COMBINATORIAL GRAPH EMBEDDING

R.A. DeMillo
S.C. Eisenstat
R.J. Lipton

January, 1980

Final Report: ARO Grant No. DAAG29-76-G-0338
<table>
<thead>
<tr>
<th>Contents</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>A. Space and Time Hierarchies for Classes of Control Structures and Data Structures¹</td>
<td>4</td>
</tr>
<tr>
<td>R. J. Lipton, S.C. Eisenstat, and R.A. DeMillo</td>
<td>4</td>
</tr>
<tr>
<td>B. Space-Time Tradeoffs in Structured Programming: An Improved Combinatorial Embedding Theorem²</td>
<td>17</td>
</tr>
<tr>
<td>R.A. DeMillo, S.C. Eisenstat, and R.J. Lipton</td>
<td>17</td>
</tr>
<tr>
<td>C. An Embedding Result for Labelled Programs³</td>
<td>28</td>
</tr>
<tr>
<td>R.A. DeMillo, S.R. Kosaraju</td>
<td>28</td>
</tr>
<tr>
<td>D. Preserving Average Proximity in Arrays⁴</td>
<td>41</td>
</tr>
<tr>
<td>R.A. DeMillo, S.C. Eisenstat, and R.J. Lipton</td>
<td>41</td>
</tr>
<tr>
<td>E. The Average Length of Paths Embedded in Trees</td>
<td>45</td>
</tr>
<tr>
<td>R.A. DeMillo, R.J. Lipton</td>
<td>45</td>
</tr>
<tr>
<td>F. On Small Universal Data Structures and Related Combinatorial Problems⁵</td>
<td>50</td>
</tr>
<tr>
<td>R.A. DeMillo, S.C. Eisenstat, and R.J. Lipton</td>
<td>50</td>
</tr>
<tr>
<td>G. A Separator Theorem for Planar Graphs⁶</td>
<td>61</td>
</tr>
<tr>
<td>R.J. Lipton and R.E. Tarjan</td>
<td>61</td>
</tr>
<tr>
<td>H. Applications of a Planar Separator Theorem⁷</td>
<td>74</td>
</tr>
<tr>
<td>R.J. Lipton and R.E. Tarjan</td>
<td>74</td>
</tr>
</tbody>
</table>

¹JACM, 23(4)(October, 1976) pp. 720-732

²JACM, 21(1)(Jan, 1980)

³Submitted for Publication

⁴[C.CM, 21(3)(March, 1978) pp. 228-231


⁷1977 FOCS Conference, Providence, R.I., November 1977, pp. 162-170
INTRODUCTION

Let $G, G'$ be directed graphs. A **combinatorial embedding** of $G$ into $G'$ is an identification of each $x \in V(G)$ with a set of vertices $S \subseteq V(G')$ such that each $S$ is bounded in size by a constant independent of $|V(G)|$ and each arc in $G$ is carried into a directed path of length bounded by a constant independent of $|V(G)|$. This concept (first defined in [A]) has formed the basis for a number of theoretical studies supported by ARO Contract No. DAAG29-76-G-0338, and the papers collected here are representative of - with one major exception - the state-of-the-art with regard to graph embeddings.

First, a word regarding the subject matter of these papers. By modelling the control structures of programs as classes of directed graphs, asymptotic properties of control structure transformations can be obtained. This is the principle aim of [A,B,C]. Knuth [1] surveys a number of results concerning control structure transformations and places the graph embedding results in context. Directed graphs also model data storage structures (vertices model nodes or records, arcs model logical adjacencies). The notion of graph embedding can be used to compare the relative storage efficiencies of classes of data structures [D,E,F]. Several researchers have attempted to generalize these results to more encompassing notions of data storage and representation (see e.g., [2,7]) and more sophisticated types of analysis [3]. The purely combinatorial notions involved in data structure embeddings also make contact with a variety of other theoretical and numerical problems [4]. In fact, one of the principle devices used in the results of [A-F] is the notion of "cutting" graphs along **boundaries** of connected regions. A boundary which cuts a graph is called a **separator** and in [G,H] a characterization of separator graphs is derived and used to obtain results in areas from Turing Machine complexity to optimization theory.
Graph Embeddings, boundary arguments and graph theoretical models of computation all appear to be related in sometimes surprising ways [5,6,7,8]. Missing from the collection is a coherent account of these connections. It will have to suffice that the connections run deeper than the surface. We anticipate reporting on this aspect of graph embedding elsewhere.

References

1. D.E. Knuth

2. A.L. Rosenberg

3. A.L. Chow


5. L.G. Valiant

6. G.S. Tseitin

7. A. George

8. S. Cook
Space and Time Hierarchies for Classes of Control Structures and Data Structures

R. J. LIPTON AND S. C. EISENSTAT

Yale University, New Haven, Connecticut

AND

R. A. DEMILLO

University of Wisconsin, Milwaukee, Wisconsin

ABSTRACT: Control structures and data structures are modeled by directed graphs. In the control case nodes represent executable statements and arcs represent possible flow of control; in the data case nodes represent memory locations and arcs represent logical adjacencies in the data structure. Classes of graphs are compared by a relation \( G \leq H \) if \( G \) can be embedded in \( H \) with at most a \( T \)-fold increase in distance between embedded nodes by making at most \( S \) “copies” of any node in \( G \). For both control structures and data structures, \( S \) and \( T \) are interpreted as space and time constants, respectively. Results are presented that establish hierarchies with respect to \( \leq_s \), for (1) data structures, (2) sequential program schemata normal forms, and (3) sequential control structures.

KEY WORDS AND PHRASES: ancestor tree, bounded simulation, complexity, control structure, data structure, directed graph, do forever program, embedding, go to program, label exit program, normal form programs, structured programming, while programs

CR CATEGORIES: 4.22, 4.34, 5.24, 5.25, 5.32

1. Introduction

The running time or computational complexity of a sequential process is usually estimated by summing weights attached to the basic operations from which the process is derived. In practice, however, the complexity of a program is often limited by how efficiently it can access its data structures and control program flow. Furthermore, it has been extensively argued \([4]\) that certain limitations on the process sequencing mechanisms available to the programmer result in more “efficient” representations for the underlying processes. In this paper we examine these issues in an attempt to assess the “power” of various data and control structures.

A key observation about sequential processes is that they usually do not reference
Space and Time Hierarchies

their data randomly. For instance, algorithms that organize their data structures as arrays often access the array elements in a "local" manner (e.g. the conventional matrix multiplication algorithm accesses its arrays by rows and by columns). Thus in a paging environment how one stores an array is especially important (cf. Moler [13], Rosenberg [16]), and it is natural to investigate how arrays can be stored so that elements "near" one another in the array are stored near one another in memory. Data structures are compared by the relation \( LT : \)

For data structures \( G \) and \( G^* \), \( G \leq_{L,T} G^* \) if \( G \) can be embedded in \( G^* \) so that there is at most a \( T \)-fold increase in distance between embedded objects.

It is somewhat unexpected that an analogous study for control structures uses the same basic insights. It is well known that process sequencing disciplines found in programming practice (e.g. go to, while) can simulate each other and are thus equivalent in the sense of yielding functionally equivalent programs, but are inequivalent relative to the stronger requirement of structural isomorphism [1-3, 10, 11]. We argue that the fundamental issue is neither the construction of functionally equivalent programs nor the inability to preserve structure exactly, but rather the "naturalness" of the simulation. Control structures are compared by the relation \( \leq_{S,T} : \)

For algorithms \( G \) and \( G^* \) with distinct process sequencing mechanisms, \( G \leq_{S,T} G^* \) if \( G^* \) simulates \( G \) by making at most \( S \) copies of each operation in \( G \) and increasing the cost of sequential access of embedded operations by a factor of at most \( T \).

Thus comparing the power of data structures and control structures involves analyzing the one-one and many-one aspects of embedding (or simulation) techniques whose efficiency is bounded by \( S \) and \( T \). In a natural way, the relation \( \leq_{S,T} \) represents an intertwining of space and time complexities.

In Section 2 basic combinatorial definitions are presented, and the combinatorial models used for representing data structures and control structures are introduced. In Section 3 the relation \( \leq_{S,T} \) is defined by means of graph embeddings. This relation is viewed as an embedding in the data structure case and as a simulation relation in the control structure case. Section 4 contains the main result for data structure embeddings: For certain families of structures \( \{G_i\}_{i=0}^\infty \) and \( \{G_i^*\}_{i=0}^\infty \), if \( G_i \leq_{1,T} G_i^* \), then \( T = \omega(n_i) \) (see footnote 1) for some positive constant \( c \) whose choice is independent of \( n_i \), the number of components of \( G_i \).

The main theorem in Section 5 generalizes the result in Section 4 by allowing \( S \geq 1 \). In this case, if \( G_i \leq_{1,T} G_i^* \) for certain natural choices of \( \{G_i\}_{i=0}^\infty \) and \( \{G_i^*\}_{i=0}^\infty \), then \( T = \omega(n_i) \), where \( n_i \) is the number of components of \( G_i \) and \( c \) is a positive constant independent of \( n_i \). A direct result of this theorem is that certain schema constructions, such as Engeler normal form [6], cannot be achieved "uniformly" with respect to the \( \leq_{S,T} \) relation. More exactly, for any constants \( S \) and \( T \) there is a go to program \( G \) such that for no program \( H \) in Engeler normal form is \( G \leq_{S,T} H \). Thus, the construction of Engeler normal forms—while always possible—does not preserve time and space in a bounded way. This result also demonstrates how our results will be asymptotic in their nature: For any go to program \( G \) there are \( S \) and \( T \) such that \( G \leq_{S,T} H \) where \( H \) is the Engeler normal form; however, the values of \( S \) and \( T \) must grow with the size of the program \( G \).

In Section 6 the relation \( \leq_{S,T} \) is placed in the context of relations used in previous studies of control structure simulation. The main simulation results for control structures are then developed, giving rise to the hierarchy of control structures shown in Figure 1. An important result is that go to programs are strictly more powerful than label exit programs. Since the class of label exit programs includes many of the standard constructs that are allowed in "structured" programs, this result can be viewed as a precise sense in

1 When we establish results of this form, we are asserting that there is a minimal rate of growth for \( T \) as a function of \( n \). In the sequel we will consistently abuse our notation by writing \( f(S, T) \geq g(n) \) instead of the less convenient \( f(S(n), T(n)) \geq g(n) \). It will usually be clear from context when \( S, T \) are to be considered constants and when \( S \) and \( T \) are parameterized.
which there is a time-space speedup between \texttt{goto} programs and "structured" programs. There are \texttt{goto} programs whose only "structured" counterparts explode in either time or space. This result seems to make precise the comments of Knuth \cite{9} on the efficiency of \texttt{goto} and "structured" programs.

While the results in this paper are motivated by our interest in the power of data and control structures, they may have interest purely as combinatorial results.

2. The Combinatorial Representations

A directed graph $G$ is an ordered pair $(V, E)$ of nodes and arcs. If there is an arc from $x$ to $y$ and an arc from $y$ to $x$, then we say there is an edge between $x$ and $y$. Moreover, the arcs shown in Figure 2(a) are represented as in Figure 2(b). A path from $x$ to $y$ is defined by any sequence of arcs from $x = x_0$ to $x_1$ to $x_2$ . . . $x_n = y$. We define a metric $d_G(x, y)$ on $G$ as the number of arcs in a minimal length path from $x$ and $y$.

A binary tree $^2$ is a finite set of nodes that either is a single node or consists of a root $x$ and an edge between $x$ and the root of each of two binary trees called the left and right subtrees of the root (cf. Knuth \cite{8}). Note that nodes in a binary tree are connected by edges so that the metric is symmetric. If $G$ is a binary tree, then a node $x$ of $G$ is a leaf of $G$ if $x$ has no sons.

We represent both control structures and data structures by directed graphs. In the control case, the nodes of a graph $G$ represent executable statements and the arcs represent possible flow of control; in the data case, the nodes of the graph represent memory locations and the arcs represent logical adjacencies in the data structure. Thus in either case what is to be modeled is the "difficulty" of accessing nodes: The complexity of a control structure $^3$ is given by the cost of accessing and sequencing noncontrol instructions, while the complexity of a data structure is determined by the cost of accessing successive data elements. Each class of control structure or data structure can be studied in terms of restrictions on what graphs are allowed in that class.

2.1. DATA STRUCTURES. The two classes of data structure we deal with are arrays and ancestor trees.

Arrays. $G_n$ denotes the data structure corresponding to an $n \times n$ array. If the nodes of $G_n$ are indexed by $(i, j)$ where $1 \leq i \leq n$ and $1 \leq j \leq n$, then there is an edge between $(i, j)$ and $(i, j + 1)$, for $1 \leq i \leq n, 1 \leq j < n$, and between $(i, j)$ and $(i + 1, j)$, for $1 \leq i < n, 1 \leq j \leq n$.

Ancestor trees. Ancestor trees are binary trees with an additional feature: A node $x$ of an ancestor tree may be connected by an arc to any of its ancestors. For example, the graph shown in Figure 4 is an ancestor tree because $y$ is both an ancestor and a successor of $x$. Some care must be exercised in viewing control structures that are represented in this way; our representations do not always correspond to (temporal) flow of control and are not to be looked at as flowcharts. Rather, what is being modeled is the potential control connectivity of an underlying algorithm or process.

\footnote{1}{If $x$ is a root with subtree $H$ and $y$ is a root of one of the subtrees of $x$, then $y$ is a son of $x$; further, $x$ is an ancestor of each node in $H$, while each node in $H$ is a successor of $x$.}

\footnote{2}{3}
We view $x \rightarrow y$ as meaning that statement $x$ can "push" into a substructure with first statement $y$; $x \Rightarrow y$ as meaning that statement $x$ is "sequentially" followed by $y$; and $x \Rightarrow y$ as meaning that statement $x$ can "exit" some structure and return to statement $y$.

A program is a **while** program provided it is an ancestor tree that satisfies: $y \rightarrow x$ implies $\exists y_1, \ldots, y_k$ such that $x \rightarrow y_1 \rightarrow y_2 \cdots \rightarrow y_k = y$, where $y$ is a leaf and no $y_i$ for $i < k$ has an ancestor pointer (see Figure 6(a)). The last restriction reflects the fact that only the last statement in a **while** loop is allowed to exit the loop.

A program is a **do forever** program provided it is an ancestor tree that satisfies: $y \leftarrow x$ implies $\exists y_1, \ldots, y_k = y$ such that $x \leftarrow y_1 \leftarrow y_2 \leftarrow \cdots \leftarrow y_k$, where each $y_i$ can have ancestor pointers only to $x$ (see Figure 6(b)). The key distinction between **while** programs and **do forever** programs is that in a **do forever** program all statements in a loop can potentially exit immediately out of the looping structure. Clearly, **do forever** programs correspond to the $\Omega_n (n \geq 1)$ structures of Böhm and Jacopini [2].

Finally, a label **exit** program is any program that is also an ancestor tree. Essentially label **exit** programs allow any jumping out of substructures as long as the return is always to an ancestor. The class of label **exit** programs is therefore quite extensive and includes many types of so-called "structured" programs (cf. Peterson et al. [15]). For example, all label **exit** programs are reducible in the sense of [7]; moreover, they correspond to programs in Engeler normal form [6].

**Example.** The following program contains label **exit**, **do forever**, and **while** control structures; its representation using the conventions outlined above is shown in Figure 7.

```
L: S;
   while B, do
   begin S2;
   do forever
   begin S2; exit L;
   do forever begin S3; exit S2; end
   S2;
   end;
S2; □
```

3. $S, T$ Bounded Embeddings

The following definition is fundamental to what follows. Let $G = (V, E)$ and $G^* = (V^*, E^*)$ be directed graphs with associated metrics $d_G$ and $d_{G^*}$. We say that $G^*$ can simulate $G$ (or $G$ can be embedded into $G^*$) with space constant $S$ and time constant $T$, written $G \preceq_{S,T} G^*$, if there is a mapping (called an embedding) $\Phi : V^* \rightarrow V \cup \{A\}$ of the nodes
of $G^*$ to the nodes of $G$ and a special node $A$, so that:

(1) $\forall v^* \in V^*$ with $\Phi(v^*) \neq A$,
    $\forall w \in V$ such that $d_G(\Phi(v^*), w) < \infty,$
    $\exists w^* \in V^*$ such that $\Phi(w^*) = w$ and $d_G(v^*, w^*) \leq T \cdot d_G(\Phi(v^*), w);$  
(2) $\forall v \in V$, $0 < |\Phi^{-1}(v)| = |\{v^* \in V^* : \Phi(v^*) = v\}| \leq S.$

If $\Phi$ is an embedding and $\Phi(v^*) = A$, then we refer to $v^*$ as a bookkeeping node. If $\Phi(v^*) = v \neq A$, then $v^*$ is said to be a copy of $v$. If $S = 1$, we often write $\leq_T$ instead of $\leq_{I_T}$.

Condition (1) states that when $G$ and $G^*$ are control structures (or data structures) simulation involves at most a $T$-fold increase in the cost of statement sequencing (or data element accessing), i.e. the embedding induces at most a $T$-fold increase in path length. Condition (2) states that there are at most $S$ copies of any $v \in V$ in $G^*$. Note that although $G \leq_{S,T} G^*$ may hold between data structures $G$ and $G^*$ when $S > 1$, it is unlikely that such a simulation would be of value (e.g. if an array is being stored as a list structure with multiple copies of array elements, then selective updating of the array may involve multiple updating of list nodes). For control structures, however, simulations with $S > 1$ are frequently used and are quite natural; this is sometimes called node splitting.

Example. Consider the flow diagrams shown in Figure 8. Figure 8(b) is the result of applying a standard "restructuring" algorithm [1] to Figure 8(a) to remove the multiple exit loop $x_3$, $x_4$, $x_5$. Viewing both diagrams as directed graphs, the graph in Figure 8(b) is a 2,2 simulation of Figure 8(a) by defining $\Phi$ as follows:

$$\Phi(x^*_1) = x_1, \quad \Phi(x^*_2) = x_2, \quad \Phi(x^*_3) = (x^*_4) = \Lambda, \quad \Phi(x^*_5) = \Phi(x^*_6) = x_7, \quad \Phi(x^*_7) = x_3, \quad \Phi(x^*_8) = \Phi(x^*_9) = x_8, \quad \Phi(x^*_9) = x_4, \quad \Phi(x^*_10) = x_9.$$

4. Data Structure Embeddings

In this section we present our main result for data structures, settling negatively the question whether arrays can be stored as arbitrary lists with linear bounds on proximity and determining a nontrivial lower bound on the growth rate of $T$ as a function of $n$ for an $n \times n$ array. This result generalizes a result of Rosenberg [16] showing that arrays
cannot be stored in linear-memory with only bounded loss of proximity. But since the arguments are fundamentally different, it is interesting to compare the two proofs. Recall that Rosenberg's arguments are essentially "volumetric": The number of neighbors within distance $n$ of a node in an array can be quadratic in $n$, while a node in a linear list can have at most $2n$ such neighbors. A volumetric argument then demonstrates that arrays cannot be stored in a system with such linear neighborhood structure with only bounded loss of proximity. In contrast, such methods do not seem to apply to our problems; e.g. a node in a binary tree can have more than $2^n$ neighbors within distance $n$.

To obtain our result we need a series of lemmas. Let $G = (V, E)$ be a directed graph with associated metric $d_G$ and suppose $A \subseteq V$. We define the boundary of $A$ as follows:

$$\partial(A) = \{y \in A : \exists x \in A \text{ such that } d_G(x, y) = 1\}.$$ 

In other words, $\partial(A)$ is the set of nodes in $A$ reachable from some node not in $A$ by an arc of $G$.

**Lemma 4.1.** Let $G_n = (V_n, E)$ be an $n \times n$ array and suppose that $A \subseteq V_n$ is such that $|A| \leq \frac{1}{2}n^2$. Then $|A| \leq 2|\partial(A)|$.

**Proof.** We assume $|A| > 0$, since otherwise the lemma is trivially true, and let $A_1, \ldots, A_n$ be the columns of $A$; that is, if $\{(1, i), (2, i), \ldots, (n, i)\}$ is the $i$th column of $G_n$, then $A_i$ is that subset of the column that is included in $A$. Let $k$ be the number of columns $A_i$ such that $|A_i| < n$, and let $l = k$ be the number of columns $A_i$ with $0 < |A_i| < n$. Since $|A| \leq \frac{1}{2}n^2$, it follows that $(n - k)n \leq \frac{1}{4}n^2$ and hence

$$k \geq \frac{1}{4}n.$$  

Notice that if $0 < |A_i| < n$, then at least one node in $A_i$ is adjacent to a node not in $A$ and thus contributes at least one node to $\partial(A)$; therefore

$$|\partial(A)| \geq l.$$  

Suppose that $|A_{i_0}| = 0$ for some $i_0$, $1 \leq i_0 \leq n$. We then claim
To show this, let $A_j$ be maximal in size and assume $i_0 < j$, the case $j < i_0$ being handled symmetrically. Select any row $r$ of $G$, such that $(r, j) \in A_j$. Now, $(r, i_0) \notin A$ by assumption, but some one or more of $(r, i_0 + 1), \ldots, (r, j)$ is in $A$. Therefore each row $r$ of $G$ for which $(r, j) \in A_j$ contributes at least one node to $\partial(A)$, which establishes (3).

To complete the proof of the lemma we consider two cases.

I. No $A_i$ is empty. In this case, $l = k$, and by combining (1) and (2),

$$2 \mid \partial(A) \mid \geq 2^k = 2k^2 \geq \ln^2 n \geq |A|.$$ 

II. Some $A_i$ is empty. Let $c_1, \ldots, c_m$ denote the cardinalities of the nonempty columns $A_i$. If some $c_m = n$, then the result follows directly from (3). If not, then $m = l$ and $c_1 + \cdots + c_m = |A|$, so that $\max_j |A_j| \geq |A| / l$. By (2) and (3) it follows that $2 \mid \partial(A) \mid \geq l + \frac{|A|}{l}$. The lemma is now immediate by calculation.

**LEMMA 4.2.** Let $G_n = (V_n, E)$ and suppose $x, y \in V_n$ then $d_{G_n}(x, y) \leq 2n$.

**PROOF.** This is an elementary property of arrays.

Lemmas 4.1 and 4.2 and the fact that $|V_n| = n^2$ summarize the basic properties of arrays that will be used in the proof of our main result.

**LEMMA 4.3.** Let $H = (V, E)$ be an ancestor tree and let $H_0 = (V_0, E_0)$ be a subtree of $H$. If $x \in V_0$ and $y \in V - V_0$, then $d_H(x, y)$ is greater than or equal to the depth of $x$ in $H_0$.

**PROOF.** Since $y \in V_0$, any path from $y$ to $x$ must pass through the root of $H_0$.

**LEMMA 4.4.** Let $H^* = (V^*, E^*)$ be an ancestor tree; let $H_0^* = (V_0^*, E_0^*)$ be a subtree of $H^*$; and let $A = \Phi(V_0^*) - \{A\}$. If $G_n \simeq_T H^*$ and $|A| \leq \ln^2 n$, then $T \geq \frac{1}{2} \log |A| - 1$; in other words, $|A| \leq 2^{2T+1}$.

**PROOF.** Assume that $|\partial(A)| > 2^T$. Since the root of $H_0^*$ has at most $2^T$ descendants of depth less than $T + 1$, there is a node $x^* \in V_0^*$ of depth greater than or equal to $T + 1$ in $H_0^*$ such that $\Phi(x^*) \in \partial(A)$. Since $\Phi(x^*) \in \partial(A)$, there is a $y \in V_n - A$ with $d_{G_n}(y, \Phi(x^*)) \leq 1$. Now there exists a $y^*$ such that $d_{H^*}(y^*, x^*) \leq T$, by the definition of $\simeq_T$. Since $y \in A$ it follows that $y^* \in V_n^*$. But by Lemma 4.3, $d_{H^*}(y^*, x^*) \geq T + 1$, which is a contradiction. Therefore, $|\partial(A)| \leq 2^T$ and by Lemma 4.1 $|A| \leq |\partial(A)| \leq 2^{2T+1}$.

**THEOREM 4.5.** Let $H^* = (V^*, E^*)$ be an ancestor tree. If $G_n \simeq_T H^*$, then $T \geq \frac{1}{2} \log n - \frac{1}{2}$.

**PROOF.** Assume $G_n \simeq_T H^*$ and for any subtree $H_i = (V_i^*, E_i^*)$ of $H^*$ let $A_i = \Phi(V_i^*) - \{A\}$. Let $H_1$ and $H_2$ be subtrees of some node in $H^*$. Either $|A_1| \leq \ln^2 n$ or $|A_2| \leq \ln^2 n$, since $\Phi$ is 1-1. Using this fact, we may assume that $H^*$ is of the form shown in Figure 9, where $|A_i| \leq \ln^2 n$ for $1 \leq i \leq k$. (We have suppressed explicit representation of ancestral links.) Without loss of generality we assume always that the "smaller" subtree is on the right. By Lemma 4.4, $|A_i| \leq 2^{2T+1}$ for all $i$.

Let $i$ be the smallest integer such that $|A_i| \neq 0$, and let $j$ be the largest such integer. Then $\sum_{i=1}^j |A_i| \leq (j - i + 1)2^{2T+1}$. Since $|V_n| = n^2$,
Space and Time Hierarchies

must exist. Then $|\Psi(R^*_i)| < 1/4 n^2$ for $i < j \leq p$, and since $|\Psi(V^*_i)| = |\Psi(L^*_i) \cup \Psi(x^*_i)|$, we have

$$|\Psi(V^*_i)| \leq 1 + |\Psi(L^*_i)| \leq 1 + |\Psi(R^*_i)| < 1 + 1/2 n^2.$$  

Thus $H^*_i$ is small for $i < j \leq p$. But this implies $\Psi(R^*_i) \subseteq \bigcup_{U \cap \psi \neq 0} \Psi(V^*_i) \subseteq D_p$. By our choice of $i$, however, we conclude that $|D_p| \geq |\Psi(R^*_i)| \geq 1/4 n^2$, establishing our claim. $\square$

We now introduce a variant of the concept of boundary. If $A$ is a set of nodes of $G_n$, then the coboundary of $A$ is defined by

$$\tilde{\delta}(A) = \{y \in A: \text{there exists } x \in A \text{ such that } d_G(x, y) = 1\} = \tilde{\delta}(V_n - A).$$

In other words, $\tilde{\delta}(A)$ is the set of nodes not in $A$ reachable from some node in $A$ in one step. The proof of the following result is similar to that of Lemma 4.1 and is omitted.

**Lemma 5.3.** Let $A$ be a set of nodes of $G_n$ with $|A| \leq 1/4 n^2$. Then $|A| \leq 2 |\tilde{\delta}(A)|^2$. Let $k$ satisfy Lemma 5.2. By Lemma 5.3, $|\tilde{\delta}(D_k)| \geq |D_k|^{1/2}/2 \geq n/2^{1/2}$. Now let $\ell = \lfloor |H^*_i| : H^*_i \text{ is large} \rfloor$, the number of large subtrees. Since at most $S$ copies of any node in $G_n$ appear in $H^*$,

$$\ln^2/4 \leq \sum_{H^*_i \text{ large}} |\Psi(V^*_i)| \leq S n^2.$$  

Hence $\ell \leq 4S$.

In order to complete the proof we proceed as follows. We have already shown that $|\tilde{\delta}(D_k)| \geq n/2^{1/2}$; we show next that this implies that there are too many paths into the large trees $H^*_i$ from the small trees for $S$ and $T$ to be bounded.

Let $Q_T = \{v^* \in V^*_i : H^*_i \text{ is large and there exist } x^* \in V^*_i \text{ such that } d_{H^*_i}(x^*, v^*) \leq T\}$.  

In other words, $Q_T$ is the set of nodes in large subtrees $H^*_i$ that can be reached from some node in some small subtree $H^*_i$ in at most $T$ steps. We define a one-to-one mapping $g$ from $\tilde{\delta}(D_k)$ into $Q_T$ as follows. Select some $y \in \tilde{\delta}(D_k)$. Then $y \in D_k$ and, for some $x \in D_k$, $d_G(x, y) \leq 1$. Let $x^*$ be a copy of $x$ in some small $H^*_i$. Such a copy exists by the definition of $D_k$. Since $G_{s, t, T, \ell}$, there is a copy $y^*$ of $y$ such that $d_{H^*_i}(x^*, y^*) \leq T$. Now $y^*$ is not in any small $H^*_i$ since $y \in D_k$. Thus we can define $g(y) = y^*$, and $g$ is indeed a mapping from $\tilde{\delta}(D_k)$ to $Q_T$. In order to see that $g$ is one-one, we note that for any $y \in \tilde{\delta}(D_k)$, $\Phi(y^*) = \Phi(y^*) = y$; hence $g$ is one-one, so that $|Q_T| \geq |\tilde{\delta}(D_k)|$.

Thus we have, on the one hand, that $|Q_T| \geq |\tilde{\delta}(D_k)| \geq n/2^{1/2}$ and, on the other hand, that

$$|Q_T| \leq |\{H^*_i : H^*_i \text{ is large}\}| \cdot |\{v^* : v^* \in \text{large } H^*_i \text{ within depth } T \text{ of the root of } H^*_i\}| \leq 1 \cdot 2^T \leq 4S \cdot 2^T.$$  

Combining the upper and lower bounds on $|Q_T|$, we deduce that $T + \log S \geq \log n - \log 8/2$. $\square$

As an application of Theorem 5.1, we present the following result. Informally a flowchart is said to be in Engeler normal form if it is represented by a tree augmented by pointers from nodes to ancestors. or nodes at an earlier level but along the same branch. More precisely, a go to program $G$ has an $S, T$ Engeler normal form if $G \leq_{s, t, T} H$ for some ancestor tree $H$.

**Corollary 5.4.** (1) If $G_n$ has an $S, T$ Engeler normal form and $T$ is fixed, then $S \geq c \cdot n$. (2) If $G_n$ has an $S, T$ Engeler normal form and $S$ is fixed, then $T \geq c \cdot \log n$.

Thus in the worst case, either time or space must be unbounded in the construction of Engeler normal forms.
6. Control Structures

In this section we establish our main results for control structures, using the relation $\preceq_{S,T}$ (see Figure 1). For classes $X$ and $Y$ of control structures, i.e. classes of graph representations of programs constructed using only control structures from the indicated restricted class of control structures, we say that $X$ is more powerful than $Y$ when there exist constants $S'$, $T'$ such that (1) for all $H \in Y$ there exist $G \in X$ such that $H \preceq_{S',T'} G$, but for no constants $S$, $T$ is it true that (2) for all $G \in X$ there exists $H \in Y$ such that $G \preceq_{S,T} H$.

Since for the hierarchy of Figure 1 if $X$ is more powerful than $Y$, then the control structures in $Y$ are restrictions of the control structures in $X$, condition (1) is trivially satisfied with $S' = T' = 1$. It is, of course, the results that establish condition (2) that have the greatest novelty.

To place our results in historical perspective, we follow Ledgard [12] in distinguishing the following extremes in simulations among control structures:

(1) $G$ is functionally simulated by $H$ (written $G \preceq_f H$) if, under identical interpretations, $G$ and $H$ compute the same function.

(2) $G$ is very strongly simulated by $H$ (written $G \preceq_{vS} H$) if $G \preceq_{1.1} H$ and if $\Phi$ is an embedding inducing $\preceq_{1.1}$, then the domain of $\Phi$ and the range of $\Phi$ are identical sets.

In [2] it is shown that for each go to program $G$ there exists a while program $H$ such that $G \preceq_f H$, while in [10] it is shown that for some go to program $G$ there does not exist a while program $H$ such that $G \preceq_{vS} H$. Several other notions of simulation intermediate to $\preceq_f$ and $\preceq_{vS}$ have also been used to study the relative power of classes of control structures [1, 3, 11, 15].

The connection between our relation $\preceq_{S,T}$ and these relations is:

(1) $\preceq_{S,T}$ is weaker than $\preceq_{vS}$, since we allow both space and time to increase and do not require $\Phi$ to have identical range and domain;

(2) $\preceq_{S,T}$ is stronger than $\preceq_f$, since we require that paths be preserved in a weak sense;

(3) $\preceq_{S,T}$ deals only with combinatorial aspects of program structure, and thus we make no assumptions about adding program variables or extra predicates (as were made for example in [1, 2, 11]).

We thus claim that the hierarchy theorems presented in this section span the relations used in previous studies. For the remainder of this section we adopt the notation $X \preceq Y$ to indicate that $Y \subseteq X$ but the graphs in $X$ are not uniformly simulated by the graphs in $Y$, i.e. $X$ is more powerful than $Y$.

We will make use of the following definitions. For any directed graph $G$ let

\[ N^c_{in}(l, x) = \{y : d_G(y, x) \leq l\} \quad N^c_{out}(l, x) = \{y : d_G(x, y) \leq l\}. \]

**Lemma 6.1.** Suppose that $G \preceq_{S,T} G^*$ and let $x$ be a node in $G$. Then (1) for any copy $x^*$ of $x$, $N^c_{out}(l, x) \subseteq N^c_{out}(Tl, x^*)$; and (2) for some copy $x^*$ of $x$, $N^c_{in}(1, x) \subseteq S \cdot N^c_{in}(T, x^*)$.

The proofs of both (1) and (2) follow easily from the definition of $\preceq_{S,T}$ and are left to the reader.

**Theorem 6.2.** do forever $\preceq$ while.

**Proof.** Let $S$, $T$ be such that for all do forever programs $G$, there is a while program $H$ for which $G \preceq_{S,T} H$. By part (2) of Lemma 6.1, for any node $x$ in $G$ there exists a copy $x^*$ of $x$ such that $N^c_{in}(1, x) \subseteq S \cdot N^c_{in}(T, x^*)$. But since $H$ is a while program, nodes in $H$ have at most one ancestor pointer to them, so that $N^c_{in}(T, x^*) \subseteq 2^T$. Thus $N^c_{in}(1, x) \subseteq 2^S$ for any do forever program $G$ and any node $x$ in $G$. This is a contradiction, since the number of ancestor pointers to nodes in do forever programs can be unbounded. \[\square\]

**Theorem 6.3.** label exit $\preceq$ do forever.

**Proof.** Let $S$, $T$ be such that for any label exit program $G$ there exists a do forever program $H$ such that $G \preceq_{S,T} H$. Consider the label exit graph $G^{\omega}$ defined as follows:

(1) $V^{\omega} = \{x_1, \ldots, x_n\}$, (2) $x_i \rightarrow x_{i+1}$, for all $1 \leq i < n$. (3) $x_n \rightarrow x_i$, for all $1 \leq i < n$.

(See Figure 11.) Then, by construction, $N^c_{in}(1, x_1) = n - 1$. 

R. J. LIPTON, S. C. EISENSTAT, AND R. A. DEMILLO
SPACE-TIME TRADEOFFS IN STRUCTURED PROGRAMMING:
AN IMPROVED COMBINATORIAL EMBEDDING THEOREM

Richard A. DeMillo
Stanley C. Eisenstat†
Richard J. Lipton†

* School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

† Computer Science Department
Yale University
New Haven, CT 06520

These results were announced at the 1976 Johns Hopkins Conference on Information Sciences and Systems. This research was supported in part by the U.S. Army Research Office, Grant Nos. DAHC04-74-G-0179 and DAAG29-76-G-0338; the Office of Naval Research, Grant No. N00014-67-097-0016; and the National Science Foundation, Grant No. DCR-74-12870.
Abstract: Let $G$ and $G^*$ be programs represented by directed graphs. We define a relation $\leq_{S,T}$ between $G$ and $G^*$ that formalizes the notion of $G^*$ simulating $G$ with $S$-fold loss of space efficiency and $T$-fold loss of time efficiency, and prove that if $G \leq_{S,T} G^*$, where $G$ has $n$ statements and $G^*$ is structured, then in the worst case $T + \log_2 \log_2 S \geq \log_2 n + O(\log_2 \log_2 n)$.

Keywords and Phrases: ancestor tree, complexity, control structure, directed graph, embedding

X:\Categories: 4.22, 4.34, 5.24, 5.32
i.e., there are goto programs that can only be simulated by either very slow
or very large structured programs.

In the sequel, we will concentrate on the combinatorial theorem that
achieves these bounds. The programming language significance of the graphs
and relations studied here is discussed extensively in [1].

2. Preliminaries

A directed graph \( G \) is an ordered pair \((V, E)\) of vertices \( V \) and edges
\( E \subseteq V \times V \). A path in \( G \) is an ordered sequence of vertices connected by edges.
For vertices \( x, y \in V \), let \( d_G(x, y) \) denote the length of a minimum length path
from \( x \) to \( y \). If no such path exists, then \( d_G(x, y) = \infty \).

A binary tree is a directed graph that consists of either a single vertex
or a root \( x \) and edges between \( x \) and the root of each of two binary trees called
the left and right subtrees of \( x \). A vertex \( x \) in a binary tree is a leaf if it
has no sons. If \( H = (V, E) \) is a binary tree with root \( r \in V \) and leaf \( \ell \in V \), and
\( P = (x_1, \ldots, x_n) \) is a direct path from \( x_1 = r \) to \( x_n = \ell \), then \( P \) is called a
branch of \( H \). An ancestor tree \( G = (V, E) \) is a directed graph with the following
properties:

1) There exists a subset \( E_0 \subseteq E \) such that \( G_0 = (V, E_0) \) is a binary tree;
2) If \((x, y) \in E - E_0\), then \( y \) is an ancestor of \( x \) in \( G_0 \).

Let \( G_n \) denote the \( n \times n \) rook-connected array of vertices. If the vertices
of \( G_n \) are indexed by \((i, j)\) for \( 1 \leq i, j \leq n \), then, except for the obvious extremal
conventions, there are symmetric edges between \((i, j)\) and \((i, j+1), (i+1, j)\).

For any directed graph \( G = (V, E) \), the notion of boundary makes sense.
Let \( A \subseteq V \). Then the boundary of \( A \) is defined as

\[ \partial(A) = \{ y \in V - A : \exists x \in A \text{ such that } (x, y) \in E \} \]

Clearly, \( \partial(A) \) denotes the set of vertices not in \( A \) which are reachable from \( A \) by
a single edge.†

By a simple improvement of a result from [1], we have the following important property of arrays:

**Lemma I:** (Boundary Lemma) Let \( A \) be a set of vertices of \( G_n \) with \( |A| \leq n^2/2 \). Then

\[
2|A| \leq \beta(A)^2.
\]

3. **Graph Embedding**

The following relation was defined in [1]. Let \( G = (V,E) \) and \( G^* = (V^*,E^*) \) be directed graphs, and let \( S, T > 0 \). Then \( G \leq_{S,T} G^* \) if there is a partial function (called an *embedding*) \( \phi: V^* \rightarrow V \cup \{A\} \), of the nodes of \( G^* \) to the nodes of \( G \) and a special node \( A \), such that

1) \( 0 \leq |\phi^{-1}(x)| \leq S \) for all \( x \in V \);

2) For all \( x^* \in \phi^{-1}(V) \), if \( d_{G^*}(\phi(x^*),y) < \infty \) for some \( y \in V \), then there exists \( y^* \in \phi^{-1}(y) \) such that \( d_{G^*}(x^*,y^*) \leq d_G(\phi(x^*),y) \).

If \( \phi(v^*) = A \), then we refer to \( v^* \) as a *bookkeeping node*. If \( \phi(v^*) = v \times A \), then \( v^* \) is said to be a *copy* of \( v \). Condition (1) states that there are at most \( S \) copies of any \( v \in V \) in \( G^* \). Condition (2) states that the embedding induces at most a \( T \)-fold increase in path length.

**Theorem I:** [1, Theorem 5.2] If \( S(n), T(n) \) are such that \( G_n \leq_{S(n),T(n)} G^* \) for some ancestor tree \( G^* \), then

\[
T(n) + \log_2 S(n) \geq \log_2 n + c_1.
\]

(1)

The right hand side of inequality (1) cannot be improved, since with \( S(n) = 1 \), the construction of [2] shows that

\[
T(n) = 0(\log_2 n)
\]

† The notion of boundary used here corresponds to the coboundary of [1].
is achievable for any n vertex graph. Theorem 1, however, gives only a linear bound on $S(n)$, and it has been conjectured that a non-polynomial lower bound on $S(n)$ exists. In the next section we obtain such a bound.

4. Main Theorem

In this section, we obtain the following improvement of Theorem 1:

**Theorem 2:** If $G^*$ is an ancestor tree and $G \preceq S(n), T(n) \geq G^*$, then

$$T(n) + \log_2 \log_2 S(n) \geq \log_2 n - O(\log_2 \log_2 n).$$

**Proof:** For notational convenience, let us systematically confuse a graph with its set of vertices, so that "$x \in G" and "$x \in V" mean the same thing if $G = (V,E)$.

We assume $G \preceq S, T G^*$ via an embedding $\phi$. For any $A^* \subseteq G^*$, we use $\phi(A^*)$ to denote the set of $x \in G$ which are $\phi$-images of some $x^* \in A^*$. Henceforth, we assume that $G^*$ is a binary tree; it will be obvious as we progress that if $G^*$ contains ancestor edges, then the proof is completely unaffected.

Let $P = (x^*_1, ..., x^*_k)$ be a path of $G^*$. Then $P$ is an admissible path if it is constructed as follows: For each $x^*_i$ ($1 \leq i \leq k$), let $L^*_i$ denote the subtree of $x^*_i$ containing $x^*_{i+1}$, and let $R^*_i$ denote the other subtree of $x^*_i$; then either

a) $\phi(R^*_i) \geq \phi(L^*_i)$

or

b) $\phi(R^*_i) \geq n^2/4$.

Note that the definition of admissible path is more general than that used in [1]. Indeed, it is by proving the existence of many such admissible paths that we obtain our result.

We fix an arbitrary admissible path $P = (x^*_1, ..., x^*_k)$ and define for $i = 1, ..., k$ the subtree $H^*_i = L^*_i \cup \{x^*_i\}$. We shall say that $H^*_i$ is small if $|\phi(H^*_i)| \leq n^2/4$; otherwise $H^*_i$ is said to be large. Let
\[ D_j = \bigcup_{1 \leq i \leq j} \phi(H^*_i); \]

\( H^*_1 \) is small

in particular, \( D_k \) is the set of vertices in \( G_n \) which have copies in some small \( H^*_i \).

**Lemma 3:** For some \( j \),

\[ \frac{n^2}{4} \leq |D_j| \leq \frac{n^2}{2}. \]

**Proof:** We need only show that there exists an integer \( j \) such that \(|D_j| \geq n^2/4\),

since if \( j \) is the least such integer, then (assuming \(|D_0| = 0\))

\[ |D_j| \leq |D_{j-1}| + |\phi(H^*_i)| < \frac{n^2}{4} + \frac{n^2}{4} = n^2/2. \]

We claim that \(|\phi(R^*_1)| \geq n^2/4\). For suppose otherwise, whence \(|\phi(L^*_1)| \leq |\phi(R^*_1)|\)

by the definition of an admissible path. Now

\[ \phi(G^*) = \phi(H^*_1) \cup \phi(R^*_1), \]

so that

\[ n^2 = |\phi(G^*)| \leq |\phi(L^*_1)| + 1 + |\phi(R^*_1)| \leq 2|\phi(R^*_1)| + 1, \]

and thus

\[ |\phi(R^*_1)| \geq n^2/4. \]

Let \( j \) be such that \(|\phi(R^*_j)| = 0\), and let \( i \) be the largest integer such that

\[ |\phi(R^*_i)| \geq n^2/4. \]

Then

\[ |\phi(R^*_\ell)| < n^2/4, \text{ for } \ell = i+1, \ldots, j. \]

Hence,

\[ |\phi(H^*_\ell)| \leq 1 + |\phi(L^*_\ell)| \leq 1 + |\phi(R^*_\ell)| < 1 + n^2/4 \text{ for all } \ell = i+1, \ldots, n. \]

But then each such \( H^*_\ell \) is small, and therefore

\[ \phi(R^*_1) \subseteq \bigcup_{i \leq \ell \leq j} \phi(H^*_\ell) \subseteq D_j. \]

But by the definition of \( j \), \(|D_j| \geq n^2/4. \) \( \square \)
Letting $k$ satisfy Lemma 3, we find that $D_k$ satisfies the hypothesis of the Boundary Lemma, so that

$$|\partial D_k| \geq \sqrt{2} \frac{n}{\sqrt{2}} \geq \frac{n}{\sqrt{2}}$$

Lemma 4: If $l_P$ is the number of large trees $H_i^*$ along an admissible path $P$, then

$$\frac{n}{\sqrt{2}} \leq l_P 2^T.$$ 

Proof: Let

$$Q_T = \{v^* \in H_i^*, \text{ large: for some small } H_j^* \text{ and } x^* \in H_j^*, d_G(x^*, v^*) \leq T\}.$$ 

i.e., $Q_T$ is the set of vertices in large $H_i^*$ which are reachable from some node in a small $H_j^*$ by a path of length at most $T$. We show that $|\partial(D_k)| \leq |Q_T|$ by defining an injection $g: \partial(D_k) \rightarrow Q_T$. For $y \in \partial(D_k)$, choose some $x \in D_k$ adjacent to $y$.

Let $x^*$ be a copy of $x$ in a small $H_j^*$, let $y^*$ be a copy of $y$ such that $d_G(x^*, y^*) \leq T$, and set $g(y) = y^*$. Since $g(y) = \Phi(y^*) = y$, $g$ is one-one. Thus, from (2),

$$|Q_T| \geq |\partial(D_k)| \geq \frac{n}{\sqrt{2}},$$

but

$$|Q_T| \leq |\{H_i^* : H_i^* \text{ large}\}|$$

- $|\{v^* : v^* \in H_i^*, \text{ large; } v^* \text{ within distance } T \text{ of root of } H_i^*\}|$

$$\leq l_P 2^T$$

To complete the proof, we now show that there are at least $2^T$ admissible paths. Since each admissible path corresponds to a distinct leaf† of $G^*$ and $G_n \leq S, T G^*$, we have

$$\frac{n}{\sqrt{2}} 2^{-T} \leq |\Phi^{-1}(V)| \leq S|V| = S^n$$

and the result follows.

† Without loss of generality, we assume that no leaf of $G^*$ is a bookkeeping node.
Lemma 5: There exist at least $2^{\ell_{\min}}$ admissible paths, where $\ell_{\min} = \frac{n}{\sqrt{2}} \cdot 2^{-T}$.

Proof: We prove the result by showing that at least $\ell_{\min}$ independent binary choices must be made to construct an arbitrary admissible path. Consider a partial admissible path $x_1, \ldots, x_k$ (i.e., the initial segment of an admissible path). If only one subtree of $x_k$ is large, then the admissible path can only be extended down that subtree. However, if both subtrees are large, then the admissible path can be extended down either subtree without violating the condition (a-b). By Lemma 4, there are at least $\ell_{\min}$ large subtrees along every admissible path, and, for each such subtree, there is a node in the admissible path with two large subtrees.

By using the modeling strategy detailed in [1], we obtain the following:

Corollary: For each $n$ there is an $n$ statement goto program $Q$ such that for any structured simulation of $Q$ either

1) the simulating program is slower than $Q$ by a factor of $c_1 \log n$, or
2) the simulating program is larger than $Q$ by a factor of $2^{c_2n+c_3}$.

An interesting interpretation of this result as a space-time tradeoff is shown in Figure 1, which illustrates, for fixed $n > 0$,

$$S(T,n) \geq 2^{n/2}$$

For any fixed value $K \leq T \leq c_1 \log n$, limiting the loss of time efficiency in the simulating program, the shaded region of Figure 1 shows the only values of $S,T$ which are achievable.

Acknowledgements: We would like to thank Nancy Lynch, Ronald Rivest, Albert Meyer and Arnold Rosenberg for suggesting that we look for the improved embedding theorem contained in this paper.
AN EMBEDDING RESULT FOR LABELLED PROGRAMS

Richard A. DeMillo*
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

S. Rao Kosaraju**
Department of Electrical Engineering
Johns Hopkins University
Baltimore, MD 21218

* Supported in part by the U.S. Army Research Office, Grant Nos. DAAG29-76-G-0338 and DAHC04-74-G-0179.

** Supported in part by the National Science Foundation, Grant No. DCR75-09904.
INTRODUCTION

There are two natural methods of limiting the use of labels in structured programs: bounding the number of labels that can be referenced by a single statement and bounding the total number of labels which can appear in a program. It is implicit in an argument of [1] in the former case and in the unbounded analog of both cases that a genuine limitation is imposed and power increases with the number of labels. We show here that, in the latter case, programs with differing bounded numbers of labels are provably equivalent in the precise sense of [1,2]. From [3] it is known that suitable restrictions on the notion of equivalence result in provable differences among these constructs; these restrictions, however, rely on the details of program organization. Hereafter, we deal only with combinatorial arguments. Further motivation for the combinatorial properties in the sequel may be found in [1].

PRELIMINARIES

A directed graph G is composed of a set of vertices, V(G), and arcs E(G) ⊆ V(G) × V(G). The arcs (x,y) and (y,x) together form an edge of G. Arcs and edges are represented by directed and undirected arrows, respectively. A path from x to y is a sequence of arcs

\[(x, x_1), (x_1, x_2), \ldots, (x_{n-2}, x_{n-1}), (x_{n-1}, y),\]

and such a path is said to be of length n. We define the distance metric

\[d_G(x, y)\]

to be the minimum of the lengths of all paths from x to y.
A binary tree with root \( x \) is a directed graph that is either a single vertex \( x \) or contains a vertex \( x \) connected by edges to root(s) \( y_1 \) of subtree(s) \( G \), \( i < 2 \). Note that \( d_G \) is symmetric on binary trees. Let \( G \) be a binary tree with root \( x_0 \) and consider the path \( (x_0, x_1), \ldots, (x_{n-1}, x_n) \), where \( x_i \neq x_j \) for \( i \neq j \). We define the following relations on \( G \):

1. \( x_j \) is a descendant of \( x_i \) \((0 \leq i < j \leq n)\)
2. \( x_i \) is an ancestor of \( x_j \) \((0 \leq i < j \leq n)\)
3. \( x_i \) is the father of \( x_{i+1} \) \((0 \leq i < n)\), and we write \( x_i = f_G(x_{i+1}) \)
4. \( x_{i+1} \) is a son of \( x_i \) \((0 \leq i < n)\)
5. \( x_n \) is leaf of \( G \) if \( f_G(y) \not\in x_n \) for all \( y \in V(G) \)

In a binary tree \( G \), the subtree with root \( x \) is denoted by \( G_x \).

An ancestor tree is a directed graph \( G \) whose arcs may be partitioned into two maximal subsets \( E_1, E_2 \) such that \((V(G), E_1)\) is a binary tree and if \((x,y) \in E_2\), then \( y \) is an ancestor of \( x \) in the binary tree \((V(G), E_1)\). Thus, in an ancestor tree a vertex may be connected by an arc to any of its ancestors. We use the special notation \( x \rightarrow_a y \), if \((x,y) \in E_2\). The following terminology is suggested by [1]: if \( y \rightarrow_a x \) then \( x \) is a label and \( y \) is an exit.

We then say that an ancestor tree, \( G \), is a \( k \)-label program if it contains at most \( k \)-labels; we also say that \( G \) is a \( k \)-exit program if it contains at most \( k \)-exits.
SPACE-TIME BOUNDED SIMULATIONS

The following definition is from [1]; it introduces a fundamental mechanism for comparing programs. Let $G$, $G^*$ be directed graphs. We say that $G^*$ simulates $G$ with space dilation $S > 0$ and time dilation $T > 0$, written $G \preceq_{S,T} G^*$ if there is a map (called an embedding of $G$ in $G^*$).

\[ \Phi: V(G^*) \to V(G) \cup \{A\}, A \notin V(G) \cup V(G^*) \]

such that:

1. $\forall u \in V(G)$
   \[ 0 < |\Phi^{-1}(u)| \leq S, \]
   and

2. $\forall v^* \in V(G^*)$ such that $\Phi(v^*) \notin A$

   $\forall w \in V(G)$ such that
   \[ d_G(\Phi(v^*), w) < \infty \]
   \[ \exists w^* \in V(G^*) \text{ such that } \Phi(w^*) = w, \text{ and } d_{G^*}(v^*, w^*) \leq T \cdot d_G(\Phi(v^*), w). \]

If $\Phi$ is an embedding and $\Phi(u^*) = A$, then $u^*$ is said to be a bookkeeping vertex; on the other hand, if $\Phi(u^*) = u \notin A$, then $u^*$ is said to be a copy of $u$.

Clearly, in a simulation of $G$ with space dilation $S$, no vertex of $G$ can have more than $S$ copies in the preimage of the embedding. In the sequel, we will avoid some notational unpleasantness by agreeing that $\lambda_1, \lambda_2, \ldots$ always denote bookkeeping vertices and that $u_1^*, u_2^*, \ldots u_k^*, k \leq S$, always denote copies $u$. 
It is known that for every $S, T > 0$, there is an ancestor tree which cannot be simulated with space and time dilation $S$ and $T$ by any 1-exit program and for every $S, T$ there is a 1-exit program which cannot be simulated with space and time dilation $S$ and $T$ by any 1-label program. This is very suggestive of a hierarchy in the number of labels for the $\leq_{S,T}$ relation, among ancestor trees.

We can now show that such hierarchies collapse. That is, we show that for every $k > 1$, there is a $T > 0$ such that every $k$-label program can be simulated by some 1-label program with space dilation $S = 1$ and time dilation $T = T(k)$. We begin by considering a general embedding procedure which dilates space by $S(k) > 1$, since this result is technically easier.

AN OBSERVATION

Let $G$ be a binary tree with root $x$ and consider a vertex $y$ which is not the father of two vertices. $G$ can be modified by viewing $y$ as the root and inverting the father-son relationship along the path from $x$ to $y$. Obviously, the resulting graph is still a binary tree; we denote this tree by $G^y$.

MAIN SIMULATION RESULT

Let $H$ be an ancestor tree and choose a vertex $x$ of $H$ such that among the descendents of $x$ there is exactly one label $y$. Let $H'$ be obtained from $H$ by replacing the subtree $H_x$ by the graph shown in Figure 1.

In this graph, $K$ is $H_x$ with its vertices subscripted by "1", $L$ is $(H_x - H_y)^f_H(y)$ with its vertices subscripted by "2", and $M$ is $H_y$ with its vertices subscripted by "2". Thus $\alpha$ is a "second copy" of $f_H(y)$. In addition,
each arc $u \xrightarrow{a} y$ or $u \xrightarrow{a} x$ is replaced by $u_1 \xrightarrow{a} \lambda_2$, $u_2 \xrightarrow{a} \lambda_2$, while every $u \xrightarrow{a} v$ with $v \neq x, y$ is replaced by $u_1 \xrightarrow{a} v$ and $u_2 \xrightarrow{a} v$. Then we have $H \leq 2,3 H'$, which may be proved easily by a case analysis of the possible arcs in $H$ and their copies in $H'$. Note further that if $x$ is not a label, then $H'$ has the same number of labels as $H$, while if $x$ is a label, the total number of labels is decreased by one.

We now prove that any $k$-label program ($k \geq 2$) can be simulated by a $(k-1)$-label program. To this end, let $H$ be a $k$-label program, $k \geq 2$. Two cases arise.

Case I. Some vertex $x$ contains exactly one label in each of its subtrees. If the root of either subtree is a label, no transformation is required for that subtree. In all other cases, replace each subtree as above to yield a $k$-label program $H'$, where $H \leq 2,3 H'$ and in $H'$ the sons of $x$ are both labels. Let the sons of $x$ by $y, z \in V(H')$. Now, clearly each arc $u \xrightarrow{a} y$ or $u \xrightarrow{a} z$ can be replaced by an arc $u \xrightarrow{a} x$ at the expense of dilating path lengths by one arc. Hence, this transformation has the effect of replacing the pair of labels $y, z$ by a single label $x$. If $H''$ is the result of such a transformation, then since $d_{H''}(x, y) = d_{H''}(x, z) = 1$, we have $H \leq 2,3 H''$.

Case II. Some label vertex $x$ has exactly one label $y$ as a descendent. The transformation given above when applied to the subtree rooted at $x$ yields a $k-1$ label program $H'$ such that $H \leq 2,3 H'$.

Thus, every $k$-label program is simulated with $S = 2$, $T = 4$ by a $k-1$ label program. We have immediately that every $k$-label program $H$ is simulated by some 1-label program $G$ with $S, T$ independent of $|V(H)|$; more specifically
$S = 2^{k-1}, T = 4^{k-1}$. It is easily seen that for every $S$, $T$ there is a 1-label program which cannot be simulated by any 0-label program (i.e., by a binary tree).

**AN IMPROVEMENT**

The vertex duplication in the construction above is somewhat artificial; it is used only to keep track of "end points" of circuits, and we might try to use some inherent symmetry in the problem to avoid such duplication. In fact, such duplication need never be introduced. That is, we can prove that for every $k$-label program $H$, ($k > 2$) there is a $(k-1)$-label program $H'$ such that $H \leq_{1,4} H'$.

Let $B_n$ denote the regular graph on $n$ vertices with degree 2, shown in Figure 2. If $V(B_n) = \{a_1, \ldots, a_n\}$, then $B_n \leq_{1,3} G_0$; and $B_n \leq_{1,4} G_1$, when $G_0$ and $G_1$ are as shown in Figures 3(a) and 3(b), respectively.

The first simulation is apparently the better of the two, but in fact the simulation $H \leq_{1,4} G_1$ is the one which is to be preferred for the simulation to be described.

Using this transformation, given a tree $H$, as shown in Figure 4(a), we embed $H \leq_{1,4} H^*$ where $H^*$ is as in Figure 4(b). As in the case $B_n \leq_{1,4} G$, we now have $a_1$ and $a_n$ relatively close to each other. For obvious reasons, we call this transformation a folding of $H$.

Now suppose that $H$ is a subtree of a $k$-label program $H_o$ ($k > 2$) that $a_1$ and $a_n$ are both labels, $V(H_1)$ has no other label, and none of $a_2, \ldots, a_{n-1}$ is a label. We then form $H'$ by folding $H$ and replacing each $u \rightarrow a_n$ by $u \rightarrow a_1$. If no such subtree $H$ of $H_o$ exists, then no label is related to any other as either
a descendent or an ancestor. But since each \( \{x,y\} \subseteq V(H_0) \) share a common ancestor (viz. the root of \( H_0 \)) choose any two labels \( x,y \) and let \( z \) be their deepest common ancestor. This identifies subtrees of the form \( H \) with \( a_1 = z \) and \( a_n \in \{x,y\} \). Fold each of these subtrees and replace each arc \( u \rightarrow x \) or \( u \rightarrow y \) by \( u \rightarrow z \). Then, if the resulting ancestor tree is \( H' \), we have \( H_0 \leq 1,4^{4H'} \).

Note that the passage from a \( k \)-label program to a 1-label program still requires \( T = 4^{k-1} \). It is not known if this is (asymptotically) the best possible.
REFERENCES


Figure 1. Modification of $H$
Figure 2. The Graph $B_n$
Figure 3. The Graphs

$G_0$ (upper) and

$G_1$ (lower)
Figure 4. The trees $H$ (upper) and $H^*$ (lower)
1. Introduction

Efficient algorithms often require specific data structures on which to operate. In many of the algorithms of [1], for instance, the running time depends critically on the number of probes of the data structure needed to access a single data item. In practical computation, however, many factors interfere with optimal data organization. Among the most important of these factors is the locality of references in data structures. Since most data structures are arranged linearly in memory, a set of local references to the structures can require accessing of data elements widely separated in memory. This is particularly relevant in paging environments where a page fault may occur during access of elements which are logically adjacent in a data structure.

In [6], Rosenberg examined the problem of storing arrays as a linear structure with bounded loss of proximity between data elements and showed that any such storage scheme must, in the worst case, induce unbounded loss of proximity. In previous papers [2, 4], we considered the proximity-preserving issue in a more general setting. We used graphs to represent data structures as described in [3]: vertices represent data elements or nodes, and edges represent logical adjacencies; if \( G \) and \( H \) are graphs, we write \( G \preceq_r H \) if \( G \) can be “stored” as \( H \) so that no adjacent nodes of \( G \) are more than a distance \( T \) apart in \( H \). Using this model we were able to show that if \( G_n \) is an \( n \times n \) array and \( H \) is any of a very general type of structure (including as subcases linear lists, binary trees, and threaded lists [5]), then \( G_n \preceq^* H \) only if

\[
T(n) \geq (1/3) \log n - 2/3. \tag{1}
\]

In particular, (1) shows that arrays cannot be stored in trees or lists with bounded loss of proximity, extending the result of Rosenberg [6].

The lower bound represented by inequality (1) is a worst-case result. Since algorithms which manipulate data structures tend to “look at” all of the data, it seems natural to investigate the average loss of proximity involved in storing arrays as various list structures. We find a distinction between linear memory and arbitrary lists: arrays can be stored as nonlinear lists with bounded loss of average proximity, but cannot be so stored in linear memory.

2. Graphs and Embeddings

By a graph \( G = (V, E) \) we mean a set \( V \) of vertices and a set \( E \) of unordered pairs of vertices, the edges. Note that all graphs we discuss are undirected, i.e. \( \{x, y\} \in E \iff \{y, x\} \in E \). An edge between two vertices \( x \) and \( y \) is represented

\[
\begin{array}{c}
& x \\
\text{---}
& y
\end{array}
\]
3. Preserving Average Proximity

The relation $\leq^r$ represents a worst-case analysis of proximity-preserving transformations. Since data structures are frequently accessed "uniformly" in that the probability of accessing a particular node to a particular edge is uniformly distributed. Theorem 1 leaves open the question of whether or not there are ways to store arrays as various other structures which, on the average, preserve proximity. To investigate this problem we need an additional definition.

Definition. Let $G = (V, E)$ and $G^* = (V^*, E^*)$ be graphs. We say that $G$ is $A$-average embeddable (written $G \leq^A G^*$) if there is an embedding $\Phi: V \rightarrow V^*$ such that

$$\sum_{(x,y) \in E} d_G(\Phi(x), \Phi(y)) \leq A \cdot |E|.$$ 

Theorem 2. Let $\{G_n\}_{n \geq 1}$ be the class of $n \times n$ arrays. If $G_n \leq^A G^*$ for a line $L$, then

$$A(n) \geq n/12.$$

Proof. Assume $G = (V, E)$, where $L = (x_1, \ldots, x_n)$ is a line. It is clearly sufficient to assume that $m = n^2$ since, if $G_n = (V, E)$, then $G$ also defines an $A(n)$ average embedding of $G_n$ into the line with $x_n$ removed.

Let $D_i = \Phi^{-1}(x_j); 1 \leq j \leq i$ for $1 \leq i \leq n/2$ so that $|D_i| = i \leq n/2$ (see Figure 5). For each $y \in \partial(D_i)$ there is an edge between $y$ and some node not in $D_i$; hence there is a path between $\Phi(y)$ and some node not in $\Phi(D_i)$ which must pass through $x_i$. Since $|D_i| \leq n/2$, by the Boundary Lemma each $\Phi(D_i)$ makes a contribution of at least $(|D_i|/2)^{1/2}$ to

$$\sum_{(x,y) \in E_{x,y}} d_G(\Phi(x), \Phi(y)).$$

For suppose $\Phi^{-1}(x_k)$ and $\Phi^{-1}(x_l)$ ($k < l$) are adjacent in $G$. Then the $l - k + 1$ that this adjacency should add to the sum accrues by $x_k$'s membership in $\Phi(D_k)$, each membership contributing 1 to the sum.

Therefore

$$\sum_{(x,y) \in E} d_G(\Phi(x), \Phi(y)) \geq \frac{1}{\sqrt{2}} \sum_{i=1}^{n/2} |D_i|^{1/2} = \frac{1}{\sqrt{2}} \sum_{i=1}^{n/2} \sqrt{i} \geq \frac{1}{\sqrt{2}} \int_0^{n/2} (\sqrt{x})dx = \frac{n^3}{6}.$$

Hence

$$n^3/6 \geq A(n)|E_n| = A(n)(2n^2 - 2n),$$

which yields $A(n) \geq n/12$. 

An interesting interpretation of this result is that any reasonable sequential method of array storage is asymptotically optimal with respect to proximity. Consider, for instance, an embedding $\Phi$ of $G_n$ into a line $L$ which places nodes of $G_n$ in row-major order; that is, $\Phi(x,y) = x(i-1) + y(i-1).$ With this embedding into $L$,

$$G_n \leq^A \Phi G_n.$$

By Theorem 4.2 of [6], row-major storage is also optimal for worst-case proximity.

In contrast to Theorems 1 and 2, we have the following.

Theorem 3. When $n$ is a power of 2, there is a binary tree $H$ such that $G_n \leq^A H$.

Proof. Let $G_n = (V_n, E_n)$ be given and suppose $n = 2^k$ for some $k$. We shall describe a recursive method of embedding $G_n$ into a complete binary tree. Divide $G_n$ into four $n/2 \times n/2$ subarrays (see Figure 6(a)) and attach the subarrays as leaves of a complete binary tree $H$ as shown in Figure 6(b).
Clearly, 

\[ \sum_{x \in A} d_A(\Phi(x), \Phi(y)) \leq N + \sum_{x \in A} \sum_{y \in A} d_A(\Phi(x), \Phi(y)), \tag{2} \]

where \( N \) is the sum of the lengths of the \( 2n \) paths between the \( 4n - 4 \) nodes along the boundaries of the \( A_y \). By continuing this process recursively, we can suppose that the \( A_y \) of Figure 6(b) are themselves complete binary trees, and therefore we may bound \( N \) in inequality (2) by 

\[ N \leq 2n(2 \log(n^2/4) + 4) = 8n \log n. \]

This leads to the following recurrence for \( f(n) \), the sum of the lengths of paths which correspond under \( \Phi \) to embedded edges of an \( n \times n \) array:

\[ f(1) = 0 \]
\[ f(n) \leq 4f(n/2) + 8n \log n. \]

The solution to this recurrence satisfies

\[ f(n) \leq 16(4^{\log n}) - 8n \log n - 16n. \]

Thus

\[ A = \sum_{x \in A} d_A(\Phi(x), \Phi(y)) / \left| E \right| \]
\[ \leq \frac{f(n)}{2(n^2 - n)} \leq \frac{8n^2 - 4n \log n - 8n}{n^2 - n} \leq 8. \]

For \( n \) not a power of 2, Theorem 3 clearly can be modified to hold with a slightly larger constant.

Even though the number of vertices reachable by a path of length \( k \) grows as \( 2^k \) in a complete binary tree versus \( k^2 \) in an array, it is important to note that not any embedding technique will work in the proof of Theorem 3. Similarly, not every graph in which fewer than \( 2^k \) vertices can be reached by paths of length \( k \) can be embedded by using the recursive decomposition of Theorem 3; the graph must have the property that it can be "cut" into regions with boundaries that are not too large. A basic question to be resolved is whether or not any family of graphs with neighborhoods growing slower than \( 2^k \) are \( A \)-average embeddable in trees for some constant \( A \).

Acknowledgments. We would like to thank Arnold Rosenberg for his thoughtful comments on a draft of this paper and Larry Snyder for several helpful suggestions.

Received March 1976; revised January 1977

References

(x_{i,j}, x_{i+1,j}) \in E(G_n), \text{ and}

(x_{i,j}, x_{i,j+1}) \in E(G_n).

Such graphs are also called rook-connected. A binary tree is as defined in \cite{1,2}; that is, a binary tree \( H \) is a connected acyclic graph with a designated root and ancestor-descendant relation defined so that each \( x \in V(H) \) has at most two immediate descendents.

Let us write \( G \preceq H \) when there is a one-one mapping (called an embedding of \( G \) into \( H \)) \( \phi : V(G) \rightarrow V(H) \), such that for all \((x,y) \in E(G)\),

\[ d_H(\phi(x), \phi(y)) \leq T. \]

As described in \cite{1}, it follows from simple volumetric arguments that for all \( T > 0 \), there exists a binary tree \( H \) such that \( H \npreceq G_n \), for all \( n \geq 1 \). The corresponding intuition for \( G_n \preceq H \) does not hold. It would now seem that since in \( G_n \)

\[ | \{ x \in V(G_n) : d_{G_n}(x,y) \leq k \} | = O(k^2) \quad (1) \]

while in a complete binary tree \( H \)

\[ | \{ x \in V(H) : d_H(x,y) \} | \geq 2^{k-1} \quad (2) \]

that \( G_n \preceq H \) would now be possible for some bounded \( T \). It is therefore somewhat surprising that \( G_n \preceq H \) only if

\[ T \geq \log n - 1.5 \]

(See \cite{1}, for details).
It is still obvious from inspection that neighborhoods in trees can be much more densely growing than neighborhoods in arrays, and therefore by choosing a suitably global measure of loss of proximity, this difference should be distinguishable. In [2] we considered such a measure:

\[ G \preceq^{\text{edge}}_{A} G^* \text{ if for some embedding } \phi : V(G) \rightarrow V(G^*) \]

\[ \sum_{(x,y) \in E(G)} d_{G^*}(\phi(x), \phi(y)) \leq A \mid E(G) \mid. \]

It follows [2] that for \( b = 8.5 \)

\[ G_n \preceq^{\text{edge}}_{b} H \]

for some binary tree \( H \). This upper bound can be improved to \( b = 7 - o(1) \)

The relation \( \preceq^{\text{edge}}_{A} \) may be thought of as averaging - with relative frequencies uniformly distributed to the edges \( E(G) \) - over the edges of \( G \).

We now make a more global definition which finally may be used to recover our original, although imprecise, intuitions about path lengths in binary trees. We will essentially average over shortest paths:

\[ G \preceq^{\text{paths}}_{A} G^* \text{ if one is an embedding } \phi : V(G) \rightarrow V(G^*) \text{ such that} \]

\[ \Gamma_n \preceq A \cdot A_n \]

where

\[ \Gamma_n = \sum_{\phi(x), \phi(y)} d_{G^*}(\phi(x), \phi(y)) \]

\( \dagger \) L. Snyder, private communication.
and

$$\Delta_n = \sum_{x,y} d_{G}(x,y).$$

We then have the following theorem.

**Theorem.** For each $n \geq 0$, let $A_n$ be the least real number such that

$$G_n \leq \text{paths } A_n H,$$

for a binary tree $H$. Then

$$\lim_{n \to \infty} A_n = \lim_{n \to \infty} \Gamma_n / \Delta_n = 0.$$

**Proof** we first show

$$\Delta_n = \Omega(n^5)$$

Let us choose $B_1, B_2 \subseteq V(G_n)$ so that

$$B_1 = \{x_{i,j} : 1 \leq i, j \leq n / 4\}$$

$$B_2 = \{x_{i,j} : \frac{3n}{4} \leq i, j \leq n\}$$

so that $|B_1 \times B_2| = \left[\frac{n^2}{16}\right]^2$. Now clearly, for any $(x,y) \in B_1 \times B_2$

$$d_{G_n}(x,y) \geq n / 2,$$
and hence by definition

\[ \Delta_n > \frac{n^5}{512} = \Omega(n^5) \]

We now obtain the following upper bound for \( \Gamma_n \)

\[ \Gamma_n = O(n^4 \log n). \]

As in [2] let \( A_{ij} \in V(G_n), 1 \leq i, j \leq 2, \quad |A_{ij}| = \frac{n^2}{4}, \)

Denote the \( n/2 \times n/2 \) decomposition of \( G_n \) and notice that

\[ \Gamma(n) \leq 4 \Gamma \left( \frac{n}{2} \right) + \frac{1}{2} n^4 \log n. \]

Thus \( \Gamma(n) \leq \alpha n^4 \log n + \beta n^4 \) from which the theorem follows directly.


ON SMALL UNIVERSAL DATA STRUCTURES
AND RELATED COMBINATORIAL PROBLEMS †
(Preliminary Report)

Richard DeMillo

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Stanley Eisenstat
Richard J. Lipton

Department of Computer Science
Yale University
New Haven, CT 06520

† Work supported in part by U.S. Army Research Office, Grant No. DAAG29-76-
G-0338, and by the Mathematics Research Center, University of Wisconsin-
Madison.
INTRODUCTION

One of the most significant changes in theoretical computer science has been the recent infusion of the methods and problems from combinatorial analysis. Among the most powerful combinatorial theorems which have been imported to computer science are those of extremal graph theory [1]: in extremal graph theory, one is interested in the largest (or in complementary problems, the smallest) graph which avoids (or contains) a given structure. Purely combinatorial results (which have significance, e.g., for the design of circuit boards) have been obtained by Chung and Graham [2] and by Chung, Graham, and Pippenger [3]. In this paper, we extend this theory to encompass results concerning data structures.

As motivation for the results to the described, note that many of the large data structures manipulated by the programs described in [4,5] have two characteristics

(i) they are sequentially accessed, and

(ii) many distinct structures convolve in the same physical memory.

For applications of this sort, it would obviously be desirable to have available a universal data structure in which all data structures from a given class may gracefully reside. In view of (i), by "graceful" we mean that the sequential accessing characteristics of the embedded data structures are not too drastically altered. Let us measure such alterations by the dilation of logical adjacencies [6,7] needed to embed all structures from a given class into a universal structure; this is then a complementary extremal graph theory problem: what is the size (number of edges) of the smallest universal graph for a given dilation factor.
The main results contained in this paper address such problems from a number of points of view.

1. We give several asymptotically optimal universal data structures for graphs of n vertices when average dilation [7] is used as a measure.

2. We discuss a universal data structure for graphs of n vertices where worst-case dilation is used as a measure [6].

3. We consider variations of the average dilation measure which gives favorable comparisons between data structures studied in [6,7].

4. We consider the kinds of "sharing" that can take place between "almost linear" and "almost complete tree-like" structures.

5. Finally, we propose a data structure embedding model which recovers some aspects of random accessing of data items, and prove a space-time tradeoff which seems to indicate that no savings is possible in RAM models which assess accessings costs uniformly [8].

PRELIMINARIES

A graph, G, is defined by its vertices, V(G), and edges, E(G) ⊆ V(G) × V(G). Edges are assumed to be undirected: a pair of vertices x, y are connected if either (x, y) ∈ E(G) or (y, x) ∈ E(G). A path between x₀, xₙ is said to be of length n. The distance metric d₆(x₀, xₙ) is defined to be n if there is no shorter path than x₀,...,xₙ.

A graph represents a data structure in the obvious way: vertices represent nodes or records and connectedness models logical adjacency. The following relations and their significance for data structures can be found in [6,7]. Let G, G* be graphs. We say that G is T-worst case embeddable in G* (G< T G*) if there is a one-one φ: V(G)→V(G*) such that (x,y) ∈ E(G) implies

\[ d_{G*}(φ(x), φ(y)) \leq T. \]
Similarly, $G$ is $A$-average case embeddable in $G^*$ ($G \leq_{avg}^A G^*$) if there is a one-one $\phi$ as above such that

$$\sum_{x,y} d_{G^*}(\phi(x), \phi(y)) \leq A^*|E(G)|.$$  \hspace{1cm} (2)

In $[4,5]$, comparisons between several natural classes of graphs give asymptotic bounds on $T, A$ in (1), (2) as functions of $|V(G)|$. Shortly after the announcement of the results of [6], R. M. Karp suggested to us the following class of problems connected with extremal graph theory: what are the characteristics of $\leq_T$ universal data structures; i.e., those structures which $T$-worst case embed all graphs in a given class. This paper grew out of considering these problems.

**UNIVERSAL GRAPHS**

Let $\zeta^n$ be a given class of graphs $G$, $|V(G)| = n$. Let us ask about a data structure which is $\leq_T$ or $\leq_{avg}^A$ universal for $\zeta^n$. In particular, let us define

$$w(\zeta^n, T) = \min \{|E(G)|: G^n \in \zeta^n, G^n \leq_T G\}$$  \hspace{1cm} (3)

and

$$a(\zeta^n, A) = \min \{|E(G)|: G^n \in \zeta^n, G^n \leq_{avg}^A G\}.$$  

For $T = 1$, (3) becomes the complementary extremal graph problem studied in $[2,3]$.

By an $n$-tree $G$, we mean a connected acyclic graph $G$, with $|V(G)| = n$. It is also convenient to think of trees as rooted in the following sense: accompanying $G$, there is an ancestor-descendant relation that assigns direct ancestors and direct descendents to vertices in the obvious way so that a vertex with no ancestors can be designated as the root of the tree.
(Obviously this choice is not going to be unique, but we assume that $G$ is not characterized until such a choice is made). A d-ary $n$-tree is an $n$-tree in which each vertex has at most $d$ direct descendants. We denote, respectively, the classes of $n$-trees and d-ary $n$-trees by $\Gamma^n$ and $\Gamma^n_d$.

By [2] it is known that $\frac{1}{2}n \log n < w(\Gamma^n, 1) < n^{1+k(n)}$, $k(n) = [\log \log n]^{-1}$.$^+$

The upper bound was improved in [3] to

$$w(\Gamma^n, 1) = \Theta(n \log n \log \log n)$$

The bounds on $a(\Gamma^n, 1)$ are apparently not elsewhere considered.

Superficially, at least, all interest in further characterization of (3) is destroyed by the following obvious

**Theorem.** For $T \geq 2$

$$w(\Gamma^n, T) = n$$

Of course, in (3), the "target" graph $G$ may have unbounded degree. Therefore, it is natural to consider $w(\zeta^n, T, S)$ and $a(\zeta^n, T, S)$ where in both cases the target graph $G$ is restricted to be in the set $S$. Note that now the theorem just cited is no longer obviously true.

---

* Thus $\Gamma^n_2 = \text{binary trees or } n \text{ vertices.}$

$^+$ In the sequel, we use $\log x$ for $\log_2 x$ and $\ln x$ for $\log_e x$. 
The best that is known is the upper bound of [3] (S = all cubic graphs)

\[ w(r^n, 1, S) \leq \frac{2\sqrt{2}}{n} \exp \left(\frac{\log^2 n}{2 \log 2}\right). \]  

(4)

It is not obvious that when (i) "targets" are restricted to binary trees and (ii) \( w(r^n, T, r^n) \) is considered, that it is possible to do any better than the union of all trees in \( r^n \), giving a structure of size \( 4^n / 2^{\pi n} \).

But, we have the following

**Theorem.** For each \( T > 1 \), there is a binary tree \( H \), such that \( G \geq_T H \) for all \( G \in r^n \), and

\[ \ln |E(G)| \leq \frac{2n^2}{\ln n} \]

or in other words

\[ w(r^n, T, r^n) = \exp \frac{1}{\ln n} (\ln n)^2 + O((\ln n)^2). \]

A key step in the proof of this theorem hinges on the solution to the fascinating "almost linear" recurrence

\[ u_n = u_{n-1} + u\left\lfloor \frac{n}{2} \right\rfloor, \]

(5)

first considered by Knuth [9]. This also establishes a connection between the theorem and ineq. (4): \( u_n \) is also the number of partitions of \( 2n \) of the form \( \sum \alpha_i 2^i \), \( \alpha_i = 0, 1 \). Knuth [9] bounds the partition function

\[ P(m) = \frac{1}{4\sqrt{3m}} \exp \left(\pi \frac{2}{3} m\right). \]
There are two possibilities for improving the bounds in $w(r^n_2, T, \Gamma^n_2)$. The first possibility is to introduce circuits to the target graph of the previous theorem, but this does not appear to give an asymptotically better bound than (4). The second possibility is to prove that balanced trees and unbalanced trees are \leq_T - equivalent. This seems unlikely since combining such a result with the proof method of the previous theorem gives a polynomial sized universal tree. However, in trying to improve the bounds on $w(r^n_2, T, \Gamma^n_2)$ it may be desirable to ignore irregular trees, letting only very balanced or very unbalanced trees reside in the same universal data structure.

In any case, it seems unlikely that polynomial structures are possible. We are, however, far from proving this; indeed, the best known lower bound is the following

**Theorem.** For all $n > N$

$$w(r^n_2, T, \Gamma^n_2) > c(T) n \log n ,$$

where $c(T) > 0$ is a constant for fixed $T \geq 1$.

Certain other subcases are also of interest. Erdös, Chung, and Graham\(^+\), consider $w(S,1)$ and obtain

$$w(S,1) \leq \frac{4}{11} n^2 .$$

The following theorem is an improvement, but is surely not the best possible bound.

**Theorem**

$$w(S,1) \leq \frac{2}{9} n^2$$

\(^+\) Private Communication.
A non-trivial lower bound would clearly be desirable. Another class of interest are graphs of high genus.\textsuperscript{++} We conjecture that for graphs of fixed genus $\gamma$, it is possible to do better than the naive $\binom{n}{2}$ bound obtained by embedding in the complete graph.

Our next series of results show impressive improvements by passing to average dilations. We now get optimal constructions, even in a variety of limited settings.

We have, for instance, the

**Theorem.** For $\alpha > 0$,

$$a(\Gamma_2^n, \frac{1}{\alpha}, S) = O(n^{\log(2+\alpha)}) .$$

Since there is a linear lower bound on $a(\cdot, \cdot, \cdot)$, this construction is optimal. By a slight modification of the construction, this gives

$$a(\Gamma_2^n, A, S) = 0(n),$$

for all $A \geq 1$, but this result may be superceded by the following

**Theorem.** For each $A \geq 1$, there is a binary tree $H$, such that

$$G \preceq_{\text{avg}} A H$$

for all $G \in \Gamma_2^n$ and

$$|E(G)| = 0(n) ;$$

or, in other words

$$a(\Gamma_2^n, A, \Gamma_2^n) = 0(n) .$$

\textsuperscript{++} A graph is of genus $\gamma$ if it can be embedded in a sphere with $\gamma$ handles [10].
These results are related to the ability to "cut" graphs in advantageous ways. For example, a generalization of the planar separator theorem [11] to graphs of high genus, obtained by Lipton and Tarjan, gives us the following

**Theorem.** Let $L_n^\gamma$ be the class of graphs $G$ with genus $\gamma$ and $|V(G)| = n$. Then, for all $n > N$,

$$a(L_n^\gamma, A, T_2^n) \leq c(A) \cdot n,$$

where $c(A)$ does not depend on $n$.

**EXTENDED MODEL**

In comparing classes of data structures (see, e.g., [6,7], the measures of "efficiency" have implicitly assumed that only sequential accessing is important. Thus, when in [6], we bound the efficiency, $T$, of an embedding of $n \times n$ array into binary trees by

$$T(n) \geq c \log n,$$

the function $T(n)$ captures the dilation factor in an embedding. We now describe a generalization of this concept which recovers a certain kind of random accessing. Since the precise definitions are quite complex, we will settle for a less exact -- but more picturesque -- rendering. Let us assume that we have in front of us an illustration of a graph $G$, and also a number of friends who agree to lend us their forefingers for use in tracing the paths of the graph. Our friends oblige us as follows: We may start traversing at any vertex already visited. The traversal rule is, then, that we must either traverse graph edges or "jump" to a vertex pointed to
by a friend. The *time* required to traverse a sequence of vertices is then simply the number of applications of traversal rules. Notice that the result of a traversal is not necessarily a path of G. The connection between fingers and random accessing is that traversals requiring k-fingers also require k-"addresses" for the vertices pointed to.

We then say that G ≤ₖ,ₜ G* if there is a one-one ϕ: V(G) → V(G*), so that for every x, y ∈ V(G) with dₕ(x, y) = m, there is a k-finger traversal from ϕ(x) = x* to ϕ(y) = y* with time at most Δ, and Δ ≤ Tₕ(x*, y*).

We have the following

**Theorem.** If Gₙ is the n x n array [7], H is a binary tree and

\[ G_n \leq k, T(n) H, \]

then

\[ k + T(n) \geq c \log n, \]

where c is a constant independent of n.

**OTHER TYPES OF AVERAGE EMBEDDING**

The relation ≤₅ₐᵥₖ may be thought of as *averaging* - with relative frequencies uniformly distributed to the edges E(G) - over the edges of G.

We now make a more global definition which may be used to recover our intuitions about path lengths in binary trees [7]. We will essentially average our shortest paths:

\[ G \leq _{₅ₐᵥₖ} G* \text{ if there is an embedding } \phi: V(G) \rightarrow V(G*) \text{ such that} \]

\[ \sum d_{G*}(\phi(x), \phi(y)) \leq A \cdot \sum d_G(x, y), \]

for \( \phi(x), \phi(y) \).
We then have the following

**Theorem.** For each \( n \geq 0 \), let \( A_n \) be the least real number such that

\[
G_n < \frac{\text{paths}}{A} H,
\]

for a binary tree \( H \). Then

\[
\lim_{n \to \infty} A_n = 0
\]

Thus, we see that if the average embedding is required to work well on all shortest paths, then the embedding cost goes to zero. In a sense, then \( \frac{\text{avg}}{A} \) "charges" more heavily than \( \frac{\text{paths}}{A} \) for any bottlenecks.

**REFERENCES**

A SEPARATOR THEOREM FOR PLANAR GRAPHS

RICHARD J. LIPTON† AND ROBERT ENDRE TARJAN‡

Abstract. Let $G$ be any $n$-vertex planar graph. We prove that the vertices of $G$ can be partitioned into three sets $A$, $B$, $C$ such that no edge joins a vertex in $A$ with a vertex in $B$, neither $A$ nor $B$ contains more than $2n/3$ vertices, and $C$ contains no more than $2\sqrt{2n}$ vertices. We exhibit an algorithm which finds such a partition $A$, $B$, $C$ in $O(n)$ time.

1. Introduction. A useful method for solving many kinds of combinatorial problems is "divide-and-conquer" [1]. In this method the problem of interest is divided into two or more smaller problems. The subproblems are solved by applying the method recursively, and the subproblem solutions are combined to give the solution to the original problem. Three things are necessary for the success and efficiency of divide-and-conquer: (i) the subproblems must be of the same type as the original and independent of each other (in a suitable sense); (ii) the cost of solving the original problem given the solutions to the subproblems must be small; and (iii) the subproblems must be significantly smaller than the original. One way to guarantee that the subproblems are small is to make them all roughly the same size [1].

We wish to study general conditions under which the divide-and-conquer approach is useful. Consider problems which are defined on graphs. Let $S$ be a class of graphs* closed under the subgraph relation (i.e., if $G_1 \in S$ and $G_2$ is a subgraph of $G_1$, then $G_2 \in S$). An $f(n)$-separator theorem for $S$ is a theorem of the following form:

There exist constants $a < 1$, $\beta > 0$ such that if $G$ is any $n$-vertex graph in $S$, the vertices of $G$ can be partitioned into three sets $A$, $B$, $C$ such that no edge joins a vertex in $A$ with a vertex in $B$, neither $A$ nor $B$ contains more than $an$ vertices, and $C$ contains no more than $\beta f(n)$ vertices.

If such a theorem holds for the class of graphs $S$, and if the appropriate vertex partitions $A$, $B$, $C$ can be found fast, then a number of problems defined on graphs in $S$ can be solved efficiently using divide-and-conquer. For a given graph $G$ in $S$, the sets $A$ and $B$ define the subproblems. The cost of combining the subproblem solutions is a function of the size of $C$ (and thus of $f(n)$).

Previously known separator theorems include the following:

(A) Any $n$-vertex binary tree can be separated into two subtrees, each with no more than $2n/3$ vertices, by removing a single edge. For an application of this theorem, see [13].

(B) Any $n$-vertex tree can be divided into two parts, each with no more than $2n/3$ vertices, by removing a single vertex.

(C) A grid graph is any subgraph of the infinite two-dimensional square grid illustrated in Fig. 1. A $\sqrt{n}$-separator theorem holds for the class of grid graphs. For an application, see [5].

* Received by the editors August 10, 1977.
† Computer Science Department, Yale University, New Haven, Connecticut 06520. This research was supported in part by the U.S. Army Research Office under Grant DAAG 29-76-G-0338 and The National Science Foundation under Grant MCS 78-81486.
‡ Computer Science Department, Stanford University, Stanford, California 94305. This research was supported in part by National Science Foundation under Grant MCS-75-22870 and in part by the Office of Naval Research under Contract N00014-76-C-0688.

† The Appendix contains the graph-theoretic definitions used in this paper.
A one-tape Turing machine graph [16] is a graph representing the computation of a one-tape Turing machine. A \( \sqrt{n} \)-separator theorem holds for such graphs. For an application, see [15].

One might conjecture that the class of all suitably sparse graphs has an \( f(n) \)-separator theorem for some \( f(n) = o(n) \). However, the following result of Erdős, Graham and Szemerédi [4] shows that this is not the case.

**Theorem C.** For every \( \epsilon > 0 \) there is a positive constant \( c = c(\epsilon) \) such that almost all\(^2\) graphs \( G \) with \( n = (2 + \epsilon)k \) vertices and \( ck \) edges have the property that after the omission of any \( k \) vertices, a connected component of at least \( k \) vertices remains.

Although sparsity by itself is not enough to give a useful separator theorem, planarity is. In § 2 of this paper we prove that a \( \sqrt{n} \)-separator theorem holds for all planar graphs. In § 3 we provide a linear-time algorithm for finding a vertex partition satisfying the theorem. This algorithm and the divide-and-conquer approach combine to give efficient algorithms for a wide range of problems on planar graphs. Section 4 mentions some of these applications, which we shall discuss more fully in a subsequent paper.

### 2. Separator theorems

To prove our results we need to use three facts about planarity.

**Theorem 1** (Jordan curve theorem [6]). Let \( C \) be any closed curve in the plane. Removal of \( C \) divides the plane into exactly two connected regions, the "inside" and the "outside" of \( C \).

**Theorem 2** [7]. Any \( n \)-vertex planar graph with \( n \geq 3 \) contains no more than \( 3n - 6 \) edges.

\(^2\)By "almost all" we mean that the fraction of graphs possessing the property tends with increasing \( n \) to one.
A SEPARATOR THEOREM

\[ \text{Fig. 2. Kuratowski subgraphs: (a) } K_5; (b) K_{3,3}. \]

**THEOREM 3** (Kuratowski's theorem [12]). A graph is planar if and only if it contains neither a complete graph on five vertices (Fig. 2(a)) nor a complete bipartite graph on two sets of three vertices (Fig. 2(b)) as a generalized subgraph.

From Kuratowski's theorem we can easily obtain the following lemma and its corollary.

**LEMMA 1.** Let \( G \) be any planar graph. Shrinking any edge of \( G \) to a single vertex preserves planarity.

**Proof.** Let \( G^* \) be the shrunken graph, let \( (x_1, x_2) \) be the edge shrunk, and let \( x \) be the vertex corresponding to \( x_1 \) and \( x_2 \) in \( G^* \). If \( G^* \) is not planar then \( G^* \) contains a Kuratowski graph as a generalized subgraph. But this subgraph corresponds to a Kuratowski graph which is a generalized subgraph of \( G \). Figure 3 illustrates the possibilities. \( \square \)

**COROLLARY 1.** Let \( G \) be any planar graph. Shrinking any connected subgraph of \( G \) to a single vertex preserves planarity.

**Proof.** The proof is immediate from Lemma 1 by induction on the number of vertices in the subgraph to be shrunk. \( \square \)

In some applications it is useful to have a result more general than the kind of separator theorem described in the Introduction. We shall therefore consider planar graphs which have nonnegative costs on the vertices. We shall prove that any such graph can be separated into two parts, each with cost no more than two-thirds of the total cost, by removing \( O(\sqrt{n}) \) vertices. The desired separator theorem is the special case of equal-cost vertices.

**LEMMA 2.** Let \( G \) be any planar graph with nonnegative vertex costs summing to no more than one. Suppose \( G \) has a spanning tree of radius \( r \). Then the vertices of \( G \) can be partitioned into three sets \( A, B, C \), such that no edge joins a vertex in \( A \) with a vertex in \( B \), neither \( A \) nor \( B \) has total cost exceeding \( 2/3 \), and \( C \) contains no more than \( 2r + 1 \) vertices, one the root of the tree.
Proof. Assume no vertex has cost exceeding 1/3; otherwise the lemma is true. Embed $G$ in the plane. Make each face a triangle by adding a suitable number of additional edges. Any nontree edge (including each of the added edges) forms a simple cycle with some of the tree edges. This cycle is of length at most $2r + 1$ if it contains the root of the tree, at most $2r - 1$ otherwise. The cycle divides the plane (and the graph) into two parts, the inside and the outside of the cycle. We claim that at least one such cycle separates the graph so that neither the inside nor the outside contains vertices whose total cost exceeds 2/3. This proves the lemma.

Proof of claim. Let $(x, z)$ be the nontree edge whose cycle minimizes the maximum cost either inside or outside the cycle. Break ties by choosing the nontree edge whose cycle has the smallest number of faces on the same side as the maximum cost. If ties remain, choose arbitrarily.

Suppose without loss of generality that the graph is embedded so that the cost inside the $(x, z)$ cycle is at least as great as the cost outside the cycle. If the vertices inside the cycle have total cost not exceeding $2/3$, the claim is true. Suppose the vertices inside the cycle have total cost exceeding $2/3$. We show by case analysis that this contradicts the choice of $(x, z)$. Consider the face which has $(x, z)$ as a boundary edge and lies inside the cycle. This face is a triangle; let $y$ be its third vertex. The properties of $(x, y)$ and $(y, z)$ determine which of the following cases applies. Figure 4 illustrates the cases.

---

![Figure 4](image-url)
A SEPARATOR THEOREM

1) Both \((x, y)\) and \((y, z)\) lie on the cycle. Then the face \((x, y, z)\) is the cycle, which is impossible since vertices lie inside the cycle.

2) One of \((x, y)\) and \((y, z)\) (say \((x, y)\)) lies on the cycle. Then \((y, z)\) is a nontree edge defining a cycle which contains within it the same vertices as the original cycle but one less face. This contradicts the choice of \((x, z)\).

3) Neither \((x, y)\) nor \((y, z)\) lies on the cycle.

   a) Both \((x, y)\) and \((y, z)\) are tree edges. This is impossible since the tree itself contains no cycles.

   b) One of \((x, y)\) and \((y, z)\) (say \((x, y)\)) is a tree edge. Then \((y, z)\) is a nontree edge defining a cycle which contains one less vertex (namely \(y\)) within it than the original cycle. The inside of the \((y, z)\) cycle contains no more cost and one less face than the inside of the \((x, z)\) cycle. Thus if the cost inside the \((y, z)\) cycle is greater than the cost outside the \((y, z)\) cycle, \((y, z)\) would have been chosen in place of \((x, z)\).

   On the other hand, suppose the cost inside the \((y, z)\) cycle is no greater than the cost outside. The cost outside the \((y, z)\) cycle is equal to the cost outside the \((x, z)\) cycle plus the cost of \(y\). Since both the cost outside the \((x, z)\) cycle and the cost of \(y\) are less than \(1/3\), the cost outside the \((y, z)\) cycle is less than \(2/3\), and \((y, z)\) would have been chosen in place of \((x, z)\).

   c) Neither \((x, y)\) nor \((y, z)\) is a tree edge. Then each of \((x, y)\) and \((y, z)\) defines a cycle, and every vertex inside the \((x, z)\) cycle is either inside the \((x, y)\) cycle, inside the \((y, z)\) cycle, or on the boundary of both. Of the \((x, y)\) and \((y, z)\) cycles, choose the one (say \((x, y)\)) which has inside it more total cost. The \((x, y)\) cycle has no more cost and strictly fewer faces inside it than the \((x, z)\) cycle. Thus if the cost inside the \((x, y)\) cycle is greater than the cost outside, \((x, y)\) would have been chosen in place of \((x, z)\).

   On the other hand, suppose the cost inside the \((x, y)\) cycle is no greater than the cost outside. Since the inside of the \((x, z)\) cycle has cost exceeding \(2/3\), the \((x, y)\) cycle and its inside together have cost exceeding \(1/3\), and the outside of the \((x, y)\) cycle has cost less than \(2/3\). Thus \((x, y)\) would have been chosen in place of \((x, z)\).

   Thus all cases are impossible, and the \((x, z)\) cycle satisfies the claim.

Lemma 3. Let \(G\) be any \(n\)-vertex connected planar graph having nonnegative vertex costs summing to no more than one. Suppose that the vertices of \(G\) are partitioned into levels according to their distance from some vertex \(v\), and that \(L(l)\) denotes the number of vertices on level \(l\). If \(r\) is the maximum distance of any vertex from \(v\), let \(r + 1\) be an additional level containing no vertices. Given any two levels \(l_1\) and \(l_2\) such that levels \(0\) through \(l_1 - 1\) have total cost not exceeding \(2/3\) and levels \(l_2 + 1\) through \(r + 1\) have total cost not exceeding \(2/3\), it is possible to find a partition \(A, B, C\) of the vertices of \(G\) such that no edge joins a vertex in \(A\) with a vertex in \(B\), neither \(A\) nor \(B\) has total cost exceeding \(2/3\), and \(C\) contains no more than \(L(l_1) + L(l_2) + \max\{0, 2(l_2 - l_1 - 1)\}\) vertices.

Proof. If \(l_1 \geq l_2\), let \(A\) be all vertices on levels \(0\) through \(l_1 - 1\), \(B\) all vertices on levels \(l_1 + 1\) through \(r\), and \(C\) all vertices on level \(l_1\). Then the lemma is true. Thus suppose \(l_1 < l_2\). Delete the vertices in levels \(l_1\) and \(l_2\) from \(G\). This separates the remaining vertices of \(G\) into three parts (all of which may be empty): vertices on levels \(0\) through \(l_1 - 1\), vertices on levels \(l_1 + 1\) through \(l_2 - 1\), and vertices on levels \(l_2 + 1\) and above. The only part which can have cost exceeding \(2/3\) is the middle part.

If the middle part does not have cost exceeding \(2/3\), let \(A\) be the most costly part of the three, let \(B\) be the remaining two parts, and let \(C\) be the set of vertices on levels \(l_1\) and \(l_2\). Then the lemma is true.

Suppose the middle part has cost exceeding \(2/3\). Delete all vertices on levels \(l_2\) and above and shrink all vertices on levels \(l_1\) and below to a single vertex of cost zero.
These operations preserve planarity by Corollary 1. The new graph has a spanning tree of radius $l_i - l_i - 1$ whose root corresponds to vertices on levels $l_i$ and below in the original graph.

Apply Lemma 2 to the new graph. Let $A^*$, $B^*$, $C^*$ be the resulting vertex partition. Let $A$ be the set among $A^*$ and $B^*$ having greater cost, let $C$ consist of the vertices on levels $l_i$ and $l_{i-1}$ in the original graph plus the vertices in $C^*$ minus the root of the tree, and let $B$ contain the remaining vertices in $G$. By Lemma 2, $A$ has total cost not exceeding $2/3$. But $A \cup C^*$ has total cost at least $1/3$, so $B$ also has total cost not exceeding $2/3$. Furthermore $C$ contains no more than $L(l_i) + L(l_{i-1}) + 2(l_i - l_{i-1}) - 1$ vertices. Thus the lemma is true.

**Theorem 4.** Let $G$ be any $n$-vertex planar graph having nonnegative vertex costs summing to no more than one. Then the vertices of $G$ can be partitioned into three sets $A$, $B$, $C$ such that no edge joins a vertex in $A$ with a vertex in $B$, neither $A$ nor $B$ has total cost exceeding $2/3$, and $C$ contains no more than $2\sqrt{2n}$ vertices.

**Proof.** Assume $G$ is connected. Partition the vertices into levels according to their distance from some vertex $v$. Let $L(0)$ be the number of vertices on level 0. If $r$ is the maximum distance of any vertex from $v$, define additional levels $-1$ and $r+1$ containing no vertices.

Let $l_i$ be the level such that the sum of costs in levels 0 through $l_i - 1$ is less than $1/2$, but the sum of costs in levels 0 through $l_i$ is at least $1/2$. (If no such $l_i$ exists, the total cost of all vertices is less than $1/2$, and $B = C = \emptyset$ satisfies the theorem.) Let $k$ be the number of vertices on levels 0 through $l_i$. Find a level $l_0$ such that $l_0 \leq l_i$ and $|L(l_0)| + 2(l_i - l_0) \leq 2\sqrt{k}$. Find a level $l_2$ such that $l_i + 1 \leq l_2$ and $|L(l_2)| + 2(l_i - l_{i-1} - 1) \leq 2\sqrt{n} - k$. If two such levels exist, then by Lemma 3 the vertices of $G$ can be partitioned into three sets $A$, $B$, $C$ such that no edge joins a vertex in $A$ with a vertex in $B$, neither $A$ nor $B$ has total cost exceeding $2/3$, and $C$ contains no more than $2\sqrt{n} + k$ vertices. But $2(\sqrt{k} + \sqrt{n} - k) \leq 2(\sqrt{n}/2 + \sqrt{n}/2) = 2\sqrt{2n}$. Thus the theorem holds if suitable levels $l_0$ and $l_2$ exist.

Suppose a suitable level $l_0$ does not exist. Then, for $i \leq l_i$, $L(i) \geq 2\sqrt{k} - 2(l_i - i)$. Since $L(0) = 1$, this means $1 \geq 2\sqrt{k} - 2l_i$, and $l_i + 1/2 \geq \sqrt{k}$. Thus $l_i = \lfloor l_i + 1/2 \rfloor \geq \lfloor \sqrt{k} \rfloor$, and

$$k = \sum_{i=0}^{l_i} L(i) \geq \sum_{i=1}^{l_i} 2\sqrt{k} - 2(l_i - i) \geq (4\sqrt{k} - 2\lfloor \sqrt{k} \rfloor)(\lfloor \sqrt{k} \rfloor + 1)/2 \geq \sqrt{k}(\lfloor \sqrt{k} \rfloor + 1) > k.$$

This is a contradiction. A similar contradiction arises if a suitable level $l_2$ does not exist. This completes the proof for connected graphs.

Now suppose $G$ is not connected. Let $G_1, G_2, \ldots, G_k$ be the connected components of $G$, with vertex sets $V_1, V_2, \ldots, V_k$, respectively. If no connected component has total vertex cost exceeding $1/3$, let $i$ be the minimum index such that the total cost of $V_i \cup V_{i+1} \cup \cdots \cup V_k$ exceeds $1/3$. Let $A = V_1 \cup V_2 \cup \cdots \cup V_{i-1}$, let $B = V_{i+1} \cup V_{i+2} \cup \cdots \cup V_k$, and let $C = \emptyset$. Since $i$ is minimum and the cost of $V_i$ does not exceed $1/3$, the cost of $A$ does not exceed $2/3$. Thus the theorem is true.

If some connected component (say $G_i$) has total vertex cost between $1/3$ and $2/3$, let $A = V_n, B = V_1 \cup \cdots \cup V_{n-1} \cup V_{i+1} \cup \cdots \cup V_k$, and $C = \emptyset$. Then the theorem is true.

Finally, if some connected component (say $G_i$) has total vertex cost exceeding $2/3$, apply the above argument to $G_i$. Let $A^*, B^*, C^*$ be the resulting partition. Let $A$ be the set among $A^*$ and $B^*$ with greater cost, let $C = C^*$, and let $B$ be the remaining vertices of $G$. Then $A$ and $B$ have cost not exceeding $2/3$ and the theorem is true.
A SEPARATOR THEOREM

This proves the theorem for all planar graphs. In all cases the separator $C$ is either empty or contained in only one connected component of $G$. ◼

**Corollary 2 ($\sqrt{n}$-separator theorem).** Let $G$ be any $n$-vertex planar graph. The vertices of $G$ can be partitioned into three sets $A$, $B$, $C$ such that no edge joins a vertex in $A$ with a vertex in $B$, neither $A$ nor $B$ contains more than $2n/3$ vertices, and $C$ contains no more than $2\sqrt{2n}$ vertices.

*Proof.* Assign to each vertex of $G$ a cost of $1/n$. The corollary follows from Theorem 4. ◼

It is natural to ask whether the constant factor of $2/3$ in Theorem 1 can be reduced to $1/2$ if the constant factor of $\sqrt{2}$ is allowed to increase. The answer is yes.

**Corollary 3.** Let $G$ be any $n$-vertex planar graph having nonnegative vertex costs summing to no more than one. Then the vertices of $G$ can be partitioned into three sets $A$, $B$, $C$ such that no edge joins a vertex in $A$ with a vertex in $B$, neither $A$ nor $B$ has total cost exceeding $1/2$, and $C$ contains no more than $2\sqrt{2}/(1-\sqrt{2}/3)$ vertices.

*Proof.* Let $G = (V, E)$ be an $n$-vertex planar graph. We shall define sequences of sets $(A_i)$, $(B_i)$, $(C_i)$, $(D_i)$ such that:

(i) $A_i$, $B_i$, $C_i$, $D_i$ partition $V$,
(ii) no edge joins $A_i$ with $B_i$, $A_i$ with $D_i$, or $B_i$ with $D_i$,
(iii) the cost of $A_i$ is no greater than the cost of $B_i$, and the cost of $B_i$ is no greater than the cost of $A_i \cup C_i \cup D_i$,
(iv) $|D_i| \leq 2|D_{i-1}|/3$.

Let $A_0 = B_0 = C_0 = D_0 = V$. Then (i)-(iv) hold. If $A_{i-1}$, $B_{i-1}$, $C_{i-1}$, $D_{i-1}$ have been defined and $D_{i-1} \neq \emptyset$, let $G^*$ be the subgraph of $G$ induced by the vertex set $D_{i-1}$. Let $A^*$, $B^*$, $C^*$ be a vertex partition satisfying Corollary 2 on $G^*$. Without loss of generality, suppose $A^*$ has no more cost than $B^*$. Let $A_i$ be the set among $A_{i-1} \cup A^*$, $B_{i-1}$ with less cost, let $B_i$ be the set among $A_{i-1} \cup A^*$, $B_{i-1}$ with greater cost, let $C_i = C_{i-1} \cup C^*$, and let $D_i = B^*$. Then (i), (ii), (iii), and (iv) hold for $A_i$, $B_i$, $C_i$, $D_i$.

Let $k$ be the largest index for which $A_k$, $B_k$, $C_k$, $D_k$ are defined. Then $D_k = \emptyset$. Let $A = A_k$, $B = B_k$, $C = C_k$. By (i), $A$, $B$, $C$ partition $V$. By (ii), no edge joins a vertex in $A$ with a vertex in $B$. By (iii), neither $A$ nor $B$ has cost exceeding $1/2$. By (iv), the total number of vertices in $C$ is bounded by

$$\sum_{i=0}^{\infty} 2\sqrt{2}\sqrt{n}(2/3)^{i/2} = 2\sqrt{2}\sqrt{n}\frac{1}{1-\sqrt{2}/3}. \quad \Box$$

Another natural question is whether graphs which are "almost" planar have a $\sqrt{n}$-separator theorem. The finite element method of numerical analysis gives rise to one interesting class of almost-planar graphs. We shall extend Theorem 4 to apply to such graphs.

A **finite element graph** is any graph formed from a planar embedding of a planar graph by adding all possible diagonals to each face. (The finite element graph has a clique corresponding to each face of the embedded planar graph.) The embedded planar graph is called the **skeleton** of the finite element graph and each of its faces is an **element** of the finite element graph.

**Theorem 5.** Let $G$ be an $n$-vertex finite element graph with nonnegative vertex costs summing to no more than one. Suppose no element of $G$ has more than $k$ boundary vertices. Then the vertices of $G$ can be partitioned into three sets $A$, $B$, $C$ such that no edge joins a vertex in $A$ with a vertex in $B$, neither $A$ nor $B$ has total cost exceeding $2/3$, and $C$ contains no more than $4\lfloor k/2 \rfloor \sqrt{n}$ vertices.
Proof. Let \( G^* \) be the skeleton of \( G \). Form \( G^{**} \) from \( G^* \) by inserting one new vertex into each face of \( G^* \) containing four or more vertices and connecting the new vertex to each vertex on the boundary of the face. Then \( G^{**} \) is planar. Apply Theorem 4 to \( G^{**} \). Let \( A^{**}, B^{**}, C^{**} \) be the resulting vertex partition. This partition satisfies the theorem except that certain edges in \( G \) but not in \( G^{**} \) may join \( A^{**} \) and \( B^{**} \). These edges are diagonals of certain faces of \( G^* \); call these bad faces. Each bad face must contain one of the new vertices added to \( G^* \) to form \( G^{**} \), and this vertex must be in \( C^{**} \).

Form \( G \) from \( C^{**} \) by deleting all new vertices and adding to \( G^{**} \), for each bad face, either the set of vertices in \( A^{**} \) on the boundary of the bad face, or the set of vertices in \( B^{**} \) on the boundary of the bad face, whichever is smaller. Let \( A \) be the remaining old vertices in \( A^{**} \) and let \( B \) be the remaining old vertices in \( B^{**} \). Then no edge in \( G \) joins \( A \) and \( B \), neither \( A \) nor \( B \) contains more than \( 2n/3 \) vertices, and \( C \) contains no more than \( 4(k/2)^2 + a \) vertices, where \( a \) is the number of faces of \( G^* \) containing four or more vertices. By use of Euler's theorem, it is not hard to show that the number of faces of \( G^* \) containing four or more vertices is at most \( n - 2 \). Thus \( |C| \leq (k/2)^2 \), and the theorem is true. \( \square \)

Corollary 4. Let \( G \) be any \( n \)-vertex finite element graph. Suppose no element of \( G \) has more than \( k \) boundary vertices. The vertices of \( G \) can be partitioned into three sets \( A, B, C \) such that no edge joins a vertex in \( A \) with a vertex in \( B \), neither \( A \) nor \( B \) contains more than \( 2n/3 \) vertices and \( C \) contains no more than \( 4(k/2)^2 \sqrt{n} \) vertices.

The last result of this section shows that Theorem 4 and its corollaries are tight to within a constant factor; that is, if \( f(n) = o(\sqrt{n}) \), no \( f(n) \)-separator theorem holds for planar graphs.

Theorem 6. For any \( k \), let \( G = (V, E) \) be a \( k \times k \) square grid graph (a \( k \times k \) square section of the infinite grid graph in Fig. 1). Let \( A \) be any subset of \( V \) such that \( \alpha n \leq |A| \leq n/2 \), where \( n = k^2 \) and \( \alpha \) is a positive constant less than \( 1/2 \). The number of vertices in \( V - A \) adjacent to some vertext in \( A \) is at least \( k \cdot \min \{1/2, \sqrt{\alpha} \} \).

Proof. Without loss of generality, suppose that the number \( r \) of rows of \( G \) which contain vertices in \( A \) is no less than the number \( c \) of columns of \( G \) which contain vertices in \( A \). Then \( \alpha n \leq |A| \leq n/2 \), and \( r \geq \sqrt{\alpha} \).

If \( r^* \) is the number of rows of \( G \) which contain only vertices in \( A \), then \( kr^* \geq |A| \geq n/2 \), and \( r^* \leq k/2 \). Let \( S = \{x \in V : x \text{ is adjacent to a vertex of } A\} \). If \( r^* = 0 \), then \( |S| \geq r \geq \sqrt{\alpha} \). If \( r^* \neq 0 \), then \( r = k \) and \( |S| \geq r - r^* = k - r^* \geq k/2 \). \( \square \)

It is an open problem to determine the smallest constant factor which can replace \( 2\sqrt{2} \) in Theorem 4.

3. An algorithm for finding a good partition. The proof of Theorem 4 leads to an algorithm for finding a vertex partition satisfying the theorem. To make this algorithm efficient, we need a good representation of a planar embedding of a graph. For this purpose we use a list structure whose elements correspond to the edges of the graph. Stored with each edge are its endpoints and four pointers, designating the edges immediately clockwise and counter-clockwise around each of the endpoints of the edge. Stored with each vertex is some incident edge. Figure 5 gives an example of such a data structure.

Partitioning Algorithm.
Step 1. Find a planar embedding of \( G \) and construct a representation for it of the kind described above.

\( \text{Time: } O(n) \), using the algorithm of [10].
**A SEPARATOR THEOREM**

![Graph Image]

**Vertex incidences**

<table>
<thead>
<tr>
<th>Vertex</th>
<th>$e_1$</th>
<th>$e_2$</th>
<th>$e_3$</th>
<th>$e_4$</th>
<th>$e_5$</th>
<th>$e_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Edges and neighbors**

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>$cc_1$</th>
<th>$c_2$</th>
<th>$cc_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>1</td>
<td>2</td>
<td>$e_3$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>1</td>
<td>3</td>
<td>$e_1$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>1</td>
<td>4</td>
<td>$e_2$</td>
</tr>
<tr>
<td>$e_4$</td>
<td>2</td>
<td>3</td>
<td>$e_5$</td>
</tr>
<tr>
<td>$e_5$</td>
<td>2</td>
<td>4</td>
<td>$e_6$</td>
</tr>
<tr>
<td>$e_6$</td>
<td>3</td>
<td>4</td>
<td>$e_1$</td>
</tr>
</tbody>
</table>

**FIG. 5.** Representation of an embedded planar graph. (c = clockwise, cc = counter-clockwise.)

**Step 2.** Find the connected components of $G$ and determine the cost of each one. If none has cost exceeding $2/3$, construct the partition as described in the proof of Theorem 4. If some component has cost exceeding $2/3$, go to Step 3.

*Time:* $O(n)$ [9].

**Step 3.** Find a breadth-first spanning tree of the most costly component. Compute the level of each vertex and the number of vertices $L(l)$ in each level $l$.

*Time:* $O(n)$.

**Step 4.** Find the level $l_1$ such that the total cost of levels 0 through $l_1 - 1$ does not exceed $1/2$, but the total cost of levels 0 through $l_1$ does exceed $1/2$. Let $k$ be the number of vertices in levels 0 through $l_1$.

*Time:* $O(n)$.

**Step 5.** Find the highest level $l_0 \equiv l_1$ such that $L(l_0) + 2(l_1 - l_0) \leq 2\sqrt{k}$. Find the lowest level $l_2 \equiv l_1 + 1$ such that $L(l_2) + 2(l_2 - l_1 - 1) \leq 2\sqrt{n - k}$.

*Time:* $O(n)$.

**Step 6.** Delete all vertices on level $l_2$ and above. Construct a new vertex $x$ to represent all vertices on levels 0 through $l_0$. Construct a Boolean table with one entry per vertex. Initialize to **true** the entry for each vertex on levels 0 through $l_0$ and
initialize to false the entry for each vertex on levels \( l_0 + 1 \) through \( l_2 - 1 \). The vertices on levels 0 through \( l_0 \) correspond to a subtree of the breadth-first spanning tree generated in Step 3. Scan the edges incident to this tree clockwise around the tree. When scanning an edge \((v, w)\) with \(v\) in the tree, check the table entry for \(w\). If it is true, delete edge \((v, w)\). If it is false, change it to true, construct an edge \((x, w)\), and delete edge \((v, w)\). The result of this step is a planar representation of the shrunken graph to which Lemma 2 is to be applied. See Fig. 6.

**Time:** \(O(n)\).

**Step 7.** Construct a breadth-first spanning tree rooted at \(x\) in the new graph. (This can be done by modifying the breadth-first spanning tree constructed in Step 3.) Record, for each vertex \(v\), the parent of \(v\) in the tree, and the total cost of all descendants of \(v\) including \(v\) itself. Make all faces of the new graph into triangles by scanning the boundary of each face and adding (nontree) edges as necessary.

**Time:** \(O(n)\).

**Step 8.** Choose any nontree edge \((v_1, w_1)\). Locate the corresponding cycle by following parent pointers from \(v_1\) and \(w_1\). Compute the cost on each side of this cycle by scanning the tree edges incident on either side of the cycle and summing their associated costs. If \((v, w)\) is a tree edge with \(v\) on the cycle and \(w\) not on the cycle, the cost associated with \((v, w)\) is the descendant cost of \(w\) if \(v\) is the parent of \(w\), and the cost of all vertices minus the descendant cost of \(v\) if \(w\) is the parent of \(v\). Determine which side of the cycle has greater cost and call it the "inside". See Fig. 7.

**Time:** \(O(n)\).
Step 9. Let \((v_n, w_i)\) be the nontree edge whose cycle is the current candidate to complete the separator. If the cost inside the cycle exceeds 2/3, find a better cycle by the following method.

Locate the triangle \((v_n, y, w_i)\) which has \((v_n, w_i)\) as a boundary edge and lies inside the \((v_n, w_i)\) cycle. If either \((v_n, y)\) or \((y, w_i)\) is a tree edge, let \((v_{i+1}, w_{i+1})\) be the nontree edge among \((v_n, y)\) and \((y, w_i)\). Compute the cost inside the \((v_{i+1}, w_{i+1})\) cycle from the cost inside the \((v_n, w_i)\) cycle and the cost of \(v_{i+1}\) and \(w_{i+1}\). See Fig. 4.

If neither \((v_n, y)\) nor \((y, w_i)\) is a tree edge, determine the tree path from \(y\) to the \((v_n, w_i)\) cycle by following parent pointers from \(y\). Let \(z\) be the vertex on the \((v_n, w_i)\) cycle reached during this search. Compute the total cost of all vertices except \(z\) on this tree path. Scan the tree edges inside the \((y, w_i)\) cycle, alternately scanning an edge in one cycle and an edge in the other cycle. Stop scanning when all edges inside one of the cycles have been scanned. Compute the cost inside this cycle by summing the associated costs of all scanned edges. Use this cost, the cost inside the \((v_n, w_i)\) cycle, and the cost on the tree path from \(y\) to \(z\) to compute the cost inside the other cycle. Let \((v_{i+1}, w_{i+1})\) be the edge among \((v_n, y)\) and \((y, w_i)\) whose cycle has more cost inside it.

Repeat Step 9 until finding a cycle whose inside has cost not exceeding 2/3.

Time: \(O(n)\) (see proof below).

Step 10. Use the cycle found in Step 9 and the levels found in Step 4 to construct a satisfactory vertex partition as described in the proof of Lemma 3. Extend this partition from the connected component chosen in Step 2 to the entire graph as described in the proof of Theorem 4.

Time: \(O(n)\).

This completes our presentation of the algorithm. All steps except Step 9 obviously run in \(O(n)\) time. We urge readers to fill in the details of this algorithm; we content ourselves here with proving that Step 9 requires \(O(n)\) time.

Proof of Step 9 time bound. Each iteration of Step 9 deletes at least one face from the inside of the current cycle. Thus Step 9 terminates after \(O(n)\) iterations. The total
running time of one iteration of Step 9 is $O(1)$ plus time proportional to the length of the tree path from $y$ to $z$ plus time proportional to the number of edges scanned inside the $(v, y)$ and $(y, w_1)$ cycles. Each vertex on the tree path from $y$ to $z$ (except $z$) is inside the current cycle but on the boundary or outside of all subsequent cycles. For every two edges scanned during an iteration of Step 9, at least one edge is inside the current cycle but outside all subsequent cycles. It follows that the total time spent traversing tree paths and scanning edges, during all iterations of Step 9, is $O(n)$. Thus the total time spent in Step 9 is $O(n)$. ❑

By making minor modifications to this algorithm, one can construct an $O(n)$ time algorithm to find a vertex partition satisfying Theorem 5, and $O(n)$ time algorithms to find vertex partitions satisfying Corollary 2 and Corollary 4.

4. Applications. The separator theorem proved in § 2 allows us to obtain many new complexity results since it opens the way for efficient application of divide-and-conquer on planar graphs. We mention a few such applications here; we shall present the details in a subsequent paper.

Generalized nested dissection. Any system of linear equations whose sparsity structure corresponds to a planar or finite element graph can be solved in $O(n^{3/2})$ time and $O(n, \log n)$ space. This result generalizes the nested dissection method of George [5].

Pebbling. Any $n$-vertex planar acyclic directed graph with maximum in-degree $k$ can be pebbled using $O(n + k \log n)$ pebbles. See [8], [16] for a description of the pebble game.

The post office problem. Knuth’s “post office” problem [11] can be solved in $O((\log n)^2)$ time and $O(n)$ space. See [3], [17] for previous results.

Data structure embedding problems. Any planar data structure can be efficiently embedded into a balanced binary tree. See [2], [14] for a description of the problem and some related results.

Lower bounds on Boolean circuits. Any planar circuit for computing Boolean convolution contains at least $cn^2$ gates for some positive constant $c$.

Appendix. Graph-theoretic definitions. A graph $G = (V, E)$ consists of a set $V$ of vertices and a set $E$ of edges. Each edge is an unordered pair $(v, w)$ of distinct vertices. If $(v, w)$ is an edge, $v$ and $w$ are adjacent and $(v, w)$ is incident to both $v$ and $w$. A path of length $k$ with endpoints $v, w$ is a sequence of vertices $v = v_0, v_1, v_2, \ldots, v_k = w$ such that $(v_{i-1}, v_i)$ is an edge for $1 \leq i \leq k$. If all the vertices $v_0, v_1, \ldots, v_{k-1}$ are distinct, the path is simple. If $v = w$, the path is a cycle. The distance from $v$ to $w$ is the length of the shortest path from $v$ to $w$. (The distance is infinite if $v$ and $w$ are not joined by a path.) The level of a vertex $v$ in a graph $G$ with respect to a fixed root $r$ is the distance from $r$ to $v$.

If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are graphs, $G_1$ is a subgraph of $G_2$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. $G_1$ is a generalized subgraph of $G_2$ if $V_1 \subseteq V_2$ and there is a mapping $f$ from $E_1$ into the set of paths of $G_2$ such that, for each edge $(v, w) \in E_1$, $f((v, w))$ has endpoints $v$ and $w$, and no two paths $f((v_1, w_1))$ and $f((v_2, w_2))$ share a vertex except possibly an endpoint of both paths. If $G = (V_1, E_1)$ is a graph and $V_1 \subseteq V_2$, the graph $G_1 = (V_1, E_1)$ where $E_1 = E_2 \cap \{(v, w) | v, w \in V_1\}$ is the subgraph of $G_2$ induced by the vertex set $V_1$. If $G_1 = (V_1, E_1)$ is a subgraph of $G_2 = (V_2, E_2)$, then shrinking $G_1$ to a single vertex in $G_2$ means forming a new graph $G_3$ from $G_2$ by deleting from $G_2$ all vertices in $V_1$ and all their incident edges, adding a new vertex $x$ to $G_2$, and adding a new edge $(x, w)$ to $G_2$ for each edge $(v, w) \in E_2$ such that $v \in V_1$ and $w \notin V_1$.

A graph is connected if any two vertices in it are joined by a path. The connected components of a graph are its maximal connected subgraphs. A clique is a graph such
A SEPARATOR THEOREM

that any two vertices are joined by an edge. A tree is a connected graph containing no cycles. We shall generally assume that a tree has a distinguished vertex, called a root. If \( T \) is a tree with root \( r \) and \( v \) is on the (unique) simple path from \( r \) to \( w \), \( v \) is an ancestor of \( w \) and \( w \) is a descendant of \( v \). If in addition \((v, w)\) is an edge of \( T \), then \( v \) is the parent of \( w \) and \( w \) is a child of \( v \). The radius of a tree is the maximum distance of any vertex from the root. A spanning tree \( T \) of a graph \( G \) is a subgraph of \( G \) which is a tree and which contains all the vertices of \( G \). \( T \) is a breadth-first spanning tree with respect to a root \( r \) if, for any vertex \( v \), the distance from \( r \) to \( v \) in \( T \) is equal to the distance from \( r \) to \( v \) in \( G \).

A graph \( G = (V, E) \) is planar if there is a one-to-one map \( f_1 \) from \( V \) into points in the plane and a map \( f_2 \) from \( E \) into simple curves in the plane such that, for each edge \((v, w) \in E\), \( f_2((v, w)) \) has endpoints \( f_1(v) \) and \( f_1(w) \), and no two curves \( f_2((v_1, w_1)), f_2((v_2, w_2)) \) share a point except possibly a common endpoint. Such a pair of maps \( f_1, f_2 \) is a planar embedding of \( G \). The connected planar regions formed when the ranges of \( f_1 \) and \( f_2 \) are deleted from the plane are called the faces of the embedding. Each face is bounded by a curve corresponding to a cycle of \( G \), called the boundary of the face. We shall sometimes not distinguish between a face and its boundary. A diagonal of a face is an edge \((v, w)\) such that \( v \) and \( w \) are nonadjacent vertices on the boundary of the face.

Acknowledgments. We would like to thank Stanley Eisenstat, Rich A. DeMillo, Robert Floyd, Donald Rose, and Daniel Sleator for many helpful discussions and much thoughtful criticism.

REFERENCES

APPLICATIONS OF A PLANAR SEPARATOR THEOREM

Richard J. Lipton
Computer Science Department
Yale University
New Haven, Connecticut 06520

Robert Endre Tarjan
Computer Science Department
Stanford University
Stanford, California 94305

August 1977

Abstract.
Any n-vertex planar graph has the property that it can be divided into components of roughly equal size by removing only $O(\sqrt{n})$ vertices. This separator theorem, in combination with a divide-and-conquer strategy, leads to many new complexity results for planar graph problems. This paper describes some of these results.

Keywords: algorithm, Boolean circuit complexity, divide-and-conquer, geometric complexity, graph embedding, lower bounds, maximum independent set, non-serial dynamic programming, pebbling, planar graphs, separator, space-time tradeoffs.

1. Introduction.
One efficient approach to solving computational problems is "divide-and-conquer" [1]. In this method, the original problem is divided into two or more smaller problems. The subproblems are solved by applying the method recursively, and the solutions to the subproblems are combined to give the solution to the original problem. Divide-and-conquer is especially efficient when the subproblems are substantially smaller than the original problem.

In [14] the following theorem is proved.

Theorem 1. Let $G$ be any $n$-vertex planar graph with non-negative vertex costs summing to no more than one. Then the vertices of $G$ can be partitioned into three sets $A, B, C$, such that no edge joins a vertex in $A$ with a vertex in $B$, neither $A$ nor $B$ contains more than $2/3$ of the total vertex cost exceeding $2/3$, and $C$ contains no more than $2\sqrt{2}n$ vertices. Furthermore $A, B, C$ can be found in $O(n)$ time.

In the special case of equal-cost vertices, this theorem becomes

Corollary 1. Let $G$ be any $n$-vertex planar graph. The vertices of $G$ can be partitioned into three sets $A, B, C$, such that no edge joins a vertex in $A$ with a vertex in $B$, neither $A$ nor $B$ contains more than $2n/3$ vertices, and $C$ contains no more than $2\sqrt{2}n$ vertices.

Theorem 1 and its corollary open the way for efficient application of divide-and-conquer to a variety of problems on planar graphs. In this paper we explore a number of such applications. Each section of the paper describes a different use of divide-and-conquer. The results range from an efficient approximation algorithm for finding maximum independent sets in planar graphs to lower bounds on the complexity of planar Boolean circuits. The last section mentions two additional applications whose description is too lengthy to be included in this paper.

Divide-and-conquer in combination with Theorem 1 can be used to rapidly find good approximate solutions to certain NP-complete problems on planar graphs. As an example we consider the maximum independent set problem, which asks for a maximum number of pairwise non-adjacent vertices in a planar graph.

Theorem 2. Let $G$ be an $n$-vertex planar graph with non-negative vertex costs summing to no more than one and let $0 < \epsilon < 1$. Then there is some set $C$ of $O(\sqrt{n/\epsilon})$ vertices whose removal leaves $G$ with no connected component of cost exceeding $\epsilon$. Furthermore the set $C$ can be found in $O(n \log n)$ time.

Proof. Apply the following algorithm to $G$.

Initialization: Let $C = \emptyset$.

General Step: Find some connected component K in $G$ minus $C$ with cost exceeding $\epsilon$. Apply Corollary 1 to $K$, producing a partition $A_1, B_1, C_1$ of its vertices. Let $C = C \cup C_1$.

If one of $A_1$ and $B_1$ (say $A_1$) has cost exceeding two-thirds the cost of $K$, apply Theorem 1 to the subgraph of $G$ induced by the vertex set $A_1$, producing a partition $A_2, B_2, C_2$ of $A_1$. Let $C = C \cup C_2$.

Repeat the general step until $G$ minus $C$ has no component with cost exceeding $\epsilon$.

The effect of one execution of the general step is to divide the component $K$ into smaller components, each with no more than two-thirds the cost of $K$ and each with no more than two-thirds as many vertices as $K$. Consider all components which arise during the course of the algorithm. Assign a level to each component as follows. If the component exists when the algorithm halts, the component has level zero. Otherwise the level of the component is one greater than the maximum level of the components formed when it is split by the general step. With this definition, any two components on the same level are vertex-disjoint.

Each level one component has cost greater than $\epsilon$, since it is eventually split by the general step. It

This research partially supported by the U. S. Army Research Office, Grant No. DAAG 29-76-C-0338.

**/ This research partially supported by National Science Foundation grant MCS-75-22870 and by the Office of Naval Research contract N00014-76-C-0688.

Reproduction in whole or in part is permitted for any purpose of the United States government.
follows that, for i \geq 1, each level i component has cost at least \((3/2)^{1-1/c}\) and contains at least \((3/2)^{i}\) vertices. Since the total cost of G is at most one, the total number of components of level i is at most \((2/3)^{i-1}/c\).

The total running time of the algorithm is \(O\left(\sum_{i=1}^{E} |K| \mid K \text{ is a component split by the general step}\right)\). Since a component of level i contains at least \((3/2)^{i}\) vertices, the maximum level k must satisfy \((3/2)^{k} \leq n\), or \(k < \log_{3/2} n\). Since components in each level are vertex-disjoint, the total running time of the algorithm is \(O(n \log_{3/2} n) = O(n \log n)\).

The total size of the set C produced by the algorithm is bounded by \(O\left(\sum_{i=1}^{E} (2/3)^{1-1/c} \frac{n_j}{n} \mid n_j \leq n \text{ and } n_j \geq 0\right)\).

The following algorithm uses Theorem 2 to find an approximately maximum independent set I in a planar graph G = (V, E).

**Step 1.** Apply Theorem 2 to G with \(c = (\log \log n)/n\) and each vertex having cost \(1/n\) to find a set of vertices C containing \(O(n \log \log n)\) vertices whose removal leaves no connected component with more than \(\log \log n\) vertices.

**Step 2.** In each connected component of G minus C, find a maximum independent set by checking every subset of vertices for independence. Form I as a union of maximum independent sets, one from each component.

Let I* be a maximum independent set of G. The restriction of I* to one of the connected components formed when C is removed from G can be no larger than the restriction of I to the same component. Thus \(|I^*| - |I| = O(n/\log \log n)\). Since G is planar, G is four-colorable, and \(|I^*| \geq n/4\). Thus \(|I^*|/|I^*| = O(1/\log \log n)\), and the relative error in the size of I tends to zero with increasing n.

Step 1 of the algorithm requires \(O(n \log n)\) time by Theorem 2. Step 2 requires \(O(n_i \geq \star)\) time on a connected component of \(n_i\) vertices. The total time required by Step 2 is thus \(O\left(\max\left\{\sum_{i=1}^{E} n_i^{1-1/c} \mid \sum_{i=1}^{E} n_i = n \text{ and } 0 \leq n_i \leq \log \log n\right\}\right) = O\left(\sum_{i=1}^{E} \frac{n_j}{n} \log \log n \right) = O(n \log n)\).

Hence the entire algorithm requires \(O(n \log n)\) time.

3. **Nonserial Dynamic Programming.**

Many NP-complete problems, such as the maximum independent set problem, the graph coloring problem, and others, can be formulated as non-serial dynamic programming problems [2,20]. Such a problem is of the following form: minimize the objective function \(f(x_1, \ldots, x_n)\), where f is given as a sum of terms \(f_k(\cdot)\), each of which is a function of only a subset of the variables. We shall assume that all variables \(x_i\) take on values from the same finite set S, and that the values of the terms \(f_k(\cdot)\) are given by tables. Associated with such an objective function f is an interaction graph G = (V, E), containing one vertex \(v_i\) for each variable \(x_i\) in f, and an edge joining \(x_i\) and \(x_j\) for any two variables \(x_i\) and \(x_j\) which appear in a common term \(f_k(\cdot)\).

By trying all possible values of the variables, a nonserial dynamic programming problem can be solved in \(O(n!\log n)\) time. We shall show that if the interaction graph of the problem is planar, the problem can be solved in \(O(n\log n)\) time. This means that substantial savings are possible when solving typical NP-complete problems restricted to planar graphs.

Note that if the interaction graph of f is planar, no term \(f_k(\cdot)\) of f can contain more than four variables, since the complete graph on five vertices is not planar.

In order to describe the algorithm, we need one additional concept. The restriction of an objective function \(f(x_1, \ldots, x_n) = \sum_{k=1}^{m} f_k\) to a set of variables \(x_1, \ldots, x_j\) is the objective function \(f^* = \sum_{k=1}^{m} f_k\) depends upon one or more of \(x_1, \ldots, x_j\).

Given an objective function \(f(x_1, \ldots, x_n) = \sum_{k=1}^{m} f_k\) and a subset S of the variables \(x_1, \ldots, x_n\) which are constrained to have specific values, the following algorithm solves the problem: maximize f subject to the constraints on the variables in S.

In the presentation, we do not distinguish between the variables \(x_1, \ldots, x_n\) and the corresponding vertices in the interaction graph.

**Step 1.** If \(n \leq 9\), solve the problem by exhaustively trying all possible assignments to the unconstrained variables. Otherwise, go to Step 2.
Step 2. Apply Corollary 1 to the interaction graph \( G \) of \( f \). Let \( A, B, C \) be the resulting vertex partition. Let \( f_1 \) be the restriction of \( f \) to \( A \cup C \) and let \( f_2 \) be the restriction of \( f \) to \( B \cup C \). For each possible assignment of values to the variables in \( C \cup S \), perform the following steps:

(a) Maximize \( f_1 \) with the given values for the variables in \( C \cup S \) by applying the method recursively;
(b) Maximize \( f_2 \) with the given values for the variables in \( C \cup S \) by applying the method recursively;
(c) Combine the solutions to (a) and (b) to obtain a maximum value of \( f \) with the given values for the variables in \( C \cup S \).

Choose the assignment of values to variables in \( C \cup S \) which maximizes \( f \) and return the appropriate value of \( f \) as the solution.

The correctness of this algorithm is obvious. If \( n > 9 \), the algorithm solves at most \( 2^{O(Vn)} \) subproblems in Step 2, since \( C \) is of \( O(Vn) \) size. Each subproblem contains at most \( 2n/3 + 2Vn/4 < 2Vn/3 \) variables. Thus if \( t(n) \) is the running time of the algorithm, we have
\[
t(n) \leq O(n \log n) + 2^{O(Vn)}, t(29n/30) \text{ if } n > 9,
\]
\[
t(n) = O(1) \text{ if } n \leq 9 .
\]
An inductive proof shows that \( t(n) \leq 2^{O(Vn)} \).

4. Pebbling.

The following one-person game arises in register allocation problems [21], the conversion of recursion to iteration [16], and the study of time-space trade-offs [4,10,15]. Let \( G = (V,E) \) be a directed acyclic graph with maximum in-degree \( k \). If \((v,w)\) is an edge of \( G \), \( v \) is a predecessor of \( w \) and \( w \) is a successor of \( v \). The game involves placing pebbles on the vertices of \( G \) according to certain rules. A given step of the game consists of either placing a number of pebbles on an empty vertex of \( G \) (called pebbling the vertex) or removing a pebble from a previously pebbled vertex. A pebble on an empty vertex of \( G \) is called a pebble on an empty vertex of \( G \) to pebble all predecessors of \( v \) to be pebbled. The total number of pebbles required by this method on any \( n \)-vertex graph, then
\[
p(n) = n \text{ if } n \leq n_0 ,
\]
\[
p(n) \leq \alpha \sqrt{n} + k + p(2n/3 + \alpha \sqrt{n}) \text{ if } n > n_0 .
\]
An inductive proof shows that \( p(n) \) is \( O(Vn) + k \log_2 n \).

It is also possible to obtain a substantial reduction in pebbles while preserving a polynomial bound on the number of pebbling steps, at the following theorem shows.

Theorem 4. Any \( n \)-vertex planar acyclic directed graph with maximum in-degree \( k \) can be pebbled using \( O(n^{2/3} + k) \) pebbles in \( O(kn^{5/3}) \) time.

Proof. Let \( C \) be a set of \( O(n^{2/3}) \) vertices whose removal leaves \( G \) with no weakly connected component containing more than \( n^{2/3} \) vertices. Such a set \( C \) exists by Theorem 2. The following pebbling procedure places pebbles permanently on the vertices of \( C \). Pebble the vertices of \( G \) in topological order. To pebble a vertex \( v \), pebble each predecessor \( u \) of \( v \) and then pebble \( v \). To pebble a predecessor \( u \), delete all pebbles from \( G \) on \( u \) except those on vertices in \( C \) or on predecessors of \( v \). Find the weakly connected component in \( G \) minus \( C \) containing \( u \). Pebble all vertices in this component in topological order.

The total number of pebbles required by this strategy is \( O(n^{2/3}) \) to pebble vertices in \( C \) plus \( n^{2/3} \) to pebble each weakly connected component plus \( k \) to pebble predecessors of the vertex \( v \) to be pebbled. The total number of pebbling steps is at most \( O(n^{2/3}) + O(kn^{5/3}) \).

5. Lower Bounds on Boolean Circuit Size.

A Boolean circuit is an acyclic directed graph such that each vertex has in-degree zero or two, the predecessors of each vertex are ordered, and corresponding to each vertex \( v \) of in-degree two is a binary Boolean operation \( b_v \). With each vertex of the circuit we associate a Boolean function which the vertex computes, defined as follows. With each of the \( k \) vertices \( v_i \) of in-degree zero (inputs)

\[
\text{If } n \geq n_0 , \text{ where } n_0 = (\alpha(1-\beta))^2 , \text{ pebble all vertices of } G \text{ without deleting pebbles. If } n > n_0 , \text{ find a vertex partition } A, B, C \text{ satisfying Corollary 1. Pebble the vertices of } G \text{ in topological order. To pebble a vertex } v \text{, delete all pebbles except those on } C . \text{ For each predecessor } u \text{ of } v \text{, let } C(u) \text{ be the subgraph of } G \text{ induced by the set of vertices with pebble-free paths to } u . \text{ Apply the method recursively to each } C(u) \text{ to pebble all predecessors of } v \text{, leaving a pebble on each such predecessor. Then pebble } v .
\]

If \( p(n) \) is the maximum number of pebbles required by this method on any \( n \)-vertex graph, then
\[
p(n) = n \text{ if } n \leq n_0 ,
\]
\[
p(n) \leq \alpha \sqrt{n} + k + p(2n/3 + \alpha \sqrt{n}) \text{ if } n > n_0 .
\]
An inductive proof shows that \( p(n) \) is \( O(Vn) + k \log_2 n \).

It is also possible to obtain a substantial reduction in pebbles while preserving a polynomial bound on the number of pebbling steps, at the following theorem shows.

Theorem 4. Any \( n \)-vertex planar acyclic directed graph with maximum in-degree \( k \) can be pebbled using \( O(n^{2/3} + k) \) pebbles in \( O(kn^{5/3}) \) time.

Proof. Let \( C \) be a set of \( O(n^{2/3}) \) vertices whose removal leaves \( G \) with no weakly connected component containing more than \( n^{2/3} \) vertices. Such a set \( C \) exists by Theorem 2. The following pebbling procedure places pebbles permanently on the vertices of \( C \). Pebble the vertices of \( G \) in topological order. To pebble a vertex \( v \), pebble each predecessor \( u \) of \( v \) and then pebble \( v \). To pebble a predecessor \( u \), delete all pebbles from \( G \) except those on vertices in \( C \) or on predecessors of \( v \). Find the weakly connected component in \( G \) minus \( C \) containing \( u \). Pebble all vertices in this component in topological order.

The total number of pebbles required by this strategy is \( O(n^{2/3}) \) to pebble vertices in \( C \) plus \( n^{2/3} \) to pebble each weakly connected component plus \( k \) to pebble predecessors of the vertex \( v \) to be pebbled. The total number of pebbling steps is at most \( O(n^{2/3}) + O(kn^{5/3}) \).

That is, in an order such that if \( v \) is a predecessor of \( w \), \( v \) is pebbled before \( w \).

A weakly connected component of a directed graph is a connected component of the undirected graph formed by ignoring edge directions.
we associate a variable $x_i$ and an identity function $f_{x_i} = x_i$. With each vertex $v$ of in-degree two having predecessors $u,v$ we associate the function $f_v = b_v(u,v)$. The circuit computes the set of functions associated with its vertices of out-degree zero (outputs).

We are interested in obtaining lower bounds on the size (number of vertices) of Boolean circuits which compute certain common and important functions. Using Theorem 1 we can obtain such lower bounds under the assumption that the circuits are planar. Any circuit can be converted into a planar circuit by the following steps. First, embed the circuit in the plane, allowing edges to cross if necessary. Next, replace each pair of crossing edges by the crossover circuit illustrated in Figure 1. It follows that any lower bound on the size of planar circuits is also a lower bound on the total number of vertices and edge crossings in any planar representation of a non-planar circuit. In a technology for which the total number of vertices and edge crossings is a reasonable measure of cost, our lower bounds imply that it may be expensive to realize certain commonly used functions in hardware.

A superconcentrator is an acyclic directed graph with $m$ inputs and $m$ outputs such that any set of $k$ inputs and any set of $k$ outputs are joined by $k$ vertex-disjoint paths, for all $k$ in the range $1 < k < m$.

**Theorem 5.** Any $m$-input, $m$-output planar superconcentrator contains at least $m^2/12$ vertices.

**Proof.** Let $G$ be an $m$-input, $m$-output planar superconcentrator. Assign to each input and output of $G$ a cost of $1/(2m)$, and to every other vertex a cost of zero. Let $A$, $B$, $C$ be a vertex partition satisfying Theorem 1 on $G$ (ignoring edge directions). Suppose $C$ contains $p$ inputs and outputs. Without loss of generality, suppose that $A$ is no more costly than $B$, and that $A$ contains no more outputs than inputs, $A$ contains between $2m/3 - p$ and $m - p/2$ inputs and outputs. Hence $A$ contains at least $m/3 - p/2$ inputs and at most $m/2 - 3p/4$ outputs. Let $k = \min\{m/3 - p/2; (m/2 - 3p/4)\}$. Since $G$ is a superconcentrator, any set of $k$ inputs in $A$ and any set of $k$ outputs in $B$ are joined by $k$ vertex-disjoint paths. Each such path must contain a vertex in $C$ which is neither an input nor an output, Thus $2\sqrt{2/n} - p > \min\{m/3 - p/2; m/2 - 3p/4\} = m/3 - p$, and $n \geq m^2/12$.

The property of being a superconcentrator is a little too strong to be useful in deriving lower bounds on the complexity of interesting functions. However, there are weaker properties which still require $O(m^2)$ vertices. Let $G = (V,E)$ be an acyclic directed graph with $m$ numbered inputs $V_1,V_2,\ldots,V_m$ and $m$ numbered outputs $V'_1,V'_2,\ldots,V'_m$. $G$ is said to have the shifting property if, for any $k$ in the range $1 \leq k \leq m$, any subset of $k$ sources $\{v_{i_1},v_{i_2},\ldots,v_{i_k}\}$ such that $1 \leq i_1 < i_2 < \ldots < i_k \leq m$, there are $k$ vertex-disjoint paths joining the set of inputs $\{v_{i_1},v_{i_2},\ldots,v_{i_k}\}$ with the set of outputs $\{v'_{i_1},v'_{i_2},\ldots,v'_{i_k}\}$.

**Theorem 6.** Let $G$ be a planar acyclic directed graph with the shifting property. Then $G$ contains at least $\lceil m/2 \rceil^2/162$ vertices.

**Proof.** Suppose that $G$ contains $n$ vertices. Assign a cost of $1/n$ to each of the first $\lceil m/2 \rceil$ inputs and to each of the last $\lceil m/2 \rceil$ outputs of $G$, and a cost of zero to every other vertex of $G$. Call the first $\lfloor m/2 \rfloor$ inputs and the last $\lfloor m/2 \rfloor$ outputs of $G$ costly. Let $A,B,C$ be a vertex partition satisfying Theorem 1 on $G$ (ignoring edge directions).

Without loss of generality, suppose that $A$ is no more costly than $B$, and that $A$ contains no more costly outputs than costly inputs. Let $A'$ be the set of costly inputs in $A$, $B'$ the set of costly outputs in $B$, $p$ the number of costly inputs and outputs in $C$, and $q$ the number of costly inputs and outputs in $A$. Then $2\lfloor m/2 \rfloor /3 - p \leq q \leq \lfloor m/2 \rfloor - p/2$. Hence $|A'| \geq q/2 \geq \lfloor m/2 \rfloor /3 - p/2$. Also $|A'| \cdot |B'| \geq |A'| \cdot (\lfloor m/2 \rfloor - p - (q - |A'|)) \geq q/2 \cdot (\lfloor m/2 \rfloor - p/2) \geq (\lfloor m/2 \rfloor /3 - p/2)(\lfloor m/2 \rfloor - p - |A'|) \geq 2\lfloor m/2 \rfloor /q - p| A' |/2$.

For $v_i \in A'$, $w_j \in B'$, and $t$ in the range $1 \leq t \leq \lfloor m/2 \rfloor$, call $v_i$, $w_j$, $t$ a match if $j = t$. For every $v_i \in A'$ and $w_j \in B'$, there is exactly one value of $t$ which produces a match; hence the total number of matches for all possible $v_i$, $w_j$, $t$ is $|A'| \cdot |B'| \geq 2\lfloor m/2 \rfloor /q - p| A' |/2$. Since there are only $\lfloor m/2 \rfloor$ values of $t$, some value of $t$ produces at least $2\lfloor m/2 \rfloor /q - p/2$ matches. Thus, for $k = 2\lfloor m/2 \rfloor /q - p/2$, there is some value of $t$ and some set of $k$ inputs $A'' = \{v_{i_1},v_{i_2},\ldots,v_{i_k}\}$ such that $B'' = \{w_{j_1},w_{j_2},\ldots,w_{j_k}\} \subseteq B'$.

Since $G$ has the shifting property, there must be $k$ vertex-disjoint paths between $A''$ and $B''$. But each such path must contain a vertex in $C$ which is neither an input nor an output. Hence $2\sqrt{2/n} - p > 2\lfloor m/2 \rfloor /q - p/2$, and $n \geq m^2/162$.
Corollary 2. Any planar shifting circuit has at least \(\lfloor m/2 \rfloor^2/162\) vertices.

Proof. Any shifting circuit has the shifting property. See [23,24]. □

Corollary 3. Any planar circuit for computing Boolean convolution has at least \(\lfloor m/2 \rfloor^2 / 162\) vertices.

Proof. A circuit for computing Boolean convolution is a shifting circuit if we regard \(x_1, \ldots, x_m\) as the primary inputs and \(z_{i,j}, \ldots, z_{m,1}\) as the outputs. □

Corollary 4. Any planar circuit for computing the product of two \(m\times m\) Boolean matrices has at least \(\lfloor m/2 \rfloor^2 / 162\) vertices.

Proof. A circuit for multiplying two \(m\)-bit binary integers is a shifting circuit. □

The last result of this section is an \(\Omega(m^2)\) lower bound on the size of any planar circuit for multiplying two \(m \times m\) Boolean matrices. We shall assume that the inputs are \(x_{i,j}\) for \(1 \leq i, j \leq m\) and the outputs are \(z_{i,j}\) for \(1 \leq i \leq m\). The circuit computes \(Z = X \cdot Y\), where \(Z = (z_{i,j})\), \(X = (x_{i,j})\), and \(Y = (y_{i,j})\). We use the following property of circuits for multiplying Boolean matrices, called the matrix concentration property [23,24]. For any \(k\) in the range \(1 \leq k \leq n\), any set \(\{x_{i,j} : 1 \leq i \leq k\}\) of \(k\) inputs from \(X\), and any permutation \(\sigma\) of the integers one through \(n\), there exist \(k\) vertex-disjoint paths from \(\{x_{i,j} : 1 \leq i \leq k\}\) to \(\{y_{i,j} : 1 \leq i \leq k\}\) for \(1 \leq j \leq n\).

Similarly, for any \(k\) in the range \(1 \leq k \leq n\), any set \(\{y_{i,j} : 1 \leq j \leq k\}\) of \(k\) inputs from \(Y\), and any permutation \(\sigma\) of one through \(n\), there exist \(k\) vertex-disjoint paths from \(\{y_{i,j} : 1 \leq j \leq k\}\) to \(\{z_{i,j} : 1 \leq i \leq k\}\).

Theorem 7. Any planar circuit for multiplying two \(m \times m\) Boolean matrices contains at least \(m^2\) vertices, for some positive constant \(c\).

Proof. This proof is somewhat involved, and we make no attempt to maximize the constant factor. Suppose \(G\) contains \(n\) vertices, and that \(m\) is even.

Assign a cost of \(1/(km^2)\) to each input \(x_{i,j}\) and each output \(z_{i,j}\), a cost of \(1/(k^2 m^2)\) to each output \(y_{i,j}\), and a cost of zero to every other vertex. There is a partition \(A, B, C\) of the vertices of \(G\) such that neither \(A\) nor \(B\) has total cost exceeding \(1/2\); no edge joins a vertex in \(A\) with a vertex in \(B\) and \(C\) contains no more than \(2/\sqrt{n}/(1 - 2/\sqrt{2}) = c_1 \sqrt{n}\) vertices. This is a corollary of Theorem 1; see [14]. Without loss of generality, suppose that \(B\) contains no fewer outputs than \(A\), and that \(A\) contains no fewer inputs \(x_{i,j}\) than inputs \(y_{i,j}\). Then \(B\) contains at least \((m^2 - c_1 \sqrt{n})/2\) outputs, which contribute at least \(1/4 - c_1 \sqrt{n}/(km^2)\) to the cost of \(B\). Thus inputs contribute at most \(1/4 - c_1 \sqrt{n}/(km^2)\) to the cost of \(B\), and \(B\) contains at most \(m^2 - c_1 \sqrt{n}\) inputs. A contains at least \(2m^2 - (m^2 - c_1 \sqrt{n}) - c_1 \sqrt{n}\) inputs, of which at least \(m^2/2 - c_1 \sqrt{n}\) are inputs \(x_{i,j}\). One of the following cases must hold.

Case 1. A contains at least \(7m^2/5\) inputs \(x_{i,j}\). Let \(p\) be the number of columns of \(X\) which contain at least \(km/7\) elements of \(A\). Then \(p^m - (p-1)(km/7) \geq 3m^2/5\), and \(p \geq m/7\). Let \(q\) be the number of columns of \(Z\) which contain at least \(km/9\) elements of \(B\). Then \(q^m - (q-1)(km/9) \geq m^2/2 - c_1 \sqrt{n}/2\) and \(q \geq m/10\). Hence assume \(q > 0\). Then all columns in \(B\) must contain at least \(m/10\) elements of \(Y\), and \(m/2\) columns of \(Z\) which contain the most elements in \(B\).

Subcase 2a. \(B\) contains at least \(3m^2/10\) elements in \(B\). Let \(p\) be the number of columns of \(X\) which contain at least \(km/9\) elements of \(A\). Then \(p^m - (p-1)(km/9) \geq m^2/2 - c_1 \sqrt{n}\), and \(p \geq m/10\). Let \(q\) be the number of columns of \(Z\) which contain at least \(km/7\) elements of \(B\). Then \(q^m - (q-1)(km/7) \geq m^2/20\), and \(q \geq m/30\). A proof similar to that in Case 1 shows that \(n \geq c m^2\) for some positive constant \(c\).

Subcase 2b. \(B\) contains fewer than \(3m^2/10\) elements in \(B\). Then the \(m/2\) columns of \(Z\) not in \(B\) must contain at least \(m^2/2 - c_1 \sqrt{n}/2\) elements in \(B\). Let \(q\) be the number of columns of \(Z\) not in \(B\) which contain at least \(m/10\) elements of \(B\). Then \(q^m - (q-1)(m/10) \geq m^2/20\), and \(q \geq m/10\). A proof similar to that in Case 1 shows that \(n \geq c m^2\) for some positive constant \(c\).
Z must contain at least $m/10$ elements in $B$.

Let $p$ be the number of columns of $Y$ which contain at least $m/25$ elements of $A$. Then

$$p + (m-p)m/25) \geq 2cm/5 - 2c\sqrt{n},$$

and

$$p \geq 5m/8 - 2c\sqrt{n}/12m.$$ 

For any input $y_{ij}$ of $A$ and integer $t$ in the range $-n \leq t \leq n-1$, call $y_{ij}, t$ a match if

$$t_1(t, y_{ij}) \in B.$$ 

By the previous computations, there are at least $2m^2/5 - 5c_1\sqrt{n}/(9m) + 3m/8 - 5c_1\sqrt{n}/(12m) - m/1250 + 5c_1\sqrt{n}/(1250) + m/25 - c_1\sqrt{n}/12m.$$ 

Such that $y_{ij}$ contains $m/10$ elements of $A$ and $z_{ij}$ contains $m/10$ elements of $B$. Each such column produces $m^2/250$ matches; thus the total number of matches is at least $m^2/5250 - mc_1\sqrt{n}/250$.

Since there are only $2m-1$ values of $t$, some value of $t$ produces at least $k = m^2/1250 - c_2\sqrt{n}/500$ matches. Since $G$ has the matrix concentration property, this set of matches corresponds to a set of $k$ elements in $Y\cap A$ and a set of $k$ elements in $Y\cap B$ which must be joined by $k$ vertex-disjoint paths. Each such path must contain a vertex in $C$.

Thus $k \leq c_1\sqrt{n}$, which means

$$m^2/(1250(c_1 + c_2/500))^2 \leq n.$$ 

In all cases $n \geq cm$ for some positive constant $c$. Choosing the minimum $c$ over all cases gives the theorem for every $m$. The theorem for odd $m$ follows immediately.

The bounds in Theorems 5-7 and Corollaries 2-4 are tight to within a constant factor. We leave the proof of this fact as an exercise.


Let $G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$ be undirected graphs. An embedding of $G_1$ in $G_2$ is a one-to-one map $\beta : V_1 \rightarrow V_2$. The worst-case proximity of the embedding is $\max \{d(v,\beta(v)) : (v,w) \in E_1\}$, where $d(x,y)$ denotes the distance between $x$ and $y$ in $G_2$. The average proximity of the embedding is

$$\frac{1}{|E_1|} \sum \{d(\beta(v),\beta(w)) : (v,w) \in E_1\}.$$ 

These notions arise in the following context. Suppose we wish to represent some kind of data structure by another kind of data structure, in such a way that if two records are logically adjacent in the first data structure, their representations are close together in the second. We can model the data structures by undirected graphs, with vertices denoting records and edges denoting logical adjacencies. The representation problem is thus a graph embedding problem in which we wish to minimize worst-case or average proximity. See [5,13,19] for research in this area.

Theorem 8. Any planar graph with maximum degree $k$ can be embedded in a binary tree so that the average proximity is a constant depending only on $k$.

Proof. Let $G$ be an $n$-vertex planar graph. Embed $G$ in a binary tree $T$ by using the following recursive procedure. If $G$ has one vertex $v$, let $T$ be the tree of one vertex, the image of $v$. Otherwise, apply Corollary 1 to find a partition $A,B,C$ of the vertices of $G$. Let $G_1$ be any subgraph of $G$ (if $C$ is empty, let $v$ be any vertex). Embed the subgraph of $G$ induced by $A\cup C[v]$ in a binary tree $T_1$ by applying the method recursively. Embed the subgraph of $G$ induced by $B$ in a binary tree $T_2$ by applying the method recursively. Let $T$ consist of a root (the image of $v$) with two children, the root of $T_1$ and the root of $T_2$. Note that the tree $T$ constructed in this way has exactly $n$ vertices.

Let $h(n)$ be the maximum depth of a tree $T$ of $n$ vertices produced by this algorithm. Then

$$h(n) \leq g(n) \text{ if } n \leq g(n),$$

$$h(n) \leq h(2n/3 + 2\sqrt{2n/3} - 1) \leq h(2n/3 + 2\sqrt{2n/3})$$

if $n > g(n)$.

It follows that $h(n)$ is $O(\log n)$.

Let $G = (V,E)$ be an $n$-vertex graph to which the algorithm is applied, let $G_1$ be the subgraph of $G$ induced by $A\cup C$, and let $G_2$ be the subgraph induced by $B$. If $s(G) = s(G_1) + s(G_2)$ plus a small correction term, then $s(G) = \log n$ if $n > g(n)$, and $s(G) = s(G_1) + s(G_2) + O(\log n)$ if $n > g(n) + 1$. This follows from the fact that any edge of $G$ not in $G_1$ or $G_2$ must be incident to a vertex of $C$.

If $s(n)$ is the maximum value of $s(G)$ for any $n$-vertex graph $G$, then

$$s(n) \leq \max [s(1) + s(n-1) + n\log n]$$

$$n/3 - 2n/3 + 2\sqrt{2n/3} \leq k s(n/2 + 2\sqrt{2n/3})$$

if $n > 1$, for some positive constant $c$.

An inductive proof shows that $s(n)$ is $O(n\log n)$.

If $G$ is a connected $n$-vertex graph embedded by the theorem, then $G$ contains at least $n-1$ edges, and the average proximity is $O(1)$. (If $G$ is not connected, embedding each connected component separately and combining the resulting trees arbitrarily achieves an $O(k)$ average proximity.)

It is natural to ask whether any graph of bounded degree can be embedded in a binary tree with $O(1)$ average proximity. (Graphs of unbounded degree cannot be so embedded; the star of Figure 2 requires $O(n\log n)$ proximity.) Such is not the case, and in fact the property of being embeddable in a binary tree with $O(1)$ average proximity is closely related to the property of having a good separator.

To make this statement more precise, let $S$ be a class of graphs. The class $S$ has an $f(n)$-separator theorem if there exist constants $a < 1$, $b > 1$ such that the vertices of any $n$-vertex graph in $S$ can be partitioned into three sets $A,B,C$ such that $|A|,|B| \leq an$, $|C| \leq bn$, and no vertex in $A$ is adjacent to any vertex in $B$.

Let $S$ be any class of graphs of bounded degree closed under the subgraph relation (i.e., if $G \in S$ and $G_1 \subset G$, then $G_1 \in S$). Suppose $S$ satisfies an $n^g(n)/\log n$-separator theorem for
some non-decreasing function \( g(n) \). Using the idea in the proof of Theorem 8, it is not hard to show that any graph in \( S \) can be embedded in a binary tree with \( O(g(n)) \) average proximity. Conversely, suppose any graph in \( S \) can be embedded in a binary tree with \( O(g(n)) \) average proximity. Then \( S \) satisfies an \( ag(n)/\log n \) separator theorem. In particular, if \( S \) satisfies no \( o(n) \)-separator theorem, then embedding the graphs of \( S \) in binary trees requires \( O(g(n)) \) average proximity. Erdős, Graham, and Szemerédi [7] have shown the existence of a class of graphs of bounded degree having no \( o(n) \)-separator theorem.

7. The Post Office Problem.

In [11], Knuth mentions the following problem: given \( n \) points (post offices) in the plane; determine, for any new point (house), which post office it is nearest. Any preprocessing of the post offices is allowed before the houses are processed. Shamos [22] gives an \( O(\log n) \)-time, \( O(n^{3}) \)-space algorithm and an \( O((\log n)^{2}) \)-time, \( O(n \log n) \)-space algorithm. See also [6]. Using Theorem 2 we can give a solution which requires \( O(\log n) \) time and \( O(n) \) space, both minimum if only binary decisions are allowed.

A polygon is a connected, open planar region bounded by a finite set of line segments. (For convenience, we allow the point at infinity to be an endpoint of a line segment; thus a line is a line segment.) A polygon partition of the plane is a partition of the plane into polygons and bounding line segments. A triangulation of the plane is a polygon partition, all of whose polygons are bounded by three line segments. A triangulation of a polygon partition is a refinement of the partition into a triangulation. Two polygons in a polygon partition are adjacent if their boundaries share a line segment. A set of polygons is connected if any two polygons in the set are joined by a sequence of adjacent polygons.

We shall solve the following triangle problem: given \( n \) points (post offices) in the plane; determine which triangle or line segment of the triangulation contains the point. The post office problem can be reformulated as triangle problem; the set of points closest to each post office forms a connected set of points which we do not prove.

Lemma 1. Any \( n \)-polygon partition has a refinement whose total number of triangles is bounded by \( n \) plus the number of line segments bounding non-triangles plus a constant (a line segment bounding two non-triangles counts twice in this bound).

We shall build up a sequence of more and more complicated (but more and more efficient) algorithms, the last of which is the desired one.

Theorem 2. Given an \( O(\log n) \)-time, \( O(n^{2+\varepsilon}) \)-space algorithm for the triangle problem with \( \varepsilon > 0 \), one can construct an \( O(\log n) \)-time, \( O(n^{2+\varepsilon}/5) \)-space algorithm.

Proof. Let \( T \) be a triangulation and \( v \) be a vertex for which the triangle problem is to be solved. By Theorem 2 there is a set of \( O(n^{2}/5) \) triangles \( C_{0} \) whose removal from \( T \) leaves no connected set of more than \( O(n^{2}/5) \) triangles.

Merge pairs of adjacent triangles which are not in \( C_{0} \) to form a polygon partition \( P_{1} \). \( P_{1} \) contains at most \( O(n^{2}/5) \) line segments. Find a triangulation \( T_{0} \) of \( P_{1} \) with \( O(n^{2}/5) \) triangles. Using an \( O(\log n) \)-time, \( O(n^{1+2\varepsilon/3}) \)-space algorithm, determine which triangle or line segment of \( T_{0} \) contains \( v \).

If \( v \) is in some triangle of \( C_{0} \), the problem is solved. Otherwise \( v \) is known to be in some connected set \( C_{1} \) of triangles in \( T \) minus \( C_{0} \).

Merge pairs of adjacent triangles which are not in \( C_{1} \) to form a polygon partition \( P_{2} \). Each line segment bounding a non-triangular polygon of \( P_{2} \) must bound a triangle of \( C_{0} \). Thus there is a triangulation \( T_{1} \)
of $F_1$ containing $|c_1| + O(n^{3/5})$ triangles. Apply the algorithm recursively to discover which triangle of $F_1$ contains $v$. This solves the problem.

The sets $C_1$, polygon partitions $P_1$, and triangulations $T_1$ are all precomputed. If $t(n)$ is the worst-case time required by the algorithm on an n-triangle triangulation, then

$$t(n) = O(1) \quad \text{if } n \leq n_0,$$

$$t(n) = O(n^{1/5}) + O(\log n) \quad \text{otherwise}.$$  

An inductive proof shows that $t(n)$ is $O(\log n)$ if $n_0$ is chosen sufficiently large.

If $s(n)$ is the worst-case storage space required by the algorithm on an n-triangle triangulation, then

$$s(n) = O(1) \quad \text{if } n \leq n_0,$$

$$s(n) \leq n^{7/10} + \max \left[ \sum a_i + O(n^{3/5}) \right] \left[ \sum n_i \leq n \right. \quad \text{and} \quad ^{2/5} \leq n_i \leq c_2 n^{2/5} \right]$$

for some positive constants $c_1$ and $c_2$.

An inductive proof shows that $s(n)$ is $O(n)$.

The preprocessing time required by the algorithm of Theorem 10 is $O(n \log n)$. See [22]. We do not advocate this algorithm as a practical one, but its existence suggests that there may be a practical algorithm with an $O(\log n)$ time bound and $O(n)$ space bound.

5. Other Applications.

As illustrated in this paper, Theorem 1 and its corollaries have many interesting applications, and the paper does not exhaust them. We have obtained two additional results which require fuller discussion than is possible here. One is the application of Theorem 1 to Gaussian elimination. George [8] has proposed an $O(n \log n)$-space, $O(n^{3/5})$-time method of carrying out Gaussian elimination on a system of equations whose sparsity structure corresponds to a $\sqrt{n} \times \sqrt{n}$ square grid. We can generalize his method so that it applies to any system of equations whose sparsity structure corresponds to a planar or almost-planar graph. Such systems arise in the solution of two-dimensional finite-element problems [15]. We shall discuss this application in a subsequent paper; we hope that it will prove of practical, as well as theoretical, value.

Another application involves the power of non-determinism in one-tape Turing machines. We can prove that any non-deterministic $t(n)$-time-bounded one-tape Turing machine can be simulated by a $t(n)^7$ alternating one-tape Turing machine with a constant number of alternations, where $\gamma < 1$ is a suitable constant and $t(n)$ satisfies certain reasonable restrictions. Alternation generalizes the concept of non-determinism and is discussed in [3,12]. Our result strengthens Paterson's space-efficient simulation of one-tape Turing machines [17].

References.


Figure 1. Elimination of a crossover by use of three "exclusive or" gates. Reference [9] contains a crossover circuit which uses only "and" and "not".

Figure 2. A star.