcal{G}) \\ 0 & \text{if } (x, y) \notin \mathbf{E}(\mathcal{G}) \end{cases}$$

(Recall, when viewing the preceding inequality, the unit edge capacities in the network.) In the UMFP the values of all $N(N-1)$ flows are constrained to be *equal*, that is, for any *feasible flow* in $\mathcal{G}$, there must be a value F such that, for all distinct pairs of nodes $u, v \in \mathbf{N}(\mathcal{G})$, we have $|f_{\langle u,v \rangle}| = F/2$. (The factor 1/2 reflects the fact that there is a flow in both directions between u and v.) A solution to UMFP is a feasible flow that maximizes F. It is a straightforward exercise to formulate UMFP as a linear programming problem, which, as is well known, can be solved in polynomial time.

Any nonempty, proper subset X of $\mathbf{N}(\mathcal{G})$ combines with its complementary set $\bar{X} = \mathbf{N}(\mathcal{G}) - X$ to yield a partition $(X, \bar{X})$ of $\mathbf{N}(\mathcal{G})$, which is also called a *cut*. Defining the *minimum cut* for $\mathcal{G}$ to be the ratio

$$C(\mathcal{G}) = \min_{\text{cuts}(X,\bar{X})} \frac{\text{cap}(X, \bar{X})}{|X|\,|\bar{X}|}$$

it is clear that for any feasible flow, we have $F \leqslant C(\mathcal{G})$. Unfortunately, there is no simple analogue of Theorem 3.5.1 for the UMFP, which characterizes when $F = C(\mathcal{G})$. Therefore, we have to forge a connection between UMFP and graph separation via auxiliary notions. The *minimum edge expansion* of a graph $\mathcal{G}$ is the ratio

$$\alpha(\mathcal{G}) = \min_{\text{cuts}(X,\bar{X})} \frac{\text{cap}(X, \bar{X})}{\min(|X|, |\bar{X}|)}$$

A cut $(X, \bar{X})$ that achieves this minimum ratio is called a *minimum quotient separator* for $\mathcal{G}$. The parameter $\alpha(\mathcal{G})$, which quantifies the expansion properties of $\mathcal{G}$, is related to the minimum cut via the bounds

$$\tfrac{1}{2}N \cdot C(\mathcal{G}) \leqslant \alpha(\mathcal{G}) \leqslant N \cdot C(\mathcal{G}) \tag{3.5.1}$$

The following lemma is one of the building blocks we need to develop and analyze the separation algorithm we are striving for.

LEMMA 3.5.4. *For any connected graph* $\mathcal{G}$, $\alpha(\mathcal{G}) \leqslant 2E/N$.

PROOF. Since the lemma is clearly true for $N \leqslant 2$, we assume that $N > 2$. Let $(X, \bar{X})$ be a random bisection of $\mathcal{G}$, wherein $|X| = \lfloor N/2 \rfloor$. For each edge $(u, v) \in \mathbf{E}(\mathcal{G})$, the probability that (u, v) crosses the cut $(X, \bar{X})$ is

$$\frac{2\lfloor N/2 \rfloor \lceil N/2 \rceil}{N(N-1)}$$

By the linearity of expectation, the expected number of edges crossing the cut is, therefore,

$$Z = \frac{2\lfloor N/2 \rfloor \lceil N/2 \rceil E}{N(N-1)}$$

Select from among these random bisections a cut $(Y, \bar{Y})$ which is crossed by at most Z edges. By elementary computation we have

$$\begin{aligned} \frac{\text{cap}(Y, \bar{Y})}{\min(|Y|, |\bar{Y}|)} &\leqslant \frac{2\lfloor N/2 \rfloor \lceil N/2 \rceil E}{N(N-1)\lfloor N/2 \rfloor} \\ &= \frac{2\lceil N/2 \rceil E}{N(N-1)} \\ &\leqslant 2E/N \end{aligned}$$

which allows us to conclude that $\alpha(\mathcal{G}) \leqslant 2E/N$, as claimed. □

In the remainder of the section, we prepare in stages for the approximation algorithm for (l/3)-edge-separation of arbitrary connected graphs. We begin by deriving a dual formulation of UMFP.

There is a well-known reformulation of UMFP that depends on the duality theory of linear programming. For this section only, let us define a *distance function* D for the graph $\mathcal{G}$ to be any function that assigns a nonnegative weight to each edge of $\mathcal{G}$ in such a way that

$$\sum_{(u,v) \in \mathbf{E}(\mathcal{G})} D(u, v) = 1 \tag{3.5.2}$$

extend D to paths in $\mathcal{G}$ by summing the edge-weights in a path. For any distinct nodes $u, v \in \mathbf{N}(\mathcal{G})$, let Dmin$(u, v)$ be the distance of a shortest path from u to v. The *total pathlength* of D is

$$\text{TPL}(D) = \sum_{u,v \in \mathbf{N}(\mathcal{G})} \text{Dmin}(u, v)$$

By the triangle inequality, for every $u \in \mathbf{N}(\mathcal{G})$, we have

$$\mathrm{TPL}(D) \leqslant 2N \sum_{v \in \mathbf{N}(\mathcal{G})} \mathrm{Dmin}(u, v) \tag{3.5.3}$$

We can now present the dual formulation of UMFP in the following lemma, whose proof is beyond the scope of the current section. (See Iri [1967] or Shahrokhi and Matula [1990] for a proof.)

LEMMA 3.5.5. *Let $\mathcal{G}$ be a connected graph, and let*

$$\mu = \max_{D} \mathrm{TPL}(D)$$

where the maximization is over all distance functions D for $\mathcal{G}$. Then there is a feasible flow in the UMFP for $\mathcal{G}$ with value $F = 1/\mu$.

Our second step toward the desired algorithm is a demonstration that a maximum feasible flow yields a partition of $\mathbf{N}(\mathcal{G})$ that is "close to" a minimum cut. The following theorem, which contains this demonstration, can thus be viewed as an approximate analogue for the UMFP of the Max-Flow/Min-Cut Theorem (Theorem 3.5.1) of network flow theory.

THEOREM 3.5.6. *For any connected graph $\mathcal{G}$ and any distance function D for $\mathcal{G}$,*

$$\mathrm{TPL}(D) = O\left(\frac{N \log N}{\alpha(\mathcal{G})}\right)$$

PROOF. By (3.5.3), it suffices to establish the existence of a node $u \in \mathbf{N}(\mathcal{G})$ for which

$$\sum_{v \in \mathbf{N}(\mathcal{G})} \mathrm{Dmin}(u, v) = O\left(\frac{\log N}{\alpha(\mathcal{G})}\right)$$

Our search for such a u begins with the construction of the following graph $\mathcal{G}' = (\mathbf{N}', \mathbf{E}')$. Start by setting $\mathbf{N}' = \mathbf{N}(\mathcal{G}')$ equal to $\mathbf{N}(\mathcal{G})$. Next, for every edge $(x, y) \in \mathbf{E}(\mathcal{G})$, place a path having $\lfloor 1 + D(x, y)E \rfloor$ edges between nodes x and y in $\mathcal{G}'$; this endows $\mathcal{G}'$ with the remainder of its nodes and with all of its edges. Assign every edge of $\mathcal{G}'$ a uniform distance of 1/E, and observe that the distance in $\mathcal{G}'$ between any two nodes $x, y \in \mathbf{N}(\mathcal{G})$ is $\geqslant \mathrm{Dmin}(x, y)$. Note also that, by definition of distance function [cf. (3.5.2)],

$$E' \leqslant \sum_{(x,y) \in \mathbf{E}(\mathcal{G})} 1 + D(x, y)E = 2E \tag{3.5.4}$$

For any $v \in \mathbf{N}'$, let $L(v, i)$ denote the set of nodes at level (hence, distance in the usual sense of the word) i in the breadth-first spanning tree of $\mathcal{G}'$ rooted at v, and let $\mathcal{G}'(v, i) = (\mathbf{N}(v, i), \mathbf{E}(v, i))$ be the subgraph induced by levels 0 through i of this tree. With this definition in hand, apply Algorithm FIND-SUBGRAPH in Figure 3.5-4 to $\mathcal{G}'$. We proceed to show that the returned values u and R satisfy the claim in the algorithm, namely, that $\mathcal{G}'(u, R)$ contains at least half of $\mathbf{N}(\mathcal{G})$.

In order to show that Algorithm FIND-SUBGRAPH always finds u and R such that $N'(u, R)$ contains at least half of $\mathbf{N}(\mathcal{G})$, we consider the subgraph $\mathcal{H}' = (Y, \mathbf{E}(\mathcal{H}'))$ of $\mathcal{G}'$ induced by the set Y of discarded nodes. Assume, for contradiction, that at some point during the execution of Algorithm FIND-SUBGRAPH$(\mathcal{G}')$, Y contains $\geq N/4$ of $\mathcal{G}'$s nodes. Since Y cannot accrue as many as $N/2$ of $\mathcal{G}'$s nodes during a single iteration of the outer **while** loop, there is a point at which

$$N/4 \leq |Y| < 3N/4$$

By the definition of $\alpha(\mathcal{G})$, then, the number of edges crossing the cut

$$(Y \cap \mathbf{N}(\mathcal{G}),\ \bar{Y} \cap \mathbf{N}(\mathcal{G}))$$

in $\mathcal{G}$ is no smaller than $\alpha(\mathcal{G})N/4$. But then the same number of edges cross the cut $(Y, \bar{Y})$ in $\mathcal{G}'$; i.e.,

$$\text{cap}(Y, \bar{Y}) \geq \alpha(\mathcal{G})N/4$$

By the manner in which the algorithm adds discarded nodes to Y, the ratio of edges leaving Y to the number of edges in $\mathbf{E}(\mathcal{H})$ cannot be too large; in particular,

$$\frac{\text{cap}(Y, \bar{Y})}{|\mathbf{E}(\mathcal{H})|} < \varepsilon$$

Combining these inequalities, we have

$$|\mathbf{E}(\mathcal{H})| > \frac{\alpha(\mathcal{G})N}{4\varepsilon} = \frac{9E\alpha(\mathcal{G})N}{4\alpha(\mathcal{G})N} = \frac{9E}{4}$$

Since $|\mathbf{E}(\mathcal{H})| \leq E' \leq 2E$ [cf. (3.5.4)], we have a contradiction. We conclude that Algorithm FIND-SUBGRAPH always returns the claimed node u and depth R.

Algorithm FIND-SUBGRAPH($\mathcal{G}'$)

▷ Find a node $u \in \mathbf{N}(\mathcal{G})$ and a depth $R \geq 0$ so that $R = O(\log N/\varepsilon)$ and $\mathcal{G}'(u, R)$ contains at least half of $\mathbf{N}(\mathcal{G})$.

▷ Parameter ε is the "expected" expansion in successive levels, measured in number of edges. We let ε be $\alpha(\mathcal{G})N/(9E)$.

1. $X \leftarrow \mathbf{N}'$; $Y \leftarrow \varnothing$

   ▷ X contains the nodes of $\mathcal{G}'$ that remain under consideration.

   ▷ Y contains the nodes that have been discarded. (It is needed in the proof.)

2. **while** $X \cap \mathbf{N}(\mathcal{G}) \neq \varnothing$

   do

   (a) Select an arbitrary $u \in X \cap \mathbf{N}(\mathcal{G})$

   ▷ Choose an undiscarded node as the root of a breadth-first spanning tree.

   (b) $i \leftarrow 1$

   ▷ i is the depth reached by the breadth-first search.

   (c) **while** $|\mathbf{E}'(u, i)| \geq (1 + \varepsilon)|\mathbf{E}'(u, i - 1)|$

   do

   ▷ Continue to accrue depth as long as expansion is satisfied.

   if $|N'(u, i) \cap \mathbf{N}| \geq N/2$

   ▷ Check whether half the nodes of $\mathbf{N}(\mathcal{G})$ have been reached.

   then return "Node u and depth R"

   else $i \leftarrow i + 1$

   ▷ Current u was not successful.

   ▷ Eliminate subgraph $\mathcal{G}'(u, i - 1)$ and try again.

   (d) $X \leftarrow X - \mathbf{N}'(u, i - 1)$; $Y \leftarrow Y \cup \mathbf{N}'(u, i - 1)$.

   (e) Redefine $\mathcal{G}'$ to be the subgraph induced by X.

▷ Never exits unsuccessfully. See text for proof.

Figure 3.5-4. Find subgraph in $\mathcal{G}'$.

Because of the factor $(1 + \varepsilon)$ expansion that the algorithm requires before it increases the depth by 1, we have

$$(1 + \varepsilon)^{R-1} \leqslant E' \leqslant 2E$$

or, equivalently,

$$R \leqslant \frac{(\log E) + 1}{\log(1 + \varepsilon)} + 1$$

Using Lemma 3.5.4 (finally!), we bound ε as follows:

$$\varepsilon = \frac{\alpha(\mathcal{G})N}{9E} \leqslant \frac{2E}{N} \cdot \frac{N}{9E} = \frac{2}{9}$$

concluding therefrom that $\log(1 + \varepsilon) \leqslant \log(11/9) \leqslant 1$ and that $\log(1 + \varepsilon) \leqslant (9/2)\varepsilon$. Hence, $\log(1 + \varepsilon) = \Theta(\varepsilon)$, which allows us to bound R as follows:

$$R \leqslant \frac{2(\log E) + 2}{9\varepsilon} + 1 = O\left(\frac{\log N}{\varepsilon}\right)$$

Recalling that the distance of each edge of $\mathcal{G}'$ is $1/E$, the cumulative pathlength in $\mathcal{G}'(u, R)$ from node u to all nodes in $\mathbf{N}(u, R) \cap \mathbf{N}(\mathcal{G})$ is (using evocative but not quite grammatical notation)

$$\frac{NR}{E} = O\left(\frac{N \log N}{E\varepsilon}\right) = O\left(\frac{\log N}{\alpha(\mathcal{G})}\right)$$

It remains only to show that the cumulative pathlength in $\mathcal{G}'$ from node u to all nodes outside of $\mathcal{G}'(u, R)$ is bounded by $O(\log N/\alpha(\mathcal{G}))$. To this end, for each i, let $L_i = |L(u, i) \cap \mathbf{N}|$ be the number of nodes of $\mathcal{G}$ at distance i from u in $\mathcal{G}'$. Since there are at most $N/2$ nodes of $\mathcal{G}$ outside of $\mathcal{G}'(u, R)$, the cumulative pathlength from u to those nodes is just

$$\frac{1}{E} \sum_{i=R+1}^{\infty} iL_i \leqslant \frac{1}{2E} NR + \frac{1}{E} \sum_{i=R+1}^{\infty} \sum_{j=i}^{\infty} L_j$$

Since $\Sigma_{j=i}^{\infty} L_j \leqslant N/2$ whenever $i > R$, the number of edges of $\mathcal{G}'$ between levels $i - 1$ and i is no smaller than $\alpha(\mathcal{G}) \Sigma_{j=i}^{\infty} L_j$. This implies that

$$\alpha(\mathcal{G}) \sum_{i=R+1}^{\infty} \sum_{j=i}^{\infty} L_j \leqslant E' \leqslant 2E$$

The cumulative pathlength to the outside nodes is hence bounded by

$$\frac{NR}{2E} + \frac{2}{\alpha(\mathcal{G})} = O\left(\frac{\log N}{\alpha(\mathcal{G})}\right)$$

The theorem follows. □

COROLLARY 3.5.7. *For any connected graph $\mathcal{G}$ with minimum cut $C(\mathcal{G})$, the maximum feasible flow F for the UMFP satisfies $F = \Omega(C(\mathcal{G})/\log N)$.*

PROOF. Letting μ be the maximum total pathlength for $\mathcal{G}$ as in Lemma 3.5.5, Theorem 3.5.6 shows that

$$\mu = O\left(\frac{N \log N}{\alpha(\mathcal{G})}\right)$$

From our earlier observation (3.5.1) that $\alpha(\mathcal{G}) = \Theta(N \cdot C(\mathcal{G}))$, we have

$$\mu = O\left(\frac{\log N}{C(\mathcal{G})}\right)$$

By Lemma 3.5.5, the value of the maximum feasible flow of the UMFP for $\mathcal{G}$ is $F = 1/\mu = \Omega(C(\mathcal{G})/\log N)$, as claimed. □

Using the linear programming formulation for UMFP, we can find the maximum feasible flow of value F and the corresponding worst-case distance function D^*. We would like now to invoke FIND-SUBGRAPH $(\mathcal{G})$ with distance function D^* to actually locate a cut whose size is within a factor $O(\log N)$ of the minimum cut's size. The difficulty is that Algorithm FIND-SUBGRAPH is, in a practical sense, nonconstructive, because determining the value of $\alpha(\mathcal{G})$, hence the value of ε, is NP-hard. However, one can easily modify Algorithm FIND-SUBGRAPH so that it maintains an upper bound $\bar{\alpha}$ on $\alpha(\mathcal{G})$; one can then use $\bar{\alpha}$ to compute an upper bound $\bar{\varepsilon}$ on ε. The modified algorithm can obtain an initial value for $\bar{\alpha}$ from an arbitrary cut of $\mathcal{G}$. Whenever Algorithm FIND-SUBGRAPH encounters a cut that gives a smaller value, the modified algorithm will update its current values of both $\bar{\alpha}$ and $\bar{\varepsilon}$ in order to obtain a more accurate estimate of $\alpha(\mathcal{G})$. As a consequence of the adjustment of $\bar{\alpha}$ and of the analysis, the modified algorithm successfully returns the desired u and R. If the analyzed upper bound for the total pathlength outside of $\mathcal{G}'(u, R)$ fails to hold, then again the algorithm must have encountered a cut that yields a smaller value for $\bar{\alpha}$. We can then run Algorithm FIND-SUBGRAPH with a better bound on $\bar{\alpha}$ to

obtain another u and R. This iteration of updating estimates and invoking Algorithm FIND-SUBGRAPH must end after polynomially many iterations, since there are only $O(N^3)$ possible values for $\tilde{\alpha}$. We thereby have a polynomial-time approximation algorithm for computing minimum edge expansion.

As our final step toward the desired algorithm, we now show how to use an algorithm that approximates minimum edge expansion into one that computes an approximately minimal (1/3)-edge-separator. We ease into this demonstration by first showing how an algorithm that computes the exact minimum edge expansion yields an approximate separation algorithm.

THEOREM 3.5.8. *Given an algorithm* MIN-QUOTIENT-SEPARATOR *that produces a minimum quotient separator for any graph* $\mathcal{G}$, *one can craft an algorithm* APPROXIMATE-SEPARATOR *that, with at most* $N/3$ *invocations of* MIN-QUOTIENT-SEPARATOR $(\mathcal{G})$, *together with at most* $O(NE)$*-time additional work, produces a* (1/3)*-edge-separator of* $\mathcal{G}$ *whose size is within a factor* $O(\log N)$ *of optimal*

PROOF. Let $(X, \bar{X})$, where $N/3 \leqslant |X| \leqslant N/2$, be a cut of $\mathcal{G}$ that witnesses a minimum-size (1/3)-edge-separator, and let $M = \mathbf{cap}(X, \bar{X})$ be the number of edges of $\mathcal{G}$ that cross the cut. We present the claimed Algorithm APPROXIMATE-SEPARATOR in Figure 3.5-5. The fact that R decreases by at least 1 during each execution of the **while** loop yields the claimed bounds on the number of calls to MIN-QUOTIENT-SEPARATOR and the additional work performed by Algorithm APPROXIMATE-SEPARATOR. By construction the returned cut $(Y, \bar{Y})$ is a (1/3)-edge-separator of $\mathcal{G}$.

To conclude the proof, we need only show that $\mathbf{cap}(Y, \bar{Y}) = O(M \log N)$. To this end, say that the **while** loop is executed $K \geqslant 1$ times when the algorithm is invoked on graph $\mathcal{G}$. After the kth of these executions, let

- $(Y_k, \bar{Y}_k)$ be the current $(Y, \bar{Y})$ cut
- R_k be the current value of R
- $\mathcal{G}'_k$ be the current remaining subgraph of $\mathcal{G}$

For notational convenience we denote the initial values of these entities as follows:

- $(Y_0, \bar{Y}_0)$ is the "noncut" $(\varnothing, \mathbf{N}(\mathcal{G}))$.
- $R_0 = N/3$.
- $\mathcal{G}'_0 = \mathcal{G}$.

During the kth execution of the **while** loop, Algorithm MIN-QUOTIENT-SEPARATOR returns the minimum quotient separator $(V_k, \bar{V}_k)$ of $\mathcal{G}'_{k-1}$. Then $Q_k \stackrel{\text{def}}{=} |V_k|/R_{k-1}$ is the fraction of the remaining nodes that must be sliced off $\mathcal{G}$ that is supplied by V_k. One verifies the following:

Algorithm APPROXIMATE-SEPARATOR($\mathcal{G}'$)

▷ Return a (l/3)-edge-separator that is approximately optimal.

▷ Use MIN-QUOTIENT-SEPARATOR ($\mathcal{G}$) as the key subroutine.

1. $Y \leftarrow$ "undefined"
   ▷ $(Y, \bar{Y})$ is the current cut, with the invariant that $|Y| \leqslant N/2$.
2. $\mathcal{G}' \leftarrow \mathcal{G}$
   ▷ $\mathcal{G}'$ is an induced subgraph of $\mathcal{G}$ that remains to be sliced up.
3. $R \leftarrow N/3$
   ▷ R is the remaining number of nodes to be sliced off $\mathcal{G}$.
4. **while** $R > 0$
   do
   ▷ Augment Y with a minimum quotient separator of the
   ▷ larger remaining component.
   (a) $(V, \bar{V}) \leftarrow$ MIN-QUOTIENT-SEPARATOR($\mathcal{G}'$)
   ▷ Let V be the smaller side: $|V| \leqslant |\bar{V}|$.
   (b) **if** $Y =$ "undefined"
   (c) **then** $Y \leftarrow V$
   (d) **else** $Y \leftarrow Y \cup V$
   (e) $R \leftarrow R - |V|$
   (f) Let $\mathcal{G}'$ be the subgraph of $\mathcal{G}$ induced by $\bar{V}$.
   ▷ Prepare to iterate on a smaller subgraph of $\mathcal{G}$.
5. **return** "$(Y, \bar{Y})$ is an approximate separator."
   ▷ By construction $N/3 \leqslant |Y| \leqslant 2N/3$.

Figure 3.5-5. An approximate separator algorithm.

1. For all $k \in Z_K$:
   a. $0 \leqslant |Y_k| < N/3$.
   b. The number of nodes that remain to be sliced off $\mathcal{G}'_k$ is
   $$R_k = R_{k-1} - |V_k| = (1 - Q_k)R_{k-1}$$
2. For $k \in Z_K - \{0\}$, we have the following capacity bounds:
   $$\text{cap}(V_k, \bar{V}_k) \leqslant \frac{M|V_k|}{R_{k-1}} = MQ_k$$
   $$\text{cap}(V_K, \bar{V}_K) = O(M)$$

which are consequences of the following two facts.

a. $(X, \bar{X})$ induces a cut in $\mathcal{G}'_{k-1}$ whose capacity is $\leqslant M$ and whose smaller part has cardinality $\geqslant R_{k-1}$.

b. $(V_k, \bar{V}_k)$ is a minimum quotient separator of $\mathcal{G}'_{k-1}$.

3. The capacity of the returned cut is

$$\text{cap}(Y_K, \bar{Y}_K) \leqslant \sum_{k=1}^{K} \text{cap}(V_k, \bar{V}_k) \leqslant \text{cap}(V_K, \bar{V}_K) + M \sum_{k=1}^{K-1} Q_k \quad (3.5.5)$$

To complete the proof, we next bound Q_k, for $k \in Z_K - \{0\}$, as follows:

$$Q_k = \frac{R_{k-1} - R_k}{R_{k-1}} = \sum_{i=R_k+1}^{R_{k-1}} \frac{1}{R_{k-1}} \leqslant \sum_{i=R_k+1}^{R_{k-1}} \frac{1}{i}$$

Using this bound and inequality (3.5.5), we finally bound the capacity of the returned cut:

$$M \sum_{k=1}^{K} Q_k \leqslant \text{cap}(V_K, \bar{V}_K) + M \sum_{i=R_K+1}^{N/3} \frac{1}{i} = O(M \log N)$$

the final conclusion following by the logarithmic bound on the harmonic series. The theorem follows. □

With appropriate modification the preceding algorithm and proof allow us to devise a polynomial-time approximation algorithm for (1/3)-edge-separators from a polynomial-time approximation algorithm for minimum quotient separators. The details reside in the following theorem.

THEOREM 3.5.9. *Given a polynomial-time algorithm* APPROXIMATE-MIN-QUOTIENT-SEPARATOR *that produces a cut of a graph* $\mathcal{G}$ *whose size is within a factor* $O(\log N)$ *of that of a minimum quotient separator, one can craft a polynomial-time algorithm* APPROXIMATE-SEPARATOR *that produces a* (1/3)*-edge-separator of* $\mathcal{G}$ *whose size is within an* $O(\log^2 N)$ *factor of optimal.*

3.6. Heuristic Approaches to Graph Separation

Heuristic approaches to the minimum edge-bisection problem began with the elegant Kernighan–Lin (KL) heuristic algorithm, which remains a standard of comparison for all heuristics for this and related partition problems. The Fiduccia–Mattheyses (FM) heuristic algorithm speeds up the KL heuristic and processes general wiring networks (which, because of the use of multipoint nets, may be hypergraphs rather than graphs). We devote this section to reviewing these classical exemplars of heuristic algorithms for edge-bisecting graphs.

3.6.1. The Kernighan-Lin Heuristic

The KL heuristic addresses the problem of finding a small (l/2)-edge separator in a graph $\mathcal{G} = (\mathbf{N}, \mathbf{E})$. For simplicity we assume that N *is* even, leaving to the reader the clerical details needed to deal with odd N. Using the terminology of Section 3.5, we seek a cut $(X, \bar{X})$ of $\mathcal{G}$, for which $|X| = N/2$ and $\mathbf{cap}(X, \bar{X})$ is as small as possible. The heuristic takes an arbitrary initial partition $(W, \bar{W})$ of $\mathcal{G}$, constructs therefrom subsets $Y \subset W$ and $V \subset \bar{W}$, and returns an improved cut, $(W \cup V - Y, \bar{W} \cup Y - V)$.

Now, there clearly exists a choice of Y and V that yields an optimal (l/2)-edge separator for $\mathcal{G}$. The question is whether one is willing to spend the time to look for it—a questionable pursuit, given the NP-hardness result of Theorem 3.2.3. Indeed, given the probable intractability of the optimal-bisection problem, one is probably well advised to be content with a fast heuristic that returns a "good" choice for Y and V. The KL heuristic constructs such a choice by iterating on a greedy search for swaps that improve the capacity of the current cut. We present the heuristic in Figures 3.6-1 and 3.6-2.

NOTE. The careful reader may be taken aback by our assertion in Step 3 of Algorithm KL that we plan to use heaps to achieve maximum-element retrieval in *constant* time. Of course, such retrieval takes *logarithmic* time in an ordinary heap (Cormen *et al.* [1990]). We shall soon see, however, that the heaps needed here can be implemented to yield constant-time retrieval.

The following informal description of the KL heuristic should aid the reader in understanding the complete descriptions in the figures.

1. Choose a random bisection (L_0, R_0) of $\mathbf{N}(\mathcal{G})$.
2. For each $i \in \{1, 2, \ldots, N/2\}$, in turn, choose a pair $(u_i, v_i) \in L_{i-1} \times R_{i-1}$ such that
   - Neither u_i nor v_i has been swapped yet.
   - The cut (L_i, R_i) obtained from (L_{i-1}, R_{i-1}) by swapping u_i and v_i has the smallest capacity over all single-swap updates of (L_{i-1}, R_{i-1}).
3. Return the cut (L_k, R_k) that has the best capacity, i.e., for which

$$\mathbf{cap}(L_k, R_k) = \min_{0 \leq i \leq N/2} \mathbf{cap}(L_i, R_i)$$

One selects pairs of nodes to swap based on the $\mathrm{Adv}(u, v)$ values, which, at any time in the execution of the algorithm, represent the "advantage"

Algorithm KL($\mathcal{G}$)

▷ Return a (1/2)-edge-separator for graph $\mathcal{G}$.

1. Let L be a random subset of $\mathbf{N}(\mathcal{G})$ of cardinality $N/2$; $R \leftarrow \mathbf{N}(\mathcal{G}) - L$

   ▷ The partition (L, R) is the working cut.

2. **for** $u \in \mathbf{N}$

   do $L(u) \leftarrow \{v : v \in L, (u, v) \in \mathbf{E}(\mathcal{G})\}$:
   $R(u) \leftarrow \{v : v \in R, (u, v) \in \mathbf{E}(\mathcal{G})\}$

   ▷ Initialize left and right neighbor lists

3. $H \leftarrow \varnothing$

   ▷ Initialize a heap H to support maximum-element retrieval in constant time.

4. **for** $u \in L, v \in R$

   do Adv $(u, v) \leftarrow |R(u)| - |R(v)| + |L(v)| - |L(u)| - 2\text{cap}(u, v)$

   ▷ Initialize the "advantage" array that evaluates cuts.
   Insert (u, v) into heap H, ordered by Adv (u, v).

5. $A \leftarrow \text{cap}(L, R)$; $B \leftarrow 0$; $C_B \leftarrow A$

   ▷ A is the capacity of the current cut.

   ▷ B represents the number of nodes swapped in the best cut found so far.

   ▷ C_B is the capacity of this best cut.

6. **for** $i \leftarrow 1$ **to** $N/2$

   do KL-STEP(i)

   ▷ Do one step of the KL iteration.

   if $A < C_B$

   then $B \leftarrow i$; $C_B \leftarrow A$

7. $L \leftarrow \{v_1, v_2, \ldots, v_B\} \cup \{u_{B+1}, u_{B+2}, \ldots, u_{N/2}\}$;
   $R \leftarrow \{u_1, u_2, \ldots, u_B\} \cup \{v_{B-1}, v_{B+2}, \ldots, v_{N/2}\}$

   ▷ Swap $u_1, u_2, \ldots, u_B$ with $v_1, v_2, \ldots, v_B$.

8. **return** "(L, R) is the KL cut."

Figure 3.6-1. The KL heuristic.

Algorithm KL-STEP(i)

▷ Perform the ith iteration of the main KL loop.

▷ Update the data structures; return no result.

1. Select the pair (u, v) in heap H having maximum Adv (u, v) value.
2. Remove every pair containing either u or v from H.
   ▷ Guarantee that neither u nor v will ever be selected again.
3. $u_i \leftarrow u$; $v_i \leftarrow v$
   ▷ Record u and v as the ith choices.
4. $L \leftarrow L \cup \{v_i\} - \{u_i\}$; $R \leftarrow R \cup \{u_i\} - \{v_i\}$; $A \leftarrow A - \text{Adv}(u_i, v_i)$
   ▷ Swap u_i and v_i between L and R; compute resulting change in capacity.
5. ▷ Update values for neighbors of u_i.
   ▷ Note that changes to advantage values have side effects on H.
   for $w \in L(u_i) \cup R(u_i)$
   do $L(w) \leftarrow L(w) - \{u_i\}$
   $R(w) \leftarrow R(w) \cup \{u_i\}$
6. **for** $w \in L(u_i)$, $z \in R(w)$
   do Adv $(w, z) \leftarrow$ Adv $(w, z) + 2$
7. **for** $w \in R(u_i)$, $z \in L(w)$
   do dAdv $(z, w) \leftarrow$ Adv $(z, w) - 2$
8. ▷ Update values for neighbors of v_i.
   for $w \in L(v_i) \cup R(v_i)$
   do $L(w) \leftarrow L(w) \cup \{v_i\}$
   $R(w) \leftarrow R(w) - \{v_i\}$
9. **for** $w \in L(v_i)$, $z \in R(w)$
   do Adv $(w, z) \leftarrow$ Adv $(w, z) - 2$
10. **for** $w \in R(v_i)$, $z \in L(w)$
    do Adv $(z, w) \leftarrow$ Adv $(z, w) + 2$

Figure 3.6-2. The central iteration of the KL heuristic.

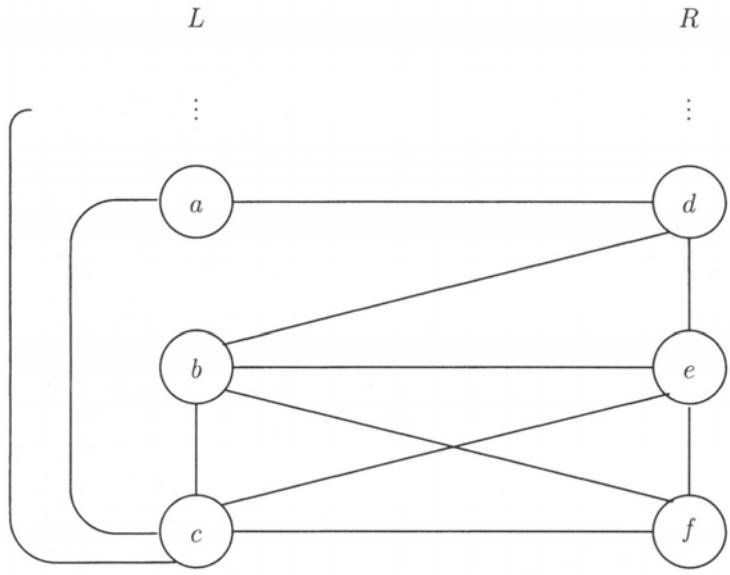


Figure 3.6-3. An example for the KL heuristic.

(i.e., the net decrease in capacity) that would result from swapping u and v in the current cut. The pairs that remain candidates for swapping are maintained in a heap ordered by advantage value. These data structures allow one to implement efficiently the key operation in each iteration of the KL heuristic, namely, the selection of the optimal pair for swapping.

To illustrate the procedure, consider Figure 3.6-3, which depicts a six-node segment of a graph $\mathcal{G}$. Say that the KL heuristic has been run on $\mathcal{G}$ for a sufficient number of iterations that the only unswapped nodes are $a, b, c \in L$ and $d, e, f \in R$. The result of swapping nodes b and f is shown in Figure 3.6-4, which has one fewer edge crossing the cut. The advantage values for the eligible pairs of nodes before any swap are tabulated in Table 3.6-1. We can read from the table that swapping nodes b and f decreases the capacity of the cut by 1—which we have already observed—and that a similar benefit would accrue from swapping nodes a and f or nodes b and d; moreover, these are the only swaps that have positive advantage, hence are the ones among which the KL heuristic would choose at this time.

In order to gauge the time complexity of the KL heuristic, we observe the following:

- Every node participates in exactly one swap.

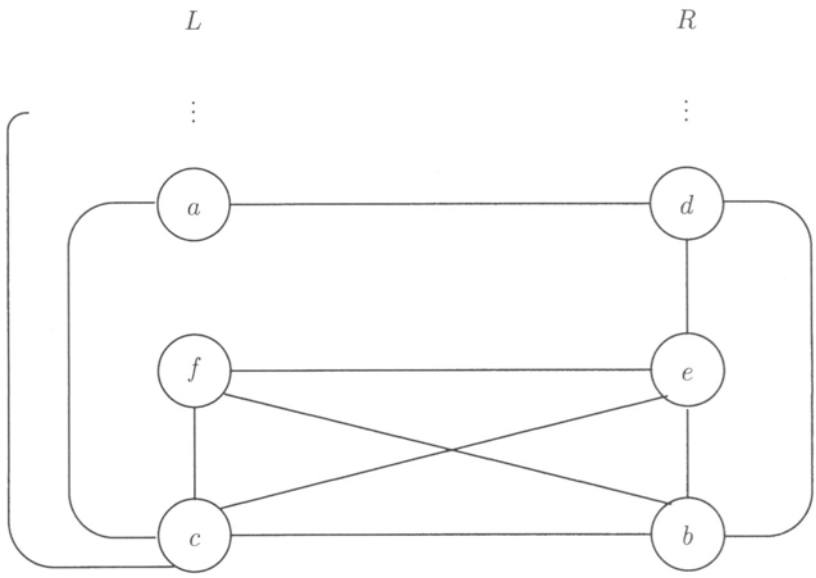


Figure 3.6-4. The example after the swap of *b* and *f*.

- Except when all cuts have the same capacity, it is likely that some swaps will *increase* the capacity of the current cut.
- Except for the reversal of left and right (i.e., of L and R), the last cut $(L_{N/2}, R_{N/2})$ is the same as (L_0, R_0).
- Each advantage value is an integer in the range $-N, \ldots, N$. Hence, the heap can be maintained using $O(N)$ buckets (think of bin sorting), so that each heap operation can be accomplished in *constant* time.

Table 3.6-1. Advantage Values for the Example

$u \in L$	$v \in R$	Adv (u, v)
a	d	-1
a	e	0
a	f	1
b	d	1
b	e	0
b	f	1
c	d	0
c	e	-3
c	f	-2

It is clear from these properties of the algorithm that its worst-case time complexity is $\Theta(N^3)$. With a bit more thought, one sees that, given a constant upper bound on the degree of $\mathcal{G}$, the time complexity drops to $\Theta(N^2)$.

Experience with the KL heuristic has shown it to be very effective in practice, especially on dense graphs. Of course, as noted earlier, for *very* dense random graphs almost any heuristic is effective, because a random cut is close in capacity to an optimal one (with high probability). The KL heuristic performs poorly on graphs that have small degree, especially degree $\leqslant 3$. As an egregious example, say that one uses the heuristic on the two-dimensional $2 \times n$ rectangular mesh $\mathcal{M}_{2,n}$, with the (admittedly foolish) initial cut (L_0, R_0) being the two rows of $\mathcal{G}$. One verifies easily that the heuristic returns a cut of capacity $\Theta(n)$, even though there is an obvious cut of capacity 2. This extreme example aside, the KL heuristic tends to perform better if it begins with a cut that is better (i.e., smaller in capacity) than a random one; hence, the heuristic can be particularly useful to improve a cut found by some other heuristic.

3.6.2. The Fiduccia-Mattheyses Heuristic

Although the original FM heuristic addresses a more general problem—the bisection of hypergraphs—than does the KL heuristic, we view it solely as a modification of KL that provides a faster heuristic for the (l/2)-edge-separator problem for graphs.

The key observation underlying the FM heuristic is that, whereas KL swaps two nodes at each iteration, thereby maintaining an identical number of nodes in each part of the cut, it may sometimes be advantageous to move only one node at a time, as long as one is careful to maintain almost the same number of nodes in each part. Applying just this idea, we obtain the heuristic in Figures 3.6-5 and 3.6-6.

Now, the advantage values in the FM heuristic depend on single nodes rather than on pairs of nodes as in the KL heuristic. Consequently, the heaps H_L and H_R in Algorithm FM never contain more than $O(N)$ elements each, in contrast to the $\Theta(N^2)$ elements that the heap in Algorithm KL contains at the start of each iteration. This leaner setting allows Algorithm FM to spend time $O(N + E)$ per iteration, in contrast to the quadratic time that Algorithm KL needs. In a typical application of Algorithm FM, the cut returned by one pass is used as the initial cut in the next pass. The iterations continue until Algorithm FM produces no further improvements in the capacity of the cut returned. Fiduccia and Mattheyses [1982] claim that, in practice, this convergence usually occurs in a small constant number of passes.

Algorithm FM($\mathcal{G}$)

▷ Return a (1/2)-edge-separator for $\mathcal{G}$.

1. Let L be a random subset of $\mathbf{N}$ of cardinality $N/2$; $R \leftarrow \mathbf{N}(\mathcal{G}) - L$
   ▷ cut (L, R) is the working cut.
2. **for** $u \in \mathbf{N}(\mathcal{G})$
   do $L(u) \leftarrow \{v : v \in L, (u, v) \in \mathbf{E}\}$
   $R(u) \leftarrow \{v : v \in R, (u, v) \in \mathbf{E}\}$
   ▷ Initialize left and right neighbor lists
3. $H_L \leftarrow \varnothing$; $H_R \leftarrow \varnothing$
   ▷ Initialize two heaps that support maximum element retrieval in constant time.
4. **for** $u \in L$
   do Adv $(u) \leftarrow |R(u)| - |L(u)|$
   Insert u in heap H_L ordered by Adv (u)
   ▷ Initialize the advantage array and heap for the left side.
5. **for** $v \in R$
   do Adv $(v) \leftarrow |L(v)| - |R(v)|$
   Insert u in heap H_R ordered by Adv (v)
   Initialize the advantage array and heap for the right side.
6. $A \leftarrow$ cap(L, R); $B \leftarrow 0$; $C_B \leftarrow A$
   ▷ A is the capacity of the current cut.
   ▷ B is the number of nodes swapped in the best cut found so far.
   ▷ C_B is the capacity of this best cut.
7. **for** $i \leftarrow 1$ to $N/2$
   do ▷ One step of the FM iteration
   FM-STEP(i)
   if $A < C_B$
   then $B \leftarrow i$; $C_B \leftarrow A$
8. $L \leftarrow \{v_1, v_2, \ldots, v_B\} \cup \{u_{B+1}, u_{B+2}, \ldots, u_{N/2}\}$;
   $R \leftarrow \{u_1, u_2, \ldots, u_B\} \cup \{v_{B+1}, v_{B+2}, \ldots, v_{N/2}\}$
   ▷ Swap $u_1, u_2, \ldots, u_B$ with $v_1, v_2, \ldots, v_B$.
9. **return** "(L, R) is the FM cut."

Figure 3.6-5. The FM heuristic.

Algorithm FM-STEP(i)

▷ Perform the ith iteration of the main FM loop.

▷ Update the data structures; return no result.

1. Select the node u in heap H_L having maximum Adv (u) value.
2. Remove u from H_L.
   ▷ Guarantee that u will not be selected again.
3. $u_i \leftarrow u$
   ▷ Record u as the ith choice.
4. $L \leftarrow L - \{u_i\}$; $R \leftarrow R \cup \{u_i\}$; $A \leftarrow A - \textbf{Adv}\ (u_i)$
   ▷ Move u_i from L to R; compute resulting change in capacity.
5. ▷ Update values for neighbors of u_i.
   ▷ Note that changes to advantage values have side effects in H_L and H_R.
   for $w \in L(u_i) \cup R(u_i)$
   do $L(w) \leftarrow L(w) - \{u_i\}$
   $R(w) \leftarrow R(w) \cup \{u_i\}$
6. **for** $w \in L(u_i)$, $z \in R(w)$
   do **Adv** $(w) \leftarrow$ **Adv** $(w) + 2$
7. **for** $w \in R(u_i)$, $z \in L(w)$
   do **Adv** $(w) \leftarrow$ **Adv** $(w) - 2$
8. Repeat the above, *mutadis mutandis*, with left (L) and right (R) interchanged, to determine which v_i to move from R to L.

Figure 3.6-6. One iteration of the FM heuristic.

3.6.3. Concluding Thoughts

Since both the KL and FM heuristics can suffer from a poor initial cut, it is always advisable to run each several tunes with different, independently chosen initial cuts. (This rule of thumb applies to many heuristics for many optimization problems.)

The basic idea that underlies both the KL and FM heuristics is that one may be able to overcome the tendency of greedy optimizers to get stuck in local extrema (either minima or maxima) by iterating a sequence of greedy optimizations, with each supplying the initial solution to its suc-

cessor. This strategy can easily, and fruitfully, be adapted to a large variety of optimization problems.

3.7. Sources

Interest in algorithms for separating graphs goes back at least to the early days of laying out electrical circuits. The first publication on the subject that we are aware of is the classic 1970 paper by Kernighan and Lin [1970]. Many variations on their heuristic have appeared, with Fiduccia and Mattheyses [1982] possibly being the best known. Other successors to Kernighan and Lin [1970] include Berry and Goldberg [1999], Goldberg and Burstein [1983], Karypis and Kumar [1999a,b], and Saab [1995]. Additionally, one can usually adapt general heuristic frameworks for combinatorial optimization problems—such as genetic algorithms (Bui and Moon [1996] and Goldberg [1989]), simulated annealing (van Laarhoven and Aarts [1987]), and taboo search (Glover [1989, 1990])—to graph separation. For examples, see Johnson *et al.* [1989] and Ganley and Heath [1994a, 1998].

The nowadays always-expected bad news of the NP-hardness of graph separation arrived among the results of Garey, Johnson, and Stockmeyer [1976]. They first show that the problem of partitioning the node-set of a graph so as to cut the maximum number of edges (SIMPLE MAX-CUT) is NP-hard, then use that result to show that the problem of bisecting the node-set so as to cut the minimum number of edges (MINIMUM BISECTION-WIDTH) is also NP-hard. The proof in this chapter follows a more direct path to the NP-hardness of MINIMUM BISECTION-WIDTH than the proof in Garey *et al.* [1976], as the SIMPLE MAX-CUT problem is not of interest here. Other papers that extend and refine this result include Bui and Jones [1992], Park and Phillips [1993], and Wagner and Wagner [1993]. In particular, Bui and Jones [1992] extend Corollary 3.2.7 to approximating the cardinality of the cut within $N^{2-\varepsilon}$ for any $\varepsilon > 0$. Some other approximation results are reported in Arora *et al.* [1995], Park and Phillips [1993], and Rao [1987, 1992]. Approximation results for MAX-CUT problems can be found in Frieze and Jerrum [1997], Goemans and Williamson [1995], and Poljak and Tuza [1995].

Lipton and Tarjan [1979] provide the classic example of a separator theorem for a class of graphs, in this case for graphs embedded in the plane. Gilbert, Hutchinson, and Tarjan [1984] extend that result to graphs embedded in orientable surfaces (2-manifolds) of genus g. Interest in the connections between topological embedding and graph separators remains

lively, as evidenced by Aleksandrov and Djidjev [1996], Awerbuch *et al.* [1998]; Bui and Peck [1988, 1992], Diks *et al.* [1988], Djidjev [1988], Garg *et al.* [1994], Miller [1986], Park and Phillips [1993], Rao [1987, 1992], Richards [1986], Spielman and Teng [1996] and Sýkora and Vrťo [1993]. The combinatorial topology of graph embeddings is covered in numerous texts, including Gross and Tucker [1987], Henle [1979], Massey [1967], and White [1984]. Filotti, Miller, and Reif [1979] give an algorithm to embed a graph in an oriented 2-manifold of its genus; it is not a polynomial-time algorithm. In fact, determining the genus of a graph is NP-hard, as shown by Thomassen [1989]. See Heath and Istrail [1992] for further results on graphs embedded in surfaces.

The results on geometric separators are derived from Miller and Thurston [1990] and Miller and Vavasis [1991], with a slight improvement in the separation factor from $1/(d + 2)$ to $1/(d + 1)$. Further results can be found in Eppstein *et al.* [1995], Gremban *et al.* [1997], Guattery [1998a,b], Guattery and Miller [1998], Miller *et al.* [1991], and Teng [1998]. The existence of a centerpoint is proved in Edelsbrunner [1987]. Algorithms to find a centerpoint in linear time are referenced in Eppstein *et al.* [1995]. Hölder's inequality for integrals is demonstrated in Hardy, Littlewood, and Pólya [1952]. The formulas for volume and surface area, as well as other results of d-dimensional geometry, can be found in Fejes Tóth [1964], Grünbaum [1967], and Sommerville [1958].

An early use of network flow techniques for a problem related to graph separators occurs in Alia and Maestrini [1976]. Bui *et al.* [1987] attack the problem of finding optimal (l/2)-edge-separators in random d-regular graphs. This class is more interesting from the algorithmic point of view because, for typical classes of random graphs, a randomly chosen cut is close to an optimal (l/2)-edge-separator with high probability (see Bui *et al.* [1987] and Ganley and Heath [1994b]). They show that this variant of the graph separator problem remains NP-complete. One of their algorithms for classes of d-regular graphs with known unique, small bisections is the basis for Section 3.5.2. Bui *et al.* also report results of some interesting experiments comparing their algorithms to greedy, simulated annealing and the Kernighan-Lin heuristic. Leighton and Rao [1988] provide the approximation algorithm based on multicommodity flows of Section 3.5.3. See Rao [1987] for the technique of converting an algorithm for a minimum quotient separator to an approximation algorithm for a (l/3)-edge-separator. More recent results are reported in Even *et al.* [1999]. For a proof of the duality result of Lemma 3.5.5, see Iri [1967] or Shahrokhi and Matula [1990]. Good sources of references to more recent results on multicommodity flow problems are Klein *et al.* [1995], Leighton *et al.* [1995], and McBride [1998]. Randomized approximation algorithms for finding good edge-separators

based on standard network flow techniques can be found in Plaisted [1990].

Fiedler [1973,1975a,b] and Donath and Hoffman [1973] pioneered the algebraic approach that uses eigenvalues for finding edge separators. In particular, Donath and Hoffman provide a lower bound on separation-width as a function of the eigenvalues of the Laplacian. More recent work in this direction can be found in Barnes [1982], Boppana [1987], Chung and Yau [1994], and Hendrickson and Leland [1995], among others. Guattery and Miller [1998] provide a critical review of eigenvalue-based heuristics; see that paper for further citations. Mohar [1988, 1989] discusses the relationship between the spectrum of a graph and its isoperimetric number. Alon [1986] applies eigenvalues to expander graphs.

Notes

1. A graph is *outerplanar* if it admits a planar embedding in which all nodes reside on a single face; cf. Section 3.3.1.
2. Throughout this section, "linear time" in a graph $\mathcal{G}$ means time $O(V(\mathcal{G}) + E(\mathcal{G}))$.
3. We are grateful to the authors and publisher of Lipton and Tarjan [1979] for permission to paraphrase from that source.
4. Two of the three edges incident on f_r belong to the tree $\mathcal{T}$; hence, there is only one *nontree* edge incident on f_r.
5. This is not as obvious as it is in the planar case. The reader is invited to prove that $\mathcal{G}^D$ cannot have multiple edges. (Since it is the dual of a triangulated graph, it cannot have any self-loops.)
6. The notation $(dx)^k$ in an integral indicate that x takes values from a k-dimensional subset of R^d—generally from a ball $(k \leqslant d)$, a sphere $(k \leqslant d - 1)$, or a hyperplane (k = d — 1).
7. As usual, we say of real functions f and g that $f(x) = o(g(x))$ if and only if the ratio $f(x)/g(x)$ tends to zero as x increases without bound.

4

Lower-Bound Techniques

4.1. Overview of Lower-Bound Techniques

4.1.1. Introduction

4.1.1.1. Main Concerns

This chapter is devoted to developing techniques for deriving lower bounds on the size of a graph's smallest separators, i.e., on its separation-width. The general setting of this study is as follows. We are presented with an N-node graph $\mathcal{G}$ and an integer M satisfying $1 \leqslant M < N$. Our task is to determine a lower bound on the *M-separation-width* of $\mathcal{G}$, which we denote by $W_{\mathcal{G}}(M)$. This quantity is the smallest number of edges of $\mathcal{G}$ that one must remove (colloquially, "cut") in order to partition $\mathcal{G}$ into two disjoint subgraphs, one containing M nodes and the other containing $N - M$ nodes. We call any such partition of $\mathcal{G}$ an *M-bipartition*.

While the development in this chapter focuses only on *edge*-bipartitions of graphs, most of the techniques we present here can be adapted with little conceptual modification to *node*-bipartitions. Moreover, for most graphs $\mathcal{G}$ one can obtain reasonably good (upper and lower) bounds on either the edge-separation-width of $\mathcal{G}$ or the node-separation-width of $\mathcal{G}$ from the other. To wit, $\mathcal{G}$'s M-node-separation-width can be no greater than its M-edge-separation-width, for one can just remove one node incident to each separator-edge rather than removing the edge itself. And, $\mathcal{G}$'s M-node-separation-width cannot be smaller than its M-edge-separation-width by more than the factor $(d_{\mathcal{G}} - 1)$, for one can just remove all but one edge incident to each separator-node rather than removing the node itself. For bounded-degree families of graphs (such as trees, meshes, butterflies, and de Bruijn networks), therefore, the M-node-separation-width and the M-edge-

separation-width are just small constant multiples of one another. For unbounded-degree families (such as hypercubes), the difference between the two measures can be significant; for instance, we shall see that the edge-bisection-width of an N-node hypercube is $\frac{1}{2}N$ (Applications 4.2.5 and 4.3.9), while its node-bisection-width is easily no larger than $\binom{N}{N/2}$ (just remove all nodes of weight $\frac{1}{2}N$), which is smaller by roughly the factor $\sqrt{N}$.

For the sake of emphasis, we reiterate (from Section 1.4) that quests for *lower* bounds on the ease of bipartitioning graphs (which is the focus of this chapter) differ from quests for *upper* bounds (which is the focus in Chapter 3) in two fundamental respects. First, when searching for lower bounds, we are concerned with partitioning graphs into subgraphs having prespecified *numbers* of nodes rather than prespecified *ratios* of sizes. Second, our cutting goals are "shallower" in a study of lower bounds than in a study of upper bounds: we seek here to bipartition only the given graph $\mathcal{G}$, rather than recursively to bipartition $\mathcal{G}$ and all subgraphs that arise from our bipartitions.

The techniques we develop in this chapter fall into four main categories, corresponding to the four technical sections that complement the present one. We begin, in Sections 4.2 and 4.3, with two bounding techniques, the packing and congestion techniques, respectively, that apply to broad classes of graphs; we hint at the strengths and weaknesses of the techniques by applying them to many different genres of graph families. Both techniques derive lower bounds on the separation-width of a graph indirectly, via *upper* bounds on some structural characteristic of the graph being separated. We continue the technical development in Section 4.4, with a bounding technique that applies specifically to complete trees. Trees require special treatment because they resist the broadly applicable techniques of Sections 4.2 and 4.3, for reasons we discuss later. We complete the chapter with a bounding technique that differs from all the others, in its using "semantic" rather than "syntactic" information to bound the difficulty of bipartitioning a graph. Specifically, this technique focuses on the computations that one can perform on a graph (in a sense made precise in that section), rather than on the structure of the graph, to bound its separation-width; the basic concept underlying the technique is the amount of information that must flow through the graph as one performs the computations.

We now provide an overview of the techniques we study in the chapter, before turning to a detailed study of each of them. Throughout, let $\mathcal{G}$ denote the graph whose separation-width we wish to discover (or bound).

Section 4.2 develops a powerful indirect technique for establishing lower bounds, so-called *packing* arguments. This technique derives a lower bound on the M-separation-width of $\mathcal{G}$ by bounding *from above* the number of edges of $\mathcal{G}$ that can be *packed* into an M-node subgraph of $\mathcal{G}$, i.e., the

number of edges in the densest M-node induced subgraph of $\mathcal{G}$. The intuition behind packing arguments is that when $\mathcal{G}$ is partitioned into subgraphs $\mathcal{G}_0$ and $\mathcal{G}_1$, those edges of $\mathcal{G}$ that cannot be "packed" into the subgraphs must have been cut while effecting the partition. The *packing lemmas* that bound (from above) the number of edges that can be "packed" into the subgraphs are somehow dual to separator theorems, except that they share the "shallowness" alluded to above. Packing arguments are among the most widely applicable lower-bound techniques.

Section 4.3 develops a lower-bound technique that proceeds by bounding *from above* the *congestion* of certain embeddings *onto* graph $\mathcal{G}$ (i.e., the guest and host graphs in the embedding have equally many nodes). Typically, one embeds *onto* $\mathcal{G}$ a graph $\mathcal{G}^*$ whose separation-width (or at least a lower bound on whose separation-width) is known in advance; the complete graph on $N(\mathcal{G})$ nodes is often a good candidate for $\mathcal{G}^*$. One then reasons that, if this embedding has congestion C, then cutting a single edge of $\mathcal{G}$ is tantamount to cutting no more than C edges of $\mathcal{G}^*$.It follows that the M-separation-width of $\mathcal{G}$ cannot be smaller than the M-separation-width of $\mathcal{G}^*$ by more than the factor C. Congestion arguments are often among the easiest to apply, reducing to little more than counting arguments, especially when the graph $\mathcal{G}$ is very "symmetric" and the graph $\mathcal{G}^*$ has a "natural" embedding onto $\mathcal{G}$.

Section 4.4 develops a technique for bounding the separation-widths of complete trees of arbitrary, uniform arity — trees in which all nonleaf nodes have equally many children. As we note there, complete trees do not yield to either packing or congestion arguments, and hence require special treatment. The charm of the major technique in Section 4.4 is its building on an analogy between the difficulty of cutting M nodes out of an N-node complete b-ary tree and the complexity of representing the fraction M/N in base b.

Section 4.5 combines the information-transfer arguments that appear in the study of VLSI layouts (cf. Section 2.4.3) with the space-time transformations that have been used to generate efficient special-purpose parallel computers for a variety of tasks. The technique we develop in this section can be applied to graphs whose structure allows them to be used as "efficient" computers for "complex" functions. While it is difficult to delimit the quoted words exactly, the intuitive meaning of "efficient" is that an desired N-argument function in only roughly $\log AT$ steps; the intuitive N-node graph can "compute"—in a sense made precise in that section—the meaning of "complex" is that any computation of the function requires the transfer of a lot of information about the input bits to a lot of the output bit-positions. The class of functions that are "complex" in this sense but that admit "efficient" special-purpose computers includes permuters of N-bit

words, cyclic shifters of N-bit words, and multipliers of N-bit numbers. It is difficult to be much clearer in a few words, so we just refer the reader to Section 4.5 at this point.

4.1.1.2. An Isolated Result

This subsection is devoted to a result that does not fall within the scope of the other sections in the chapter, yet is too important to ignore. This result concerns *expander graphs*, which have proven so useful in a variety of computational contexts in recent years; see, e.g., Babai [1991] and Friedman and Pippenger [1987]. For our purposes, we say that a graph $\mathcal{G}$ is a γ-*expander*, for some real $\gamma \in (0, 1]$, if every subset $S \subset \mathbf{N}(\mathcal{G})$ that contains $\leqslant \frac{1}{2}N(\mathcal{G})$ nodes has at least $\gamma|S|$ neighbors in $\mathbf{N}(\mathcal{G}) - S$. We remark that good lower bounds on the separation-widths of expander graphs emerge from the very definition of expansion.

PROPOSITION 4.1.1. *If the N-node graph $\mathcal{G}$ is a γ-expander, where $\gamma \in (0, 1]$, then for all $M < N$,*

$$W_{\mathcal{G}}(M) \geqslant \gamma \min(M, N - M)$$

We leave the straightforward proof to the reader.

4.2. Packing Arguments for Bounding Separation-Width

4.2.1. Introduction to Packing Arguments

4.2.1.1. Overview

This section is devoted to developing a powerful technique for bounding from below the M-edge-separation-width of a given graph $\mathcal{G}$ for any given integer $M < N(\mathcal{G})$. The technique has its origins in the following intuition. Say that we have effected the desired bipartition of $\mathcal{G}$ into subgraphs $\mathcal{G}_0$ and $\mathcal{G}_1$ of the appropriate sizes. The act of bipartitioning $\mathcal{G}$ has automatically partitioned $\mathbf{E}(\mathcal{G})$ into three disjoint collections: the edges of $\mathcal{G}$ that have become edges of $\mathcal{G}_0$, those that have become edges of $\mathcal{G}_1$, and those that connect $\mathcal{G}_0$ with $\mathcal{G}_1$ (the "cut" edges). Now, if we know (somehow) that, by dint of its size, $\mathcal{G}_0$ cannot have more than some number, say E_0, edges and that, by dint of its size, $\mathcal{G}_1$ cannot have more than some

number, say E_1, edges, then we automatically know that the number of "cut" edges can be no smaller than $E(\mathcal{G}) - E_0 - E_1$. Surprisingly, this simple observation often leads to excellent lower bounds on the number of "cut" edges.

In this section we use packing arguments to obtain bounds on the separation-widths of the following families of graphs:

- X-trees (Section 4.2.2)
- Binary and ternary hypercubes (Section 4.2.3)
- "Equilateral" meshes of arbitrary dimensionality (Section 4.2.4)

The order in which we present these bounding arguments is dictated by the arguments' complexity: X-trees yield to a simple elegant analysis; hypercubes require somewhat more attention, with an analysis whose clerical complexity grows quickly with the base of the hypercube; meshes demand sophisticated combinatorial calculations, even in the two-dimensional case. We close the section with a case study (in Section 4.2.5) that uses complete binary trees to illustrate that packing arguments can be used to obtain bounds on the mincing-widths of graphs, as well as on their separation-widths.

We opened the section by referring to packing arguments as a "powerful" bounding technique. The reader should note that the discrepancies between the actual lower bounds on M-separation-width and the bounds we obtain here range, depending on the graph family and the value of M, from nil (e.g., in the case of the bisection width of boolean hypercubes) to an additive error (e.g., in the case of the bisection-width of X-trees) to a small multiplicative factor (e.g., a factor of less than 2 in the case of multidimensional meshes and of the (l/4)-separation-width of boolean hypercubes). We know of no competing technique that gives such good bounds over such a range of values of M. We shall return to this point in Section 4.3.

4.2.1.2. Packing Functions for Graph Families

We find upper bounds on the number of edges in the subgraphs $\mathcal{G}_0$ and $\mathcal{G}_1$ that result from our bipartition of $\mathcal{G}$, via the notion of a *packing function* for $\mathcal{G}$, which is a sort of dual to the notion of a separator function for $\mathcal{G}$, in the sense that the former function measures how many edges one can keep within a subgraph of $\mathcal{G}$ of given size, while the latter measures how many edges inevitably leave the subgraph. Fortunately (for the endeavor of deriving lower bounds on separation-width), graphs belonging to a variety of well-structured graph families have easily derived packing functions.

The integer function $P_{\mathcal{G}}(n)$ is a *packing function* for the graph $\mathcal{G}$ if, for all integers $M \in \{1, 2, \ldots, N(\mathcal{G}) - 1\}$, no M-node subgraph of $\mathcal{G}$ has more than $P_{\mathcal{G}}(M)$ edges. In other words, $P_{\mathcal{G}}(M)$ bounds from above the number of edges in *any* induced subgraph of $\mathcal{G}$ on M nodes.

The formalization of our motivating intuition resides in the following fundamental lemma, which plays the role in packing arguments that a separator theorem plays in a separation argument.

LEMMA 4.2.1 (The Packing Lemma). *For all* $M \in \{1, 2, \ldots, \lfloor N/2 \rfloor\}$, *the* M*-separation-width* $W_{\mathcal{G}}(M)$ *of an* N*-node graph* $\mathcal{G}$ *that has a packing function* $P_{\mathcal{G}}(x)$ *is no smaller than*

$$E(\mathcal{G}) - P_{\mathcal{G}}(M) - P_{\mathcal{G}}(N - M)$$

PROOF. When the graph $\mathcal{G}$ is cut into two subgraphs $\mathcal{G}_0$ and $\mathcal{G}_1$, the number of edges crossing between the two graphs is precisely

$$E(\mathcal{G}) - E(\mathcal{G}_0) - E(\mathcal{G}_1)$$

Since $P_{\mathcal{G}}$ is a packing function for $\mathcal{G}$, this quantity is no smaller than

$$E(\mathcal{G}) - P_{\mathcal{G}}(N(\mathcal{G}_0)) - P_{\mathcal{G}}(N(\mathcal{G}_1))$$

The lemma follows.

In the remainder of this section, we derive upper bounds on packing functions for a variety of families of graphs that are important in a variety of computational situations; we then invoke Lemma 4.2.1 to convert these upper bounds to lower bounds on the separation-widths of these graph families. Section 4.2.5 deviates from this general theme by considering the problem of mincing the graph $\mathcal{G}$. We see that, in certain circumstances, an analogue of packing functions yields good lower bounds on the mincing-width of $\mathcal{G}$, even when the edges of $\mathcal{G}$ are weighted.

A recurring observation in our case studies is that the best way to "pack" edges into a subgraph $\mathcal{G}'$ of $\mathcal{G}$ is to have $\mathcal{G}'$ approximate as closely as possible the structure of $\mathcal{G}$, i.e., to be a smaller instance of the same graph family. For example, we verify that the most densely packed subgraphs of meshes are submeshes, the most densely packed subgraphs of hypercubes are subcubes, and so on. This insight is an important one, in that it often affords one a good starting point for the formal derivation of upper bounds on packing functions.

4.2.2. A Bound for X-Trees

The first family of graphs we study is one that can be viewed as arising by superimposing a mesh-like structure on a complete binary tree: this is the family of X-trees. The argument we use to bound the sizes of packing functions for X-trees explicitly invokes both the mesh-like and the tree-like aspects of the structure of these graphs.

THEOREM 4.2.2 (The Packing Theorem for X-Trees). *For all integers* h *and* $M < 2^{h+1} - 1$,

$$P_{\mathcal{X}_h}(M) = 2M - \lceil \log M \rceil - 1$$

is a packing function for X-trees. Moreover, the function $P_{\mathcal{X}_h}$ *is optimal, in the following sense.*

1. *Any sub-X-tree* $\mathcal{X}$ *of* $\mathcal{X}_h$ *contains* $P_{\mathcal{X}_h}(N(\mathcal{X}))$ *edges.*
2. *For any* N *there is an* N*-node induced subgraph* $\mathcal{X}$ *of* $\mathcal{X}_h$ *that contains no more than* $2N - \lceil \log N \rceil \approx P_{\mathcal{X}_h}(N)$ *edges.*

PROOF. We first show that no M-node subgraph of $\mathcal{X}_h$ contains more than $P_{\mathcal{X}_h}(M)$ edges. Then we verify the "quality" of $P_{\mathcal{X}_h}$ as a packing function. Toward the first of these goals, let us be given a maximally dense subgraph $\mathcal{X}$ of $\mathcal{X}_h$.

Establishing the bound. The inspiration for the proof resides in two simple observations about "compacting" the nodes of the subgraph $\mathcal{X}$, along levels of $\mathcal{X}_h$ and between adjacent levels, followed by a less simple argument about the "shape" of the node-set of $\mathcal{X}$ that admits a maximal-size packing. The formal argument proceeds as follows.

Let $\mathcal{X}$ be an M-node subgraph of $\mathcal{X}_h$. It is easy to argue that the number of edges of $\mathcal{X}$ can only be increased if we insist that

- For each level of $\mathcal{X}_h$, all nodes of $\mathcal{X}$ residing within that level are contiguous (so as to maximize the number of intralevel edges);
- All of the levels of $\mathcal{X}_h$ that contain nodes of $\mathcal{X}$ are contiguous (so as to maximize the potential for interlevel edges).

If we now let M_i, for $i \in Z_{h+1}$, denote the number of nodes of $\mathcal{X}$ that reside at level i of $\mathcal{X}_h$, then we find that the number of edges in $\mathcal{X}$ is bounded

above by

$$\begin{aligned} P_{\mathscr{X}_h}(M) &\leqslant \sum_i (M_i - 1) + \text{the number of interlevel edges in } \mathscr{X} \\ &\leqslant M - \ell + \text{the number of interlevel edges in } \mathscr{X} \end{aligned} \tag{4.2.1}$$

The summation in (4.2.1) ranges over those levels i of $\mathscr{X}_h$ that contain $M_i > 0$ nodes of $\mathscr{X}$; ℓ is the number of such levels. We obtain the desired bound by distinguishing two cases, according as $\ell \geqslant \log M$ or not. In both cases we exploit the (obvious) fact that the interlevel edges in $\mathscr{X}$ form a *forest of binary trees*.

Case 1: $\ell \geqslant \log M$. Since the number of edges in a forest is strictly less than the number of nodes (which is M), in this case we have directly

$$\begin{aligned} P_{\mathscr{X}_h}(M) &\leqslant M - \ell + \text{the number of interlevel edges in } \mathscr{X} \\ &\leqslant 2M - \log M - 1 \end{aligned}$$

Case 2: $\ell < \log M$. Say that $\ell = \lceil \log M \rceil - k$ for some integer $k \geqslant 1$. Let $\mathscr{F}$ be a connected subgraph of $\mathscr{X}$, hence of $\mathscr{X}_h$, that contains nodes from some ℓ levels of $\mathscr{X}_h$ and that contains only interlevel edges from $\mathscr{X}_h$. Then $\mathscr{F}$ has no more than $2^\ell - 1$ nodes—and it has that many nodes only if it is a (complete binary) *tree*. Now, since the graph $\mathscr{X}$ has M nodes, and since it contains nodes from only $\log M - k$ levels of $\mathscr{X}_h$, it follows that the subgraph of $\mathscr{X}$ that uses only interlevel edges from $\mathscr{X}_h$ has more than 2^k disjoint connected subgraphs, each of which is a tree. We conclude that the number of interlevel edges in $\mathscr{X}$ cannot exceed $M - 2^k$. We have, therefore,

$$\begin{aligned} P_{\mathscr{X}_h}(M) &\leqslant M - \ell + \text{the number of interlevel edges in } \mathscr{X} \\ &\leqslant 2M - \log M + k - 2^k \end{aligned}$$

The bound is completed by noting that $2^k - k \geqslant 1$. This completes the proof that $P_{\mathscr{X}_h}$ is a packing function for X-trees.

The quality of the bound. One verifies the optimality of the function $P_{\mathscr{X}_h}$ as follows. Let the subgraph $\mathscr{X}$ be obtained from $\mathscr{X}_h$ by removing some number of the biggest levels and some number of the rightmost nodes on the biggest remaining level. (Of course, when a node is removed, so also are all incident edges.) In the graph $\mathscr{X}$, every node save the "root" has an edge that connects it to its parent in the underlying tree, and every node save the rightmost in each level has an edge that connects it to its right neighbor. Details are left to the reader. □

By combining Lemma 4.2.1 and Theorem 4.2.2, we discover a lower bound on the M-separation-width of $\mathcal{X}_h$ (which, recall, has $2^{h+2} - h - 4$ edges).

APPLICATION 4.2.3. *For all integers h and all $M < 2^{h+1} - 1$, the M-separation-width of the height-h X-tree $\mathcal{X}_h$ is no smaller than*

$$W_{\mathcal{X}_h}(M) \geq \log\left(\frac{M(N-M)}{N+1}\right) + 1$$

where $N = N(\mathcal{X}_h) = 2^{h+1} - 1$. In particular, when M is of the form $M = \varphi N$ for some $0 < \varphi < 1$, this bound becomes

$$W_{\mathcal{X}_h}(\varphi N) \geq \log\left(N - 1 + \frac{1}{N+1}\right) + 1 + \log(\varphi(1-\varphi))$$

4.2.3. A Bound for Hypercubes

4.2.3.1. A Bound for Boolean Hypercubes

This section applies the approach of Section 4.2.1.2 to base-2 (*boolean*) hypercubes.[1] The n-dimensional boolean hypercube $\mathcal{Q}_n$ is identical to the side-2 n-dimensional mesh $\mathcal{M}_2(d)$, which we study in Section 4.2.4. Therefore, the packing functions and bounds on separation-width that we seek for boolean hypercubes are also available in the more general Theorem 4.2.8 and Application 4.2.9. The motivation for the apparently gratuitous work of the current subsection is threefold. First, because this subsection concentrates on the detailed structure of the hypercube, the bounds we obtain here have better constants than those obtained when $\mathcal{Q}_n$ is treated as a special case of the more general class of meshes. Second, the bounds obtained by viewing a boolean hypercube as a mesh are good only when the integer M is close to $N/2$; the arguments we present in this subsection exploit the detailed structure of the hypercube, hence yield good bounds throughout the range of sizes of the subgraphs. Finally, as we show in Section 4.2.3.2, the arguments we present here extend to hypercubes of higher bases, which are *not* meshes.

THEOREM 4.2.4 (The Packing Theorem for Boolean Hypercubes). *For all integers n and $M < 2^n$,*

$$P_{\mathcal{Q}_n}(M) = \tfrac{1}{2} M \log M$$

is a packing function for boolean hypercubes. Moreover, the function $\mathbf{P}_{\mathcal{Q}_n}$ *is optimal, in the following sense.*

1. *Any subcube* $\mathcal{Q}$ *of* $\mathcal{Q}_n$ *contains* $\mathbf{P}_{\mathcal{Q}_n}(N(\mathcal{Q}))$ *edges.*
2. *For any N there is an N-node induced subgraph of* $\mathcal{Q}_n$ *that contains at least* $\frac{1}{2}N\lfloor \log N \rfloor \approx \mathbf{P}_{\mathcal{Q}_n}(N)$ *edges.*

PROOF. A natural strategy for studying graphs that are (special cases of) multidimensional meshes proceeds by peeling off one dimension from a d-dimensional instance of the graph and inducing a bound for the d-dimensional case from a bound for the $(d-1)$-dimensional case. This is the strategy we use here, as well as in Sections 4.2.3.2 and 4.2.4. We turn now to the formal argument.

We proceed by induction on the dimensionality of the hypercube whose subgraphs we are packing with edges, building on the following generic equation for graphs $\mathcal{G}$:[2]

$$E(\mathcal{G}) = \frac{1}{2} \sum_{v \in \mathbf{N}(\mathcal{G})} d_{\mathcal{G}}(v) \tag{4.2.2}$$

Recalling that $d_{\mathcal{Q}_n}(v) = n$ for all $v \in \mathbf{N}(\mathcal{Q}_n)$, and invoking our intuition that the most densely packed subgraphs of $\mathcal{Q}_n$ are (almost) subcubes, (4.2.2) translates into the following inductive hypothesis for boolean hypercubes:

Hypothesis. For all dimensionalities $n' < n$, for every M-node subgraph $\mathcal{Q}$ of $\mathcal{Q}_{n'}$,

$$2E(\mathcal{Q}) = \sum_{v \in \mathbf{N}(\mathcal{Q})} d_{\mathcal{Q}}(v) \leqslant M \log M$$

This hypothesis being easily verified directly for small values of n, let us extend the induction by assuming the hypothesis for $(n-1)$-dimensional hypercubes and focusing on M-node subgraphs of $\mathcal{Q}_n$.

Let us be given a *nondegenerate* M-node subgraph $\mathcal{Q}$ of $\mathcal{Q}_n$, i.e., one that is truly n-dimensional, in the sense of *not* being a subgraph of $\mathcal{Q}_{n-1}$. (If $\mathcal{Q}$ were degenerate in this sense, then it would satisfy the theorem by the inductive hypothesis.) We begin by bipartitioning $\mathcal{Q}$ across dimension $n-1$. The nondegeneracy of $\mathcal{Q}$ ensures that this is a true bipartition of $\mathcal{Q}$ into two nonempty subgraphs. Let $\mathcal{Q}^{(0)}$ be the M_0-node induced subgraph of $\mathcal{Q}$ on those nodes that have a 0 in bit-position $n-1$, and let $\mathcal{Q}^{(1)}$ be the M_1-node induced subgraph of $\mathcal{Q}$ on those nodes that have a 1 in bit-position $n-1$ (so $M_0 + M_1 = M$). Say, with no loss of generality, that $M_0 \leqslant M_1$. Say that E edges of $\mathcal{Q}$ connect $\mathcal{Q}^{(0)}$ with $\mathcal{Q}^{(1)}$: these are the dimension-$(n-1)$ edges

that were cut in order to create $\mathcal{Q}^{(0)}$ and $\mathcal{Q}^{(1)}$ from $\mathcal{Q}$. Clearly $E \leqslant M_0$ because the edges across any dimension of $\mathcal{Q}_n$ are mutually independent, so no two cut edges touch the same node of $\mathcal{Q}^{(0)}$.

Our inductive hypothesis allows us to conclude that, for $i = 0,1$, when we consider only nodes in each $\mathcal{Q}^{(i)}$ individually,

$$M_i \log M_i \geqslant \sum_{v \in \mathbf{N}(\mathcal{Q}^{(i)})} d_{\mathcal{Q}^{(i)}}(v) = \sum_{v \in \mathbf{N}(\mathcal{Q}^{(i)})} d_{\mathcal{Q}}(v) - E$$

Summing these inequalities over the nodes of *both* subgraphs of $\mathcal{Q}$, and recalling that $E \leqslant M_0$, we achieve the following inequalities:

$$M_0 \log M_0 + M_1 \log M_1 + 2M_0 \geqslant \sum_{v \in \mathbf{N}(\mathcal{Q})} d_{\mathcal{Q}}(v)$$

Our sought packing bound will, therefore, follow from verifying that the following inequality holds whenever $M_1 \geqslant M_0 \geqslant 1$:

$$(M_0 + M_1) \log(M_0 + M_1) \geqslant M_0 \log M_0 + M_1 \log M_1 + 2M_0 \quad (4.2.3)$$

Let us simplify our task by setting $M_1 = aM_0$ for an appropriate constant $a \geqslant 1$. In this new notation, the left-hand side of inequality (4.2.3) takes the form

$$(1 + a)M_0 \log((1 + a)M_0)$$

or, equivalently,

$$M_0 \log M_0 + M_1 \log M_1 + M_0 \log(1 + a) + aM_0 \log(1 + 1/a)$$

The task of verifying inequality (4.2.3) now reduces, in our new notation, to verifying that

$$\log(1 + a) + a \log \frac{1 + a}{a} \geqslant 2$$

whenever $a \geqslant 1$. This inequality easily holds (as an equality) when $a = 1$ moreover, both terms on the left-hand side are increasing functions of a; therefore, the inequality holds for all a. This completes the proof that $\mathbf{P}_{\mathcal{Q}_n}$ is a packing function for boolean hypercubes.

The straightforward verification of the optimality of $\mathbf{P}_{\mathcal{Q}_n}$ is left to the reader. □

By combining Lemma 4.2.1 and Theorem 4.2.4, we get a lower bound on the M-separation-width of $\mathcal{Q}_n$ (which, recall, has $n2^{n-1}$ edges).

APPLICATION 4.2.5. *For all integers n and all* $M < 2^n$, *the M-separation-width of the n-dimensional boolean hypercube* $\mathcal{Q}_n$ *is no smaller than*

$$W_{\mathcal{Q}_n}(M) \geq \frac{1}{2}\log\left(\frac{N^N}{M^M(N-M)^{N-M}}\right)$$

where $N = N(\mathcal{Q}_n) = 2^n$. *In particular, when M is of the form* φN *for some* $0 < \varphi < 1$, *this bound becomes*

$$W_{\mathcal{Q}_n}(\varphi N) \geq -\tfrac{1}{2}(\varphi \log \varphi + (1-\varphi)\log(1-\varphi))N$$

4.2.3.2. A Bound for Ternary Hypercubes

As we promised earlier, the analysis that comprises the proof of Theorem 4.2.4 extends directly to hypercubes of higher base.[3] We give just a hint of how this extension goes by considering the packing argument for base-3 (*ternary*) hypercubes. Our proof of the Packing Theorem for ternary hypercubes is very close to that of Theorem 4.2.4, differing mainly in certain details that depend on the base of the hypercube.

THEOREM 4.2.6 (The Packing Theorem for Ternary Hypercubes). *For all integers n and* $M < 3^n$,

$$P_{\mathcal{Q}_n(3)}(M) \leq M \log_3 M$$

is a packing function for ternary hypercubes. Moreover, the function $P_{\mathcal{Q}_n(3)}$ *is optimal, in the following sense.*

1. *Any subcube* $\mathcal{Q}$ *of* $\mathcal{Q}_n(3)$ *contains* $P_{\mathcal{Q}_n(3)}(N(\mathcal{Q}))$ *edges.*
2. *For any N there is an N-node induced subgraph of* $\mathcal{Q}_n(3)$ *that contains at least* $N\lfloor \log_3 N \rfloor \approx P_{\mathcal{Q}_n(3)}(N)$ *edges.*

PROOF. Our proof follows the logic of Theorem 4.2.4, but with somewhat more challenging calculations and estimates. As in the proof of that theorem, we build on the generic equation (4.2.2). Recalling that $d_{\mathcal{Q}_n(3)}(v) = 2n$ for all $v \in \mathbf{N}(\mathcal{Q}_n(3))$, our intuition that the most densely packed subgraphs of $\mathcal{Q}_n(3)$ are (almost) subcubes leads us to the following inductive hypothesis for ternary hypercubes:

Hypothesis. For all dimensionalities $n' < n$, for every M-node subgraph $\mathcal{Q}$ of $\mathcal{Q}_{n'}(3)$,

$$2E(\mathcal{Q}) = \sum_{v \in \mathbf{N}(\mathcal{Q})} d_{\mathcal{Q}}(v) \leqslant 2M \log_3 M$$

This hypothesis being easily verified directly for small values of n, let us extend the induction by assuming the hypothesis for $(n-1)$-dimensional hypercubes and focusing on M-node subgraphs of $\mathcal{Q}_n(3)$.

Let us be given a *nondegenerate* M-node subgraph $\mathcal{Q}$ of $\mathcal{Q}_n(3)$. In the case of ternary hypercubes, nondegeneracy has two meanings: first, $\mathcal{Q}$ must be truly n-dimensional; i.e., it cannot be a subgraph of $\mathcal{Q}_{n-1}(3)$ (or else the theorem would hold for it by induction); second, $\mathcal{Q}$ must be truly ternary; i.e., it cannot be a subgraph of $\mathcal{Q}_n$ (or else the theorem would hold for it by Theorem 4.2.4). We begin by tripartitioning $\mathcal{Q}$ across dimension $n-1$: $\mathcal{Q}$'s nondegeneracy ensures that this is a true tripartition of $\mathcal{Q}$ into three nonempty subgraphs. For $i = 0, 1, 2$, let $\mathcal{Q}^{(i)}$ be the M_i-node induced subgraph of $\mathcal{Q}$ on those nodes having digit i in digit-position $n-1$ (so $M_0 + M_1 + M_2 = M$). Assume, with no loss of generality, that $M_0 \leqslant M_1 \leqslant M_2$. Say that these three subgraphs are interconnected as follows. We have $E_{\{0,1\}} \leqslant M_0$ edges connecting $\mathcal{Q}^{(0)}$ with $\mathcal{Q}^{(1)}$, $E_{\{0,2\}} \leqslant M_0$ edges connecting $\mathcal{Q}^{(0)}$ with $\mathcal{Q}^{(2)}$, and $E_{\{1,2\}} \leqslant M_1$ edges connecting $\mathcal{Q}^{(1)}$ with $\mathcal{Q}^{(2)}$. The inequalities on the quantities $E_{\{i,j\}}$ follows from the fact that the edges across dimension $n-1$ in $\mathcal{Q}_n(3)$ that connect $\mathcal{Q}^{(i)}$ with $\mathcal{Q}^{(j)}$ are mutually independent.

By our inductive hypothesis, for each $i \in \{0, 1, 2\}$, when we consider only nodes in $\mathcal{Q}^{(i)}$, and we set $\{i, j, k\} = \{0, 1, 2\}$, we have

$$2(M_i \log_3 M_i) \geqslant \sum_{v \in \mathbf{N}(\mathcal{Q}^{(i)})} d_{\mathcal{Q}^{(i)}}(v) = \sum_{v \in \mathbf{N}(\mathcal{Q}^{(i)})} d_{\mathcal{Q}}(v) - E_{\{i,j\}} - E_{\{i,k\}}$$

Summing these inequalities over *all* nodes of $\mathcal{Q}$, and recalling the inequalities on the $E_{\{i,j\}}$, we achieve the following inequality:

$$2(M_0 \log_3 M_0 + M_1 \log_3 M_1 + M_2 \log_3 M_2) + 4M_0 + 2M_1 \geqslant \sum_{v \in \mathbf{N}(\mathcal{Q})} d_{\mathcal{Q}}(v)$$

Our sought packing bound will follow, therefore, from verifying that the following inequality holds whenever $M_2 \geqslant M_1 \geqslant M_0 \geqslant 1$:

$$\begin{aligned}(M_0 + M_1 + M_2) &\log_3(M_0 + M_1 + M_2) \\ &\geqslant M_0 \log_3 M_0 + M_1 \log_3 M_1 + M_2 \log_3 M_2 + 2M_0 + M_1 \quad (4.2.4)\end{aligned}$$

As in the proof of Theorem 4.2.4, we simplify our task by setting $M_1 = aM_0$ and $M_2 = bM_0$, for appropriate constants $b \geq a \geq 1$; the left-hand side of inequality (4.2.4) then takes the form

$$(1 + a + b)M_0 \log_3((1 + a + b)M_0)$$

or, equivalently,

$$M_0 \log_3 M_0 + M_1 \log_3 M_1 + M_2 \log_3 M_2 + M_0 \log_3(1 + a + b)$$
$$+ aM_0 \log_3 \frac{1 + a + b}{a} + bM_0 \log_3 \frac{1 + a + b}{b}$$

The task of verifying inequality (4.2.4) thus reduces, in our new notation, to verifying that

$$\log_3(1 + a + b) + a \log_3 \frac{1 + a + b}{a} + b \log_3 \frac{1 + a + b}{b} \geq a + 2$$

whenever $b \geq a \geq 1$. Exponentiation converts this inequality to the more perspicuous form

$$(1 + a + b)^{1+a+b} \geq 9(3a)^a b^b \tag{4.2.5}$$

Consider first the case $b = a$, wherein elementary manipulation converts inequality (4.2.5) to

$$(1 + 2a)\left(\frac{(1 + 2a)^2}{3a^2}\right)^a \geq 9$$

This inequality easily holds (as an equality) when $a = 1$; the fact that both factors on the left increase with a assures us that the inequality persists for all greater values of a.

Consider next the case where a is fixed to any value $a \geq 1$ and b is allowed to vary in the range $b \geq a$. We already know that inequality (4.2.5) holds when $b = a$. If we rewrite the inequality as

$$\frac{(1 + a + b)^{1+a+b}}{b^b} \geq 9(3a)^a$$

then we see that the expression on the left-hand side increases with b (when

a is fixed), so the inequality must hold for all $b \geq a$. This completes the proof that $P_{\mathcal{Q}_n(3)}$ is a packing function for ternary hypercubes.

The straightforward verification of the optimality of $P_{\mathcal{Q}_n(3)}$ is left to the reader. □

By combining Lemma 4.2.1 and Theorem 4.2.6, we get a lower bound on the M-separation-width of $\mathcal{Q}_n(3)$ (which, recall, has $n3^n$ edges).

APPLICATION 4.2.7. *For all integers n and all* $M < 3^n$, *the M-separation-width of the n-dimensional ternary hypercube* $\mathcal{Q}_n(3)$ *is no smaller than*

$$W_{\mathcal{Q}_n(3)}(M) \geq \log_3\left(\frac{N^N}{M^M(N-M)^{N-M}}\right)$$

where $N = N(\mathcal{Q}_n(3)) = 3^n$. *In particular, when M is of the form* $M = \varphi N$ *for some* $0 < \varphi < 1$, *this bound becomes*

$$W_{\mathcal{Q}_n}(\varphi N) \geq -(\varphi \log_3 \varphi + (1-\varphi)\log_3(1-\varphi))N$$

4.2.4. A Bound for Meshes

The final family of graphs we try to pack edges into comprises the d-dimensional side-n meshes $\mathcal{M}_n(d)$.[4]

4.2.4.1. A Bound for d-Dimensional Meshes

THEOREM 4.2.8 (The Packing Theorem for d-Dimensional Meshes). *For all integers d, n, and* $M < n^d$,

$$P_{\mathcal{M}_n(d)}(M) = d(M - M^{1-1/d})$$

is a packing function for d-dimensional meshes. Moreover, the function $\mathcal{P}_{\mathcal{M}_n(d)}$ *is optimal, in thefollowing sense.*

1. *Any d-dimensional submesh* $\mathcal{M}$ *of* $\mathcal{M}_n(d)$ *contains* $P_{\mathcal{M}_n(d)}(N(\mathcal{M}))$ *edges.*
2. *For any N there is an N-node induced subgraph of* $\mathcal{M}_n(d)$ *that contains at least* $d(\lfloor N^{1/d}\rfloor^d - \lceil N^{1-1/d}\rceil) \approx P_{\mathcal{M}_n(d)}(N)$ *edges.*

PROOF. We commented in Section 4.2.3 that the analyses of hypercubes and of arbitrary multidimensional meshes share the structure of

peeling off one dimension from a d-dimensional instance of the graph and inducing a bound for the d-dimensional case from a bound for the $(d - 1)$-dimensional case. The reader should note that the major difference between our analysis here (of multidimensional meshes) and our analysis in Section 4.2.3 (of hypercubes) resides in a substantially different argument for extending the induction (which is the difficult part of the analysis). This difference is necessitated by the fact that the one-dimensional case (which is both the base case for the induction and the "glue" that binds the instances of the $(d - 1)$-dimensional case together) is structurally degenerate in the case of hypercubes but not with arbitrary meshes.

The intuition behind the current proof, which is similar to the intuition underlying the proof of Theorem 4.2.2, is seen most easily by considering the two-dimensional case. Imagine $\mathcal{M}_{n,n}$ as an overlay of two edge-disjoint graphs that share a single node-set; one graph contains only the row-edges of $\mathcal{M}_{n,n}$ while the other contains only the column-edges. Say that we wish to select an M-node subgraph $\mathcal{M}$ of $\mathcal{M}_{n,n}$ that is maximal in number of edges. We begin with the easy observation that the number of "intrarow" edges of $\mathcal{M}$ is maximized if, in every row of $\mathcal{M}_{n,n}$, the nodes that belong to $\mathcal{M}$ are all contiguous. We observe next that we can simultaneously maximize the number of intracolumn edges of $\mathcal{M}$ if we adjust the contiguous blocks of nodes of $\mathcal{M}$ in each row of $\mathcal{M}_{n,n}$ so that (a) they occupy adjacent rows, and (b) the blocks are aligned, say by "left justification." Finally—and this is the only difficult part of the proof, because it is the only truly two-dimensional argument—we show that the number of total edges of $\mathcal{M}$ is maximized if all of the contiguous blocks in the adjacent rows of $\mathcal{M}_{n,n}$ come as close as possible to having equal size. We now formalize this argument while generalizing it to meshes of arbitrary dimensionalities.

One can view the construction of size-M sets $S \subset \mathbf{N}(\mathcal{M}_n(d))$ in the following way. (These will be the node-sets of the M-node subgraphs of interest.) One partitions $\mathcal{M}_n(d)$ into n subgraphs, namely, the $(d - 1)$-dimensional submeshes that are its induced subgraphs on the node-sets

$$V_i \stackrel{\text{def}}{=} \{i\} \times Z_n^{d-1}$$

and one selects from each submesh some number $0 \leqslant M_i \leqslant M$ nodes in such a way that $\Sigma_i M_i = M$. The edges comprehended by the set so constructed come in $2n - 1$ packets:

- The submesh $\mathcal{M}_i$ on node-set V_i contributes either 0 edges (if $M_i = 0$) or $0 < M_i \leqslant P_{\mathcal{M}_n(d-1)}(M_i)$ edges (if $M_i \neq 0$).
- Each interface between "adjacent" submeshes $\mathcal{M}_i$ and $\mathcal{M}_{i+1}$ contributes at most $\min(M_i, M_{i+1})$ edges.

It is clear that this latter, interface contribution is maximized if all those $\mathcal{M}_i$ with $M_i \neq 0$ are *adjacent*, while the former, internal contribution of the submeshes is unaffected by such adjacency. Let us assume, therefore, that precisely k of the quantities M_i are nonzero and, in fact, that these nonzero ones are $\{M_i \mid i \in Z_k\}$. We have thus established the following bound on the quantity $P_{\mathcal{M}_n(d+1)}(M)$. (We use $d + 1$ rather than d as the parameter for later notational convenience.)

$$P_{\mathcal{M}_n(d+1)}(M) \leqslant \max_k \left\{ \sum_{i=1}^{k} P_{\mathcal{M}_n(d)}(M_i) + \sum_{i=1}^{k-1} \min(M_i, M_{i+1}) \,\middle|\, \sum_{i=1}^{k} M_i = M; 0 \leqslant M_i \leqslant M \right\} \tag{4.2.6}$$

We have thus easily achieved the first two steps in our proof: we have shown that the subgraph $\mathcal{M}$ should have the same mesh-like structure as $\mathcal{M}_n(d)$. We now embark on the harder task of showing that the submesh should be as close to "equilateral" as possible. We proceed by induction on d. The result being transparent for $d = 1$, let us assume that, for given $d \geqslant 1$, for all M,

$$\mathcal{P}_{\mathcal{M}_n(d)}(M) \leqslant d(M - M^{1-1/d}) \tag{4.2.7}$$

and attempt to prove as a consequence that

$$P_{\mathcal{M}_n(d+1)}(M) \leqslant (d+1)(M - M^{1-1/d+1)}) \tag{4.2.8}$$

Now compare the known inequality (4.2.6) with the desired inequality (4.2.8), and substitute the assumed generic inequality (4.2.7) wherever possible in the right-hand side of (4.2.6). After simplifying the result of these substitutions, we see that the desired inequality (4.2.8) will follow from a proof that, for arbitrary k,

$$(d+1)M^{1-1/(d+1)} \leqslant \left[M - \sum_{i=1}^{k-1} \min(M_i, M_{i+1}) \right] + d \sum_{i=1}^{k} M_i^{1-1/d} \tag{4.2.9}$$

We can simplify the task of verifying inequality (4.2.9) by noting that the bracketed term in the inequality's right-hand side is no smaller than $m \stackrel{\text{def}}{=} \max_i \{M_i\}$ because M is the sum of *all* the M_i. Hence, inequality (4.2.9)

will follow from the inequality

$$(d+1)M^{1-1/(d+1)} \leqslant m + d\sum_{i=1}^{k} M_i^{1-1/d} \tag{4.2.10}$$

We claim that this inequality holds because of the concavity of the function $g(x) = x^{1-1/d}$. To wit, letting $\mu = \lfloor M/m \rfloor$, this concavity assures us that the right-hand side of inequality (4.2.10) is minimized by taking $k = \lfloor M/m \rfloor + 1$ and setting

$$M_i = \begin{cases} m & \text{for } 1 \leqslant i \leqslant \mu \\ M - m\mu & \text{for } i = \mu + 1 = k \end{cases} \tag{4.2.11}$$

Under this assignment of values to the variables M_i, we find that

$$\sum_{i=1}^{k} M_i^{1-1/d} = \mu m^{1-1/d} + (M-\mu)^{1-1/d} \tag{4.2.12}$$

Moreover, since $1 - 1/d < 1$ and since $0 \leqslant M/m - \mu < 1$, we have

$$(M/m - \mu)^{1-1/d} \geqslant M/m - \mu = M/m - \lfloor M/m \rfloor$$

Using this inequality and (4.2.12), we infer the following lower bound on the right-hand expression of (4.2.10):

$$m + d\sum_{i=1}^{k} M_i^{1-1/d} \geqslant m + dMm^{-1/d} \tag{4.2.13}$$

Considering the right-hand side of this relation as a function of m, we can use elementary calculus to deduce that this function assumes its minimum at

$$m = M^{1-1/(d+1)}$$

Substituting this value of m in the right-hand side of (4.2.13), we finally obtain the desired inequality (4.2.10), at least for the M_i of (4.2.11). But (4.2.11) is a minimizing assignment of values to the M_i, whence (4.2.10) holds for any choice of the M_i. This completes the proof. □

By combining Lemma 4.2.1 and Theorem 4.2.8, we get a lower bound on the M-separation-width of $\mathcal{M}_n(d)$ (which, recall, has $d(n^d - n^{d-1})$ edges).

APPLICATION 4.2.9. *For all integers d and n and all* $M < n^d$, *the M-separation-width of the side-n d-dimensional mesh* $\mathcal{M}_n(d)$ *is no smaller than*

$$W_{\mathcal{M}_n(d)}(M) \geq d(M^{1-1/d} + (N - M)^{1-1/d} - N^{1-1/d})$$

where $N = N(\mathcal{M}_n(d)) = n^d$. *In particular, when M is of the form* φN *for some* $0 < \varphi < 1$, *this simplifies to*

$$W_{\mathcal{M}_n(d)}(\varphi N) \geq (\varphi^{1-1/d} + (1 - \varphi)^{1-1/d} - 1)\, dN^{1-1/d}$$

4.2.4.2. A Bound for Two-Dimensional Meshes

When $d = 2$, we can sharpen the bounds of Theorem 4.2.8 by making more careful estimates. This sharper packing bound yields a sharper bound on M-separation-width.

THEOREM 4.2.10 (The Packing Theorem for Two-Dimensional Meshes). *For all integers n and* $M < n^2$,

$$P_{\mathcal{M}_{n,n}}(M) = 2M - \lfloor \sqrt{M} \rfloor - \left\lceil \frac{M}{\lfloor \sqrt{M} \rfloor} \right\rceil$$

is a packing function for two-dimensional meshes. Moreover, the function $P_{\mathcal{M}_{n,n}}$ *is "absolutely" optimal, in that its bound is always attainable, for any M.*

PROOF. We improve the bound of Theorem 4.2.8 by deriving finer combinatorial estimates to verify that $\mathcal{M}$ should be chosen to approximate, as closely as possible, a submesh of $\mathcal{M}_{n,n}$. While such refinement can be done for any fixed value of d, the resulting estimations quickly get discouragingly tedious.

Let $r = \lfloor \sqrt{M} \rfloor$. We begin with inequality (4.2.6) in the proof of Theorem 4.2.8 and prove from it the inequality

$$P_{\mathcal{M}_{n,n}}(M) \leq 2M - r - \lceil M/r \rceil \tag{4.2.14}$$

Mimicking the development in Theorem 4.2.8, we note that, if we invoke inequality (4.2.6) for the case $d = 1$, then inequality (4.2.14) will follow from a proof that, for some k,

$$r + \lceil M/r \rceil \leq M - \sum_{i=1}^{k-1} \min(M_i, M_{i+1}) + k$$

which, in turn, will follow from a proof that

$$r + \lceil M/r \rceil \leqslant \max_i \{M_i\} + k$$

Since $\max_i\{M_i\} \geqslant \lceil M/k \rceil$, this last inequality and, hence, inequality (4.2.14) will follow from a proof that

$$r + \lceil M/r \rceil = \min_{k \in \{1,\ldots,M\}} \{k + \lceil M/k \rceil\} \tag{4.2.15}$$

We prove this fact by showing that

$$r + \lceil M/r \rceil - 2\sqrt{M} < 1 \tag{4.2.16}$$

which will imply the desired equality (4.2.15) by dint of the relations

$$2\sqrt{M} = \min_x \{x + M/x\}$$

and

$$x + \lceil M/x \rceil \geqslant x + M/x$$

To establish (4.2.16), write M in the form $M = ar + b$, where $a = \lfloor M/r \rfloor$ and $b = (M \bmod r)$, and consider three cases, based on the possible values of a and b:

1. [$a = r$ and $b = 0$]
   In this case, $r = \sqrt{M}$, so $r + \lceil M/r \rceil = 2\sqrt{M}$.
2. [$a = $ r and $b \neq 0$] or [$a = $ r + 1 and $b = 0$]
   Now, $r < \sqrt{M}$, so $r + \lceil M/r \rceil = 2r + 1 < 2\sqrt{M} + 1$.
3. [$a = r + 1$ and $b \neq 0$] or [$a = r + 2$, so, by necessity, $b = 0$]
   In this case, $r < \sqrt{M} - 1/2$, so $r + \lceil M/r \rceil = 2r + 2 < 2\sqrt{M} + 1$.

Because $r^2 \leqslant M < (r + 1)^2$, these cases exhaust all possible values of a and b and, therefore, establish the desired inequality.

The quality of the bound. Finally, we note that our bound cannot be improved since, for any value $M < n^2$, the induced subgraph of $\mathcal{M}_{n,n}$ on the

following M-node subset $S \subset \mathbf{N}(\mathcal{M}_{n,n})$ contains

$$2M - \lfloor\sqrt{M}\rfloor - \left\lceil \frac{M}{\lfloor\sqrt{M}\rfloor} \right\rceil$$

edges: if the chosen M satisfies

$$M = a\lfloor\sqrt{M}\rfloor + b, \qquad 0 \leqslant b < \lfloor\sqrt{M}\rfloor$$

then the set S comprises a adjacent columns of $\lfloor\sqrt{M}\rfloor$ nodes each, abutting an additional column with b nodes. □

The interested reader can calculate the bound on M-separation-width that ensues from Therem 4.2.10.

4.2.5. Mincing-Width via Packing Functions

We return now to an important variation on the theme of partitioning a graph into disjoint subgraphs, namely, the problem of *mincing* a graph—cutting it into many equal-size pieces.[5] Recall that in Section 1.4 we presented a general bound on the *mincing-width* of any graph in terms of the M-separation-width of the graph; specifically, we denoted by $M_{\mathcal{G}}(k)$ the k-mincing-width of the graph $\mathcal{G}$, namely, the number of edges that one must remove from $\mathcal{G}$ in order to partition it into k "equal-size" pieces. (In particular, $M_{\mathcal{G}}(2) = W_{\mathcal{G}}(N(\mathcal{G})/2)$.) In this section, we show that mincing analogues of packing functions yield good lower bounds on the mincing-width of complete b-ary trees for values of k that are powers of b. The packing technique we introduce here yields better bounds than does the crude technique of Section 1.4 (see Section 4.2.5.1); moreover, the technique can be applied even to complete trees whose edges are weighted, so one wants to minimize the cumulative weight of the edges that one cuts rather than their number (see Section 4.2.5.2). (Of course, these two quantities coincide when all edges are weighted equally.) We present a detailed argument only for complete *binary* trees, i.e., the case $b = 2$; the reader should easily be able to devise an analogous argument for any fixed arity.

4.2.5.1. Mincing Unweighted Complete Binary Trees

When we deal with unweighted graphs, as we have to this point, packing functions and theorems generalize to the study of mincing graphs in the following way.

The integer function $\hat{P}_{\mathcal{G}}(x)$ is a *mincing packing function* for the graph $\mathcal{G}$ if for all k-sum subgraphs $\mathcal{G}'$ of $\mathcal{G}$,

$$\hat{P}_{\mathcal{G}}(k) \geq E(\mathcal{G}')$$

In other words, $\hat{P}_{\mathcal{G}}(k)$ bounds from above the number of edges in *any* k-sum subgraph of $\mathcal{G}$. Note that

$$\hat{P}_{\mathcal{G}}(2) \leq P_{\mathcal{G}}(N(\mathcal{G})/2)$$

where the inequality may be strict if the edges of $\mathcal{G}$ are "clustered" in some half-size subgraph.

The formal analogue of Lemma 4.2.1 for mincing packing functions resides in the following lemma, whose straightforward proof is left to the reader.

LEMMA 4.2.11 (The Mincing Packing Lemma). *If $\hat{P}_{\mathcal{G}}(x)$ is a mincing packing function for the N-node graph $\mathcal{G}$, then, for all $k \in \{2,3,\ldots, N\}$, the k-mincing-width $M_{\mathcal{G}}(k)$ of $\mathcal{G}$ is no smaller than $E(\mathcal{G}) - \hat{P}_{\mathcal{G}}(k)$.*

When the integer k is a power of 2, packing arguments give us good bounds on the mincing-width of complete binary trees.

THEOREM 4.2.12. *Let h be a nonnegative integer, and let $k < 2^{h+1}$ be a power of 2. There is a function $f(h, k) = O(1)$ such that*

$$\hat{P}_{\mathcal{T}_h}(k) \stackrel{\text{def}}{=} 2^{h+1} - \frac{3}{2}k + f(h, k)$$

is a mincing packing function for k-sum subgraphs of $\mathcal{T}_h$.

PROOF. Let $k = 2^c$ and let $d = h + 1 - c$. Any k-sum subgraph of $\mathcal{T}_h$ consists of $k - 1$ subgraphs with 2^d nodes each and one subgraph with $2^d - 1$ nodes. We wish to bound from above the number of edges in any such subgraph of $\mathcal{T}_h$.

We note first that, because $\mathcal{T}_h$ is a tree, no M-node subgraph of $\mathcal{T}_h$ contains more than $M - 1$ edges; moreover, a subgraph contains this many edges only if it is connected. Our initial goal, therefore, is to determine how many pairwise disjoint, connected, 2^d-node subgraphs $\mathcal{T}_h$ has. Imagine that we have isolated a maximal set S_d of such subgraphs. We consider the relationship between the set S_d and the set of height-d subtrees of $\mathcal{T}_h$, i.e., the subtrees that each contain 2^d of $\mathcal{T}_h$'s leaves.

FACT 4.2.13. *The root of each of the* 2^{c-1} *height-d subtrees of* $\mathcal{T}_h$ *resides in one of the isolated subgraphs in* S_d.

PROOF OF FACT 4.2.13. If one of these roots did not reside in one of these subgraphs, then, by the assumed connectivity of the subgraphs, no other node of that root's height-d subtree could reside in one of the subgraphs. But any such "new" subtree would yield another subgraph to add to the set S_d, namely, the graph induced on the root of the subtree together with one of its height-$(d-1)$ subtrees. The existence of this new subgraph contradicts the assumed maximality of the set S_d. This proves the fact. □

By connectivity considerations, all nodes in any height-d subtree of $\mathcal{T}_h$ that reside in *some* isolated graph in S_d reside in *the same* isolated graph. We lose no generality, therefore, by assuming that each height-d subtree of $\mathcal{T}_h$ *contains* an isolated graph.

Let us now prune $\mathcal{T}_h$ by removing all of its height-d subtrees—that is, all nodes below level $h-d-1$, thereby obtaining a copy of $\mathcal{T}_{h-d-1}$. We repeat the preceding argument to obtain our goal. This is permissible since any isolated subgraph that contains a node at or above level $h-d-1$ in $\mathcal{T}_h$ may, by our argument, be assumed to contain no node below that level. Once we repeat the argument, we find that

FACT 4.2.14. *The number of mutually disjoint, connected,* 2^d*-node subgraphs of* $\mathcal{T}_h$ *is no greater than*

$$C_d \stackrel{\text{def}}{=} \sum_{i=1}^{\lfloor h/(d+1) \rfloor} 2^{h-i(d+1)+1} = \frac{2^{h+1}}{2^{d+1}-1} + O\left(\frac{1}{k}\right)$$

We have not yet produced the desired quantity, namely $k = 2^c$, of 2^d-node subgraphs of $\mathcal{T}_h$. But, as we have argued, we have produced as many such subgraphs as possible that contain $2^d - 1$ edges each. So, the best we can hope for is to find $2^{h-d+1} - 1 - C_d$ additional 2^d-node subgraphs that contain $2^d - 2$ edges each, together with one $(2^d - 1)$-node subgraph that contains $2^d - 2$ edges. If we choose our isolated subgraphs carefully, then we can achieve this goal!

Specifically, let us partition $\mathcal{T}_h$ into copies of $\mathcal{T}_d$ in the manner suggested by our counting argument; that is, we choose $2^{h-d} = 2^{c-1}$ copies of $\mathcal{T}_d$ rooted at level $h-d$ of $\mathcal{T}_h$, 2^{h-2d-1} copies rooted at level $h-2d-1$, and so on. We then let each of our isolated subgraphs comprise the root of a copy of $\mathcal{T}_d$ together with one of its height-$(d-1)$ subtrees: Each such

subgraph is a 2^d-node subgraph of $\mathcal{T}_h$ that contains $2^d - 1$ edges; moreover, this procedure yields the maximum number C_d such subgraphs, roughly $2^{h+1}/(2^{d+1} - 1)$. The remaining nodes of $\mathcal{T}_h$ lie either in the C_d nonisolated height-$(d - 1)$ subtrees of our copies of $\mathcal{T}_d$ or in the subtree of $\mathcal{T}_h$, of height $<d$, which remains at the top of $\mathcal{T}_h$ after our partition. (To be precise, this "top" subtree is empty if $d + 1$ divides $h + 1$ and has height $(h + 1 \bmod (d + 1)) - 1$ otherwise.) We complete the partition of $\mathcal{T}_h$ into $2^c - 1$ subgraphs of 2^d nodes each and one subgraph of $2^d - 1$ nodes as follows. We take the nodes (if there are any) of the "top" subtree and some $n \geq 0$ of the C_d height-$(d - 1)$ subtrees of $\mathcal{T}_h$, and we give one each of these nodes to $C_d - n - 1$ of the nonisolated height-$(d - 1)$ subtrees. We have *exactly* enough nodes to accomplish this distribution because $2^{h+1} - 1 \equiv 2^d - 1 \bmod 2^d$.

We have now achieved the desired partition of $\mathcal{T}_h$. We have 2^c mutually disjoint subgraphs of $\mathcal{T}_h$:

- C_d of them have 2^d nodes and $2^d - 1$ edges each.
- $2^c - C_d - 1$ of them have 2^d nodes and $2^d - 2$ edges each.
- One of them has $2^d - 1$ nodes and $2^d - 2$ edges.

Now, the k-sum subgraph just described must be maximal in number of edges. To wit, say that we are given any partition of $\mathcal{T}_h$ into $2^c - 1$ subgraphs, call them $\mathcal{T}_{(1)}, \mathcal{T}_{(2)}, \ldots, \mathcal{T}_{(2^c-1)}$, each having 2^d nodes, and one subgraph, call it $\mathcal{T}'$, having $2^d - 1$ nodes. Say, for $i = 1,2,\ldots, 2^c - 1$, that $a_i \geq 0$ of the subgraphs $\mathcal{T}_{(i)}$ have $2^d - i$ edges each; of course, $a_1 \leq C_d$. The k-sum subgraph of $\mathcal{T}_h$ that we have just described has, therefore, no more than $\sum_{i=1}^{2^c-1} a_i(2^d - i)$ edges. Elementary reasoning establishes that this sum is maximized when a_1 assumes its maximum value, namely C_d, and when $a_i = 0$ for all $i > 2$. It follows that, no matter how carefully we partition $\mathcal{T}_h$ into k "equal-size" pieces, the resulting k-sum subgraph cannot contain more than E_d edges of $\mathcal{T}_h$, where

$$E_d = (2^d - 1)\frac{2^{h+1}}{2^{d+1} - 1} + (2^d - 2)\left(2^{h-d+1} - \frac{2^{h+1}}{2^{d+1} - 1}\right) + O(1)$$

which simplifies to

$$E_d = 2^{h+1}\left(1 - \frac{1}{2^{d-1}} + \frac{1}{2^{d+1} - 1}\right) + O(1)$$

The claimed bound on $\hat{P}_{\mathcal{T}_h}(k)$ is now immediate. □

Lemma 4.2.11 and Theorem 4.2.12 combine to bound the mincing-width of complete binary trees from below.

APPLICATION 4.2.15. *Let h be any nonnegative integer, and let k be power of 2. The k-mincing-width $M_{\mathcal{T}_h}(k)$ of the height-h complete binary tree $\mathcal{T}_h$ is no smaller than $\frac{3}{2}k + O(1)$.*

4.2.5.2. Weighted Complete Binary Trees

In a variety of computational situations modeled by both graph partitioning and graph mincing, one wants to *weight* the edges of one's graph to reflect the *usage pattern* of the edges. For instance, when graphs are used to model the interconnection networks of parallel architectures—so graph nodes represent the processors of an architecture and graph edges represent interprocessor communication links—then the edge-weights might reflect the anticipated traffic loads on the graphs' edges. When graphs are used to model parallel computations—so graph nodes represent processes and graph edges represent (control or data) dependences between processes—then the edge-weights might reflect the cost of transferring either control information or data between the processors handling adjacent processes. When graphs are used to model data structures—so graph nodes represent data items and graph edges represent inter-item links—then the weights might reflect the anticipated traversal patterns of the data structure. In such situations, one would assess the difficulty of partitioning a graph $\mathcal{G}$ in terms of the *cumulative weight* of the edges one must remove in order to partition $\mathcal{G}$ in some way rather than in terms of the *number* of edges one must remove. The weighted-edge analogs of our graph partitioning problems are, in fact, generalizations of the unweighted partitioning problem we have considered thus far, because when all of $\mathcal{G}$'s edges are weighted equally, the cumulative-weight measure is just a fixed constant times the standard, edge-count measure.

When we deal with weighted graphs, we further generalize the mincing functions and theorems of Section 4.2.5.1 in the following way.

An *edge-weighting function* for the graph $\mathcal{G}$ is a total function $\omega: \mathbf{E}(\mathcal{G}) \rightarrow \mathbf{Reals}$. The real-valued function $\tilde{P}_{\mathcal{G}}(x; \omega)$ is an *edge-weighted mincing packing function* for the graph $\mathcal{G}$ if for all k-sum subgraphs $\mathcal{G}'$ of $\mathcal{G}$ and all edge-weighting functions ω for $\mathcal{G}$,

$$\tilde{P}_{\mathcal{G}}(k; \omega) \geq \sum_{e \in \mathbf{E}(\mathcal{G}')} \omega(e)$$

In other words, $\tilde{P}_{\mathcal{G}}(k;\omega)$ bounds from above the cumulative weight that ω assigns to the edges in *any* k-sum subgraph of $\mathcal{G}$.

Finally, let us be given a graph $\mathcal{G}$, an integer k and an edge-weighting function ω for $\mathcal{G}$. The *weighted mincing-width* $M_{\mathcal{G}}(k;\omega)$ is the smallest cumulative weight of edges of $\mathcal{G}$ whose removal partitions the graph into k "equal size" pieces.[6]

The formal analogue of Lemma 4.2.1 for edge-weighted mincing packing functions resides in the following lemma, whose proof is left to the reader.

LEMMA 4.2.16 (The Weighted Mincing Packing Lemma). *Let $\mathcal{G}$ be an N-node graph whose edges are weighted by the edge-weighting function ω; let $\tilde{P}_{\mathcal{G}}(x;\omega)$ be an edge-weighted mincing packing function for $\mathcal{G}$. For all $k \in \{2,3,\ldots,N\}$, the edge-weighted k-mincing-width $M_{\mathcal{G}}(k;\omega)$ of $\mathcal{G}$ is no smaller than*

$$\sum_{e \in \mathbf{E}(\mathcal{G})} \omega(e) - \tilde{P}_{\mathcal{G}}(k;\omega)$$

We simplify our study of edge-weighted graph mincing by constraining our domain of inquiry in two ways, the first purely clerical, the second truly restrictive (but reflecting some common applications).

1. We assume that the edges of our graphs are weighted with *probabilities*; this is often the natural way that weights arise anyway. As a consequence, we assume henceforth that $0 \leqslant \omega(e) \leqslant 1$ for each edge $e \in \mathbf{E}(\mathcal{G})$, and that the sum of all edge-weights is $\Sigma_{e \in \mathbf{E}(\mathcal{G})}\, \omega(e) = 1$. As a mnemonic device, we henceforth use π (for "probability") to denote edge-weighting functions.
2. We illustrate our bounding technique by means of a (probability distribution) edge-weighting function π which *honors the leveled structure of* $\mathcal{T}_h$, in the sense that every **level-ℓ** edge of $\mathcal{T}_h$, i.e., every edge having one terminus at level $\ell - 1$ and one at level ℓ, is assigned the same weight/probability; let us henceforth call this common weight $\pi(\ell)$, to stress that the domain of π is really $\{1,2,\ldots,h\}$ rather than $\mathbf{E}(\mathcal{G})$.

The particular edge-weighting function that we study arises in several computational situations concerning trees. It is defined implicitly by the equations

$$\pi(\ell) = 2\pi(\ell + 1) \qquad \text{for } \ell \in \{1, 2, \ldots, h\}$$

and explicitly by the equations

$$\pi(\ell) = \frac{1}{2^\ell h} \qquad \text{for } \ell \in \{1, 2, \ldots, h\}$$

In order to lend some intuition into the origins of the edge-weighting function π, we now digress to describe a few computational situations that are modeled when we use π to weight the edges of complete binary trees.

Where edge-weights come from. Consider first using $\mathcal{T}_h$ as a *binary search tree* that has 2^h equiprobable keys at its *leaves.* When accessing a key in this tree, one traverses, with equal probability, one of the 2^h length-h root-to-leaf paths in the tree. Each **level-ℓ** edge in the tree enters into precisely $2^{h-\ell}$ of these paths, hence has traversal probability $\pi(\ell)h$.

Consider next using $\mathcal{T}_h$ as a *binary search tree* that has $2^h - 1$ equiprobable keys at its *internal* (*nonleaf*) *nodes.* When accessing a key in this tree, one traverses, with equal probability, one path of length 0, two paths of length $1, \ldots, 2^\ell$ paths of length $\ell, \ldots, 2^{h-1}$ paths of length $h - 1$. Cumulatively, then, accessing all keys in the tree requires traversing

$$\sum_{i=1}^{h-1} i2^i = (h-2)2^h + 2$$

edges and, particularly, crossing each **level-ℓ** edge, where $1 \leqslant \ell < h, 2^{h-\ell} - 1$ times. Thus, each **level-ℓ** edge has traversal probability

$$\pi'(\ell) = \frac{2^{h-\ell} - 1}{(h-2)2^h + 2} = \pi(\ell) + o(1)$$

(as $h \to \infty$). Since our bounds in this section are asymptotic in nature, our bounding technique will not distinguish between the weighting functions π and π'; that is, the arguments that yield our bounds on $\tilde{P}_{\mathcal{T}_h}(k; \pi)$ will yield asymptotically identical bounds on $\tilde{P}_{\mathcal{T}_h}(k; \pi')$.

Finally, consider a *Turing machine* (TM) whose tape has the structure of $\mathcal{T}_h$: the nodes of the tree are the tape squares, and the edges of the tree delimit the permissible moves for the read-write head of the TM. Say that the TM operates in an *oblivious* manner, in the sense that its head-trajectory is predetermined, independent of the contents of the tape squares. A not-unreasonable strategy for the TM would be to scan its tape systematically in a breadth-first fashion: under such a regimen, the TM would start at the root of the tree-tape, visit all level-1 nodes in turn, returning to the

root between visits, then visit all level-2 nodes in turn—perforce, returning to the root between successive visits—and so on. At each stage $1 \leqslant \ell \leqslant h$ in this scenario, the TM visits all **level-ℓ** nodes, each visit entailing a root-to-node-to-root **length-2ℓ** path. Cumulatively, then, the TM would traverse

$$2 \sum_{i=1}^{h} i2^i = (h - 1)2^{h+2} + 4$$

edges, and, in particular, would cross each **level-ℓ** edge (where $1 \leqslant \ell \leqslant h$) $2^{h-\ell+2} - 2$ times. Thus, each **level-ℓ** edge would have traversal probability

$$\pi''(\ell) = \frac{2^{h-\ell+1-1}}{(h-1)2^{h+1}+2} = \pi(\ell) + o(1)$$

(as $h \to \infty$). Once again, the arguments that yield our bounds on $\tilde{P}_{\mathcal{T}_h}(k;\pi)$ will yield asymptotically identical bounds on $\tilde{P}_{\mathcal{T}_h}(k;\pi'')$.

Return to Packing Functions. When k is a power of 2, we can use a packing argument to obtain good lower bounds on $M_{\mathcal{T}_h}(k;\pi)$.

THEOREM 4.2.17. *For any nonnegative integers h and $k < 2^{h+1}$,*

$$\tilde{P}_{\mathcal{T}_h}(k;\pi) = \left(1 + \frac{1}{h}\right)\left(1 - \frac{1}{k+1}\right)$$

is a mincing packing function for any k-sum subgraph of $\mathcal{T}_h$ whose edges are weighted by the edge-weighting function π.

PROOF. We proceed by bounding from above the cumulative weight of the "heaviest" 2^r-node subgraph of $\mathcal{T}_h$, when the tree's edges are weighted by the function π. Some auxiliary definitions are useful.

- For any induced subgraph $\mathcal{G}$ on some of the nodes of the edge-weighted tree $\mathcal{T}_h$, Top($\mathcal{G}$) denotes that set of edges from $\mathbf{E}(\mathcal{T}_h)$ that have one terminus at a level-i node $(1 \leqslant i < h)$ that is *not* in $\mathbf{N}(\mathcal{G})$ and one terminus at a level-$(i+1)$ node that is in $\mathbf{N}(\mathcal{G})$. (Informally, the edge lies *above* a node in $\mathcal{G}$.)
- For any set $\mathbf{E}$ of edges of $\mathcal{T}_h$, Wgt($\mathbf{E}$) is the cumulative weight of the edges in $\mathbf{E}$ (the edge-weighting π being understood from context).

The importance of these notions resides in the following fact.

FACT 4.2.18. *For any induced subgraph $\mathcal{G}$ of $\mathcal{T}_h$ that does not contain the root,*

$$\mathbf{Wgt}(\mathbf{Top}(\mathcal{G})) \geq \frac{\mathbf{Wgt}(\mathbf{E}(\mathcal{G}))}{\lfloor \log(N(\mathcal{G}) + 1) \rfloor}$$

PROOF OF FACT 4.2.18. The graph $\mathcal{G}$ is clearly a subforest of $\mathcal{T}_h$. Assume first that $\mathcal{G}$ is connected (in the graph-theoretic sense), i.e., is a tree, so that $\mathbf{Top}(\mathcal{G})$ is a singleton. Let p be the weight of the single edge in $\mathbf{Top}(\mathcal{G})$. Of the $N(\mathcal{G}) - 1$ edges in $\mathbf{E}(\mathcal{G})$, it is clear that

at most 2 have probability $p/2$
at most 4 have probability $p/4$

at most 2^{l-1} have probability $p/2^{l-1}$
at most $N(\mathcal{G}) - 2^l + 1$ have probability $p/2^l$

where $l = \lfloor \log(N(\mathcal{G}) + 1) \rfloor$. It follows that

$$\mathbf{Wgt}(\mathbf{E}(\mathcal{G})) \leq p(l - 1 + 2^{-l}(N(\mathcal{G}) + 1 - 2^l))$$

whence the claimed inequality.

If $\mathcal{G}$ is not connected, then it is a sum of trees, say $\mathcal{G} = \mathcal{T}_{(1)} + \mathcal{T}_{(2)} + \cdots + \mathcal{T}_{(m)}$. By the preceding paragraph, for each $1 \leq i \leq m$,

$$\mathbf{Wgt}(\mathbf{Top}(\mathcal{T}_{(i)})) \geq \frac{\mathbf{Wgt}(\mathbf{E}(\mathcal{T}_{(i)}))}{\lfloor \log(N(\mathcal{T}_{(i)}) + 1) \rfloor} \geq \frac{\mathbf{Wgt}(\mathbf{E}(\mathcal{T}_{(i)}))}{\lfloor \log(N(\mathcal{G}) + 1) \rfloor}$$

The result follows by adding these m inequalities. □

We return now to the proof of the weighted, minced packing lemma for complete binary trees. Let $\mathcal{G} = \mathcal{G}_1 + \mathcal{G}_2 + \cdots + \mathcal{G}_k$ be a k-sum subgraph of $\mathcal{T}_h$. Say, with no loss of generality, that the root of $\mathcal{T}_h$ resides in graph $\mathcal{G}_1$. Fact 4.2.18 combines with the identity

$$\mathbf{Wgt}(\mathbf{E}(\mathcal{T}_h)) = \mathbf{Wgt}(\mathbf{E}(\mathcal{G}_1)) + \sum_{i=2}^{k} [\mathbf{Wgt}(\mathbf{E}(\mathcal{G}_i)) + \mathbf{Wgt}(\mathbf{Top}(\mathbf{E}(\mathcal{G}_i)))]$$

to allow us to infer the following inequality.

$$\mathbf{Wgt}(\mathbf{E}(\mathscr{T}_h)) \geqslant \mathbf{Wgt}(\mathbf{E}(\mathscr{G}_1)) + \left(1 + \frac{1}{k}\right) \sum_{i=2}^{k} \mathbf{Wgt}(\mathbf{E}(\mathscr{G}_i)) \qquad (4.2.17)$$

Now, the fact that (by definition) $\mathbf{Wgt}(\mathbf{E}(\mathscr{T}_h)) = 1$ combines with inequality (4.2.17) to yield the inequality

$$\mathbf{Wgt}(\mathbf{E}(\mathscr{G}_1)) \geqslant (k+1) \sum_{i=1}^{k} \mathbf{Wgt}(\mathbf{E}(\mathscr{G}_i)) - k \qquad (4.2.18)$$

Next we note, by an argument analogous to that of Fact 4.2.18, that

$$\mathbf{Wgt}(\mathbf{E}(\mathscr{G}_1)) \leqslant \frac{k - 1 + 2^{-k}}{h} < \frac{k}{h} \qquad (4.2.19)$$

(The maximizing $\mathscr{G}_1$ is just the "top" of $\mathscr{T}_h$.) We finally combine inequalities (4.2.18) and (4.2.19) to complete the proof via the inequality

$$\left(1 + \frac{1}{k}\right) \sum_{i=1}^{k} \mathbf{Wgt}(\mathbf{E}(\mathscr{G}_i)) - 1 \leqslant \frac{1}{h} \qquad \square$$

Lemma 4.2.16 and Theorem 4.2.17 combine to bound the edge-weighted mincing-width of complete binary trees from below.

APPLICATION 4.2.19. *For any integer h, the mincing-width* $\boldsymbol{M}_{\mathscr{T}_h}(k; \pi)$ *of the height-h complete binary tree* $\mathscr{T}_h$ *with edges weighted by the edge-weighting* π *is no smaller than*

$$1 - \left(1 + \frac{1}{h}\right)\left(1 - \frac{1}{k+1}\right) = \frac{1}{k+1}\left(1 + \frac{1}{h}\right) - \frac{1}{h}$$

4.3. Congestion Arguments for Bounding Separation-Width

4.3.1. Introduction to Congestion Arguments

This section is devoted to a general technique for bounding the M-separation-width of a graph $\mathscr{G}$ from below by bounding from above the congestions of embeddings of certain simple graphs into $\mathscr{G}$; cf. Sections 1.5 and 2.3 for background on graph embeddings. Because the technique we

develop here mandates using graphs with simple structure (mainly, with lots of symmetries), the task of bounding the congestion of the embedding, hence the M-separation-width of $\mathcal{G}$, often boils down to a simple counting argument.

The families of graphs we use to illustrate the use of congestion arguments to bound separation-width are

- "Rectangular" meshes-of-cliques (Section 4.3.2)
- "Rectangular" toroidal meshes (Section 4.3.3)
- Hypercubes of arbitrary base (Section 4.3.4)
- De Bruijn graphs of arbitrary base (Section 4.3.5)
- Butterfly graphs of arbitrary base (Section 4.3.6)
- Arbitrary binary trees (Section 4.3.7)

We close the section with two subsections that deviate from our general focus of deriving bounds for popular networks via congestion arguments. The first digressive subsection (Section 4.3.8) indicates a strength of the congestion technique by illustrating how to adapt the technique to bound the *I/O-bisection-widths* of FFT networks. The second digressive subsection (Section 4.3.9) exposes a weakness of the congestion technique. This section, which focuses on bounding the bisection-width of a less familiar family of networks, the so-called *product-shuffle graphs*, illustrates how the quality of the bounds one obtains using congestion arguments can vary substantially with the choice of the *auxiliary graph* one uses as the source graph in the congestion-bounding embedding. The section thus exposes an "artistic" component in the congestion technique.

The auxiliary graphs one uses in congestion arguments should have a structure that facilitates small-congestion embeddings into the target graphs whose separation-widths one wants to bound. The arguments we present in this section generally use cliques, complete bipartite graphs, and meshes-of-cliques as auxiliary graphs. The simple symmetric structures of these graphs allow us to argue that naive embeddings suffice to achieve embeddings with small congestion.

4.3.1.1. The Underpinnings of Congestion Arguments

The principle underlying congestion arguments is quite simple. Say that we are given a graph $\mathcal{G}$ whose M-separation-width $W_{\mathcal{G}}(M)$ we wish to bound from below. Say that we have access to an auxiliary graph $\mathcal{H}$ of the same size as $\mathcal{G}$ (i.e., $N(\mathcal{G}) = N(\mathcal{H})$) whose M-separation-width $W_{\mathcal{H}}(M)$ we know. If we can argue that there exists an embedding $\langle \alpha, \rho \rangle$ of $\mathcal{H}$ into $\mathcal{G}$ with congestion not exceeding C, then it follows that $W_{\mathcal{G}}(M) \geq$

$(1/C)W_{\mathscr{H}}(M)$. This inequality holds because the embedding $\langle \alpha, \rho \rangle$ allows us to view the act of partitioning $\mathscr{G}$ into two disjoint subgraphs having M and $N(\mathscr{G}) - M$ nodes, respectively, as simultaneously partitioning $\mathscr{H}$ into two disjoint subgraphs having the same respective sizes. With this view in mind, one can view the act of removing (or, cutting) any particular edge e of $\mathscr{G}$ as simultaneously removing (or, cutting) all edges of $\mathscr{H}$ that are routed over e by the routing map ρ. If we know that ρ never routes more than C edges of $\mathscr{H}$ over any edge of $\mathscr{G}$ (which information is implicit in our upper bound on the congestion of embeddings of $\mathscr{H}$ into $\mathscr{G}$), then we know that cutting an edge of $\mathscr{G}$ simultaneously cuts no more than C edges of $\mathscr{G}$. Since we also know that at least $W_{\mathscr{H}}(M)$ edges of $\mathscr{H}$ must be cut in order to effect the desired partition of $\mathscr{H}$, we can infer that at least $(1/C)W_{\mathscr{H}}(M)$ edges of $\mathscr{G}$ must be cut in order to effect the desired partition of $\mathscr{G}$. We now present this enabling bound formally, relying on the just-stated argument for justification.

LEMMA 4.3.1 (The Congestion Lemma). *Let $\mathscr{G}$ and $\mathscr{H}$ be graphs with $N(\mathscr{G}) = N(\mathscr{H})$. If there is an embedding of $\mathscr{H}$ into $\mathscr{G}$ with congestion $\leqslant C$, then for all integers M,*

$$W_{\mathscr{G}}(M) \geqslant \frac{1}{C} W_{\mathscr{H}}(M)$$

One consequence of Lemma 4.3.1 is that the quasi-isometric embeddings of Section 1.6 allow us to infer, with no further argumentation:

- Bounds on the separation-widths of paths from bounds on the separation-widths of cycles, and vice versa
- Bounds on the separation-widths of meshes from bounds on the separation-widths of toroidal meshes, and vice versa
- Bounds on the separation-widths of de Bruijn graphs from bounds on the separation-widths of shuffle-exchange graphs, and vice versa
- Bounds on the separation-widths of butterfly graphs from bounds on the separation-widths of cube-connected cycles graphs, and vice versa

In fact, the perceptive reader will note from our opening argument that, even when the target graph $\mathscr{G}$ and the auxiliary graph $\mathscr{H}$ are not the same size, one can infer information from the existence of small-congestion embeddings of $\mathscr{H}$ into $\mathscr{G}$: one need only adjust the ratio of the sizes of the subgraphs that result from partitioning $\mathscr{G}$ to a *range of ratios* that are consistent with the ratio of $\mathscr{G}$'s size to $\mathscr{H}$'s. Because the fixed ratio becomes

a range of ratios, this extension is most useful when M and $N - M$ are commensurate rather than when M is very small. When so adapted, Lemma 4.3.1 can be used, for instance, to obtain bounds on the separation-widths of butterfly graphs from bounds on the separation-widths of FFT graphs, and vice versa.

4.3.1.2. Complete Graphs

The challenge of taking an arbitrary graph $\mathcal{G}$ whose M-separation-width one wants to bound and finding a graph $\mathcal{H}$ that

- Has the same number of nodes as $\mathcal{G}$ (or of I/O nodes, when we want to focus on I/O-separation-width)
- Has known M-separation-width (or M-I/O-separation-width)
- Admits an embedding into $\mathcal{G}$ that has easily bounded congestion

seems quite formidable; but we shall see that the family of complete graphs or of complete bipartite graphs can supply such graphs $\mathcal{H}$ in many situations. And, as one establishes the M-separation-width for more and more graphs $\mathcal{G}$ (by whatever means) each of these graphs becomes a candidate auxiliary graph $\mathcal{H}$ for successive applications of the congestion technique.

4.3.1.2a. Separating Complete Graphs. We use the *cliques* $\mathcal{K}_n$ as our first family of auxiliary graphs (the $\mathcal{H}$ graphs) in congestion arguments, because cliques have easily analyzed embeddings into many families of graphs. Just as important, it is easy to determine the M-separation-widths of cliques exactly for all values of M.

PROPOSITION 4.3.2: *For all integers n and $M < n$, the M-separation-width $W_{\mathcal{K}_n}(M)$ of $\mathcal{K}_n$ is exactly $M(n - M)$.*

PROOF. When one partitions $\mathcal{K}_n$ into subgraphs of the desired sizes, one must "cut," for each node v in the resulting M-node subgraph of $\mathcal{K}_n$, the $M - n$ edges that connect v with all nodes in the resulting $(M - n)$-node subgraph of $\mathcal{K}_n$. The bound follows. □

4.3.1.2b. Separating Complete Bipartite Graphs. We use the *complete bipartite graphs* $\mathcal{K}_{n,n}$ as the second family of auxiliary graphs in congestion arguments. Complete bipartite graphs, too, have easily analyzed embeddings into many families of graphs. The only congestion arguments we know of which use $\mathcal{K}_{n,n}$ as an auxiliary graph use its bisection-width to bound the

minimum (I/O)-bisection-widths of other graphs. As we see now, the bisection-width of $\mathscr{K}_{n,n}$ is easy to determine exactly. (Determining $\mathscr{K}_{n,n}$'s arbitrary M-separation-widths is clerically somewhat complicated.)

PROPOSITION 4.3.3. *For all integers n, the bisection-width* $W_{\mathscr{K}_{n,n}}(n)$ *of* $\mathscr{K}_{n,n}$ *is exactly* $\frac{1}{2}n^2$.

PROOF. Consider an arbitrary partition of $\mathscr{K}_{n,n}$ into two equal-size subgraphs. Say that the "left" block of the partition contains m of $\mathscr{K}_{n,n}$'s "input" nodes and $n - m$ of its "output" nodes, so that the "right" block contains the complementary count. In order to effect this partition, one clearly must have "cut" the $m(n - m)$ edges that connect the "left" inputs with the "right" outputs, *plus* the $m(n - m)$ edges that connect the "right" inputs with the "left" outputs, for a total of $2m(n - m)$ "cut" edges. Easily, the total number of "cut" edges is minimized when $m = n - m = \frac{1}{2}n$. The bound follows. □

4.3.1.3. A Prospective on the Congestion Technique

It is useful to begin discussing the strengths and weaknesses of the congestion technique before we present even one use of the technique, so that the reader will be sure to notice certain features of the arguments that will appear imminently.

The approach has two major strengths. We have already noted the first strength in the preceding subsection: as one establishes lower bounds on the M-separation-widths of more and more graphs, by whatever means, each of these graphs becomes a candidate auxiliary graph $\mathscr{H}$ for successive applications of the congestion technique.

The second strength of the technique is more purely technical. When one succeeds in finding an auxiliary graph $\mathscr{H}$ which admits an embedding into the target graph $\mathscr{G}$ that is at once easily specified and analyzed (with respect to congestion), then the argument needed to bound $\mathscr{G}$'s separation-width reduces to simple combinatorial calculation and estimation. This is why we begin our study of congestion arguments by using the clique $\mathscr{K}_n$ as the auxiliary graph $\mathscr{H}$. Because of $\mathscr{K}_n$'s "total symmetry"—it is both node- and edge-transitive[7] —all embeddings of $\mathscr{K}_n$ into an n-node $\mathscr{G}$ have, for all practical purposes, the same node-assignment; the congestion of the embedding depends, therefore, only on the edge-routing procedure. Experience with congestion arguments suggests that a simple "greedy" edge-routing strategy often congests the edges of the target graph $\mathscr{G}$ (close to) uniformly, hence gives a good lower bound, at least on *bisection*-width (more about this imminently).

Countervailing its two advantages, the approach has two major weaknesses. First, experience demonstrates that the technique yields good bounds only when the target graph $\mathcal{G}$ is highly "uniform in structure," even when one embeds a simple graph such as $\mathcal{K}_n$ into $\mathcal{G}$. In the presence of such "uniformity," one can often find an embedding that spreads congestion rather evenly among the edges of $\mathcal{G}$, so it is not difficult to bound the congestion of the embedding; absent such "uniformity," however, one often finds the analysis of the embedding's congestion bogged down in an unilluminating proliferation of cases. To illustrate this point, the reader should compare the congestion argument in the proof of Theorem 4.3.6, which studies the *toroidal* mesh $\tilde{\mathcal{M}}_{m,n}$, with the analogous argument for the ordinary mesh $\mathcal{M}_{m,n}$ (which is left as an exercise). In Theorem 4.3.6 we have to distinguish only two different categories of edges in $\tilde{\mathcal{M}}_{m,n}$, namely, the row- and column-edges; in the ordinary mesh argument we would have to distinguish a number of categories that is proportional to the side-dimensions, m and n, of $\mathcal{M}_{m,n}$. We must emphasize that what we are calling "uniformity" here is not easily formalized. It is not simply some strong degree of symmetry such as node-transitivity: some graphs which have very few symmetries (de Bruijn graphs, for instance) are as easy to deal with by this technique as are toroidal meshes, which are node-transitive (compare Theorems 4.3.6 and 4.3.10); in contrast, some graphs which are more symmetric than the de Bruijn graph (the ordinary mesh, for instance) give rise to lengthy case analyses when one uses $\mathcal{K}_n$ as the auxiliary graph $\mathcal{H}$.

The second, and in some ways more serious, major weakness of the congestion technique is that it tends to be quite sensitive to the size of the separation parameter M—much more so, say, than the packing technique of Section 4.2. When M is roughly $N/2$, i.e., when one is seeking the *bisection-width* of $\mathcal{G}$ (or some approximation thereto), then the strategy of embedding $\mathcal{K}_n$ into $\mathcal{G}$ is quite effective, in that (empirically) it yields lower bounds on M-separation-width that are close to known upper bounds. In contrast, when M is much smaller than $N/2$—even when M is proportional to N—the bounds produced by embedding $\mathcal{K}_n$ into $\mathcal{G}$ are often much too small. The reader should note, for instance, that, while the congestion technique produces the exact *bisection*-width of the boolean hypercube (cf. Section 4.3.4)—and accomplishes this via a *much* simpler argument than that needed by the packing technique (cf. Section 4.2.3.1)—it produces a bound on the $2^{-(1+k)}N$-separation-width of the boolean hypercube that is more than a factor k too small, in contrast to the packing technique's exact bound. One observes a qualitatively similar situation with rectangular meshes: the congestion technique yields an excellent bound on the *bisection-width* of the mesh, with far less complexity, albeit somewhat less accuracy,

than the packing technique; yet it yields far inferior bounds on the mesh's general M-separation-width (cf. Sections 4.2.4.2 and 4.3.3).

The reason for the observed degradation in the quality of the congestion technique's bounds seems to be the following. When M is much smaller than N, much of the congestion the technique exposes arises from the routing of edges of $\mathcal{K}_n$ that *pass through* the M-node subgraph of $\mathcal{G}$ rather than spanning the subgraph and its $(N - M)$-node complement. However, the effectiveness of the congestion technique depends precisely on being able to attribute the observed congestion to these "spanning" edges of $\mathcal{K}_n$: an excessively high bound on congestion translates into an excessively low bound on M-separation-width.

Somewhat mitigating the preceding problem is the fact that, when M is very much smaller than N, one can sometimes regain rather good bounds on M-separation-width via the congestion technique by choosing an auxiliary graph $\mathcal{H}$ whose structure somewhat closely mirrors that of $\mathcal{G}$. Section 4.3.9 illustrates this possibility.

4.3.2. A Bound for the Mesh-of-Cliques

THEOREM 4.3.4 (The Congestion Theorem for $\mathcal{MC}_{m,n}$). *For all integers m and n, one can embed the clique*[8] $\mathcal{K}_{m \cdot n}$ *into the* $m \times n$ *mesh-of-cliques* $\mathcal{MC}_{m,n}$ *with congestion* $\leqslant \max(m, n)$.

PROOF. Because all embeddings of $\mathcal{K}_{m \cdot n}$ into $\mathcal{MC}_{m,n}$ have effectively the same node assignment, we need focus only on the routing of the edges of $\mathcal{K}_{m \cdot n}$ within $\mathcal{MC}_{m,n}$. Let us effect this routing greedily. Consider an arbitrary edge of $\mathcal{K}_{m \cdot n}$ whose endpoints are assigned to nodes (i, j) and (k, ℓ) of $\mathcal{MC}_{m,n}$, where $i \leqslant k$. We route this edge in $\mathcal{MC}_{m,n}$ via the length-2 path[9] $(i, j) \leftrightarrow (k, j) \leftrightarrow (k, \ell)$. The first, inter-row, edge of the path incurs congestion n, because it is used to route all edges from node (i, j) into row k of $\mathcal{MC}_{m,n}$. The second, intercolumn, edge of the path incurs congestion k, because it is used to route all edges from nodes (h, j), where $h \leqslant k$, to node (k, ℓ). Thus, the smallest upper bound we can claim on congestion for this embedding is $\max(m, n)$, whence the result. □

As we stated earlier, an upper bound on the congestion of the embedding yields a lower bound on the M-separation-width of the host graph in question. In this case Lemma 4.3.1, Proposition 4.3.2, and Theorem 4.3.4 combine to give a lower bound on the M-separation-width of $\mathcal{MC}_{m,n}$.

APPLICATION 4.3.5. *For all integers m, n, and* $M < mn$, *the M-separation-width* $W_{\mathcal{MC}_{m,n}}(M)$ *of the* $m \times n$ *mesh-of-cliques* $\mathcal{MC}_{m,n}$ *is no smaller*

than

$$M(N - M)/\max(m, n),$$

where $N = N(\mathcal{MC}_{m,n}) = mn$. *In particular, when* $m = n$, *so that* $N = N(\mathcal{MC}_{n,n}) = n^2$, *then*

$$W_{\mathcal{MC}_{n,n}}(M) \geq \frac{M(N - M)}{\sqrt{N}}$$

Finally, when M is of the form $M = \varphi N$ *for some* $0 < \varphi < 1$, *we have*

$$W_{\mathcal{MC}_{n,n}}(\varphi N) \geq \varphi(1 - \varphi)N^{3/2}$$

4.3.3. A Bound for Toroidal Meshes

In this subsection we focus on the *toroidal* mesh $\tilde{\mathcal{M}}_{m,n}$. As we noted in Subsection 4.3.1.1, the bounds we obtain here will be off from the bounds for the "ordinary" rectangular mesh $\mathcal{M}_{m,n}$ by no more than a factor of 2, because of the quasi-isometry of the two families. While we could obtain separation-width bounds for toroidal meshes by embedding $\mathcal{K}_{m \cdot n}$ into $\tilde{\mathcal{M}}_{m,n}$, we get better bounds, via simpler arguments, by embedding $\mathcal{MC}_{m,n}$ into $\tilde{\mathcal{M}}_{m,n}$ and invoking Application 4.3.5. This illustrates the recommended strategy of trying to design a congestion argument around an auxiliary graph whose structure is close to that of the target graph $\mathcal{G}$. We return to this point in more detail in Section 4.3.9.

THEOREM 4.3.6. (The Congestion Theorem for $\tilde{\mathcal{M}}_{m,n}$). *For all integers m and n, one can embed the* $m \times n$ *mesh-of-cliques* $\mathcal{MC}_{m,n}$ *into the* $m \times n$ *toroidal mesh* $\tilde{\mathcal{M}}_{m,n}$ *with congestion* $\leq \frac{1}{8}\max(m^2, n^2) + O(\max(m, n))$.

PROOF. We assign the nodes of $\mathcal{MC}_{m,n}$ to those of $\tilde{\mathcal{M}}_{m,n}$ via the identity map; i.e., we assign node (i, j) of $\mathcal{MC}_{m,n}$ to node (i, j) of $\tilde{\mathcal{M}}_{m,n}$. We route each edge of $\mathcal{MC}_{m,n}$ in $\tilde{\mathcal{M}}_{m,n}$ *greedily*, as follows. Consider first a column-edge of $\mathcal{MC}_{m,n}$, say the one between nodes (a, b) and (c, b). We route between these nodes in $\tilde{\mathcal{M}}_{m,n}$ by following a shortest path along column b. Similarly, for each row-edge of $\mathcal{MC}_{m,n}$, say the one between nodes (a, b) and (a, d), we route between these nodes in $\tilde{\mathcal{M}}_{m,n}$ by following a shortest path along row a.

Let us estimate the congestion of this embedding. Focus first on an arbitrary row-edge of $\tilde{\mathcal{M}}_{m,n}$:

$$(u, v) \leftrightarrow (u, v + 1 \bmod n) \tag{4.3.1}$$

This edge will be crossed by every routing-path in $\tilde{\mathcal{M}}_{m,n}$ that originates at a node (u, w) with $w \in \{(v + 1 - n/2 \bmod n), \ldots, v\}$ and terminates at a node (u, y) with $y \in \{v + 1, \ldots, (w + n/2 \bmod n)\}$. (The mandate to use shortest paths means that no path has length exceeding $n/2$.) One now calculates easily that the congestion on edge (4.3.1) is no greater than

$$\sum_{w=v+1-n/2}^{v} w - v + \frac{n}{2}$$

If we make the substitution $j = w - v + n/2$ in this summation, then we achieve the simply evaluated form

$$\sum_{j=1}^{n/2} j = \frac{n}{4}\left(\frac{n}{2} + 1\right) = \frac{n^2}{8} + O(n)$$

Symmetric reasoning shows that the congestion on an arbitrary column-edge of $\tilde{\mathcal{M}}_{m,n}$ is no greater than $m^2/8 + O(m)$ □

Lemma 4.3.1, Application 4.3.5, and Theorem 4.3.6 combine to give a lower bound on the M-separation-width of $\tilde{\mathcal{M}}_{m,n}$.

APPLICATION 4.3.7. *For all integers m and n, the M-separation-width* $W_{\tilde{\mathcal{M}}_{m,n}}(M)$ *of the* $m \times n$ *toroidal mesh* $\tilde{\mathcal{M}}_{m,n}$ *is no smaller than*

$$\frac{8M(N - M)}{\max(m^3, n^3)} + l.o.t.$$

where $N = N(\tilde{\mathcal{M}}_{m,n}) = mn$. *In particular, when* $m = n$, *so* $N = N(\tilde{\mathcal{M}}_{n,n}) = n^2$, *then we have*

$$W_{\tilde{\mathcal{M}}_{n,n}}(M) \geq \frac{8M(N - M)}{N^{3/2}} + l.o.t.$$

When, additionally, M is of the form $M = \varphi N$ *for some* $0 < \varphi < 1$, *then*

$$W_{\tilde{\mathcal{M}}_{n,n}}(\varphi N) \geq 8\varphi(1 - \varphi)\sqrt{N} + l.o.t.$$

Note that the bound of Application 4.3.7 for the case $M = \varphi N$ is too small by a factor proportional to $\sqrt{\varphi(1 - \varphi)}$ when φ is small; cf. Application 4.2.9.

4.3.4. A Bound for Hypercubes

In order to emphasize the flexibility of the congestion technique, we apply it here—with no significant increase in technical complexity—to hypercubes of *arbitrary base b*, rather than just to the more familiar boolean hypercubes. This flexibility is one of the major advantages of congestion arguments over the packing arguments of Section 4.2. To appreciate this point, the reader should compare our treatment of arbitrary base-b hypercubes here with the arguments for base-2 and base-3 (ternary) hypercubes in Section 4.2; the packing argument for ternary hypercubes is materially more complicated clerically than the packing argument for binary hypercubes, and the clerical complexity promises to grow quite fast with the base of the hypercube.

We obtain our bound by analyzing embeddings of $\mathcal{K}_{b^n}$ into $\mathcal{Q}_n(b)$.

THEOREM 4.3.8 (The Congestion Theorem for $\mathcal{Q}_n(b)$). *For all integers b and n, one can embed the clique $\mathcal{K}_{b^n}$ into the base-b n-dimensional hypercube $\mathcal{Q}_n(b)$, with congestion $\leqslant b^{n-1} = (1/b)N(\mathcal{Q}_n(b))$.*

PROOF. As with all surjective (onto) embeddings of cliques into like-sized graphs, we need concentrate only on crafting an edge-routing for the embedding that witnesses the claimed bound. We achieve this simply by routing each edge of $\mathcal{K}_{b^n}$ within $\mathcal{Q}_n(b)$ *greedily*, by rewriting the lexicographically smaller of the edges' endpoints dimension by dimension, in increasing order of dimensions (i.e., left to right). For illustration, focus on the edge of $\mathcal{K}_{b^n}$ whose endpoints reside (under the embedding) at nodes $x = \xi_0\xi_1 \cdots \xi_{n-1}$ and $y = \eta_0\eta_1 \cdots \eta_{n-1}$ of $\mathcal{Q}_n(b)$, where, with no loss of generality, x precedes y in lexicographic order. We route this edge via the following path in $\mathcal{Q}_n(b)$:

$$
\begin{aligned}
x &= \xi_0\xi_1 \cdots \xi_{n-2}\xi_{n-1} \\
&\leftrightarrow \eta_0\xi_1 \cdots \xi_{n-2}\xi_{n-1} \\
&\leftrightarrow \eta_0\eta_1 \cdots \xi_{n-2}\xi_{n-1} \\
&\;\;\vdots \qquad\quad \vdots \\
&\leftrightarrow \eta_0\eta_1 \cdots \eta_{n-2}\xi_{n-1} \\
&\leftrightarrow \eta_0\eta_1 \cdots \eta_{n-2}\eta_{n-1} \\
&= y
\end{aligned}
$$

Toward the end of analyzing the worst-case congestion of our embedding, focus on an arbitrary edge

$$x\beta y \leftrightarrow x\gamma y \tag{4.3.2}$$

of $\mathcal{Q}_n(b)$, where $x \in Z_b^k$, $y \in Z_b^{n-k-1}$ for some $k \in Z_n$ and $\beta, \gamma \in Z_b$. Under the chosen routing regimen, edge (4.3.2) is crossed by all and only paths in $\mathcal{Q}_n(b)$ that originate at a node of the form $w\beta y$, where $w \in Z_b^k$, and that terminate at a node of the form $x\gamma z$, where $z \in Z_b^{n-k-1}$. Because there are precisely b^{n-1} such paths, the theorem follows. □

Lemma 4.3.1, Proposition 4.3.2, and Theorem 4.3.8 combine to give a lower bound on the M-separation-width of $\mathcal{Q}_n(b)$.

APPLICATION 4.3.9. *For all integers b and n, the M-separation-width* $W_{\mathcal{Q}_n(b)}(M)$ *of the base-b n-dimensional hypercube* $\mathcal{Q}_n(b)$ *is no smaller than*

$$b(1 - M/N)M,$$

where $N = N(\mathcal{Q}_n(b)) = b^n$. *In particular, when M is of the form* $M = \varphi N$ *for some* $0 < \varphi < 1$, *this bound simplifies to*

$$W_{\mathcal{Q}_n(b)}(\varphi N) \geqslant \varphi(1 - \varphi)bN$$

Note that the bound of Application 4.3.9 for the case $M = \varphi N$ is too small by a factor proportional to $(\varphi - 1)/\log \varphi$ when φ is smaller than 1/2; cf. Application 4.2.5.

4.3.5. A Bound for de Bruijn Graphs

As in Subsection 4.3.4, we emphasize the flexibility of the congestion argument technique by applying it to de Bruijn graphs of arbitrary base b rather than just to the more familiar binary (base-2) de Bruijn graphs.

We obtain our bound by analyzing embeddings of $\mathcal{K}_{b^n}$ into $\mathcal{D}_n(b)$.

THEOREM 4.3.10 (The Congestion Theorem for $\mathcal{D}_n(b)$). *For all integers b and n, one can embed the clique* $\mathcal{K}_{b^n}$ *into the base-b order-n de Bruijn graph* $\mathcal{D}_n(b)$, *with congestion* $\leqslant nb^{n-1} = (n/b)N(\mathcal{D}_n(b))$.

PROOF. As with all onto embeddings of cliques, we need concentrate only on crafting an edge-routing for the embedding that witnesses the claimed bound. As in Section 4.3.4, we route each edge of $\mathcal{K}_{b^n}$ within $\mathcal{D}_n(b)$ *greedily* by rewriting the lexicographically smaller of the endpoints from left to right. For illustration, focus on the edge of $\mathcal{K}_{b^n}$ whose endpoints reside (under the embedding) at nodes $x = \xi_0\xi_1 \cdots \xi_{n-1}$ and $y = \eta_0\eta_1 \cdots \eta_{n-1}$ of $\mathcal{D}_n(b)$, where, with no loss of generality, x precedes y in lexicographic order.

We route this edge via the following path in $\mathscr{D}_n(b)$:

$$\begin{aligned} x &= \xi_0\xi_1\xi_2 \cdots \xi_{n-2}\xi_{n-1} \\ &\leftrightarrow \xi_1\xi_2 \cdots \xi_{n-2}\xi_{n-1}\eta_0 \\ &\leftrightarrow \xi_2 \cdots \xi_{n-2}\xi_{n-1}\eta_0\eta_1 \\ &\ \ \vdots \qquad \vdots \\ &\leftrightarrow \xi_{n-1}\eta_0\eta_1 \cdots \eta_{n-2} \\ &\leftrightarrow \eta_0\eta_1 \cdots \eta_{n-2}\eta_{n-1} \\ &= y \end{aligned}$$

To the end of analyzing the worst-case congestion of this embedding, focus on an arbitrary edge

$$\beta z \leftrightarrow z\gamma \tag{4.3.3}$$

of $\mathscr{D}_n(b)$, where $z \in Z_b^{n-1}$ and $\beta, \gamma \in Z_b$. In order to estimate the congestion on edge (4.3.3), let us focus on the congestion at a given fixed but arbitrary step k in the length-n routing procedure. (For definiteness, let "step k" be the step in which symbol ξ_k is rewritten in our sample path.) We can characterize the routing-paths in $\mathscr{D}_n(b)$ that correspond to edges of $\mathscr{K}_{b^n}$ that are routed across edge (4.3.3) in step k as follows. These paths are precisely those that

- Originate at a node of $\mathscr{D}_n(b)$ of the form $w\beta u$, where $w \in Z_b^k$, and u is the length-$(n - k - 1)$ prefix of z.
- Terminate at a node of $\mathscr{D}_n(b)$ of the form $w'\gamma u'$, where w' is the length-k-rewrite of w, hence is the length-k suffix of z, and u' is the length-$(n - k - 1)$ rewrite of u.

There are clearly b^{n-1} such paths. It follows that over the n steps of the routing, corresponding to the n possible values of k, the congestion across edge (4.3.3) is at most nb^{n-1}. The theorem follows. □

NOTE. The factor-of-n difference in congestion we have observed between embeddings of $\mathscr{K}_{b^n}$ into $\mathscr{Q}_n(b)$ (Theorem 4.3.8) and into $\mathscr{D}_n(b)$ (Theorem 4.3.10) is due to the different ways the two host-graphs' edges rewrite their string-nodes. Hypercubes rewrite their string-nodes "in place," so when we cross edge (u, v) of $\mathscr{Q}_n(b)$ in the midst of a path we know precisely which string-position of the path's source node we are rewriting.

(It is the position in which strings u and v differ.) In contrast, de Bruijn graphs rewrite their strings cyclically, so when we cross edge (u, v) of $\mathcal{D}_n(b)$ in the midst of a path, we do not know which string-position of the path's source node we are rewriting, unless we know which position the current edge occupies in the path we are traversing. The resulting "uncertainty" about the identity of the source and destination nodes of the path translates into the additional congestion.

Lemma 4.3.1, Proposition 4.3.2, and Theorem 4.3.10 combine to give a lower bound on the M-separation-width of $\mathcal{D}_n(b)$.

APPLICATION 4.3.11. *For all integers b and n, the M-separation-width $W_{\mathcal{D}_{n(b)}}(M)$ of the base-b order-n de Bruijn graph $\mathcal{D}_n(b)$ is no smaller than*

$$b\left(1 - \frac{M}{N}\right)\frac{M}{n}$$

where $N = N(\mathcal{D}_n(b)) = b^n$. In particular, when M is of the form $M = \varphi N$ for some $0 < \varphi < 1$, this bound simplifies to

$$W_{\mathcal{D}_{n(b)}}(\varphi N) \geqslant \varphi(1 - \varphi)\frac{bN}{\log_b N}$$

4.3.6. A Bound for Butterfly Graphs

As in the previous two subsections, we apply the congestion-argument technique, at no additional cost, to butterfly graphs of arbitrary base rather than just to the more familiar binary (base-2) butterfly graphs.

We obtain our bound by analyzing embeddings of $\mathcal{K}_{nb^n}$ into $\mathcal{B}_n(b)$.

THEOREM 4.3.12 (The Congestion Theorem for $\mathcal{B}_n(b)$). *For all integers $b \geqslant 2$ and n, one can embed the clique $\mathcal{K}_{nb^n}$ into the base-b order-n butterfly graph $\mathcal{B}_n(b)$, with congestion no greater than*

$$\left(\frac{n}{4} + \frac{n}{2b} - \frac{1}{2}\right)nb^n \leqslant \frac{b+2}{4b}\,nN(\mathcal{B}_n(b))$$

PROOF. Once again, we need concentrate only on crafting an edge-routing for the embedding that witnesses the claimed bound. We route each edge of $\mathcal{K}_{nb^n}$ within $\mathcal{B}_n(b)$ as follows. Focus on an arbitrary edge of $\mathcal{K}_{nb^n}$, the images of whose endpoints reside at nodes $u = (\ell, w)$ and $v = (\ell', w')$ of $\mathcal{B}_n(b)$, where $\ell, \ell' \in Z_n$ and $w, w' \in Z_b^n$. Say, for definiteness, that node u

precedes node v in lexicographic order, by which we mean that *either* $\ell = \ell'$ and w precedes w' in the natural lexicographic order of strings *or* $\ell < \ell'$.[10] We route this edge in two phases:

1. *Rewriting*: We follow a sequence of n edges to rewrite PWL string w to PWL string w', i.e., to go from node (ℓ, w) to node (ℓ, w'). At each level k of $\mathscr{B}_n(b)$, we choose the unique edge to level $k + 1 \bmod n$ that replaces the kth symbol of w by the kth symbol of w'. Specifically, if the kth symbol of w differs from that of w', then we traverse the cross edge, thereby rewriting the symbol; if the symbols are identical, then we traverse the straight edge.

2. *Positioning*. We follow a shortest path of $\leqslant n/2$ straight edges to go from node (ℓ, w') to the target node (ℓ', w').

Toward the end of bounding from above the worst-case congestion of our embedding, let us focus on an arbitrary edge of $\mathscr{B}_n(b)$:

$$(k, x\beta y) \leftrightarrow (k + 1 \bmod n, x\gamma y) \tag{4.3.4}$$

If edge (4.3.4) is a cross edge (i.e., if $\beta \neq \gamma$), then it is traversed only during the rewriting phase of edge-routing; hence, the congestion on the edge is just the "rewriting congestion." If edge (4.3.4) is a straight edge (i.e., if $\beta = \gamma$), then it is (potentially) traversed during both phases of edge-routing; hence the congestion on the edge is the sum of the "rewriting congestion" and the "positioning congestion." We now bound from above the congestion across edge (4.3.4) incurred during each of the two routing phases.

In order to estimate the "rewriting congestion" on edge (4.3.4), let us assume for the moment that the edge is the jth one along the rewriting path. Since j can assume any of n values, the "total rewriting congestion" along the edge will be a factor of n greater than the "jth symbol rewriting congestion." (We are using here the same bounding technique as when we analyzed the embedding of $\mathscr{K}_{b^n}$ into $\mathscr{D}_n(b)$ in Section 4.3.5.) Now, the paths in $\mathscr{B}_n(b)$ that correspond to edges of $\mathscr{K}_{nb^n}$ that are routed across edge (4.3.4) in the jth step of the rewriting path are precisely one-half[11] of those that (doing all arithmetic modulo n)

- Originate at one of the b^j nodes of $\mathscr{B}_n(b)$ of the form $(k - j, w)$, where the PWL string $w \in Z_b^n$ agrees with $x\beta y$ in all digit-positions, except perhaps positions $k - j, k - j + 1, k + 1, \ldots, k - 1$.
- Terminate at one of the nb^{n-j-1} nodes of $\mathscr{B}_n(b)$ of the form (ℓ, z), where $\ell \in Z_n$, and where $z \in Z_b^n$ agrees with $x\gamma y$ in all digit-positions except perhaps positions $k + 1, k + 2, \ldots, k - j - 1$.[12]

By this accounting, no more than $\frac{1}{2}nb^{n-1}$ paths cross edge (4.3.4) during step j of the rewriting phase of our embedding's edge-routing. It follows that the

worst possible congestion attributable to the entire rewriting phase is no greater than $\frac{1}{2}n^2b^{n-1}$. (In fact, one can show that this congestion is optimal.)

In order to estimate the "positioning congestion" on edge (4.3.4) (which is incurred only when it is a straight edge, i.e., when $\beta = \gamma$), we note that the paths in $\mathscr{B}_n(b)$ that correspond to edges of $\mathscr{K}_{nb^n}$ that are routed across edge (4.3.4) while being positioned are precisely those that satisfy one of the following alternatives (doing all arithmetic modulo n):

- The path originates at one of the $\lfloor n/2 \rfloor b^n$ nodes of $\mathscr{B}_n(b)$ of the form (m, w), where $w \in Z_b^n$ and $m \in \{k + 1 - \lfloor n/2 \rfloor, \ldots, k - 1, k\}$, and terminates at one of the $m - k + \lfloor n/2 \rfloor$ nodes of the form $(k + i, x\gamma y)$, where $i \in \{1, 2, \ldots, m - k + \lfloor n/2 \rfloor\}$.
- The path originates at one of the $\lfloor n/2 \rfloor b^n$ nodes of $\mathscr{B}_n(b)$ of the form (m, w), where $w \in Z_b^n$ and $m \in \{k + 1, k + 2, \ldots, k + \lfloor n/2 \rfloor\}$, and terminates at one of the $k + \lfloor n/2 \rfloor - m$ nodes of the form $(k - i - 1, x\gamma y)$, where $i \in \{1, 2, \ldots, k - m + \lfloor n/2 \rfloor\}$.

These two cases being symmetric, each comprises the same number of paths, namely,

$$b^n \sum_{m=k+1}^{k+\lfloor n/2 \rfloor} \left(k + \left\lfloor \frac{n}{2} \right\rfloor - m\right) = b^n \sum_{j=1}^{\lfloor n/2 \rfloor} \left(\left\lfloor \frac{n}{2} \right\rfloor - j\right) = \frac{1}{2}\left\lfloor \frac{n}{2} \right\rfloor \left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right) b^n$$

It follows that the worst possible congestion incurred during the positioning phase of our embedding's edge-routing is

$$\left\lfloor \frac{n}{2} \right\rfloor \left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right) b^n \leqslant \left(\frac{n}{4} - \frac{1}{2}\right) nb^n$$

Combining the two sources of congestion on the straight edges of $\mathscr{B}_n(b)$ (which are the more congested edges), the maximum congestion along any edge of $\mathscr{B}_n(b)$ under our embedding is no greater than

$$\left(\frac{n}{4} + \frac{n}{2b} - \frac{1}{2}\right) nb^n$$

This completes the proof of the theorem. □

Lemma 4.3.1, Proposition 4.3.2, and Theorem 4.3.12 combine to give a lower bound on the M-separation-width of $\mathscr{B}_n(b)$.

APPLICATION 4.3.13. *For all integers* $b \geq 2$ *and* n, *the M-separation-width* $W_{\mathscr{B}_n(b)}(M)$ *of the base-b order-n butterfly graph* $\mathscr{B}_n(b)$ *is no smaller than*

$$\frac{4b}{b+2}\left(1-\frac{M}{N}\right)\frac{M}{n}$$

where $N = N(\mathscr{B}_n(b)) = nb^n$. *In particular, when M is of the form* $M = \varphi N$ *for some* $0 < \varphi < 1$, *this bound simplifies to*

$$W_{\mathscr{B}_n(b)}(\varphi N) \geq \frac{4b}{b+2}\varphi(1-\varphi)\frac{N}{\log_b N}$$

Since cutting the 2^n edges between any two consecutive levels of $\mathscr{B}_n$ bisects the graph, one sees that the bound of Application 4.3.13 is "the right answer" when $b = 2$ and $\varphi = 1/2$. The quality of the bound deteriorates quickly with the distance of φ from 1/2.

4.3.7. A Bound for Arbitrary Binary Trees

It is well recognized in the world of data structures that arbitrary binary trees can "mimic" the shape of arbitrary trees; cf. Chapter 2.3.2 of Knuth [1973]. One can exploit a formal analogue of this fact — which resides in small-congestion embeddings of complete trees into (usually noncomplete) binary trees — to show that, in the worst case, cutting binary trees into two pieces of prescribed sizes requires cutting logarithmically many edges of the tree. The arguments that yield the bounds share one conceptual framework, differing only in clerical details. Therefore, we illustrate the argument here with just one example that uses complete ternary trees as the guest graph, hence uses fractions of the form $1/(3^k - 1)$ as the "difficult" fractions to achieve.

THEOREM 4.3.14 (The Congestion Theorem for Arbitrary Binary Trees). *For every integer h there is a binary tree* $\mathscr{T}^{(h)}$ *having* $N = \frac{1}{2}(3^{h+1} - 1)$ *nodes, such that the height-h complete ternary tree* $\mathscr{T}_h(3)$ *can be embedded into* $\mathscr{T}^{(h)}$ *with congestion* 3.

PROOF. We merely sketch the straightforward proof. The advertised embedding is described most easily by imagining that we are embedding $\mathscr{T}_h(3)$ inductively, level by level, into the *infinite* rooted complete binary tree $\mathscr{T}^*$ (which is the union of all finite complete binary trees). The desired finite binary tree $\mathscr{T}^{(h)}$ will then be the finite "prefix" of $\mathscr{T}^*$ to which nodes of $\mathscr{T}_h(3)$ have been assigned.

Table 4.3-1. Illustrating Case 1 of the Embedding of $\mathcal{T}_h(3)$ into $\mathcal{T}^*$

Nodes of $\mathcal{T}_h(3)$	Images in $\mathcal{T}^*$
w	z
x	$z0$
y	$z1$
$w0$	$z01$
$w1$	$z10$
$w2$	$z11$
$x0$	$z00$
$x1$	$z000$
$x2$	$z001$
$y0$	$z100$
$y1$	$z101$
$y2$	$z110$

We assign the root of $\mathcal{T}_h(3)$, which is the unique node at level 0 of the tree, to the root of $\mathcal{T}^*$. We then proceed locally and inductively, distinguishing three mutually exclusive and collectively exhaustive cases. Let us assume that we have embedded all **level-ℓ** nodes of $\mathcal{T}_h(3)$ into $\mathcal{T}^*$.

Case 1. Some three **level-ℓ** nodes of $\mathcal{T}_h(3)$ are clustered together in a 2-level subtree of $\mathcal{T}^*$.

The layout in $\mathcal{T}^*$ of the **level-ℓ** nodes of $\mathcal{T}_h(3)$, call them w, x, and y, and the prescribed layout of their nine children are described schematically in Table 4.3-1.

Case 2. Case 1 does not hold, but some two **level-ℓ** nodes of $\mathcal{T}_h(3)$ are clustered together in a 2-level subtree of $\mathcal{T}^*$.

The layout in $\mathcal{T}^*$ of the **level-ℓ** nodes of $\mathcal{T}_h(3)$, call them w and x, and the prescribed layout of their six children are described schematically in Table 4.3-2.

Case 3. Cases 1 and 2 do not hold.

The layout in $\mathcal{T}^*$ of the "isolated" **level-ℓ** node of $\mathcal{T}_h(3)$, call it w, and the prescribed layout of its three children are described schematically in Table 4.3-3.

The local nature of this embedding allows us to analyze its congestion by merely analyzing the local subembeddings presented in these three tables. One finds that no edge of $\mathcal{T}^{(h)}$ suffers more traffic than is caused by allowing one node of $\mathcal{T}_h(3)$, say w, to cross the edge to reach one of its children, while another node of $\mathcal{T}_h(3)$, say y, crosses the edge to reach two of its children: the edge connecting binary nodes $z1$ and $z10$ in Table 4.3-1 illustrates this situation. Details are left as an exercise. □

Table 4.3-2. Illustrating Case 2 of the Embedding of $\mathcal{T}_h(3)$ into $\mathcal{T}^*$

Nodes of $\mathcal{T}_h(3)$	Images in $\mathcal{T}^*$
w	z
x	$z0$
$w0$	$z1$
$w1$	$z10$
$w2$	$z11$
$x0$	$z00$
$x1$	$z01$
$x2$	$z000$

The embedding of Theorem 4.3.14 is onto each image tree $\mathcal{T}_h$; therefore, Theorem 4.3.14, Lemma 4.3.1, and Theorem 4.4.1 (in Section 4.4) combine to show that logarithmically many edges of $\mathcal{T}^{(h)}$ must be cut in order to cut the tree (roughly) in half.

APPLICATION 4.3.15. *For every integer h there is a binary tree* $\mathcal{T}^{(h)}$ *having* $N = \frac{1}{2}(3^{h+1} - 1)$ *nodes, whose bisection-width satisfies*

$$W_{\mathcal{T}^{(h)}}\left(\frac{N}{2}\right) > \frac{1}{6}h - \frac{1}{36} = \Omega(\log N)$$

4.3.8. A Bound on I/O-Bisections of FFT Networks

We now illustrate how congestion arguments can be used to obtain bounds on the *I/O-separation-widths* of graphs, as well as on their full separation-widths. For simplicity, we focus just on a bound on I/O-*bisection*-width; the extension to arbitrary separations should be conceptually transparent. To add interest to the exercise, we consider an interesting

Table 4.3-3. Illustrating Case 3 of the Embedding of $\mathcal{T}_h(3)$ into $\mathcal{T}^*$

Nodes of $\mathcal{T}_h(3)$	Images in $\mathcal{T}^*$
w	z
$w0$	$z0$
$w1$	$z1$
$w2$	$z00$

family of networks that are naturally endowed with designated sets of input and output nodes, namely, FFT networks.

We obtain our bound by letting $N = 2^n$ and analyzing the embedding of the $N \times N$ complete bipartite graph $\mathcal{K}_{N,N}$ (which has N input nodes and N output nodes) into the order-n FFT network $\mathcal{F}_n$.

THEOREM 4.3.16 (The I/O-Congestion Theorem for $\mathcal{F}_n$). *For all integers $N = 2^n$, one can embed the complete bipartite graph $\mathcal{K}_{N,N}$ into the order-n FFT network $\mathcal{F}_n$, with congestion $\leqslant 2^{n-1} = \frac{1}{2}N$.*

PROOF. Because of the symmetry of complete bipartite graphs, our embedding can employ any node-assignment α that assigns the N "input nodes" (resp., the N "output nodes") of $\mathcal{K}_{N,N}$ to the N input nodes (resp., the N output nodes) of $\mathcal{F}_n$ in a one-to-one, onto fashion. Given any such α, we route the edges of $\mathcal{K}_{N,N}$ within $\mathcal{F}_n$ *greedily* in the following sense. We route each edge (u, v) of $\mathcal{K}_{N,N}$ by rewriting the PWL string of FFT-input node $\alpha(u)$ from left to right, level by level within $\mathcal{F}_n$, to the PWL string of FFT-output node $\alpha(v)$. This procedure is

- *Possible* because edges between levels i and $i + 1$ of $\mathcal{F}_n$ rewrite bit-position i of the level-i node's PWL string,
- *Effective* because one of u, v is an input node of $\mathcal{K}_{N,N}$, while the other is an output node, and α preserves these designations within $\mathcal{F}_n$.

To the end of analyzing the worst-case congestion of this embedding, focus on an arbitrary edge of $\mathcal{F}_n$:

$$(\langle \ell, x\beta y\rangle, \langle \ell + 1, x\gamma y\rangle) \tag{4.3.5}$$

where $\ell \in Z_n$, $x \in Z_2^{\ell}$, $y \in Z_2^{n-\ell-1}$, and $\beta, \gamma \in Z_2$. We can characterize the routing-paths in $\mathcal{F}_n$ that correspond to edges of $\mathcal{K}_{n,n}$ that are routed across edge (4.3.5), as follows. These paths are precisely those that

- Originate at one of the 2^{ℓ} input nodes of $\mathcal{F}_n$ of the form $\langle 0, w\beta y\rangle$, where $w \in Z_2^{\ell}$
- Terminate at one of the $2^{n-\ell-1}$ output nodes of $\mathcal{F}_n$ of the form $\langle n, x\gamma z\rangle$, where $z \in Z_2^{n-\ell-1}$

There are clearly 2^{n-1} such paths. The theorem follows. □

We now challenge the reader to derive an analogue of Lemma 4.3.1 that applies to I/O-separations. As a hint, we remark that the onto embeddings demanded by the lemma are replaced by embeddings that map the input nodes of the source graph onto the input nodes of the target graph and the output nodes of the former onto the output nodes of the latter. Invoking this analogue, in conjunction with Proposition 4.3.3 and Theorem 4.3.16, one obtains the following lower bound on the I/O-bisection-width of $\mathscr{F}_n$.

APPLICATION 4.3.17. *For all integers n, the I/O-bisection-width of the order-n FFT network* $\mathscr{F}_n$ *is*

$$W_{\mathscr{F}_n}^{(\mathrm{I/O})}(N) \geqslant 2^n = N$$

4.3.9. A Bound for Product-Shuffle Graphs

The final family of graphs we discuss in this section are the *product-shuffle* graphs that are studied in Rosenberg [1992]. Although these graphs are not as familiar as the others we have been considering, their rich structure makes them a good illustration of the types of choices one must make when crafting a congestion argument. In particular, the lesson from this subsection is that one must seek an embedding that congests the edges of the host graph (more or less) uniformly in order to obtain good lower bounds on M-separation-width from a congestion argument.

The *order-*(m, n) *product-shuffle graph* $\mathscr{PS}_{m,n}$ is the product of de Bruijn graphs, $\mathscr{PS}_{m,n} = \mathscr{D}_m \times \mathscr{D}_n$. Because our goal here is to focus on congestion arguments rather than on this family of graphs, we look only at the binary (base-2) family, abjuring the generality we have sought in other subsections.

If one seeks to bound the M-separation-width of $\mathscr{PS}_{m,n}$ from below, one has access to the following easily verified facts about embeddings into $\mathscr{PS}_{m,n}$.

THEOREM 4.3.18 (The Congestion Theorem for $\mathscr{PS}_{m,n}$). (a) *For all integers m and n, one can embed* $\mathscr{D}_{m+n}$ *into* $\mathscr{PS}_{m,n}$, *with congestion* 2.

(b) *For all integers m and n, one can embed* $\mathscr{MC}_{2^m,2^n}$ *into* $\mathscr{PS}_{m,n}$, *with congestion* $\max(m2^{m-1}, n2^{n-1})$.

PROOF. We merely sketch the proof.

(a) The desired embedding of $\mathscr{D}_{m+n}$ into $\mathscr{PS}_{m,n}$ assigns node

$$xy \in \mathbf{N}(\mathscr{D}_{m+n})$$

where $x \in Z_2^m$ and $y \in Z_2^n$, to node $(x, y) \in \mathbf{N}(\mathscr{PS}_{m,n})$. The edge-routing is the

natural one: for all $w \in Z_2^{m-1}$, $z \in Z_2^{n-1}$ and $\beta, \gamma, \delta \in Z_2$, the edge-routing maps each edge

$$\beta w \delta z \leftrightarrow w \delta z \gamma$$

of $\mathscr{D}_{m+n}$ to the length-2 path

$$(\beta w, \delta z) \leftrightarrow (w\delta, \delta z) \leftrightarrow (w\delta, z\gamma)$$

in $\mathscr{PS}_{m,n}$.

(b) Note first that each row-clique (resp., each column-clique) of $\mathscr{MC}_{2^m,2^n}$ is isomorphic to the clique $\mathscr{K}_{2^n}$ (resp., the clique $\mathscr{K}_{2^m}$). Our embedding of $\mathscr{MC}_{2^m,2^n}$ into $\mathscr{PS}_{m,n}$ exploits these isomorphisms by embedding each row-clique (resp., each column-clique) of $\mathscr{MC}_{2^m,2^n}$ into the corresponding row-de Bruijn graph (resp., the corresponding column-de Bruijn graph) of $\mathscr{PS}_{m,n}$, in just the way that $\mathscr{K}_{2^n}$ is embedded into $\mathscr{D}_n$ in Theorem 4.3.10. Details are left to the reader. □

We now compare the bounds on the separation-width of $\mathscr{PS}_{m,n}$ that we achieve via Theorem 4.3.18(a), on the one hand, and Theorem 4.3.18(b), on the other.

By combining Theorem 4.3.18(a) with Lemma 4.3.1 and Application 4.3.11, one obtains the following bound on the M-separation-width of $\mathscr{PS}_{m,n}$.

APPLICATION 4.3.19. *For all integers m and n, the M-separation-width of the order-(m, n) product-shuffle graph $\mathscr{PS}_{m,n}$ is*

$$W_{\mathscr{PS}_{m,n}}(M) \geqslant \frac{M(N-M)}{N \log N}$$

where $N = N(\mathscr{PS}_{m,n}) = 2^{m+n}$.

By combining Theorem 4.3.18(b) with Lemma 4.3.1 and Application 4.3.5, one obtains the following bound on the M-separation-width of $\mathscr{PS}_{m,n}$.

APPLICATION 4.3.20. *For all integers m and $n \geqslant m$, the M-separation-width of the order-(m, n) product-shuffle graph $\mathscr{PS}_{m,n}$ is*

$$W_{\mathscr{PS}_{m,n}}(M) \geqslant \frac{M(N-M)}{n^2 2^{n-1}}$$

Except when m and n differ dramatically (specifically, when one is exponentially larger than the other) the bound of Application 4.3.20 is much larger (hence, better) than that of Application 4.3.19. While we cannot explain this difference in quality definitively, we believe that it is due to the following fact. In the embedding of $\mathcal{MC}_{2^m,2^n}$ into $\mathcal{PS}_{m,n}$ [Theorem 4.3.18(b)], the congestion along each "row" (resp., along each "column") is attributable to edges of $\mathcal{MC}_{2^m,2^n}$ both of whose endpoints reside (under the embedding) within that "row" (resp., within that "column"). There is no such assurance with the embedding of $\mathcal{D}_{m+n}$ into $\mathcal{PS}_{m,n}$ [Theorem 4.3.18(a)]. This means, in effect, that the weakness of the embedding of $\mathcal{D}_{m+n}$ into $\mathcal{PS}_{m,n}$ *vis-à-vis* congestion arguments is basically the same as the general weakness of congestion arguments in bounding M-separation-widths when M differs considerably from N.

4.4. A Technique for Complete Trees

As we mentioned in Section 4.1, trees present a special problem for techniques that bound separation-width from below. The packing technique of Section 4.2 produces poor bounds when applied to *any* trees, because an M-node forest (which is what an M-node subgraph of a tree always is) contains at most $M - 1$ edges, no matter what the "shape" of the forest. This means, for instance, that a packing argument will give the same bound for bisecting a complete binary tree (which can be done by cutting a single edge) as for bisecting a complete ternary tree (which we shall see in this section requires cutting logarithmically many edges). The congestion technique of Section 4.3 produces poor bounds when applied to complete trees, because of what one might call the "nonuniform communication structure" of such trees. The source of the problem is that just over half of the nodes of a complete binary tree $\mathcal{T}$ are leaves. (Indeed, this is true of any rooted binary tree in which each nonleaf node has two children.) It is, therefore, likely that, in any embedding of any auxiliary graph $\mathcal{H}$ into $\mathcal{T}$, many edges of $\mathcal{H}$ will have both endpoints residing at leaves of $\mathcal{T}$. The fact that such pairs of leaves of $\mathcal{T}$ have to be "logically adjacent" creates a "traffic pattern" that congests the edges of $\mathcal{T}$ very unevenly. For illustration, say that we use the clique $\mathcal{H} = \mathcal{K}_{N(\mathcal{T})}$ as our auxiliary graph. When we embed $\mathcal{H}$ into $\mathcal{T}$, the congestion roughly doubles with each successive level closer to the root of $\mathcal{T}$. Because the bounds on separation-width that are derived from congestion arguments are based on the *maximum* congestion when one embeds the auxiliary graph $\mathcal{H}$ into $\mathcal{T}$, the imbalance in the congestion among the various edges of $\mathcal{T}$ guarantees excessively small lower bounds from

congestion arguments. The challenge we address in this section is to develop bounding techniques that yield good lower bounds for complete trees.[13]

The technique we describe here is based on an interesting formal analogy between the complexity of bipartitioning a complete tree $\mathcal{T}(b)$ of arity b, as measured by the number of edges of $\mathcal{T}(b)$ that one must cut to effect the bipartition, and the complexity of representing fractions in base b, as measured by the lengths of the representations. Informally and intuitively, if the fraction $1/k$ admits a short representation in base b, then one can bipartition $\mathcal{T}(b)$ into pieces whose sizes are in the ratio $1:k-1$ by cutting only a few of $\mathcal{T}(b)$'s edges; if the fraction $1/k$ has a long representation in base b, then any bipartition of $\mathcal{T}(b)$ into pieces whose sizes are in the ratio $1:k-1$ requires cutting many of $\mathcal{T}(b)$'s edges. To give one concrete example: the fact that fractions of the form $1/2^k$ admit binary representations comprising a length-$(k-1)$ string of 0s followed by a single 1 parallels the fact that partitioning a complete binary tree into two subgraphs whose sizes are in the ratio $1:2^k-1$ (to within rounding) is easy, requiring cutting just one edge. In contrast, the fact that fractions of the form $1/(2^k-1)$ have infinitely many alternations in their binary representations parallels the fact that partitioning a complete binary tree into two subgraphs whose sizes are in the ratio $1:2^k-2$ (to within rounding) is difficult, requiring cutting logarithmically many edges. One last remark will round out the intuition in the preceding discussion, by explaining why "infinitely many alternations" in number representations translates to "logarithmically many cuts" in tree bipartitions. In short, the explanation is that the study of graph-bipartitioning takes place in a world dominated by integers, hence of bounded resolution. What we are really doing is bipartitioning an N-node tree into a desired ratio, *subject to the restriction that the smaller side of the bipartition contains at least one node.* This restriction limits our interest in alternations to those that appear in the first $\log_b N$ places in the representation of the fraction. In homely terms, logarithmic in N is "infinite" to us within this setting.

Because the analogy we have just described really does have a substantive formalization, the technique we develop in this section yields nontrivial lower bounds only for certain values of M, which depend on the arity of the target tree. Our technique produces good (i.e., large) bounds on the M-separation-widths of N-node complete b-ary trees when $M/N \approx 1/(b^k - 1)$. The reader will recall that the lower bound that we derive for complete ternary trees ($b = 3$) in this section is used in Section 4.3.7 to derive a good lower bound on the bisection-width of *arbitrary* binary trees.

Because of the formal analogy between cutting complete b-ary trees into the proportions $1:m-1$ and representing the fraction $1/m$ in base b, we find it convenient to phrase our partitioning task here in terms of the

proportion $M{:}N - M$ rather than in terms of the number M. Accordingly, for this section only, we modify the separation-width-function $W_{\mathcal{T}_h(b)}$ so that it takes the fraction $1/m$, rather than some rounding of N/m, as an argument. To the end of being precise in our arguments, we now translate the somewhat informal phrase, "partitioning the N-node tree $\mathcal{T}$ into the proportions $1{:}\, m - 1$" into the formal assertion, "partitioning the N-node tree $\mathcal{T}$ into two subforests, the smaller having size M, where

$$\left\lfloor \frac{N}{m} \right\rfloor \leqslant M \leqslant \left\lceil \frac{N}{m} \right\rceil"$$

We concentrate here on exposing "difficult" partitions of a complete tree, i.e., those that require cutting many edges of the tree. We show specifically that cutting the height-h complete b-ary tree $\mathcal{T}_h(b)$ into the proportions $1 : b^k - 2$, where $2 \leqslant k \leqslant h$, is always "difficult." When $b > 2$, we can add the case $k = 1$, i.e., the proportion $1{:}\, b - 2$, to this claim. (This proportion makes no sense, of course, in the case of binary trees, where $b = 2$.)

> Whereas Sections 4.2 and 4.3 produced explicit "applications" of underlying packing lemmas and congestion theorems, the upcoming theorem produces the desired bound on separation-width directly; it is, thus, simultaneously the underlying theorem and the "application."

THEOREM 4.4.1 (The Cutting Theorem for Complete Trees). (a) *For all integers b and h and all $k \in \{2, 3, \ldots, h\}$, the $(1/(b^k - 1))$-separation-width $W_{\mathcal{T}_h(b)}(1/(b^k - 1))$ of the height-h complete b-ary tree $\mathcal{T}_h(b)$ is no smaller than*

$$\frac{b-1}{kb^2} h - 2$$

(b) *For all integers $b > 2$ and $h \geqslant 2$, the $(1/(b - 1))$-separation-width $W_{\mathcal{T}_h(b)}(1/(b - 1))$ of the height-h complete b-ary tree $\mathcal{T}_h(b)$ is no smaller than*

$$\frac{1}{2} h - \frac{3}{8}$$

PROOF. The argument that establishes the claimed bounds distinguishes between parts (a) and (b) only toward the end, so we begin discussing both parts together. The flow of our proof assumes that we have achieved the desired bipartition of $\mathcal{T}_h(b)$ and asks how we got to that goal.

Focus on an arbitrary $k \in \{1 + \lfloor 2/b \rfloor, \ldots, h\}$. Say that we have partitioned $\mathcal{T}_h(b)$ into two subforests whose sizes are in the proportion $1 : b^k - 2$. Let us ambiguously denote by *SMALL* (resp., by *BIG*) both the former, smaller (resp., the latter, larger) subforest and its set of nodes. In order to determine how many edges we must have cut in order to achieve the indicated partition, we investigate the effect of our partition level by level in $\mathcal{T}_h(b)$. Assigning the *nodes* of $\mathcal{T}_h(b)$ to levels in the usual way (cf. Section 1.3), we now also assign the tree's *edges* to levels, by assigning each edge of $\mathcal{T}_h(b)$ that connects a level-$(\ell - 1)$ node to a level-ℓ node to edge-level $\ell \in \{1, 2, \ldots, h\}$. We denote by e_ℓ the number of level-ℓ edges of $\mathcal{T}_h(b)$ that we have cut while effecting the desired bipartition.

The following obvious fact, whose proof is left as an exercise, is used repeatedly in our argument.

FACT 4.4.2. *Each cut of a level-ℓ edge of $\mathcal{T}_h(b)$ produces a subtree of size (measured in nodes) $(b^{h-\ell+1} - 1/(b-1)$ rooted at the level-ℓ node incident to the cut edge.*

Finally, we define the *base-b correction function* κ_b, which will be a useful tool in our analysis:

$$\kappa_b(\ell) = \sum_{i=1}^{h-\ell} e_{\ell+i} b^{-i} = \sum_{\ell'=\ell+1}^{h} e_{\ell'} b^{\ell'-\ell'}$$

We use $\kappa_b(\ell)$ in our argument to bound from above the extent to which edge-cuts *below* edge-level ℓ (i.e., at higher-numbered edge-levels) can "correct" errors in rounding committed at edge-level ℓ. (This role will become clearer as we use κ_b in our argument.)

Now we begin our argument, which the reader will notice has the flavor of an analysis of an iterative-correction algorithm, i.e., an algorithm that spends each iteration i diminishing the impact of sins committed at level $i-1$.

Look at any node-level $\ell \in Z_{h+1}$ of $\mathcal{T}_h(b)$. Let $\boldsymbol{BIG}_\ell$ denote the set of nodes from level ℓ that are assigned by our bipartition to subforest *BIG*, and let $\boldsymbol{SMALL}_\ell$ denote the corresponding set for subforest *SMALL*. Since $b^k - 1$ does not divide b^ℓ (which is the number of nodes at level ℓ of $\mathcal{T}_h(b)$), the sizes of the sets $\boldsymbol{SMALL}_\ell$ and $\boldsymbol{BIG}_\ell$ cannot be quite in the "ideal" ratio[14] $1 : b^k - 2$; one set must be at least a "trifle" too big. To the end of bounding the size of this "trifle," define

$$p_{k,\ell} \stackrel{\text{def}}{=} \frac{(b^k - b^\ell - 1) \bmod (b^k - 1)}{b^k - 1} \quad \text{and} \quad P_{k,\ell} \stackrel{\text{def}}{=} \frac{b^\ell \bmod (b^k - 1)}{b^k - 1}$$

and note that

- $p_{k,\ell}$ (resp., $P_{k,\ell}$) is the smallest positive rational ε such that

$$\frac{1}{b^k - 1} b^\ell + \varepsilon \qquad \left(\text{resp.}, \frac{b^k - 2}{b^k - 1} b^\ell + \varepsilon\right)$$

  is an integer.
- $p_{k,\ell} + P_{k,\ell} = 1$.

In terms of these "trifles," we must have either

$$|SMALL_\ell| \geq \frac{1}{b^k - 1} b^\ell + p_{k,\ell} \qquad \text{or} \qquad |BIG_\ell| \geq \frac{b^k - 2}{b^k - 1} b^\ell + P_{k,\ell}$$

In the former case, say that $SMALL_\ell$ is "too big," and in the latter case, say that BIG_ℓ is "too big." Now, if $SMALL_\ell$ is "too big," then let X denote the set *SMALL*, and define

$$\sigma_\ell \stackrel{\text{def}}{=} \frac{1}{b^k - 1} \qquad \text{and} \qquad \pi_{k,\ell} \stackrel{\text{def}}{=} p_{k,\ell}$$

Alternatively, if BIG_ℓ is "too big," then let X denote the set *BIG*, and define

$$\sigma_\ell \stackrel{\text{def}}{=} 1 - \frac{1}{b^k - 1} \qquad \text{and} \qquad \pi_{k,\ell} \stackrel{\text{def}}{=} P_{k,\ell}$$

The inequalities on $|SMALL_\ell|$ and $|BIG_\ell|$ imply that

$$|X| \geq (\sigma_\ell b^\ell + \pi_{k,\ell}) \frac{b^{h-\ell+1} - 1}{b - 1} - \sum_{i=1}^{h-\ell} \frac{b^{h-\ell-i+1} - 1}{b - 1} e_{\ell+i}$$

The first term in this inequality reflects the fact that, *barring cuts below a level-ℓ node v*, placing v into X automatically places the $(b^{h-\ell+1} - 1)/(b - 1)$ nodes in the subtree of $\mathcal{T}_h(b)$ rooted at v into X; cf. Fact 4.4.2. The second term reflects the extent to which this contribution can be diminished (or, *corrected*) by edge-cuts below edge-level ℓ. By definition of κ_b and by elementary manipulation, then

$$|X| \geq \sigma_\ell \left(\frac{b^{h+1} - 1}{b - 1} - \frac{b^\ell - 1}{b - 1}\right) + \frac{b^{h-\ell+1} - 1}{b - 1} \pi_{k,\ell} - \frac{b^{h-\ell+1} - 1}{b - 1} \kappa_b(\ell) \quad (4.4.1)$$

We now present and verify two facts that build on inequality (4.4.1). These facts will establish the main claim of the theorem, namely, that the sizes of the subforests of our bipartition stand in the desired proportions only if we have cut logarithmically many edges of $\mathcal{T}_h(b)$ while effecting the bipartition.

FACT 4.4.3. *When ℓ has the form $\ell = h/2 - j$ for some $1 \leqslant j \leqslant h/2$,*

$$\kappa_b(\ell) > \pi_{k,\ell} - \sigma_\ell b^{-2j-1} - b^{-j-h/2} = \pi_{k,\ell} - \sigma_\ell b^{2\ell-h-1} - b^{\ell-h}$$

NOTE. The reader can verify that the quantity bounding $\kappa_b(\ell)$ from below in Fact 4.4.3 is positive when j is in the indicated range.

PROOF OF FACT 4.4.3. Assume, for contradiction, that for some $\ell_0 = h/2 - j$, where $1 \leqslant j \leqslant h/2$,

$$\kappa_b(\ell_0) \leqslant \pi_{k,\ell_0} - b^{2\ell_0-h-1}\sigma_{\ell_0} - b^{\ell_0-h}$$

Substituting this upper bound on κ_b in our lower bound (4.4.1) for $|X|$, and making all possible simplifications, we find that the quantity $(b-1)|X|$ is no smaller than

$$\begin{aligned}(b^{h+1}-1)\sigma_{\ell_0} - (b^{\ell_0}-1)\sigma_{\ell_0} + (b^{h-\ell_0+1}-1)\pi_{k,\ell_0} - (b^{h-\ell_0+1}-1)\kappa_b(\ell_0)\\ \geqslant (b^{h+1}-1)\sigma_{\ell_0} - (b^{2\ell_0-h-1}-1)\sigma_{\ell_0} + b - b^{\ell_0-h}\end{aligned}$$

By substituting $h/2 - j$ for ℓ_0 in the last expression, we obtain the following bound on $|X|$:

$$|X| \geqslant \frac{1}{b-1}((b^{h+1}-1)\sigma_{\ell_0} + (1-b^{-2j-1})\sigma_{\ell_0} + (b - b^{-j-h/2})) \quad (4.4.2)$$

Noting that the second term of inequality (4.4.2) is positive, while the third term is no smaller than $b - 1$, we infer immediately the inequality that establishes Fact 4.4.3, namely,

$$|X| \geqslant \sigma_{\ell_0}\frac{b^{h+1}-1}{b-1} + 1 = \sigma_{\ell_0}N(\mathcal{T}_h(b)) + 1$$

This inequality means, however, that

$$|BIG| > \frac{b^k - 2}{b^k - 1}\left(\frac{b^{h+1} - 1}{b - 1}\right) + 1 = \frac{b^k - 2}{b^k - 1} N(\mathcal{T}_h(b)) + 1$$

if $X = BIG$, and that

$$|SMALL| > \frac{1}{b^k - 1}\left(\frac{b^{h+1} - 1}{b - 1}\right) + 1 = \frac{1}{b^k - 1} N(\mathcal{T}_h(b)) + 1$$

if $X = SMALL$. Either of these contingencies contradicts the alleged sizes of *BIG* and *SMALL*. □

The upshot of Fact 4.4.3 is that for each edge-level $\ell = h/2 - j$ of $\mathcal{T}_h(b)$, where $1 \leqslant j \leqslant h/2$, we must have

$$\kappa_b(\ell) > \pi_{k,\ell} - \frac{b^k - 2}{b^k - 1} b^{2\ell - h - 1} - b^{\ell - h} \tag{4.4.3}$$

Inequality (4.4.3) allows us to bound from below the number of edges C that we must have cut in order to effect the bipartition: the number C is the sum of all of the edge-cut quantities e_ℓ. We proceed via a chain of inequalities. We note first that

$$\sum_{\ell=1}^{h} e_\ell \geqslant (b - 1) \sum_{\ell=0}^{h} \kappa_b(\ell) \tag{4.4.4}$$

which follows from regrouping the terms in the right-hand sum so as to achieve the form

$$\sum_{\ell=0}^{h} \kappa_b(\ell) = \sum_{\ell=0}^{h} c_\ell e_\ell$$

and noting that each of the resulting coefficients c_ℓ is a geometric sum of distinct inverse powers of b, hence is always less than $1/(b-1)$. Next, we lower the upper limit of the summation in inequality (4.4.4) to $h/2 - 1$, which can only decrease the sum, and we incorporate inequality (4.4.3), to transform inequality (4.4.4) to

$$\sum_{\ell=1}^{h} e_\ell \geqslant (b - 1)\left(\sum_{\ell=0}^{h/2-1} \pi_{k,\ell} - \frac{b^k - 2}{b^{h+1}(b^k - 1)} \sum_{\ell=0}^{h/2-1} b^{2\ell} - \frac{1}{b^h} \sum_{\ell=0}^{h/2-1} b^\ell\right) \tag{4.4.5}$$

Finally, by explicitly evaluating the second and third summations in inequality (4.4.5), we obtain the bound

$$\sum_{\ell=1}^{h} e_\ell \geq (b-1) \sum_{\ell=0}^{h/2-1} \pi_{k,\ell} - \left(1 - \frac{1}{b^k - 1}\right) \frac{1}{b(b+1)} - \frac{1}{b^{h/2}} \qquad (4.4.6)$$

We are left now with the task of estimating the one remaining summation in inequality (4.4.6). It is at this point that we must branch depending on whether or not $b = 2$ and $k = 1$.

Case (a). *Arbitrary* $b \geq 2$, *with* $k > 1$. In this case, we build on the fact that the base-b representation of $1/(b^k - 1)$ has many alternations of digits.

FACT 4.4.4. *For all* $k \in \{2, 3, \ldots, h\}$,

$$\sum_{\ell=0}^{h/2-1} \pi_{k,\ell} \geq \frac{1}{b^2}\left(\frac{h-2}{k} - 2\right)$$

PROOF OF FACT 4.4.4. We begin by noting that the base-b expansion of $1/(b^k - 1)$ consists of repeating length-$(k-1)$ blocks of 0s, separated by single 1s. (We insert commas in the next two setoffs to emphasize the block structure.) That is, in base b,

$$\frac{1}{b^k - 1} = 0.00\cdots01, 00\cdots01, 00\cdots01, \cdots$$

Similarly, the base-b expansion of $1 - 1/(b^k - 1)$ consists of repeating length-$(k-1)$ blocks of the digit $\bar{b} \stackrel{\text{def}}{=} b - 1$, separated by single occurrences of the digit $\bar{\bar{b}} \stackrel{\text{def}}{=} b - 2$:

$$1 - \frac{1}{b^k - 1} = 0.\bar{b}\bar{b}\cdots\bar{b}\bar{\bar{b}}, \bar{b}\bar{b}\cdots\bar{b}\bar{\bar{b}}, \bar{b}\bar{b}\cdots\bar{b}\bar{\bar{b}}, \cdots$$

One consequence of the form of these expansions is that, in any consecutive sequence of m bits, each of these fractions has at least

$$2\left(\left\lceil\frac{m}{k}\right\rceil - 2\right) + 2 = 2\left\lceil\frac{m}{k}\right\rceil - 2$$

alternations of digits, i.e., two-digit substrings of the form $\beta\gamma$ where $\beta \neq \gamma$. To wit, each subblock of k digits, except possibly the last, contains one alternation, while each subblock except for the first starts off with a different digit than its predecessor ends with.

A second consequence of the form of these expansions is that we can systematically bound the fractional parts of the relevant multiples of these

fractions. Toward this end, let n be any integer, and let the reciprocal of n have the base-b expansion

$$1/n = 0.\beta_0\beta_1 \cdots \beta_\ell\beta_{\ell+1}\cdots$$

Then, for all $\ell \geqslant 0$, the fractional part of b^ℓ/n is given by

$$\left\{\frac{b^\ell}{n}\right\} \stackrel{\text{def}}{=} \frac{b^\ell}{n} - \left\lfloor\frac{b^\ell}{n}\right\rfloor = 0.\beta_\ell\beta_{\ell+1}\cdots$$

Now, if $\beta_\ell \neq \beta_{\ell+1}$ (so there is an alternation at position ℓ), then both $\{b^\ell/n\}$ and $1 - \{b^\ell/n\}$ are no smaller than $1/b^2$; i.e.,

$$\min\left(\left\{\frac{b^\ell}{n}\right\}, 1 - \left\{\frac{b^\ell}{n}\right\}\right) \geqslant \frac{1}{b^2}$$

The importance of this bound is that when n is of the form $n = b^k - 1$, then $\pi_{k,\ell}$ equals either $\{b^\ell/n\}$ or $1 - \{b^\ell/n\}$; hence, at least the minimum of these quantities is contributed to the sum by a digit alternation at position ℓ.

Combining the estimates and bounds that follow from the form of the base-b expansion of $1/(b^k - 1)$, we deduce the inequality

$$\sum_{\ell=0}^{m} \pi_{k,\ell} \geqslant \frac{2}{b^2}\left(\left\lceil\frac{m}{k}\right\rceil - 1\right)$$

If we now let $m = h/2 - 1$ in this inequality, we find that

$$\sum_{\ell=0}^{h/2-1} \pi_{k,\ell} \geqslant \frac{2}{b^2}\left(\left\lceil\frac{h/2-1}{k}\right\rceil - 1\right) \geqslant \frac{1}{b^2}\left(\frac{h-2}{k} - 2\right)$$

as was claimed. □

Combining inequality (4.4.6) with Fact 4.4.4 yields the bound of part (a) of the theorem.

Case (b). $b > 2$ *and* $k = 1$. This case is even easier than Case (a), because of the simplicity of the base-b representation of $1/(b-1)$.

FACT 4.4.5. *When* $k = 1$ *and* $b > 2$,

$$\sum_{\ell=0}^{h/2-1} \pi_{k,\ell} \geqslant \frac{1}{2b-2}h$$

PROOF OF FACT 4.4.5. We begin by noting that the base-b expansion of $1/(b-1)$ consists of an infinite string of 1s:

$$\frac{1}{b-1} = 0.111\cdots$$

while the base-b expansion of $1 - 1/(b-1)$ consists of an infinite string of the digit $\overline{b} \stackrel{\text{def}}{=} b-2$:

$$1 - \frac{1}{b-1} = 0.\overline{b}\,\overline{b}\,\overline{b}\cdots$$

One consequence of these representations is that, for all $\ell \geq 0$,

$$\left\{\frac{b^\ell}{b-1}\right\} = \frac{1}{b-1} \qquad \text{and} \qquad 1 - \left\{\frac{b^\ell}{b-1}\right\} = 1 - \frac{1}{b-1}$$

so that

$$\min\left(\left\{\frac{b^\ell}{b-1}\right\}, 1 - \left\{\frac{b^\ell}{b-1}\right\}\right) = \frac{1}{b-1}$$

Since $\pi_{1,\ell}$ equals either $\{b^\ell/(b-1)\}$ or $1 - \{b^\ell/(b-1)\}$, it is immediate that

$$\sum_{\ell=0}^{h/2-1} \pi_{1,\ell} \geq \frac{1}{b-1}\left(\left\lceil \frac{h}{2} \right\rceil\right)$$

which establishes the claim. □

Combining inequality (4.4.6) with Fact 4.4.5 and noting that $b \geq 3$ yields the bound of part (b) of the theorem. □

4.5. Information-Transfer Arguments

We close this chapter with a bounding technique that is dramatically different from any of our other techniques, in that it employs "semantic" information about the graph $\mathcal{G}$ in order to bound $\mathcal{G}$'s separation-width. More explicitly, whereas our other bounding techniques have all derived their bounds by exploiting purely structural properties of $\mathcal{G}$, the current technique builds on the question: What functions does $\mathcal{G}$ "support;" i.e.,

what functions can be computed by circuits that have the structure of $\mathcal{G}$? The technique produces a lower bound on $\mathcal{G}$'s separation-width via an argument that bounds from below the amount of information transfer that is necessary to compute the supported functions. (One can view the proof of Proposition 4.1.1 as using a rather degenerate form of such an information-transfer argument.)

The technique we develop here has its origins in two genres of study that seem on the surface to have little to do with the enterprise of this section. The first group of studies concentrate, within the framework of VLSI theory, on the amount of information transfer that is necessary to compute various functions; see, e.g., Abelson and Andreae [1980], Bilardi [1985], Siegel [1986], Thompson [1980], and Vuillemin [1983]. In Section 2.4.3 we reviewed the use of such arguments within VLSI theory to bound from below the minimum area of a VLSI layout of circuits that compute various functions. More sophisticated use of the arguments establishes bounds on the AREA-TIME^2 product of combinational (i.e., memoryless) circuits that compute the functions. The second group of studies create the syntactic objects that enable our development in this section. These studies "unfold" loops in program schemes along the time axis, to create "acyclic" schemes that represent, within the space-time domain, the entire computation at once (rather than over time); see, e.g., Miranker and Winkler [1984], Quinton [1984], and Quinton and VanDongen [1989]. The way these two genres of study meld to accomplish our goal should become clear imminently, as we turn now to the technical development.

4.5.1. Computation Digraphs

For any graph $\mathcal{G}$ and positive integer k, the *k-step computation digraph* of $\mathcal{G}$, denoted $\mathcal{G}^{[k]}$, is the *directed* graph whose node-set is

$$\mathbf{N}(\mathcal{G}^{[k]}) \stackrel{\text{def}}{=} \mathbf{N}(\mathcal{G}) \times \mathbb{Z}_{k+1}$$

and whose arcs are given as follows. For all $u, v \in \mathbf{N}(\mathcal{G})$, if (u, v) is an *edge* of $\mathcal{G}$, then, for all $i \in \mathbb{Z}_k$, $(\langle u, i\rangle, \langle v, i+1\rangle)$ is an *arc* of $\mathcal{G}^{[k]}$. In words: the arcs of $\mathcal{G}^{[k]}$ lead from the "time-i copy of $\mathcal{G}$" to the "time-$(i+1)$ copy."

The importance of computation digraphs is that they (sometimes) afford one a static, syntactic view of the dynamic, semantic behavior of computations on parallel computers whose communication structure corresponds to the structure of graph $\mathcal{G}$. We illustrate this use of computation digraphs in a simple scenario that allows us to bound from below the separation-widths of de Bruijn graphs. The sources we cited earlier will suggest many other applications of this bounding strategy.

4.5.2. de Bruijn Graphs as Permuters

Recall from Section 2.4.3 that an $(M \geq 2N)$-node graph $\mathcal{H}$ is an *N-input permutation network* if it has N input nodes, N output nodes, and enjoys the following property: given any permutation π of Z_N, viewed as a permutation of $\mathcal{H}$'s input nodes, there are N *edge-disjoint* paths in $\mathcal{H}$ that connect each input node i to output node $\pi(i)$.

We claim that the $(N = 2^n)$-node de Bruijn network $\mathcal{D}_n$ is able to compute all permutations of Z_N, in the sense of the following lemma.

LEMMA 4.5.1. *For all n, the 3n-step computation digraph $\mathcal{D}_n^{[3n]}$ of the order-n de Bruijn graph $\mathcal{D}_n$ is a 2^n-input permutation network with input nodes $\mathbf{N}(\mathcal{D}_n) \times \{0\}$ and output nodes $\mathbf{N}(\mathcal{D}_n) \times \{3n\}$.*

PROOF. We present only the "front end" of the proof. Specifically, we show that the computation digraph $\mathcal{D}_n^{[3n]}$ is "computationally equivalent" to the order-*n triple-FFT network*, which can be shown to be a permutation network (see, e.g., Problem 3.104 in Leighton [1992]). We employ a technique that derives from Annexstein *et al.* [1990].

The *order-n triple*-FFT *network* $\mathcal{F}_n'''$ is obtained by taking three copies of the order-n FFT network $\mathcal{F}_n$—call them $\mathcal{F}_n^{[0]}$, $\mathcal{F}_n^{[1]}$, and $\mathcal{F}_n^{[2]}$—and "splicing" them together by *identifying* each output node $\langle n, x\rangle$ of copy $\mathcal{F}_n^{[0]}$ with input node $\langle 0, x\rangle$ of copy $\mathcal{F}_n^{[1]}$, and identifying each output node $\langle n, x\rangle$ of copy $\mathcal{F}_n^{[1]}$ with input node $\langle 0, x\rangle$ of copy $\mathcal{F}_n^{[2]}$; see Figure 4.5-1. In a natural way this splicing gives nodes of $\mathcal{F}_n'''$ labels of the form $\langle k, y\rangle$, where $k \in Z_{3n+1}$ and $y \in Z_2^n$.

We claim that the computation digraph $\mathcal{D}_n^{[3n]}$ is "computationally equivalent" to $\mathcal{F}_n'''$. We verify this assertion with the help of the following strategy for labeling the nodes of $\mathcal{F}_n'''$ with n-bit strings $y \in Z_2^n$.

- Label each node $\langle 0, x\rangle$ of $\mathcal{F}_n'''$, where $x \in Z_2^n$, with string x; this has been done in Figure 4.5-1.
- Inductively, consider node $v = \langle \ell, x\rangle$ of $\mathcal{F}_n'''$, where $\ell \in Z_{3n}$ and $x \in Z_2^n$. If node v has been assigned string-label z, then[15]
  - label node $\langle \ell + 1, x\rangle$ of $\mathcal{F}_n'''$ with the *shuffle* of string z
  - label node $\langle \ell + 1, x'\rangle$ of $\mathcal{F}_n'''$, where x' is the srting obtained from x by complementing bit-position ℓ mod n, with the *shuffle-exchange* of string z.

Figure 4.5-2(a) illustrates this labeling on two consecutive levels of $\mathcal{F}_n'''$. We leave to the reader the exercise of verifying that this labeling procedure is *well defined* (i.e., each node of the network receives precisely one label) and

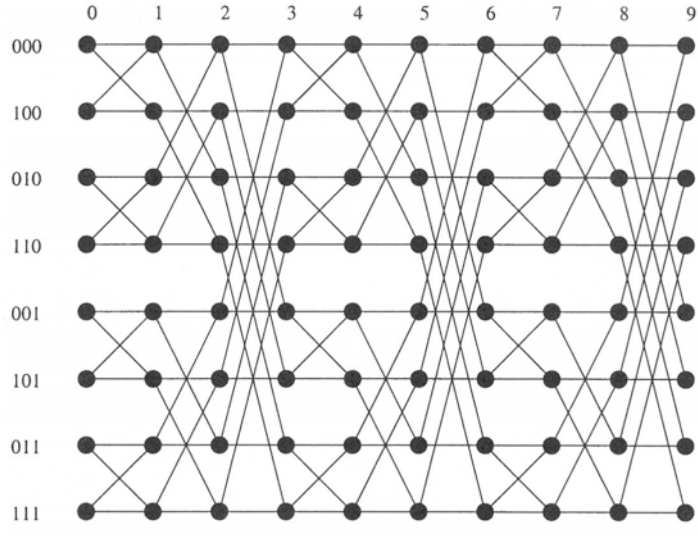


Figure 4.5-1. The triple-FFT network.

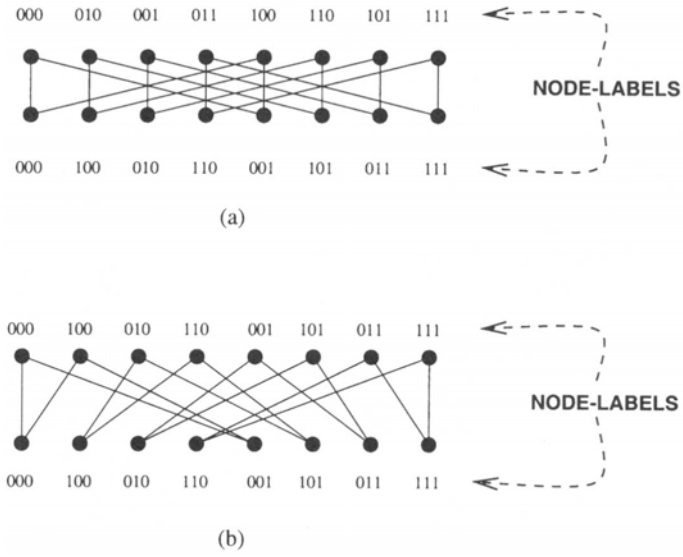


Figure 4.5-2. Two "typical" adjacent labeled levels of the triple-FFT network $\mathcal{F}_n'''$: (a) in their "natural" order; (b) permuted so that like-labeled nodes line up.

uniform (i.e., each level of $\mathcal{F}'''_n$ gets labeled with the entire set of strings Z_2^n). As an aid in this verification, verify first that, for all $\ell \in Z_{n+1}$ and $x \in Z_2^n$, the labeling procedure assigns identical labels to nodes $\langle \ell, x \rangle$, $\langle \ell + n, x \rangle$, and $\langle \ell + 2n, x \rangle$. It will then suffice to prove the well-definition of the labeling for $\mathcal{F}_n$.

We claim that this labeling of the nodes of $\mathcal{F}'''_n$ reveals that traversing one level of the network is equivalent to a single computation step of $\mathcal{D}_n$. To see this, permute the levels of the labeled version of $\mathcal{F}'''_n$ so as to "line up" like-labeled nodes; see Figure 4.5-2(b). Now identify like-labeled nodes on each of the two levels, i.e., literally paste like-labeled nodes together. The resulting network is easily seen to be $\mathcal{D}_n$: that was the aim of the labeling! We can now easily extend this equivalent between each successive pair of levels of the network $\mathcal{F}'''_n$, on the one hand, and the one-step computation digraph $\mathcal{D}_n^{[1]}$, on the other, to an equivalence between the $3n$-step computation digraph $\mathcal{D}_n^{[3n]}$ and the order-n triple-FFT network $\mathcal{F}'''_n$.

AN AMUSING ASIDE. If one takes the FFT network $\mathcal{F}_n$ (the butterfly network $\mathcal{B}_n$ will work also) and "collapses levels" by identifying all nodes that share the same PWL string, then one obtains a copy of the n-dimensional boolean hypercube $\mathcal{Q}_n$. On the other hand, if one labels the nodes of $\mathcal{F}_n$ (or of $\mathcal{B}_n$) in the manner indicated in the section *and then* collapses levels, one obtains a copy of the order-n de Bruijn network $\mathcal{D}_n$. At first blush this appears to be a bit of sleight of hand: after all, the node-labeling has not changed the graph at all. It is worth contemplating why it works.

4.5.3. Exploiting the Equivalence

Now that we know that $\mathcal{D}_n^{[3n]}$ is a permutation network, we can use an argument related to the analysis in Section 2.4.3 to show that $\mathcal{D}_n$ must have large separation-widths.

THEOREM 4.5.2. *The M-separation-width $W_{\mathcal{D}_n}(M)$ of the order-n de Bruijn network $\mathcal{D}_n$ is no smaller than*

$$\min(M, N - M)/(3 \log N)$$

where $N = N(\mathcal{D}_n) = 2^n$. In particular, the bisection-width of $\mathcal{D}_n$ (for which $M = \frac{1}{2}N = 2^{n-1}$) is no smaller than $2^n/6n$.

PROOF. We consider bipartitions of the digraph $\mathcal{D}_n^{[3n]}$ by removing edges. Call such a bipartition into subdigraphs $\mathcal{D}_0$ and $\mathcal{D}_1$ *interesting* if it is induced by a bipartition of $\mathcal{D}_n$. In other words, in an interesting bipartition,

there exist nonempty subsets S_0 and S_1 of $\mathbf{N}(\mathscr{D}_n)$ such that $\mathbf{N}(\mathscr{D}_0) = S_0 \times Z_{3n+1}$ and $\mathbf{N}(\mathscr{D}_1) = S_1 \times Z_{3n+1}$.

Focus on an arbitrary interesting bipartition of $\mathscr{D}_n^{[3n]}$ for which the set S_0 contains M nodes and the set S_1 contains $N - M$ nodes; with no loss of generality, say that $M \leqslant N - M$. As we noted in Lemma 4.5.1, the graph $\mathscr{D}_n^{[3n]}$ is an N-input permutation network with input nodes $\mathbf{N}(\mathscr{D}_n) \times \{0\}$ and output nodes $\mathbf{N}(\mathscr{D}_n) \times \{3n\}$. We see that subgraph $\mathscr{D}_0$ of $\mathscr{D}_n^{[3n]}$ contains M of the input nodes and M of the output nodes, while subgraph $\mathscr{D}_1$ contains $N - M$ of the input nodes and $N - M$ of the output nodes.

Since $M \leqslant N - M$, there is a permutation of Z_N that maps the M input nodes that reside in $\mathscr{D}_0$ into the set of $N - M$ output nodes that reside in $\mathscr{D}_1$. (Note that if $M \geqslant N - M$, then we would just interchange the roles of $\mathscr{D}_0$ and $\mathscr{D}_1$.) Because $\mathscr{D}_n^{[3n]}$ is an N-input permuter, there must be M edge-disjoint paths that connect the M input nodes of $\mathscr{D}_0$ to the appropriate set of M output nodes of $\mathscr{D}_1$. Now, in order to effect the desired edge-bipartition of $\mathscr{D}_n^{[3n]}$ into $\mathscr{D}_0$ and $\mathscr{D}_1$ we must have had to cut all of these paths! In other words,

FACT 4.5.3. *At least M edges of $\mathscr{D}_n^{[3n]}$ were cut in the process of partitioning $\mathscr{D}_n^{[3n]}$ into $\mathscr{D}_0$ and $\mathscr{D}_1$.*

We wrap up the argument by considering the impact of this bipartition of $\mathscr{D}_n^{[3n]}$ on the bipartition of $\mathscr{D}_n$ that induced it. We claim that cutting all of the edges of $\mathscr{D}_n$ that are images of the cut edges of $\mathscr{D}_n^{[3n]}$ effects this bipartition of $\mathscr{D}_n$. This follows from the mutual disjointness of subdigraphs $\mathscr{D}_0$ and $\mathscr{D}_1$ of $\mathscr{D}_n^{[3n]}$. Now, cutting these edges of $\mathscr{D}_n$ will likely cut certain edges more than once, for each edge of $\mathscr{D}_n$ spawns $3n$ edges of $\mathscr{D}_n^{[3n]}$. But, clearly, no edge of $\mathscr{D}_n$ will get cut more than $3n$ times. It follows that effecting this bipartition of $\mathscr{D}_n$ requires cutting at least $M/3n$ edges. The theorem follows. □

4.6. Sources

The strategy of bounding the M-separation-widths of graphs via packing function seems to have originated in Rosenberg [1979a], where one finds (analogues of) Lemma 4.2.1 and Theorems 4.2.8 and 4.2.10. Theorem 4.2.2 is original here, as far as we know. Theorem 4.2.4 derives from Chung *et al.* [1988]; its extension, Theorem 4.2.6, to ternary cubes comes from Heath *et al.* [1992]. The bounding of mincing-widths, both weighted and unweighted, via packing functions is derived from Rosenberg *et al.* [1979].

The inspiration for the congestion argument was Leighton's use (in Leighton [1983]) of the congestion of embeddings of $\mathcal{K}_n$ into n-node graphs $\mathcal{G}$ to obtain lower bounds on the number of crossings in any drawing of $\mathcal{G}$ in the plane. Our bounds on the separation-widths of arbitrary binary trees derive in spirit from Hong *et al.* [1983], where an *ad hoc* argument is used to obtain a less robust bound.

Theorem 4.4.1 generalizes to arbitrary arities a version of the same result for binary trees in Chung and Rosenberg [1986]. This latter result, in turn, improves and sharpens an argument from Hong *et al.* [1983].

The material in Section 4.5 comes primarily from Thompson [1980] and Vuillemin [1983]. Information-transfer arguments attracted much attention throughout the 1980's. Additional work along this line appears in Abelson and Andreae [1980], Aho *et al.* [1983], Bilardi [1985], Cole and Siegel [1988], JáJá and Kumar [1984], and Siegel [1986], as well as other places.

Notes

1. We are grateful to the authors and publisher of Chung *et al.* [1988] for permission to paraphrase from that source as the starting point of this section.
2. Recall that $d_{\mathcal{G}}(v)$ denotes the degree of node v in $\mathcal{G}$.
3. We are grateful to the publisher of Heath *et al.* [1992] for permission to paraphrase from that source.
4. We are grateful to the publisher of Rosenberg [1979a] for permission to paraphrase from that source.
5. We are grateful to the publisher of Rosenberg *et al.* [1979] for permission to paraphrase from that source.
6. "Equal size" has the same meaning as in previous sections.
7. That is to say, given any two nodes (resp., any two edges) of $\mathcal{K}_n$, there is an automorphism of the graph that maps each of the nodes (resp., each of the edges) to the other.
8. The reader should note carefully that we are talking here about the mn-node clique $K_{m \cdot n}$, *not* the complete bipartite graph $K_{m,n}$.
9. Throughout, we use the notation $u \leftrightarrow v$, where u and v are nodes of the graph $\mathcal{G}$ being discussed, to denote ambiguously the fact that $u = v$ *or* the fact that u and v are adjacent in $\mathcal{G}$. This convention greatly simplifies notation and should never lead to an unresolvable ambiguity.
10. We need some such ordering convention in order to avoid double-counting routing-paths.

11. This factor accounts for our requiring that the source node of each routing-path precede the target node lexicographically.
12. Of course, the n-step rewriting phase will not allow us to connect all possible source nodes with all possible target nodes; we shall need the positioning phase to effect these connections. But, all source node-target node pairs contribute to the "rewriting congestion."
13. We are grateful to the publisher of Chung *et al.* [1988] for permission to paraphrase from that source.
14. This ratio would be "ideal" because it would guarantee *on a level-by-level basis* that our bipartition achieves the desired proportions.
15. The *shuffle-exchange* of string z is obtained from z via a shuffle (cf. Section 1.3.5) followed by a complementation of the rightmost bit-position.

This page intentionally left blank

Appendix A

Applications of Graph Separators, Revisited

A.1. Introduction

This appendix is devoted to applying the lower-bound techniques of Chapter 4 and their applications to three of the application areas of Chapter 2, namely, VLSI layout, graph embeddings, and strongly universal interval hypergraphs. In each of the areas, we use lower bounds on graph separation-width proved in Chapter 4 to establish one or more lower bounds within the application area. Throughout this appendix, the reader is referred to the relevant sections of Chapters 2 and 4 for definitions and terminology.

A.2. Graph Embeddings via Separators

In this section we revisit the subject of lower bounds on the costs of graph embeddings, as studied in Section 2.3.2, in the light of Chapter 4's bounds on separation-width. In Section A.2.1 we revisit the development of Section 2.3.1.1, deriving bounds on the congestions of embeddings involving certain guest-host pairings. Section A.2.2 revisits the development of Section 2.3.1.2, presenting analogous bounds on the dilations of embeddings. Finally, Section A.2.3 revisits the development of Section 2.3.1.3, studying bounds on the cumulative costs of embeddings. The reader should note that, whereas good lower bounds on the dilations and congestions of embeddings can be derived using good lower bounds on just the bisection-widths of the guest graphs—hence can make full use of the techniques of Chapter 4—good lower bounds on the cumulative costs of embeddings require good lower bounds on a variety of separation-widths of the guest graphs. As we

have noted, such bounds are generally attainable only via packing arguments such as those in Section 4.2.

Since most of the bounds in the section are obtained merely by instantiating and manipulating expressions derived in Section 2.3 and Chapter 4, we justify the bounds here only via sketches. The interested reader can, therefore, view this section as a set of exercises with hints.

A.2.1. Bounds on Congestion

Since both paths and complete binary trees have recursive node-bisectors of size $R(n) \equiv 1$, the congestion of any embedding of a graph $\mathcal{G}$ into a like-sized path or tree is bounded below by $W_{\mathcal{G}}$, the bisection-width of $\mathcal{G}$.

APPLICATION A.2.1. *The edge-congestion of any embedding into the N-node path or the N-node complete binary tree of the N-node*[1]

boolean hypercube $\mathcal{Q}_n$	*can be no smaller than* $\frac{1}{2}N$
butterfly graph $\mathcal{B}_n$	*can be no smaller than* $N/\log N$
complete ternary tree $\mathcal{T}_h(3)$	*can be no smaller than* $\frac{1}{2}\log N - l.o.t.$
de Bruijn graph $\mathcal{D}_n$	*can be no smaller than* $\frac{1}{2}N/\log N$
d-dimensional mesh $\mathcal{M}_n(d)$	*can be no smaller than* $2^{1/d}dN^{1-1/d}$
X-tree $\mathcal{X}_h$	*can be no smaller than* $\log(N-1+1/(N+1)) - 1$

Since all of the listed guest graphs, except the de Bruijn graph, have "honest" recursive node-bisectors, i.e., ones whose sizes satisfy $R_{\mathcal{G}}(N) = W_{\mathcal{G}}(N)$, it is easy to devise embeddings of each of these graphs into a like-sized path or complete binary tree, whose edge-congestions match the lower bounds of Application A.2.1. Our lower bound on the cutwidth of the de Bruijn graph is within a small constant factor of the upper bound. A linearization of $\mathcal{D}_n$ that witnesses this assertion can be derived, via projection, from Leighton's optimal embedding of the shuffle-exchange graph into the mesh (Leighton [1983]), in the light of the quasi-isometry of shuffle-exchange and de Bruijn graphs (Proposition 1.6.3).

One more set of examples should suffice to illustrate the instantiation of our general bounds on edge-congestion.

APPLICATION A.2.2. *The edge-congestion of any embedding of the N-node boolean hypercube into the N-node*

butterfly graph $\mathcal{B}_n$	*can be no smaller than* $\frac{1}{2}\log N$
de Bruijn graph $\mathcal{D}_n$	*can be no smaller than* $\log N$
2-dimensional mesh $\mathcal{M}_{n,n}$	*can be no smaller than* $\frac{1}{2}\sqrt{N}$
X-tree $\mathcal{X}_h$	*can be no smaller than* $\frac{1}{2}N/\log N$.

A.2.2. Bounds on Dilation

We present a few straightforward instantiations of our general lower bound.

APPLICATION A.2.3. *Let* $\mathbf{H}$ *be a bounded-degree family having a recursive node-bisector of size* $R_{\mathbf{H}}(n) = n^{O(1)}$. *Any embedding of an N-node boolean hypercube into an N-node graph* $\mathcal{H} \in \mathbf{H}$ *must have dilation* $\Omega(\log N)$. *Such graph families* $\mathbf{H}$ *include, among others, meshes of fixed dimensionality, trees of fixed arity, and X-trees.*

APPLICATION A.2.4. *Let* $\mathbf{G}$ *be a bounded-degree family whose graphs have bisection-widths* $W_{\mathbf{G}}(n) = n^{\Omega(1)}$. *Any embedding of an N-node graph* $\mathcal{G} \in \mathbf{G}$ *into an N-node complete binary tree must have dilation* $\Omega(\log N)$. *Such graph families* $\mathbf{G}$ *include meshes of arbitrary dimensionalities, hypercubes of arbitrary dimensionalities, butterfly graphs of arbitrary bases, and de Bruijn graphs of arbitrary bases.*

APPLICATION A.2.5. *The dilations of the following embeddings can be no smaller than* $\Omega(\log \log N)$:

- *any embedding of the N-node hypercube* $\mathcal{Q}_n$ *into the N-node butterfly graph*
- *any embedding of the N-node hypercube* $\mathcal{Q}_n$ *into the N-node de Bruijn graph* $\mathcal{D}_n$
- *any embedding of the N-node X-tree* $\mathcal{X}_h$ *into the N-node complete binary tree* $\mathcal{T}_h$

Since our generic lower bound on the bandwidth of a graph is just a factor of 2 smaller than our lower bound on its cutwidth, the reader can easily rewrite Application A.2.1. to obtain a set of illustrations of the lower bounds our techniques yield on bandwidth.

The lower bounds one obtains using separation-widths are often close to tight; however, the one scenario in which these bounds are much too small occurs when the host graph has *much* larger diameter than the guest graph. As but one instance, the present technique yields a trivial (constant) lower bound on the dilations of embeddings of the N-node complete binary tree $\mathcal{T}_h$ into a like-sized mesh or path. On the other hand, a simple comparison of the diameters of the tree and the mesh yields a lower bound of $\Omega(\sqrt{N}/\log N)$ for such embeddings (which can be shown to be tight, to within constant factors (Ullman [1984])). We leave this diameter-induced lower bound as an exercise for the reader.

A.2.3. Bounds on Cumulative Cost

The final topic of this section is the cumulative-cost of graph embeddings. The bounds in the following applications derive from the logical development in Section 2.3.1.3, coupled with the lower bounds on separation-widths derived in Section 4.2. We organize our bounds around the three families of host graphs considered at the end of Section 2.3.1.3, namely, paths, trees, and meshes. For each of these hosts, we consider three guests, namely, X-trees, binary hypercubes, and two-dimensional meshes, in order to suggest that lower bounds obtained via packing arguments often yield good lower bounds on the cumulative-costs of graph embeddings. Since the bounds of this section require estimating somewhat complicated summations, we derive our results only to within undetermined constant factors.

A.2.3.1. Paths

The *average* dilations and congestions of embeddings of our three guest graphs into paths, as exposed by the cumulative-costs of the embeddings, can be at most a constant factor smaller than the smallest *worst-case* dilations and congestions for these graphs.

APPLICATION A.2.6. *The cumulative-cost of any embedding into a path of an N-node*

X-tree $\mathcal{X}_h$	*can be no smaller than* $cN \log N$
2-dimensional mesh $\mathcal{M}_{m,n}$	*can be no smaller than* $cdN^{3/2}$
boolean hypercube $\mathcal{Q}_n$	*can be no smaller than* $cN^2 \log N$

for some constant $c > 0$.

VERIFICATION. We merely suggest how one evaluates the relevant summations. For X-trees:

$$\sum_{M=1}^{N-1}\left(\log\left(\frac{M(N-M)}{N+1}\right)+1\right) > 2\sum_{M=1}^{N-1}\log M-(N-1)\log(N+1)$$
$$= 2\log(N-1)!-(N-1)\log(N+1)$$
$$= \Omega(N\log N)$$

For meshes:

$$2\sum_{M=1}^{N-1}(M^{1/2}+(N-M)^{1/2}-N^{1/2}) = 4\sum_{M=1}^{N-1}M^{1/2}-2(N-1)N^{1/2}$$
$$= \Omega(N^{3/2})$$

For hypercubes:

$$\frac{1}{2}\sum_{M=1}^{N-1}\log\left(\frac{N^N}{M^M(N-M)^N-M)}\right) = \frac{1}{2}N(N-1)\log N-\sum_{M=1}^{N-1}M\log M$$
$$= \Omega(N^2\log N)$$

The indicated summations can be adequately estimated via integration. Details are left to the reader. □

The bounds of Application A.2.6 are within constant factors of optimal. To wit:

- The embedding of the N-node X-tree into the path, which is induced by the inorder embedding of the complete binary tree, has average dilation proportional to $\log N$.
- The row-major embedding of the $\sqrt{N}\times\sqrt{N}$ mesh into the path has average dilation $\sqrt{N}$.
- The recursive, dimension-by-dimension, embedding of the N-node boolean hypercube into the path has average dilation $N/2$.

A.2.3.2. Meshes

Our final example points out that the cumulative-cost of embeddings of boolean hypercubes into two-dimensional meshes is just a factor of 2 smaller than the cumulative-cost of embeddings of hypercubes into paths.

We leave the verification of the following to the reader.

APPLICATION A.2.7. *The cumulative-cost of any embedding of the N-node boolean hypercube into the N-node two-dimensional mesh is no smaller than* $cN^2 \log N$ *for some constant* $c > 0$.

A.3. Laying Out VLSI Circuits

We remarked in Section 2.4 that the abstract VLSI layouts produced by the strategy presented there are often within a constant factor of optimal in AREA rather than just within a few logarithmic factors of optimal. In this section we exhibit three families of graphs that illustrate our point, namely, boolean hypercubes, FFT graphs, and multidimensional meshes. In all three cases we sketch how to establish the upper bounds using the layout strategy of Section 2.4.1 (but using simple recursive edge-bisectors that the families admit, rather than $\sqrt{2}$-bifurcators), and we invoke Chapter 4's bounds on bisection-width to allow us to instantiate the lower-bound technique of Section 2.4.2.

A.3.1. Boolean Hypercubes

It is a simple exercise to verify that the family of boolean hypercubes has a recursive edge-bisector of size $R(x) = x$. To wit, one bisects a given hypercube $\mathcal{Q}_n$ by removing the $N(\mathcal{Q}_n)$ edges that cross any given dimension, thereby producing two copies of $\mathcal{Q}_{n-1}$ (which allows the recursion to continue).

If one uses the indicated recursive edge-bisector in the layout algorithm of Section 2.4, then one obtains a layout of $\mathcal{Q}_n$ of dimensions $H_0 \times W_0$, which is obtained from a sequence of sublayouts, the ith of which has dimensions $H_i \times W_i$ and holds a copy of $\mathcal{Q}_{n-i}$. As in Section 2.4, we estimate the area of the layout via the following recurrences.

$$H_i = W_{i+1} + 2^{n-i+1} = 2H_{i+2} + \tfrac{5}{2}2^{n-i}$$

$$W_i = 2H_{i+1} + 2^{n-i} \quad = 2W_{i+2} + \tfrac{3}{2}2^{n-i}$$

from which we infer that

$$\text{AREA}(\mathcal{Q}_n) \leqslant \tfrac{15}{4}4^n = \tfrac{15}{4}(N(\mathcal{Q}_n))^2$$

Theorem 2.4.2 now combines with the bisection-width bounds for $\mathcal{Q}_n$ that we obtain in Sections 4.2 and 4.3 (Applications 4.2.5 and 4.3.6) to establish that this bound is within a constant factor of the true AREA of $\mathcal{Q}_n$.

To be specific, we established in the cited applications that

$$W_{\mathcal{Q}_n}(\tfrac{1}{2}N(\mathcal{Q}_n)) \geq 2^{n-1} = \tfrac{1}{2}N(\mathcal{Q}_n)$$

By Theorem 2.4.2 we therefore may infer that

$$\text{AREA}(\mathcal{Q}_n) \geq \tfrac{1}{4}(2^n - 1)^2 = \tfrac{1}{4}4^n + \text{l.o.t.} = \tfrac{1}{4}(N(\mathcal{Q}_n))^2 + \text{l.o.t.}$$

A.3.2. FFT Networks

It is a simple exercise to verify that the family of FFT networks has a recursive edge-bisector of size $R(x) = x/\log x$ + l.o.t. To wit, one can bisect $\mathcal{F}_n$ by removing the edges that go between levels $n - 1$ and n, thereby producing two copies of $\mathcal{F}_{n-1}$ (which allows the recursion to continue). The removed edges number

$$\frac{N(\mathcal{F}_n)}{\log N(\mathcal{F}_n)} + \frac{2^n \log n}{n + \log n} = 2^n$$

If one uses the indicated recursive edge-bisector in the layout algrithm of Section 2.4, then one obtains a layout of $\mathcal{F}_n$ of dimensions $H_0 \times W_0$, which is obtained from a sequence of sublayouts, the ith of which has dimensions $H_i \times W_i$ and holds a copy of $\mathcal{F}_{n-i}$. As in Section 2.4, we estimate the area of the layout via the following recurrences:

$$H_i \leq 2H_{i+2} + c_h 2^{n-i}$$
$$W_i \leq 2W_{i+2} + c_w 2^{n-i}$$

for some constants c_h and c_w; we conclude that

$$\text{AREA}(\mathcal{F}_n) \leq c_a 4^n = c_a \left(\frac{N(\mathcal{F}_n)}{\log N(\mathcal{F}_n)} \right)^2$$

for some constant c_a.

Theorem 2.4.2 now combines with the bisection-width bounds we obtain for $\mathcal{F}_n$ in Section 4.3—by combining Application 4.3.13 with the quasi-isometry of $\mathcal{F}_n$ and $\mathcal{B}_n$ (Proposition 1.6.5)—to establish that this bound is within a constant factor of the true AREA of $\mathcal{F}_n$.

To be specific, we establish in Section 4.3 that

$$W_{\mathcal{F}_n}\left(\tfrac{1}{2}N(\mathcal{F}_n)\right) \geqslant c2^n \geqslant c'\frac{N(\mathcal{F}_n)}{\log N(\mathcal{F}_n)}$$

for some constants c, c'. By Theorem 2.4.2, we now conclude the following bound.

THEOREM A.3.1. *There is a constant* $c'' > 0$ *such that*

$$\mathrm{AREA}(\mathcal{F}_n) \geqslant c''(2^n - 1)^2 = c''\left(\frac{N(\mathcal{F}_n)}{\log N(\mathcal{F}_n)}\right)^2 + l.o.t.$$

A.3.3. Multidimensional Meshes

In this section we consider VLSI layouts of the family of *equilateral d-dimensional meshes* $\{\mathcal{M}_n(d)\}$ *whose (common) side-length*[2] *n is a power of* 2, for arbitrary but fixed dimensionality d. The bound we present is correct, but trivial, when $d = 2$ (since formal VLSI layouts are embeddings into two-dimensional meshes).

It is a simple exercise to verify that one can recursively bisect any given side-n mesh $\mathcal{M}_n(d)$ in our family by cutting no more than $R(k) = 2^{-k}n^{d-1}$ edges at the kth level of the recursion. To wit, one can recursively bisect $\mathcal{M}_n(d)$ by cyclically cutting the edges midway along dimensions 1,2, ... , in that order. In three dimensions, for instance, the sequence of dimensions cut and numbers of edges cut are given in Table A.3-1.

If one uses the indicated recursive edge-bisector in the layout algorithm of Section 2.4, then one obtains a layout of $\mathcal{M}_n(d)$ of dimensions $H_0 \times W_0$, which is obtained from a sequence of sublayouts, the ith of which has dimensions $H_i \times W_i$. As in Section 2.4, we estimate the area of the layout via the following recurrences:

$$H_i \leqslant 2H_{i+2} + c_h n^{d-1}$$
$$W_i \leqslant 2W_{i+2} + c_w n^{d-1}$$

for some constants c_h and c_w; we conclude that

$$\mathrm{AREA}(\mathcal{M}_n(d)) \leqslant c_a n^{2d-2} = c_a N(\mathcal{M}_n(d))^{2-2/d}$$

for some constant c_a.

Table A.3-1. Sequence of Dimensions Cut and Numbers of Edges Cut for $\mathcal{M}_n(3)$

Dimension cut	Size of cut
1	n^2
2	$n^2/2$
3	$n^2/4$
1	$n^2/8$
2	$n^2/16$
3	$n^2/32$
⋮	⋮

Theorem 2.4.2 now combines with the bisection-width bounds we obtain for $\mathcal{M}_n(d)$ in Section 4.3 (in Application 4.2.9) to establish that this bound is within a constant factor of the true AREA of $\mathcal{M}_n(d)$.

To be specific, we established in Section 4.3 that

$$W_{\mathcal{M}_n(d)}(\tfrac{1}{2}N(\mathcal{M}_n(d))) \geq (2^{1/d} - 1)dn^{d-1} \geq cN(\mathcal{M}_n(d))^{1-1/d}$$

for some constant c (which depends on d but is constant for fixed d). By Theorem 2.4.2 we infer that

$$\text{AREA}(\mathcal{M}_n(d)) \geq c'n^{2(d-1)} = c'N(\mathcal{M}_n(d))^{2-2/d}$$

for some constant c' (which depends on d).

A.4. Strongly Universal Interval Hypergraphs

In Section 2.5 we constructed, for any finite family of graphs **G**, a strongly universal I-hypergraph $\mathcal{S}(\mathbf{G})$, based on the size of a β-separator for the family **G**. In this section we prove that the I-hypergraphs produced there are often within a small constant factor of optimal in SIZE. In Section A.4.1 we establish this optimality, via a very simple argument, for finite subfamilies of any family of graphs that is *algebraically decomposable* in the sense of having both β-separator-size (for some constant β) and bisection-width $\Theta(n^\epsilon)$. Hypercubes and rectangular meshes are two examples of algebraically decomposable graph families. In Section A.4.2 we establish this optimality, via a rather delicate argument, for finite families of complete binary trees.

A.4.1. Algebraically Decomposable Graphs

Let $\mathbf{G}$ be any (possibly infinite) family of graphs. We say that $\mathbf{G}$ is *algebraically decomposable* if there exist constants $\beta, \gamma, \delta \in (0, 1)$, $\varepsilon \in (0, 1]$, and $c > 0$ such that

1. The family $\mathbf{G}$ has a β-separator of size $S(n) = O(n^\varepsilon)$.
2. For each graph $\mathcal{G} \in \mathbf{G}$, there is a graph $\mathcal{G}' \in \mathbf{G}$ for which $\gamma N(\mathcal{G}) \leqslant N(\mathcal{G}') \leqslant \delta N(\mathcal{G})$.
3. For each graph $\mathcal{G} \in \mathbf{G}$, $W_{\mathcal{G}}(\frac{1}{2}N(\mathcal{G})) \geqslant c(N(\mathcal{G}))^\varepsilon$.

We remark that the family of boolean hypercubes is algebraically decomposable, with $\beta = \gamma = \delta = c = 1/2$ and $\varepsilon = 1$; the family's recursive bisector is obtained by removing the edges that cross a single dimension, while the value of c is established via packing arguments in Section 4.2 and via congestion arguments in Section 4.3. Similarly, the family of rectangular two-dimensional meshes whose dimensions are powers of 2 is algebraically decomposable, with $\beta = \gamma = \delta = \varepsilon = 1/2$ and $c = 2(\sqrt{2} - 1)$: the family's recursive bisector is obtained by cutting each rectangle along its longer dimensions, while the value of c is established in Section 4.2.

Consider now any finite family of graphs $\mathbf{G}$ that comprises the smallest m graphs in an algebraically decomposable family $\mathbf{G}$ that has parameters β, ε, a, b, for some integer m. On the one hand, Theorem 2.5.4 establishes that there is a strongly universal I-hypergraph $\mathcal{I}(\mathbf{G})$ for family $\mathbf{G}$ of size

$$\begin{aligned}\text{SIZE}(\mathcal{I}(\mathbf{G})) &= N(\mathcal{G}_{\mathbf{G}}) \cdot \left(\sum_{k=1}^{r} \sum_{i=0}^{-k/\log(1-\beta)} S((1-\beta)^i 2^k) \right) \\ &= c(\beta, \varepsilon)(N(\mathcal{G}_{\mathbf{G}}))^{1+\varepsilon}\end{aligned}$$

where (1) $\mathcal{G}_{\mathbf{G}}$ is the largest graph in $\mathbf{G}$, and (2) the constant $c(\beta, \varepsilon) > 0$ is the reciprocal of $(2^\varepsilon - 1)(1 - (1 - \beta)^\varepsilon)$.[3]

Focus now on any graph $\mathcal{G} \in \mathbf{G}$ such that $\gamma N(\mathcal{G}_{\mathbf{G}}) \leqslant N(\mathcal{G}) \leqslant \delta N(\mathcal{G}_{\mathbf{G}})$. Since the I-hypergraph $\mathcal{I}(\mathbf{G})$ is strongly universal for family $\mathbf{G}$, we are assured that we can embed graph $\mathcal{G}$ into the following set of nodes of $\mathcal{I}(\mathbf{G})$: the leftmost $\lfloor \frac{1}{2}(N(\mathcal{G}_{\mathbf{G}}) - N(\mathcal{G})) \rfloor$ and the rightmost $\lceil \frac{1}{2}(N(\mathcal{G}_{\mathbf{G}}) - N(\mathcal{G})) \rceil$ nodes of $\mathcal{I}(\mathbf{G})$. Now, our ability to embed $\mathcal{G}$ into $\mathcal{I}(\mathbf{G})$ using these nodes implies that the I-hypergraph must have at least

$$W_{\mathcal{G}}(\tfrac{1}{2}N(\mathcal{G})) \geqslant c(N(\mathcal{G}))^\varepsilon \geqslant c\gamma^\varepsilon (N(\mathcal{G}_{\mathbf{G}}))^\varepsilon$$

hyperedges, each of size

$$\frac{1}{2}(N(\mathcal{G}_{\mathbf{G}}) - N(\mathcal{G})) \geq \frac{1-\delta}{2} N(\mathcal{G}_{\mathbf{G}})$$

This means, however, that

$$\text{SIZE}(\mathcal{I}(\mathbf{G})) \geq \frac{c}{2}\gamma^{\varepsilon}(1-\delta)(N(\mathcal{G}_{\mathbf{G}}))^{1+\varepsilon}$$

which is within a constant factor of the upper bound.

It follows that the construction of Section 2.5 is within a constant factor of optimal for algebraically decomposable graph families.

A.4.2. The Family of Binary Trees

The construction in Section 2.5 produces an I-hypergraph that is strongly universal for binary trees having at most N nodes, of SIZE proportional to $(N \log^2 N)$. (This is easily verified via the fact that the family of binary trees has a (1/3)-separator of size $S(n) \equiv 1$ (Valiant [1981]).) We prove in this section that no I-hypergraph that is strongly universal even for the family $\mathbf{T}_h$ of *complete* binary trees having depth no greater than h can have SIZE that is smaller by more than a constant factor.

Whereas the lower bound on the SIZEs of strongly universal I-hypergraphs for algebraically decomposable families (Section A.4.1) emerges just from considering the minimum bisection-widths of the graphs in the subject family $\mathbf{G}$, such consideration does not work with complete binary trees, due to their unit-size bisection-widths. Instead, we work here with three results developed in earlier chapters. Collectively, these results will help us establish that any I-hypergraph that is strongly universal for the family $\mathbf{T}_h$ must have SIZE proportional to $h^2 2^h = N \log^2 N$.

The first result we need comes from Section 4.4, where we proved that the (1/3)-separation-width of complete binary trees is logarithmic in the size of the tree. Specifically, we proved the following, as a special case of Theorem 4.4.1.

For all integers h, the (1/3)-separation-width of the height-h complete binary tree $\mathcal{T}_h$ is no smaller than[4]

$$W_{\mathcal{T}_h}\left(\frac{1}{2}\right) \geq \frac{1}{8}h - 2$$

Using the identical reasoning that yields this bound, one can establish the following family of bounds, whose proof is left to the reader.

PROPOSITION A.4.1. *For all integers h and all* $k \in \{2, 3, \ldots, h\}$, *the* $(1/(3 \cdot 2^k))$*-separation-width of the height-h complete binary tree* $\mathcal{T}_h$ *is no less than*

$$W_{\mathcal{T}_h}\left(\frac{1}{3 \cdot 2^k}\right) \geq \frac{1}{8}\left(h - 2k - \frac{19}{6}\right)$$

The second result on the road to our bound comes from Section 1.4, where we proved the following.

For any graph $\mathcal{G}$ *and for any integer k, the k-mincing-width of* $\mathcal{G}$ *can be no smaller than*

$$M_{\mathcal{G}}(k) \geq \frac{1}{2}\left((N(\mathcal{G}) \bmod k) W_{\mathcal{G}}\left(\left\lceil \frac{N(\mathcal{G})}{k} \right\rceil\right) + (k - (N(\mathcal{G}) \bmod k)) W_{\mathcal{G}}\left(\left\lceil \frac{N(\mathcal{G})}{k} \right\rceil\right)\right)$$

When we combine this bound with Proposition A.4.1, we obtain the following bounds on the mincing-width of complete binary trees.

PROPOSITION A.4.2. *For all integers h and all* $k \in \{2, 3, \ldots, h\}$, *the* $(3 \cdot 2^k)$*-mincing-width of the height-h complete binary tree* $\mathcal{T}_h$ *is no less than*

$$M_{\mathcal{T}_h}(3 \cdot 2^k) \geq \tfrac{3}{16} 2^k (h - 2k - b)$$

for some constant $b > 0$.

Finally, we invoke the following result, which is a specialization of Theorem 2.5.4 to families of complete binary trees, followed by some elementary arithmetic.

PROPOSITION A.4.3. *Define the sequence of integers* $c_0 < c_1 < \cdots < c_{h-2}$ *by*

$$c_i \stackrel{\text{def}}{=} 3 \cdot 2^i$$

for each index i. Any I-hypergraph $\mathcal{I}$ *that is strongly universal for the family* $\mathbf{T}_h$ *of complete binary trees must have*

$$\text{SIZE}(\mathcal{I}) \geq b_1 2^h \sum_{k=0}^{h-2} (h - 2k - b_2) \geq b_3 h^2 2^h$$

for some constants $b_1, b_2, b_3 > 0$.

Since 2^h is just a constant fraction of $N(\mathcal{T}_h)$, Proposition A.4.3 establishes the desired bound on the SIZE of $\mathcal{I}$.

A.5. Pebbling Games

The classical lower bounds on the number of enabling pebbles required for plays of the pebble game (e.g., in Cook [1974] and Paterson and Hewitt [1970]) do not derive from bounds on the separation-widths of the dags involved. It is gratifying (given the purpose of this book) to note that these pebble-number bounds *can* be derived from separation-width bounds, with the resulting benefit of creating a uniform framework for studying such problems. The following result surveys some of the pebble-number lower bounds one can derive via the separation-width bounds of Chapter 4. Note that most of the bounds in that chapter are monotonic in the size of the smaller subgraph produced by a separation, the bound on trees being the notable exception.

APPLICATION A.5.1. *The number of enabling pebbles in a play of the pebble game on an N-node directed acyclic version of the*

boolean hypercube	*can be no smaller than* $\frac{1}{2}N$
butterfly graph	*can be no smaller than* $N/\log N$
complete b-ary tree	*can be no smaller than* $\frac{1}{2}\log N - l.o.t.$
de Bruijn graph	*can be no smaller than* $\frac{1}{2}N/\log N$
d-dimensional mesh	*can be no smaller than* $2^{1/d}dN^{1-1/d}$
X-tree	*can be no smaller than* $\log(N-1+1/(N+1))-1$

VERIFICATION. The lower bounds on separation-width that we derived in Chapter 4 yield Application A.5.1 by elementary calculation. In particular, our expressions for the separation-widths of

- Complete binary trees are maximized when one partitions the tree into subgraphs whose sizes are in the ratio 1:2
- Complete b-ary trees, for any fixed $b \geq 3$, are maximized when one partitions the tree into subgraphs whose sizes are in the ratio $1: b-2$
- All other listed graphs are maximized when one *bisects* the graph

In seeking these maxima, we employed the bounds of Section 4.2 for X-trees, hypercubes, and multidimensional meshes, the bounds of Section 4.3 for butterfly and de Bruijn graphs, and the bounds of Section 4.4 for trees.

The proof is completed by evaluating the relevant expressions from Chapter 4 at their maximizing values. □

A.6. Sources

The entire development in Section A.4.2 comes from Chung and Rosenberg [1986]. The remainder of the appendix presents results that are largely known throughout the literature, though often via quite different proofs. Relevant citations appear *in situ.*

Notes

1. The reader should ignore all guest-host matchups wherein one of the listed graphs cannot exist; e.g., the number of nodes in a boolean hypercube must be a power of 2, while the number of nodes in a complete binary tree must be one less than a power of 2.
2. Our assumption that mesh sides are powers of 2 avoids a proliferation of floors and ceilings in what follows.
3. The value of $c(\beta, \varepsilon)$ is easily calculated since the forbidding double summation becomes a double geometric sum in this case.
4. Recall from Section 4.4 that our notation for the separation-widths of trees differs from our customary notation.

Bibliography

Abelson, H., and Andreae, P. [1980]. Information transfer and area-time tradeoffs for VLSI multiplication. *C. ACM* **23**, 20–23.

Aho, A. V., Garey, M. R., and Hwang, F. K. [1977]. Rectilinear Steiner trees: Efficient special-case algorithms. *Networks* **7**, 37–58.

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. [1974]. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass.

Aho, A. V., Ullman, J. D., and Yannakakis, M. [1983]. On notions of information transfer in VLSI circuits. *15th ACM Symp. on Theory of Computing*, pp. 133–139.

Aiello, W., and Leighton, F. T. [1991]. Coding theory, hypercube embeddings, and fault tolerance. *3rd ACM Symp. on Parallel Algorithms and Architectures*, pp. 125–136.

Aleksandrov, L., and Djidjev, H. [1996]. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Discr. Math.* **9**, 129–150.

Aleliunas, R., and Rosenberg, A. L. [1982]. On embedding rectangular grids in square grids. *IEEE Trans. Comp.* **C-31**, 907–913.

Alia, G., and Maestrini, P. [1976]. A procedure to determine optimal partitions of weighted hypergraphs through a network-flow analogy. *Estratto da Calcolo* **XIII,** 191–211.

Alon, N. [1986]. Eigenvalues and expanders. *Combinatorica* **6**, 83–96.

Alon, N., and Chung, F. R. K. [1988]. Explicit construction of linear sized tolerant networks. *Discrete Math.* **72,** 15–19.

Alon, N., Seymour, P., and Thomas, R. [1994]. Planar separators. *SIAM J. Discr. Math.* **7,** 184–193.

Alon, N., and West, D. B. [1986]. The Borsuk-Ulam theorem and bisection of necklaces. *Proc. Am. Math. Soc.* **98,** 623–628.

Annexstein, F. S., and Baumslag, M. [1993]. On the diameter and bisector size of Cayley graphs. *Math. Syst. Th.* **26**, 271–292.

Annexstein, F. S., Baumslag, M., and Rosenberg, A. L. [1990]. Group action graphs and parallel architectures. *SIAM J. Comput.* **19**, 544–569.

Antonelli, S., and Pelagatti, S. [1992]. On the complexity of the mapping problem for massively parallel architectures. *Int. J. Found. Comput. Sci.* **3**, 379–387.

Arora, S., Karger, D., and Karpinski, M. [1995]. Polynomial time approximation schemes for dense instances of NP-hard problems. *27th ACM Symp. on Theory of Computing*, pp. 284–293.

Avior, A., Calamoneri, T., Even, S., Litman, A., and Rosenberg, A. L. [1996]. A tight layout of the butterfly network. *8th ACM Symp. on Parallel Algorithms and Architectures*, pp. 170–175.

Awerbuch, B. Berger, B., Cowen, L., and Peleg, D. [1998]. Near-linear time construction of sparse neighborhood covers. *SIAM J. Comput.* **28**, 263–277.

Babai, L. [1991]. Local expansion of vertex-transitive graphs and random generation in finite groups. *23rd ACM Symp. on Theory of Computing*, pp. 164–174.

Babai, L., and Szegedy, M. [1991]. Local expansion of symmetrical graphs. Tech. Rpt. CS91-22, Department of Computer Science, Univ. Chicago.

Barnard, S. T., and Simon, H. D. [1994], Fast multilevel implementation of recursive bisection for partitioning unstructured problems. *Concurrency: Practice and Experience* **6**, 101–117.

Barnes, E. R. [1982]. An algorithm for partitioning the nodes of a graph. *SIAM J. Alg. Disc. Meth.* **3**, 541–550.

Beck, J. [1983]. On size Ramsey number of paths, trees, and circuits, I. *J. Graph Th.* **7**, 115–129.

Beck, J. [1990]. On size Ramsey number of paths, trees and circuits, II. *Mathematics of Ramsey Theory*, Springer, Berlin, pp. 34–45.

Beneš, V. E. [1964]. Optimal rearrangeable multistage connecting networks. *Bell Syst. Tech. J.* **43**, 1641–1656.

Bentley, J. L., and Kung, H. T. [1979]. A tree machine for searching problems. *Intl. Conf. on Parallel Processing*, pp. 257–266.

Berger, M. J., and Bokhari, S. H. [1987]. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Comp.* **C-36**, 570–580.

Berkman, O., and Vishkin, U. [1993]. Recursive star-tree parallel data structure. *SIAM J. Comput.* **22**, 221–242.

Berman, F., and Snyder, L. [1987]. On mapping parallel algorithms into parallel architectures. *J. Parallel Distr. Comput.* **4**, 439–458.

Bermond, J.-C., and Peyrat, C. [1989]. The de Bruijn and Kautz networks: A competitor for the hypercube? In *Hypercube and Distributed Computers* (F. Andre and J. P. Verjus, eds.), North-Holland, Amsterdam, pp. 279–293.

Berry, J. W., and Goldberg, M. K. [1999]. Path optimization for graph partitioning problems. *Discr. Appl. Math.* **90**, 27–50.

Bertele, U., and Brioschi, F. [1972]. *Nonserial Dynamic Programming*. Academic Press, New York.

Bhatt, S. N., Chung, F. R. K., Hong, J.-W., Leighton, F. T., Obrenić, B., Rosenberg, A. L., and Schwabe, E. J. [1996]. Optimal emulations by butterfly-like networks. *J. ACM* **43**, 293–330.

Bhatt, S. N., Chung, F. R. K., Leighton, F. T., and Rosenberg, A. L. [1992]. Efficient embeddings of trees in hypercubes. *SIAM J. Comput.* **21**, 151–162.

Bhatt, S. N., Chung, F. R. K., Leighton, F. T., and Rosenberg, A. L. [1996]. Scheduling tree-dags using FIFO queues: A control-memory tradeoff. *J. Parallel Distr. Comput.* **33**, 55–68.

Bhatt, S. N., Greenberg, D. S., Leighton, F. T., and Liu, P. [1991]. Tight bounds for on-line tree embeddings. *2nd ACM-SIAM Symp. on Discrete Algorithms*, pp. 344–350.

Bhatt, S. N., and Leighton, F. T. [1984]. A framework for solving VLSI graph layout problems. *J. Comp. Syst. Sci.* **28**, 300–343.

Bhatt, S. N., and Leiserson, C. E. [1984]. How to assemble tree machines. In *Advances in Computing Research* 2 (F. P. Preparata, ed.) JAI Press, Greenwich, CT, 95–114.

Bilardi, G. [1985]. *The Area-Time Complexity of Sorting*. Ph.D. thesis, Univ. Illinois.

Blum, N. [1985]. An area-maximum edge length tradeoff for VLSI layout. *Inform. Contr.* **66**, 45–52.

Blumofe, R., and Toledo, S. [1992]. Personal communication.

Bokhari, S. H. [1981]. On the mapping problem. *IEEE Trans. Comp.* **C-30**, 207–214.

Boppana, R. B. [1987]. Eigenvalues and graph bisection: An average-case analysis. *28th IEEE Symp. on Foundations of Computer Science*, pp. 280–285.

Browning, S. A. [1980]. *The Tree Machine: A Highly Concurrent Computing Environment.* Ph.D. thesis, CalTech.

Bruck, J., Cypher, R., and Ho, C.-T. [1993]. Fault-tolerant meshes and hypercubes with minimal numbers of spares. *IEEE Trans. Comp.* **C-42**, 1089–1104.

Bui, T. N. [1983]. *On Bisecting Random Graphs.* M.S. thesis, MIT.

Bui, T. N. [1986], *Graph Bisection Algorithms.* Ph.D. thesis, MIT.

Bui, T. N., Chaudhuri, S., Leighton, F. T., and Sipser, M. [1987]. Graph bisection algorithms with good average case behavior. *Combinatorica* **7**, 171-191.

Bui, T. N., Heigham, C., Jones, C., and Leighton, F. T. [1989]. Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms. *26th ACM-IEEE Design Automation Conf.*, pp. 775–778.

Bui, T. N., and Jones, C. [1992]. Finding good approximate vertex and edge partitions is NP-hard. *Inform. Proc. Let.* **42**, 153–159.

Bui, T. N., and Moon, B. R. [1996]. Genetic algorithm and graph partitioning. *IEEE Trans. Comp.* **45**, 841–855.

Bui, T. N., and Peck, A. [1988]. Algorithms for bisecting planar graphs. *26th Ann. Allerton Conference on Communication,* Control*, and Computing*, pp. 798–807.

Bui, T. N., and Peck, A. [1992]. Partitioning planar graphs. *SIAM J. Comput.* **21**, 203–215.

Burstein, M. [1981]. Partitioning of VLSI networks. IBM Report RC-9180.

Carlson, D. A. [1984]. Parallel processing of tree-like computations. *4th Intl. Conf. on Distributed Computing Systems.*

Chan, M. Y. [1991]. Embedding of grids into optimal hypercubes. *SIAM J. Comput.* **20**, 834–864.

Chung, F. R. K. [1989]. Improved separators for planar graphs. Typescript, Bell Communications Research.

Chung, F. R. K., Füredi, Z., Graham, R. L., and Seymour, P. [1988]. On induced subgraphs of the cube. *J. Comb. Th. (A)* **49**, 180–187.

Chung, F. R. K., and Rosenberg, A. L. [1986]. Minced trees, with applications to fault-tolerant VLSI processor arrays. *Math. Syst. Th.* **19**, 1–12.

Chung, F. R. K., and Yau, S.-T. [1994]. A near optimal algorithm for edge separators. *26th ACM Symp. on Theory of Computing*, pp. 1–8.

Cole, R., and Siegel, A. [1988]. Optimal VLSI circuits for sorting. *J. ACM* **35**, 777–809.

Cook, S. A. [1974]. An observation on time-storage tradeoff. *J. Comp. Syst. Sci.* **9**, 308–316.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. [1990]. *Introduction to Algorithms.* McGraw-Hill, New York.

Dally, W. J., and Seitz, C. L. [1986]. The torus routing chip. *J. Distributed Systems* **1,** 187–196.

David, V., Fraboul, Ch., Rousselot, J. Y., and Siron, P. [1992]. Partitioning and mapping communication graphs on a modular reconfigurable parallel architecture. *Parallel Processing: CONPAR 92–VAPP V. Lecture Notes in Computer Science 634*, Springer-Verlag, Berlin, pp. 43–48.

de Bruijn, N. G. [1946]. A combinatorial problem. *Proc. Koninklijke Nederlandische Akademe van Wetenschappen (A)* **49**, Part 2, 758–764.

DeGroot, D. [1983]. Partitioning job structures for SW-Banyan networks. *Intl. Conf. on Parallel Processing*, *pp.* 106–113.

DeMillo, R. A., Eisenstat, S. C., and Lipton, R. J. [1978a]. Preserving average proximity in arrays. *C. ACM* **21**, 228–231.

DeMillo, R. A., Eisenstat, S. C., Lipton, R. J. [1978b]. On small universal data structures and related combinatorial problems. *Johns Hopkins Conf. on Inform. Sci. and Syst.*, pp. 408–411.

Despain, A. M., and Patterson, D. A. [1978]. X-tree—a tree structured multiprocessor architecture. *5th Intl. Symp. on Computer Architecture*, pp. 144–151.

Diks, K., Djidjev, H. N., Sykora, O., and **Vrťo, I.** [1988]. Edge separators for planar graphs and their applications. *1988 Conf. on Mathematical Foundations of Computer Science*, pp. 280–290.

Diks, K., Djidjev, H. N., Sykora, O., and **Vrťo,** I. [1993]. Edge separators of planar and outerplanar graphs with applications. *J. Algorithms* **14**, 258–279.

Djidjev, H. N. [1988]. Linear algorithms for graph separation problems. In *1st Scandinavian Wkshp. on Algorithm Theory*, *Lecture Notes in Computer Science 318*, Springer-Verlag, Berlin, pp. 216–222.

Donath, W. E., and Hoffman, A. J. [1973]. Lower bounds for the partitioning of graphs. *IBM J. Res. Devel.* **17**, 420–425.

Edelsbrunner, H. [1987]. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin.

Ellis, J. A., Sudborough, I. H., and Turner, J. S. [1994]. The vertex separation and search number of a graph. *Inform. Comput.* **113**, 50–79.

Eppstein, D., Miller, G. L., and Teng, S.-H. [1995]. A deterministic linear time algorithm for geometric separators and its applications. *Fund. Informat.* **22,** 309–329.

Etchells, R. D., Grinberg, J., and Nudd, G. R. [1981]. Development of a three-dimensional circuit integration technology and computer architecture. *Soc. Photogr. Instrum. Eng.,* **282**, 64–72.

Even, G., Naor, J., Rao, S., and Schieber, B. [1999]. Fast approximate graph partitioning algorithms. *SIAM J. Comput.* **28**, 2187–2214.

Fejes Tóth, L. [1964]. *Regular Figures.* Pergamon Press, Oxford.

Feldmann, R., and Unger, W. [1992]. The cube-connected cycles network is a subgraph of the butterfly network. *Parallel Proc. Lett.* **2**, 13–19.

Fellows, M. R., and Langston, M. A. [1988]. Processor utilization in a linearly connected parallel processing system. *IEEE Trans. Comp.* **37**, 594–603.

Fiduccia, C. M., and Mattheyses, R. M. [1982]. A linear-time heuristic for improving network partitions. *19th ACM-IEEE Design Automation Conf.*, pp. 175–181.

Fiedler, M. [1973]. Algebraic connectivity of graphs. *Czechoslovak Math. J.* **23**, 298–305.

Fiedler, M. [1975a]. Eigenvectors of acyclic matrices. *Czechoslovak Math. J.* **25**, 607–618.

Fiedler, M. [1975b]. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Math. J.* **25**, 619–633.

Filotti, I. S., Miller, G. L., and Reif, J. [1979]. On determining the genus of a graph in $O(v^{O(g)})$ steps. *11th ACM Symp. on Theory of Computing*, pp. 27–37.

Franklin, M. A., Wann, D. F., and Thomas, W. J. [1982]. Pin limitations and partitioning of VLSI interconnection networks. *IEEE Trans. Comp.* **C-31**, 1109–1116.

Friedman, J., and Pippenger, N. [1987]. Expanding graphs contain all small trees. *Combinatorica* **7**, 71–76.

Frieze, A., and Jerrum, M. [1997]. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. *Algorithmica* **18**, 67–81.

Galil, Z., Kannan, R., and Szemerédi, E. [1989a]. On nontrivial separators for k-page graphs and simulations by nondeterministic one-tape Turing machines. *J. Comp. Syst. Sci.* **38**, 134–149.

Galil, Z., Kannan, R., and Szemerédi, E. [1989b]. On 3-pushdown graphs with large separators. *Combinatorica* **9**, 9–19.

Ganley, J. L., and Heath, L. S. [1994a]. Heuristics for laying out information graphs. *Computing* **52**, 389–405.

Ganley, J. L., and Heath, L. S. [1994b]. Optimal and random partitions of random graphs. *Computer J.* **37**, 641–643.

Ganley, J. L., and Heath, L. S. [1998]. An experimental evaluation of local search heuristics for graph partitioning. *Computing* **60**, 121–132.

Gannon, D. [1980]. A note on pipelining a mesh-connected multiprocessor for finite element problems by nested dissection. *Intl. Conf. on Parallel Processing,* pp. 197–204.

Garey, M. R., and Johnson, D. S. [1979]. *Computers and Intractability.* W.H. Freeman, San Francisco.

Garey, M. R., Johnson, D. S., and Stockmeyer, L. [1976]. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.* **1**, 237–267.

Garg, N., Saran, H., and Vazirani, V. V. [1994]. Finding separator cuts in planar graphs within twice the optimal. *35th IEEE Symp. on Foundations of Computer Science,* pp. 14–23.

Gilbert, J. R., Hutchinson, J. P., and Tarjan, R. E. [1984]. A separator theorem for graphs of bounded genus. *J. Algorithms* **5**, 391–407.

Glover, F. [1989]. Tabu search — Part I. *ORSA J. Computing* **1**, 190–206.

Glover, F. [1990]. Tabu search — Part II. *ORSA J. Computing* **2**, 4–32.

Goemans, M. X., and Williamson, D. P. [1995]. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM* **42**, 1115–1145.

Goldberg, C. H., and West, D. B. [1985]. Bisection of circle colorings. *SIAM J. Algebr. Discr. Meth.* **6**, 93–106.

Goldberg, D. E. [1989]. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass.

Goldberg, M. K., and Burstein, M. [1983]. Heuristic improvement technique for bisection of VLSI networks. *IEEE Intl. Conf. on Computer Design: VLSI in Computers*, pp. 122–125.

Gottlieb, A. [1986]. An overview of the NYU Ultracomputer project. Ultracomputer Note No. 100, New York Univ.

Greenberg, D. S., Heath, L. S., and Rosenberg, A. L. [1990]. Optimal embeddings of butterfly-like graphs in the hypercube. *Math. Syst. Th.* **23**, 61–77.

Greenberg, R. I., and Leiserson, C. E. [1988]. A compact layout for the three-dimensional tree of meshes. *Appl. Math. Lett.* **1**, 171–176.

Gremban, K. D., Miller, G. L., and Teng, S.-H. [1997]. Moments of inertia and graph separators. *J. Comb. Optim.* **1**, 79–104.

Gross, J. L., and Tucker, T. W. [1987]. *Topological Graph Theory.* Wiley, New York.

Grünbaum, B. [1967]. *Convex Polytopes.* Wiley, New York.

Guattery, S. [1998a]. Graph embeddings and Laplacian eigenvalues. ICASE Report No. 98-23.

Guattery, S. [1998b]. Graph embeddings, symmetric real matrices, and generalized inverses. ICASE Report No. 98–34.

Guattery, S., and Miller, G. L. [1998]. On the quality of spectral separators. *SIAM J. Matrix Anal. Appl.* **19,** 701–719.

Hamidoune, Y. O., and Serra, O. [1996]. On small cuts separating an abelian Cayley graph into equal parts. *Math. Syst. Th.* **29**, 407–409.

Hardy, G. H., Littlewood, J. E., and Pólya, G. [1952]. *Inequalities.* Cambridge University Press, Cambridge.

Harper, L. H. [1964]. Optimal assignments of numbers to vertices. *J. Soc. Ind. Appl. Math.* **12**, 131–135.

Harper, L. H. [1966]. Optimal numberings and isoperimetric problems on graphs. *J. Comb. Th.* **1**, 385–393.

Harper, L. H. [1967]. A necessary condition on minimal cube numberings. *J. Appl. Prob.* **4**, 397–401.

Heath, L. S. [1997]. Graph embeddings and simplicial maps. *Theory of Comp. Syst.* **30**, 599–625.

Heath, L. S., and Istrail, S. [1992]. The pagenumber of genus g graphs is $O(g)$. *J. ACM* **39**, 479–501.

Heath, L. S., Leighton, F. T., and Rosenberg, A. L. [1992]. Comparing queues and stacks as mechanisms for laying out graphs. *SIAM J. Discr. Math.* **5**, 398–412.

Heath, L. S., Rosenberg, A. L., and Smith, B. T. [1988]. The physical mapping problem for parallel architectures. *J. ACM* **35**, 603–634.

Hendrickson, B., and Leland, R. [1995]. An improved spectral graph partitioning algorithm for mapping parallel algorithms. *SIAM J. Sci. Comput.* **16**, 452–469.

Henle, M. [1979]. *A Combinatorial Introduction to Topology.* W.H. Freeman, San Francisco.

Hong, J.-W., Mehlhorn, K., and Rosenberg, A. L. [1983]. Cost trade-offs in graph embeddings, with applications. *J. ACM* **30**, 709–728.

Hong, J.-W., and Rosenberg, A. L. [1982]. Graphs that are almost binary trees. *SIAM J. Comput.* **11**, 227–242.

Hromkovič, J. [1991]. Nonlinear lower bounds on the number of processors of circuits with sublinear separators. *Inform. Comput.* **95**, 117–128.

Iordansk'ii, M. A. [1976]. Minimal numeration of tree vertices (Minimalnye numeratsii vershin derevyev; in Russian). *Prob. Kibernet.* **31**, 109–132.

Iri, M. [1967]. On an extension of the maximum-flow minimum-cut theorem to multicommodity flows. *J. Oper. Res. Soc. Jpn.* **13**, 129–135.

JáJá, J., and Prasanna Kumar, V. K. [1984]. Information transfer in distributed computing with applications to VLSI. *J. ACM* **31**, 150–162.

Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. [1989]. Optimization by simulated annealing: Part I, Graph partitioning. *Oper. Res.* **37**, 865–892.

Johnsson, S. L. [1987]. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distr. Comput.* **4**, 133–172.

Karypis, G., and Kumar, V. [1999a]. Parallel multilevel k-way partitioning scheme for irregular graphs. *SIAM Rev.* **41**, 278–300.

Karypis, G., and Kumar, V. [1999b]. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* **20**, 359–392.

Kernighan, B. W., and Lin, S. [1970]. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.* **49**, 291–307.

Klein, P., Rao, S., Agrawal, A., and Ravi, R. [1995]. An approximate max-flow min-cut relation for undirected multicommodity flow, with applications. *Combinatorica* **15**, 187–202.

Knuth, D. E. [1973]. *The Art of Computer Programming, I: Fundamental Algorithms.* Addison-Wesley, Reading, Mass.

Koch, R., Leighton, F. T., Maggs, B., Rao, S., and Rosenberg, A. L. [1997]. Work-preserving emulations of fixed-connection networks. *J. ACM* **44**, 104–147.

Kosaraju, S. R., and Atallah, M. J. [1988]. Optimal simulations between mesh-connected arrays of processors. *J. ACM* **35**, 635–650.

Krishnamurthy, B. [1984]. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. Comp.* **C-33**, 438–446.

Kunde, M. [1993]. Block gossiping on grids and tori: Deterministic sorting and routing match the bisection bound. *1st European Symp. on Algorithms*, Bad Honnef, Germany (T. Lengauer, ed.) *Lecture Notes in Computer Science 726*, Springer-Verlag, Berlin, pp. 272–283.

Kung, H. T., and Picard, R. L. [1984]. One-dimensional systolic arrays for multidimensional convolution and resampling. In *VLSI for Pattern Recognition and Image Processing*, Springer-Verlag, Berlin, pp. 9–24.

Kung, H. T., and Stevenson, D. [1977]. A software technique for reducing the routing time on a parallel computer with a fixed interconnection network. In *High Speed Computer and Algorithm Organization*, Academic Press, New York, pp. 423–433.

Leighton, F. T. [1982]. A layout strategy for VLSI which is provably good. *14th ACM Symp. on Theory of Computing*, pp. 85–98.

Leighton, F. T. [1983]. *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.* MIT Press, Cambridge, Mass.

Leighton, F. T. [1992]. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, San Mateo, Calif.

Leighton, F. T., Makedon, F., Plotkin, S., Stein, C., Tardos, E., and Tragoudas, S. [1995]. Fast approximation algorithms for multicommodity flow problems. *J. Comp. Syst. Sci.* **50**, 228–243.

Leighton, F. T., Makedon, F., and Tragoudas, S. [1990]. Approximation algorithms for VLSI partition problems. *IEEE Intl. Symp. on Circuits and Systems*, pp. 2865–2868.

Leighton, F. T., and Rao, S. [1988]. An approximate max-flow min-cut theorem for uniform multicommodity flow problems, with applications to approximation algorithms. *29th IEEE Symp. on Foundations of Computer Science,* pp. 422–431.

Leighton, F. T., and Rosenberg, A. L. [1983]. Automatic generation of three-dimensional circuit layouts. *IEEE Intl. Conf. on Computer Design: VLSI in Computers*, pp. 633–636.

Leighton. F. T., and Rosenberg, A. L. [1986]. Three-dimensional circuit layouts. *SIAM J. Comput.* **15,** 793–813.

Leiserson, C. E. [1983]. *Area-Efficient VLSI Computation.* MIT Press, Cambridge, Mass.

Leiserson, C. E. [1985]. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comp.* **C-34**, 892–901.

Lempel, A. [1970]. On a homomorphism of the de Bruijn graph and its applications to the design of feedback shift registers. *IEEE Trans. Comp.* **C-19,** 1204–1209.

Lengauer, T. [1981]. Black-white pebbles and graph separation. *Acta Inform.* **16,** 465–475.

Lipton, R. J., Eisenstat, S. C., and DeMillo, R. A. [1976]. Space and time hierarchies for classes of control structures and data structures. *J. ACM* **23**, 720–732.

Lipton, R. J., Sedgewick, R. [1981]. Lower bounds for VLSI. *13th ACM Symp. on Theory of Computing*, pp. 300–307.

Lipton, R. J., and Tarjan, R. E. [1979]. A separator theorem for planar graphs. *SIAM J. Appl. Math.* **36**, 177–189.

Lipton, R. J., and Tarjan, R. E. [1980]. Applications of a planar separator theorem. *SIAM J. Comput.* **9**, 615–627.

Massey, W. S. [1967]. *Algebraic Topology: An Introduction.* Harcourt, Brace & World, New York.

McBride, R. D. [1998]. Progress made in solving the multicommodity flow problem. *SIAM J. Optim.* **8**, 947–955.

Miller, G. L. [1986]. Finding small simple cycle separators for 2-connected planar graphs. *J. Comp. Syst. Sci.* **32**, 265–279.

Miller, G. L., Teng, S.-H., and Vavasis, S. A. [1991]. A unified geometric approach to graph separators. *32nd IEEE Symp. on Foundations of Computer Science*, pp. 538–547.

Miller, G. L., and Thurston, W. [1990]. Separators in two and three dimensions. *22nd ACM Symp. on Theory of Computing,* pp. 300–309.

Miller, G. L., and Vavasis, S. A. [1991]. Density graphs and separators. *2nd ACM-SIAM Symp. on Discrete Algorithms*, pp. 331–336.

Miranker, W. L., and Winkler, A. [1984]. Spacetime representations of computational structures. *Computing* **32**, 93–114.

Mohar, B. [1988]. The Laplacian spectrum of graphs. *Graph Theory, Combinatorics, and Applications.* Wiley, New York, pp. 871–898.

Mohar, B. [1989]. Isoperimetric numbers of graphs. *J. Comb. Th. (B)* **47**, 274–291.

Moldovan, D. I., and Fortes, J. A. B. [1986]. Partitioning and mapping algorithms into fixed size systolic arrays. *IEEE Trans. Comp.* **C-35,** 1–12.

Obrenic, B. [1994]. An approach to emulating separable graphs. *Math. Syst. Th.* **27**, 41–63.

Papadimitriou, C. H., and Ullman, J. D. [1987]. A communication-time tradeoff. *SIAM J. Comput.* **16,** 639–646.

Park, J. K., and Phillips, C. A. [1993]. Finding minimum-quotient cuts in planar graphs. *25th ACM Symp. on Theory of Computing,* pp. 766–775.

Paterson, M. S., and Hewitt. C. E. [1970]. Comparative schematology. *Project MAC Conf. on Concurrent Systems and Parallel Computation*, ACM Press, pp. 119–128.

Peterson, G. L., and Ting, Y.-H. [1982]. Trade-offs in VLSI for bus communication networks. Tech. Rpt. 111, Univ. Rochester.

Peterson, W. W., and Weldon, E. J. [1981]. *Error-Correcting Codes.* MIT Press, Cambridge, Mass.

Plaisted, D. A. [1990]. A heuristic algorithm for small separators in arbitrary graphs. *SIAM J. Comput.* **19,** 267–280.

Poljak, S., and Tuza, Z. [1995]. Maximum cuts and large bipartite subgraphs. *Combinatorial Optimization* (W. Cook, L. Lovász, and P. Seymour, eds.), Amer. Math. Soc., Providence, Rhode Island, pp. 181–244.

Preparata, F. P. [1983]. Optimal three-dimensional VLSI layouts. *Math. Syst. Th.* **16,** 1–8.

Preparata, F. P., and Vuillemin, J. E. [1981]. The cube-connected cycles: A versatile network for parallel computation. *C. ACM* **24**, 300–309.

Quinton, P. [1984]. Automatic synthesis of systolic arrays from uniform recurrence equations. *11th IEEE Intl. Symp. on Computer Architecture*, pp. 208–214.

Quinton, P., and VanDongen, V. [1989]. The mapping of linear recurrence equations on regular arrays. *J. VLSI Signal Processing* **1**, 95–113.

Rao, S. [1987]. Finding near optimal separators in planar graphs. *28th IEEE Symp. on Foundations of Computer Science*, pp. 225–237.

Rao, S. B. [1992]. Faster algorithms for finding small edge cuts in planar graphs. *24th ACM Symp. on Theory of Computing*, pp. 229–240.

Rettberg, R. D. [1986]. Shared memory parallel processors: the Butterfly and the Monarch. *4th MIT Conf. on Advanced Research in VLSI* (C. E. Leiserson, ed.) MIT Press, Cambridge, Mass., p. 45.

Richards, D. [1986]. Finding short cycles in planar graphs using separators. *J. Algorithms* **7**, 382–394.

Rosenberg, A. L. [1975]. Preserving proximity in arrays. *SIAM J. Comput.* **4**, 443–460.

Rosenberg, A. L. [1978]. Data encodings and their costs. *Acta Inform.* **9**, 273–292.

Rosenberg, A. L. [1979a]. Encoding data structures in trees. *J. ACM* **26**, 668–689.

Rosenberg, A. L. [1979b]. On embedding graphs in grids. IBM Report RC-7559.

Rosenberg, A. L. [1981a]. Issues in the study of graph embeddings. In *Graph-Theoretic Concepts in Computer Science: Proceedings of the International Workshop WG80*, Bad Honnef, Germany (H. Noltemeier, ed.) *Lecture Notes in Computer Science 100*, Springer-Verlag, Berlin, pp. 150–176.

Rosenberg, A. L. [1981b]. Routing with permuters: Toward reconfigurable and fault-tolerant networks. Tech. Rpt. CS-1981-13, Duke Univ.

Rosenberg, A. L. [1983]. Three-dimensional VLSI: A case study. *J. ACM* **30**, 397–416.

Rosenberg, A. L. [1985]. A hypergraph model for fault-tolerant VLSI processor arrays. *IEEE Trans. Comp.* **C-34,** 578–584.

Rosenberg, A. L. [1989]. Interval hypergraphs. In *Graphs and Algorithms* (R. B. Richter, ed.) *Contemporary Mathematics 89*, Amer. Math. Soc., Providence, Rhode Island, pp. 27–44.

Rosenberg, A. L. [1992]. Product-shuffle networks: Toward reconciling shuffles and butterflies. *Discr. Appl. Math.* **37/38**, 465–488.

Rosenberg, A. L. and Snyder, L. [1978]. Bounds on the costs of data encodings. *Math. Syst. Th.* **12**, 9–39.

Rosenberg, A. L., and Sudborough, I. H. [1983]. Bandwidth and pebbling. *Computing* **31**, 115–139.

Rosenberg, A. L., Wood, D., and Galil, Z. [1979]. Storage representations for tree-like data structures. *Math. Syst. Th.* **13**, 105–130.

Rosenthal, A. [1982]. Dynamic programming is optimal for nonserial optimization problems. *SIAM J. Comput.* **11,** 47–59.

Saab, Y. G. [1995]. A fast and robust network bisection algorithm. *IEEE Trans. Comp.* **C-44,** 903–913.

Sadayappan, P., Ercal, F., and Ramanujam, J. [1989]. Parallel graph partitioning on a hypercube. *4th Conf. on Hypercube Concurrent Computation and Applications*, pp. 67–70.

Sarkar, V. [1989]. *Partitioning and Scheduling Parallel Programs for Multiprocessors.* MIT Press, Cambridge, Mass.

Sarkar, V., and Hennessy, J. [1986]. Compile-time partitioning and scheduling of parallel programs. *SIGPLAN Notices* **21** [7), 17–26.

Savage, J. E. [1984]. The performance of multilective VLSI algorithms. *J. Comp. Syst. Sci.* **29,** 243–273.

Savage, J. E., and Wloka, M. G. [1991]. Parallelism in graph-partitioning. *J. Parallel Distr. Comput.* **13**, 257–272.

Schwabe, E. J. [1993]. Constant-slowdown simulations of normal hypercube algorithms on the butterfly network. *Inform. Proc. Lett.* **45**, 295–301.

Schwartz, J. T. [1980]. Ultracomputers. *ACM Trans. Prog. Lang.* **2**, 484–521.

Seitz, C. L. [1985]. The cosmic cube. *C. ACM* **28**

Teng, S.-H. [1998]. Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation. *SIAM J. Sci. Comput.* **19**, 635–656.

Thomassen, C. [1989]. The graph genus problem is NP-complete. *J. Algorithms* **10**, 568–576.

Thompson, C. D. [1980]. *A Complexity Theory for VLSI.* Ph.D. thesis, CMU.

Ullman, J. D. [1984]. *Computational Aspects of VLSI.* Computer Science Press, Rockville, Md.

Valiant, L. G. [1981]. Universality considerations in VLSI circuits. *IEEE Trans. Comp.* **C-30,** 135–140.

van Laarhoven, P. J. M., and Aarts, E. H. L. [1987]. *Simulated Annealing: Theory and Applications.* D. Reidel, Boston, Mass.

Vuillemin, J. [1983]. A combinatorial limit to the computing power of VLSI circuits. *IEEE Trans. Comp.* **C-32,** 294–300.

Wagner, D., and Wagner, F. [1993]. Between min cut and graph bisection. *1993 Conf. on Mathematical Foundations of Computer Science* (A.M. Borzyszkowski and S. Sokolowski, eds.) *Lecture Notes in Computer Science 711,* Springer-Verlag, Berlin, 744–750.

Weste, N., and Eshraghian, K. [1988]. *Principles of CMOS VLSI Design.* Addison-Wesley, Reading, Mass.

White, A. T. [1984]. *Graphs, Groups and Surfaces.* Elsevier, Amsterdam, Holland.

Wu, A. Y. [1985]. Embedding of tree networks into hypercubes. *J. Parallel Distr. Comput.* **2**, 238–249.

Yoeli, M. [1962]. Binary ring sequences. *Amer. Math. Monthly* **69,** 852–855.

About the Authors

Arnold L. Rosenberg received a B.A. in mathematics from Harvard College in 1962, and an M.A. and Ph.D. in applied mathematics from Harvard University, in 1963 and 1966, respectively.

Dr. Rosenberg is Distinguished University Professor of Computer Science at the University of Massachusetts at Amherst, where he codirects the Theoretical Aspects of Parallel and Distributed Systems (TAPADS) Laboratory. Prior to his tenure at the University of Massachusetts, he was a professor of computer science at Duke University from 1981 to 1986, and a research staff member at the IBM Watson Research Center from 1965 to 1981. He held visiting positions at Yale University and the University of Toronto; he was a Lady Davis visiting professor at the Technion (Israel Institute of Technology) and a Fulbright research scholar at the University of Paris-South.

Dr. Rosenberg's research focuses on theoretical aspects of parallel architectures and communication networks, with emphasis on developing algorithmic techniques for designing better networks and architectures and using them more efficiently. He is the author of more than 130 technical papers on these and other topics in theoretical computer science and discrete mathematics.

Dr. Rosenberg is a Fellow of the ACM, a Fellow of the IEEE, a Golden Core member of the IEEE Computer Society, and a member of SIAM. He has just ended a 12-year stint as editor-in-chief of *Theory of Computing Systems* (formerly, *Mathematical Systems Theory*); he continues to serve on the editorial boards of *TOCS* and other journals. Information on his publications and other activities can be found at **Http://www.cs.umass.edu/~rsnbrg/.**

Lenwood S. Heath received a B.S. in mathematics from the University of North Carolina in 1975, an M.S. in mathematics from the University of Chicago in 1976, and a Ph.D. in computer science from the University of North Carolina in 1985.

Dr. Heath is an Associate Professor of Computer Science at Virginia Tech. Prior to his tenure at Virginia Tech, he was an Instructor of Applied Mathematics at the Massachusetts Institute of Technology and a member of the MIT Laboratory of Computer Science.

Dr. Heath's research has been in various areas of theoretical computer science, mostly emphasizing graphs and algorithms. He has published in the areas of graph theory, complexity theory, computational algebra, computational biology, parallel architectures, graph embeddings, topology, computational geometry, and experimental algorithmics. Dr. Heath is currently concentrating on the Hopf project, an NSF sponsored project that is developing a computational algebra for noncommutative algebras, with an emphasis on new and improved algorithms for algebraic computations.

Dr. Heath is a member of the ACM, a senior member of the IEEE, and a member of SIAM. Information on his publications and other activities can be found at **Http://www.cs.vt.edu/~heath/.**

INDEX

Adjacent nodes, 3
Algebraic approach, 100, 159
Algebraically decomposable graph, 238
Algorithm APPROXIMATE-SEPARATOR, 147
Algorithm BISECT-REGULAR, 137
Algorithm BUCKET, 59
Algorithm CYCLE-FINDING, 120;
Algorithm DYNPROG, 51
Algorithm FIND-SUBGRAPH, 142
Algorithm FM-STEP, 157
Algorithm FM, 155, 156
Algorithm KL-STEP, 151
Algorithm KL, 150
Algorithm MIN-QUOTIENT-SEPARATOR, 146
Algorithm PLANAR-SEPARATOR, 118
Anti-symmetry, 131, 138
Applications of graph separators, 47, 229
Approximation, 131, 138, 141, 158, 159
Approximation to NP-hard problem, 107
Arc, 3
Arity of a (node in a) tree, 9, 163, 181

Balance, 17, 54, 55, 71
Bandwidth, 29, 33, 66, 96, 231
$\mathcal{Q}_n$: Base-2 (boolean) n-dimensional hypercube, 7
$\mathcal{B}_n$: Base-2 order-n butterfly graph, 11
$\mathcal{D}_n$: Base-2 order-n de Bruijn graph, 10
$\mathcal{B}_n(b)$: Base-b order-n butterfly graph, 11
$\mathcal{D}_n(b)$: Base-b order-n de Bruijn graph, 10
$\mathcal{Q}_n(b)$: Base-b n-dimensional hypercube, 7
Bifurcator, 16, 17, 24, 45, 63, 69, 234
Binary search tree traversal, 187
Binary tree, 17, 26, 33, 53, 78, 83, 84, 88, 94, 113, 165, 181, 185, 191, 205, 211, 212, 226, 231, 239, 241
Bisection-width, 13, 45, 53, 54, 65, 66, 76, 78, 133, 165, 191, 193, 194, 195, 230, 231
Breadth-first tree, 113, 115, 117, 118, 141, 142, 187
Bucket tree, 57, 58, 59, 61, 62, 63, 64, 94
Butterfly graph, 11, 31, 37, 38, 39, 55, 191, 202, 224, 230, 231, 241
  with wraparound, 11, 202
  without wraparound: the FFT graph, 12, 207

Capacity constraint, 131, 138
Capacity function, 131, 149
Capacity of the cut, 132
Cardinality, 3
2-Cell embedding, 110
Centerpoint, 124, 159
Child node in a tree, 9
Clique, 5, 32, 108, 191, 193
Coding theory, 44
k-Color recursive node-bisector, 56
Column-edge, 6
Compact surface, 110
Complete β-ary tree, 212, 213, 241
Complete binary tree, 17, 19, 21, 22, 26, 33, 53, 54, 55, 57, 61, 67, 78, 94, 181, 185, 211, 230, 231, 239, 240, 241
Complete bipartite graph, 5, 191, 193, 208
Complete graph, 5, 193
Complete ternary tree, 33, 205, 211, 230

Complete tree, 7
  β-ary, 8, 181, 212, 241
  binary, 9, 165, 181, 182, 185, 190, 239
  ternary, 230
Computation digraph, 221
Congestion argument, 190
  binary trees, 205
  butterfly graphs, 202
  de Bruijn graphs, 200
  hypercubes, 199
  I/O congestion: FFT graph, 207
  mesh-of-cliques, 196
  product-shuffle graphs, 209
  toroidal meshes, 197
Congestion of a graph embedding, 53, 64, 65, 191, 230
Connected graph, 4
Cross edge in a butterfly-like graph, 11, 12
Crossing the cut, 132
Cube-connected cycles graph, 12, 37, 38, 192
Cumulative-cost of an embedding, 30, 66, 232
Cut, 132, 139, 149
Cutwidth, 29, 33, 65, 230, 231
Cycle, 5, 34
Cyclic shifter, 80, 81, 164

Data structures, v, 5, 6, 7, 8, 27, 111, 185, 205
De Bruijn graph, 10, 31, 35, 44, 55, 191, 200, 222, 224, 230, 231, 241
Decomposition tree, 14, 17, 86, 87, 88
  edge imbalance, 19
  fully balanced, 18
  node imbalance, 18
Degree, 3, 9, 32
Density (geometric), 124
Density function (geometric), 124
Density function (probabilistic), 125, 127
α-Density graph, 124
Dependent edges, 3
Diameter of a graph, 4
Digraph, 3, 131
Dilation of a graph embedding, 28, 31, 33, 34, 39, 53, 54, 65, 229, 231
d-Dimensional ball, 123
d-Dimensional geometry, 122, 159
  $(d-1)$-dimensional sphere, 123
  d-dimensional ball, 123
  centerpoint, 124
  exterior of a sphere, 123
  halfspace, 124
d-Dimensional geometry (*cont.*)
  hyperplane, 123
  integral notation, 125
  interior of a sphere, 123
  norm, 123
  oriented hyperplane, 123
  proper embedding, 122
  random embedding, 122
  sphere, 123
  surface area in d dimensions, 123
  volume in d dimensions, 123
$\mathcal{M}_n(d)$: d-Dimensional side-n mesh, 6
$(d-1)$-Dimensional sphere, 123
Directed acyclic graph, 93
Directed graph, 3
Distance function, 140, 141
Dual graph, 111
Duality, 140

Eccentricity, 124
Edge, 3
Edge imbalance, 19
Edge occurrence, 4
Edge separation, 140
Edge separator, 13, 66, 101, 131, 138, 149
$\mathsf{E}(\mathcal{G})$: Edge-set of graph $\mathcal{G}$, 3
Edge-congestion, 29, 32
Edge-weighted mincing packing function, 185
Eigenvalue, 100, 159
Enabling pebbles, 93
Euler's formula, 110, 113, 118
Execution pebbles, 93
Expander graph, 100, 159, 164
Expansion of a graph embedding, 29, 32
Expansion property, 13, 139
Exposure function, 13
Exterior of a sphere, 123

Face of an embedding, 110
Fault tolerance, 82
Feasible flow, 138
FFT graph, 12, 37, 39, 191, 207, 222, 224, 235
Fiduccia–Mattheyses (FM), 148, 154, 156
Flow conservation, 131, 138
Full separation, 15
Fully balanced, 18

Gamma function, 123
Genetic algorithms, 158
Genus of a surface, 110

Genus-g graph, 117, 158
Genus-g separator theorem, 117
Geometric separator, 100, 122, 159
Graph, 3
Graph area, 69, 79, 234, 235, 236, 237
Graph bisection, 133
Graph bisector, 131
Graph boundary, 12, 13
Graph embedding
  combinatorial, 28, 33, 45, 53, 68, 69, 83, 84, 229
  congestion, 53, 191, 230
  dilation, 28, 31, 33, 34, 39, 53, 54, 65, 229, 231
  edge-routing function, 28
  expansion, 29, 32
  guest graph, 28, 54, 229
  host graph, 28, 229
  node-assignment function, 28
  node-congestion, 29, 32
  source graph, 28, 191
  target graph, 28, 191, 194
  topological, 110, 111
Graph layout, 68
Graph separator, 12
Graph spectra, 100
Graphs as computational models, v, 47, 48, 49, 229
Greedy heuristics, 149, 159

Halfspace, 124
$\mathcal{T}_h(b)$: Height-h complete b-ary tree, 8
$\mathcal{X}_h$: Height-h X-tree, 9
Helly's theorem, 124
Hereditary β-separator of size $S(n)$, 24
Heuristics, 100, 148, 156
Hölder's inequality for integrals, 128
Host graph, 54
Hypercube, 7, 44, 165, 169, 191, 199, 231
  base-b n-dimensional, 7, 199
  boolean, 7, 31, 32, 55, 169, 195, 224, 230, 231, 232, 234, 241
  ternary, 172
Hypergraph, 82, 83, 148, 154
  as a model of buses, 82
Hyperplane, 123

I/O separation, 15, 191, 207
Imbalance, 18, 19
Incident, 3
Independent edges, 3, 171, 173
Induced subgraph, 4
Information-transfer argument, 220
Inside of a cycle,113
Integral notation, 125
Z_n: Integers modulo n, 3
Interior of a sphere, 123
Interval hypergraph, 82, 83, 237
  strongly universal, 82, 83, 84, 237
c-Isometric, 33
Isoperimetric inequality, 13
Isoperimetric number, 159

Jordan Curve Theorem, 111

Kernighan–Lin (KL), 52, 148, 154, 156

Laplacian, 100
Leaf node in a tree, 9
Length of a path, 4
Length of a string, 3
$\mathcal{C}_n$: Length-n cycle, 5
$\mathcal{P}_n$: Length-n path, 5
Level edge in a CCC graph, 12
Level in a butterflylike graph, 11, 12
Level in a tree
  edge, 214
  node, 9
Linear programming, 138
  duality, 140
Lower bounds, 13, 161
$\mathcal{MC}_{m,n}$: $m \times n$ mesh-of-cliques, 7
$\mathcal{M}_{m,n}$: m × n rectangular mesh, 6
$\mathcal{Q}_n(b)$: Base-b n-dimensional hypercube, 7

Manifold, 110
Max-flow, 132
Max-flow/min-cut theorem, 100, 132
Maxdegree, 3
MAXIMUM 2-SATISFIABILITY (MAX 2SAT), 101
MAXIMUM BISECTION WIDTH (MaxBW), 104
Mesh, 6, 31, 33, 35, 68, 87, 165, 175, 176, 191, 197, 231
  2-dimensional, 179, 231, 232, 233
  d-dimensional, 6, 68, 175, 230, 236, 241
  rectangular, 6, 35, 179, 195, 197
  toroidal, 6, 35, 191, 197
Mesh-of-cliques, 7, 191, 196

Min-cut, 132
Mincing a graph, 13, 16, 181
Mincing packing function, 182, 185
  complete binary trees, 182
  edge-weighted complete trees, 187
Mincing Packing Lemma, 182
Mincing-width, 17, 26, 181, 182, 190
  complete binary trees, 185
  edge-weighted complete trees, 189
MINIMUM BISECTION WIDTH (MinBW), 10
Minimum cut, 139
Minimum edge expansion, 139
Minimum quotient separator, 139, 159
Multicommodity flow, 138, 159
  anti-symmetry, 138
  capacity constraint, 138
  cut, 139
  feasible flow, 138
  flow conservation, 138
  minimum cut, 139
Mutual embeddability, 39

$\mathcal{K}_n$: n-Node clique, 5
$\mathcal{K}_{m,n}$: n-Node complete bipartite graph, 5
Neighboring nodes, 3, 164
Network, 131
Network flow, 100, 131, 159
  anti-symmetry, 131
  capacity constraint, 131
  capacity function, 131, 149
  capacity of the cut, 132
  crossing the cut, 132
  cut, 132
  flow conservation, 131
  max-flow, 132
  max-flow/min-cut theorem, 132
  min-cut, 132
  undirected graph as a network, 133
Networks of processors, v, 82
Node, 3
Node imbalance, 18
Node separator, 13, 50, 113, 122
$\mathbf{N}(\mathcal{G})$: Node-set of graph $\mathcal{G}$, 3
Node-congestion, 29, 32
Nonplanar edge, 119
Nonserial dynamic programming, 49
Norm, 123
NP-completeness, 99, 101, 158, 159
NP-hard, 102

λ: Null string, 3

$\mathcal{CCC}_n$: Order-n cube-connected cycles graph, 12
$\mathcal{F}_n$: Order-n FFT graph, 12
$\mathcal{S}_n$: Order-n shuffle-exchange graph, 10
Orientable surface, 110
Oriented hyperplane, 123
Outerplanar graph, 160
Outside of a cycle, 113

Packing argument, 164
Packing function, 165
  2-dimensional meshes, 179
  d-dimensional meshes, 175
  boolean hypercubes, 169
  ternary hypercubes, 172
  X-trees, 167
Packing Lemma, 166
Parallel architectures, v, 7, 8, 10, 11, 82
Parallel edges, 10
Parent node in a tree, 9
Path, 4, 6, 33, 34, 67, 230, 232
Pebble "games", vii, 48, 49, 92, 93, 94, 241
Perfect shuffle graphs, 10
Permutation network, 80, 81, 164, 222
Planar embedding, 111
Planar graph, 24, 49, 109, 111, 158
Planar separator theorem, 112
Position-within-level string, 11
Processor scheduling, 93
Product of graphs, 6
Product-shuffle graph, 209
Proper embedding, 122
Pseudo-code, 100
Pseudorandom sequences, 44
PWL string, 11

Quasi-isometry, 33, 34, 35, 37, 39, 45

Random algorithms, 129
Random bisection, 149
Random embedding, 122
Random graph, 133, 154, 159
Rectangular mesh, 112
Recursive bisector, 16, 25
Recursive edge-bisector, 64, 65, 66, 234
Recursive node-bisector, 230, 231
Register allocation, 93
Regular, 3

Regular graph, 133
Root of a tree, 9
Rooted b-ary tree, 9
Rotation of a graph embedding, 111
Row-edge, 6

3-SATISFIABILITY (3SAT), 101
Self-loops, 10
Separation profile, 84, 86
Separation-width, 13, 15, 26, 33, 45, 49, 66 94, 159, 161, 162, 163, 164, 169, 179, 190, 220, 225, 229, 239, 241
  2-dimensional meshes, 179
  d-dimensional meshes, 175
  boolean hypercubes, 169
  butterfly graphs, 205
  de Bruijn graphs, 202, 224
  hypercubes, 200
  I/O bisections of FFT networks, 209
  mesh-of-cliques, 196
  product-shuffle graphs, 210
  ternary hypercubes, 172
  toroidal meshes, 198
  trees, 207, 213
  X-trees, 167
S^*: Set of finite strings over S, 3
Shuffle (of a string), 10
Shuffle-exchange (of a string), 222
Shuffle-exchange graph, 10, 35
SIMPLE MAX CUT, 158
Simulated annealing, 158, 159
k-Sum subgraph, 17, 89, 182, 185, 188
Spanning subgraph, 4
Spanning tree, 113
Spectrum, 159
Sphere, 110, 123
Straight edge in a butterflylike graph, 11, 12
String, 3
String of Pearls, 20
Subgraph, 4
Surface, 110
Surface area in d dimensions, 123, 159

Taboo search, 158
Ternary tree, 31, 230
Topology, 100, 109, 158
  2-cell embedding, 110
  compact surface, 110
  dual graph, 111
  face of an embedding, 110
  genus of a surface, 110
  manifold, 110
  orientable surface, 110
  rotation of a graph embedding, 111
  sphere, 110
  surface, 110
  triangulated embedding, 112, 117
Total pathlength, 140
Tree, 230
  ternary, 205
Triangulated embedding, 112, 117
Triple-FFT network, 222
Turing machine tape traversal, 187

Undirected graph
  as a network, 133
Uniform multicommodity flow, 138
  UMFP, 138
Upper bounds, 14, 99

VLSI, vi, 221
VLSI layout, v, 48, 68, 82, 234
  area lower bound, 70, 77, 79
  area upper bound, 71

Volume in d dimensions, 123, 159

Weight of a string, 3
Weighted mincing-width, 186

X-tree, 9, 165, 167, 230, 231, 232, 241

This page intentionally left blank