On Nontrivial Separators for k-Page Graphs and Simulations by Nondeterministic One-Tape Turing Machines

Zvi Galil, Ravi Kannan, Endre Szemeredi

Technical Report
CUCS-301-87

On Nontrivial Separators for k-Page Graphs and Simulations by Nondeterministic One-Tape Turing Machines

Zvi Galil*
Columbia University
and Tel-Aviv University

Ravi Kannan**
Carnegie Mellon University

Endre Szemerédi
University of Chicago
and the Hungarian Academy of Sciences

* The work of the first author was supported in part by NSF Grants MCS-83-03139 and DCR-85-11713.

** The work of the second author was supported in part by NSF Grants MCS-84-16190.

Abstract

We show that the following statements are equivalent :

Statement 1: 3-pushdown graphs have sublinear separators.

Statement 1*: k-page graphs have sublinear separators.

Statement 2: A one-tape nondeterministic Turing machine can simulate a two-tape machine in subquadratic time.

None of the statements is known to be true or false at present. However our proof of equivalence is quantitative - it relates exactly the separator size of the two kinds of graphs to the running time of the simulation in Statement 3. Using this equivalence we derive several graph-theoretic corollaries.

There are known examples where upper bounds on graph properties imply upper bounds on computation time or space. There are other examples where lower bounds on graph properties are used to derive lower bounds on computation time in restricted settings. However, our results may constitute the first example where a graph problem is shown to be **equivalent** to a problem in computational complexity.

In a companion paper we construct graphs and prove a lower bound or their separators. Using the equivalence we prove an almost linear lower bound for the size of separators for 3-pushdown graphs and an almost quadratic lower bound for simulating two-tape nondeterministic Turing machines by one-tape machines. Specifically, for an integer s let $l^s(n)$, the s-iterated logarithm function, be defined inductively: $l^0(n) = n$, $l^{s+1}(n) = \log_2(l^s(n))$ for $s \geq 0$. Then:

— For every fixed s and all n, there is an n-vertex 3-pushdown graph whose smallest separator contains at least $\Omega(n/l^s(n))$ vertices.

— There is a language L recognizable in real time by a two-tape nondeterministic Turing machine, but every on-line one-tape nondeterministic Turing machine that recognizes L requires $\Omega(n^2/l^s(n))$ time for any positive integer s.

0. Introduction

Let $S : N \rightarrow N$ be a monotone increasing function. An n-vertex graph $G = (V, E)$ (directed or undirected) has an **S-separator** C if there is a partition $V = A \cup B \cup C$, $|A|, |B| \geq n/3$, $|C| \leq S(n)$ and $E \cap (A \times B) = \phi$. A family of graphs is **S-separable** if every graph in the family has an S-separator. A family is **separable** if it is S-separable for some $S(n) = o(n)$.

For convenience, we restict attention to **nice** functions S. A function S is nice if for every a, $0 < a < 1$, there is b, $a < b < 1$, such that $aS(n) < S(an) < bS(n)$.

Remark 1. The planar separator theorem [11] can be restated as follows: the family of planar graphs is $O(\sqrt{n})$-separable.

Unless specified otherwise we will deal with graphs of constant degree. They always have linear separators which we call trivial separators.

Remark 2. The nonexistence of nontrivial separators is closely related to expansion properties in graphs. For a graph $G = (V, E)$ and $A \subseteq V$, $\Gamma_G(A)$ is the set of neighbors of A. A family of graphs is expanding (with expansion constant d) if for every n-vertex graph $G = (V, E)$ in the family and every $A \subseteq V$ $|\Gamma_G(A) - A| \geq d|A||A^c|/n$. It follows that every family of expanding graphs is nonseparable. Since expanding graphs "expand" also small sets (that contain less than one third of the vertices), the converse is not necessarily true.

Outerplanar graphs are graphs that can be embedded in the plane so that all vertices lie on the outer face; equivalently, such a graph can be embedded on the plane so that all vertices lie on one straight line and all edges can be embedded on one of the half planes defined by the line. Formally, an **outerplanar** graph is a graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$ for some n and $E = S \cup R$, where the **spine** $S \subseteq \{(i, i+1) | \ i = 1, \ldots, n-1\}$ and in R edges do not cross: specifically for each pair of edges in R $(i_1, j_1), (i_2, j_2)$ with $i_1 < i_2 < j_1$ we have $j_2 \leq j_1$. A **k-page graph** is a graph which consists of k outerplanar graphs sharing the same spine. k-page graphs can be considered as undirected graphs or as directed graphs where an edge always goes from a smaller- to a larger-numbered vertex. If every vertex has at most one incident edge in each page of a k-page graph, the graph is called a **k-pushdown graph** (or **k-pd graph** in short). We use a slightly broader definition of 3-pd graphs by allowing some of the spine edges to be missing. As a result this family of 3-pd graphs is closed under containment (i.e. if G is in the family then so are all the subgraphs of G).

Obviously, a 2-page graph is planar. Conversely, it was shown in [3] that every planar graph can be embedded in eight pages. The number has been improved to seven [6] and very recently Yannakakis [17] improved it to four and showed that four pages are necessary. Computation graphs of Turing machines are k-pd graphs, where k depends on the number of tapes of the Turing machine. This has been the reason for substantial interest in such graphs. In [13] it was shown that k-pd graphs (considered as directed graphs) contain nontrival **segregators**. This graph property was used to show that nondeterministic multitape Turing machines are strictly more powerful than their deterministic counterparts, settling a long-standing open problem. A family of directed graphs contains a non-trivial segregator if every n-vertex graph in the family contains a set of $o(n)$ vertices (the segregator) whose deletion leaves each remaining vertex with at most $o(n)$ (not necessarily

immediate) predecessors in the remaining graph. It is quite easy to show that if a family of directed graphs is closed under containment has a nontrivial separator then it has a nontrivial segregator.

k-page graphs also arise in connection with embedding of VLSI circuits [4] and fault tolerant arrays of processors [14]. Intuitively, k-page graphs can be drawn on a "book" with k "pages" with all vertices placed on the "binding", all edges placed on the pages, and no two edges on a page crossing. For this reason, the minimum k for which a graph is k-page embeddable is called the page number of the graph [3]. See also [1],[15] for a discussion of outerplanar graphs.

The following problems are open:

Problem 1. Is the family of 3-pd graphs separable?

Problem 1*. Is the family of k-page graphs separable for any $k \geq 3$?

In [8], the second author showed that for any fixed $k \geq 3$, the family of k-page graphs is separable if and only if the family of 3-page graphs is. The equivalence of problems 1 and 1* (derived in Corollary 5 below) is slightly stronger since Problem 1 deals with 3-pd graphs, which are special 3-page graphs.

Remark 3. One can similarly ask if there is an expanding family of 3-pd graphs (or k-page graphs).

We relate these problems to an open problem from an entirely different domain. Consider a real-time nondeterministic Turing machine with two working tapes and a separate input tape, or 2NTM. (By real-time we mean that the machine reads a new symbol each step.) We want to simulate it by an on-line one-tape nondeterministic Turing machine, or 1NTM. By on-line we mean that the additional input tape is one way. We refer to this as **the simulation.** It is well known that the simulation can be done in time $O(n^2)$. But the following problem in still open:

Problem 2. Can the simulation be done in subquadratic ($o(n^2)$) time?

Remark 4. In the deterministic case the answer is negative, and was proved by Maass [12]. In the nondeterministic case Maass [12] proved a lower bound of $\Omega(n^2/(\log^2 n \log\log n))$ for the time of the simulation. Li [9] claimed a better bound of $\Omega(n^2/(\log n \log\log n))$.

In Section 1 we show that if 3-pd graphs have small separators then one can derive a fast simulation:

Theorem 1. If the family of 3-pd graphs is S-separable then there is a simulation of time $t(n) = O(S(n/\log n)n \log n)$.

Corollary 1. If $S(n) = o(n)$ then $t(n) = o(n^2)$.

Corollary 1 states that if the answer to Problem 1 is positive, then so is the answer to Problem 2.

The connection between small separators and upper bounds is due to Li [10]. Theorem 1 can be viewed as a generalization of Li's result that a 2-pd nondeterministic machine can be simulated by a 1NTM in time $O(n^{1.5}\sqrt{\log n})$. Li's result follows from Theorem 1 by considering 2-pd graphs (which are planar and thus $S(n) = O(\sqrt{n})$).

In Section 2 we introduce families F_k of graphs. Informally, an n-vertex graph G in F_k is defined by a string x_G of length at most kn. x_G is a sequence of instructions for a 3-pushdown machine M. (Such a machine has three pushdown stores as its work tapes.) M manipulates the stack symbols $1,\ldots,n$ (the vertices of G). Initially, each of the three pushdown stores of M contain $1,\ldots,n$. M has two kinds of states: regular states and special states. If $i_1, i_2, i_3, i_4, \ldots$ is the sequence of vertices popped in a special state, the edges of G are $(i_1, i_2), (i_3, i_4), \ldots$ One can easily show that every k-page graph is in F_{6k}. We define a language L which consists of strings associated with graphs in $\cup_k F_k$. L is recognizable in real time by a 2NTM. We show:

Theorem 2. Assume M' is an on-line 1NTM that accepts L in time $t(n)$ and $k > 0$. Then there is a constant $c = c(M')$ such that if $t(n) \leq cn^2/k$, then F_k is S_k-separable, where $S_k(n) = O(k^2 t(n) \log(n^2/kt(n))/n)$.

Theorem 2 can be considered a generalization of Maass' result [12]. Maass defined a language associated with very simple graphs, the doubling graphs, all of which are in F_6. The connection between the graphs and the language is much simpler in Maass' proof. Maass proved implicitly a lower bound of $\Omega(n/\log n)$ on the separator of the family of doubling graphs. Theorem 2 yields an $\Omega(n^2/(\log n \log\log n))$ lower bound for the simulation which is Li's claimed improved lower bound for Maass' language. Li [10] showed that Maass' language can be recognized in time $O(n^2 \log\log n/\sqrt{\log n})$, because the family of doubling graphs has a nontrivial separator. Hence, if the answer to Problem 2 is negative, Maass' language cannot be used to prove it. Our proof of Theorem 2 uses some ideas introduced by Maass, mainly Maass' sophisticated use of Kolmogorov complexity.

Corollory 2. If $t(n) = o(n^2)$ then $S_k(n) = o(n)$ for every k.

In particular ($k = 3$) if the answer to Problem 1 is negative so is the answer to Problem 2. Hence,

Corollary 3. Problems 1 and 2 are equivalent.

Corollary 4. If the answer to Problem 1 is negative, then L requires time $\Omega(n^2)$.

In the case that S in Theorem 1 (S_k in Theorem 2) is not nice, the corresponding theorem should be slightly modified (the corresponding expressions are uglier). But corollaries 1–4 still hold.

Several examples are known of a graph property which implies a theorem concerning computation. One example of such a property is the existence of nontrivial segregators mentioned above. Another well known example is from [7], where it was first shown how to pebble an n-vertex directed acyclic graph of constant indegree with $O(n/\log n)$ pebbles. and then this was used to prove that "space is better than time". In both examples an upper bound on a graph property implied an upper bound on time or space complexity. In both cases it is unlikely that the converse theorem holds. (The differences between space and time and between nondeterminism and determinism are believed to be exponential.) On the other hand, there are examples where lower bounds on the sizes of graphs satisfying certain connectivity properties imply a lower bound on time for certain types of

computations [16]. However, our results may constitute the first example where a graph problem is shown to be **equivalent** to a problem in computational complexity.

The following corollories are easily obtained by using Theorem 1 and 2. They state properties of graphs and they are proved via a detour through Turing machines. All of them probably have direct proofs.

Corollary 5. For every k, F_k is separable if and only if the family of k-page graphs is separable if and only if the family of 3-pd graphs is separable.

The next corollary deals with different definitions of a separator and separability. For $0 < \alpha < 1$, let us define an (α, S)-separator as we defined an S-separator except that $|A|, |B| \leq \alpha n$. Thus, an S-separator is a $(\frac{1}{3}, S)$-separator.

Corollary 6. All the results above hold if we replace S-separators by (α, S)-separators. In particular, k-page graphs are S-separable if and only if they are (α, S)-separable for some $0 < \alpha < 1$.

The **bandwidth** of a graph $G = (V, E)$ with respect to the naming $V = \{1, \ldots, n\}$ is $\sum_{(i,j)\in E} |i - j|$. The bandwidth of a graph is the minimum bandwidth with respect to all possible namings. The proof of Theorem 1 uses only the fact that the existence of an $o(n)$ separator implies that the bandwidth is $o(n^2)$. The latter is used to derive a fast simulation. Consequently,

Corollary 7. k-page graphs have nontrivial separators if and only if they have a subquadratic bandwidth.

One can define a different family of graphs F'_k by using a sequence of instructions x'_G for a 2-pd machine (instead of a 3-pd machine). Theorem 2 still holds for the corresponding language L' and the corresponding bounds on the separators $S'_k(n)$. But L' is recognizable in real time by a 2-pd machine. Also, one observes that the simple grid is in F'_6, which implies that $S'_6(n) = \Omega(\sqrt{n})$. From this version of Theorem 2 with $k = 6$ one obtains the following corollary which is due to Li [9],[10].

Corollary 8. There is a language L' which is recognizable in real time by a deterministic 2-pd machine such that every on-line 1NTM that accepts L' requires $\Omega(n^{1.5}/\sqrt{\log n})$ time.

Let $l^s(n)$, the s-iterated logarithm function, be defined inductively: $l^0(n) = n$, $l^{s+1}(n) = \log_2(l^s(n))$ for $s \geq 0$. In a companion paper [5] we define n-vertex graphs G^n_k in F_{6k} and derive a lower bound on the sizes of their separators:

Theorem 3. $S_{6k}(n) \geq \Omega(n/k10^{2k(k+1)}l^k(n)))$.

By combining theorems 2 and 3 we get:

Corollary 9. For any positive integer s, the time of the simulation is at least $\Omega(n^2/l^s(n))$.

It is possible to let s grow as a function of n and derive slightly stronger versions of Theorem 3 and Corollary 9. These results yield the currently best lower bound for the simulation.

The graphs G_k^n are not k-page graphs. But one can modify them to get k-pd graphs that have large separators.

Corollary 10. There are n-vertex k-pd graphs whose smallest separators have at least $\Omega(n/(2^k k 10^{2k(k+1)} l^k(n)))$ vertices.

A stronger version of Corollary 10 can be proved by combining Theorems 1 and Corollary 9:

Corollary 11. For every positive integer s and all n, there is an n-vertex 3-pd graph whose smallest separator contains at least $\Omega(n/l^s(n))$ vertices.

Corollary 11 means that the family of 3-pd graphs (and thus F_{18}) is a family of graphs that is hardest to separate.

1. From a Small Separator to a Fast Simulation

In this section we prove Theorem 1, showing that if the family of 3-pushdown graphs is separable then a fast simulation exists. Any nondeterministic two-tape machine can be simulated in real time by a nondeterministic three-pushdown machine [2]. The computation graph of the latter is a 3-pd graph that by our assumption has a small separator. It follows that the vertices of the graph can be embedded on one tape so that the sum over the edges of the graph of the distance on the tape between the embedded endpoints of each edge is relatively small. This total distance is the most expensive part of the simulation by a one-tape machine as this simulation mostly traverses the edges of the computation graph.

We consider a nondeterministic real-time 3-pd machine M. Given input x of length n accepted by M, we fix an accepting computation C of M on x.

Let $b = \lceil \log_2 n \rceil$. A **$b$-partition** of the pd tape is a partition of the tape into consecutive blocks, the first of size $s \leq b$ and the others of size exactly b.

Proposition 1. There is a b-partition of each pd tape such that during the computation, the total number of times heads of M cross a block boundary of any pd tape is at most $3n/b$.

Proof: Choose s for each partition so that the corresponding number of crossings is minimized. □

We fix a partition as in the proposition. We define a **time interval** as the time that passes between two consecutive crossings of boundaries by any pd head. By Proposition 1, the number of time intervals $\tilde{n} \leq 3n/b$. We define the following graph $G = (V, E)$; $V = \{1, \ldots, \tilde{n}\}$, $E = S \cup R$ where $S = \{(i, i+1) | 1 \leq i < \tilde{n}\}$, $R = R_1 \cup R_2 \cup R_3$, and for $r = 1, 2, 3$ $R_r = \{(i, j) |$ there is a block on pd number r on which M pushes a symbol

in time interval i and the first time interval after i that M pops from this block is time interval j}. Obviously G is a 3 pd-graph.

Let $G = (V, E)$ be a 3-pd graph. For $D \subseteq V$ let $G_D = (D, E \cap D \times D)$. Obviously G_D is a 3-pd graph. We define a **good ordering** of the vertex set V of such a G recursively. Since by hypothesis G has an S-separator, $V = A \cup B \cup C$ as in the definition of an S-separtor. The good ordering is any ordering of C, following a good ordering of A, following a good ordering of B. The last two are defined because G_A and G_B are 3-pd graphs.

We describe a nondeterministic one-tape machine M' that simulates M. During the simulation M' may guess several things. We can assume that it always guesses correctly, because bad guesses either lead to failure (the machine stops) or to a waste of time, but never to accepting an input not accepted by M.

First, M' guesses a good ordering of V, the vertex set of G. Each vertex will be represented by a record – a block of size b on the work tape – which has seven tracks. The first track contains the number i of the vertex. This number is guessed when M' guesses the good ordering of M'. Tracks 2-4 will contain the heights of the pd's in time interval i. Heights are measured in number of blocks. Tracks 5-7 will be used to store the contents of the top blocks of the three pd's in time interval i.

After guessing the good ordering, M' nondeterministically finds on its tape the record of vertex number 1. It nondeterministically marks in tracks 5-7 the end of the first block of each pd. (Recall that the first block may have any length up to b.) It then simulates M until either M stops (in which case M' stops), or M tries to leave the first block of one of the pd's.

We now assume that M' has already simulated time interval $j-1$ and show how it simulates time interval j. M' keeps track of the state of M. The head of M' is now scanning the record of vertex $j-1$. M' nondeterministically finds the record of vertex j and copies tracks 2-7 from the record of vertex $j-1$ to those of the record of vertex j. By comparing the first track, M' verifies that it has copied these tracks to the right record. Among the three blocks in tracks 5-7 there may be some that the simulated head of M did not try to leave at the end of time interval $j-1$. Each such block is left unchanged, as well as the corresponding height (in track 2, 3 or 4). At least one head tried to leave its block. (That is why time interval $j-1$ ended.) We describe next what M' does if a head tried to leave the block of pd 1 (on track 2). M' acts similarly in each case of a head trying to cross a block boundary.

If M tried to cross a right boundary, pushing a symbol on top of the top block, M' cleans track 5, then it increments the height in track 2 and pushes this symbol at the left of track 5. If M tried to cross a left boundary, popping off a symbol from the block that is below the top block (that is now empty), M' acts as follows. M' first decrements the height on track 2. Then it nondeterministically finds a record of another vertex, say $i < j$. It checks if the fifth track of that record is not empty. If it is not empty it does the following: (1) It copies this track to track 5 of the record of vertex j; (2) it erases this track from the record of vertex i; (3) it verifies that the height on track 2 of vertices i and j are the same; if not it stops. Next M' simulates the part of the computation during time interval j, again until either M stops or M tries to cross one of the boundaries.

Obviously, if M accepts x, M' accepts x. To prove the converse, one uses an induction on the time interval j. The induction assumption asserts first that at the first $j-1$ time intervals M' simulated $j-1$ first time intervals of some computation of M. Assume at the end of these $j-1$ time intervals M had n_1, n_2, n_3 blocks respectively in its pd's. We say that block p of pd r of M ($1 \leq r \leq 3$) is represented in record q at certain time, if track number $4+r$ of record q on M''s tape contains the contents of that block at that time. The second part of the induction assumption is that at the end of time interval $j-1$, for $1 \leq r \leq 3$ only block $p \leq n_r$ of pd r is represented in some record; it is represented in exactly one record; this record corresponds to the last time interval M pushed a symbol on this block, and the record contains the contents of that block at the end of that time interval. This hypothesis implies that at the end of time interval $j-1$ the records in tracks $5-7$ represent the contents of the 3 pd's after some computation of the corresponding length, and that M' brings the correct block in case the block above it becomes empty. Hence M' has the entire information it needs for the simulation of time interval j. It follows that the induction assumption holds after the first j time intervals.

Note that the graph G is represented only by the good ordering of V which is important for the time analysis. The edges of the graph appear only implicitly: when M' moves from record $j-1$ to record j it "moves along" a spine edge. When M' brings information from record i to record j in the description above it moves along an edge in R_r for $r = 1, 2, 3$.

To analyze the time bound we assume that M' guessed correctly an accepting computation C of time n and a correct good ordering of V. We also assume that it always made the correct guesses in creating the records, in finding records and in simulating M. We compute the time of M' by considering three parts separately: Initialization and locating the record of vertex 1 takes time $O(n)$. Simulation of a time interval takes $O(b^2)$ for a total of $O(nb)$. The major part is that of transporting records. As we saw above we only move along edges of G and we use each edge once during the simulation. The cost of transporting a record is b times the distance corresponding to the edge (namely, the distance on the tape between the records representing the endpoints of the edge). In this point we exploit the good ordering.

Let $T(m)$ be the maximal time to transport blocks in a graph represented by m records. Assume that of the $\tilde{n}$ records of V $\tilde{n}_1$ correspond to A and $\tilde{n}_2$ to B (A, B and C as above). Then

$$(*) \qquad T(\tilde{n}) \leq T(\tilde{n}_1) + T(\tilde{n}_2) + cS(\tilde{n})\tilde{n}b^2.$$

The additonal term is explained as follows. There are at most $S(\tilde{n})$ vertices in C; hence at most $5S(\tilde{n})$ edges are incident to vertices of C. Their corresponding distance is at most $3\tilde{n}b$. Hence, the contribution of edges incident to C is $O(S(\tilde{n})\tilde{n}b^2)$. (Recall that there are no edges connecting A and B.) Using the fact that S is nice and that $\tilde{n}_1, \tilde{n}_2 \leq 2\tilde{n}/3$ we get by induction that $T(\tilde{n}) \leq dS(\tilde{n})\tilde{n}b^2 = O(S(n/\log n)n\log n)$. Note that Corollary 1 follows from $(*)$ without assuming that S is nice.

2. From a Fast Simulation to Small Separators

In this section we prove Theorem 2, showing that if there is a a simulation (by a machine M') with a certain time bound then the family of k-page graphs has separators of a certain size. In fact we show it for a larger family, which we call F_k. We first define a language L with strings of the form $x\#z\#$, where x is any string and z is a sequence of queries and answers about symbols of x. The sequence corresponds to graphs in the family F_k. In this correspondence positions in x correspond to vertices of the graph and positions of consecutive queries correspond to edges of the graph. We choose such strings $x\#z\#$, where x is (Kolmogorov) random and z corresponds to a graph in F_k which is hardest to separate (i.e. its smallest separator is largest among all graphs in F_k with the corresponding number of vertices). We partition the work tape into regions and define a special region (called Desert in Maass' proof) R such that very few symbols of x were scanned when (the working head of) M' was in R, while for a substantial portion of the symbols of x (the working head of) M' was to the right of R, and for a substantial portion it was to the left of R when it scanned them. Using Kolmogorov complexity we show that there must be many special regions. Next we show that if the graphs in F_k do not have small separators, then the simulation time must be large, obtaining a contradiction. We consider a special region and consider the part when the queries (z) are processed. By the definition of special region and the fact that the separator is large we find that many consecutive queries query symbols x_i and x_j such that when M' scanned x_i it was to the left of R and when M' scanned x_j it was to the right of R. Using Kolmogorov complexity we derive that for most of these queries M' must cross R, yielding the desired lower bound on the time.

We now define the language L and the graph families F_k mentioned above. $L = \{x\#z\#|x \in \{0,1\}^*$ and z is a legal query sequence$\}$. We define legal query sequences below.

The language L is defined in such a way that it is easily recognizable in real time by a 3-pushdown deterministic machine. This virtual machine M first pushes x on each of its three pd's. The string z contains a sequence of instructions for M some of which may query certain symbols of x. z is legal if following each query in z the correct answer appears.

Formally, let $I = \{1,2,3\}$ be the index set of the pd's and let $\Gamma = \{$(pop, push, query)$|$ pop $\in I$, push $\in 2^I$, query $\in \{T,F\}\}$. A **query sequence** is a string over $(\Gamma \cup \{0,1\})^*$. If z is a query sequence, then we denote by $h(z)$ the string in Γ^* resulting from deleting all $0,1$'s from z. A symbol $\gamma = (\gamma_1, \gamma_2, \gamma_3) \in \Gamma$ can be interpreted as follows: pop a symbol from pd number γ_1, push it on top of the pd's listed in γ_2, and if $\gamma_3 = T$, then γ is a query symbol. A 0 or 1 is an answer to a query. Hence, $h(z)$ is the string obtained from z by deleting all the answers to the queries.

A query sequence z is **legal** if (1) $h(z)$ is executable by a 3-pd machine (an empty pd is never popped); and (2) each query symbol is followed by the correct answer (0 or 1), and 0 and 1 may follow only a query symbol. More precisely, assume $z = z_1 z_2 \ldots$ and z_i is a query symbol. The string $u = h(z_1 \ldots z_{i-1} z_i)$ is executable by a 3-pd machine that

pushes and pops symbols of x. Thus z_i pops a specific bit x_r of x where r depends only on u. This bit (the correct answer) must be the same as z_{i+1}. Note that indeed, a 3-pd machine can recognize L easily in real time. Consequently, a 2NTM accepts L in real time [2].

To prove a lower bound for L, it suffices to prove a lower bound for a subset of L. Let $b : N \rightarrow N$ to be specified later. Let $L^b = \{x\#z\# \in L |$ if $|x| = m$, then $b(m)$ divides m and $h(z)$ consists of blocks of the form $\gamma^{b(m)}$, $\gamma \in \Gamma\}$. So, for $y = x\#z\# \in L^b$ with $|x| = m$, x consists of $\tilde{m} = m/b(m)$ blocks of size $b(m)$ and the query sequence z pushes, pops, and queries entire blocks of x (sometimes from left to right and sometimes from right to left).

Let $i_1, i_2, i_3, \ldots$ be the sequence of block numbers of blocks queried in $y = x\#z\# \in L^b$. There is a graph corresponding to y, $G(y) = (V, E)$, where $V = \{1, \ldots, \tilde{m}\}$ and $E = \{(i_1, i_2), (i_3, i_4), \ldots\}$. The graph $G(y)$ depends only on z (actually only on $h(z)$). We denote by $F_k = \{G(y) | y = x\#z\# \in L^b, |y| = k|x|\}$.

Note that one could define L^b by starting with graphs definable by 3-pd machines. The three pd's are initialized with $1, \ldots, m$ and then a sequence of pushes and pops manipulates the pd's. Occasionaly, the machine is in a query state. Those integers popped at a query state generate the sequence from which the set of edges is derived. The set F_k consists of graphs definable by a sequence of length $= (k-1)m - 2$. Due to a simple counting argument, for $ck < n$, most k degree graphs **are not** in F_{ck}. Hence in some sense F_{2k} contains graphs with average degree k that have some simple structure. We assume that F_k is S_k-separable and take S_k to be as large as possible. (Thus, every $G \in F_k$ with $\tilde{m}$ vertices has a separator of size $S_k(\tilde{m})$ and some $G \in F_k$ with $\tilde{m}$ vertices does not have a smaller separator.)

We fix k, and assume that a 1NTM M' accepts L in time $t(n)$. Let m be any large enough power of 2, and let b be a smaller power of 2 to be specified below. We consider an input $y = x\#z\# \in L^b, |y| = n = km, |x| = m = \tilde{m}b$ such that $G(y) \in F_k$. We choose x to be a (Kolmogorov) random string of length m, and fix $h(z)$ below. Each choice for $h(z)$ of the correct length can be uniquely extended to a choice of z such that $y = x\#z\# \in L^b$; so it suffices to specify $h(z)$. $h(z)$ starts with $\gamma^{\tilde{m}}$, where $\gamma = (1, \emptyset, T)$. (So initially the entire string x is queried in reverse order.) The rest of $h(z)$ is chosen to be the lexicographically first string that gives a y such that the corresponding graph $G(y)$ has a smallest separator of size $S_k(\tilde{m})$. From now on we fix an accepting computation of M' on y of time $\leq t = t(n)$.

We will use Kolmogorov Complexity to derive contradictions in the following way. Recall that x is partitioned into $\tilde{m}$ blocks of size $b = b(m)$. We will consider sequences of r blocks A and find a way to describe them in less than $3rb/4$ bits given the other blocks (concatenated to form one string). By describing the sequence we mean (1) giving the sequence of block numbers in A, (2) giving a program P of constant size and input w for P, such that P given w and the string representing the other blocks generates the contents of the blocks in A. Since the other blocks can be described literally, we actually give a way to describe x in $m - rb/4$ bits, which is impossible.

To describe an increasing sequence of of integers, we need the following well known fact:

Proposition 2. An increasing sequence $0 < i_1 < i_2 < \ldots < i_r < N$ of $r \leq N/2$ integers can be described in at most $2r \log(N/r)$ bits.

In Proposition 2 and in other cases below we use an additional factor of 2 so that the encoding will include various types of endmarkers.

We divide the work tape into **regions** of size D to be defined later. A region R **mentions** a block B if when M' reads the block it visits R. R **covers** B if B is mentioned only by R. A block B **belongs to** R if R is the first region that mentions B. A **super region** $\tilde{R}$ of a region R is the area of the work tape consisting of R and its two neighboring regions. The time $t_{\tilde{R}}$ of a super region $\tilde{R}$ is the total time M' spent in $\tilde{R}$ and in its two neighboring regions.

Consider a boundary between two cells on the work tape of M'. An **extended crossing sequence** or an **e.c.s.** at this boundary is the usual crossing sequence augmented by the corresponding sequence of input head positions.

We now describe two situations where a sequence of blocks can be described in an indirect way. They are summarized in Propositions 3 and 4. Both are due to Maass [12]. Assume there is a region R and there is a sequence of blocks A covered by R. Assume t_0 is the time M' read the # following the end of x, and consider the part of the computation until t_0.

Proposition 3. Given the contents of the blocks not in A, the contents of the blocks in A can be generated by a constant size program using the following input: (1) m (the length of x) and b (the block size); (2) the list of block numbers of the blocks in A; (3) the state and the position of the head of the work tape of M' at t_0; (4) two boundaries, one on the left and one on the right of R; (5) two e.c.s.'s at these boundaries; and (6) the contents of the work tape between these two boundaries at time t_0.

Proof Sketch: Let z_i $(i = 1, \ldots, 2^m)$ be the enumeration of all strings $\hat{z}$ with $h(\hat{z}) = h(z)$. The program first generates $h(z)$ by a brute force enumeration of all possibilities and a brute force calculation of the corresponding separators. Next, it successively simulates M' on inputs of the form $x\#z_i\#$. On each such input, the program tries all possible computations. (Recall that M' is nondeterministic.) It puts the blocks not in A in their places and for each block in A it puts b $*$'s. Let us call the region between the two boundaries $\hat{R}$. Whenever the head of M' is not in $\hat{R}$ the program simulates M' directly. Whenever in the simulation the head of the work tape tries to enter $\hat{R}$, the program first checks that the state and head position are consistent with the e.c.s. It continues this simulation only if they are consistent, in which case it uses the two e.c.s.'s to skip to the next step in which the simulation leaves $\hat{R}$. At time t_0 the program fills in the given contents of $\hat{R}$ and subsequently simulates M' directly.

The proof follows from the following observations:

— When $z_i = z$ and the program tries the chosen accepting computation, the simulation can be carried out because until t_0 no block in A is read (since those were read in the parts that were skipped).

— The input $x\#z\#$ is accepted by the simulation.

— No other input enumerated by the program is accepted by the simulation. (Otherwise

by a cut and paste argument M' would accept a string not in L.)

The program infers the contents of the blocks in A from the first $\tilde{m}$ answers to the queries in an accepting computation. □

Assume there is a boundary d on the worktape of M' and a sequence A of q blocks such that when they were being read M' was always to the right of d. Assume also that z contains q queries, one to each of these blocks, such that while processing these queries M' was always to the left of d.

Proposition 4. Given the contents of the blocks not in A, the contents of the blocks in A can be generated by a constant size program using the following input: (1) m (the length of x) and b (the block size); (2) the list of block numbers of the blocks in A; (3) the position of d; and (4) an e.c.s. at d.

Proof Sketch: The program is similar to the program above except that $\hat{R}$ is the part of the worktape to the right of d. The program simulates M' on inputs of the form $x\#z_i\#$, skipping the parts in which M' scanned $\hat{R}$ by using the given e.c.s. (also after t_0). The three obsevations and the conclusion in the previous proof now follow similarly. The program can infer the contents of the blocks in A because they are queried when M' was to the left of d (i.e., not in $\hat{R}$). □

Assume M' has at most 2^q states and an alphabet of size at most 2^r. We define the following parameters:

$p = \lceil 8000rkt/n \rceil$,

$b =$ the smallest power of two $\geq 16\log(n/p)$ (the block size),

$D = \lceil pb/16r \rceil$ (the region size).

Recall that Theorem 2 assumes that $t(n) \leq cn^2/k$. We choose $c = c(M')$ small enough so that $p < n/2$ and $q < b/16$. Note that (since $\tilde{m} = n/kb$)

$$t \leq \frac{1}{500} D\tilde{m}.$$

To complete the proof of Theorem 2, we will show that

$$S_k(\tilde{m}) < 8p = O(kt/n).$$

This yields the inequality of Thorem 2 since S_k is nice. This also directly implies Corrolary 2 without the assumption that S is nice.

Lemma 1. If a super region $\tilde{R}$ covers at least αp blocks, $\alpha > 1$, then $t_{\tilde{R}} > \alpha pD$.

Proof. Assume that $t_{\tilde{R}} \leq \alpha pD$. We describe the αp blocks in less than $3\alpha pb/4$ bits to derive a contradiction. The description includes (1) - (6) of Proposition 3 together with the constant size program. We choose the two e.c.s.'s in the neighboring regions of $\tilde{R}$ that have a shortest length. Thus, their length is at most $t_{\tilde{R}}/D \leq \alpha p$.

The sizes of these six parts are at most (1) $4\log_2 n$; (2) $2\alpha p\log(\tilde{m}/\alpha p) \leq 2\alpha p\log(n/\alpha p)$; (3) $2\log_2 n + 2q$; (4) $8\log_2 n$; (5)$2\alpha p(q + \log(n/\alpha p))$; and (6) $5Dr = 5pb/16$. It follows that the total is $\leq 4\alpha p\log(n/\alpha p) + \alpha pq + 5pb/16+$ lower order terms $\leq 3\alpha pb/4$. □

Let m_i (n_i) be the number of blocks covered by (mentioned in) the first i regions. R is called a **special region** if $m_{i-1}, \tilde{m} - m_i \geq \tilde{m}/3 + p$, and the number of blocks mentioned in R is $\leq 6p$.

Lemma 2. The number of special regions is $\geq \tilde{m}/6p$.

Proof. We define **bad regions** and **bad blocks** of two types. A block B is type 1 (bad) because of region R if while reading B M' crossed R. A region is type 1 (bad) if there are more than p type 1 blocks because of it. A region is type 2 (bad) if more than p blocks which are not type 1 belong to it. These blocks are type 2. Since $t \leq \frac{1}{500}\tilde{m}D$, the number of type 1 blocks is at most $\frac{1}{500}\tilde{m}$, and the number of type 1 regions is $\leq \frac{1}{500}\tilde{m}/p$.

Consider a type 2 region R. The αp type 2 (but not type 1) blocks that belong to R ($\alpha > 1$) are covered by $\tilde{R}$, the super region of R. Hence, by Lemma 1, $t_{\tilde{R}} \geq \alpha pD$. But $\sum_{\tilde{R}} \alpha pD \leq \sum_{\tilde{R}} t_{\tilde{R}} \leq 5t \leq \frac{1}{100}\tilde{m}D$, and the number of type 2 blocks ($\sum_{\tilde{R}} \alpha p$) is $\leq \frac{1}{100}\tilde{m}$, and the number of type two regions is $\leq \frac{1}{100}\tilde{m}/p$. Consequently, the number of bad blocks (regions) is at most $\frac{1}{80}\tilde{m}$ ($\frac{1}{80}\tilde{m}/p$).

Consider a **quadruple** of consecutive regions. We say that it is bad if at least one of the four regions is bad, and good otherwise. Obviously at most $\frac{1}{20}\tilde{m}/p$ of the quadruples are bad. From now on we consider only regions R_i such that $\frac{2}{5} \leq n_i/\tilde{m} \leq \frac{3}{5} + \frac{1}{20} + \frac{1}{80}$. At least $(\frac{1}{5} + \frac{1}{20})\tilde{m}$ good blocks belong to at least $(\frac{1}{5} + \frac{1}{20})\tilde{m}/p$ such good regions. At least $\tilde{m}/5p - 3 \geq \tilde{m}/6p$ of these good blocks start good quadruples. We complete the proof by showing that the third region in each such good quadruple is special.

Let a quadruple be $\{R_i, R_{i+1}, R_{i+2}, R_{i+3}\}$. At most $3p$ blocks that are mentioned in R_{i+2} are type 1 because one of $\{R_{i+1}, R_{i+2}, R_{i+3}\}$, because none of these regions is type 1. At most $3p$ blocks that are not type 1 belong to these three regions because they are not type 2. Hence at most $6p$ blocks are mentioned in R_{i+2}.

Since R_{i+1} is not type 1, all except for at most p blocks mentioned in the first i regions are covered by the first $i+1$ regions, so $m_{i+1} \geq n_i - p \geq \tilde{m}/3 + p$. Finally, $\tilde{m} - m_{i+2} \geq \tilde{m} - n_{i+2} \geq \tilde{m}/3 + p$. □

To complete the proof of Theorem 2, we assume that $S_k(\tilde{m}) \geq 8p$ and derive a contradiction. Let R be a special region, and let A (B) be the vertices of the graph $G(y)$ corresponding to blocks covered to the left (right) of R and let C be the vertices corresponding to blocks mentioned in R. Since R is special, $|A|, |B| \geq \tilde{m}/3 + p$ and $|C| \leq 6p \leq 3S_k(\tilde{m})/4$. By induction on r, there are $r \geq p$ disjoint edges (a matching) $e_i = (u_i, v_i), u_i \in A$, $v_i \in B$. (The induction step follows from the observation that $C \cup \{u_i, v_i |\ i = 1, \ldots, r\}$ does not separate $A - \{u_i |\ i = 1, \ldots, r\}$ from $B - \{v_i |\ i = 1, \ldots, r\}$.) We associate these p edges with the region R.

Since $t \leq \frac{1}{500}D\tilde{m}$ and there are at least $\tilde{m}/6p$ special regions, in at least one of them, R, M' spent less time than $\frac{1}{80}Dp$. Consider the p edges associated with R. For at least $p/2$ of them M' did not cross the middle third of R when it read the queries corresponding to the endpoints of these edges. (Otherwise, M' would spend time at least $Dp/6$ on R.) Without loss of generality, for $p/4$ of the edges M' is always to the left of the third (i.e. rightmost) third of R. Each of the $p/4$ edges has one vertex in B, which corresponds to a block. Recall that when B was read, M' was to the right of R. By Proposition 4, we

describe these $p/4$ blocks by giving a fixed size program and (1) m (the length of x) and b (the block size); (2) the list of $p/4$ blocks; (3) a shortest e.c.s. in the third third of R and its position.

The e.c.s. is of length at most $p/26$ and therefore its description has size $\leq (p/13)(q + \log(26n/p)) \leq (p/13)b(1+\frac{1}{13}) \leq (pb/4)/3$. The list of blocks has length $\leq (p/2)\log(4\tilde{m}/p)$ $\leq (pb/4)/7$. The length of the other parts has smaller order of magnitude. So the full description has length less than $(pb/4)/2$ – a contradiction. □

3. Conclusion

The main open problems left are Problem 1 and 2. Theorem 3 and Corollary 11 imply that even if the answer to the problems is positive the smallest separator of 3-pd graphs must be almost linear and the fastest simulation must be almost quadratic. Two other problems are the following:

— Can the families of 3-pd graphs be replaced by a simple family of graphs that is hard to separate?

— Theorems 1 and 2 provide upper and lower bounds (for the size of the separators or the time of the simulation) that are tight only if the answer to problems 1 and 2 is negative: In Theorem 1, if $S(n) = n/\alpha(n)$ then $t(n) = O(n^2/\alpha(n))$; while in Theorem 2, if $t(n) = n^2/\beta(n)$ then for fixed k $S_k(n) = O(n\log(\beta(n))/\beta(n))$. Can the gap be closed (even without settling these problems, or if the answer is positive)?

Acknowledgment: This paper was conceived during a visit by the three authors at the Department of Computer Science of the University of Chicago in May 1985. The authors would like to thank Bob Soare and the members of the department for their kind hospitality. Thanks also to Stuart Haber for his suggestions concerning the presentation.

References

[1] Baker, B. S. (1983), Approximation algorithms for NP-complete problems, in "Proc. 24th IEEE Annual Symp. on the Foundations of Computer Science," pp. 265-273.

[2] Book, R. V., and Greibach, S. A. (1970), Quasi real time languages, *Math. Systems Theory 4*, pp. 97-111.

[3] Buss, J., and Shor, P. (1984), On the pagenumber of planar graphs, in "Proc. 16th ACM Symp. on the Theory of Computing," pp. 98-101.

[4] Chung, F., Leighton, F. T., and Rosenberg, A. (1984), Embedding graphs in books: A layout problem with applications to VLSI design, manuscript.

[5] Galil, Z., Kannan, R., and Szemerédi E. (1986), k-page graphs with separators of almost

linear size, to appear.

[6] Heath, L. (1984), Embedding planar graphs in seven pages, in "Proc. 25th IEEE Symp. on the Foundations of Computer Science," pp. 74-83.

[7] Hopcroft, J., Paul, W., and Valiant, L. (1977), On time versus space, *J. Assoc. Comput. Math. 24*, pp. 332-337.

[8] Kannan, R. (1985), Unraveling k page graphs, *Info. & Control 66*, pp. 1-5.

[9] Li, M. (1985), Lower bounds by Kolmogorov complexity, in "Proc. 12th Colloq. on Automata, Languages and Programming," pp. 383-393.

[10] Li, M. (1985), Simulating two pushdowns by one tape in $O(n^{1.5}\sqrt{\log n})$ time, in "Proc. 26th IEEE Symp. on the Foundations of Computer Science," pp. 56-64.

[11] Lipton, R. J. and Tarjan, R. E. (1980), Applications of a planar graph separator theorem, *SIAM J. Comput. 9*, No. 3, pp. 615-627.

[12] Maass, W. (1985), Combinatorial lower bound arguments for deterministic and non-deterministic one-tape Turing machines, *Trans. Amer. Math. Soc. 292*, No. 2, pp. 675-693.

[13] Paul, W. I., Pippenger, N., Szemerédi, E., and Trotter, W. T. (1983), On determinism versus non-determinism, in "Proc. 24th Symp. on the Foundations of Computer Science," pp. 429-238.

[14] Rosenberg, A. L. (1983), The Diogenes approach to testable fault-tolerant arrays of processors, *IEEE Trans. Comput. C-32*, pp. 902-910.

[15] Syslo, M. M. (1979), Characterisations of outerplanar graphs, *Discrete Math. 26*, pp. 47-53.

[16] Valiant, L. G. (1976), Graph-theoretic properties in computational complexity, *J. Comput. System Sci. 13*, pp. 278-285.

[17] Yannakakis, M. (1986), Four pages are necessary and sufficient, in "Proc. 18th ACM Symp. on the Theory of Computing," pp. 104-108.