Lower Bounds by Kolmogorov-Complexity*

Ming Li**

TR 85-666
March 1985

Department of Computer Science
Cornell University
Ithaca, New York 14853

---

* This paper will appear in the 12th ICALP conference, Greece, 1985.

** This work was supported in part by an NSF grant DCR-8301766.

LOWER BOUNDS BY KOLMOGOROV-COMPLEXITY *

Ming Li**
Department of Computer Science
Cornell University
Ithaca, New York 14853

Abstract

Using Kolmogorov-complexity, we obtain the following new lower bounds.

For on-line nondeterministic Turing machines,

(1) simulating 2 pushdown stores by 1 tape requires $\Omega(n^{1.5}/logn)$ time; together with a newly proved $O(n^{1.5}\sqrt{logn})$ upper bound [L3], this basically settled the open problem 1 in [DGPR] for 1 tape vs. 2 pushdown case (the case of 1 tape vs 2 tapes was basically settled by [M]);

(2) simulating 1 queue by 1 tape requires $\Omega(n^{4/3}/logn)$ time; this brings us closer to a newly proved $O(n^{1.5}\sqrt{logn})$ upper bound [L3];

(3) simulating 2 tapes by 1 tape requires $\Omega(n^2/lognloglogn)$ time; this is a minor improvement of [M]'s $\Omega(n^2/log^2nloglogn)$ lower bound; it is also claimed (full proof contained in [L3]) that the actual languages used in [M] (also here) and [F] do not yield $\Omega(n^2)$ lower bound.

To cope with an open question of [GS] of whether a k-head 1-way DFA (k-DFA) can do string matching, we develop a set of techniques and show that 3-DFA cannot do string matching, settling the case $k=3$. Some other related lower bounds are also presented.

---

* This paper will appear in the 12th ICALP conference, Greece, 1985.
** This work was supported in part by an NSF grant DCR-8301766.

1. Introduction

Obtaining good lower bounds has been one of the most important issues in theoretical computer science. This paper represents a continuous effort in searching for good lower bounds by means of Kolmogorov-complexity (K-complexity). Some important open questions are answered (or partially answered) and new techniques are developed along with the solution of each problem.

In the traditional lower bound proofs, complicated counting arguments are usually involved. The messy counting arguments blur the essence of the problem, increase the level of difficulty, and therefore limit our ability to understand and obtain better lower bounds. However, the beauty of simplicity and intuition is brought back in the lower bound proofs by a recently discovered tool, the K-complexity. The concept of K-complexity was independently introduced by Kolmogorov [K] and Chaitin [Cha]. The use of K-complexity as a tool in lower bound proofs was first introduced by Barzdin and Paul [P]. Since then, many interesting results have been obtained (e.g. [P1], [P2], [PSS], [RS1]).

Definition 1.1: The *K–complexity* of a finite string X, written as $K(X)$, is the size of the smallest TM (may be nondeterministic) which, starting from empty input tape, *accepts* (or *prints*) only X.

Definition 1.2: The *K–complexity* of X relative to string Y, written as $K(X\,|\,Y)$, is the length of the smallest TM (deterministic or nondeterministic), with Y as its extra information, that *accepts* (or *prints*) only X.

Definition 1.3: String X is *random* if $K(X) \geq |X|-1$. String X is *random relative* to string Y if $K(X\,|\,Y) \geq |X|-log\,|Y|$.

Fact 1: More than half of the finite binary strings are random.

Fact 2: If $X=uvw$, and X is random, then $K(v\,|uw) \geq |v|-log\,|X|$. That is, random strings are locally almost random.

Fact 3: If F is any formal system, X is random and $|X| >> |F|$, then it is not provable in F that '*X is random*'.

The paper is organized as follows: in Section 2 we prove new lower bounds (and present new upper bounds) for on-line computations; in Section 3 we obtain lower bounds on string matching. This paper will concentrate on lower bound proofs, the proofs for all the upper bounds claimed will appear in a companion paper [L3].

2. Lower bounds for on-line computations

In this Section we consider on-line computations which are generally used for investigating the dependency of the computational power on the number of tapes. We call a TM M a k-tape *on-line* machine if M has a 1-way read only input tape and k work tapes. Without explicit indication, all the machines in this section will be on-line machines. An on-line machine M works in *real time* if each time M reads an input symbol it makes only a constant number of moves. A k-pushdown machine is like a usual pushdown automaton, but has k pushdown stores. Similarly, a k-queue machine has k (first in first out) queues instead of k-pushdowns.

One classical question about TM's is how much power an additional work tape gives a machine. For real time deterministic computations, early in 1963, Rabin [R] proved 2 tapes are better than 1. Eleven years later Aanderaa [A] generalized Rabin's result to $k+1$ versus k. In 1982, Duris and Galil [DG] proved, by the crossing sequence technique, that two tapes are better than one in the nondeterministic case. In 1982 Paul [P] proved, using Kolmogorov-complexity, that on-line simulation of $k+1$ tapes by k tapes requires $\Omega(n(logn)^{1/k+1})$ time. Duris, Galil, Paul, and Reischuk [DGPR] later proved that for nondeterministic machines simulating 2 tapes by 1 tape requires $\Omega(nlogn)$ time and simulating k tapes by k pushdown stores requires $\Omega(nlog^{1/(k+1)}n)$ time (for deterministic machines). The following open questions (open problems 1 and 6) were listed in [DGPR]: Can the gaps between the Hartmanis-Stearns [HS] $O(n^2)$ upper bound and the $\Omega(nlogn)$ lower bound in both deterministic case and nondeterministic case for 1 tape simulating k tapes (pushdowns) can be narrowed? Notice that according to [HS1], 2 tapes can deterministically simulate k tapes in time $tlogt$; and according to [BGW] 2 tapes can nondeterministically simulate k tapes without losing any time. For deterministic on-line machines Maass [M], the author [L], and Vitanyi [V] independently proved: it requires $\Omega(n^2)$ time to deterministically simulate 2 tapes (pushdown stores) with 1 tape. (Note, a preliminary $\Omega(n^{2-\epsilon})$ lower bound of Maass antidates both of [L] and [V].) This settled the deterministic case (open question 6 in [DGPR]). In addition, Vitanyi [V] also obtained an $\Omega(n^2)$ lower bound for 1 tape simulating 1 queue. (Note, [P] and [V] uses the 'on-line' model that produces an output each step.) In the nondeterministic case, the situation is quite different: Maass obtained an $\Omega(n^2/log^2nloglogn)$ lower bound for 1 tape vs 2 tapes via a ingenious language and a nice combinatorial lemma, but it does not apply to 1 tape vs 2 pushdown case.

In this section we try to complete our knowledge on nondeterministic case of above open question. An $\Omega(n^{1.5}/logn)$ optimal lower bound, which is obtained in parallel to above deterministic results, is presented for the 1 tape vs 2 pushdown store question. This greatly improves the $\Omega(nlogn)$ bound of [DGPR]. It is optimal because of a recently discovered new simulation by the author that shows unexpectedly: 1 nondeterministic tape can simulate 2 pushdown stores or 1 queue in time $O(n^{1.5}\sqrt{logn})$ [L3]. We will also obtain an $\Omega(n^{4/3}/logn)$ lower bound for 1 tape nondeterministically simulating 1 queue, which is the first lower bound in this case. For the case of 1 tape versus 2 tapes, we improve the Maass' lower bound to $\Omega(n^2/lognloglogn)$. Unlike above results, this result is based on on Maass' proof. Surprisingly, we claim that that the actual language used by [M] (and a similar language, in a different context but for the same purpose, used by Freivalds [F] seven years ago) can be accepted by a one tape machine in $O(\frac{n^2loglogn}{\sqrt{logn}})$ time. Therefore the $\Omega(n^2)$ lower bound, whose existence we doubt, needs a new language.

Throughout this paper, variables X, Y, x_i, y_i ... denote strings in Σ^* for $\Sigma=\{0,1\}$. Consider a 1-tape on-line machine M. We call M's input tape head h_1, and its work tape head h_2.

Definition 2.1: Let x_i be a block of input, and R be a region on the work tape. We say that M *maps* x_i *into* R if while h_1 is reading x_i h_2 never goes out of region R; We say M maps x_i *onto* R if in addition h_2 travels the *entire* region R while h_1 reads x_i.

A crossing sequence (c.s.) for a point on the work tape of M is a sequence of ID's, where each ID is of form (state of M, h_1's position). We write $|c.s.|$ to mean the space needed to represent the c.s.

Remark 2.1: Since h_1 only moves to the right, we can represent the *ith* ID (ID_i) in a c.s. as follows:

ID_i=(state of M, current h_1's position – h_1's position of ID_{i-1}),

where ID_0=(–,0). Thus if a c.s. has d ID's and the input length is n, then $|c.s.| \leq d\,|M| + \log k_1 + \ldots + \log k_d$, with $\sum_{i=1}^{d} k_i = n$. This is less than $d\,|M| + d\log(n/d)$ by a standard calculation (i.e. maximize the function).

Remark 2.2: Let $x_1, \ldots, x_k$ be blocks of equal length C on the input tape. Suppose d of these blocks are deleted, and that we want to represent the remaining blocks in the smallest space possible but still remember their relative distances. We can use following representation,

$$m\bar{x}_1\bar{x}_2 \cdots \bar{x}_k(p_1,d_1)(p_2,d_2)\ldots$$

where m is the number of (non-empty) $\bar{x}_i$'s; $\bar{x}_i$ is x_i if it is not deleted, and is empty string otherwise; (p_i,d_i) indicates that the next p_i consecutive x_k's (of length C) are one group (adjacent to each other), and followed by a gap of $d_i C$ long. m, the p_i's, and the d_i's are self-delimited. (A string x is self-delimited if each bit of x is doubled and with '01' at both ends. For instance, 01001100111101 is the self-delimited version of 01011.) By a standard calculation (similar to Remark 2.1), the space needed is $\sum_{i=1}^{k} |\bar{x}_i| + d\log(n/d)$.

We now prove an intuitively straightforward lemma which coincides with our intuition that a small region with short c.s.'s around it cannot hold a lot of information. Stated formally:

Jamming Lemma: Suppose on input beginning $x_1x_2 \cdots x_k\#$ ..., where the x_i's have equal length, M maps each of $x_{i_1}, \cdots, x_{i_l}$ into region R by the time h_1 reaches # sign. Then the contents of the work tape of M at that time can be reconstructed by using only $\{x_1, \cdots, x_k\} - \{x_{i_1}, \ldots, x_{i_l}\}$, the contents of R, and the two c.s.'s on the left and right boundaries of R.

Remark 2.3: Roughly speaking, if $\sum_{j=1}^{l} |x_{i_j}| > 2(|R| + 2|c.s.| + |M|)$, then the Jamming Lemma implies the either $X = x_1 \ldots x_k$ is not random or some information about X has been lost.

Proof of Jamming Lemma: Name the two positions at the left boundary and the right boundary of R to be l and r, respectively. We now simulate M. Put

$\{x_1,...,x_k\}-\{x_{i_1},...,x_{i_l}\}$ at their correct positions on the input tape (as indicated by the c.s.'s). Run M with h_2 staying to the left of R: whenever h_2 reaches point l, the left boundary of R, we interrupt M (in the nondeterministic case we also match the current state and h_1 position) and consider the next ID in the c.s. at point l, using this we relocate h_1, adjust state of M and then go on running M. After we finish at the left of R, we do the same thing at the right of R. Finally we put the contents of R into region R. Notice that although there are many empty regions on the input tape corresponding to those unknown x_i's, h_1 never reads those regions because h_2 never goes into R. □ (Jamming Lemma)

Remark 2.4: If M is nondeterministic, then we need to rephrase 'contents of work tape' as 'legal contents of work tape' which simply means some computation path on the same input would create this work tape contents.

Define $L=\{x_1\$x_0\$x_2\$x_0 \cdots \$x_t\$x_0\#x_1x_2 \cdots x_t \mid x_i \in \{0,1\}^* \text{ for } i=0,..,t\}$.

Theorem 2.1: It requires $\Omega(n^{1.5}/\log n)$ time to nondeterministically simulate 2 pushdown stores by 1 tape.

The theorem will follow from Lemma 2.1. We shall concentrate on explaining the ideas of the proof.

Lemma 2.1: It requires $\Omega(n^{1.5}/\log n)$ time to accept L by any 1-tape nondeterministic on-line machine.

Proof of Lemma 2.1: Suppose a nondeterministic 1-tape M accepts L in time $o(n^{1.5}/\log n)$. We fix a large n and a large constant C such that all the subsequent formulas are meaningful.

Fix a random string X of length n. Equally partition X into $x_0x_1 \cdots x_k$, where $k=n^{1/2}/C\log n$. Consider input $Y=x_1\$x_0\$x_2\$x_0 \cdots \$x_k\$x_0\#x_1x_2 \cdots x_k$ to M. Observe that $|Y|<3n$. M should accept this input Y. Let us fix a ***shortest*** accepting computation P of M on input Y. We shall show that P is long.

Consider the k pairs $x_i\$x_0\$$ in Y. If half of them are mapped *onto* some regions of sizes larger than n/C^3, then M uses time $O(n^{1.5}/\log n)$, a contradiction. Thus in the following we will assume that for more than half of the above such pairs, M maps each into some region of size $\leq n/C^3$. Let S be the set of such pairs. When h_1 gets to # sign, we consider two cases:

Case 1: (jammed) Assume there do not exist two pairs in S that are mapped into 2 regions n/C^2 apart. In this case, it is clear that all the pairs in S are mapped into a region R of size $3n/C^2$, since every pair in S is mapped into a region of size $\leq n/C^3$. Consider the two regions R_l and R_r of length $|R|$, left and right neighboring to R respectively. Find a point l in R_l and a point r in R_r with shortest c.s. in R_l and R_r, respectively. If either of the two c.s.'s is of length more than $n^{0.5}/C^4 logn$ then M uses $O(n^{1.5}/logn)$ time. If they are both shorter than $n^{0.5}/C^4 logn$, then the Jamming Lemma can be applied. We can reconstruct the contents of the work tape at the time when h_1 gets to # sign by a short program. By Jamming Lemma, the construction only requires the following information:

(1) $\{x_0, x_1, \cdots, x_k\} - \{x_i \,|\, x_i x_0 \in S\}$, which, by Remark 2.2, requiring less than $2|X|/3$ space;

(2) two c.s.'s that require lees than $|X|^{1/2}$ space; and

(3) the tape contents of regions R, R_l, and R_r which requires no more than $9|X|/C^2$ (note $C>>9$) space.

We then find X as follows: for each Y such that $|Y|=|X|$, equally divide $Y=y_0y_1 \cdots y_k$ as dividing X. Check if $y_0=x_0$. Attach $y_1 \cdots y_k$ after # sign and continue to simulate M with the work tape constructed as above; M accepts iff $Y=X$. This program is short, showing $K(X)<|X|$. One might worry about the nondeterminism here, but notice that the nondeterminism is also defined in the K-complexity, and we can simply simulate M nondeterministically in the above, making sure that the c.s.'s are matched.

Case 2: (not jammed) Assume there are two pairs, say $x_i\$x_0$ and $x_j\$x_0$, that are mapped n/C^2 apart, that is, the distance between the two regions *onto* which these two pairs mapped is $\geq n/C^2$. Let R_0 be the region between above two regions. Hence $|R_0| \geq n/C^2$. As before we search for a shortest c.s in R_0. If the shortest c.s. is longer than $n^{1/2}/C^3 logn$, then M runs for $O(n^{1.5}/logn)$ time. Otherwise we record this short c.s. and try to reconstruct x_0 in below. But notice that a simple minded approach such as finding a shortest c.s. in the middle is not enough here, because some other x_0's can be mapped on both side of the c.s.

To overcome above difficulty, observe that since the size of shortest c.s. is $n^{1/2}/C^3 logn$ there can only be this many bits in x_0 that are mapped to both sides of the c.s. From this observation, we reconstruct x_0 as below.

(1) In each *ID* of the shortest c.s., we add a bit which specifies the current bit read by h_1. Fortunately, this does not cause the increase of c.s. size: for the c.s. of length $n^{1/2}/C^3 logn$, less than $2n^{1/2}/C^3$ space is needed by Remark 2.1.

(2) For each Y such that $Y=X$, we equally divide $Y=y_0y_1 \cdots y_k$. Check if $y_i=x_i$ for all $i>0$. If not, then $Y \neq X$; otherwise, arrange y_i's in their corresponding positions on the input tape.

(3) Simulate M only at the left of the shortest c.s. Every time h_2 meets the c.s., M check if the current *ID* matches the current state of M (including the bit added in (1)), and then take the next *ID* continuing the simulation. $Y=X$ iff the simulation ends with everything coincides.

The above program uses the information of $x_1, \cdots, x_k$ and the c.s. which needs less than $|x_0|/2$ space to represent. This contradicts the relative randomness of x_0. □ (Lemma 2.1)

Proof of Theorem 2.1: The language L can be easily accepted by a two tape machine. For two pushdown stores, we modify L: reverse $x_1x_2...x_k$ following # sign. The modified L can be accepted by $\overline{M}$ with two pushdown stores in linear time as follows: put x_1 in stack1, put next x_0 in stack1 and in stack2, put x_2 in stack2, put next x_0 in stack1 and stack2, and x_3 in stack1, ..., and so on. When the input head reads to #, $\overline{M}$ starts to match in an obvious way. To make this process real time we further modify L by simply putting a $1^{2|x_0|}$ padding after every other reversed x_i. Since all these changes do not hurt our lower bound proof in Lemma 2.1, the proof is complete. □ (Theorem 2.1)

Combined with Theorem A below recently proved in [L3], we essentially close the gap for 1 tape vs 2 pushdown stores, answering the open question 1 of [DGPR].

Theorem A: 2 pushdown stores or 1 queue can be simulated by 1 nondeterministic tape in $O(n^{1.5}\sqrt{logn})$ time (for both on-line and off-line machines). (The proof is contained in [L3].)

Theorem 2.2: It requires $\Omega(n^{4/3}/logn)$ time to nondeterministically simulate 1 queue by 1 tape.

Idea of the Proof: At first glance, one might think the language L in above can be used (and therefore an $\Omega(n^{1.5}/logn)$ optimal lower bound). Unfortunately, with a second thought, a 1 queue machine probably has no way to accept L in linear time. But if you persist, the following observation can be made. If $|x_0|=n^{1/3}$, and $|x_{i\geq 1}|=n^{2/3}$, then 1 queue machine can accept in linear time on the condition that it could *count fast* (to make sure that the sizes of x_i's are correct). How does a queue count fast? Probably no way. But this leads us to the following language

$$L_{pad}=\{x_1x_0x_2x_0 \cdots x_kx_0\#x_1 \cdots x_k\#1^{k|x_0|^2}\}.$$

We claim that a 1 queue machine can accept L_{pad} in linear time, but a 1 tape machine would need $\Omega(n^{4/3}/logn)$ in the worst case. The algorithm for accepting L_{pad} by 1 queue is as follows.

(1) Put $x_1x_0 \cdots x_kx_0$ into the queue

(2) Match $x_1, \cdots, x_k$ by the input head the the reading head of the queue, while copying the x_0's back to the queue and deleting all other $x_{i>0}$'s

(3) Match all x_0's bit by bit in the obvious way in $k|x_0|^2$ time, while the input head scanning the padding

The lower bound can be proved in the same way as in Theorem 2.1. The $\Omega(n^{4/3}/logn)$ lower bound comes from the padding and size of each x_i ($|x_0|=n^{1/3}$, $|x_{i>0}|=n^{2/3}$, $k=n^{1/3}$). □ (Theorem 2.2)

Theorem 2.2 brings us closer to the $O(n^{1.5}\sqrt{logn})$ upper bound of Theorem A, although a gap is still to be closed. This is also the first lower bound for 1 tape vs 1 queue in the nondeterministic case. For deterministic case with output each step, a $\Omega(n^2)$ lower bound was proved in [V].

For the nondeterministic case of 1 tape vs 2 tapes, Maass [M] obtained an $\Omega(\frac{n^2}{(logn)^2loglogn})$ lower bound. Recently, the author found a theorem claimed 7 years ago by Freivalds [F] (Theorem 2 in [F], without proof) which, if true, would immediately imply the tight $\Omega(n^2)$ lower bound. Both [F] and [M] independently constructed two similar ingenious languages (although the language by [F] was less complete).

In [M], although a very general language L_I was introduced, only a simple subset, $\hat{L}$, of it was used. The language $\hat{L}$ can be defined as follows (w.l.g. let k be odd).

$$\hat{L}=\{b_0^1 b_1^1 \cdots b_k^1$$
$$b_0^2 b_0^3 b_1^2 b_2^2 b_1^3 b_3^2 \cdots b_{2i}^2 b_i^3 b_{2i+1}^2 \cdots b_{k-1}^2 b_{(k-1)/2}^3 b_k^2$$
$$b_0^4 b_{(k+1)/2}^3 b_1^4 b_2^4 b_{(k+3)/2}^3 b_3^4 \cdots b_{2j(modk+1)}^4 b_j^3 b_{2i+1(modk+1)}^4 \cdots b_{k-1}^4 b_k^3 b_k^4$$
$$\mid b_i^1=b_i^2=b_i^3=b_i^4 \text{ for } i=0,\cdots,k\}$$

The length of each b_i^j (a binary string) may be different. We can also define a delimited version L^* of $\hat{L}$ where every b_i^j in $\hat{L}$ is replaced by $*b_i^j*$ of an uniform length.

The language, B, constructed in [F] is similar (but less complete) if we let $c(i)=a(i)b(i)$ and replace each single $a(i)$ or $b(i)$ by $c(i)$ in the following. Here is the construction of [F]. Let B' consist of all strings

$$a(1)b(1)a(2)b(2)\cdots a(2n)b(2n)2a(2n)b(2n)b(2n-1)a(2n-1)b(2n-2)b(2n-3)\cdots$$
$$\cdots a(n+1)b(2)b(1)$$

where all $a(i)$ and $b(i)$ are from $\{0,1\}$. B is defined to be the set of all strings $0x$ or $1y$, where $x \in B'$ and $y \in \overline{B'}$. [F] claimed that it requires $\Omega(n^2)$ time for a 1 tape nondeterministic on-line TM to accept B.

Theorem B [L3]: $\hat{L}$ (L^* and B) can be accepted in $O(\frac{n^2 loglogn}{\sqrt{logn}})$ time by a 1-tape *nondeterministic* on-line machine.

The proof of Theorem B is based on Lemma B (details contained in [L3]).

Lemma B [L3]: Let $S=\{0,1,\cdots,k-1\}$ where $k=2^l$ for for some integer l. Let R be a binary (neighboring) relation defined on S such that, for s_1 and s_2 in S, $s_1 R s_2$ if

(1) $s_1=2*s_2(modk)$ or $s_1=2*s_2+1(modk)$, or

(2) $s_1=s_2+1$, or

(3) $s_2 R s_1$.

Then there exists a partition of S into two sets S_1 and S_2 such that,

(a) $|S_1|=|S_2|$.

(b) $S_1 \cap S_2 = \phi$,

(c) $|N| = O(k/\sqrt{logk})$, where $N=\{s_1 \in S_1 \mid s_1 R s_2$ for some $s_2 \in S_2\}$. (N is the set of elements in S_1 that are 'neighbors' of some elements in S_2.) (Proof contained in [L3].)

Remark: This gives an upper bound on Theorem 3.1 of [M] and Lemma 2.3 below.

Remark: Assertion (c) in Lemma B is true for any partition as long as for all $s_1 \in S_1$ and $s_2 \in S_2$, $\#bin(s_1) \leq \#bin(s_2)$ where $\#bin(x) = \#$ of 1's in binary x. It is this property which is used in the proof of Theorem B.

Corollary B [$L3$]: Language L^* and B can be accepted by a 1 tape deterministic on-line machine in $O(n^2 loglogn/\sqrt{logn})$ time. B can be accepted by a 1 tape nondeterministic on-line machine in time $O(n^{1.5}\sqrt{logn})$ □ (Proof contained in [L3])

Remark: Other upper bounds have also been obtained. For example, 1 nondeterministic tape can probabilistically simulate 2 tapes in less than square time with any fixed small error ϵ (*i.e.* reject with $Pr(exist\ accepting\ path) \leq \epsilon$ accept if there is a path P, $Pr(P\ accepts) > 1-\epsilon$. Also W. Ruzzo showed that a multitape Σ_k ATM running in time T can be simulated by a 1 tape $\Sigma_{k'}$ ATM in time $O(TlogT)$ where $k'=k+1$ if k is odd, $k'=k$ otherwise [R2], the $k=1$ case has been proved by N. Pippenger [R2].

In the rest of this section, trying to meet above upper bound, we improve the [M]'s lower bound to $\Omega(\frac{n^2}{lognloglogn})$. Unlike Theorem 2.1 (which was obtained in parallel to those of [M] and [V]), the next theorem is based on [M]'s approach. We assume the reader is familiar with [M].

Theorem 2.3: It requires $\Omega(n^2/lognloglogn)$ time to nondeterministically simulate 2 tapes by 1 tape.

We will show that the language L^* (and $\hat{L}$) requires $\Omega(n^2/lognloglogn)$ time for 1 tape machines. We will only give ideas to show where and how the improvement is made. We refer the readers to [M] for details. In [M], Maass proved an important combinatorial lemma (Theorem 3.1 in [M]) which is generalized below,

Lemma 2.3: Let S be a sequence of numbers from $\{0,..,k-1\}$, where $k=2^l$ for some l. Assume that every number $b \in \{0,..,k-1\}$ is somewhere in S adjacent to the number $2b(mod\ k)$ and $2b(mod\ k)+1$. Then for every partition of $\{0,..,k-1\}$ into two sets G and R such that $|G|$, $|R|>k/4$ there are at least $k/clogk$ (for some fixed c) elements of G that occur somewhere in S adjacent to a number from R.

The proof of this lemma is a simple reworking of [M]'s proof. An $n/\sqrt{logn}$ upper bound of this lemma is contained in Lemma B.

Notice that any sequence S in L^* satisfies the requirements in Lemma 2.3. Let n be the length of a random string that is divided into $k=n/loglogn$ blocks. A sequence S in L^* is constructed from these k blocks. A new idea is to find *many* (instead of 1 as in [M]) 'deserts' on the work tape.

Lemma 2.4: ('Many Desert Lemma') For some constant C, and for large n, there are $I=logn/C$ regions $D_1,\ D_2,\ \cdots,\ D_I$ on the work tape such that,

(1) for all $i \neq j$, $D_i \cap D_j=0$;

(2) for each i, $|D_i|=n/c^{12}logn$, where $c \geq 2$ is the constant in Lemma 2.2;

(3) for each i, at least $k/4=(n/4loglogn)$ blocks are mapped to each side of D_i.

Proof of Lemma 2.4: Again we only give the ideas behind the proof. Divide the whole work tape into regions of length $n/c^{13}logn$. By the Jamming Lemma, no region can hold more than $n/c^{11}logn$ blocks. By a standard counting argument, we can find regions $D_1, D_2,...,D_{logn/C}$ for some constant C in the 'middle' of work tape such that (1), (2), and (3) above are satisfied. □ (Lemma 2.4)

To prove Theorem 2.3, we apply the proof of [M] for each desert D_i in Lemma 2.4. Instead of using Theorem 3.1 of [M] we use Lemma 2.3 above. Notice that since each D_i is 'short', the total number of blocks mapped outside D_i is more than $k-(k/c^9logk)$. Therefore Lemma 2.2 can be applied. Now for each region D_i, M has to spend $O(n^2/(logn)^2loglogn)$ time. We sum up the time M spent at all $O(logn)$ regions, getting the $\Omega(n^2/lognloglogn)$ lower bound. □ (Theorem 2.3)

3. Lower bounds on string-matching

The string-matching problem is defined [GS] as follows: given a character string x, called the *pattern* and a character string y, called the *text*, find all occurrences of x as a subword of y. The string-matching problem is very important in practice.

Since the publication of linear time algorithms by [BM], [C2], and [KMP], there has been a constant effort to search for better algorithms which run in real time and save space. Finally, Galil and Seiferas [GS] showed that string-matching can be performed by a six-head *two-way* deterministic finite automaton in linear time. They ask whether a k-head one-way deterministic finite automaton (from now on k-DFA) can perform string-matching. In [LY], we answered this question for case $k=2$ by showing that 2-DFA cannot do string-matching. Efforts have been made for the cases where $k>2$, but even the case $k=3$ has not been solved. It is believed that a solution to the case of $k=3$ would give some important insights into the general case.

Towards answering the Galil-Seiferas conjecture, we develop a set of techniques which enable us to settle the case of $k=3$ negatively. (Note, a weaker version of this result has been reported in [L1].) We hope that the methods used here combined with that of [LY] would help in providing useful techniques for the general problem.

In addition, we obtain lower bound on string-matching by 2-way k-head DFA with $k-1$ heads blind, and on probabilistic matching and moving strings on one Turing machine tape.

Because of the space limitation, we assume the familiarity of automata theory [HU] and we have to omit many proofs. The details can be found in [L1, L2]. We assume that the standard input to M is $\#pattern\$text\not{c}$, where $pattern, text \in \Sigma^*$ for the alphabet $\Sigma=\{0,1\}$. M starts with all heads at # sign.

An *ID* of M *on* input I is the $k+2$ tuple: $(I, q, i_1, i_2, \cdots, i_k)$ where q is a state and i_j for $1 \le j \le k$ is the position of the j-th head. An $\overline{ID}_t$ of M on input I is the *ID* of M without I at time t.

Let I_1 and I_2 be *ID*'s. We write $I_1 \vdash I_2$ if M, started in *ID* I_1, reaches *ID* I_2 in one step. We write $I_1 \vdash_* I_2$ either if $I_1 = I_2$ or if M, started in *ID* I_1, reaches *ID* I_2 in a finite number of steps.

Superscripts are used to denote different occurrences of the same string. Subscripts are used to denote different binary strings. Everything in the following concerns a fixed 3-DFA M, and a long random string $Y=kXk'$. M and Y will be chosen in Theorem 3.1

We name the three heads of M as h_a, h_b, h_c. We will also use h_1, h_2, and h_3 to mean the leading head, the second head, and the last head respectively at a specific time. So, h_a, h_b, and h_c are fixed names, whereas h_1, h_2, and h_3 are only transient names.

We use $p(h)$ to denote the phrase 'the position of the head h'. Let x be a string (a segment of M's input) of length greater than 0. At a particular step in the simulation of M, we make the following definitions: $p(h_i)=x$ denotes that the position of h_i is at the last bit of x; $p(h_i)>x$ means that h_i has passed the last bit of x; $p(h_i)<x$ means that h_i has not reached the first bit of x.

a_0 always stands for the *pattern* which is going to be of form $1^kX1^{k'}$ where $Y=kXk'$. In the following, we always consider input of form $\# 1^kX1^{k'}\$text\cent$. X is always equally partitioned into six parts $X=x_1x_2...x_6$. In general, given strings s or $x_{uv...w}$, without explicit definition, we always implicitly assume that they are equally partitioned into six parts, and written as $s_1s_2 \cdots s_6$ or $x_{uv...w\,1}x_{uv...w\,2}...x_{uv...w\,6}$, respectively. When the ranges of the indices are not explicitly stated, they are always assumed to be from 1 to 6.

If $X=xyz$, then $X-y=xz$. If y is not a substring of X, then $X-y=X$.

Definition 3.1 The *text* in the input $\# 1^kX1^{k'}\$text\cent$ is *regular* if it is concatenated, no more than 1000 times, from the following blocks:

(1) $1^l01^l0 \cdots 1^l0$, where 1^l is repeated less than $log\,|X|$ times and $K(l) \leq log\,|X| + 2\,|M|$;

(2) X;

(3) X', where X' is obtained from X by replacing a substring x (only one) by x' that has an equal length and satisfies $K(x'\,|X) \leq 100 log\,|X|$;

(4) Prefixes of X;

(5) $\overline{X}$, where $\overline{X}$ is obtained from X by replacing a substring x (only one) by $\overline{x}$ that has an equal length and satisfies $K(\overline{x}\,|X-x) \leq |k| + |k'| + 100 log\,|X|$;

(6) 1^k and $1^{k'}$.

The *text* is *easy* if only blocks from (1)-(4) are allowed in above. □ (Definition 3.1)

Proposition 3.1: If the *text* is *easy* then: (1) the *text* can be constructed from $|X|$ and $O(log\,|X|)$ information; (2) there is a constant C_0 ($<<k,k'$) not depending on k or k' such that each head position in the *text* at a specific time can be described by $|k| + C_0$ information, and further, if a head h is in an occurrence of X, X', or some prefix of one of them, then $p(h)$ can be specified by $C_0 log\,|X|$ information. □ (Proposition 3.1)

Definition 3.2: Let x be a string segment of an *easy text*. x is *independent* (with respect to *text*) if for every string X', or its prefix in Definition 3.1 that appears in the *text*, $K(X'-x\,|X-x) \leq 50 log\,|X|$. □ (Definition 3.2)

Definition 3.3: Let x be *independent*. We say x *is compared with* y if (1) $|x| = |y|$, and (2) there is a time when one head, say h_i, is at x and simultaneously another head, say h_j, is at y (excluding the first bit and last bit of x and y). If y is just another occurrence of x, then we say this occurrence of x *is matched*, or matched by (h_i,h_j). We also say (h_i,h_j) did the matching. Let x^1 and x^2 be different occurrences of string x in text of above input. We say x^1 is *matched* to x^2 if there is a sequence of occurrences of x's starting from x^1 and ending with x^2, each being compared with the next. An occurrence of x is *well-matched* if this occurrence of x is matched to the x of a_0. □ (Definition 3.3)

The idea behind the proof of next theorem comes from the following observation: Let kXk' be a random string. Suppose that there is a time that all three heads have left the *pattern* $1^kX1^{k'}$ and no head is reading the ¢ sign, the *text* is *easy* with no occurrences of *pattern*, and two heads are reading some occurrences of X. Then we would lose the information of either k or k'. At this time we attach $1^lX1^{l'}$ to the end of *text*, if the machine does string-matching correctly, we would be able to recover k and k' by finding the minimum l and l' such that the machine finds a matching. Therefore, we show that kXk' is not random. So our goals are to (1) make *text easy* and (2) drive the heads out of *pattern* (or 1^k of *pattern*). To make *text easy* we construct *text* to be (essentially) a sequence of a_i's and block of 1's, where $a_i = 1^m X 1^m$ for some non-random m greater than k and k'. To drive the heads out of the *pattern*, we have to do an exhaustive adversary proof. The goal is to 'construct an *text easier* than the *pattern*'

Theorem 3.1: No 3-DFA accepts $L = \{\# a_0 \$ text ¢ \mid a_0$ is a substring of *text*$\}$.

Proof of Theorem 3.1 (*sketch*): Suppose a 3-DFA M accepts L. Fix a long enough random string Y, as mentioned before. We will show that Y is not random for a contradiction. Divide $Y = kXk'$, where $|k| = |k'|$, $|X|^{1/4} >> |k|$, $|X| >> \sqrt{|X|}$, and $|k| >> log\,|X|$. Let $m = \min\{2^f \mid 2^f > k, k'\}$. X is divided into $x_1 x_2 x_3 x_4 x_5 x_6$ of equal length. Consider only inputs of form $\# 1^k X 1^{k'} \$ text ¢$ to M. That is, $a_0 = 1^k X 1^{k'}$. We will always assume that we are in the process of simulating M.

The following strategy **P** is needed to play our adversary proof. The purpose of **P** is to obtain an invariant value such that after h_1 has passed a block of 1's, many more 1's can be added without changing the status of M. Further this block of 1's can be used to recover k if it is followed by $X 1^m$. To understand it better, one may want to read **P** later when **P** is called.

Strategy P(x): Given $\# a_0 \$ text ¢$ on tape, $p(h_1) = text$ with corresponding state of M and h_2, h_3 positions. The parameter x is a substring of X.

$i := 1$;

repeat

append $b_i = 1^{m\,|Q|}0$ to the input (before ¢);

continue to simulate M until $p(h_1) = b_i 0$;

$i := i + 1$

until $S_1 \vee S_2 \vee S_3 = true$;

The three predicates S_1, S_2, and S_3 are defined as below:

S_1: A matching of one occurrence of x (parameter of **P**) to another occurrence of x by (h_2, h_3) happened in the last loop;

S_2: Neither h_2 nor h_3 moved more than $|Q||X|$ steps in the last loop;

S_3: h_1 and h_2 are separated by only 1-blocks.

If S_2 is true, then there exist constants $C_1, C_2 < |Q||X|+1$ such that for all l, should we let $b_{i-1} = 1^{C_1 + l*C_2}$ in the input, M would be in a fixed state with same h_2, h_3 positions when $p(h_1) = b_{i-1}$. Replace the last appended $b_{i-1} = 1^{m|Q|}0$ by $a_f = 1^{C_1 + l*C_2}X1^m$ where $l=1$ at this moment.

If S_1 or S_3 is true, do nothing.

end_P.

Claim P: (1) Only one of the S_i's can be true; (2) The number of times that the *repeat* loop is executed is less than twice the number of X-blocks and 1-blocks in the input. □ (Claim P)

Nine technical lemmas are needed in the process of simulating M. Note that the $a_{i>0}$'s used in each of the following lemmas are all 'local', that is, they have no relation with any $a_{i>0}$'s used in the proofs of other lemmas or main theorem.

Lemma 1 (The Matching Lemma): Let the *text* be *regular* (Def. 3.1) and with exactly one occurrence, a_g, of a_0 in it. Let x be a segment of X such that (1) x is *independent* (Def. 3.2), and (2) $|x| > \sqrt{|X|}$. Then the occurrence of x in a_g, must be well-matched.

Proof of Lemma 1: Suppose Lemma 1 is not true. Let x^l, for $l=1,2, \cdots l_0 \leq 1000$, be all the occurrences of x, including the one in a_g, that are not well-matched in the *text*. Now for each x^l, we record 3 pairs of information for the 3 heads,

h_a pair: (positions of h_b and h_c and state of M when h_a enters this occurrence of x^l, positions of h_b and h_c and state of M when h_a leaves this x^l.);

h_b pair: exchange h_a and h_b in above;

h_c pair: exchange h_a and h_c in h_a's pair.

We now show that Y is not random by giving a short program which accepts only Y. For input Y',

(1) Compare Y' with Y except the x part which we do not need.

(2) Construct the *pattern* and the *text* with x' of Y' (the corresponding part of x) replacing all x's. Then for each of the above three pairs, starting from the first component, we simulate M until some *ID* of M coincides with the second component of the pair. If there is no such coincidence, we reject this Y'.

If Y' passes tests (1) and (2), then $Y'=Y$ (otherwise M does not accept L). Notice that we used only the following information: (i) $X-x$ and $5(|k|+|k'|)$ amount of information for constructing the regular text (excluding the x part), and (ii) h_a, h_b, h_c pairs for each x^l. The total amount of information that we used in the above program is less than $|Y|$ because of the assumption $|X|^{1/4} >> |k|$ and the fact $|x| > \sqrt{|X|}$. $\square$ (Lemma 1)

Corollary: (1) Lemma 1 is true for a k-NFA, for any k; (2) 2-DFA cannot do string matching (see [LY]).

Remark: Combined with the ideas from [YR], the proof of a theorem by Yao and Rivest [YR], which states that a k-DFA is better than a $(k-1)$-DFA, can be simplified.

Lemma 2 (The 2-head Lemma): Let s, $|s| > \sqrt{|X|}$, be an independent (Def. 3.2) segment of X. For input $I=\# 1^k X 1^{k'} \$ Z 0 a_1 0 a_2 \ldots a_{l+2} 0 \not{c}$, where $a_i = 1^m X 1^m$, let Z be regular (Def. 3.1) with no occurrence of a_0 and no more than l occurrences of s in it. If there is a time when $p(h_1) = (s \text{ of } a_{l+2})$, $p(h_2) < (s \text{ of } a_1)$, and s in $a_{1 \le i \le l+2}$'s are not matched, then X is not random.

Remark on Lemma 2: We have presented a simplified form of Lemma 2. When it is actually applied, the a_i's and contents of Z can be intermingled. Since the proofs are the same, we preferred to present a simplified version.

The next lemma suggests the basic idea of the proof of our main result.

Lemma 3 (The Easiness Lemma): Let the *text* be *easy* and contain no occurrence of a_0. If at some step t, two heads of M are out of a_0 and their positions can be described by $10 \log |X|$ long information, and if no head is in 1^k of a_0 or at the $\not{c}$ sign, then Y is not random.

Lemma 4 (The Replacement Lemma): Assume the *text* is *easy*. At time t in the simulation of M, if a segment s of X is not matched, then there exists s'

such that, (1) $|s|=|s'|$, (2) $s \neq s'$, (3) s' can be constructed from X and $O(log\ |X|)$ information, and (4) if h_i passed s at time t, then replacing s by s' will not change the status of M when $p(h_i)=s'$ (or s).

Lemmas 4-1, 4-2, 4-3 are variants of the Replacement Lemma that are needed in the application.

Lemma 4-1: In Lemma 4, if the condition that s is not matched is removed, then we can conclude that all not well-matched occurrences of s can be replaced by some s' so that (1)-(4) in Lemma 4 are still true. □ (Lemma 4-1)

Lemma 4-2: Let *text* be *easy* and contain C full occurrences of a_0, say $b_1, \cdots, b_C$. Assume that at some time t in the simulation of M, for each b_i there is a substring s_i of X not well-matched. For $i=1,..,C$, let $|s_i| \geq |X|/1000$ (s_i's may be all different). Let s_1 be independent and s_1 appears in text less than D times. Here C and D are small constants less than, say, 20. Then *text* can be changed by substituting s_i's so that:

(1) *text* is easy and does not contain any occurrence of a_0;

(2) There is a time t' such that, $\overline{ID}_{t'}$ on the changed input is same to $\overline{ID}_t$ on the old input.

(3) There is a substring e of s_1 which remains unchanged in s_1 after replacement. $|e| \geq |s_1|/2C$ and e is independent.

Lemma 4-3: Lemma 4-2 can be modified so that b_1 is not changed. That is, the resulting *text* contains exactly one occurrence b_1 of a_0.

Lemma 5: Let *text* in input $\#a_0\$text¢$ be easy and contain no occurrence of a_0. Let s be a substring of X where s is independent with respect to *text* and $|s| > \sqrt{|X|}$. If there is a time t of M such that $p(h_3) > (1^k$ of $a_0)$, and $p(h_1) < ¢$, then Y is not random.

Lemma 6: Let the *text* in input $\#a_0\$text¢$ be *easy*. Let there be exactly one occurrence of a_0 in $a_{j>0}$ in *text*, call it a_j. Let s be a substring of X in a_j such that (1) s is independent, (2) $s < p(h_1), p(h_2) <$ ¢, (3) s of a_j is not well-matched (to a_0), (4) s is not matched to other occurrences of s that h_2 can still see, and (5) $|s| > |X|/1000$. Then Y is not random.

Now we continue to prove Theorem 3.1. We construct an *easy text*. Let the partial input be

(A) $\#a_0\$a_10\cdots,$

where $a_1=1^mX1^m$. Consider the time t when $p(h_1)=(X$ of $a_1)$. Note, at t, $p(h_3)<(X$ of $a_0)$, since otherwise we can remove second 1^m from a_1 and apply Lemma 3.

(1) All x_i's of a_1 are matched (by h_1,h_2), then there is a time of M such that $p(h_1)=(x_2$ of $a_1)$, and $p(h_2)>(x_1$ of $a_0)$. Change a_1 to $a_1{}'=1^mx_1x_2$. Add $a_2=1^mX1^m0$ to get the partial input

(B) $\#a_0\$a_1{}'0a_20\cdots$

Simulate M on the new input and consider time $p(h_1)=(X$ of $a_2)$ for the new input (B). There must exist an x_p in a_2 not yet matched (assuming $p(h_3)<(1^k$ of $a_0))$. We go on constructing the input by the following process.

(C) *For* $t=3$ to 8 *repeat* the following. Add $a_t=1^mX1^m$ to the input and and run M on the new changed input. Consider time $p(h_1)=(X$ of $a_t)$. If all $x_i{}'$'s of a_t are matched, then let $a_t=1^mx_1x_2x_3$.

Now at time $p(h_1)=(X$ of $a_8)$. We consider following cases.

(1.1) If, for $j=1,..,6$, all x_{pj} of a_2 are matched to some $a_{i>2}$'s, we find the smallest i such that x_{p4} of a_2 is matched to x_{p4} of a_i. Change *text* to

(D) $\ldots a_20a_30\ldots a_{i-1}01^mx_{p1}\ldots x_{p4}0\cancel{c}.$

Simulate M on the new input (D) until $p(h_1)=text$. We then apply $\mathbf{P}(x_p)$ which results exactly one of S_1, S_2, or S_3 true. If S_1 is true, then $p(h_3)>(1^k$ of $a_0)$, we apply Lemma 4-2, then Lemma 5. If S_3 is true, then x_{p5} of a_2 cannot be well-matched, to satisfy the conditions of Lemma 1 we apply Lemma 4-3; If S_2 is true, then $\mathbf{P}$ adds $a_f=1^{C_1+l*C_2}X1^m$ to the input, and we simulate M on the new input and stop M as soon as one of the following cases happens.

(1.1.1) $p(h_3)=(1^k$ of $a_0)$. Then x_{p6} of a_2 is not well-matched. Other un-matched x_j's in $a_{i>2}$ remain un-matched. We delete second 1^m from a_f and we are done by Lemma 4-2 and Lemma 5.

(1.1.2) $p(h_2)>(x_{p5}$ of $a_2)$. If (h_1,h_2) did not match x_{p5} of a_f to that of a_2, then x_{p5} of a_2 has not been matched yet. Delete second 1^m of a_f, then apply Lemma 4-3 followed by Lemma 6. If (x_{p5} of a_2) is matched to (x_{p5} of a_f), then case (1.1.3) applies.

(1.1.3) $p(h_1)>(x_4$ of $a_f)$. Then $x_{p1}x_{p2}x_{p3}$ of a_f is not matched, because when $p(h_1)=(x_4$ of $a_i)$ we have $p(h_2)>(x_3$ of $a_2)$. And x_{p2} of a_2 is not well-matched. Also for every a_i in between a_2 and a_f there is a long not well-matched part. Now continue to run M and stop M as soon as one of the following cases happens.

(1.1.3.1) h_2 reaches a_f. Apply Lemma 4-3 so that only a_2 still contains a_0 and s, which is a long substring of x_{p2}, is *independent*. If s is not a substring of the changed a_f then the Matching Lemma can be applied; suppose it is, since s in a_f is not matched, there must be a substring s' of s in a_2, satisfying $|s'|\geq|s|/i$ where i is defined in (D), that cannot be well-matched, again we can apply the Matching Lemma.

(1.1.3.2) (h_2,h_3) do some matching of X. If h_1 is not at $\not{c}$, we are done by Lemma 4-2 and Lemma 5. And if $p(h_1)=\not{c}$ we apply Lemma 4-3 so that only a_f contains a_0 and *text* is *easy*, then we can vary l to find k by the method of Lemma 3. This shows Y is not random.

(1.2) There exists j such that x_{pj} is not matched to any $a_{i>2}$.

(1.2.1) $p(h_2)>a_2$. Apply Lemma 4-3 so that only a_2 contains a_0, then use Lemma 6;

(1.2.2) $p(h_2)\leq a_2$. Since many a_t's are left in form of 1^mX1^m, with some common part un-matched, in (C), the 2-head Lemma can be applied.

(2) Some x_p in a_1 is not matched. We construct new input by process (C), and exactly the same argument as in (1.1)&(1.2) applies. (Change a_2 to a_1.) $\square$ (Theorem 3.1)

Remark: (1) We hope the idea of easier text and harder pattern suggests some possible approaches to the general $k>3$ case. (2) Though almost all the lemmas we proved can be generalized, to keep them readable we chosed not to. However, we do hope the techniques and the lemmas developed here find themselves applications elsewhere, like the Matching Lemma in the proof of Yao and Rivest Theorem.

A 2-way k-DFA is just like a k-DFA but each head can go both directions. A head is *blind* if it can see only end-markers. In [DG] it is proved that 2-way 2-DFA with one head blind cannot do string-matching. Obviously 2-way 3-DFA with 2 heads blind can do string-matching. Here in contrast to the impossibility result of Theorem 3.1, we prove a lower bound on the time to do string-matching required by a 2-way k-DFA with $k-1$ blind heads. We hope this can shed some light on the other important open problem concerning the lower bound of doing string-matching by a 2-way 2-DFA.

Theorem 3.2: String-matching requires $\Omega(n^2/logn)$ time for a k-head two way DFA with $k-1$ heads blind, where n is the length of the input.

Theorem 3.3: $\bar{L}=\{\#x\$yx¢\}$ (used in above proof) can be accepted by a 2-way 3-DFA with 2 heads blind in time $O(n^2/logn)$.

Remark: It is proved in [LY] that $\bar{L}$ defined above cannot be accepted by a 1-way 2-DFA. By a similar proof the language $L'=\{\#a_0\$a_1*a_2*...*a_l¢ \mid a_0=a_i$ for some $i\}$, defined and shown to be not acceptable by a 2-way 2-DFA with one head blind in [DG], is acceptable in time $n^2/logn$ by a 2-way 4-DFA with three blind heads.

It has been an interesting philosophical question [W]: is (probabilistic) checking easier than (probabilistic) generating? For example, given matrix A, B, and C, Freivalds showed (see [W]) that we can probabilistically check AB=C in n^2 time, but no one knows how to calculate AB in $O(n^2)$ time even probabilistically (open problem 2.6 in [W]). Also similarly it is known [W] that given polynomials $p_1(x),p_2(x),p_3(x)$, the probabilistic checking of $p_1(x)p_2(x)=p_3(x)$ can also be done faster than the known generating $(p_3(x))$ algorithms. Here we shall provide an example which does show that checking is easier than generating.

A PTM [G] is a TM equipped with a random number generator. It decides the next move by a random choice from two possible branches. A PTM P performs a task with error probability ϵ if it outputs the correct answer with probability $1-\epsilon$. Language L is accepted by a PTM P in time $t(n)$ if there exist an $\epsilon<1/2$ such that if $x \in L$ then P accepts x in with probability greater than $1-\epsilon$ in time $t(n)$, otherwise P accepts x with probability less than ϵ in time $t(n)$. In this section, we solely consider the 1-tape probabilistic machines (1-tape PTM's) without an extra input tape, i.e., the input is presented on this single work tape at the beginning of the computation.

Freivalds [F] proved a very interesting result that a 1-tape PTM can match two strings on 1-tape in time $O(nlogn)$ with any fixed error probability $\epsilon > 0$. In contrast we show the following.

Theorem 3.4: Consider a 1-tape PTM M, with input $x\#^{|x|}0^{|x|}$ presented on its only work tape. To move x to the 0's positions with a fixed error probability $\epsilon < 1/2$, (i.e., to output $x\#^{|x|}x$ where $x\#^{|x|}$ stays at original position) M requires $\Omega(n^2)$ time.

Remark: Comparing to the $nlogn$ probabilistic algorithm for accepting $x\#^{|x|}x$ (with any fixed small error ϵ) by a 1-tape PTM [F], this lower bound leads us to an interesting conclusion: checking is indeed easier than generating. Notice that this is not true for 1-tape deterministic or nondeterministic machines since an $\Omega(n^2)$ lower bound for accepting the palindromes were proved long time ago by Hennie [H2].

4. Open problems

There are several open questions: (1) Close the gap for 1 tape vs. 1 queue; (2) Prove that k-DFA cannot do string matching, and give a simple proof of Theorem 3.1; (3) Can 1 tape nondeterministically simulate 2 tapes in less than square time? (This question will be discussed in [L3].) Most open problems listed in [DGPR] are still open except 1 (not completely solved) and 6 (completely solved). Similar questions for off-line machines also need to be answered.

5. Acknowledgements

I am greatly indebted to Juris Hartmanis, my thesis advisor, for his guidance, criticism and encouragement. I also wish to thank Chanderjit Bajaj, Zvi Galil, John Gilbert, Luc Longpre, Joel Seiferas, Yaacov Yesha, and Zhen Zhang for very helpful discussions on various topics contained in this paper.

6. References

[A] S.O. Aanderaa, On k-tape versus (k-1)-tape real time computation, in Complexity of Computation. R. Karp Ed. (1974) pp. 75-96.

[BGW] R.V. Book, S.A. Greibach, and B. Wegbreit, Time- and tape-bound Turing acceptors and AFL's, JCSS 4,6 (Dec. 1970) pp. 606-621.

[BM] R.S. Boyer and J.S. Moore, A fast string searching algorithm, CACM 20, 10 (Oct. 1977) pp. 762-772.

[Cha] G. Chaitan, Algorithmic Information Theory, IBM J. Res. Dev. 21 (1977) pp. 350-359.

[C2] S.A. Cook, Linear time simulation of deterministic two-way pushdown automata, Proc. IFIP Congress 71, TA-2. North-Holland, Amsterdam (1971) pp. 172-179.

[DGPR] P. Duris, Z. Galil, W.J. Paul, and R. Reischuk, Two nonlinear lower bounds, Proc. 15th ACM STOC (1983) pp. 127-132. (Revised June 1983)

[DG] P. Duris and Z. Galil, Two tapes are better than one for nondeterministic machines, Proc. 14th ACM STOC (1982) pp. 1-7.

[DG1] P. Duris and Z. Galil, Fooling a two-way automaton or one pushdown store is better than one counter for two way machines, Proc. 13th ACM STOC (1981) pp. 177-188.

[F] R. Freivalds, Probabilistic machines can use less running time, Info. Processing, 77 (1977) pp. 839-842.

[F1] M. Furer, The tight deterministic time hierarchy, Proc. 14th ACM STOC (1982) pp. 8-16.

[GS] Z. Galil and J.I. Seiferas, Time-space optimal string-matching, Proc. 13th ACM STOC (1981) pp. 106-113.

[G] J. Gill, Computational complexity of probabilistic Turing machines, SIAM J. Comp. 6 (1977) pp. 675-695.

[HS] J. Hartmanis and R.E. Stearns, On the computational complexity of algorithms, Trans. Amer. math. Soc. 117 (1965) pp. 285-306.

[H2] F.C. Hennie, One-tape off-line Turing machine computations, Inf. and Control 8 (1965) pp. 533-578.

[HS1] F.C. Hennie and R.E. Stearns, Two tape simulation of multitape Turing machines, J.ACM, 4 (1966) pp. 533-546.

[HU] J.E. Hopcroft and J.D. Ullman, Introduction to automata theory, languages, and computation, Addison-Wesley (1979).

[K] A. Kolmogorov, Three approaches to the quantitative definition of information, Problems of Information Transmission, 1-1, 1-7, Jan-Mar (1965).

[KMP] D.E. Knuth, J.H. Morris, Jr., and V.R. Pratt, Fast pattern matching in strings, SIAM J. Comp. 6, 2 (Jun. 1977) pp. 323-350.

[L] M. Li, On 1 tape versus 2 stacks, TR-84-591, Dept. of Comp. Sci., Cornell University (Jan. 1984).

[L1] M. Li, Lower bounds on string-matching, TR-84-636, Dept of Comp. Sci., Cornell University (July 1984).

[L2] M. Li, Lower bounds in computational complexity, Ph.D. Thesis, Cornell University (Jan. 1985).

[L3] M. Li, Simulating two pushdowns by one nondeterministic tape in $O(n^{1.5}\sqrt{logn})$ time, abstract (Jan. 1985).

[LY] M. Li and Y. Yesha, String-matching cannot be done by a two-head one-way deterministic finite automaton, TR 83-579, Department of Computer Science, Cornell University (Oct. 1983).

[M] W. Maass, Quadratic lower bounds for deterministic and nondeterministic one-tape Turing machines, Proc. 16th ACM STOC (May 1984) pp. 401-408. (Revised summer 1984).

[P] W.J. Paul, Kolmogorov complexity and lower bounds, 2nd International Conference on Fundamentals of Computation Theory (1978).

[P1] W.J. Paul, On heads versus tapes, Proc. 22nd IEEE FOCS (1981) pp. 68-73.

[P2] W.J. Paul, On-line simulation of $k+1$ tapes by k tapes requires nonlinear time, Proc. 23rd IEEE FOCS (1982) pp. 53-56.

[P3] W.J. Paul, On time hierarchies, Proc. 9th ACM STOC (1977) pp. 218-222.

[PSS] W.J. Paul, J.I. Seiferas, and J. Simon, An information-theoretic approach to time bounds for on-line computations, Proc. 12th ACM STOC (1980) pp. 357-367.

[R] M.O. Rabin, Real time computation, Israel J. of Math, 1,4 (1963) pp. 203-211.

[R1] A. Rosenberg, On multihead finite automata, IBM J., (1966).

[R2] W. Ruzzo, *Private communication*. (1984)

[RS1] S. Reisch and G. Schnitger, Three applications of Kolmogorov-complexity, Proc. 23rd IEEE FOCS (1982) pp. 45-52.

[V] P.M.B. Vitanyi, One queue or two pushdown stores take square time on a one-head tape unit, Report CS-R8406, Center for Mathematics Computer Science, Amsterdam (Mar. 1984).

[W] D.J.A. Welsh, Randomized Algorithms, Discrete Applied Math. 5 (1983) pp. 133-145.

[YR] A.C. Yao and R. Rivest, $k+1$ heads are better than k, J. ACM, 25 (1978) pp. 337-340.