Optimal Emulations by Butterfly-Like Networks

Sandeep N. Bhatt
Bell Communications Research
Morristown, New Jersey

Fan R. K. Chung
Bell Communications Research
Morristown, New Jersey

Jia-Wei Hong
New York University
New York, New York

F. Thomson Leighton
Massachusetts Institute of Technology
Cambridge, Massachusetts

Bojana Obrenić
Queens College of CUNY
Flushing, New York

Arnold L. Rosenberg
University of Massachusetts
Amherst, Massachusetts

Eric J. Schwabe
Northwestern University
Evanston, Illinois

Contact Author:
Eric J. Schwabe
Department of EECS
Northwestern University
2145 Sheridan Road
Evanston, IL 60208

Acknowledgments of Support:
The research of S. N. Bhatt was supported in part by NSF Grant CCR-88-07426, NSF/DARPA Grant CCR-89-08285, and Air Force Grant AFOSR-89-0382; the research of F. T. Leighton was supported in part by Air Force Grant AFOSR-F49620-92-J-0125, DARPA Contracts N00014-91-J-1698 and N00014-92-J-1799, and an NSF Presidential Young Investigator Award with matching funds from AT&T and IBM; the research of B. Obrenić and A. L. Rosenberg was supported in part by NSF Grant CCR-88-12567; the research of E. J. Schwabe was supported in part by DARPA Contract N00014-89-J-1988, NSF Grant CCR-93-09111, and a DARPA/NASA Research Assistantship in Parallel Processing administered by the Institute for Advanced Computer Studies, University of Maryland. A portion of this research was done while F. T. Leighton and A. L. Rosenberg were visiting Bell Communications Research.

Authors' Present Addresses:
Sandeep N. Bhatt: Computing and Communications Research Department, Bell Communications Research, Morristown, NJ 07960;
Fan R. K. Chung: Mathematics, Information Sciences and Operations Research Division, Bell Communications Research, Morristown, NJ 07960;
Jia-Wei Hong: Courant Institute of Mathematical Sciences, New York University, New York, NY 10003;
F. Thomson Leighton: Department of Mathematics and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139;
Bojana Obrenić: Department of Computer Science, Queens College of CUNY, Flushing, NY 11367;
Arnold L. Rosenberg: Department of Computer Science, University of Massachusetts, Amherst, MA 01003;
Eric J. Schwabe: Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures – *interconnection architectures*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems – *Computations on discrete structures*; G.2.1 [**Discrete Mathematics**]: Combinatorics – *combinatorial algorithms*; G.2.2 [**Discrete Mathematics**]: Graph Theory – *graph algorithms*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Mapping algorithms, mapping problems, parallel architectures, processor arrays, embeddings, emulations

ABSTRACT

The power of butterfly-like networks as multicomputer interconnection networks is studied, by considering how efficiently the butterfly can *emulate* other networks. Emulations are studied formally via graph embeddings, so the topic here becomes: How efficiently can one embed the graph underlying a given interconnection network in the graph underlying the butterfly network? Within this framework, the *slowdown* incurred by an emulation is measured by the sum of the *dilation* and the *congestion* of the corresponding embedding (respectively, the maximum amount that the embedding stretches an edge of the guest graph, and the maximum traffic across any edge of the host graph); the *efficiency of resource utilization* in an emulation is measured by the *expansion* of the corresponding embedding (the ratio of the sizes of the host to guest graph).

Three main results expose a number of optimal emulations by butterfly networks. Call a family of graphs *balanced* if complete binary trees can be embedded in the family with simultaneous dilation, congestion, and expansion $O(1)$.

1. The family of butterfly graphs is balanced.
2. (a) Any graph $\mathcal{G}$ from a family of maxdegree-d graphs having a recursive separator of size $S(x)$ can be embedded in any balanced graph family with simultaneous dilation $O(\log(d \sum_i S(2^{-i}|\mathcal{G}|)))$ and expansion $O(1)$.
   (b) Any dilation-D embedding of a maxdegree-d graph in a butterfly graph can be converted to an embedding having simultaneous dilation $O(D)$ and congestion $O(dD)$.
3. Any embedding of a planar graph $\mathcal{G}$ in a butterfly graph must have dilation $\Omega\left(\frac{\log \Sigma(\mathcal{G})}{\Phi(\mathcal{G})}\right)$, where: $\Sigma(\mathcal{G})$ is the size of the smallest (1/3, 2/3)-node-separator of $\mathcal{G}$, and $\Phi(\mathcal{G})$ is the size of $\mathcal{G}$'s largest interior face.

Applications of these results include:

1. The n-node X-tree network can be emulated by the butterfly network with slowdown $O(\log \log n)$ and expansion $O(1)$; no embedding has dilation smaller than $\Omega(\log \log n)$, independent of expansion.
2. Every embedding of the $n \times n$ mesh in the butterfly graph has dilation $\Omega(\log n)$; any expansion-$O(1)$ embedding in the butterfly graph achieves dilation $O(\log n)$.

These applications provide the first examples of networks that can be embedded more efficiently in hypercubes than in butterflies.

We also show that analogues of these results hold for networks that are structurally related to the butterfly network. The upper bounds hold for the hypercube and the de Bruijn networks, possibly with altered constants. The lower bounds hold—at least in weakened form—for the de Bruijn network.

1 Introduction

1.1 An Overview of the Paper

This paper reports on an ongoing program of research, dedicated to comparing the respective computational powers of the various networks that have been proposed for use as multicomputer interconnection networks ([3], [5], [13], [14], [19]). We focus here on one member of the family of *butterfly-like* networks[1] that have become one of the benchmark architectures for processor arrays; our results have direct implications for all butterfly-like networks and nontrivial implications for even more distant relatives of butterfly networks. As is customary, we represent networks as (simple) undirected graphs whose nodes represent the processors at the nodes of the network and whose edges represent the interprocessor communication links. Indeed, a slightly broader perspective would allow the graphs we study to represent also the task-interdependency structures of parallel computations. We compare the powers of networks by determining how efficiently each of the networks can *emulate* the other. Again as is customary, we formalize the notion of emulation in the formal notion of *embedding* one graph in another [24].

A. Embeddings and Emulations

An *embedding* of one simple undirected graph, $\mathcal{G}$, in another, $\mathcal{H}$, comprises a one-to-one *assignment* of the nodes of $\mathcal{G}$ to nodes of $\mathcal{H}$, plus a *routing* of each edge of $\mathcal{G}$ within $\mathcal{H}$, i.e., an assignment of a path in $\mathcal{H}$ connecting the images of the endpoints of each edge of $\mathcal{G}$. We gauge the efficiency of an embedding in terms of its

- *dilation* — the length of the longest path in $\mathcal{H}$ that routes an edge of $\mathcal{G}$
- *congestion* — the maximum number of edges of $\mathcal{G}$ that are routed over any single edge of $\mathcal{H}$
- *expansion* — the ratio[2] $|\mathcal{H}|/|\mathcal{G}|$.

The expansion of an embedding is a first stab at measuring the *efficiency of resource utilization* in an emulation. Less obviously, dilation and congestion combine additively to measure the *slowdown* incurred by an emulation, as the following result indicates.

Proposition 1.1 [20] *Say that one can embed the graph $\mathcal{G}$ in the graph $\mathcal{H}$, with dilation D and congestion C. Then a processor array based on network $\mathcal{H}$ can emulate*

[1] The networks of interest are defined in Section 1.3.

[2] For any graph $\mathcal{G}$, we denote by $|\mathcal{G}|$ the number of nodes in $\mathcal{G}$.

T steps of a processor array based on network $\mathcal{G}$, on a general computation, in $O((C+D)T)$ steps.

B. The Main Results about Butterfly Networks

The major results of this paper establish the following facts about emulations by butterfly networks. Formal statements of the underlying results about graph embeddings, which go well beyond embeddings in butterfly graphs, appear in Section 1.4.

1. Butterfly networks can emulate like-sized complete binary tree networks with only constant factor slowdown.

2. n-node butterfly networks can emulate like-sized networks whose underlying graphs have maximum degree d and recursive separators of size $S(x)$, with slowdown
   $O(d \log(d \sum_i S(n/2^i)))$.

3. Any embedding of any planar network $\mathcal{G}$ in a butterfly network must incur dilation
   $$\Omega\left(\frac{\log \Sigma(\mathcal{G})}{\Phi(\mathcal{G})}\right)$$
   where $\Sigma(\mathcal{G})$ is the size of the smallest (1/3, 2/3)-node-separator of $\mathcal{G}$, and $\Phi(\mathcal{G})$ is the size of $\mathcal{G}$'s largest interior face in a planar embedding.

The latter two results lead to optimal[3] emulations by butterfly networks of a variety of networks, including X-trees and meshes: By Result 2, n-node X-trees can be emulated by $O(n)$-node butterfly networks with slowdown $O(\log \log n)$; and n-node meshes can be emulated by $O(n)$-node butterfly networks with slowdown $O(\log n)$. By Result 3, no embedding can improve on these slowdowns, independent of expansion. The embeddings underlying these results expose X-trees and meshes as the *first known graphs that can be embedded very efficiently in the hypercube* (simultaneous dilation $O(1)$ and expansion $O(1)$; cf. [5], [8] [26]) *but have no efficient embeddings in butterfly-like graphs.* An unexpected corollary of our results is that if we focus only on dilation, then—to within constant factors—these graphs cannot be embedded any more efficiently in butterfly graphs than they can in complete binary trees!

As we note in detail in the next subsection, the techniques of embedding and analysis that we develop here allow us us to derive nontrivial analogues of our butterfly-oriented results for both hypercubes and *shuffle-oriented* networks such as the de Bruijn and shuffle-exchange networks.

[3]Throughout this paper, optimality is measured to within constant factors.

1.2 Placing Our Results in Perspective

Our Notion of Emulation. Our use of graph embeddings as the formal vehicle for studying emulations among interconnection networks means that we are studying a notion of emulation that is at once very *strict*—the host network $\mathcal{H}$ is not allowed to do any "fancy" computations in the course of simulating the guest network $\mathcal{G}$— and very *general*—the host network $\mathcal{H}$ simulates a "generic" computation by the guest network $\mathcal{G}$. One major benefit of this strictness and generality is that it affords one *automatic* translation of any program for the processor array based on network $\mathcal{G}$ to a functionally equivalent program for the processor array based on network $\mathcal{H}$, that suffers no more slowdown than is incurred by the emulation. Less demanding notions of emulation/simulation, that do not guarantee the automatic-translation property, appear in the literature. In [17], the range of computations that the emulating network can perform is broadened. In that extended model, butterfly networks can, indeed, emulate meshes and X-trees efficiently. Another approach to comparing the power of networks involves implementing and analyzing specific algorithms on the networks. This approach is used in [1], to compare the networks we have studied here, and in [12], to compare the hypercube with some of its bounded-degree derivatives.

Our Host Networks. We restrict attention here to embeddings into *butterfly graphs* because focussing on one specific network structure facilitates a rigorous presentation. In fact, our results—both upper and lower bounds—hold (to within constant factors) for a large variety of "leveled" hypercube-derivative host graphs that we collectively term *butterfly-like* and that include the Beneš, cube-connected cycles, and multistage cube networks, as well as all bidelta networks (such as the Omega, flip, baseline, and reverse baseline); cf., [4], [18], [23], [27]. In fact, versions of our results hold for an even broader class of networks; cf. Section 5. By dint of the efficient embeddings of butterfly-like graphs in hypercubes reported in [13], our upper bound results extend verbatim to emulations by hypercubes. In particular: *Any maxdegree-d n-node network whose underlying graph has a recursive separator of size $S(x)$, where $S(x) = O(x^{1-\epsilon})$ for some $\epsilon > 0$, can be emulated by a hypercube with simultaneous slowdown $O(d \log(d \sum_i S(n/2^i)))$ and expansion $O(1)$.* Because shuffle-oriented networks such as the de Bruijn and shuffle-exchange networks "almost" contain like-sized complete binary trees, the algorithms yielding our dilation-optimal embeddings translate verbatim to these networks, though our congestion-optimal ones do not immediately appear to. Section 5 discusses this in greater detail. Because the shuffle-oriented networks "almost" contain moderate-size butterfly-like networks, our lower bounds yield weaker analogues for emulations of a given network by a like-sized shuffle-oriented one; the weakness in the analogues resides both in a restriction on the size of the host graph (which can exceed the size of the guest graph by at most a polynomial

factor) and in the presence of an extra logarithm in the bounds.

Some Technical Remarks.
(**a**) Our lower-bound results provide the first examples of networks that can be embedded more efficiently in hypercubes than in butterflies.
(**b**) The proofs of optimality of certain of our emulations proceed by bounding from below the dilation of the underlying graph embeddings. We believe that nontrivial lower bounds on congestion exist also, at least for embeddings with expansion $O(1)$, but such bounds have as yet eluded us.
(**c**) Our lower-bound results cannot be improved in general, as one can see from considering homeomorphs of the mesh.
(**d**) Our lower bound for embedding the mesh into the butterfly can be extended also to embedding higher-dimensional meshes and to pyramid graphs; thus, these are examples of other popular networks that embed efficiently in the hypercube but not in butterfly-like graphs.

1.3 The Graph Families of Interest

We now define the graphs of interest formally, beginning with the host butterflies and going on with the guest complete binary trees, X-trees, and meshes; see Figs. 1, 2.

- Let m be a positive integer. The *m-level butterfly graph* $\mathcal{B}(m)$ has node-set[4]

$$V_m = \{0, 1, \ldots, m-1\} \times \{0,1\}^m.$$

The subset $V_{m,\ell} = \{\ell\} \times \{0,1\}^m$ of V_m $(0 \leq \ell < m)$ is the ℓth *level* of $\mathcal{B}(m)$. The string $x \in \{0,1\}^m$ of node $\langle \ell,\ x\rangle$ is the *position-within-level string* (*PWL string*, for short) of the node. The edges of $\mathcal{B}(m)$ connect each node

$$\langle \ell,\ \beta_0\beta_1 \cdots \beta_{\ell-1}\beta_\ell\beta_{\ell+1} \cdots \beta_{m-1}\rangle$$

on level ℓ of $\mathcal{B}(m)$ $(0 \leq \ell < m$; each $\beta_i \in \{0,1\})$ with nodes

$$\langle \ell + 1 (\mathrm{mod}\ m),\ \beta_0\beta_1 \cdots \beta_{\ell-1}\gamma\beta_{\ell+1} \cdots \beta_{m-1}\rangle$$

on level $\ell + 1 (\mathrm{mod}\ m)$ of $\mathcal{B}(m)$, for $\gamma \in \{\beta_\ell, \bar{\beta}_\ell\}$. It is convenient to represent $\mathcal{B}(m)$ level by level, in such a way that at each level the PWL strings are the reversals of the binary representations of the integers $0, 1, \ldots, 2^m - 1$, in that order.

[4] $\{0,1\}^m$ denotes the set of length-m binary strings.

- The *height-h complete binary tree* $\mathcal{T}(h)$ has $(2^{h+1}-1)$ nodes, conventionally viewed as the set of binary strings of length at most h; the edges of $\mathcal{T}(h)$ connect each node x of length less than h with nodes $x0$ and $x1$. The (unique) string of length 0 is the *root* of the tree, which is the sole occupant of level 0; the 2^ℓ strings of length ℓ are the *level-ℓ* nodes; the strings of length h (i.e., the level-h nodes) are the *leaves*.

- The *height-h X-tree* $\mathcal{X}(h)$ is obtained from $\mathcal{T}(h)$ by adding edges that connect the nodes at each level of $\mathcal{T}(h)$ in a path whose nodes are in lexicographic order. $\mathcal{X}(h)$ inherits the level structure of $\mathcal{T}(h)$.

- The $s \times s$ *mesh* $\mathcal{M}(s)$ has s^2 nodes, conventionally viewed as the set of ordered pairs of integers $\{\langle i,j\rangle \mid 1 \le i,j \le s\}$; the edges of $\mathcal{M}(s)$ connect nodes $\langle a,b\rangle$ and $\langle c,d\rangle$ just when $|a-c|+|b-d|=1$.

All of these networks have been seriously proposed as interconnection networks for processor arrays ([10], [11], [16]), hence are important candidates for our study.

Our results depend on the following structural features of a graph $\mathcal{G}$:

1. The *maxdegree* of $\mathcal{G}$ is the maximum number of edges touching any one of its nodes.

2. Let α be a real in the range $0 < \alpha \le 1/2$, and let $S(x)$ be a nondecreasing integer function. The graph $\mathcal{G}$ has an $(\alpha, 1-\alpha)$-*node-separator of size* $S(x)$ either if $|\mathcal{G}| < 1/\alpha$ or if there is a set of $S(|\mathcal{G}|)$ nodes whose removal partitions $\mathcal{G}$ into subgraphs, each having $\le (1-\alpha)|\mathcal{G}|$ nodes. We denote by $\Sigma(\mathcal{G})$ the size of the smallest (1/3, 2/3)-node-separator of $\mathcal{G}$. The graph $\mathcal{G}$ has a *recursive-$(\alpha, 1-\alpha)$-node-separator of size* $S(x)$ either if $|\mathcal{G}| < 1/\alpha$ or if $\mathcal{G}$ has a $(\alpha, 1-\alpha)$-node-separator of size $S(x)$ and each of the resulting subgraphs has a recursive $(\alpha, 1-\alpha)$-node-separator of size $S(x)$. We simplify exposition in two ways: When α is either clear from context or irrelevant, we do not mention it; since we use only node-separators, we henceforth omit the qualifier "node".

3. When $\mathcal{G}$ is planar and we are given a witnessing planar embedding ϵ, we denote by $\Phi_\epsilon(\mathcal{G})$ the number of nodes in $\mathcal{G}$'s largest interior face in the embedding. When ϵ is clear from context, we omit the subscript.

1.4 A Formal Statement of the Main Results

We state our main results in terms of embeddings in butterfly graphs, the discussion to this point (especially, Section 1.1B) should allow the reader to translate these

results into their analogues concerning emulations by butterfly networks.

Call a family of graphs *balanced* if complete binary trees can be embedded in the family with simultaneous dilation $O(1)$, congestion $O(1)$, and expansion $O(1)$.

Theorem 1 *The family of butterfly graphs is balanced.*

Obviously, the embeddings that establish Theorem 1 are optimal (cf. footnote 3) in dilation, congestion, and expansion. Building on these embeddings, we derive the following general upper bound result.

Theorem 2 **(a)** *Let Γ be any family of maxdegree-d graphs having a recursive separator of size $S(x)$, where $S(x) = O(x^{1-\epsilon})$ for some $\epsilon > 0$, and let Δ be any balanced graph family. Any graph $\mathcal{G} \in \Gamma$ can be embedded in Δ, with simultaneous dilation $O(\log(d \sum_i S(2^{-i}|\mathcal{G}|)))$ and expansion $O(1)$.*
(b) *Any dilation-D embedding of a maxdegree-d graph in a butterfly graph can be converted to an embedding having simultaneous dilation $O(D)$ and congestion $O(dD)$.*

We complement Theorem 2 with one of the first broadly applicable results for bounding dilation from below.

Theorem 3 *Any nontree connected planar graph $\mathcal{G}$ having a smallest (1/3,2/3)-node-separator of size $\Sigma(\mathcal{G})$, and with no more than $\Phi(\mathcal{G})$ nodes on any interior face, when embedded in any butterfly graph requires dilation*

$$\Omega\left(\frac{\log \Sigma(\mathcal{G})}{\Phi(\mathcal{G})}\right).$$

This bound cannot be improved in general.

Direct application of Theorems 2 and 3 yields the following optimal embeddings.

Corollary 1 *The height-h X-tree $\mathcal{X}(h)$ can be embedded in a butterfly graph with simultaneous dilation and congestion $O(\log h) = O(\log\log|\mathcal{X}(h)|)$ and expansion $O(1)$. Any embedding of $\mathcal{X}(h)$ in a butterfly graph must have dilation $\Omega(\log h) = \Omega(\log\log|\mathcal{X}(h)|)$.*

Corollary 2 *Any embedding of the $s \times s$ mesh $\mathcal{M}(s)$ in a butterfly graph must have dilation $\Omega(\log s) = \Omega(\log|\mathcal{M}(s)|)$.*

Corollary 2 betokens a mismatch in the structures of meshes and butterfly graphs, because *any* expansion-$O(1)$ embedding of *any* graph $\mathcal{G}$ in $\mathcal{B}(m)$ has dilation $O(\log |\mathcal{G}|)$.[5]

Within our strict notion of emulation, Theorem 1 and Corollaries 1 and 2 can be interpreted as yielding tight bounds on the efficiency with which butterfly machines can emulate complete-binary-tree machines, X-tree machines, and mesh-structured machines, with regard to both slowdown and resource utilization.

The next three sections are devoted to proving our main results.

2 Complete Binary Trees—Theorem 1

This section is devoted to proving Theorem 1. Our strategy for embedding complete binary trees in butterfly graphs proceeds by decomposing the guest tree by levels, and then embedding the resulting top, middle, and bottom subtrees separately. We begin our proof with an amusing aside, which introduces some of the ideas, notation, and terminology for our main embedding; additionally, it exposes one of the major obstacles to the embedding.

2.1 An Aside: Finding Many Small Trees in a Butterfly

It is obvious from inspection that one can find an instance of the height-$(m-1)$ complete binary tree $\mathcal{T}(m-1)$ rooted at every node of $\mathcal{B}(m)$. Somewhat less obvious is the fact that one can find m mutually disjoint such instances.

Proposition 2.1 *For every integer m, one can find m mutually disjoint instances of $\mathcal{T}(m-1)$ as subgraphs of $\mathcal{B}(m)$.*

Proof. To simplify exposition, we represent sets of binary strings by strings over the alphabet $\{0, 1, *\}$, using * as a wild-card character. The length-k string

$$\beta = \beta_0 \beta_1 \cdots \beta_{k-1},$$

where each $\beta_i \in \{0, 1, *\}$, represents the set $\sigma(\beta)$ of all length-k binary strings that have a 0 in each position i where $\beta_i = 0$, a 1 in each position i where $\beta_i = 1$, and either a 0 or a 1 in each position i where $\beta_i = *$. For illustration, $\sigma(010) = \{010\}$, and $\sigma(0 * 1) = \{001, 011\}$. Call the string β the *code* for the set $\sigma(\beta)$.

[5] This follows from the facts that $\mathcal{B}(m)$ has $m2^m$ nodes and diameter $O(m)$.

We now embed m instances of $\mathcal{T}(m-1)$ in $\mathcal{B}(m)$, with unit dilation.

We embed the 0th instance of $\mathcal{T}(m-1)$ via the following assignment of tree nodes to butterfly nodes.[6]

$\mathcal{T}(m-1)$	$\mathcal{B}(m)$	
level	level	PWL
0	0	0^m
1	1	$*0^{m-1}$
2	2	$*^2 0^{m-2}$
$\vdots$	$\vdots$	$\vdots$
$m-1$	$m-1$	$*^{m-1}0$

For each *subsequent* instance of $\mathcal{T}(m-1)$, say the jth where $1 \leq j < m$, we assign tree nodes to butterfly nodes as follows.

$\mathcal{T}(m-1)$	$\mathcal{B}(m)$	
level	level	PWL
0	j	$0^{j-1}10^{m-j-1}1$
1	$j+1 (\text{mod } m)$	$0^{j-1}1*0^{m-j-2}1$
2	$j+2 (\text{mod } m)$	$0^{j-1}1*^2 0^{m-j-3}1$
$\vdots$	$\vdots$	$\vdots$
$m-j-1$	$m-1$	$0^{j-1}1*^{m-j-1}1$
$m-j$	0	$0^{j-1}1*^{m-j}$
$m-j+1$	1	$*0^{j-2}1*^{m-j}$
$\vdots$	$\vdots$	$\vdots$
$m-1$	$j-1 (\text{mod } m)$	$*^{j-1}1*^{m-j}$

The placement of the 1s in the PWL strings ensures that the m instances of $\mathcal{T}(m-1)$ are mutually disjoint. To verify this, via contradiction, let us look at an arbitrary level ℓ of $\mathcal{B}(m)$ and at arbitrary distinct tree nodes i and j that collide at some position within level ℓ of $\mathcal{B}(m)$. It is clear that all butterfly nodes that are images of the same instance of $\mathcal{T}(m-1)$ are distinct, so colliding nodes i and j must come from distinct instances of $\mathcal{T}(m-1)$, call them $\iota(i)$ and $\iota(j)$, where the ι-"name" of an instance of $\mathcal{T}(m-1)$ is the level of $\mathcal{B}(m)$ where its root resides. We consider four cases that exhaust the possibilities. In each case, we adduce a property of the PWL strings that precludes any overlap in the images of the trees.

$\iota(i) = 0$:

[6]For any letter a and nonnegative integer k, we denote by a^k a string of k as.

In this case, the PWL string of i ends with $0^{m-\ell}$, while the PWL string of j has a 1 in this range, specifically, in position $m-1$ if $\iota(j) \leq \ell$, and in position $\iota(j)-1$ if $\iota(j) > \ell$.

$1 \leq \iota(i) < \iota(j) \leq \ell$:

In this case, every PWL string of $\iota(i)$ starts with $0^{\iota(i)-1}1$, while every PWL string of $\iota(j)$ starts with $0^{\iota(j)-1}1$.

$1 \leq \iota(i) \leq \ell < \iota(j)$:

In this case, every PWL string of $\iota(i)$ has a 0 in position $\iota(j)-1$, while every PWL string of $\iota(j)$ has a 1 in that position.

$\ell < \iota(i) < \iota(j) < m$:

In this case, every PWL string of $\iota(i)$ has a 1 in position $\iota(i)-1$, while every PWL string of $\iota(j)$ has a 0 in that position.

The proof is complete. □

The proof of Proposition 2.1 does not serve us directly in embedding a large complete binary tree in a small butterfly because it places the roots of the m instances of $\mathcal{T}(m-1)$ at different levels of $\mathcal{B}(m)$; it is not clear how to combine these instances into a big complete binary tree without incurring large dilation. However, the overall strategy of the proof is useful in Section 2.2D.[7]

2.2 Optimally Embedding Trees in Butterfly Graphs

We turn now to the proof of Theorem 1, specifically proving the following.

For any integer m, one can embed the complete binary tree[8] $\mathcal{T}(m + \lfloor \log m \rfloor - 1)$ in the butterfly graph $\mathcal{B}(m+3)$, with dilation $O(1)$.

[7] An algebraic proof of Proposition 2.1, which is "cleaner" than our combinatorial proof here, appears in [3].

[8] All logarithms are to the base 2.

To simplify exposition, let $q =_{\text{def}} m + \lfloor \log m \rfloor - 1$, and assume henceforth that m is a power of 2; clerical changes (such as inserting "floors" into our expressions) will obviate the assumption.

A. The Embedding Strategy

We wish to embed the tree $\mathcal{T}(q)$ with dilation $O(1)$, in the smallest butterfly that is big enough to hold the tree, namely, $\mathcal{B}(m)$. We fall somewhat short of this goal, but not by much: We find an embedding with dilation $O(1)$, but we have to use a somewhat larger host butterfly graph (specifically, $\mathcal{B}(m+3)$) in order to resolve collisions in our embedding procedure. Our embedding proceeds in four stages. Stage 1 embeds the top $\log m$ levels of $\mathcal{T}(q)$ with unit dilation in $\mathcal{B}(m)$, thereby specifying implicitly the images in $\mathcal{B}(m)$ of the roots of the $m/2$ subtrees of $\mathcal{T}(q)$ rooted at level $\log m - 1$. Stage 2 expands these subtrees a further $m/2$ levels, but now in $\mathcal{B}(m+1)$, with dilation 2, thereby specifying implicitly the images in $\mathcal{B}(m+1)$ of the roots of the $m \cdot 2^{m/2-1}$ subtrees of $\mathcal{T}(q)$ rooted at level $m/2 + \log m - 1$ of the tree. In Stage 3, we embed the final $m/2$ levels of $\mathcal{T}(q)$, now in $\mathcal{B}(m+2)$, with dilation 4. The node-assignments in each stage are embeddings (i.e., are one-to-one); there is, however, *overlap* (i.e., distinct nodes of $\mathcal{T}(q)$ getting assigned to the same node of $\mathcal{B}(m+2)$) among the assignments of the three stages. In Stage 4, we eliminate this overlap by the following ploy. We expand the host butterfly graph by one more level, thereby giving us two connected isomorphic copies of $\mathcal{B}(m+2)$. At the cost of increasing dilation by 2, we then modify our assignment so that:

- we perform Stages 1 and 2 in copies of $\mathcal{B}(m+1)$ that reside in distinct copies of $\mathcal{B}(m+2)$;
- we perform Stage 3 in the two copies of $\mathcal{B}(m+1)$ that are *not* used for Stages 1 and 2.

B. Stage 1: The Top $\log m$ Levels of $\mathcal{T}(q)$

We place the root of $\mathcal{T}(q)$ at node

$$\langle m - \log m + 1,\ 0^m \rangle$$

of $\mathcal{B}(m)$. We then proceed to higher-numbered levels, embedding the top $\log m$ levels of $\mathcal{T}(q)$ as a subgraph of $\mathcal{B}(m)$, ending up with the leaves of these levels in nodes

$$\langle 0,\ 0^{m-\log m+1} *^{\log m-1} \rangle$$

of $\mathcal{B}(m)$ (because of wraparound). See Fig. 3. We call the rightmost $\log m - 1$ bits of each of the resulting PWL strings the *signature* of the butterfly node and of the

subtree rooted at that node. It is convenient to interpret a signature as an integer in the range $\{0, 1, \ldots, m/2 - 1\}$, as well as a bit string. The stage is completed by locating $\mathcal{B}(m)$ in those nodes of $\mathcal{B}(m+1)$ that have 0 in the leftmost bit-position of their PWL strings.

The embedding in Stage 1 is trivially one-to-one, with unit dilation.

C. Stage 2: The Next $m/2$ Levels of $\mathcal{T}(q)$

Call the $(m/2+1)$-level subtree of $\mathcal{T}(q)$ that has signature k, the kth *tree*. Our goal is to embed the kth tree in $\mathcal{B}(m+1)$, with dilation 2, so that its $2^{m/2}$ leaves reside in the set of butterfly nodes

$$\langle m,\ 0 * 0 * \cdots * 0 * 1 * 0 * \cdots * 0 * 0 \rangle,$$

where the 1 appears in the $(k+1)$th even-numbered position *from the right* (using 0-based counting); call this the *signatory* 1 of the tree. For instance, when $m = 8$ and $k = 2$ (so that the signature is 10), the leaves reside in nodes

$$\langle 8,\ 0 * 1 * 0 * 0 * 0 \rangle$$

of $\mathcal{B}(9)$.

Note that, since m is even, the nodes of $\mathcal{B}(m+1)$ have PWL strings with $m/2$ odd-numbered positions and $m/2+1$ even-numbered positions; we shall exploit the availability of the "extra" even-numbered position to make this stage of the embedding one-to-one.

We embed the $(m/2+1)$-level trees by alternating unary and binary branchings in $\mathcal{B}(m+1)$, starting at the "roots" placed at level-0 nodes of $\mathcal{B}(m+1)$ during Stage 1; we assign a tree-node after each binary branching. See Fig. 4. Binary branchings generate the *s in the code for the set of PWL strings, while unary branchings generate the 0s and the signatory 1 in the code. For illustration, the embedding of the second tree ($k = 2$) when $m = 8$ proceeds as follows, level by level.

$\langle 0,\ 000000010 \rangle$	embed level 0 (the root)
$\langle 1,\ 000000010 \rangle$	unary branch
$\langle 2,\ 0*0000010 \rangle$	embed level 1
$\langle 3,\ 0*1000010 \rangle$	unary branch
$\langle 4,\ 0*1*00010 \rangle$	embed level 2
$\langle 5,\ 0*1*00010 \rangle$	unary branch
$\langle 6,\ 0*1*0*010 \rangle$	embed level 3
$\langle 7,\ 0*1*0*0*0 \rangle$	unary branch
$\langle 8,\ 0*1*0*0*0 \rangle$	embed level 4

This stage of our embedding clearly incurs dilation 2. The fact that the described assignment is one-to-one (though it may produce conflicts with the embedding of Stage 1) follows from two observations. First, we are using levels 0 through m of $\mathcal{B}(m+1)$ for the $m+1$ levels of this stage, so the leaves of the embedded trees do not wrap around to conflict with their roots. Second, each signatory 1, whose placement identifies its respective tree, is set "on" *before* the signature bits are reached and altered by the sequence of branchings. This is ensured by the fact that we place the signatory 1 by counting from the right: the signature bits occupy the rightmost $\log m - 1$ bits of the PWL string; by the time the branchings have reached the ith bit from the right, only the rightmost $\log i$ bits of the signature are needed to specify the next position where branching occurs. Hence, at the point when we place the signatory 1 in the ith position, the even-numbered positions to the left of the 1 are all 0, and the positions to the right of the 1 form the binary representation of $i-1$, possibly with leading 0s.

We complete Stage 2 by locating $\mathcal{B}(m+1)$ in those nodes of $\mathcal{B}(m+2)$ that have 0 in the rightmost bit-position of their PWL strings.

D. Stage 3: The Final $m/2$ Levels of $\mathcal{T}(q)$

Our goal in Stage 3 is to use the $m \cdot 2^{m/2-1}$ leaves of the $m/2$ trees generated in Stage 2 as the roots of the $(m/2+1)$-level trees comprising the bottom $m/2$ levels of $\mathcal{T}(q)$. Each of these roots is characterized by

- the signatory 1 which identifies the tree it came from in Stage 2
- the *serial number* comprising the odd-numbered bits of the PWL string of its assigned butterfly node.

The signatory 1s will keep trees sired by different Stage-2 trees disjoint; the serial numbers will guard against collisions among trees that were sired by the same Stage-2 tree. The main challenge here is to achieve the embedding while the roots of all the trees reside at the same level of the first copy of $\mathcal{B}(m+1)$ (which is how Stage 2 has placed them). To accomplish this, we have the trees grow *upward*, in the direction of lower level-numbers, for varying amounts of time, before they start to grow *downward*, in the direction of higher level-numbers. While growing either upward or downward, a tree grows via alternating unary and binary branchings, *so as to preserve both the signatory 1 and the serial number;* this alternation will incur dilation 2. An additional dilation of 2 is incurred while a tree grows upward: each tree begins to grow upward using only every fourth level of $\mathcal{B}(m+1)$; when it "turns" from growing upward to growing downward, it reaches level $m-2$ of $\mathcal{B}(m+1)$ by using the levels it has skipped while moving upward, at which time it uses level $m+1$ of the first copy of $\mathcal{B}(m+1)$,

moves to the *second* copy of $\mathcal{B}(m+1)$ and then continues growing downward using every other level of $\mathcal{B}(m+1)$. See Fig. 5.

All trees with the same signatory 1 (i.e., rooted at the leaves of the same Stage-2 tree) will grow in lockstep. We refer to the trees whose roots share a signatory 1 in the $(k+1)$th even bit-position (from the right) as the kth *subtrees* of $\mathcal{T}(q)$, $0 \le k < m/2$, and we call k the *index* of the subtree. In order to simplify our description of this stage, we henceforth ignore the odd-numbered bit-positions of the PWL strings of butterfly nodes. The rightmost of these positions is altered only when the tree moves from the first copy of $\mathcal{B}(m+1)$ into the second; all others contain serial numbers, hence are never affected by our embedding. Thus, the root of each kth subtree of $\mathcal{T}(q)$ is viewed as residing at butterfly node $\langle m,\ 0^{m/2-k-1}10^{k+1}\rangle$. We assign the nodes of the kth subtrees of $\mathcal{T}(q)$ into $\mathcal{B}(m+1)$ as follows. Recalling that Stage 2 assigns the roots (i.e., level-0 nodes) of all subtrees at level m of $\mathcal{B}(m+1)$, we concentrate only on higher-numbered levels of the subtrees. For the kth subtrees, $k \ge 0$:

- for $1 \le \ell \le \lfloor k/2 \rfloor$, we assign the level-$\ell$ nodes at level $m - 4\ell$ of the first copy of $\mathcal{B}(m+1)$;
- *if k is odd*, we assign the level-$\lceil k/2 \rceil$ nodes at level $m - 4\lceil k/2 \rceil + 2$ of the first copy of $\mathcal{B}(m+1)$;
- for $\lceil k/2 \rceil + 1 \le \ell \le k$, we assign the level-$\ell$ nodes at level $m - 4(k-\ell) - 2$ of the first copy of $\mathcal{B}(m+1)$;
- we assign the level-$(k+1)$ nodes at level $m+1$ of the first copy of $\mathcal{B}(m+1)$;
- for $k+1 < \ell \le m/2$, we assign the level-ℓ nodes at level $2(\ell - k - 1) - 1$ of the second copy of $\mathcal{B}(m+1)$.

Now we verify that the described assignment is one-to-one. We consider separately the two potential sources of collisions.

The *serial numbers* obviously prevent collisions among the $2^{m/2}$ kth subtrees, for any k.

The *signatory 1s*, together with the rightmost bit (which distinguishes the first copy of $\mathcal{B}(m+1)$ from the second), prevent collisions among subtrees that share a serial number but that come from trees with different indices. This is seen most easily by considering how such subtrees are laid out level by level. Note first that the top $k+2$ levels of each kth subtree are assigned to butterfly nodes of the form

$$\langle \ell,\ 0^{m/2-k-1}1*^{k+1}\rangle$$

in the first copy of $\mathcal{B}(m+1)$; for each $k+1 < \ell \leq m/2$, the level-ℓ nodes of each kth subtree are placed at nodes

$$\langle 2(\ell-k-1)-1,\ *^{\ell-k-1}0^{m/2-\ell}1*^{k+1}\rangle$$

of the second copy of $\mathcal{B}(m+1)$. Focus, therefore, on a node v_j from level ℓ_j of the jth subtree and a node v_k from level ℓ_k of the kth subtree; with no loss of generality, say $j < k$. The following three cases exhaust the possibilities.

$\ell_j \leq j+1$ and $\ell_k \leq k+1$:

> For levels in the indicated range, nodes in the jth subtree are assigned to butterfly nodes having 0s in the leftmost $m/2-j-1$ even bit-positions of the PWL string; nodes of the kth subtree are assigned to butterfly nodes that have their signatory 1 in this range, specifically, in position $m/2-k-1$.

$[j+1 < \ell_j$ and $\ell_k \leq k+1]$ or $[k+1 < \ell_k$ and $\ell_j \leq j+1]$:
Assume the latter disjunct, the former following by symmetric reasoning.

> In this case, tree node v_j is assigned to a butterfly node that resides in the first copy of $\mathcal{B}(m+1)$, hence has a PWL string ending with 1; tree node v_k is assigned to a butterfly node that resides in the second copy of $\mathcal{B}(m+1)$, hence has a PWL string ending with 0.

$j+1 < \ell_j$ and $k+1 < \ell_k$:

> In this case, both v_j and v_k are assigned to butterfly nodes in the second copy of $\mathcal{B}(m+1)$. Say that they are both assigned to nodes in level ℓ of this copy; we must then have $\ell = 2(\ell_k-k-1)-1 = 2(\ell_j-j-1)-1$, whence $\ell_k-k = \ell_j-j$. Therefore, the PWL string of the butterfly node where v_k resides has its signatory 1 in the $(m/2-k-1)$th even bit-position, while the PWL string of the butterfly node where v_j resides has a 0 in that position, because $\ell_j-j-1 = \ell_k-k-1 \leq m/2-k-1$.

We conclude that the mapping in this stage is an embedding.

E. Resolving Collisions

We now have three subembeddings that accomplish the desired task, except for the fact that different stages may assign different tree nodes to the same butterfly node. We resolve these possible collisions as follows. We perform the subembeddings in $\mathcal{B}(m+3)$ (rather than $\mathcal{B}(m+2)$), placing each subembedding in a distinct copy of the corresponding butterfly: Recall from Subsection A that

- we perform Stages 1 and 2 in copies of $\mathcal{B}(m+1)$ that reside in distinct copies of $\mathcal{B}(m+2)$;
- we perform Stage 3 in the two copies of $\mathcal{B}(m+1)$ that are *not* used for Stages 1 and 2.

We make the transition between the copies as follows. As the Stage-1 embedding of the top of $\mathcal{T}(q)$ reaches level $m+1$ of its copy of $\mathcal{B}(m+1)$ (which we call the Stage-1 copy), we use a sequence of unary branchings in $\mathcal{B}(m+3)$ to reach level 0 of the Stage-2 copy of $\mathcal{B}(m+1)$. The Stage-2 subembedding takes us to level m of that copy, where a sequence of unary branchings in $\mathcal{B}(m+3)$ takes us to level m of the Stage-3 copy of $\mathcal{B}(m+1)$. We start the Stage-3 subembedding in this copy, and complete it in the one yet-unused copy. See Fig. 6. The largest additional dilation is 4; it is engendered by the transition from level m of the Stage-2 copy of $\mathcal{B}(m+1)$ to level m of the Stage-3 copy, thereby yielding the overall dilation 6.

The embedding, hence the proof, is now complete. □

2.3 The Issue of Optimality

Theorem 1 embeds complete binary trees in butterfly graphs, with simultaneous dilation $O(1)$ and expansion $O(1)$. While this achieves our stated goal of optimality to within constant factors, it does leave open the possibility of improving the constant factors. We have been unable to determine exact dilation-expansion tradeoffs for our guest-host matchup, but we can show easily that it is impossible to optimize both cost measures simultaneously. Thus, one cannot hope for the level of "perfection" found in, say, [13] (wherein $\mathcal{B}(m)$ is embedded in the smallest possible hypercube with optimal dilation $1 + (m \bmod 2)$).

Proposition 2.2 *No embedding of $\mathcal{T}(m + \lfloor \log m \rfloor - 1)$ in $\mathcal{B}(m)$ has unit dilation when m is even.*

Proof. Complete binary trees and butterfly graphs with an even number of levels are *bipartite*: one can color the nodes of either graph red and blue in a way that avoids monochromatic edges. For any butterfly graph $\mathcal{B}(r)$, on the one hand, the numbers of red and blue nodes are within a factor of $1 + O(1/r)$ of being equal; for any complete binary tree, on the other hand, one of the sets has roughly twice as many nodes as the other. It follows that one cannot achieve unit dilation when embedding a complete binary tree in the smallest possible butterfly graph. □

3 Upper Bounds—Theorem 2

This section is devoted to proving Theorem 2. Assume that we are given a maxdegree-d n-node graph $\mathcal{G}$ having a recursive separator of size $S(x)$, where $S(x) = O(x^{1-\epsilon})$ for some $\epsilon > 0$. We embed $\mathcal{G}$ efficiently in an $O(n)$-node butterfly graph in three stages: First, we find a dilation-$O(\log(d\sum_i S(n/2^i)))$ embedding of $\mathcal{G}$ in the complete binary tree $\mathcal{T}(\lg(n))$, where $\lg(n) =_{\text{def}} \lceil \log n \rceil$; perforce, this embedding has expansion ≤ 2. Second, we use the embedding of Theorem 1 to embed $\mathcal{T}(\lg(n))$ in a butterfly, with dilation and expansion $O(1)$. These two stages compose to prove part (a) of Theorem 2. Finally, we respecify the edge-routing of the composite embedding according to a general recipe that converts any dilation-D embedding of a maxdegree-d graph in a butterfly to another embedding of the same graph in the same butterfly, with simultaneous dilation $O(D)$ and congestion $O(dD)$. Since the second of these three stages is covered in detail in Section 2, we concentrate here only on the first and third stages.

3.1 Embedding $\mathcal{G}$ in $\mathcal{T}(\lg(|\mathcal{G}|))$

This subsection is devoted to proving the following.

Proposition 3.1 *One can embed any maxdegree-d n-node graph $\mathcal{G}$ having a recursive separator of size $S(x)$, where $S(x) = O(x^{1-\epsilon})$ for some $\epsilon > 0$, in the complete binary tree $\mathcal{T}(\lg(n))$, with simultaneous dilation $O(\log(d\sum_i S(n/2^i)))$ and expansion $O(1)$.*

Our proof of Proposition 3.1 employs the following refinement of the notion of separator; cf. [7].

Let k be a positive integer, and let $R_k(x)$ be a nondecreasing integer function. The graph $\mathcal{G}$ has a *k-color recursive bisector of size $R_k(x)$* either if $|\mathcal{G}| < 2$ or if the following holds for every way of labeling the nodes of $\mathcal{G}$ (independently) with one of k possible labels: By removing at most $R_k(|\mathcal{G}|)$ nodes from $\mathcal{G}$, one can partition $\mathcal{G}$ into subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$ such that

1. $||\mathcal{G}_1| - |\mathcal{G}_2|| \leq 1$.
2. Letting $|\mathcal{H}|_l$ denote the number of nodes of graph $\mathcal{H}$ having label l: For each label l, $||\mathcal{G}_1|_l - |\mathcal{G}_2|_l| \leq 1$.
3. Each of $\mathcal{G}_1$ and $\mathcal{G}_2$ has a k-color recursive bisector of size $R_k(x)$.

Remark. The reader will note that $R_1(x)$ is simply the recursive bisector of a graph, which we denote by $R(x)$ for convenience. In what follows, the proof of Proposition 3.1 builds on the availability of a "balanced" decomposition tree for the source graph $\mathcal{G}$; it does not exploit in any way the particular mechanism used to produce that tree. For the sake of definiteness, we are using node-separators and their derivative colored recursive 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 [7], is a purely clerical procedure.

Using techniques from Section 4 of [7], the reader can easily prove the following crucial technical lemma, which states that k-color recursive bisectors need not be very much bigger than "ordinary" separators.

Lemma 3.1 *For any integer k, any graph $\mathcal{G}$ that has a recursive separator of size $S(x)$ has a k-color recursive bisector of size $R_k(x) = k\sum_i S(x/2^i)$. For any graph, then, $R_k(x) = O(kS(x)\log x)$; when $S(x) = x^{\Omega(1)}$, then $R_k(x) = O(kS(x))$.*

Proof of Proposition 3.1. Our embedding uses the following auxiliary structure, which appears (in slightly different form) in [5]. A *bucket tree for $\mathcal{G}$* is a complete binary tree, each of whose level-ℓ nodes has *(bucket) capacity*

$$C(\ell) = c \cdot \log d \cdot R(|\mathcal{G}|/2^\ell)$$

for some fixed constant c to be chosen later (in Lemma 3.2). We embed $\mathcal{G}$ in $\mathcal{T}(\lg(n))$ in two stages: First, we "embed" $\mathcal{G}$ in a bucket tree, via a many-to-one function that "respects" bucket capacities (always placing precisely $C(\ell)$ nodes of $\mathcal{G}$ in each level-ℓ node of the bucket tree) and has "dilation" $O(\log d)$. Then we "spread" the contents of the bucket tree's buckets within $\mathcal{T}(\lg(n))$, to achieve an embedding of $\mathcal{G}$ in the tree, with the claimed dilation.

In order to avoid confusion, we should point out that throughout the proof of the proposition, expressions of the form "$\log d \cdot R(x)$" should be interpreted as "$(\log d) \cdot R(x)$"; they are written without parentheses to avoid cluttering. When the logarithm is to be applied to the product of more than one term, this will be denoted explicitly, as in "$\log(dR(x))$".

Formally, the first stage of the embedding is described as follows.

A. "Embedding" $\mathcal{G}$ in a Bucket Tree

Lemma 3.2 *The graph $\mathcal{G}$ can be mapped onto 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 $O(\log d)$ apart in the bucket tree.*

Proof. The basic idea is to bisect $\mathcal{G}$ recursively, using a $O(\log d)$-color recursive bisector, placing successively smaller sets of recursive bisector nodes in lower-level buckets of the bucket tree. We also place other nodes in the buckets, in order to ensure that all buckets are filled to capacity. We use the colors to control the "dilation" of the embedding.

We establish the following notation.

- We denote by B_λ the bucket residing at the root of the bucket tree; λ denotes the *null string* (i.e., the string of length 0) over the alphabet $\{0, 1\}$.
- Inductively, for each $x \in \{0,1\}^\star$,[9] we denote by B_{x0} and B_{x1} the buckets residing at the children of the node of the bucket tree where bucket B_x resides; for example, B_0 and B_1 denote the buckets residing at the children of the root node of the bucket tree, B_{00} and B_{01} denote the buckets residing at the left grandchildren of the root node, B_{10} and B_{11} denote the buckets residing at the right grandchildren of the root node, and so on.
- For integers a and b, define

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

Algorithm Bucket: Mapping $\mathcal{G}$ into a bucket tree. The value of the parameter t will be established 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 i. $(i = 1, 2, \ldots, t)$ {Initial t bisections.}
For each subgraph $\mathcal{G}_y$ $(y \in \{0,1\}^{i-1})$ of $\mathcal{G}$ created in Step $i-1$:

[9] $\{0,1\}^\star$ denotes the set of all finite strings over the alphabet $\{0, 1\}$.

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

i**.2.** Place the removed bisector nodes in bucket B_y of the bucket tree.

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

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

Step j**.** $(j = t+1, t+2, \ldots, \lg n)$ {All remaining bisections.}
For each subgraph $\mathcal{G}_y$ $(y \in \{0,1\}^{j-1})$ of $\mathcal{G}$ created in Step $j-1$:

j**.0.** Place every node of color j (MOD t) in bucket B_y.

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

j**.2.** Place the removed bisector nodes in bucket B_y of the bucket tree.

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

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

We claim that the described assignment satisfies both the bucket-capacity condition (a) and the "dilation" condition (b) of Lemma 3.2. The latter condition is transparently enforced when certain colored nodes are automatically placed in buckets (in Step j.**0**), as long as we ultimately choose $t = O(\log d)$. 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 extra nodes to each bucket "in order to fill the bucket to capacity" (specifically, Steps i.**3** and j.**3**). To the end of the analysis, let $N(k)$ denote the number of nodes of $\mathcal{G}$ that get assigned to a bucket at level $k-1$ of the bucket tree. We claim that $N(k)$ obeys the following recurrence.

$$N(k) \leq \frac{d}{2^t} N(k-t) + O\left(tR\left(\frac{n}{2^k}\right)\right)$$

with initial conditions

$$\begin{array}{rcl} N(1) & \leq & R(n) \\ N(2) & \leq & 2R(n/2) \\ N(3) & \leq & 3R(n/4) \\ & \cdots & \\ N(t) & \leq & tR(n/2^{t-1}) \end{array}$$

The recurrence is justified as follows.

- The initial conditions reflect the sizes of the appropriately colored recursive bisectors of $\mathcal{G}$: At each step $i \in \{1, 2, \ldots, t\}$, one uses an i-colored recursive bisector, followed by a $(t+1)$-color recursive bisector at all subsequent steps.

- At levels $\ell > t-1$, the buckets contain not only bisector nodes, which account for the term $O(tR(n/2^k))$ in the general recurrence; they contain also the nodes of $\mathcal{G}$ that are placed in the bucket to satisfy the "dilation" requirements. The latter nodes comprise all neighbors of the $N(k-t)$ occupants of the distance-t ancestor bucket that have not yet been placed in any other bucket. Since nodes of $\mathcal{G}$ can have no more than d neighbors, and since our $(t+1)$-color bisections allocate these neighbors equally among the descendants of a given bucket, these "dilation"-generated nodes can be no more than

  $$\left\lceil \frac{d}{2^t} N(k-t) \right\rceil \leq \frac{d}{2^t} N(k-t) + 1$$

  in number.

Thus, the recursive bisectors and their neighbors account for the occupants of the buckets and for the recurrence counting them.

As long as

$$\frac{d}{2^t} R\left(\frac{n}{2^{k-t}}\right) < (1-\delta) R\left(\frac{n}{2^k}\right)$$

at each step of the recurrence for some constant $\delta > 0$, the $O(tR(n/2^k))$ term will dominate. Since $S(x) = O(x^{1-\epsilon})$ for some $\epsilon > 0$, it is sufficient to choose $t > \frac{\log d}{\epsilon}$. Therefore by choosing an appropriate $t = O(\log d)$, we both satisfy the "dilation" condition and have

$$N(k) = C(k-1) = O\left(\log d \cdot R\left(\frac{n}{2^k}\right)\right)$$

for all levels k of the bucket tree.

□-Lemma 3.2

B. Emptying the Buckets into $\mathcal{T}(\lg(|\mathcal{G}|))$

Our final task is to refine the "dilation"-$O(\log d)$ assignment of Lemma 3.2 to a bona fide embedding of $\mathcal{G}$ in $\mathcal{T}(\lg(n))$, with simultaneous dilation $O(\log(dR(n)))$ and expansion $O(1)$. We proceed inductively, emptying buckets into $\mathcal{T}(\lg(n))$ in such a way that each tree node is assigned a unique node of $\mathcal{G}$. Let $c^\star$ be a constant to be specified later. For each $x \in \{0,1\}^\star$, let $\mathcal{T}_x$ be the complete binary tree of height $c^\star \cdot \log(dR(n))$ rooted at node x of $\mathcal{T}(\lg(n))$. We strive to place the contents of the buckets so that all nodes in each bucket B_x get placed within tree $\mathcal{T}_x$. See Fig. 7.

- Place the $O(\log d \cdot R(n))$ elements of bucket B_λ, in any order, but as densely as possible, in the topmost levels of $\mathcal{T}(\lg(n))$; easily, $h_n =_{\text{def}} \lceil \log(\log d \cdot R(n)) \rceil + c_0$ levels suffice, for some constant c_0. Let all of our trees $\mathcal{T}_x$ start with h_n levels; this is our first step in determining the constant $c^\star$. If the bucket elements fill only m nodes of level h_n 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 $2^{h_n - 1}, 2^{h_n - 1} + 1, \ldots, 2^{h_n - 1} + \lfloor m/2 \rfloor - 1$ of the level. This redistribution of nodes assigned to level h_n is an instance of a procedure we term *evening out* the unloaded bucket (described more fully imminently).

- Because we evened out bucket B_λ, there are unoccupied nodes at level $h_n - 1$ of both $\mathcal{T}_0$ and $\mathcal{T}_1$. Place the contents of bucket B_0 in $\mathcal{T}_0$, 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 3.2), 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 $h_n + c_1$.

We continue to empty buckets, level by level, into $\mathcal{T}(\lg(n))$ 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 $\mathcal{T}(\lg(n))$ in order to empty the next level of buckets. At this point the heights of the subtrees $\mathcal{T}_x$ need never be increased further. Because these heights have been increased by

constants only constantly many times, the constant $c^\star$ defined earlier is assured to exist.

The general procedure for *evening out* a bucket, say, B_x, proceeds as follows:

- 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. Split 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 split 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 $\mathcal{G}$ in $\mathcal{T}(\lg(n))$, since each node of $\mathcal{G}$ is assigned to a unique tree node.
2. The embedding has expansion $O(1)$. To wit, $\mathcal{T}(\lg(n))$ 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 occupancy).
3. The embedding has the desired dilation, namely, $O(\log(dR(n)))$. This follows from our procedure's method of spreading bucket contents throughout $\mathcal{T}(\lg(n))$. Specifically:
   - Each of the subtrees $\mathcal{T}_x$ has height $O(\log(\log d \cdot R(n)))$: $\mathcal{T}_\lambda$ starts with such a height. Subsequent $\mathcal{T}_x$ with short index strings x may have slightly larger height, but only by an additive constant.
   - All subtrees $\mathcal{T}_x$ whose index strings exceed some fixed constant in length have the same height, because the roots of such trees descend in $\mathcal{T}(\lg(n))$ at the same rate (or faster) than the levels of $\mathcal{T}(\lg(n))$ 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 $\mathcal{T}(\lg(n))$, of the set comprising the contents of any bucket plus the nodes in buckets at most $O(\log d)$ buckets up (which lie in adjacent levels $k, k+1, \ldots, k+O(\log d)$ of the bucket tree) are always within a subtree of height $O(\log d + \log(\log d \cdot R(n)))$ of $\mathcal{T}(\lg(n))$.

In other words: Consider the path in $\mathcal{T}(\lg(n))$ between a node v residing in bucket y and the root of the subtree for the bucket $O(\log d)$ levels above y. All but possibly a constant number of the $O(\log d)$ subtrees corresponding to buckets encountered on the way from y to its $O(\log d)$th ancestor have the same height, so that they each contribute at most a single edge to the path. The subtrees for the remaining buckets between y and its $O(\log d)$th ancestor are each of height at most $O(\log(dR(n)))$, so that their total contribution to the path length is at most $O(\log(dR(n)))$. The desired $O(\log d + \log(\log d \cdot R(n))) = O(\log(dR(n)))$ bound follows.

Thus, we have produced the desired embedding, thereby establishing the proposition. □-Proposition 3.1

3.2 Controlling Congestion in Butterfly Embeddings

In order to complete the proof of Theorem 2, we must alter the embedding of the previous subsection, so that it has congestion $O(d \log(dR(n)))$, rather than just $O(2^d dR(n))$. We accomplish this via a general result that allows us to control the congestion of any dilation-D embedding in a butterfly graph.

Proposition 3.2 *Any dilation-D embedding of a maxdegree-d graph $\mathcal{G}$ in a butterfly graph $\mathcal{B}(m)$ can be converted to an embedding of $\mathcal{G}$ in $\mathcal{B}(m)$ having simultaneous dilation $O(D)$ and congestion $O(dD)$.*

The proof employs two auxiliary notions, edge-coloring of graphs and permutation networks.

The following result is traditionally referred to as "Vizing's Theorem".

Lemma 3.3 [28] *One can color the edges of any maxdegree-d graph $\mathcal{H}$ with either d or $d+1$ colors, in such a way that no pair of like-colored edges share an endpoint.*

The importance of Lemma 3.3 is that if one interprets each edge of $\mathcal{H}$ as a transposition of the edge's endpoints, then one can view the process of edge-coloring $\mathcal{H}$ as partitioning $\mathcal{H}$'s edges into at most $d+1$ (partial) permutations of $\mathcal{H}$'s node-set.

The *2^m-input Beneš network* $\hat{\mathcal{B}}(m)$ is a $2m$-level graph, with 2^m nodes at each level. The induced graphs of $\hat{\mathcal{B}}(m)$ on levels $0, 1, \ldots, m$ and on levels $m-1, m, \ldots, 2m-1$ are both isomorphic to any connected component of the induced graph on any consecutive $m+1$ levels of the butterfly graph $\mathcal{B}(m+k)$, $k > 1$. The nodes on level 0 of $\hat{\mathcal{B}}(m)$ are called the network's *inputs*, and the nodes on level $2m-1$ are called the network's *outputs*. The induced subgraph on levels $0, 1, \ldots, m$ of $\hat{\mathcal{B}}(m)$ is (for historical reasons) called the *2^m-input FFT graph* $\mathcal{F}(m)$.

Lemma 3.4 [4] *Let π be any permutation of the integers $\{0, 1, \ldots, 2^m - 1\}$. There is a set of 2^m node-disjoint (hence, congestion-free) paths in the network $\hat{\mathcal{B}}(m)$ that connect each input node i to output node $\pi(i)$.*

Proof of Proposition 3.2. We begin by partitioning $\mathcal{B}(m)$ into $\lfloor m/D \rfloor$ *bands*, each comprising at least D consecutive levels of $\mathcal{B}(m)$. The first $\lfloor m/D \rfloor - 1$ bands comprise exactly D consecutive levels each: Band 0 comprises levels $\{0, 1, \ldots, D-1\}$, band 1 comprises levels $\{D, D+1, \ldots, 2D-1\}$, and so on. The final band comprises the $m - (\lfloor m/D \rfloor - 1)D < 2D$ remaining levels, $\{(\lfloor m/D \rfloor - 1)D, \ldots, m-1\}$. The assumed dilation of the given embedding of $\mathcal{G}$ in $\mathcal{B}(m)$ guarantees the following.

Fact 3.1 *In the given embedding of $\mathcal{G}$ in $\mathcal{B}(m)$, each edge of $\mathcal{G}$ is routed either within one band of $\mathcal{B}(m)$ or within two consecutive bands.*

Motivated by Fact 3.1, we note two further Facts, which the reader can easily verify.

Fact 3.2 *The induced subgraph on any k consecutive levels of $\mathcal{B}(m)$ is isomorphic to 2^{m-k} node-disjoint copies of the 2^k-input FFT graph $\mathcal{F}(k)$.*

One can view the following Fact as an embedding problem in which node-assignments need not be one-to-one.

Fact 3.3 *Let π be any permutation of the integers $\{0, 1, \ldots, 2^m - 1\}$. By mimicking the 2^m-input Beneš network $\hat{\mathcal{B}}(m)$, one can find a set of 2^m paths in the 2^m-input FFT network $\mathcal{F}(m)$ that,* with congestion 2*, connect each input node i to input node $\pi(i)$.*

Next, let us edge-color $\mathcal{G}$ using Vizing's Theorem, thereby partitioning the edges of $\mathcal{G}$ into at most $d+1$ (partial) permutations of $\mathcal{G}$'s node-set.

We are now prepared to reroute the edges of $\mathcal{G}$ in the given embedding. We reroute the edges of each respective color in turn, so let us concentrate on the edges of one particular (but arbitrary) color, c.

We process edges according to the band in which the embedding has assigned their endpoints. Let us focus on the set E_i of edges of $\mathcal{G}$ that have one endpoint assigned (by the embedding) to Band i and the other endpoint assigned either to Band i or to Band $i+1 (\text{mod } m)$; let $e \in E_i$ be a color-c edge whose endpoints are assigned (by the embedding) to nodes $\langle \ell,\ x\rangle$ and $\langle \ell',\ x'\rangle$ of $\mathcal{B}(m)$. We route edge e via the following path. We begin at node $\langle \ell,\ x\rangle$ and traverse $\ell - iD$ straight-edges[10] to node $\langle iD,\ x\rangle$, thereby "positioning" the edge to be routed. We then use Fact 3.3 to find a (congestion-2) path from node $\langle iD,\ x\rangle$ to node $\langle iD,\ x'\rangle$. We complete the routing by traversing $\ell' - iD$ straight-edges to node $\langle \ell',\ x'\rangle$.

We claim that the congestion of the entire color-c routing is $O(D)$. Note first that the traversing of straight-edges to "position nodes to be routed" incurs congestion at most $3D$. This is because each node that passes through node $\langle \ell,\ x\rangle$ in Band i to be "positioned" originates at a unique node $\langle \ell',\ x\rangle$ in either Band i or Band $i+1 (\text{mod } m)$ (note the same PWL string). Next, note that the permutation routing for each row of $< 2^{3D}$ "positioned" endpoints of color-c edges is accomplished via congestion-2 paths in each of the simulated FFT graphs; this increases congestion by at most 4 per edge (since there are two FFT graphs per band). Finally, the traversing of straight-edges to "deposit" the routed edges is symmetric with the "positioning".

Because there are at most $d+1$ colors, it follows that the congestion of the entire routing is $O(dD)$. □-Proposition 3.2

The reader will note that the above argument can be viewed as a constructive schedule which pipelines messages corresponding to the different colors through a Beneš network.

This completes the proof of Theorem 2. □

4 Lower Bounds—Theorem 3

We demonstrate the (existential) optimality (to within constant factors) of the embeddings of Section 3, by proving the lower bound of Theorem 3. We then apply the lower bound to the families of X-trees and meshes.

[10] The *straight-edges* of $\mathcal{B}(m)$ are those that change levels but not PWLs.

4.1 The Lower Bound Argument

Assume henceforth that we are given a connected planar graph $\mathcal{G}$, a planar embedding ϵ of $\mathcal{G}$, and a minimum-dilation embedding μ of $\mathcal{G}$ in a butterfly graph $\mathcal{B}(p)$; let μ have dilation δ. The existence of μ implies that $|\mathcal{G}| \leq p2^p$. (It is worth noting that ϵ and μ represent different notions of "embedding"; ϵ denotes an assignment of positions in two-dimensional space to the nodes and edges of $\mathcal{G}$, whereas μ denotes a mapping of the nodes and edges of the graph $\mathcal{G}$ to nodes and paths in another graph — in this case, a butterfly graph.)

Our overall strategy for bounding δ from below plays two facts off against one another. On the one hand, we show in Lemma 4.6 that the embedding μ must cram into a small number of levels of $\mathcal{B}(p)$ a set of nodes of $\mathcal{G}$ that (a) is a (1/3, 2/3)-node-separator of $\mathcal{G}$ (hence, has size at least $\Sigma(\mathcal{G})$) and (b) is "almost connected". This demonstration requires devising a formal notion of "almost connected" that is intimately connected to a property of planar embeddings (Lemmas 4.3, 4.4). On the other hand, we show in Lemma 4.7 that large "almost connected" sets of nodes can be crammed together in an embedding in a butterfly graph only at the cost of large dilation. These two facts combine to yield the desired lower bound on δ.

We begin with a purely technical result that simplifies our first step toward the lower bound. We note that (because we aim only for bounds that hold up to constant factors) we lose no generality by proving the Theorem for a variant of the graph $\mathcal{G}$ for which every edge resides on the boundary of some interior face in a natural extension of the embedding ϵ:

Lemma 4.1 *One can add edges to the connected graph $\mathcal{G}$ within the embedding ϵ, thereby creating a planar embedding ϵ' of a supergraph $\mathcal{G}'$ of $\mathcal{G}$ that has the following properties.*
(a) *Every edge of $\mathcal{G}'$ in the embedding ϵ' resides on the boundary of an interior face;*
(b) $\Phi_{\epsilon'}(\mathcal{G}') = \Theta(\Phi_\epsilon(\mathcal{G}))$*;*
(c) $\Sigma(\mathcal{G}') = \Theta(\Sigma(\mathcal{G}))$*;*
(d) *$\mathcal{G}'$ contains no self-loops or multiple edges;*
(e) *$\mathcal{G}'$ can be embedded in $\mathcal{G}$ with simultaneous dilation and congestion $O(1)$.*

Proof. We will show how to add edges to the exterior face of $\mathcal{G}$ without changing the separator size and face size by more than a constant factor, and with minimal embedding costs. Furthermore, any edges of $\mathcal{G}$ not on the boundary of any interior face will have interior faces constructed adjacent to them in the supergraph $\mathcal{G}'$, and all the added edges will bound interior faces of the supergraph. Also, no self-loops or multiple edges will be introduced by the construction.

Consider the set of edges of $\mathcal{G}$ that do not bound any interior face of $\mathcal{G}$ in the embedding ϵ. We partition this set of edges into paths in $\mathcal{G}$ as follows: Begin at some edge on the boundary of the exterior face of $\mathcal{G}$, and traverse the edges along the boundary. Begin with an empty path, and when we encounter an edge that is not on the boundary of any interior face and has not been visited already in the traversal, we add it to the path currently being constructed. We end the current path (and begin a new, empty path) when we encounter either an edge that has already been visited, or one that is incident to some interior face. Upon completing our traversal of the boundary of the exterior face, every edge that is not on the boundary of any interior face will be in exactly one of the constructed paths. We will consider each such path separately.

We first consider those paths containing more than one edge. Each path of even length (say, $2i$) can be covered by i additional edges, each of which covers two edges in the path and forms a new interior face of size three. Each path of odd length (say, $2i + 1$) can be covered by $i - 1$ additional edges that cover two edges each in the subsequence, and one that covers three edges and forms an interior face of size four. It is clear that this construction will add no self-loops or multiple edges to the graph, since each edge of $\mathcal{G}$ appears at most once.

Now we consider, in the order that they were traversed, those paths that consist of a single edge. For each such path, consider the edge immediately preceding it in the original traversal. This edge must either be on the boundary of some interior face, or belong to a path created by an earlier traversal of the edge. In either case, this appearance of the edge in the traversal has not been covered, so we may cover these two adjacent edges with a new edge without creating any self-loops or multiple edges. This will create a new interior face of size three. We may repeat this procedure for each path consisting of a single edge, since each edge appears in a path at most once and adjacent edges in the traversal cannot both be single-edge paths.

Clearly, after this is done for every path of edges constructed by the traversal, every edge of $\mathcal{G}'$ is on the boundary of some interior face. No new interior face of size greater than four is created, so it is the case that $\Phi_{\epsilon'}(\mathcal{G}') = \max(4, \Phi_\epsilon(\mathcal{G}))$, which satisfies the desired condition that $\Phi_{\epsilon'}(\mathcal{G}') = \Theta(\Phi_\epsilon(\mathcal{G}))$. Also, since no edge of $\mathcal{G}'$ is in more than two new faces, the separator size does not grow by more than a constant factor (clearly, it cannot decrease since $\mathcal{G}'$ is a supergraph of $\mathcal{G}$); it follows that $\Sigma(\mathcal{G}') = \Theta(\Sigma(\mathcal{G}))$. Finally, by mapping each new edge of $\mathcal{G}'$ to the edges in $\mathcal{G}$ that it covers, we see that $\mathcal{G}'$ can be embedded in $\mathcal{G}$ with constant dilation and congestion. Thus the described construction satisfies the requirements of the Lemma. □

In light of Lemma 4.1, we may assume for the remainder of the section that every

edge of the graph $\mathcal{G}$ in question resides on the boundary of some interior face. More formally, the following lemma establishes that in order to prove Theorem 3 for the graph $\mathcal{G}$, it is sufficient to prove it for the corresponding graph $\mathcal{G}'$:

Lemma 4.2 *For any graph $\mathcal{H}$, if $\mathcal{G}'$ requires dilation $\Omega\left(\frac{\log \Sigma(\mathcal{G}')}{\Phi(\mathcal{G}')}\right)$ to be embedded in $\mathcal{H}$, then $\mathcal{G}$ requires dilation $\Omega\left(\frac{\log \Sigma(\mathcal{G})}{\Phi(\mathcal{G})}\right)$ to be embedded in $\mathcal{H}$.*

Proof. Assume that $\mathcal{G}$ can be embedded in $\mathcal{H}$ with dilation $o\left(\frac{\log \Sigma(\mathcal{G})}{\Phi(\mathcal{G})}\right)$. Since $\mathcal{G}'$ can be embedded in $\mathcal{G}$ with constant dilation, it follows that $\mathcal{G}'$ can also be embedded in $\mathcal{H}$ with dilation $o\left(\frac{\log \Sigma(\mathcal{G})}{\Phi(\mathcal{G})}\right)$. However, since $\Phi(\mathcal{G}') = \Theta(\Phi(\mathcal{G}))$ and $\Sigma(\mathcal{G}') = \Theta(\Sigma(\mathcal{G}))$, we have that $\frac{\log \Sigma(\mathcal{G}')}{\Phi(\mathcal{G}')} = \Theta\left(\frac{\log \Sigma(\mathcal{G})}{\Phi(\mathcal{G})}\right)$. Therefore $\mathcal{G}'$ can be embedded in $\mathcal{H}$ with dilation $o\left(\frac{\log \Sigma(\mathcal{G}')}{\Phi(\mathcal{G}')}\right)$. The statement of the Lemma follows. □

Given Lemma 4.2, we may proceed with the proof of Theorem 3 for the graph $\mathcal{G}'$, knowing that the result for $\mathcal{G}$ will be an immediate consequence. We are now prepared to approach the first step of our lower-bound argument, by showing that $\mathcal{G}'$ contains a (1/3, 2/3)-node-separator that is "almost connected". We begin with some definitions.

- Every planar embedding ϵ of $\mathcal{G}'$ gives rise to a *face-graph* $\Gamma(\mathcal{G}'; \epsilon)$ whose nodes are the interior faces of $\mathcal{G}'$ in the embedding ϵ and whose edges connect a pair of face-nodes just when the faces share a node.
- A set of faces of $\mathcal{G}'$ in the embedding ϵ is *connected* just when the subgraph induced by their corresponding nodes in the face graph $\Gamma(\mathcal{G}'; \epsilon)$ is connected.
- A set S of nodes of $\mathcal{G}'$ is *face-connected* (in ϵ) if the set of *interior faces* of $\mathcal{G}'$ that contain at least one node of S is connected.
- Let A be a connected component of the graph $\mathcal{G}'$ remaining after removing a set S of nodes from $\mathcal{G}'$. The *S-boundary of A* is the set of nodes of A that are adjacent (in $\mathcal{G}'$) to nodes of S.

The next result takes a major step in defining our notion of "almost connected".

Lemma 4.3 *If one removes from the graph $\mathcal{G}'$ a set of nodes S that is face-connected in the embedding ϵ, then the S-boundary of every resulting connected component of $\mathcal{G}'$ is face-connected in the embedding ϵ.*

Proof. Consider a connected component A remaining after removing S from $\mathcal{G}'$. Assume for contradiction that the set of S-boundary nodes of A is not face-connected. There must then be at least two distinct maximal connected components, call them F_1 and F_2, of interior faces that contain boundary nodes (so $F_1 \cup F_2$ is not connected in $\Gamma(\mathcal{G}', \epsilon)$). For $i = 1, 2$, let f_i be an interior face in component F_i, and let b_i be a boundary node in face f_i. Since each edge of $\mathcal{G}'$ lies in an interior face, we can choose each f_i to contain a node of S as well as a boundary node.

Fact 4.1 *There is a set I of interior faces, connected in $\Gamma(\mathcal{G}', \epsilon)$, such that I separates f_1 and f_2, and no face in I contains a boundary node.*

Verification. It is not possible for both F_1 to encircle f_2 and F_2 to encircle f_1, since then F_1 and F_2 would intersect, contradicting the assumption that they are disjoint. Without loss of generality, say that F_1 does not encircle f_2.

Let J be the set of interior faces of $\mathcal{G}'$ (excluding those in F_1) that are incident to the outer boundary of F_1 in $\mathcal{G}'$, so that f_2 is on the outside. (The *outer boundary* of a connected set F of interior faces is a cycle C in $\mathcal{G}'$ such that each edge in C is incident to exactly one face in F, and in the planar embedding of $\mathcal{G}'$, all faces of F are inside the closed curve defined by C. A face is considered incident to this cycle if it meets it at either an edge or a node.)

The set J does not contain any boundary nodes of A, for otherwise F_1 would not be maximal. It is clear that removing J from the face graph would disconnect the vertices representing f_1 and f_2; therefore the set J separates f_1 from f_2 in $\mathcal{G}'$. In addition, J is disjoint from both F_1 and F_2. Finally, J cannot be empty, for otherwise the outer boundary of F_1 would be precisely the edges bounding the exterior face of $\mathcal{G}'$, and thus F_1 would encircle f_2.

If J is connected, then it is the desired set I. If J is not connected, then we argue as follows. The only way J can be disconnected is if the outer boundary of F_1 meets the boundary of the exterior face of $\mathcal{G}'$ in more than one place. In this case, each connected component of J is adjacent to the exterior face (in the face graph) in two places. Therefore, this component forms a path in the face graph that (when the exterior face is ignored) separates the face graph. It follows that each connected component of J separates $\mathcal{G}'$ into disjoint subgraphs. Since F_1 is connected and does not intersect J, F_1 will always be contained in one of these subgraphs. Since each connected component of J separates F_1 from some subgraph of $\mathcal{G}'$, and in its entirety, J separates F_1 from f_2, there must be some connected component of J that separates F_1 from f_2. This component is the desired set I. □-Fact

Fact 4.2 *Every I that satisfies Fact 4.1 contains a node of A and a node of S.*

Verification. I separates f_1 from f_2, yet f_1 and f_2 both contain nodes of both A and S. Since both A and S are face-connected in $\mathcal{G}'$, some face containing a node of A and some face containing a node of S must be in I in order for I to separate f_1 and f_2. □-Fact

Since I contains nodes of both A and S and is face-connected, and since S separates the connected set A from the rest of $\mathcal{G}'$, the set I must contain at least one face that contains *both* a node from A and a node from S. Such a face must also contain a node of the S-boundary of A, contradicting Fact 4.1. The Lemma follows. □-Lemma 4.3

We are now ready to formalize our notion of "almost connected".

A set of nodes S of a graph $\mathcal{K}$ is *d-quasi-connected*, for a positive integer d, if for every two nodes $u, w \in S$, there exists a chain of nodes of S

$$u = v_0, v_1, v_2, \ldots, v_k = w,$$

where consecutive nodes v_i, v_{i+1} are distance at most d apart in $\mathcal{K}$.

The following result shows that Lemma 4.3 gives us an operational handle on quasi-connectivity.

Lemma 4.4 *Any face-connected set of nodes of $\mathcal{G}'$ is $\Phi(\mathcal{G}')$-quasi-connected.*

Proof. Any node in an f-node face is distance at most $f/2$ from any neighboring face. □

Our next goal is to show that $\mathcal{G}'$ must have a large quasi-connected separator. A technical lemma paves the way to this goal.

Lemma 4.5 *Let C be a set of nodes of the graph $\mathcal{G}'$ whose removal partitions $\mathcal{G}'$ into connected components all of size at most $|\mathcal{G}'|/2$. Then C is a $(1/3, 2/3)$-node-separator of $\mathcal{G}'$.*

Proof. Let us remove C from $\mathcal{G}'$, order the resulting connected components in order of decreasing size, and lump the components into two piles as follows.

- Place the largest component into the left pile.
- Alternate piles, adding as few of the largest remaining components as possible to the smaller pile until the smaller pile becomes bigger than the larger pile.

Clearly, when one has completed the two piles, the larger cannot be bigger than the smaller by more than the size of the third largest component, i.e., by more than $|\mathcal{G}'|/3$ nodes. It follows that each pile must contain at least $|\mathcal{G}'|/3$ nodes, whence the result. □

Lemma 4.6 *The embedding μ assigns some $\Phi(\mathcal{G}')$-quasi-connected set of $\Sigma(\mathcal{G}')$ nodes of $\mathcal{G}'$ to some set of $O(\delta)$ contiguous levels of $\mathcal{B}(p)$.*

Proof. In fact, we prove that μ must cram a $\Phi(\mathcal{G}')$-quasi-connected $(1/3, 2/3)$-node-separator of $\mathcal{G}'$, which perforce has size at least $\Sigma(\mathcal{G}')$, into $O(\delta)$ contiguous levels of $\mathcal{B}(p)$.

Our proof begins in a manner reminiscent of the proof of Proposition 3.2, by partitioning $\mathcal{B}(p)$ into *bands*, the ith band being a sequence of $c_i\delta$ consecutive levels, $2\Phi(\mathcal{G}') \leq c_i < 4\Phi(\mathcal{G}')$, where the constants c_i may be chosen in any way that achieves a partition. Let $\kappa(v)$, the *color* of node v of $\mathcal{G}'$, be the index i of the band of $\mathcal{B}(p)$ in which $\mu(v)$ resides.

We perform a modified breadth-first search in $\mathcal{G}'$, to find a $\Phi(\mathcal{G}')$-quasi-connected component of size at least $\Sigma(\mathcal{G}')$, all of whose nodes have images in a single band of $\mathcal{B}(p)$, hence the same color. The breadth-first search proceeds as follows. We select an arbitrary node v_0 of $\mathcal{G}'$ and form V_0, the maximal connected component of $\mathcal{G}'$ that contains v_0 and that consists entirely of nodes with color $\kappa(v_0)$. Removing V_0 partitions $\mathcal{G}'$ into connected components; let C_0 be the largest of these. If $|V_0| \geq \Sigma(\mathcal{G}')$ then V_0 is the desired quasi-connected component of the appropriate size, and we are done. Otherwise, since V_0 is connected, Lemmas 4.3 and 4.4 assure us that the V_0-boundary, B_0, of the component C_0 is $\Phi(\mathcal{G}')$-quasi-connected. It follows that

Fact 4.3 *All nodes of B_0 have the same color.*

Verification. Since each $v \in B_0$ is adjacent to a node of V_0, we must have $\kappa(v) \in \{\kappa(v_0) - 1, \kappa(v_0) + 1\}$. Moreover, B_0 cannot contain nodes of both colors, for two such nodes would be separated by band $\kappa(v_0)$, contradicting the fact that B_0 is $\Phi(\mathcal{G}')$-quasi-connected. □-Fact

Next, form V_1, the maximal *monochromatic*, face-connected subgraph of C_0 that contains B_0; obviously, V_1 is $\Phi(\mathcal{G}')$-quasi-connected. Removing V_1 partitions $\mathcal{G}'$ into some number of connected components. Let C_1 be the largest of these, and let B_1 be the V_1-boundary of C_1. As with B_0, one shows that B_1 is face-connected, $\Phi(\mathcal{G}')$-quasi-connected, and monochromatic.

We continue in this fashion, constructing, in turn, for $i = 2, 3, \ldots$, the following subgraphs of $\mathcal{G}'$, with the indicated properties:

- V_i: the maximal monochromatic, face-connected subgraph of C_{i-1} that contains B_{i-1}.
- C_i: the largest connected component of $\mathcal{G}'$ remaining when one removes V_i from $\mathcal{G}'$
- B_i: the (face-connected, $\Phi(\mathcal{G}')$-quasi-connected, monochromatic) V_i-boundary of C_i

One continues this construction until some subgraph V_i contains at least $\Sigma(\mathcal{G}')$ nodes. We now show that this point must occur.

Fact 4.4 *For some i, $|V_i| \geq \Sigma(\mathcal{G}')$.*

Verification. Note that at each point in our construction, V_i is whittled out of the largest component C_{i-1} of $\mathcal{G}'$ remaining after removal of V_{i-1} from $\mathcal{G}'$. Moreover, V_i disconnects the nodes of $C_{i-1} - V_i$ from the remainder of $\mathcal{G}'$, as one can verify easily by induction on i. At some point, therefore, the whittling process must reduce the size of the then-current largest component C_m so that $|C_m| \leq |\mathcal{G}'|/2$. At this point, we have achieved our goal: by Lemma 4.5, the then-current V_m is a (1/3, 2/3)-node-separator of $\mathcal{G}'$, and hence must contain at least $\Sigma(\mathcal{G}')$ nodes. □-Fact

The preceding development gives us a $\Phi(\mathcal{G}')$-quasi-connected set of nodes of size at most $\Sigma(\mathcal{G}')$, whose image resides in a single band of $c_i\delta$ levels of $\mathcal{B}(p)$, as was claimed. □-Lemma 4.6

As the final step in our proof of Theorem 3, we show that one cannot cram a large quasi-connected set of nodes into a small number of levels of a butterfly graph.

Lemma 4.7 The Fundamental Lemma for Butterfly-Like Graphs. *Say that there is a subgraph $\mathcal{K}$ of $\mathcal{G}'$ and constants c and d such that*

- *$\mathcal{K}$ is d-quasi-connected;*
- *the image of $\mathcal{K}$ under the embedding μ lies within $cd\delta$ consecutive levels of $\mathcal{B}(p)$.*

Then $\delta \geq (\alpha_c/d) \log |\mathcal{K}|$, where α_c is a constant depending only on c, and δ is the dilation of the embedding μ.

Proof. Say that the image of $\mathcal{K}$ under μ lies entirely within levels[11]

$$l+1, l+2, \ldots, l+cd\delta$$

of $\mathcal{B}(p)$. Let u and v be arbitrary nodes of $\mathcal{K}$ that are connected by a path of length $\leq d$ in $\mathcal{G}'$. The image of this path in $\mathcal{B}(p)$ must lie totally within levels

$$l - \lceil d/2 \rceil \delta + 1, \ldots, l + (cd + \lceil d/2 \rceil)\delta$$

of $\mathcal{B}(p)$, because the embedding μ has dilation δ. See Fig. 8. Since $\mathcal{K}$ is d-quasi-connected, this means that the PWL strings of *all* images of nodes of $\mathcal{K}$ can differ only in some set of at most $(cd+d+1)\delta$ bit-positions. It follows that $\mathcal{K}$ can contain no more than $cd\delta 2^{(cd+d+1)\delta}$ nodes: $cd\delta$ levels of $\mathcal{B}(p)$, with at most $2^{(cd+d+1)\delta}$ nodes per level. In other words,

$$cd\delta 2^{(cd+d+1)\delta} \geq |\mathcal{K}|,$$

whence the result. □

Theorem 3 for $\mathcal{G}'$ is now an immediate consequence of Lemmas 4.6 and 4.7, and by Lemma 4.2 it follows for $\mathcal{G}$. □

4.2 Applications to X-Trees and Meshes

Corollaries 1 and 2 now follow from the following Lemmas.

Lemma 4.8 [15] $\Sigma(\mathcal{X}(h)) = \Omega(h) = \Omega(\log|\mathcal{X}(h)|)$, *and* $\Phi(\mathcal{X}(h)) = 4$ *(under the natural embedding).*

Lemma 4.9 (e.g., [15]) $\Sigma(\mathcal{M}(s)) = \Omega(s) = \Omega(\sqrt{|\mathcal{M}(s)|})$, *and* $\Phi(\mathcal{M}(s)) = 4$ *(under the natural embedding).*

5 Extensions to Other Networks

We remarked in Section 1 that all of our results are valid for a large family of butterfly-like networks. In fact, one can prove versions of our main results also for other, non-butterfly-like hypercube-related families of networks. We now present some such extensions, with sketches of proofs.

[11] All addition is modulo p.

5.1 The Hypercube

The nodes of the *m-dimensional (boolean) hypercube* $\mathcal{Q}(m)$ comprise all binary strings of length m; the edges of $\mathcal{Q}(m)$ connect any pair of nodes that differ in precisely one bit-position. Because the butterfly graph is "essentially" a subgraph of the hypercube [13], Theorems 1 and 2 extend intact to the hypercube.

Theorem 4 (a) *The family of hypercubes is balanced.*
(b) *Any maxdegree-d n-node graph having a recursive separator of size $S(x)$, where $S(x) = O(x^{1-\epsilon})$ for some $\epsilon > 0$, can be embedded in a hypercube with simultaneous dilation $O(\log(d\sum_i S(n/2^i)))$, congestion $O(d\log(d\sum_i S(n/2^i)))$, and expansion $O(1)$.*

Proof Sketch. It is proved in [13] that one can embed the butterfly graph $\mathcal{B}(m)$ in the smallest hypercube that is big enough to hold it, namely, $\mathcal{Q}(\lceil m + \log m \rceil)$, with dilation $1 + (m \bmod 2)$; this embedding has simultaneous dilation, congestion, and expansion $O(1)$. Part (a) now follows immediately. By composing the embedding from [13] with the embedding produced in the proof of Theorem 2, one obtains a (constructive) proof of part (b). □

Of course, Theorem 3, and even Corollaries 1 and 2 do not extend to the hypercube. In fact, both meshes and X-trees are embeddable in hypercubes with simultaneous dilation, congestion, and expansion $O(1)$; [9], cf. [5, 21].

Although embeddings of complete binary trees into hypercubes with better constants appear in [6] and [14], we cannot use those embeddings to prove part (b), since at present we do not know of a congestion-control theorem for hypercubes, that would be equivalent to our Proposition 3.2. Actually, it is not known whether a result analogous to our Proposition 3.2 exists for hypercubes even in the special case of embeddings with dilation 3 (cf. [21]). So, our present inability to control congestion in hypercube embeddings leaves us with only one way to utilize the hypercube: embedding indirectly via intermediate butterflies. In place of the butterfly, we could use the *cell tree* of [22], which has sufficient congestion-control properties. In the case of hypercube embeddings, this would change only our constants (improving the expansion and worsening the slowdown); in the case of shuffle-oriented networks, however, this is the only way we know of to obtain optimal embeddings.

5.2 Shuffle-Oriented Networks

The nodes of the *order-m de Bruijn graph* $\mathcal{D}(m)$ comprise all binary strings of length m; the edges of $\mathcal{D}(m)$ connect any pair of nodes of the form $\{\beta x, x\gamma\}$, where $\beta \in$

$\{0,1\}$, $x \in \{0,1\}^{m-1}$, and $\gamma \in \{\beta, \bar{\beta}\}$. We now extend versions of our main results to interconnection networks based on de Bruijn graphs and other such *shuffle-oriented* graphs, including shuffle-exchange graphs [3] and perfect-shuffle graphs [12].

Because $\mathcal{D}(m)$ contains a copy of $\mathcal{T}(m-1)$ as a subgraph, Theorem 1 extends trivially to shuffle-oriented graphs. For the same reason, part (a) of Theorem 2, which refers only to dilation, carries over directly; indeed the proof is even easier for de Bruijn graphs than for butterfly graphs because of the triviality of embedding complete binary trees.

Unfortunately, we do not know of a direct extension to de Bruijn graphs of part (b) of Theorem 2, which refers to congestion. The closest approximation to Proposition 3.2 that we know of is:

> Any dilation-D embedding of a maxdegree-d graph $\mathcal{G}$ in a de Bruijn graph $\mathcal{D}(m)$ can be converted to an embedding of $\mathcal{G}$ in $\mathcal{D}(m)$ having simultaneous dilation $O(D)$ and congestion $O(dD^2)$.

We sketch the proof of this claim: Say that we have embedded the graph $\mathcal{G}$ in $\mathcal{D}(m)$ with dilation $D \leq m/2$. The edge structure of $\mathcal{D}(m)$ guarantees that, if node v of $\mathcal{G}$ is assigned by the embedding to node xyz of $\mathcal{D}(m)$, where $x, z \in \{0,1\}^D$ and $y \in \{0,1\}^{m-2D}$, then every neighbor of v in $\mathcal{G}$ must be assigned by the embedding to some node $x'yz'$ of $\mathcal{D}(m)$, where $x' \in \{0,1\}^c$ for some $c \in \{0,1,\ldots,2D\}$ and $z' \in \{0,1\}^{2D-c}$. We route the path from node xyz to node $x'yz'$ via the path:

$$xyz \longrightarrow yzx \longrightarrow yz'x' \longrightarrow x'yz'$$

The first and third steps consist exclusively of cyclic shifts of node labels, and therefore can be accomplished with dilation $O(D)$ and overall congestion $O(dD)$. For a fixed c, the second step can be accomplished with dilation $O(D)$ and congestion $O(dD)$ [2], yielding overall congestion $O(dD^2)$, as $0 \leq c \leq 2D$.

The analog of Theorem 2 for shuffle-oriented networks appears in [22]. Rather than embedding the bucket tree into the complete binary tree (Proposition 3.1), *cell trees* are instead used. Cell trees are special interconnection networks that have an overall structure of complete binary trees, where the nodes themselves are networks that can route permutations efficiently. Cell trees have intrinsic congestion-control properties, which in this case compensates for the absence of an equivalent to Proposition 3.2. Finally, [22] (q.v.) shows how cell trees can be embedded efficiently into shuffle-oriented networks (and hypercubes).

For shuffle-oriented graphs we can prove a weaker version of the lower bound part of Corollary 1, which allows only embeddings with polynomial expansion (meaning

that we are embedding an n-node guest network into a host network with $O(n^a)$ nodes for some constant a). However, we do achieve the same lower bound as in Corollary 1, which (because of the upper bounds in [22]) is optimal. Unfortunately, our extension of Corollary 2 to shuffle-oriented graphs suffers not only from the just-mentioned restriction to polynomial-expansion embeddings, but also establishes a lower bound that is only the logarithm of the bound of Corollary 2. We count this doubly logarithmic lower bound as a weakness, because we conjecture that the (singly) logarithmic lower bound of Corollary 2 holds also for embeddings of the mesh in shuffle-oriented graphs.

Theorem 5 **(a)** *The family of de Bruijn graphs is balanced.*
(b)(i) *Any maxdegree-d n-node graph having a recursive separator of size $S(x)$, where $S(x) = O(x^{1-\epsilon})$ for some $\epsilon > 0$, can be embedded in a de Bruijn graph with simultaneous dilation $O(\log(d \sum_i S(n/2^i)))$ and expansion $O(1)$.*
(b)(ii) *Any maxdegree-d n-node graph having a recursive separator of size $S(x)$, where $S(x) = O(x^{1-\epsilon})$ for some $\epsilon > 0$, can be embedded in a de Bruijn graph with simultaneous dilation $O(d^2 \cdot \log(d \sum_i S(n/2^i)))$, congestion $O(d)$, and expansion $O(1)$.*

Proof Sketch. We argue only about the de Bruijn graph as a host. The extension to its shuffle-oriented relatives follows from the existence of dilation-$O(1)$ embeddings of the de Bruijn graph into those graphs and of those graphs into the de Bruijn graph. (See, e.g., [25].)

(a), (b)(i): One verifies easily that $\mathcal{D}(m)$ contains a copy of $\mathcal{T}(m-1)$ as a subgraph. Therefore, part (a) is a triviality. Part (b)(i) follows from the embedding in our proof of Proposition 3.1.

(b)(ii): See [22] for a detailed proof. □

Our analogues of Corollaries 1 and 2 for the de Bruijn graph are as follows:

Theorem 6 **(a)** *Any embedding of the height-h X-tree $\mathcal{X}(h)$ in a de Bruijn graph with polynomial expansion must have dilation $\Omega(\log h) = \Omega(\log\log|\mathcal{X}(h)|)$.*
(b) *Any embedding of the $s \times s$ mesh $\mathcal{M}(s)$ in a de Bruijn graph with polynomial expansion must have dilation $\Omega(\log\log s) = \Omega(\log\log|\mathcal{M}(s)|)$.*

Proof: Let us digress for a moment, to prove a lemma that shows that most small "neighborhoods" in the de Bruijn graph look like butterflies.

We observe that each edge of the de Bruijn graph $\mathcal{D}(m)$ can be thought of as being directed from node βx to node $x\gamma$, where $\beta \in \{0,1\}$, $x \in \{0,1\}^{m-1}$, and $\gamma \in \{\beta, \bar{\beta}\}$.

Given this orientation of the edges, we can define a *forward* move in the de Bruijn graph as the traversal of an edge in its given direction. A *backward* move is the traversal of an edge against its given direction.

For each node v of the de Bruijn graph and any non-negative integer k, define $\mathcal{N}_k(v)$ as a subgraph of $\mathcal{D}(m)$ containing precisely those nodes that can be reached from v via a path constructed in the following three stages:

1. First, make k forward moves, followed by k backward moves;
2. Next, make k additional backward moves, followed by k forward moves;
3. Finally, for some $l \leq k$, make either l forward moves or l backward moves.

The set of edges of $\mathcal{N}_k(v)$ consists of precisely those edges that are contained in at least one such path. Let $v = xyz$, where $y \in \{0,1\}^{m-2k}$ and $x, z \in \{0,1\}^k$. Then the first stage can change only the k bits of x, and the second stage can change only the k bits of z; neither stage can change any of the bits of y. Thus the vertex set of $\mathcal{N}_k(v)$ is

$$V_k(v) = \{x'yz' \mid x'z' \in \{0,1\}^{2k}\}.$$

Note that all nodes within distance k of v in the de Bruijn graph are included in $V_k(v)$. We now show that for most nodes v of $\mathcal{D}(m)$, the "neighborhood" graph $\mathcal{N}_k(v)$ is equivalent to the FFT graph $\mathcal{F}(2k)$.

Lemma 5.1 *For all m and for all $k < m/4$, there is a set S containing at most 2^{4k+1} of the nodes of $\mathcal{D}(m)$, with the following property: For every node v of $\mathcal{D}(m)$ that is not in S, the induced subgraph $\mathcal{N}_k(v)$ is isomorphic to $\mathcal{F}(2k)$.*

Proof: As before, let $v = xyz$, where $y \in \{0,1\}^{m-2k}$ and $x, z \in \{0,1\}^k$. It is straightforward to verify that the mapping

$$\langle \ell,\ \beta_0\beta_1\cdots\beta_{2k-1}\rangle \leftrightarrow \beta_\ell\cdots\beta_{2k-1}y_0y_1\cdots y_{m-2k-1}\beta_0\beta_1\cdots\beta_{\ell-1}$$

is an isomorphism between $\mathcal{F}(2k)$ and $\mathcal{N}_k(v)$ as long as the string $y = y_0y_1\cdots y_{m-2k-1}$ occurs only once as a substring of each node label in $V_k(v)$.

Thus we need to show that for only a small number of v is it the case that some string in $V_k(v)$ contains more than one occurrence of the substring y, where y is the

middle $m-2k$ bits of v. To wit, suppose that the string $w \in V_k(v)$ does contain two occurrences of y. By considering the overlap of the two occurences, one verifies that y must consist of repeated instances of some length-i prefix of itself, for some $i \leq 2k$. (Recall that the length of y is at least $m-2k$.) That is, for some i in the indicated range, we must have

$$y_0 y_1 \cdots y_{(m-2k-1)-i} = y_i y_{i+1} \cdots y_{m-2k-1},$$

so that

$$y = y_0 y_1 \cdots y_{i-1} y_0 y_1 \cdots y_{i-1} y_0 y_1 \cdots y_{i-1} \cdots.$$

For a given i, there at most 2^i strings y that satisfy this condition, so the total number of such strings y over the range of possible i is at most $\sum_{i=1}^{2k} 2^i = 2^{2k+1} - 2$. Thus the number of strings v that yield such a substring y is at most $2^{2k}(2^{2k+1} - 2) \leq 2^{4k+1}$. These strings v containing "periodic" substrings y define the set S of the Lemma. □-Lemma 5.1

Return to the Proof of the Theorem. Let us concentrate on part (a). Suppose that we have embedded the X-tree $\mathcal{X}(h)$ in some de Bruijn graph — say, $\mathcal{D}(m)$ — with polynomial expansion and dilation $\delta = o(\log h)$. Let $N = 2^h - 1$ be the number of nodes in the X-tree, and let the number of nodes in the de Bruijn graph be $M = O(N^a)$ for some constant $a \geq 1$. Note that $m = \Theta(h)$. Applying Lemma 5.1 with $k = m/8a$, we see that all but $O(M^{1/2a}) = O(N^{1/2})$ of the M nodes w of the de Bruijn graph have $\mathcal{N}_{m/8a}(w)$ isomorphic to the FFT graph $\mathcal{F}(m/4a)$. (For simplicity, we assume that $m/8a$ is an integer; this assumption can be easily removed.) Therefore one of these $M - o(N)$ de Bruijn graph nodes must have some node v of the X-tree mapped to it.

For such an X-tree node v, all nodes within distance $m/8a\delta$ of v must also be assigned by the embedding to nodes in $\mathcal{N}_{m/8a}(w)$, since $\mathcal{N}_{m/8a}(w)$ includes all de Bruijn graph nodes within distance $m/8a$ of w. Recall that $\mathcal{N}_{m/8a}(w)$ is isomorphic to $\mathcal{F}(m/4a)$. Since v is mapped to node w in the middle level of this FFT graph, all nodes within distance $m/16a\delta$ of v must be assigned to locations in the middle $m/8a + 1$ levels of this FFT graph. The neighborhood of radius $m/16a\delta$ around v must contain a subgraph isomorphic to some X-tree $\mathcal{X}(O(m/\delta))$, and the distance from the assigned locations in $\mathcal{F}(m/4a)$ of any of these nodes to any node outside of the $\mathcal{F}(m/4a)$ subgraph is at least $m/16a$, which is $\Omega(\log h)$. This forces the entire embedding of the neighborhood of v to take place within the specified $\mathcal{F}(m/4a)$. Thus the given embedding of $\mathcal{X}(h)$ in $\mathcal{D}(m)$ must "contain" an embedding of $\mathcal{X}(O(m/\delta))$ in $\mathcal{F}(m/4a)$. By Corollary 1, coupled with the dilation-2 embeddability of butterfly graphs and FFT graphs in each other, this "contained" embedding must have dilation $\Omega(\log(m/\delta)) = \Omega(\log m) = \Omega(\log h)$. Part (a) follows.

Adapting this proof to embeddings of meshes in de Bruijn graphs with polynomial expansion for the proof of part (b) is just a clerical task. □

Conjecture 1 *Any embedding of the $s \times s$ mesh $\mathcal{M}(s)$ in a de Bruijn graph with polynomial expansion must have dilation $\Omega(\log s) = \Omega(\log |\mathcal{M}(s)|)$.*

ACKNOWLEDGMENTS: A preliminary version of this paper was presented at the *20th ACM Symposium on Theory of Computing*, Chicago, Ill., May 2-4, 1988. The authors wish to thank Dave Barrington and Les Valiant for helpful conversations.

References

[1] A. Aggarwal (1984): A comparative study of X-tree, pyramid, and related machines. *25th IEEE Symp. on Foundations of Computer Science*, 89-99.

[2] F.S. Annexstein (1989): Fault tolerance in hypercube-derivative networks. *1st ACM Symp. on Parallel Algorithms and Architectures*, 179-188.

[3] F.S. Annexstein, M. Baumslag, A.L. Rosenberg (1990): Group action graphs and parallel architectures. *SIAM J. Comput. 19*, 544-569.

[4] V.E. Beneš (1964): Optimal rearrangeable multistage connecting networks. *Bell Syst. Tech. J. 43*, 1641-1656.

[5] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, A.L. Rosenberg (1992): Efficient embeddings of trees in hypercubes. *SIAM J. Comput.*, Vol. 21, No. 1. (1992) 151-162. See also, Optimal simulations of tree machines. *27th IEEE Symp. on Foundations of Computer Science* (1986) 274-282.

[6] S.N. Bhatt and I. Ipsen (1985): How to embed trees in hypercubes. Yale University Research Report DCS/RR-443.

[7] S.N. Bhatt and F.T. Leighton (1984): A framework for solving VLSI graph layout problems. *J. Comp. Syst. Sci. 28,* 300-343.

[8] M.Y. Chan (1988): Dilation-2 embedding of grids into hypercubes. *Intl. Conf. on Parallel Processing*, 295-298.

[9] M.Y. Chan (1991): Embedding of grids into optimal hypercubes. *SIAM J. Comput. 20*, 834-864.

[10] A.M. Despain and D.A. Patterson (1978): X-tree – a tree structured multiprocessor architecture. *5th Symp. on Computer Architecture*, 144-151.

[11] D. Gannon (1980): On pipelining a mesh-connected multiprocessor for finite element problems by nested dissection. *Intl. Conf. on Parallel Processing*.

[12] R. Ginosar and D. Egozi (1987): Topological comparison of perfect shuffle and hypercube. Typescript, The Technion.

[13] D.S. Greenberg, L.S. Heath and A.L. Rosenberg (1990): Optimal embeddings of butterfly-like graphs in the hypercube. *Math. Syst. Th. 23*, 61-77.

[14] I. Havel and P. Liebl (1973): Embedding the polytomic tree into the n-cube. *Časopis pro Pěstování Matematiky 98*, 307-314.

[15] J.-W. Hong and A.L. Rosenberg (1982): Graphs that are almost binary trees. *SIAM J. Comput. 11*, 227-242.

[16] E. Horowitz and A. Zorat (1981): The binary tree as an interconnection network: applications to multiprocessor systems and VLSI. *IEEE Trans. Comp., C-30*, 247-253.

[17] R. Koch, F.T. Leighton, B. Maggs, S. Rao, A.L. Rosenberg, E.J. Schwabe (1990): Work-preserving emulations of fixed-connection networks. *J. ACM*, to appear; see also *21st ACM Symp. on Theory of Computing*, 227-240.

[18] C.P. Kruskal and M. Snir (1986): A unified theory of interconnection network structure. *Theor. Comp. Sci. 48*, 75-94.

[19] F.T. Leighton (1984): Parallel computation using meshes of trees. *1983 Workshop on Graph-Theoretic Concepts in Computer Science*, Trauner Verlag, Linz, pp. 200-218.

[20] F.T. Leighton, B. Maggs, S. Rao (1992): Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, to appear.

[21] F.T. Leighton (1992): *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, San Mateo, Calif.

[22] B. Obrenić (1994): An approach to emulating separable graphs. *Math. Syst. Th.* 27, 41-63.

[23] F.P. Preparata and J.E. Vuillemin (1981): The cube-connected cycles: a versatile network for parallel computation. *C. ACM 24*, 300-309.

[24] A.L. Rosenberg (1981): 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, NY, 150-176.

[25] A.L. Rosenberg (1992): Product-shuffle networks: toward reconciling shuffles and butterflies. *Discr. Appl. Math. 37/38*, 456-488.

[26] Y. Saad and M.H. Schultz (1988): Topological properties of hypercubes. *IEEE Trans. Comp. 37*, 867-872.

[27] H.J. Siegel (1985): *Interconnection Networks for Large-Scale Parallel Processing.* D.H. Heath and Co., Lexington, Mass.

[28] V.G. Vizing (1964): On an estimate of the chromatic class of a p-graph (in Russian). *Diskret. Analiz 3*, 25-30.