COST TRADEOFFS IN GRAPH EMBEDDINGS, WITH APPLICATIONS
Preliminary Version

Hong, Jia-Wei[1]* Kurt Mehlhorn[2] Arnold L. Rosenberg[3]*

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 <2 must, infinitely often, have dilation-cost $\geq$ (const)log log log n. The second result provides a stronger but less easily stated example of the same type of tradeoff. 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 <2, then these embeddings must, infinitely often, have dilation-cost $\geq$ (const)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.

1. 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:* the maximum distance in H between the images of adjacent vertices of G, and the *expansion-cost:* the ratio of the size (i.e., number of vertices) of H to the size of G. Typical 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 [7,12,16,18], of finding efficient layouts of circuits on chips [22,23], of finding efficient structured versions of programs [12], 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 [5,9,13,18]; and we are beginning to amass a body of basic

[1] Peking Municipal Computing Centre, Peking, China
[2] Fach. 10: Angewandte Mathematik und Informatik, Universität des Saarlandes, Saarbrücken, Federal Republic of Germany
[3] Mathematical Sciences Department, IBM Research Center, Yorktown Heights, NY, USA
* The research of these authors was done while visiting the Department of Computer Science, University of Toronto, Toronto, Ontario, Canada.

knowledge about embeddings, that can be applied to many "real-life" problem areas. This paper continues in the footsteps of these last-cited papers. In this paper we study cost tradeoffs in graph embeddings.

Cost tradeoffs occur in a variety of computational situations: Cobham [6] discovered a time-space tradeoff for palindrome recognition; Borodin et al. [2] found a similar tradeoff for sorting; Thompson [22] discovered an area-time tradeoff for implementing the DFT in VLSI; and Hong [8] found a tradeoff involving reversal and space complexity for Turing machines. It is not *a priori* obvious that cost tradeoffs occur in the simple world of graph embeddings; in fact, they can be found easily.

Consider embedding the n-vertex *line-graph* $L(n)$ [whose vertex-set is $[n] =_{\text{def}} \{1,2,\ldots,n\}$ 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) [18] *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.*

Part (b) is proved by showing that the root of the tree is a bottleneck when one tries to "snake" the line into the tree: any attempt to lay out the line with dilation-cost 2 gets trapped in a subtree with no way to continue the layout. Sekanina [21] proves that dilation-cost 3 is always attainable.

Now, the tradeoff 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. Are there cost tradeoffs in graph embedding problems where the savings (or penalties) are unbounded in both alternatives?
2. Are there extremal cost tradeoffs, i.e., situations where the choices are between (roughly) maximal dilation and (roughly) maximal expansion?
3. Are there cost-product tradeoffs analogous to the time-space and area-time tradeoffs cited earlier?
4. Are there spectral cost tradeoffs, i.e., situations where there are compromise choices rather than just the choice of optimizing either expansion or dilation?
5. What gives rise to cost tradeoffs in graph embeddings; i.e., are there identifiable structural features of the guest and host graphs that lead to cost tradeoffs?

This paper is devoted to studying the above questions. For several reasons, we concentrate on embeddings of graphs in tree-like graphs. First, many real-life situations that are studied using graph embeddings seek tree-like target graphs since such graphs are often easy to deal with in a computational environment. Second, certain of our results 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 generally a reasonable place to begin studying any graph-theoretic phenomenon.

In Section 2A we answer question 1 in the affirmative. Specifically, 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 $\lambda = \log_3(4/3)$; but for infinitely many h, any expansion-cost-($c<2$) embedding of that ternary tree in a complete binary tree must have dilation-cost $\geq$ (const)log log h.

In Section 2B we answer questions 2 and 3 in the affirmative. We consider embeddings of the side-n pyramid graph in the family of "leap trees", 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 $m=n^{1-\varepsilon}$, 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 tradeoff: for some absolute positive constant c,

$$\log(\text{expansion-cost}) \cdot c^{\text{dilation-cost}} \geq n/2.$$

In Section 3A we answer question 4 in the affirmative. We consider the problem of finding, for each integer n, an efficient *n-generic* binary tree, i.e., a complete binary tree into which all n-node binary trees can be embedded with small dilation-cost and small expansion-cost. We show that generic trees exist, 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 admits roughly optimal embeddings with respect to either cost measure must be very inefficient with respect to the other measure: a generic tree that admits embeddings with dilation-costs 1 requires that expansion-costs be $\Omega(n^{(\log n)/2})$; and infinitely often, a generic tree that admits embeddings with expansion-costs < 2 requires that dilation-costs be $\geq$ (const)log log log n.

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

Section 3B is devoted to an unexpected dividend of the work in Section 3A. Using the compromise (polynomial size, constant dilation) generic tree, we are able to formulate simplified proofs of a number of results in the literature concerning alternating Turing machines [19], the relationship between deterministic and nondeterministic space [20], and the space requirements of context-free parsing [11]. 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.

2. TRADEOFFS IN SIMPLE GRAPH EMBEDDINGS

A. An Unbounded Tradeoff

Our first tradeoff 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.

An *a-ary tree* ($a>0$ an integer) is a graph whose vertex-set is a *prefix-closed* set of strings over the alphabet $[a]$ (the string x is in the set whenever any extension $x\sigma$, $\sigma \in [a]$, is in the set), and whose edges connect vertices x and $x\sigma$ for all strings x and $\sigma \in [a]$. An a-ary tree is *complete* if its vertex-set comprises the set of all strings over $[a]$ of length less than some integer h, which is called the *height* of the tree. $B(h)$ (resp., $T(h)$) denotes the height-h complete binary -- i.e., 2-ary -- (resp., ternary -- i.e., 3-ary) tree.

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 $\lambda = \log_3(4/3)$).* (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 must have dilation-cost $> \alpha\log\log h$.*

Proof Sketch. Part (a) of the theorem is obvious and is left to the reader.

We sketch the proof of Part (b) in some detail. Note that any embedding of a graph G in a complete binary tree partitions G into two sets: 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 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, the cost of the embedding must be $\Omega(\log k)$.

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 they 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 signified.)

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

$$\kappa(l) = \sum_{0\le i\le h-l-1} 3^{-i} e_{l+i}.$$

Let us look at an arbitrary level l of T. There are 3^l copies of $T(h-l)$ -- each having ${}^1/_2(3^{h-l+1}-1)$ nodes -- rooted at this level of T. Since 3^l is odd, at least $\lceil 3^l/2 \rceil$ of the roots of these subtrees belong either to the left part of T or to the right part; say they belong to the left part, call it L. Now, each of the e_{l+i} "split" edges at level $l+i$ of T ($i \in \{0,\dots,h-l-1\}$) can transfer no more than $|T(h-l-i-1)| = {}^1/_2(3^{h-l-i}-1)$ nodes from the set L to the right part $T-L$ of T. Hence the cardinality of L must satisfy

$$|L| \ge {}^1/_2\lceil 3^l/2 \rceil(3^{h-l+1}-1) - \mu(l) \qquad \text{where} \qquad \mu(l) \le \sum_{0\le i\le h-l-1} {}^1/_2(3^{h-l-i}-1)e_{l+i} \le \frac{3^{h-l}}{2}\kappa(l).$$

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 constants $c>0$ such that for infinitely many integers m and $n>0$, $1\le 2^n-3^m \le 3^m/cm$. If m, n is such a pair, then $|T(m-1)| \le |B(n-2)|$, so the ternary tree fits in the binary tree. Now consider what happens when our ternary tree $T(h)$ has height $h = m-1$ and our binary tree $B(h')$ has height $h' = n-2$ for such an m, n pair. If such a $T(h)$ had a level $l_0 < {}^1/_2 \log_3 h$ for which $\kappa(l_0) < 1/4$, then we would have $\mu(l_0) < 3^{h-l_0}/8$ so that

$$|L| > \frac{1}{4}(3^{h+1} + \frac{5}{2}3^{h-l_0} - 3^{l_0} - 1).$$

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

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

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

$$\frac{5}{2}3^{h-l_0}-3^{l_0}-1 \le \frac{3^{h+1}}{c(h+1)}$$

from which it follows that (since $l_0 < {}^1/_2 \log_3 h$)

$$3^{-l_0}-\frac{2}{5}(h^{1/2}+1)3^{-h} \le \frac{6}{5c(h+1)}$$

which, for sufficiently large h, implies that $l_0 > {}^1/_2 \log_3 h$, contrary to assumption. We conclude therefore that $\kappa(l) > 1/4$ for all $l < {}^1/_2 \log_3 h$.

This bound on κ's behavior yields the following estimate for the number of edges of $T(h)$ having one end in L and the other in $T-L$ (which is just the sum of the e_l's):

$$\sum_{0\le l\le h-1} e_l > {}^2/_3 \sum_{0\le l\le h-1} \kappa(l) \ge {}^1/_{12} \log_3 h.$$

Part (b), hence the Theorem, now follows directly from our earlier reasoning. $\square$

B. A Maximal Tradeoff

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. 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 its maximum rate of growth. Some preliminary definitions are needed.

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

$$\{ <i, j> \mid i+j \le n \}$$

and whose edges connect vertices $<i, j>$ and $<k, l>$ just when $|i-k| + |j-l| \le 1$. (See Figure 1.) Note that $P(n)$ has $O(n^2)$ vertices.

The ***height-h leap-tree L(h)*** 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^c1^{m-c} and $2^{c+1}1^{m-c-1}$ (σ^0 = the null string, and $\sigma^{c+1} = \sigma\sigma^c$) for all $c \in \{0, 1, ..., m-1\}$. See Figure 2.

Our maximal tradeoff 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/λ(n)) for any function λ (perforce, λ(x) = O(x/log x)) must have dilation-cost Ω(log λ(n)).* (c) *In particular, any embedding of P(n) in $L(n^\varepsilon)$ for any $\varepsilon<1$, must have dilation-cost Ω(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 Sketch. Part (a) 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 [12,18], one can show that under any embedding of the pyramid $P = P(n)$ in the leap tree $L = L(n/\lambda(n))$, there must be some height-$(h<n/\lambda(n))$ sub-leap-tree L' of L that contains images of between $n^2/4$ and $n^2/2$ vertices of P.

Now, only $h+1$ edges connect L' with the rest of L. Using results in [12, 16], one verifies 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; hence, $\Omega(n)$ 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, some subset S of cardinality $\Omega(n/h) = \Omega(\lambda(n))$ must 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' (since there are 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 $\Omega(\log \lambda(n))$. □

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.

Theorem 2 says the following. At the price of expansion-cost $\Omega(2^n/n^2)$, we can embed $P(n)$ in $L(n)$ and attain minimum (= unit) dilation-cost. If we decrease this expansion-cost by the mth root, by embedding $P(n)$ in $L(n/m)$, then we suffer dilation-cost $\Omega(\log m)$. If we get the expansion-cost down below exponential by embedding $P(n)$ in $L(n^\epsilon)$ for some $\epsilon<1$, then we suffer dilation-cost $\Omega(\log n)$, which is within a constant factor of worst possible. In this last case, we may as well embed $P(n)$ in the smallest leap tree that is big enough to hold it, namely $L(\lceil \log \frac{n(n+1)}{2} \rceil)$; the associated dilation cost is still only $\Omega(\log n)$.

Theorem 2 has in it the hint of a cost-product tradeoff; and such a tradeoff follows from the Theorem.

Corollary. *There is an absolute constant c 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),*

$$\log(\text{expansion-cost}(\epsilon)) \cdot c^{\text{dilation-cost}(\epsilon)} \geq n/2.$$

Proof Sketch. By definition of $P(n)$ and $L(m)$, expansion-cost$(\epsilon) \geq 2^m/n^2$; by Theorem 2(b), there is a constant $d>0$ such that dilation-cost$(\epsilon) \geq d\log n/m$. Now let $c = 2^{1/d}$, and do the required arithmetic. □

C. Conclusions and Open Problems

This completes our study of cost tradeoffs in "standard" graph embeddings. At least two structural characteristics of the source and target families in an embedding can lead to tradeoffs: the tradeoff in the Introduction is due to bottlenecks formed by the roots of the subtrees of trees; the more dramatic tradeoffs in

Theorems 1 and 2 result from mismatches in the *separator* characteristics of the families: in Theorem 1, binary trees are easy to split in half, while ternary trees are hard to split in half; in Theorem 2, pyramids are densely connected in that there are usually many edge-disjoint paths connecting one section of the graph with any other, while trees, even with leap edges or breadth-first edges, are 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.

It would be worthwhile to seek and try to understand cost tradeoffs in embeddings in other families of graphs having strong connections with real-world applications. An obvious candidate is the family of (two-dimensional) grids, which is used to study problems concerning VLSI circuit layout (see, e.g., [1, 23]).

3. TRADEOFFS IN GENERIC GRAPHS

A. Generic Binary Trees

The tradeoffs in the previous sections dealt with a scenario in which one has a family of guest (= source) graphs $\{G\}$ and a family of candidate host (= target) graphs $\{H\}$, and one's task is to find a congenial host H for each guest G. The situation we consider now is different, in that we are now going to measure the congeniality of a 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 Size$(B_n)/n$.) We 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 [17], 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.*)

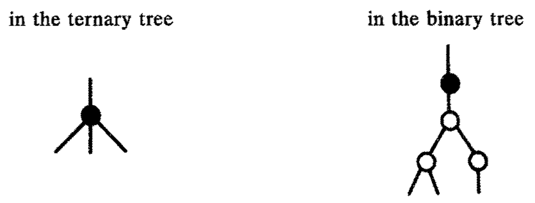
Theorem 3. (a) *There are generic trees B_n into which all n-node binary trees are embeddable with dilation-cost $O(1)$ and expansion-cost $O(n^c)$ for some fixed constant c.* (b) [5] *Any universal binary tree has size $\Omega(n^{(\log n)/2})$, even if the definition of universal is relaxed so the tree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

Thus 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, the other measure must grow insufferably large.

Proof Sketch.

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

Part (c) is proved by means of a variant of the proof of Theorem 1. We "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.



We then emulate the proof of Theorem 1: We note that for infinitely many heights h, the tree $S(h)=\beta(T(h))$ has a number of nodes that is almost a power of 2. Any embedding of such an $S(h)$ in a small binary tree cuts $S(h)$ roughly in half. But $S(h)$ is hard to partition into two pieces of roughly equal sizes: $\Omega(\log h)$ edges must be cut to effect the partition. Hence, the bottleneck argument of Theorem 1 applies.

We turn now to Part (a). Since this result seems to have numerous applications, we describe in detail the recursive construction that proves it.

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 *(= 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* $W = w(1)+w(2)+\ldots+w(n)$.

Proof. It is proved in [15] 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 $\leq 2+\log W/w_i$. We use the 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. We remedy this by augmenting T^* with *inorder edges:* we connect each leaf l of T^* that is a left (right) son to its closest ancestor which is exited via a right (resp., left) link on the path from the root of T^* to l. 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*

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

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

Lemma 1 affords us the basic step in our construction; the remainder 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. 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, ..., 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 -- rooted at v_i's brother. (D_j is empty if v_j has no brother.) View our root-to-leaf path as a line-graph, and weight its nodes as follows.

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

Note the following properties of these vertex-weights:

$n < W =_{\text{def}} \sum_i w(v_i) \leq 2n$;

for $i>0$, $w(v_i) \leq n/2$;

for $i>0$, $W/w(v_i) \geq (w(v_0)+w(v_i))/w(v_i) \geq 3$.

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, ..., 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

the dilation-cost is $\leq 2c$;

the depth of the image of vertex v_i in the host tree is $\leq (3 + \log W/w(v_i))c$;

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.

Say that we have two copies B and B' of the infinite binary tree. Embed B in B' as follows. Each node $v = \alpha_0\alpha_1 \cdots \alpha_m$ ($\in \{1,2\}^*$) of B is mapped onto node $v' = (1\alpha_0)(1\alpha_1)\cdots(1\alpha_m)1$ of B' (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 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, ..., 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:

   dilation-cost $\leq 4c$;

   depth(image(v_i)) $\leq 2c(3+\log W/w(v_i))+1$;

   depth(image(v_0)) $\leq 8c+1$.

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 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 depth of T's root in the host tree is at most $8c+1$.

Second, looking at the dilation-cost of the embedding, there are three cases to consider. Let nodes v_a and v_b be adjacent in T. (a) If both nodes lie on the root-to-leaf path π, then we remarked earlier that 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 by the induction hypothesis, their images are distance at most $8c+3$ apart in the host tree. (c) If one of the nodes is on the path π and the other is in one of the subtrees D_i, then it must be that, say, v_a is one of the v_i on the path, while v_b is the root of D_i. But then the image of v_a has 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

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

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 $2c(3+\log W/\text{w}(v_i))+1$; by our bound on the size of D_i and by the inductive hypothesis, the latter quantity is at most $(10c+1)\log \min(\text{w}(v_i), n/2)$. It follows that

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

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

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

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

$$8c+1+2c\log n-2c\log \text{w}(v_i)+(10c+1)\log \text{w}(v_i) \leq (8c+1)(1+\log n/2)+2c\log n \leq (10c+1)\log n.$$

This completes the proof of Lemma 2. □-Lemma

Part (a) of the theorem follows immediately. 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 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 prune the infinite binary tree at level $(10c+1)\log n$ to yield 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). □

B. 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" [19(b), p. 356]. But this same remark (without the word "computation") describes the content of Theorem 3(a). Indeed, the proof of Theorem 3(a) yields a single general construction that leads to simplified proofs of a number of results concerning 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 one such situation, namely the problem of efficiently simulating space-tree-size efficient alternating Turing machines by time-space efficient ones [19]. The reader is referred also to the problems of parsing (or recognizing) context-free languages [11] and of simulating nondeterministic automata by deterministic ones [20].

Theorem 5.[19] *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 $S(n) \cdot \log Z(n)$ and space $S(n)$.*

We illustrate the use of our generic tree construction by presenting a detailed sketch of the proof of Theorem 5. The reader can easily apply the same proof strategy to the results of [11, 20] and, indeed, to any computational problem desiring bushy trees to replace scrawny ones.

Proof Sketch. We assume familiarity with alternating Turing machines (ATM's, for short); see, e.g., [4, 19]. It follows easily from these sources that we lose no generality by assuming that the ATM's 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, by counting its computational steps and aborting any computation that takes more than $c^{S(n)}$ steps, where the constant c depends on the internal characterisitics of the ATM. (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 labelling 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 labelled with the initial configuration of A on input x;

(b) the leaves of T are labelled by accepting configurations of A;

(c) if v is any nonleaf node of T that is labelled by a universal (resp., existential) configuration of A, then v's descendants include both (resp., one) of the successors of that configuration.

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

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

$$\chi: \text{Nodes}(B) \rightarrow \text{CONFIG} \cup \{\#\}$$

where # is a new symbol, as follows.

$\chi(v) =$ **if** v is the image of a node u of T **then** $\lambda(u)$ **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 χ has the following properties that are crucial to our task.

Lemma 3. (a) *The infinite labelled binary tree $\chi(B)$ 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 $\chi(v)$ is the initial configuration of the ATM A on input x;

ii.) if $\chi(v)$ is a nonaccepting existential (resp., universal) configuration of the ATM A, then there is a node v_1 (resp., and a node v_2) within distance $8c+3$ of v such that $\chi(v_1)$ (resp., and $\chi(v_2)$) is a (resp., are both) successors of $\chi(v)$.

iii.) $\chi(v) = \#$ for all v at depth $> (10c+1)\log Z(n)$.

(b) *Conversely, if χ: Nodes(B) $\rightarrow$ 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 properties of the tree T and the mapping λ. Part (b) is proved by induction, using the fact that the ATM is clocked. Details appear in the full paper. □

Lemma 3 shows that A accepts input x *if and only if* there is a labelling χ of the infinite binary tree B satisfying properties (i)-(iii). We shall now indicate how an ATM of time-complexity $S(n) \cdot Z(n)$ and space-complexity $S(n)$ can, when confronted with input x, check for the existence of such a labelling function by guessing the mapping in pieces and verifying its guesses in parallel.

The *ball in T centered at node v* is 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 χ 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. A^* operates by guessing the contents of a ball (via existential moves), verifying that this new ball is consonant with the previous ball (via deterministic moves), and then splitting to check successors to 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 and 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 labelled with #. In more detail:

```
begin

Initialize, and Check Condition (i) of Lemma 3.
    $v \leftarrow$ root($B$); $d \leftarrow 0$;

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

    if no node $w$ in the current ball has $\chi(w)$ = 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 $\chi(v) \neq \#$ and $\chi(v)$ is a nonaccepting configuration of $A$ and $\chi(v)$ is universal (resp., existential)
      and there are not two (resp., one) nodes in the current ball labelled, by $\chi$, with successors of
      configuration $\chi(v)$
    then stop and reject;

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

      Guess (existentially) the value of $\chi$ 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 $\chi(u) = \#$ 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^* directly implements Lemma 3(a) and so must operate correctly by Lemma 3(b). We need thus only show that A^* operates within the claimed complexity bounds.

Space Bound. At each stage of its computation, A^* has stored (a) at most two overlapping balls, each containing a bounded number of tree nodes, (b) the values of χ on each of these nodes, 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 $\exp aS(n)$ distinct configurations for some constant a, so that an accepting computation tree of A has size at most $\exp\exp aS(n)$. 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 $O(S(n) \cdot \log Z(n))$ steps.

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

Remark. Our simulation algorithm as presented needs to know the tree-size $Z(n)$ and the space $S(n)$. In fact, the result still holds if less is known: 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, $S(n) = 1,2,4,8, \ldots$.

REFERENCES

1. R. Aleliunas and A. L. Rosenberg: On embedding rectangular grids in square grids. IBM Report RC-8404, 1980; submitted for publication.

2. A. Borodin, M. J. Fischer, D. Kirkpatrick, N. A. Lynch, M. Tompa: A time-space tradeoff for sorting on nonoblivious machines. *Proc. 20th FOCS Symp.*, 1979, 319-327.

3. J. W. S. Cassels: *An Introduction to Diophantine Approximation*, Cambridge Tracts in Math. and Math. Physics, No. 45, Cambridge U. Press, Cambridge (1957).

4. A. K. Chandra, D. C. Kozen, L. J. Stockmeyer: Alternation. *J. ACM 28* (1981) 114-133.

5. F. R. K. Chung, D. Coppersmith, R. L. Graham: On trees containing all small trees. Typescript, 1979.

6. A. Cobham: The recognition problem for the set of perfect squares. *Proc. 7th SWAT Symp.*, 1966, 78-87.

7. R. A. DeMillo, S. C. Eisenstat, R. J. Lipton: On small universal data structures and related combinatorial problems. *Proc. Johns Hopkins Conf. on Inf. Sci. and Syst.*, 1978, 408-411.

8. J.-W. Hong: On similarity and duality of computation. *J. CSS*, to appear.

9. J.-W. Hong and A. L. Rosenberg: Graphs that are almost binary trees. Typescript, 1980; submitted for publication; see also *Proc. 13th ACM Symp. on Theory of Computing*, 1981.

10. H. T. Kung and D. Stevenson: A software technique for reducing the routing 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. P. M. Lewis, R. E. Stearns, J. Hartmanis: Memory bounds for recognition of context-free and context-sensitive languages. *Proc. 6th Conf. on Switching Circuit Theory and Logical Design*, 1965, 191-202.

12. R. J. Lipton, S. C. Eisenstat, R. A. DeMillo: Space and time hierarchies for collections of control structures and data structures. *J. ACM 23* (1976) 720-732.

13. R. J. Lipton and R. E. Tarjan: A separator theorem for planar graphs. *SIAM J. Appl. Math. 36* (1979) 177-189.

14. R. J. Lipton and R. E. Tarjan: Applications of a planar separator theorem. *Proc. 18th FOCS Symp.*, 1977, 162-170.

15. K. Mehlhorn: Best possible bounds on the weighted path length of optimum binary search trees. *SIAM J. Comput. 6* (1977) 235-239.

16. A. L. Rosenberg: Encoding data structures in trees. *J. ACM 26* (1979) 668-689.

17. A. L. Rosenberg: Issues in the study of graph embeddings. In *Graph-Theoretic Concepts in Computer Science* (Proc. of the Int'l Workshop WG80), H. Noltemeier, ed. *Lecture Notes in Computer Science, vol. 100*, Springer-Verlag, NY, 1981, to appear.

18. A. L. Rosenberg and L. Snyder: Bounds on the costs of data encodings. *Math. Syst. Th. 12* (1978) 9-39.

19. W. L. Ruzzo: Tree-size bounded alternation. (a) *J. CSS 21* (1980) 218-235; (b) see also preliminary version in *Proc. 11th ACM Symp. on Theory of Computing*, 1979, 352-359.

20. W. J. Savitch: Relationships between nondeterministic and deterministic tape complexities. *J. CSS 4* (1970) 177-192.

21. M. Sekanina: On an ordering of the set of vertices of a connected graph. *Publ. Fac. Sci. Univ. Brno*, No. 412 (1960) 137-142.

22. C. D. Thompson: Area-time complexity for VLSI. *Proc. 11th ACM Symp. on Theory of Computing,* 1979, 81-88.

23. L. G. Valiant: Universality considerations in VLSI circuits. Univ. of Edinburgh Report CSR-54-80, 1980; to appear in *IEEE Trans. Elec. Comp.*.

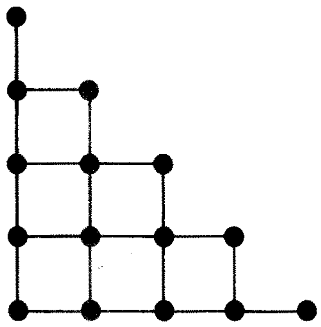


Figure 1. The side-5 pyramid $P(5)$.

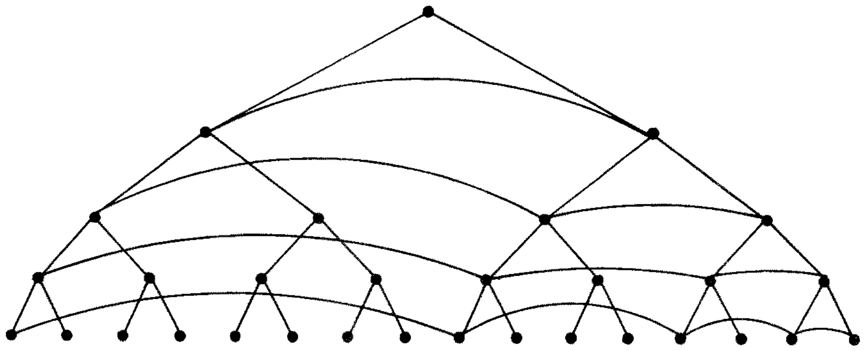


Figure 2. The depth-4 leap tree $L(4)$.