Cost Trade-offs in Graph Embeddings, with Applications

JIA-WEI HONG

Peking Computer Institute, Peking, China

KURT MEHLHORN

Universität des Saarlandes, Saarbrücken, Federal Republic of Germany

AND

ARNOLD L. ROSENBERG

Duke University, Durham, North Carolina

Abstract. An *embedding* of the graph G in the graph H is a one-to-one association of the vertices of G with the vertices of H. There are two natural measures of the cost of a graph embedding, namely, the *dilation-cost* of the embedding: the maximum distance in H between the images of vertices that are adjacent in G; and the *expansion-cost* of the embedding: the ratio of the size of H to the size of G. The main results of this paper illustrate three situations wherein one of these costs can be minimized only at the expense of a dramatic increase in the other cost. The first result establishes the following: There is an embedding of n-node complete ternary trees in complete binary trees with dilation-cost 2 and expansion cost $\Theta(n^{\lambda})$, where $\lambda = \log_3(4/3)$; but any embedding of these ternary trees in binary trees that has expansion-cost $c < 2$ must have dilation-cost $\Omega(\log\log\log n)$. The second result provides a stronger but less easily stated example of the same type of trade-off. The third result concerns *generic* binary trees, that is, complete binary trees into which all n-node binary trees are "efficiently" embeddable. There is a generic binary tree into which all n-node binary trees are embeddable with dilation-cost $O(1)$ and expansion-cost $O(n^c)$ for some fixed constant c; if one insists on embeddings whose dilation-cost is exactly 1, then these embeddings must have expansion-cost $\Omega(n^{(\log n)/2})$; if one insists on embeddings whose expansion-cost is less than 2, then these embeddings must have dilation cost $\Omega(\log\log\log n)$. An interesting application of the polynomial size generic binary tree in the first part of this three-part result is to yield simplified proofs of several results concerning computational systems with an intrinsic notion of "computation tree," such as alternating and nondeterministic Turing machines and context-free grammars.

Categories and Subject Descriptors: F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*alternation and nondeterminism*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; F.2.3 [**Analysis of Algorithms and Problem Complexity**]: Trade-offs among Complexity Measures; G.2.2 [**Discrete Mathematics**]: Graph Theory

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Graph embeddings, dilation-cost, expansion-cost, cost trade-offs, generic graphs

The research of the first and third authors was done in part while they were visiting the Department of Computer Science, University of Toronto, Toronto, Ontario, Canada. The research of the third author was done in part while he was with the Mathematical Sciences Department of the IBM Research Center, Yorktown Heights, NY, and was supported in part by the National Science Foundation under Grant MCS 81-16522.

Authors' addresses: J.-W. Hong, Peking Computer Institute, Peking 100044, China; K. Mehlhorn, Fach. 10: Angewandte Mathematik und Informatik, Universität Saarlandes, Saarbrücken, Federal Republic of Germany; A. L. Rosenberg, Department of Computer Science, Duke University, Durham, NC 27706.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1983 ACM 0004-5411/83/1000-0709 $00.75

Introduction

An *embedding* of the graph G in the graph H is a one-to-one association of the vertices of G with the vertices of H. Two natural costs of a graph embedding are the *dilation-cost*, which is the maximum distance in H between the images of adjacent vertices of G, and the *expansion-cost*, which is the ratio of the size (i.e., number of vertices) of H to the size of G. Typical of the problems studied in the area of graph embeddings are: to find the best embedding of a given G in a given H; to find the best embedding of a given G in any of a given family of H's; to find the best embeddings of a family of G's in any single member of a given family of H's.

Graph embedding problems arise naturally when one studies a variety of computational situations. Among the "real-life" problem areas that give rise to graph embedding mathematical counterparts are the problems of finding storage representations for data structures [7, 13, 17, 20], of finding efficient layouts of circuits on chips [11, 19, 24, 25], of finding efficient structured versions of programs [13], and of organizing computations on networks of processors [10]. In addition, numerous specific problems can be fruitfully formulated as graph embedding problems [14]. The wide applicability of concepts related to graph embeddings has given rise to a number of recent papers that study basic questions concerning embeddings, independent of any specific motivating situation [5, 9, 15, 20]; and we are beginning to amass a body of basic knowledge about embeddings, which can be applied to many "real-life" problem areas. This paper continues in the footsteps of these last-cited papers. In it we study cost trade-offs in graph embeddings.

Cost trade-offs occur in a number of computational situations. As but four examples: Cobham [6] discovered a time–space trade-off for the problem of palindrome recognition; Borodin et al. [2] found a similar trade-off for the problem of sorting; Thompson [24] discovered an area–time trade-off for the problem of implementing the DFT in VLSI; and Hong [8] uncovered a trade-off involving reversal and space complexity for Turing machines. It is not a priori obvious that cost trade-offs exist in the simple world of graph embeddings; but in fact, such trade-offs can be found quite easily.

Consider the problem of embedding the n-vertex line graph $L(n)$ (whose vertex-set is the set $\{1, 2, \ldots, n\}$ and whose edge-set is the set of all pairs $(i, i + 1)$) in a complete binary tree. One finds the following.

PROPOSITION. *(a) There is an embedding of $L(n)$ in the height-n complete binary tree with dilation-cost* 1 *(but, of course, with expansion-cost $\Omega(2^n/n)$). (b) Any embedding of $L(2^h - 1)$ in the height-$(h - 1)$ complete binary tree (which has expansion-cost* 1*) has dilation-cost at least* 3 [20].

Part (b) of the Proposition is proved by showing that the root of the host tree acts as a bottleneck when one attempts to "snake" the line into the tree: if one attempts to lay out the line with dilation-cost 2, then one gets trapped in a subtree of the tree with no way to continue the layout. Sekanina [23] proves that dilation-cost 3 is always attainable.

Now, the trade-off exposed in the proposition is a simple one and is far from dramatic: one trades an exponential increase in expansion for a factor of 3 in dilation. The basic questions are:

(1) Do there exist cost trade-offs in graph embedding problems where the savings (or penalties) are unbounded in both alternatives?
(2) Do there exist extremal cost trade-offs, that is, situations where the choices are between (roughly) maximal dilation and (roughly) maximal expansion?

(3) Do there exist cost–product trade-offs analogous to the time–space and area–time trade-offs cited earlier?
(4) Do there exist spectral cost trade-offs, that is, situations where there are compromise choices rather than just the choice of optimizing either expansion or dilation?
(5) What gives rise to cost trade-offs in graph embeddings; that is, are there identifiable structural features of the guest and host families that lead to cost trade-offs?

This paper is devoted to studying the above questions. We concentrate on embeddings of graphs in treelike graphs for several reasons. First, many of the real-life situations that can be studied using graph embeddings often seek target graphs that are treelike, since such graphs tend to be easy to manipulate and analyze in a computational environment. Second, certain of our results on tree-targets will have unexpected applications to the study of computational systems having an inherent notion of "computation tree"; and there are many such systems. Finally, trees are a very natural family of graphs and, so, are a reasonable place to begin studying any graph-theoretic phenomenon.

In Section 1.1 we answer question (1) in the affirmative. In particular, we show that the height-h complete ternary tree can be embedded in the height-$2h$ complete binary tree with dilation-cost 2 and expansion-cost n^{λ} where $\lambda = \log_3(\frac{4}{3})$; but any expansion-cost-($c < 2$) embedding of that ternary tree in a complete binary tree must have dilation-cost $\Omega(\log\log h)$.

In Section 1.2 we answer questions (2) and (3) in the affirmative. We consider embeddings of the side-n pyramid graph in the family of "leap trees" [9], trees with certain auxiliary edges added. We show that the side-n pyramid can be embedded in the height-n leap tree with dilation-cost 1 (and expansion-cost $\Omega(2^n/n^2)$); but any embedding of that pyramid in a leap tree of height m (which has expansion-cost $O(2^m/n^2)$) must have dilation-cost $\Omega(\log(n/m))$. When $m = n^{1-\delta}$, this is a maximal dilation-cost, since *any* embedding of the side-n pyramid in a leap tree has dilation-cost $O(\log n)$. Moreover, when $m \geq 4\log n$, this result yields the cost–product trade-off

$$\log(\text{expansion-cost})\rho^{\text{dilation-cost}} \geq \frac{n}{2}$$

for some absolute positive constant ρ.

In Section 2.1 we answer question (4) in the affirmative. We consider the problem of finding, for each integer n, an efficient *n-generic* complete binary tree, that is, a complete binary tree into which all n-node binary trees can be embedded with small dilation-cost and small expansion-cost. We establish the existence of generic trees into which any n-node binary tree can be embedded with dilation-cost $O(1)$ and expansion-cost $O(n^c)$ for some fixed integer c; but any generic tree that affords embeddings that are close to optimal with respect to either of the cost measures must be insufferably inefficient with respect to the other measure: any generic tree that gives rise to embeddings with dilation-costs 1 requires that expansion-costs be $\Omega(n^{(\log n)/2})$; and any generic tree that gives rise to embeddings with expansion-costs $c < 2$ requires that dilation-costs be $\Omega(\log\log\log n)$.

In the course of answering questions (1)–(4), we shall implicitly be answering question (5).

Section 2.2 is devoted to an unexpected dividend of the work in Section 2.1. Using the compromise (polynomial size, constant dilation) generic tree, we are able to

formulate much simplified proofs of a number of results in the literature concerning alternating Turing machines [21], the relationship between deterministic and nondeterministic space [22], and the space requirements of context-free parsing [12]. Indeed, it appears that this tree will be useful whenever one is studying computational systems in which there is a notion of a computation tree and in which short bushy trees are preferable to tall scrawny ones.

1. *Trade-offs in Simple Graph Embeddings*

1.1 AN UNBOUNDED TRADE-OFF. Our first trade-off arises from considering the problem of embedding complete ternary trees in complete binary trees. To avoid misunderstandings, let us define the graphs in question formally.

A *d-ary* tree (d a positive integer) is a graph whose vertex set is a *prefix-closed* set of strings over the alphabet $[d] \triangleq \{1, 2, \ldots, d\}$ (i.e., the string x is in the set whenever any extension $x\sigma$, $\sigma \in [d]$, is in the set), and whose edges connect vertices x and $x\sigma$ for all strings x and $\sigma \in [d]$. We say that a vertex-string x of length k (≥ 0) resides at *level k* of the tree, and that any edge connecting a level-k vertex with a level-$(k+1)$ vertex is a *level-k* edge of the tree. The unique vertex of the tree that resides at level 0 is called the *root* of the tree. A d-ary tree is *complete* if its vertex set comprises the set of all strings over $[d]$ of length not exceeding some integer h; this integer is called the *height* of the tree. We shall denote by $B(h)$ (respectively, $T(h)$) the height-h complete binary—that is, 2-ary (respectively, ternary—that is, 3-ary)—tree. Finally, given any graph G, we shall denote by $|G|$ the cardinality of G's vertex set so, for example, $|B(h)| = 2^{h+1} - 1$.

We can now present our first result.

THEOREM 1. *(a) For all h, there is an embedding of $T(h)$ in $B(2h)$ with dilation-cost 2 (and expansion-cost $\Omega(n^{\lambda})$ where $\lambda = \log_3(\frac{4}{3})$). (b) There is a constant $\alpha > 0$ such that, for infinitely many heights h, any expansion-cost-$(c < 2)$ embedding of $T(h)$ in a complete binary tree has dilation-cost $> \alpha \log\log h$.*

PROOF. Part (a) of the theorem is obvious: the advertised embedding can be viewed as replacing each node of the ternary tree by a height-1 binary tree as illustrated in the following figure:

In the ternary tree: In the binary tree:

To establish part (b), note that any embedding of a graph G in a complete binary tree can be viewed as partitioning G into two sets, namely, those vertices with images in the left half of the tree and all other vertices; call these two sets, respectively, the left and right parts of G. Say that we have embedded the graph G in a complete binary tree and that k of the vertices in the left part of G are adjacent to vertices in the right part of G. Since at least one of these k vertices must reside (i.e., have its image under the embedding) at distance $\Omega(\log k)$ from the root of the binary tree, it follows that the cost of the embedding is $\Omega(\log k)$.

The preceding observations form the basis for proving part (b). To continue the proof, we must look in detail at the source graphs we want to embed in complete binary trees, namely, complete ternary trees, and at the complete binary trees we want to embed them in, namely, the smallest possible ones (for it is only these that will yield the desired expansion-cost <2); that is, we want to embed $T(h)$ in $B(h')$

where $h' = \lceil \log |T(h)| \rceil$. (All logarithms in this paper are to the base 2 unless otherwise indicated.) By simple calculation one notes that $|B(h')|/|T(h)| < 2$ while $|B(h'+1)|/|T(h)| \geq 2$; moreover, since $|T(h)|$ is not a power of 2 for $h > 1$, for these h's, $|B(h')|/|T(h)| > 1$.

Let us look at a fixed but arbitrary h. Assume that we have embedded $T = T(h)$ in $B(h')$. For each $k \in \{0, \ldots, h-1\}$, let e_k denote the number of level-k edges of T having one end in the left part of T and the other end in the right part. Moreover, let us define the "correction" function C as follows: for each k,

$$C(k) = \sum_{i=0}^{h-k-1} 3^{-i} e_{k+i}.$$

(The name "correction" will be self-explanatory later.)

Now look at an arbitrary level k of T. There are 3^k copies of $T(h-k)$—each having $\frac{1}{2}(3^{h-k+1}-1)$ nodes—rooted at this level of T. Since 3^k is odd, at least $\lceil 3^k/2 \rceil$ of the roots of these subtrees must belong either to the left part of T or to the right part; say with no loss of generality that they belong to the left part, call it L. Now, each of the e_{k+i} "split" edges at level $k+i$ of T ($i \in \{0, \ldots, h-k-1\}$) can transfer no more than $|T(h-k-i-1)| = \frac{1}{2}(3^{h-k-i}-1)$ nodes from the set L to the right part $T - L$ of T. It follows that the cardinality of L must satisfy

$$|L| \geq \frac{1}{2}\left\lceil \frac{3^k}{2} \right\rceil (3^{h-k+1}-1) - \mu(k),$$

where μ, which is a lower bound on the number of nodes transferrable from the left part of the tree to the right part, satisfies

$$\begin{aligned}\mu(k) &\leq \sum_{i=0}^{h-k-1} \frac{1}{2}(3^{h-k-i}-1)e_{k+i} \\ &\leq \frac{1}{2} 3^{h-k} \sum_{i=0}^{h-k-1} e_{k+i} 3^{-i} \\ &= \frac{3^{h-k}}{2} C(k).\end{aligned}$$

Let us now look more closely at our source and target trees. It is well known (say from the theory of continued fractions; cf. [3]) that there exist positive constants c such that for infinitely many positive integers m and n, $1 \leq 2^n - 3^m \leq 3^m/cm$. Let m, n be one such pair. Since $3^m - 1 \leq 2^n - 2$, it follows that $|T(m-1)| \leq |B(n-2)|$, so the ternary tree fits in the binary tree. Now consider what happens when our ternary tree $T(h)$ has a height of the form $h = m - 1$ and our binary tree $B(h')$ has a height of the form $h' = n - 2$ for such an m, n pair. (The fact that the binary tree may be a bit bigger than necessary will only strengthen our result.) Let us see in particular what would be the consequences if the tree $T(h)$ had a level $k_0 < \frac{1}{4}\log_3 h$ for which $C(k_0) < \frac{1}{4}$. We would then have

$$\mu(k_0) < \frac{3^{h-k_0}}{8},$$

so that

$$\begin{aligned}|L| &> \frac{3^{k_0}+1}{4}(3^{h-k_0+1}-1) - \frac{3^{h-k_0}}{8} \\ &= \frac{1}{4}\left(3^{h+1} + \frac{5}{2}3^{h-k_0} - 3^{k_0} - 1\right).\end{aligned}$$

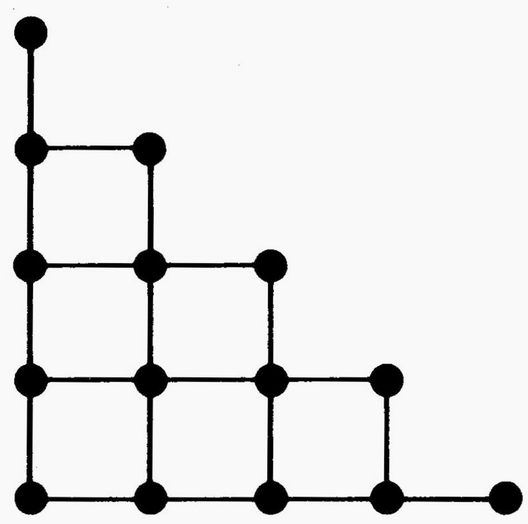

FIG. 1. The side-5 pyramid $P(5)$.

By choice of h, h', m, and n, we know that

$$|L| \leq |B(h'-1)| < 2^{n-2} \leq \frac{3^m}{4} + \frac{3^m}{4cm}.$$

These inequalities combine with our lower bound on $|L|$ to yield (recalling that $h = m - 1$)

$$\frac{5}{2}\,3^{h-k_0} - 3^{k_0} - 1 \leq \frac{3^{h+1}}{c(h+1)},$$

which in turn yields (since $k_0 < \frac{1}{2}\log_3 h$)

$$3^{-k_0} - \frac{2}{5}\,(h^{1/2}+1)3^{-h} \leq \frac{6}{5c(h+1)},$$

which, for sufficiently large h, implies that $k_0 > \frac{1}{2}\log_3 h$, contrary to assumption. We conclude therefore that $C(k) > \frac{1}{4}$ for all $k < \frac{1}{2}\log_3 h$.

Now we are in the home stretch. Since the correction function C behaves as we have just shown, we obtain the following estimate for the number of edges of $T(h)$ having one end in the left part and the other end in the right part (which, of course, is just the sum of the e_k's):

$$\sum_{k=0}^{h-1} e_k > \frac{2}{3}\sum_{k=0}^{h-1} C(k) \geq \frac{1}{12}\log_3 h.$$

Part (b) of the theorem now follows directly from our earlier reasoning. Q.E.D.

1.2 A MAXIMAL TRADE-OFF AND A COST–PRODUCT TRADE-OFF. The trade-off of Theorem 1 illustrates that one can incur unbounded increase in one of the costs of an embedding by insisting on too low a rate of growth for the other cost. But, how large can these unbounded increases be? We now present a situation where a moderate decrease in one of the costs causes the other to jump from its lowest possible value to within a constant factor of its maximum (i.e., to its maximum rate of growth). Again, some preliminary definitions are in order.

The *side-n pyramid* $P(n)$ is the graph whose vertex set is the set of ordered pairs of integers

$$\{\langle i, j\rangle \mid i + j \leq n\}$$

and whose edges connect vertices $\langle a, b\rangle$ and $\langle c, d\rangle$ just when $|a-c| + |b-d| = 1$; see Figure 1. Note that $P(n)$ has $O(n^2)$ vertices.

The *height-h leap-tree* $L(h)$ [9] is obtained from the height-h complete binary tree $B(h)$ by adding the following edges. At each level m of the tree, there are edges connecting vertices

$$2^c 1^{m-c} \quad \text{and} \quad 2^{c+1}1^{m-c-1}$$

(σ^0 = the null string, and $\sigma^{c+1} = \sigma\sigma^c$) for all $c \in \{0, 1, \ldots, m-1\}$. See Figure 2.

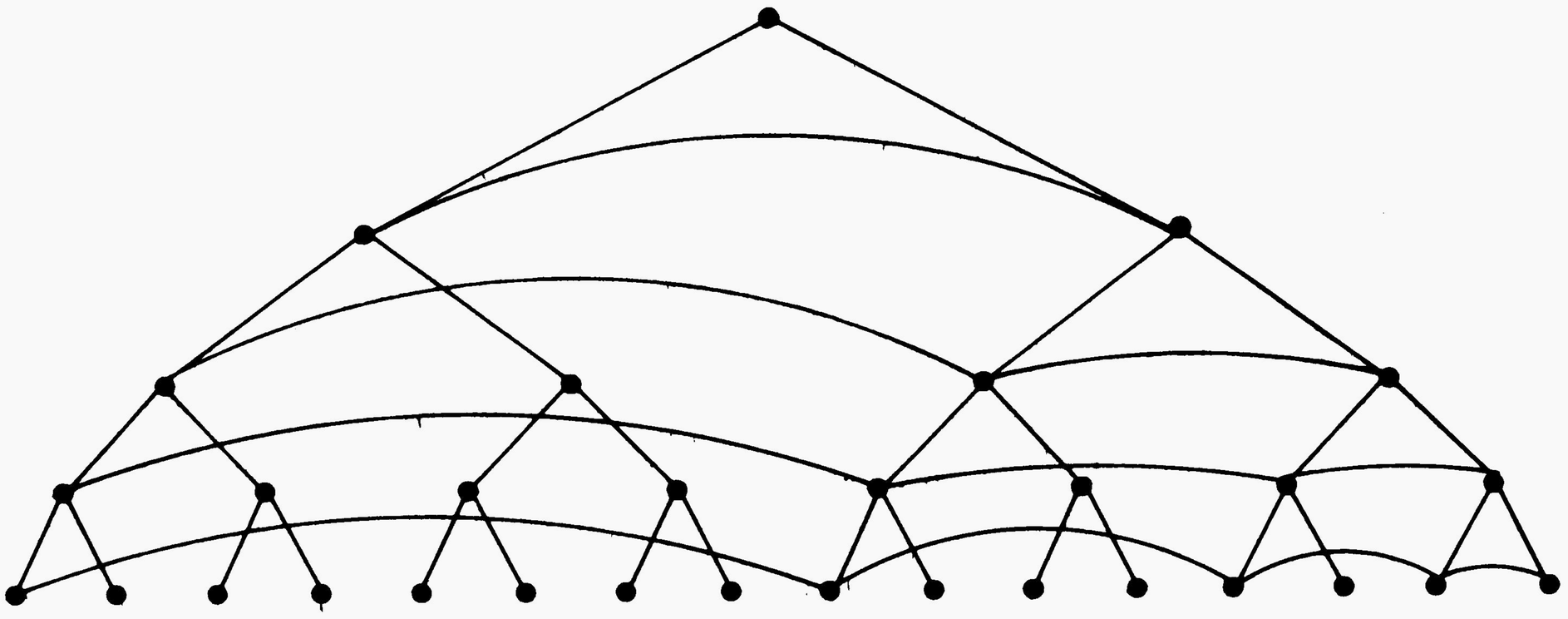

FIG. 2. The depth-4 leap tree $L(4)$.

Our maximal trade-off concerns embeddings of pyramids in leap trees.

THEOREM 2. *(a) For all n, there is an embedding of $P(n)$ in $L(n)$ with dilation-cost* 1 *(and expansion-cost $\Omega(2^n/n^2)$. (b) Any embedding of $P(n)$ in $L(n/\lambda(n))$ for any function λ (perforce, $\lambda(x) = O(x/\log x)$) must have dilation-cost $\Omega(\log\lambda(n))$. (c) In particular, any embedding of $P(n)$ in $L(n^\alpha)$ for any $\alpha < 1$, must have dilation-cost $\Omega(\log n)$.*

The dilation-cost in part (c) of Theorem 2 is worst possible in the sense that any embedding of $P(n)$ in any leap tree has dilation-cost $O(\log n)$.

PROOF. Part (a) of the Theorem is obvious since $P(n)$ is a subgraph of $L(n)$, and part (c) is a special case of part (b); therefore, we need prove only part (b).

Using techniques that are now standard [13, 20], one can show that under any embedding of the graph $P = P(n)$ in the leap tree $L = L(n/\lambda(n))$, there must be some height-$(h < n/\lambda(n))$ sub-leap-tree of L that contains images of between $n^2/4$ and $n^2/2$ vertices of P: if h^* is the largest h such that every height-h sub-leap-tree of L contains strictly fewer than $n^2/4$ images from P, then some height-$(h^* + 1)$ sub-leap-tree must contain at least $n^2/4$ images; but this sub-leap-tree cannot contain more than $n^2/2$ images, since it is made up of a root vertex plus two height-h^* sub-leap-trees. Let $h = h^* + 1$, and let L^* be the desired height-h sub-leap-tree.

From the structure of leap trees, it is obvious that only $h + 1$ edges connect L^* with the rest of L. Using the Boundary Lemma of [13] and the Packing Lemma (4.3) of [17], one can verify that no matter how one cuts the pyramid P into two pieces, one of which has between $n^2/4$ and $n^2/2$ vertices, there must be $\Omega(n)$ edges of P having one end in one of the pieces and the other end in the other piece. This means that $\Omega(n)$ of the vertices of P that have images in L^* are adjacent to vertices of P that have images in $L - L^*$. Of these $\Omega(n)$ vertices, there must be some subset S of cardinality $\Omega(n/h) = \Omega(\lambda(n))$ that satisfy the following: There is some one vertex v in $L - L^*$ and some vertex v' in L^* that is adjacent to v such that the shortest paths from the images of the vertices in S to the images of their neighbors in $L - L^*$ pass through both v and v'. (Mere counting verifies this, since there are only h candidates for v' and $h + 1$ candidates for v.) But now the bottleneck argument of Theorem 1 applies: some one of the images of the vertices in S must lie at distance $\Omega(\log\lambda(n))$ from v' because of L^*'s bounded vertex-degrees. Hence the dilation-cost of the embedding must be $\Omega(\log\lambda(n))$. Q.E.D.

Remark. The lower bounds in Theorem 2, namely, parts (b) and (c), remain valid when we embed pyramids in breadth-first trees rather than in leap trees: the *height-*

h breadth-first tree is obtained from the height-h complete binary tree by adding edges going across each level of the tree. In fact, the bounds remain valid for any embellishment of trees that allows only $O(h)$ edges to connect a height-h subtree to the rest of the tree.

The message of Theorem 2 is the following. If we are willing to suffer expansion-cost $\Omega(2^n/n^2)$, by embedding $P(n)$ in $L(n)$, then we can attain minimum (=unit) dilation cost. If we insist on decreasing the expansion-cost by the $\lambda(n)$th root, by embedding $P(n)$ in $L(n/\lambda(n))$, then we must suffer dilation-cost of $\Omega(\log\lambda(n))$. If we try to get the expansion-cost down below exponential, say by embedding $P(n)$ in $L(n^\alpha)$ for some $\alpha < 1$, then we must suffer dilation-cost $\Omega(\log n)$, which is within a constant factor of worst possible. In this last case, we may as well strive for minimal expansion-cost by embedding $P(n)$ in the smallest leap tree that is big enough to hold it, namely, $L(\lceil\log(n(n+1)/2)\rceil)$; the associated dilation-cost is still only $O(\log n)$.

The description in the last paragraph of Theorem 2's message has in it the hint of a cost–product trade-off. In fact, such a trade-off follows from Theorem 2.

COROLLARY. *There is an absolute constant ρ such that for all n and all $m \geq 4\log n$: for any embedding ϵ of the pyramid $P(n)$ in the leap tree $L(m)$,*

$$\textit{log(expansion-cost}(\epsilon))\rho^{\textit{dilation-cost}(\epsilon)} \geq \frac{n}{2}.$$

PROOF. By definition of the graphs in question, expansion-cost$(\epsilon) \geq 2^m/n^2$; by Theorem 2(b), there is an absolute constant $d > 0$ such that dilation-cost$(\epsilon) \geq d\log(n/m)$. Now let $\rho = 2^{1/d}$, and note that

$$\begin{aligned}\log(\text{expansion-cost}(\epsilon))\rho^{\text{dilation-cost}(\epsilon)} &\geq \log\left(\frac{2^m}{n^2}\right)2^{(1/d)(d)(\log(n/m))}\\ &= n - \frac{2n\log n}{m}\\ &\geq \frac{n}{2},\end{aligned}$$

the last inequality holding because $m \geq 4\log n$. Q.E.D.

1.3 CONCLUSIONS AND OPEN PROBLEMS. This completes our study of cost trade-offs in "standard" graph embeddings. It is clear that (at least) two structural characteristics of the source and target families in an embedding can lead to trade-offs. The trade-off we mentioned in the introduction is due to the bottlenecks formed by the roots of the subtrees of trees; the more dramatic trade-offs exposed in Theorems 1 and 2 result from mismatches in the *separation* characteristics of the families in question. With regard to Theorem 1, binary trees are easy to split in half, while ternary trees are hard to split in half; with regard to Theorem 2, pyramids are very densely connected in that there are usually many edge-disjoint paths connecting one section of the graph with any other, while trees, even when embellished with leap edges or breadth-first edges, are rather sparsely connected.

Open Problem. We leave to the reader the challenge of extending part (b) of Theorem 1 to the case where the expansion-cost is allowed to be $\leq$ any fixed constant c.

A weakness in the development in this subsection is the artificiality of the family of host graphs that yield our maximal trade-offs and our cost–product trade-off. We have actively sought a "natural" guest–host matchup that encounters the kinds of trade-offs exhibited in this section, but with no success. So, we bequeath to the reader

this quest for a "natural" mismatched pair of graph families. Of particular interest would be the discovery of trade-offs, even if not maximal, involving families of graphs having strong connections with real-world applications. Two obvious strong candidates for the host families here are the families of two- and three-dimensional grids, which are used to study problems concerning VLSI circuit layout (see, e.g., [1, 11, 19, 24, 25]).

2. *Trade-offs in Generic Graphs*

2.1 GENERIC BINARY TREES. The cost trade-offs in the previous sections all dealt with the following scenario. One has a family of guest (i.e., source) graphs $\{G\}$ and a family of candidate host (i.e., target) graphs $\{H\}$, and one's task is to find a congenial host H for each guest G. The situation we consider in this section differs from the one described, in that we are now going to measure the congeniality of a proposed host H in terms of how well we can embed in H *all small graphs* from the family $\{G\}$. Our family of guests will be the family of all binary trees, and our candidate hosts will be the family of all complete binary trees. Our task will be to find, for each integer n, a complete binary tree B_n that is a congenial host for all binary trees having n nodes; that is, we wish all such trees to be embeddable in B_n with small expansion-cost and small dilation-cost. (Since n and the tree B_n are fixed here, the expansion-cost of each of the proposed embeddings is just $|B_n|/n$.) We shall term any tree B_n satisfying our efficiency criteria a *generic* binary tree. The motivation for studying generic graphs is discussed in [7], where a variety of notions of "generic" are considered; in [18]; in [1], where generic rectangular grids of various dilation-costs and expansion-costs are constructed; and in [5], where dilation-cost 1 is demanded and expansion-cost is studied. (When one insists on dilation-cost 1, generic graphs are often termed *universal*.)

THEOREM 3. (*a*) *There are generic trees B_n into which all n-node binary trees are embeddable with dilation-cost $O(1)$ and expansion-cost $O(n^c)$ for some fixed constant c.* (*b*) *Any universal binary tree has size $\Omega(n^{(\log n)/2})$, even if the definition of universal is relaxed so the tree need not be complete* [5]. (*c*) *There is a constant $\alpha > 0$ such that for infinitely many n, if the generic tree B has size $< 2n$, then there are n-node binary trees T such that any embedding of T into B has dilation-cost $> \alpha \log\log\log n$.*

The theorem tells us that we can find generic binary trees of moderate efficiency providing that we are not too greedy; but if we try to optimize either of the cost measures, we can do so only by letting the other measure grow insufferably large.

PROOF. We shall prove the three parts of the theorem in turn.

(a). Part (a) of the theorem is proved by means of a recursive construction that can be summarized as follows.

Let T be any n-node binary tree. Let v_0 (the root of T), $v_1, \ldots, v_k$ be that root-to-leaf path in T obtained by always following the edge from vertex v_i that leads into the bigger of the two subtrees rooted at vertex v_i. Define the *weight* of the nodes along this path as follows:

$$\begin{aligned} w(v_0) &= n; \\ w(v_i)\ (i \neq 0) &= \text{the number of nodes in the subtree of } T \\ &\qquad \text{rooted at the } \textit{brother} \text{ of } v_i. \end{aligned}$$

By combining the construction of optimal binary search trees in [16] with the techniques in [9] for efficiently embedding outerplanar graphs in binary trees, one

can embed the line-graph represented by our root-to-leaf path in a binary tree in such a way that

(1) the embedding has dilation-cost $\leq c$ for some fixed constant c;
(2) each node v_i, $0 \leq i \leq k$, is at distance at most $d(1 + \log(W/w(v_i)))$ [where $W = \sum w(v_i)$] from the root of the host tree for some fixed constant d.

Now add an additional node to every edge of the host tree constructed thus far, and recursively use the node adjacent to the image of node v_i as the root of a tree which will hold (under an embedding using the same construction) the subtree of T rooted at the "other" son of v_i—the one not on the root-to-leaf path.

A careful analysis of this construction finds that

(1) the final embedding of T has dilation-cost $\leq c$ for some fixed constant c independent of T and n;
(2) every node of the final host binary tree is at distance $\leq d \log n$ from the root of the tree, where d is a fixed constant independent of T and n.

What these facts mean is that we can "pad" the constructed host tree out to a height-$(d \log n)$ complete binary tree B with the assurance that every n-node binary tree is embeddable in B with dilation-cost at most c.

Now on to the details of the construction. We begin by looking at embeddings of graphs in the rooted infinite binary tree. For brevity, we shall henceforth omit the qualifier "rooted."

LEMMA 1. *There is a constant c such that every n-vertex line-graph $L(n)$ with integer-valued vertex-weighting function w can be embedded into the infinite binary tree in such way that*

(a) the dilation-cost of the embedding is $\leq 2c$;
(b) the depth (i.e., distance from the root) of the image of vertex v of $L(n)$ in the host tree is $\leq (3 + \log(W/w(v)))c$, where $W = w(1) + w(2) + \cdots + w(n)$.

PROOF. It is proved in [16] that given any sequence of integers $w_1, w_2, \ldots, w_n$, summing to W, there is a binary tree T^* with n leaves such that the depth of the ith leaf (numbered from left to right) is $\leq 2 + \log(W/w_i)$. Let us use our line-graph's vertex-weights as our sequence of integers, and construct the tree T^*.

Now, consecutively numbered leaves of T^* may be very far apart. Let us remedy this by augmenting T^* with *inorder edges*: we connect each leaf k of T^* that is a left (respectively, right) son to its closest ancestor which is exited via a right (respectively, left) link on the path from the root of T^* to k. Consecutive leaves in the resulting graph are close to each other (in fact, distance 2), but the graph is no longer a tree. However, the graph is *outerplanar* (it can be embedded in the plane so that all vertices lie on the outer face), and it has maximum vertex-degree 4. Therefore, we can invoke the following result from [9], with the root of T^* as the "designated vertex," to obtain the embedding sought in the statement of the lemma.

THEOREM 4 [9]. *Given any outerplanar graph G, there is an embedding of G in a binary tree with*

$$\textit{dilation-cost} \leq 3 \log(2 \, \textit{maxdegree}(G)).$$

Moreover, this embedding can be chosen to place any designated vertex of G at the root of the host tree.

The lemma follows by composing the embedding of $L(n)$ in (the augmented version of) T^* with Theorem 4's embedding of T^* in a binary tree. □

Lemma 1 affords us the basic step in our construction. The remainder of the construction resides in the following lemma.

LEMMA 2. *There is a constant $c > 0$ such that every n-node binary tree T can be embedded in the infinite binary tree with dilation-cost $\leq 8c + 3$, in such a way that the image of the root of T is at depth at most $8c + 1$ in the host tree, and the image of every node of T is at depth at most $(10c + 1)\log n$ in the host tree.*

Remark. By carefully analyzing the construction used in [9] to prove Theorem 4, one can show that the constant c we have been talking about need not exceed 6.

PROOF OF LEMMA. We proceed by induction on n. The claim is obvious for $n = 2$, so assume $n > 2$.

Let v_0 (the root of T), $v_1, \ldots, v_k$ be a root-to-leaf path in T such that for each $i > 0$, the subtree of T rooted at node v_i has at least as many nodes as the subtree—call it D_{i-1}—rooted at v_i's brother. (Some of the D_j's may be empty, if v_j has no brother.) View our root-to-leaf path as a line-graph in the natural way, and weight its nodes as follows.

$$w(v_0) = n;$$
$$w(v_i)\ (i \neq 0) = \max(1, |D_i|).$$

Note the following properties of these vertex weights:

$$n < W \triangleq \sum_i w(v_i) \leq 2n;$$

$$w(v_i) \leq \frac{n}{2}, \qquad \text{for} \quad i > 0;$$

$$\frac{W}{w(v_i)} \geq \frac{w(v_0) + w(v_i)}{w(v_i)} \geq 3, \qquad \text{for} \quad i > 0.$$

Finally, note that each D_i has at most $w(v_i) \leq n/2$ nodes. Hence, by our inductive hypothesis, D_i is embeddable in the infinite binary tree as claimed in the statement of the lemma.

Now let us embed the line-graph $v_0, \ldots, v_k$ into the infinite binary tree in the way described in Lemma 1. Let c be the constant guaranteed by that lemma. Then our line is embedded so that

(a) the dilation-cost is $\leq 2c$;
(b) the depth of the image of vertex v_i in the host tree is $\leq (3 + \log(W/w(v_i)))c$;
(c) in particular, the depth of the image of v_0 is $\leq (3 + \log(2n/n))c = 4c$.

Assume that we have embedded our line-graph and, by recursive invocation of the lemma, the trees D_i. We wish to combine the embeddings, and in order to do so, we must "stretch" the embedding of the line-graph to make room for the embeddings of the D_i's. We accomplish this as follows.

Imagine that we have two copies B and B' of the infinite binary tree. Consider the following embedding of B in B'. Each node $v = a_0 a_1 \cdots a_m$ ($\in \{1, 2\}^*$) of B is mapped onto node $v' = (1a_0)(1a_1) \cdots (1a_m)1$ of B' (so, e.g., the root node λ of B is mapped onto node 1 of B', nodes 1 and 2 of B are mapped onto nodes 111 and 121 of B', respectively, and so on). This embedding obviously has dilation-cost 2, and it maps a node at depth d in B to a node at depth $2d + 1$ in B'. Moreover, if the node

$v' = v''1$ of B' is in the range of the embedding, then the subtree of B' rooted at the brother $v''2$ of v' contains no image points under the embedding.

We can now describe completely our embedding of the tree T in the infinite binary tree.

1. Embed the weighted line-graph $v_0, v_1, \ldots, v_k$ in the infinite binary tree by first embedding it according to Lemma 1 and then composing this embedding with the just-described embedding of the infinite binary tree into itself. This composite embedding satisfies:

$$\begin{aligned}&\text{dilation-cost} \leq 4c;\\ &\text{depth}(\text{image}(v_i)) \leq 2c(3 + \log(W/w(v_i))) + 1;\\ &\text{depth}(\text{image}(v_0)) \leq 8c + 1.\end{aligned}$$

2. Embed each D_i (in the manner prescribed by the lemma) into the subtrees of the infinite binary tree that are left "unused" by the embedding of the line-graph. More precisely, if $\text{image}(v_i) = v''1$, then embed D_i in the subtree rooted at $v''2$.

It remains to verify that the described embedding has the properties claimed in the statement of the lemma. We address each claim in turn.

First, we have already noted that the root of T is embedded at distance at most $8c + 1$ from the root of the host tree.

Second, we look at the dilation-cost of the embedding. There are three cases to consider. Let the nodes v_a and v_b be adjacent in T. (a) If both nodes lie on the root-to-leaf line-graph, then, as we have remarked earlier, their images are distance at most $4c$ apart in the host tree. (b) If both nodes reside in one of the subtrees D_i of T, then the induction hypothesis guarantees that their images are distance at most $8c + 3$ apart in the host tree. (c) Finally, if one of the nodes is on the line-graph and the other is in one of the D_i subtrees, then it must be that, say, v_a is one of the v_i on the line-graph, while v_b is the root of D_i. In this case, the image of v_a is of the form $v''1$, and the image of v_b must reside (by our induction hypothesis) at distance at most $8c + 1$ from the node $v''2$. Hence, the distance in the host tree between the images of v_a and v_b is at most $8c + 3$.

Finally, we must estimate the maximal depth in the host tree of the image of any node of T. To this end, let us consider an arbitrary node v of T. If v is one of the nodes of our line-graph, say $v = v_i$, then

$$\begin{aligned}\text{depth}(\text{image}(v)) &\leq 2c(3 + \log(W/w(v))) + 1\\ &< 2c(3 + \log 2n) + 1\\ &< (10c + 1)\log n,\end{aligned}$$

the last inequality following since $n \geq 2$. If, instead, v is a node of the subtree D_i, then we note that the depth of v is no greater than the sum of the depth of the image of v_i under the embedding and the depth of the image of v relative to the embedding of D_i in its host subtree. By our earlier remarks, the former quantity is at most $2c(3 + \log(W/w(v_i))) + 1$; by our bound on the size of D_i and by the inductive hypothesis, the latter quantity is at most $(10c + 1)\log \min(w(v_i), n/2)$. It follows that

$$\text{depth}(\text{image}(v)) \leq 1 + 2c(4 + \log(n/w(v_i))) + (10c + 1)\log \min(w(v_i), n/2).$$

Now, if $i = 0$, so $w(v_i) = n$, then this sum is bounded above by

$$8c + 1 + (10c + 1)\log(n/2) < (10c + 1)\log n;$$

if $i > 0$, so $w(v_i) \leq n/2$, then the sum is bounded above by

$$\begin{aligned} &8c + 1 + 2c\log n - 2c\log w(v_i) + (10c + 1)\log w(v_i) \\ &\quad \leq (8c + 1)(1 + \log(n/2)) + 2c\log n \\ &\quad \leq (10c + 1)\log n. \end{aligned}$$

This completes the proof of Lemma 2. □

Part (a) of the theorem follows immediately from Lemma 2. We have embedded an arbitrary n-node binary tree in the infinite binary tree in such a way that adjacent nodes in the guest tree are kept within fixed constant distance in the host tree, and every node of the guest tree is within depth $O(\log n)$ in the host tree. If we cut the infinite binary tree at level $(10c + 1)\log n$ to give us a complete binary tree of that depth, then the embedding technique of Lemma 2 assures us that the resulting complete tree is the generic tree claimed in part (a).

Remark. The authors have recently learned from W. L. Ruzzo (personal communication) that he has obtained a simplified proof of part (a), which yields smaller constants than our proof.

(b) Part (b) is proved (by a very clever argument) in [5], to which source the reader is referred.

(c) Part (c) is proved by means of a variant of the proof of Theorem 1, which we sketch here in some detail. Let us "transliterate" each n-node complete ternary tree T into a $(2n - 1)$-node binary tree $\beta(T)$ by replacing each nonleaf node of T by the height-3 binary tree depicted in the following figure.

In the ternary tree: In the binary tree:

We note a few salient characteristics of $\beta(T)$. If T has height h, then $\beta(T)$ has height $3h$. Each node at level k of $\beta(T)$ is the root of a subtree: if $k \equiv 0 \pmod 3$ and $k < 3h$, then the subtree contains $3^{h+1-k/3} - 2$ nodes; if $k \equiv 1 \pmod 3$, then the subtree contains $3^{h+1-(k-1)/3} - 3$ nodes; and if $k \equiv 2 \pmod 3$, then the subtree contains at most $2 \times 3^{h+1-(k+1)/3} - 3$ nodes.

Let us embed the tree $T = \beta(T(h))$ in the smallest complete binary tree B that will hold it. (As in Theorem 1, this assumption about the "smallest" binary tree is needed to attain the sought expansion-cost.) We wish to show that, no matter how T is embedded in such a small binary tree B, many edges of T have their endpoints in opposite halves of B. The bottleneck argument of Theorem 1 will then show that some of these endpoints must be very far apart in B, in fact distance $\Omega(\log\log\log n)$ apart.

We proceed basically as in the proof of Theorem 1. Let us assume we have embedded $T = \beta(T(h))$ in the binary tree B, and let us concentrate on levels k of T such that 3 divides k. Each such level k contains $3^{k/3}$ nodes; therefore, our embedding of T in B must relegate some $\lceil 3^{k/3}/2 \rceil$ of these nodes either to the left part of B or to the right part; say they are placed in the left part, call it L. As in the proof of Theorem 1, let e_k be the number of edges at level k of T whose endpoints are put into different

parts of B. We find that the cardinality $|L|$ of the set L satisfies

$$|L| \geq \left\lceil \frac{3^{k/3}}{2} \right\rceil (3^{h+1-k/3} - 2) - \sum_{i=0}^{h-1-k/3} 3^{h-k/3-i}(3e_{k+3i} + 2e_{k+3i+1} + e_{k+3i+2}),$$

which implies that

$$|L| > \frac{1}{2}(3^{h+1} + 3^{h+1-k/3}) - 3^{k/3} - 3^{h-k/3}C(k),$$

where

$$C(k) = \sum_{i=0}^{h-1-k/3} 3^{-i}(3e_{k+3i} + 2e_{k+3i+1} + e_{k+3i+2}).$$

Now assume that the height h of our ternary tree is chosen so that 3^{h+1} is almost a power of 2, that is, so that for some constant c,

$$1 \leq 2^n - 3^{h+1} \leq \frac{3^{h+1}}{c(h+1)}.$$

We can then assume that our host tree B has height $n-1$. Since $|L|$ is then at most 2^{n-1}, our previous bound on this quantity combines with our assumed relation between n and h to yield

$$3^{h-k/3}\left(\frac{3}{2} - C(k)\right) - 3^{k/3} \leq \frac{3^{h+1}}{2c(h+1)}.$$

Assume now that $C(k_0) \leq \frac{1}{2}$ for some $k_0 < \frac{1}{2}\log_3 3h$ that is divisible by 3. It follows that

$$3^{h-k_0/3} - 3^{k_0/3} \leq \frac{3^{h+1}}{2c(h+1)}.$$

Since $k_0 < \frac{1}{2}\log_3 3h$, this implies that

$$3^{-k_0/3} - (3h)^{1/6}3^{-(h+1)} \leq \frac{1}{2c(h+1)}.$$

However, when h is sufficiently large, this inequality implies that $k_0 > \frac{1}{2}\log_3 3h$, contrary to assumption. We conclude that $C(k) > \frac{1}{2}$ for all k satisfying: $k \equiv 0 \pmod 3$ and $k < \frac{1}{2}\log_3 3h$.

As in the proof of Theorem 1, this bound on C yields the lower bound we have been seeking on the number of edges of T whose endpoints are placed by our supposed embedding in different subtrees of B. By definition, the quantity to be bounded is just the sum of the e_k's, which is bounded as follows:

$$\sum_{0 \leq k < 3h} e_k > \sum_{\substack{0 \leq k < 3h \\ k \equiv 0 \pmod 3}} C(k) \geq \frac{1}{54}\log_3 3h.$$

By the bottleneck argument of Theorem 1, this last inequality implies that the image in B of some vertex of T is distance $\Omega(\log\log h)$ from the image of one of its neighbors in T. Part (c) follows. Q.E.D.

2.2 APPLICATIONS OF SMALL GENERIC TREES. A number of results appear in the literature that have as their intuitive content the message that "tall but sparse computation trees may be transformed into short, bushy ones" [21(b), p. 356]. But this same remark (without the word "computation") describes the content of Theorem 3(a). And, indeed, the construction of Theorem 3(a) affords one a single general

construction that yields simplified proofs of a number of results concerning "computation tree" situations, that is, situations where computational systems operate by generating computation trees and where added efficiency is attainable if these trees are short and fat rather than long and skinny. We cite but two examples of such situations, the earliest published one, namely the problem of parsing (or recognizing) context-free languages [12]; and the most recently published one, namely the problem of efficiently simulating space-tree-size efficient alternating Turing machines by time-space efficient ones [21]. The reader is referred also to the problem of simulating nondeterministic automata by deterministic ones [22]. We shall illustrate the use of our generic tree construction by presenting simple proofs of the two sample results. The reader should be able easily to apply the same proof strategy to the results of [22] and, indeed, to any computational problem desiring bushy trees to replace scrawny ones.

THEOREM 5 [12]. *Every context-free language is recognizable by a deterministic Turing machine operating in space* $S(n) = O(\log^2 n)$.

THEOREM 6 [21]. *For* $S(n) \geq \log n$, *any language that is recognizable by an alternating Turing machine operating simultaneously in space* $S(n)$ *and tree-size* $Z(n)$ *is recognizable by an alternating Turing machine operating simultaneously in time* $S(n) \log Z(n)$ *and space* $S(n)$.

Remark. It will follow from our proof of Theorem 6 that the time-space efficient machine will operate in tree-size $O(S(n)Z(n)^{O(1)})$.

We shall sketch a proof of Theorem 5 and present a detailed proof of the more difficult Theorem 6: this latter will contain all of the ideas needed to convert our proof of Theorem 3(a) to any of the dynamic computation-tree situations alluded to. In order to simplify exposition and save space, we shall prove Theorem 6 first and then use the notions developed there to prove Theorem 5.

PROOF OF THEOREM 6. We assume the reader is familiar with alternating Turing machines (ATMs, for short); the reader seeking such background should consult [4, 21] and sources cited there. It follows easily from the notions defined there that we lose no generality by assuming that the ATMs we consider operate as follows: (1) An ATM that operates in space $S(n) = \Omega(\log n)$ can *clock* itself so that it never goes into an infinite loop. It accomplishes this by counting the number of computational cycles it goes through, and by aborting any computation that takes more than $c^{S(n)}$ steps, where the constant c depends (in an easily derived way) on the internal characteristics of the ATM: the number of work tapes, the number of internal states, and the number of tape symbols. (2) Whenever an ATM splits in either an existential or universal state, it splits into two copies. Thus the *computation tree* of the ATM is a binary tree. It is in this computation tree that we find the connection between alternating Turing machines and our generic trees. Let us make this connection precise.

An ATM A that operates in space $S(n)$ and tree-size $Z(n)$ accepts an input string x of length n if and only if there is (a) a binary tree $T = T(x)$ having at most $Z(n)$ nodes and (b) a labeling function

$$\lambda\text{: Nodes}(T) \rightarrow \text{CONFIG},$$

where CONFIG is the set of configurations of the ATM using space at most $S(n)$, such that

(a) the root of T is labeled with the initial configuration of A on input x;

(b) the leaves of T are labeled by accepting configurations of A;
(c) if v is any nonleaf node of T that is labeled by a universal (respectively, existential) configuration of A, then v's descendants include both (respectively, one) of the successors of that configuration.

The tree $T(x)$ is called the *computation tree of A on input x.*

Now let ϵ: $T(x) \to B$ be any embedding of T in the infinite binary tree B that satisfies the conditions of Lemma 2. Define the *contents function*

$$K\text{: Nodes}(B) \to \text{CONFIG} \cup \{\#\},$$

where # is a new symbol, as follows:

$$K(v) = \textbf{if } v \text{ is the image of a node } u \text{ of } T \textbf{ then } \lambda(u) \textbf{ else } \#.$$

Thus we label the images of the nodes of T according to λ, and we label all nonimages by the special symbol #. The mapping K has the following properties that are crucial to our task.

LEMMA 3. (*a*) *The infinite labeled binary tree $K(B)$ has the following properties, where c is the constant of Lemma 2:*

(*i*) *There is a node v_0 at depth $\leq 8c + 1$ such that $K(v)$ is the initial configuration of the ATM A on input x;*
(*ii*) *if $K(v)$ is a nonaccepting existential (respectively, universal) configuration of the ATM A, then there is a node v_1 (respectively, nodes v_1 and v_2) within distance $8c + 3$ of v such that $K(v_1)$ (respectively, $K(v_1)$ and $K(v_2)$) is a successor (respectively, are both successors) of $K(v)$.*
(*iii*) *$K(v) = \#$ for all v at depth $> (10c + 1)\log Z(n)$.*

(*b*) *Conversely, if K: Nodes(B) $\to$ CONFIG $\cup$ {#} is any mapping satisfying conditions (i)–(iii) of part (a), then the ATM A accepts input x.*

PROOF. Part (a) is immediate from Lemma 2 and from the assumed properties of the tree T and the mapping λ.

To prove part (b), we shall assume the existence of the mapping of K satisfying conditions (i) and (ii) of part (a), and we shall construct from it a computation tree T with labeling function λ (and, implicitly, an embedding ϵ of T in B).

Let r be the root of T: we let $\epsilon(r)$ be that node v of B that contains the initial configuration of A on input x (v exists by property (i)), and we set $\lambda(r) = K(v)$.

Let u be any node of T such that $\epsilon(u)$ and $\lambda(u)$ $(=K(\epsilon(u)))$ have already been constructed. If $\lambda(u)$ is an accepting configuration of A, then u is a leaf of T. Otherwise, if $\lambda(u)$ is an existential (respectively, a universal) configuration of A, then give u a son u_1 (respectively, sons u_1 and u_2) with ϵ-value (respectively, values) set to node v_1 (respectively, nodes v_1 and v_2) of B whose existence is guaranteed by property (ii); and we set $\lambda(v_i) = K(v_i)$. Since each move of A is either an existential (i.e., 1-successor) or a universal (i.e., 2-successor) move, the described process is guaranteed to be successful.

Since the ATM A is clocked, the tree T is finite; by construction the successor relation in T leads configurations of A to their successor(s); and by construction, all leaves of T are accepting configurations of A. It follows that the ATM A accepts input x. □

Lemma 3 shows that A accepts input x *if and only if* there is a labeling K of the infinite binary tree B satisfying properties (i)–(iii). We shall now indicate how an

ATM of time-complexity $S(n)Z(n)$ and space-complexity $S(n)$ might, when confronted with input x, check for the existence of such a labeling function. The ATM A^* we describe will guess the mapping in pieces and will verify its guesses in parallel using universal splitting. We now describe this process in some detail.

By *the ball in T centered at node v* we shall mean the set of all nodes of T that are at distance at most $8c + 3$ from node v (where c is the constant of Lemma 2).

The *total state* of the ATM A^* will be a ball in T, a mapping K defined on the nodes of the ball (with values in CONFIG $\cup$ $\{\#\}$), and an integer variable whose value will be the depth in B of the image of the center of the current ball.

The mode of operation of A^* will be to guess the contents of a ball (via existential moves), to verify that this new ball is consonant with the previous ball (via deterministic moves), and then to split to check successors of the current ball centered deeper in B (via universal moves). This simulation starts with a guess of the ball centered at the root of B. It ends when the depth variable indicates a depth of $(10c + 1)\log Z(n)$, at which point A^* checks that all nodes at that depth in B are labeled with $\#$. We describe the simulation process in more detail.

begin

Initialize, and check condition (i) of Lemma 3.

$v \leftarrow \text{root}(B)$; $d \leftarrow 0$;

Guess (via existential moves) the mapping K on the nodes of the ball of B centered at v;

if no node w in the current ball has $K(w)$ = initial configuration of A on input x
then stop and reject;

Move down B and check condition (ii) of Lemma 3.

while $d \leq (10c + 1)\log Z(n)$
do if $K(v) \neq \#$ **and** $K(v)$ is a nonaccepting configuration of A **and** $K(v)$ is universal (respectively, existential) **and** there are not two (respectively, one) nodes in the current ball labeled (by K) with successors of configuration $K(v)$
then stop and reject;

$v \leftarrow$ a son of v (via universal branching);

Guess (existentially) the value of K on all nodes of the ball centered at the new v that were not in the ball centered at the old v; and discard all nodes that were in the ball centered at the old v that are not in the ball centered at the new v;

$d \leftarrow d + 1$;

od;

Check condition (iii) of Lemma 3.

if $K(u) = \#$ for all nodes u in the ball whose depth is at least equal to that of v
then stop and accept
else stop and reject

end

The ATM A^* is a direct implementation of Lemma 3(a) and so must operate correctly by Lemma 3(b). It thus remains only to prove that A^* operates within the claimed complexity bounds.

Space Bound. At any given stage in its computation, A^* has stored (a) at most two overlapping balls, each containing a bounded number of tree nodes, (b) the values of K on each node of these balls, each value taking space $O(S(n))$, and (c) the depth counter whose size need be at most $\log\log Z(n) = O(S(n))$; this bound on the size of the counter follows since the ATM A has at most $\exp sS(n)$ distinct configurations for some constant s, so that an accepting computation tree of A has size at most $\exp\exp sS(n)$. It follows that the ATM A^* uses space $O(S(n))$.

Time Bound. The computation of A^* comprises a highly parallel exploration of the first $O(\log Z(n))$ levels of the infinite binary tree B. When processing each visited node of B, A^* spends time $O(S(n))$ guessing configurations of A and checking the compatibility of the new guesses with previous ones. Thus, A^* computes for a total of $O(S(n)\log Z(n))$ steps.

Tree Size. Finally, note that A^* operates in tree size $O(S(n)Z(n)^{O(1)})$, since A^*'s computation tree is a complete binary tree of depth $O(\log Z(n))$—representing the universal branchings—where each node is really a line (i.e., 1-ary tree) of size $O(S(n))$—for representing configurations of A during the guessing and checking.

The proof of the theorem is now completed by invoking the standard time- and space-compression ("speedup") theorems for Turing machines. Q.E.D.

Remark. We have assumed in our simulation algorithm that the algorithm knows the tree-size $Z(n)$ and the space $S(n)$. In fact, the result still holds if less is known. Specifically, since $\log\log Z(n) = O(S(n))$, knowledge of $Z(n)$ is unnecessary; and even $S(n)$ need not be known, since the algorithm could try, in turn, $S(n) = 1, 2, 4, 8, \ldots$.

PROOF SKETCH FOR THEOREM 5. We assume the reader is familiar with the basics of context-free languages (CFLs) and grammars (CFGs); the unequipped reader should consult [12] and sources cited there. It is well known that every CFL can be generated by a CFG in *Chomsky Normal Form*, that is, one for which every production is of the form $A \to BC$ or of the form $A \to b$, where A, B, C are nonterminal symbols, and b is a terminal symbol. Thus we lose no generality in restricting attention to such grammars. Derivation trees generated by CFGs in Chomsky Normal Form are binary trees. It is in these derivation trees that we find the sought connection between context-free languages and our generic trees. Let us make this connection precise.

A CFG G generates an input string $x = x_1x_2 \cdots x_n$ of length n if and only if there is a binary tree $T = T(x)$ and a labeling function

$$\lambda\text{: Nodes}(T) \to \text{CONFIG},$$

where CONFIG is the set of *configurations* of G, that is, the set of triples of the form $[A, i, j]$, A being a nonterminal symbol and i, j being integers with $1 \le i \le j \le n$, such that

(a) the root of T is labeled with the initial configuration of G on string x, that is, $[S, 1, n]$ (S is the axiom of G);
(b) the leaves of T contain from left to right configurations of the form $[A, i, i]$, where $A \to x_i$ is a production of G;
(c) if v is any nonleaf node of T that is labeled by the configuration $[A, i, j]$, then v's descendants have labels of the form $[B, i, k]$ and $[C, k + 1, j]$, where $A \to BC$ is a production of G.

The tree $T(x)$ is called the *derivation tree of x under G.*

As in the proof of Theorem 6, one can now let ϵ: $T(x) \to B$ be any embedding of T in the infinite binary tree B that satisfies the conditions of Lemma 2, and define the *contents function*

$$K\text{: Nodes}(B) \to \text{CONFIG} \cup \{\#\},$$

where $\#$ is a new symbol, by

$$K(v) = \textbf{if } v \text{ is the image of a node } u \text{ of } T \textbf{ then } \lambda(u) \textbf{ else } \#.$$

Thus we label the images of the nodes of T according to λ, and we label all nonimages by the special symbol #.

One can now prove easily, using our proof of Lemma 3, an analog of that lemma for the current contents function K: the analog is obtained by merely replacing every occurrence of $Z(n)$ in the lemma and its proof (including in the program) by $f(n) = 2n - 1$.

One thus obtains an ATM A^* that recognizes the language generated by G and that operates in time $T(n) = O(\log^2 n)$. (This time bound follows since A^* works down the generic binary tree to depth $O(\log n)$, spending time $O(\log n)$ at each level to guess the mapping K and to validate its guesses.) Theorem 5 follows immediately from the existence of A^* in view of the following result of Chandra et al.

THEOREM 7 [4]. *Every language accepted by an alternating Turing machine that operates in time $T(n)$ is accepted by some deterministic Turing machine that operates in space $T(n)$.*

Of course, one could prove Theorem 5 also by means of a direct backtracking algorithm. Q.E.D.

REFERENCES

1. ALELIUNAS, R., AND ROSENBERG, A.L. On embedding rectangular grids in square grids. *IEEE Trans. Comput., C-31* (1982), 907–913.
2. BORODIN, A., FISCHER, M.J., KIRKPATRICK, D., LYNCH, N.A., AND TOMPA, M. A time–space tradeoff for sorting on nonoblivious machines. In *Proc. 20th IEEE Symp. on Foundations of Computer Science* (San Juan, Puerto Rico, Oct. 1979), IEEE, New York, pp. 319–327.
3. CASSELS, J.W.S. *An Introduction to Diophantine Approximation.* Cambridge Tracts in Mathematics and Mathematical Physics 45, Cambridge University Press, Cambridge, England, 1957.
4. CHANDRA, A.K., KOZEN, D.C., AND STOCKMEYER, L.J. Alternation. *J. ACM 28,* 1 (Jan. 1981), 114–133.
5. CHUNG, F.R.K., COPPERSMITH, D., AND GRAHAM, R.L. On trees containing all small trees. Unpublished manuscript, 1979.
6. COBHAM, A. The recognition problem for the set of perfect squares. In *Proc. 7th IEEE Symp. on Switching and Automata Theory* (Berkeley, Calif., 1966), IEEE, New York, pp. 78–87.
7. DEMILLO, R.A., EISENSTAT, S.C., AND LIPTON, R.J. On small universal data structures and related combinatorial problems. In *Proc. Johns Hopkins Conf. on Information Sciences and Systems,* Baltimore, Md., 1978, pp. 408–411.
8. HONG, J.-W. On similarity and duality of computation. *J. Comput. Syst. Sci.* (to appear).
9. HONG, J.-W., AND ROSENBERG, A.L. Graphs that are almost binary trees. *SIAM J. Comput. 11* (1982), 227–242.
10. KUNG, H.T., AND STEVENSON, D. A software technique for reducting the routine time on a parallel computer with a fixed interconnection network. In *High Speed Computer and Algorithm Optimization,* Academic Press, New York, 1977, pp. 423–433.
11. LEIGHTON, F.T., AND ROSENBERG, A.L. Three-dimensional circuit layouts. Unpublished manuscript, 1982.
12. LEWIS, P.M., STEARNS, R.E., AND HARTMANIS, J. Memory bounds for recognition of context-free and context-sensitive languages. In *Proc. 6th Conf. on Switching Circuit Theory and Logical Design,* Ann Arbor, Mich., 1965, pp. 191–202.
13. LIPTON, R.J., EISENSTAT, S.C., AND DEMILLO, R.A. Space and time hierarchies for collections of control structures and data structures. *J. ACM 23,* 4 (Oct. 1976), 720–732.
14. LIPTON, R.J., AND TARJAN, R.E. Applications of a planar separator theorem. In *Proc. 18th IEEE Symp. on Foundations of Computer Science* (Providence, R.I., Oct. 1977), IEEE, New York, pp. 162–170.
15. LIPTON, R.J., AND TARJAN, R.E. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36* (1979), 177–189.
16. MEHLHORN, K. Best possible bounds on the weighted path length of optimum binary search trees. *SIAM J. Comput. 6* (1977), 235–239.
17. ROSENBERG, A.L. Encoding data structures in trees. *J. ACM 26,* 4 (Oct. 1979), 668–689.

18. ROSENBERG, A.L. Issues in the study of graph embeddings. In *Graph-Theoretic Concepts in Computer Science* (Proc. of the Int. Workshop WG80), Lecture Notes in Computer Science 100, H. Noltemeier, Ed., Springer-Verlag, New York, 1981, pp. 150–176.
19. ROSENBERG, A.L. Three-dimensional VLSI: A case study. *J. ACM 30,* 3 (July 1983), 397–416.
20. ROSENBERG, A.L., AND SNYDER, L. Bounds on the costs of data encodings. *Math. Syst. Theory 12* (1978), 9–39.
21. RUZZO, W.L. Tree-size bounded alternation. (a) *J. Comput. Syst. Sci. 21* (1980), 218–235; (b) see also preliminary version in *Proc. 11th ACM Symp. on Theory of Computing* (Atlanta, Ga., May 1979), ACM, New York, pp. 352–359.
22. SAVITCH, W.J. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci. 4* (1970), 177–192.
23. SEKANINA, M. On an ordering of the set of vertices of a connected graph. *Publ. Fac. Sci. Univ. Brno 412* (1960), 137–142.
24. THOMPSON, C.D. Area–time complexity for VLSI. In *Proc. 11th ACM Symp. on Theory of Computing* (Atlanta, Ga., May 1979), ACM, New York, pp. 81–88.
25. VALIANT, L.G. Universality considerations in VLSI circuits. *IEEE Trans. Comput., C-30* (1981), 135–140.

RECEIVED JUNE 1981; REVISED OCTOBER 1982; ACCEPTED NOVEMBER 1982