Lower Bounds in Computational Complexity (Ph.D. Thesis)

Ming Li

TR 85-663
March 1985

Department of Computer Science
Cornell University
Ithaca, New York 14853

LOWER BOUNDS IN COMPUTATIONAL COMPLEXITY

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Ming Li

January, 1985

LOWER BOUNDS IN COMPUTATIONAL COMPLEXITY

Ming Li, Ph.D.
Cornell University 1985

Several important open problems in the field of computational complexity are resolved in this thesis. The results are naturally classified into two categories.

Class I. We study the relationship among different complexity classes.

(1) If $NTIME[R(n)] \subseteq PSPACE$, where $\lim_{n \to \infty} \frac{n^k}{R(n)} = 0$ for all k, then all higher corresponding $NTIME$ versus $SPACE$ classes are different. This answers an open question in [HIS].

(2) The problem of separating nondeterministic time hierarchies by sparse sets is studied. Among other results, an open problem in [SFM] is solved: if $\frac{T_1(n+1)}{T_2(n)} \to 0$, then $NTIME[T_2]$-$NTIME[T_1]$ contains a tally set.

(3) Among the oracles we constructed, we present a very short new construction for an oracle A, first obtained in [S], such that $NP^A \cap CoNP^A$ does not have complete sets.

Class II. The Kolmogorov-complexity is used to obtain lower bounds.

For on-line computations,

(1) it requires $\Omega(n^2)$ time to simulate a 2 pushdown store machine deterministically by a 1 tape machine, this settles an old open problem listed in [DGPR];

(2) it requires $\Omega(n^{1.5}/logn)$ time to nondeterministically simulate a 2 pushdown store machine by a 1 tape machine, this greatly improves an $\Omega(nlogn)$ lower bound of [DGPR];

(3) it requires $\Omega(n^2/logn\ loglogn)$ time to nondeterministically simulate a 2 tape machine by a 1 tape machine, this is a minor improvement of an $\Omega(n^2/log^2n\ loglogn)$ lower bound contained in [M]; and

(4) it is shown that the $\Omega(n^2)$ tight lower bound for the nondeterministic case cannot be obtained by the languages (which are shown to be acceptable by a 1 tape machine in $O(n^2 loglogn/\sqrt{logn})$ time) used in [M] or [F].

For string-matching,

(5) three-head one-way DFA cannot do string-matching, this answers the case $k=3$ of an open question in [GR]: whether some k-head DFA can do string-matching;

(6) two other related lower bounds are obtained.

Biographical Sketch

Ming Li was born in Beijing, China on July 16, 1955.

After graduating from The High School of Peking University, he spent two years in Su-Jia-Tuo village, Beijing. Then he studied at Huanan Institute of Technology, Canton, and worked as a computer programmer. He started his graduate studies in computer science at The Graduate School of Academia Sinica in 1978. In 1980, he attended Wayne State University and received an M.S. in Computer Science.

In fall, 1981, Li joined the Computer Science Department of Cornell University where he received an M.S. in 1983.

Dedication

To my parents.

Acknowledgements

I sincerely thank Juris Hartmanis, my thesis advisor, for his encouragement, guidance, and great patience. His personal and professional integrity influenced me deeply. I would also like to thank Tom Coleman and Anil Nerode, who cheerfully served on my committee.

Professors P. Fejer, Z. Galil, J. Gilbert, D. Gries, R. Kannan, and J. Seiferas deserve my special thanks for their kind help. My gratitude also goes to Professors W. Grosky, R. Jain, Z.Y. Ling, S.N. Murthy, G.B. Wang, D.Z. Wei, and H.S. Yang.

I am grateful to C. Bajaj, L. Longpre, Y. Yesha, and Z. Zhang for many illuminating discussions. Many other friends helped me greatly with this thesis. I warmly thank them all: B. Alpern, C. Buckley, R. Harper, J. Hook, P. Mark, G. Neiger, V. Nguyen, K. Perry, J. Prins, and T.K. Srikanth.

Finally, I thank the Chinese Academy of Sciences and the National Science Foundation (Grant number DCR-8301766) for their financial support.

Table of Contents

CHAPTER 1

Introduction

1.1. Historical Background

One of the major trends in modern mathematics and computer science is a systematic investigation of what cannot be done. The well-known Galois theorem and Godel incompleteness theorem kindled the research in impossibility proofs. The result of Galois, which showed that not all polynomials of higher than fourth degree were solvable by radicals, ended the hopeless efforts of hundreds of years in searching for such solutions. Godel questioned the essence of mathematics and showed that not every 'truth' is provable in a given formal axiomatic system. Following Godel's work, Turing introduced a new tool, the Turing machine, to formalize the concept of effective computability. The Turing machine, which turned out to be as powerful as our modern computers, provides a mathematically precise and intuitively satisfying tool for proving that problems were not solvable, or in the context of this thesis, that problems were not *efficiently* solvable.

To show something can be done is an existential question for which one only needs to expose a way of doing it; whereas to show something cannot be done is a universal question for which one must prove that *every* method fails. In the past few decades, researchers have successfully discovered many (recursively) unsolvable problems and proved many independence results.

With the development of digital computers and the emergence of a systematic study of computation, computational complexity theory raised questions about the feasibility of computations. In practice, not only is it important to know whether a given problem is effectively solvable, but it is also vital to know whether the problem is *feasibly* solvable by a modern computer. More precisely, it is desired to know how much resources such as time and space we need to solve the problem. A problem that takes a billion years to solve by any computer is *practically unsolvable*.

Therefore we need a mathematical model to allow us talk about not only computability but also *efficiency*. To formalize the natural concept of computational efficiency, Hartmanis, Lewis, and Stearns [HLS] started a systematic investigation of the Turing machine based computational complexity. They showed that for each time or space bound $f(n)$, there were problems that requires $f(n)$ amount of time or space with respect to the problem size n. Further, for each time (or space) bound $f(n)$ there exist problems which can be solved in time or space $f(n)$ but not much less than $f(n)$. The field bloomed over the past 20 years. Some natural problems were proved be hard, that is, impractical to compute. We now have a better understanding of what is *hard* and what is *easy*, just like the researchers in the 1930's and 1940's who started to understand what was *computable* and what was not.

To get a feeling about the concept of time complexity, let us take the King Arthur's example: King Arthur was arranging a round table meeting with his 150 (149) knights. It was known that there were many pairs of knights who hated each other. King Arthur wanted to arrange these knights around the table so that no pair of knights who hated each other would be sitting side by side. Obviously, the problem is solvable. That is, it can be decided whether such an arrangement exists, and if it exists it can be found. But King Arthur could not make a good arrangement until Merlin asked his *oracle*.

We can give King Arthur a trivial algorithm that tries all possibilities. But it takes about $O(n!)$ steps for n knights. Let's pick a computer running one billion steps per second. It still takes more than a billion years before this computer can reach a good arrangement. Now suppose we have the power of 'guessing'. After correctly guessing an arrangement (which is equivalent to Merlin's oracle), King Arthur should not have any trouble to quickly check the correctness of the guessed arrangement in n^2 steps (neither do our modern computers and the Turing machines). The power of "guessing" is what we call the ***nondeterminism***. Here we are confronted with this problem: can we ***prove*** that King Arthur was not foolish, that is, that the problem was indeed too hard? This is equivalent to ask whether ***nondeterminism*** is much more powerful than ***determinism***. The theoretical computer scientists are still not able to answer the question. According to the theory of computational complexity, this question is abstracted to the question of $P=?NP$, where P and NP are classes of languages that can be accepted in deterministic and nondeterministic polynomial time by Turing machines, respectively.

To answer this question, it obviously would help if we can find a ***hardest*** problem in *NP* such that $P{=}NP$ iff this problem is in P. Hundreds of natural ***hardest*** problems (including the King Arthur's problem) of great practical importance in *NP* have been found [C, GJ], but it is not known whether they are in P. In computational complexity theory, problems are classified into different classes according to their time/space complexity, such as, P, *NP*, *PSPACE* (which is the class of languages accepted in polynomial space by a Turing machine) and ***etc***. Therefore our task left is to find the exact relationship among all these complexity classes.

Besides the questions of ***nondeterminism*** versus ***determinism***, there are many practical problems in P that need fast (low order polynomial time) computer solutions. For those questions for which no good enough algorithms have been discovered, we would like to know whether there are faster algorithms at all. Even for those problems that already have efficient solutions we may still want to know if the known solutions are the ***best*** possible. We are now facing one of our major tasks in computational complexity: investigating the inherent complexity (or the ***lower bound*** in our jargon) of each problem.

In the field of theoretical computer science, on the one hand, better and better algorithms for many practical or theoretical problems have been discovered; on the other hand, for most of the non-trivial problems, it is not known whether the existing algorithms are the best. Although a great deal of effort has been expanded on lower bound research, we are still quite far away from a satisfactory theory of lower bounds. Many of the lower bounds have been obtained only for restricted models of computation. Some important techniques have been developed but it seems a great deal are still missing.

The purpose of this thesis is to develop new techniques to prove lower bounds of computations. Using these new techniques, we answer several important open questions in computational complexity theory. We will provide several new lower bounds and new separation results for complexity classes, as well as some oracle results for those questions that we are not able to settle. The oracles do not give us solutions to the problems, but they do suggest that with current techniques the task is formidable.

We shall use standard definitions and notation in computational complexity, which are assemblied in Appendix 1. Unusual and new technical definitions will be given where they appear.

1.2. Thesis Overview

In this thesis, we resolve several long-standing open problems in the field of computational complexity. New techniques are introduced along with the solution of each problem. The results in this thesis are naturally divided into two categories.

In the first part, Chapter 2, of this thesis we use conventional methods to study the structure of and relation among different complexity classes. In the field of computational complexity, there are hundreds of unsolved questions, like $PSPACE = ?NP = ?P$, $ESPACE = ?NE = ?E$, *etc*. Understanding their precise relationship is one of the major goals of the field. Before these goals can be achieved, it is important to understand our problems, that is, the connection between these classes and their structures. An important connection among these complexity classes was established by Hartmanis [H], who recently showed that there is a sparse set in $NP-P$ if and only if $NE \neq E$. The result is, of course, generalized to other complexity classes and sparser sets [HIS,HY]. In [HIS], a theorem of the following type was proved. If NP is as powerful as some deterministic super-polynomial time class, for example if $DTIME[n^{\log n}] \subseteq NP$, then all higher corresponding nondeterministic and deterministic time complexity classes separate. In [HIS], a similar question on nondeterministic time versus space was raised. That is, if for some super-polynomial R any language acceptable in nondeterministic time R is also accepted in $PSPACE$, say $NTIME[n^{\log n}] \subseteq PSPACE$, then what happens to the higher nondeterministic time versus space classes? This question started the research of this part of the thesis. We shall give a complete study of this question. Not only do we give an answer to this question, but also we show why the methods used for proving the corresponding result for $DTIME$ versus $NTIME$ fail. In the course of this study, several other questions of independent but related interest arose. We also resolve these questions.

One way to answer the above question is to find very sparse sets in the *NTIME* hierarchy. The reason is that if we have arbitrarily sparse sets in, say $NTIME[n^{\log n}]-NP$, then by the upward separation method of [HIS] we can separate all higher *NTIME* versus *SPACE* classes. But the task of finding arbitrarily sparse sets, or even tally sets [SFM], in the *NTIME* hierarchy was an unsolved classical open question for many years.

The hierarchy theorems have been studied from the beginning of complexity theory. The *DTIME* hierarchy was established by Hartmanis, Lewis, and Stearns [HLS, HU] in 1965. By the nature of the diagonalization used in [HLS], sparse sets exist in the *DTIME* hierarchy. The technique unfortunately does not generalize to the *NTIME* hierarchy since the direct diagonalization is not possible. *NTIME* hierarchy was not obtained until, in 1973, Cook [C1] proved $NTIME(n^a) \neq NTIME(n^{a+\epsilon})$. The tightest *NTIME* hierarchy is due to Seiferas, Fischer, and Meyer who in 1973 proved: if $\lim_{n \to \infty} T_1(n+1)/T_2(n)=0$, then $NTIME(T_2(n))-NTIME(T_1(n))$ is not empty. Unlike the *DTIME* hierarchy the proof here [SFM] used a novel diagonalization with padding and a recursion theorem which did not yield very sparse or even tally sets for the similar dense hierarchy. This question, as indicated above, has a direct implication to the problem mentioned in the last paragraph. This question was left open from 1973 (in an early version of [SFM]): can the similar separation be done by tally sets? Seiferas, Fischer, and Meyer [SFM] partially answered the question in 1978 by replacing the $n+1$ above by $n+f(n)$ for some very slowly growing function f.

In this thesis, we answer this question by showing that if $\lim_{n \to \infty} T_1(n+1)/T_2(n)=0$ then there is a tally set in $NTIME[T_2(n)]$-$NTIME[T_1(n)]$. The proof also greatly simplifies the proofs of Cook [C1] and Seiferas, Fischer, and Meyer [SFM]. (After the above results were obtained, the author was informed by Joel Seiferas that S. Zak [Z] in Czechoslovakia had also solved the problem by somewhat different methods. His result is to appear in Theoretical Computer Science.) Furthermore, we shall show how sparse a set can be in the NTIME hierarchy. On the negative side, we shall give oracle results to show that the results we have obtained are the best one can do before new techniques are invented.

The assumption that NP computations can accept languages in $DTIME[R(n)]$ for some super-polynomial function R has the consequence that $NP \cap CoNP$ computations can also accept languages in $DTIME[R(n)]$. Here again we meet another classical question in complexity theory: $NP = NP \cap CoNP$? As we have seen above, the $NTIME$ hierarchy is quite dense, but so far there has not been ***any*** nontrivial hierarchy obtained for $NTIME \cap CoNTIME$. Should there not exist a dense $NTIME \cap CoNTIME$ hierarchy, we could immediately prove that $NTIME[t(n)] \neq NTIME[t(n)] \cap CoNTIME[t(n)]$. Unfortunately, this question is closely related to the existence of complete sets in the intersection classes. We show that if for each t(n) there is a complete set for

$NTIME[t(n)] \cap CoNTIME[t(n)]$, then there is a tight hierarchy. Again should a class like $NP \cap CoNP$ not have complete sets, then NP$\neq$CoNP. Sipser has constructed an oracle X such that $NP^X \cap CoNP^X$ does not have complete sets [S], but his proof was long and complicated. We present a very simple and short construction of Sipser's oracle.

In the second part, Chapter 3, of this thesis we will use a recently discovered tool, Kolmogorov-complexity (from now on K-complexity), to resolve (or partially resolve) some long standing open questions. We develop new techniques for deriving lower bounds by the use of K-complexity of finite strings. Informally, the K-complexity of a finite string x is the size of the smallest TM that starts with empty input and prints the string x. x is *random* if $K(x) \leq |x|$. That is, if a string is random, then the only way for a TM to print it is to store it in the memory and copy it out, it cannot be *compressed*. The main idea in these proofs is as follows. A random string is chosen to construct the input. Then if a machine does not run for a long enough time on this (particular!) input, we will find a way to reconstruct our random string x from a small amount of information from the assumed short computation. And this gives us a short program to print x, implying that x is not random.

The advantage of using K-complexity is that we can put our hand on a hard and otherwise not constructible input. The use of K-complexity in lower bound proofs was introduced by Barzdin and Paul [P]. Here we extensively study the use of K-complexity in lower bound problems. We solve some old open problems and give new simple solutions to some solved problems.

We first consider deterministic on-line computations where the input tape is read only from left to right (1-way). A classical question, probably among the oldest in complexity theory, was how efficiently a one tape on-line TM could simulate a k tape on-line TM, in other words, whether more tapes were much faster than one. The first result in this direction was obtained by Hartmanis, Lewis, and Stearns [HLS] who showed that one tape (nondeterministic) TM can simulate k tape (nondeterministic) TM in quadratic time. In 1963, Rabin [R] proved that two tapes are better than one tape for real time computations. In 1982 Paul established the first nonlinear lower bound: it requires $\Omega(n(logn)^{1/2})$ time to simulate two tapes by one tape for on-line computations. In 1983, Duris, Galil, Paul, and Reischuk [DGPR] obtained an $\Omega(nlogn)$ lower bound.

We now settle this open problem by proving an $\Omega(n^2)$ tight lower bound. This lower bound meets the $O(n^2)$ upper bound provided in [HLS]. This problem was independently solved by Maass [M], the author [L], and Vitanyi[V]. Concerning the nondeterministic case, we provide an $\Omega(n^{1.5}/logn)$ lower bound for one tape nondeterministic on-line TM simulating two pushdown store on-line TM. This sharply improves the $\Omega(nlogn)$ lower bound of [DGPR]. For the nondeterministic case of one tape versus two tapes, Maass [M] obtained a $\Omega(n^2/(logn)^2loglogn)$ lower bound. We improve this lower bound to $\Omega(n^2/lognloglogn)$. We also show that the tight $\Omega(n^2)$ lower bound can not be obtained by the language used there by a graph separator theorem.

Next, we use K-complexity to give a lower bound for string-matching. The string-matching problem is defined 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. String-matching is a very important practical problem. It is extensively used in, for example, text-editing and library searches, etc. Linear time algorithms for string-matching were developed by Boyer and Moore[BM], Cook [C2], and Knuth, Morris, and Pratt [KMP]. In 1981, Galil and Seiferas [GS] proved that a 6 head two-way deterministic finite automaton, DFA, can do string-matching in linear time. The full power of Turing machine is not needed for the linear time string-matching. They raised the question of whether some k-head 1-way DFA (where the k heads scan the input only from left to right) can do string-matching. We develop new methods, settling the cases of $k=2$ and $k=3$ by showing that no 3-DFA can do string-matching, that is, accept the language $L=\{\#x\$y¢ \mid x$ is a substring of $y\}$. We also study the problem of doing string-matching by a k-head 2-way DFA with $k-1$ heads being able to see only endmarkers. Using the methods we developed, we also give a simple proof of 'k heads are better than $k-1$ heads for DFAs', first proved by Yao and Rivest [YR].

Our last lower bound derived by K-complexity shows that *checking* is easier than *generating* [W]. In [L2], we proved that one tape on-line nondeterministic TM equipped with a random number generator can simulate a two tape machine in less than quadratic time with arbitrarily small error probability, $\epsilon>0$. The proof depends on the fact that on a single tape the string $\#x\#x\#$ can be accepted very fast, in $|x|log|x|$ time, probabilistically [F]. The nondeterminism above may be removed if we can show that with $\#x\#$ on a single tape, the string $\#x\#x\#$ can be generated faster than quadratic time probabilistically with error probability $\epsilon<1/2$. Unfortunately, we prove this is not possible: on a single tape, probabilistically moving a string (requiring quadratic time) is harder than probabilistically checking the equivalence of two strings. It is interesting to note that this result is true for neither deterministic nor nondeterministic machines.

CHAPTER 2

Separating Complexity Classes

2.1. Introduction

One of the most important problems in computational complexity theory is to determine the relation among the natural complexity classes, e.g. P, NP, $PSPACE$, *etc.* Besides the classic $P=?NP$ and $NP=?PSPACE$ problems, there are a whole spectrum of other important problems like $E=?NE=?ESPACE$, $EE=?NEE=?EESPACE$, etc. Even a solution to $P\neq NP$ or $NP\neq PSPACE$ would not necessarily settle any of the other problems. Little is known about the relation among these complexity classes. Recently, an important connection was established in [H]. The following was shown.

Theorem 2.1A [H]: There is a sparse set in $NP-P$ iff $NE\neq E$. □

The generalization of this result to other complexity classes and sparser sets can be found in [HIS,HY]. Furthermore, an upward separation method by the means of sparse sets was developed in [H,HIS]. The basic idea is as follows. If there is a very sparse set S in, say, $NP-P$ then we can represent each string in S by its index using a very small amount of space, and construct a new set S'. The new set S' will be in $NE-E$ or $NEE-EE$, ..., depending on how sparse S is. In particular, Hartmanis, Immerman, and Sewelson proved the following theorem.

Theorem 2.1B [HIS]: Let $R(n)$ be time constructible and for all $k \geq 1$

$$\lim_{n \to \infty} \frac{n^k}{R(n)} = 0$$

Then $DTIME[R(n)] \subseteq NP$ implies that for any monotonically increasing, time-constructible $T(n)$, where $T(n) > 2^n$ and $2^{T^{-1}(n)}$ is computable in time polynomial in n,

$$\bigcup_{c \geq 1} NTIME[T(n)^c] \neq \bigcup_{c \geq 1} DTIME[T(n)^c]. \qquad \square \text{ (Theorem 2.1B)}$$

The idea was to construct arbitrarily sparse sets in $DTIME[R(n)]-P$ by diagonalization. These sets, from the assumption, are in turn in $NP-P$. Then we apply the upward separation method of [HIS] as explained above. Notice that this theorem is true for all relativized computation.

After seeing Theorem 2.1B, it is natural to ask that whether a similar result is also true for the *NTIME* versus *SPACE* classes. In [HIS], this question was raised: assuming, say, $NTIME[n^{\log n}] \subseteq PSPACE$, can we prove that $NE \neq ESPACE$, $NEE \neq EESPACE$, and so on? The proof for Theorem 2.1B fails here, since we cannot apply standard diagonalization methods in nondeterministic time to construct arbitrarily sparse sets in $NTIME[n^{\log n}]-NP$. As a matter of fact, we will construct an oracle A such that $NTIME^A[n^{\log n}]-NP^A$ does not contain very sparse sets. We now summarize our problems.

Problem1:

Let R be a function such that $\frac{n^k}{R(n)} \rightarrow 0$ for all k. Assume $NTIME[R(n)] \subseteq PSPACE$, can we conclude that $NE \neq ESPACE$, $NEE \neq EESPACE$, and so on?

Problem2:

Let R be defined as above. Do there exist arbitrarily sparse sets in $NTIME[R(n)]\text{-}NP$, or in $NTIME[n^{\log n}]\text{-}NP$? If this is true, then Problem 1 can be answered by the methods used to prove Theorem 2.1B.

Problem3:

If Problem 2 is not true, then what is the smallest class C such that arbitrarily sparse sets exist in $C\text{-}NP$.

Before we present a stronger answer to Problem 1, we answer Problems 2 and 3 and some other related questions to provide a complete story; each related problem is of independent importance. Among others, we give a simple solution to a classical open question concerning tally sets in the *NTIME* hierarchy.

We need one more notation: If C is a complexity class, then

$C\,|_{1^{f(n)}} = \{ S \mid S \in C \text{ and } S \subseteq \{ 1^{f(n)} \}\}$.

2.2. Difference between DTIME vs NTIME and NTIME vs SPACE

We first explain the proof of Theorem 2.1B by an example and then give an oracle which suggests that the same proof would not work for *NTIME* vs

SPACE.

Example: Assume $NTIME[n^{logn}] \subseteq NP$ we prove $NE \neq E$ (and $NEE \neq EE$, etc.). Let $M_1, M_2, \ldots$ be a enumeration of P time machines where each machine is enumerated infinitely often. Define L=$\{1^n \mid M_n$ rejects 1^n in less than $n^{loglogn}$ time$\}$. Using standard techniques developed in [HLS] (see also [HU]), we can design a machine M_L that on input 1^n simulates M_n on 1^n in n^{logn} time and M_L accepts iff M_n rejects in less than $n^{loglogn}$ steps. Obviously $L = L(M_L)$. It is not hard to see that $L \neq L(M_i)$ for all i, since each machine is enumerated infinitely often. Therefore $L \in DTIME[n^{logn}]-P$ and $L \in NP-P$ by our assumption. Then $L' = \{n \mid 1^n \in L\} \in NE-E$ because the binary n gives us exponential time to compute. Notice that, from the proof, the assumption of $DTIME[n^{logn}] \subseteq NP$ can be replaced by the weaker condition $DTIME[n^{logn}] \mid_{1^n} \subseteq NP$, or even $DTIME[R(n)] \mid_{1^{f(n)}} \subseteq NP$ for $R(n)$ a super-polynomial function and $f(n)$ any time-constructible function.

One would naturally like to obtain a similar proof for *NTIME* vs *SPACE* under a similar assumption, say, $NTIME[n^{logn}] \subseteq PSPACE$. Unfortunatedly, the diagonalization we used cannot be carried out in the nondeterministic time hierarchy. And as the following oracle indicates, with current techniques, we are not likely to be able to obtain arbitrarily sparse sets in $NTIME[n^{logn}]-NP$.

Theorem 2.2: For each recursive function $f(n) > 2^{2^{2^n}}$, there exists a recursive oracle A such that $NTIME^A(n^{clogn}) \mid_{1^{f(n)}} \subseteq NP^A \mid_{1^{f(n)}}$.

Proof of Theorem 2.2: We will recursively construct oracle A. The set A' will be used to indicate strings that cannot be put in A. From the condition on f, we know that $f(n+1)-f(n)>n$, and $f(n+1)>f(n)^{logf(n)}$. Let $N_1,N_2,\cdots$ be a standard enumeration of $NTIME^A[n^{logn}]$ machines.

Construction of A:

stage 0: $A, A'=\phi$.

stage $i+1$:

Consider only the machines in $N_1,N_2,\ldots,N_i$ which are not canceled by (b) at this stage. Run each of them on input $1^{f(i)}$.

(a) If none of the above machines has an accepting path, then do nothing on A (nor A'). Go to stage $i+2$.

(b) Suppose $N_{i_0}{}^A$ (any one of the above) accepts $1^{f(n)}$, then put the strings which are queried and not in A into A', add some y of length $f(i)+i_0$ which is not in A' to A. Cancel N_{i_0} from the list. Repeat stage $i+1$ for the rest of the machines.

end-of-construction.

Claim 1: String y in (b) exists.

Proof of Claim 1: Since $2^{f(n)}>nf(n)^{logf(n)}$, a desired y can be choose. □ (Claim 1)

Claim 2: Stage $i+1$ is repeated at most i times (injury happens $< i+1$ times).

Proof of Claim 2: Since we reserved strings on the accepting paths into set A', an accepting computation will not get injured. □ (Claim 2)

Claim 3: $NTIME^A(n^{clogn})|_{1^{f(n)}} \subseteq NP^A|_{1^{f(n)}}$.

Proof of Claim 3: For any N_i^A in $NTIME^A(n^{clogn})|_{1^{f(n)}}$, we will accept L($N_i^A$) in NP^A time as follows: for input $1^{f(j)}$, if $j<i$, we build the answer into the finite control of the machine; if $j \geq i$, then we query the oracle A and accept if there is a y in A such that $|y|=f(j)+i$. This is done in linear time. □ (Claim 3)

Theorem 2.2 is proved by Claims 1, 2, and 3. □ (Theorem 2.2)

Remark 2.2*A*: For above oracle A, $NP^A \neq P^A$.

Remark 2.2*B*: M. Krentel has proved: for each N_k such that $L(N_k) \in$ NTIME[n^{logn}], there is an oracle A such that $N_k^A \subseteq NP^A$, where $L(N_k) \subseteq \{1^{f(n)}\}$ for $f(n) > 2^{2^{2^n}}$. The oracle here was obtained after seeing Krentel's oracle. (The author later discovered that a similar technique has been used by Rackoff and Seiferas in a different context [RS].)

Under this oracle A, there cannot be very sparse tally sets exist in $NTIME^A[n^{clogn}]$-NP^A, which is not true for $DTIME^B[n^{clogn}]$-P^B for any B.

2.3. Very sparse sets in the NTIME hierarchy

Before we turn to our solution to Problem 1, we answer the following question. Regardless of above oracle, what can we say about the sparse sets in the unrelativized *NTIME* hierarchy?

Here we meet one of the oldest problems in the computational complexity. In their fundamental paper [HLS] in 1965, Hartmanis, Lewis, and Stearns obtained the well-known *DTIME* hierarchy by the method of diagonalization. For the NTIME hierarchy, in 1973, Cook [C1] showed that, for any real number $\epsilon>0$, $NTIME[n^a]\neq NTIME[n^{a+\epsilon}]$. Seiferas, Fischer, and Meyer refined this result as follows [SFM]: if $T_1(n+1)=o(T_2(n))$, then $NTIME[T_2]\text{-}NTIME[T_1]\neq 0$. They also raised the question whether the same separation of *NTIME* hierarchy can be done by tally (single letter) languages. In a later version of their paper [SFM], they partially answered this question as follows. If $T_1(n+f(n))=o(T_2(n))$, then there is a tally set in $NTIME[T_2]\text{-}NTIME[T_1]$, where f is some very slow growing function.

Here we give a general solution of this question. The proof is also much simpler than the old proofs of [C1] and [SFM]. The author was later informed by Joel Seiferas that the case (A) below was obtained by S. Zak earlier. The proof here is different and independent [L3].

Theorem 2.3: For time-constructible functions T_1 and T_2, if $T_1(f_k(n))=o(T_2(n))$, then there exists a set $S \subseteq \{ 1^{\left.2^{\cdot^{\cdot^{2^n}}}\right\}k\ 2's} \}$, and $S \subseteq$

$NTIME[T_2(n)]-NTIME[T_1(n)]$, where

(A) $f_0(n)=n+1$;

(B) $f_1(n)=2n$;

(C) $f_2(n)=n^2$;

(D) For $k\geq 3$, $f_k(n)=2^{\cdot^{\cdot^{\cdot^{2^{(\log^{k-3}n)^{\log^{k-2}n}}}}}}$ (with k-3 2's), where $\log^l n$ means taking log l times, and $\log^0 n=n$.

Claim 1: For every natural number n, there exist unique l, $l>0$, and k, $k\geq 0$, such that $n=2^{l-1}+k2^l$.

Proof of claim 1:

1. Existence: Let $2^{l-1}|n$, and $2^l|n$. Then $n=k'2^{l-1}$ for some odd k'. Let $k'=2k+1$, we have $n=k2^l+2^{l-1}$.

2. Uniqueness: Suppose $n=2^{l-1}+k2^l=2^{j-1}+k'2^j$, w.l.g. let $j<l$. We divide both sides by 2^{j-1}, then we have $2^{l-1-(j-1)}+k2^{l-(j-1)}=1+2k'$. But now 2 divides the left hand side, not the right hand side. Contradiction. □ (claim 1)

The following Claim was proved by Book, Greibach, and Wegbreit[BGW]. The idea of the proof will be given in the next section where this result is used again.

Claim 2: Any NTM running in time $T(n)$ can be simulated by a three-tape nondeterministic TM running in time $T(n)$. □ (claim 2)

Proof of Theorem 2.3: In the following we will give the proof of our theorem in which we use the new idea of constructing a "Universal diagonal language".

Case (A): For $T_2(n)$, we find a nondecreasing function $T(n)$ large enough to allow the deterministic simulation of machines that run in nondeterministic $T_2(n)$ time. $T(n)>n^{2^{T_2(n)}}$ suffices. To get a contradiction, suppose any tally set in $NTIME[T_2(n)]$ is also in $NTIME[T_1(n)]$.

In the following, all the machines will take inputs only of form 1^n, other inputs are immediately rejected. And for 1^n, we define m by: $T^m(0)\leq n$; and $T^{m+1}(0)>n$, where $T^{m+1}(n)=T(T^m(n))$. k, l are defined such that $\lceil m/2\rceil=2^{l-1}+k2^l$.

For the time bound $T_2(n)$, we design a universal machine U_2 (on tally sets). For input 1^n (rejects other inputs), U_2 finds maximum m such that $T^m(0)\leq n$, then guesses an l such that $\lceil m/2\rceil=2^{l-1}+k2^l$, and verifies this, all in $O(n)$ time. Then U_2 simulates M_k on input 1^n, works for $T_2(n)$ time, U_2 accepts only if M_k accepts in $T_2(n)/dk$ time. Note that this can be done since the nondeterministic simulation of any TM by a 3-tape TM can be done using the same amount of time, by Claim 2.

Now by assumption, there exists a U_1 running in T_1 time accepting $L(U_2)$.

Define a universal diagonal machine M_e running in time $T(n)$ as follows:

M_e will diagonalize over all $T_2(n)$ machines using input 1^n for $n = T^m(0)$, for some odd m. Each machine is diagonalized infinitely often. On input 1^n, M_e checks in linear time if $n = T^m(0)$ and m is odd; M rejects otherwise. Then M_e computes $(m+1)/2 = 2^{l-1} + k2^l$. Then M_e deterministically simulates M_l, the lth $T_2(n)$ machine, on 1^n and does the opposite. Because we assumed M_l runs in $T_2(n)$ time, $T(n)$ time is enough to do the simulation. And because we assumed each TM code is represented infinitely often by attaching prefixes, we know that $L(M_e)$ is in $NTIME[T(n)]$-$NTIME[T_2(n)]$.

Define $L_k(M_e) = \{1^n \mid n = T^m(0)$, where m odd and $(m+1)/2 = 2^{l-1} + k2^l$, and $1^n \in L(M_e)\}$. It is easily seen that $L_k(M_e) \in NTIME[T(n)]$-$NTIME[T_2(n)]$ for all k.

Now we define a machine M'. On input 1^n, M' finds the maximum m such that $T^m(0) \leq n$ in $O(n)$ time, and

(1) If m is even, and $T^m(0) = n$, then behaves like M_e on $1^{T^{m-1}(0)}$;

(2) If m is odd, then pads 1, and behaves like U_1 on 1^{n+1};

(3) Rejects otherwise.

So for input 1^n, M' runs in time $n + T_1(n+1)$. Note that we can design M' such that it has only 3 tapes by Claim 2. Now let e' be a TM code for M' and we define:

$L(M_{e_0}) = \{1^n \mid \lceil m/2 \rceil = 2^{l-1} + e'2^l,\ 1^n \in L(M')\}$, m is the maximum number such that $T^m(0) \leq n$.

So M_{e_0} runs in time $n+T_1(n+1)$. For n large enough, $n+T_1(n+1) \leq T_2(n)$. Thus M_{e_0} satisfies the following. For input 1^n, let m be the maximum such that $T^m(0) \leq n$, then

(a) if m is even, and $n=T^m(0)$, then M_{e_0} on 1^n agree with M_e on $T^{m-1}(0)$, for those m such that $\lceil m/2 \rceil = 2^{l-1} + e'2^l$;

(b) if m is odd and $\lceil m/2 \rceil = 2^{l-1} + e'2^l$, then M_{e_0} on 1^n agree with M' on 1^{n+1} which in turn agree with M_{e_0} on 1^{n+1};

(c) if m even but $n > T^m(0)$, M_{e_0} rejects.

Finally, we define M'_{e_0} operating as follows: on input 1^n, M' checks whether $n=T^m(0)$ for some odd m, and

(a) if true, acts like M_{e_0} on 1^n;

(b) if false, rejects.

Now we have $L(M'_{e_0}) = L_{e'}(M_e)$. And M'_{e_0} on input 1^n runs in time $C_1 T_1(n+1)$, which is in $NTIME[T_2(n)]$. But this contradicts to the fact that $L_{e'}(M_e)$ is not in $NTIME[T_2(n)]$. □ (of Case A)

Cases (B), (C), and (D): From above proof, to get the proofs for cases of (B), (C), and (D) we can simply change the padding to be $f_k(n)$-(n) instead of 1, and change all $n+1$ to $f_k(n)$. One more detail needed to be mentioned: if $n < T^m(0) < f_k(n)$ then we pad only $T^m(n)-n$. □ (of Theorem 2.3)

Corollary 1: If $T_1(n+1)=O(T_2(n))$, and $T_1(n)=o(T_2(n))$, then there is a tally set in $NTIME[T_2(n)]$-$NTIME[T_1(n)]$.

Proof : [SFM]'s proof works for tally set also (except should be careful with the translation argument for tally sets). □ (Corollary 1)

Example: There is a tally set in $NTIME[2^{2^{n+1}}]$-$NTIME[2^{2^n}]$. This solved an open problem mentioned in [SFM].

Corollary 2: There is a tally set in $NTIME[T_2]$-$\cup\{NTIME[T_1] \mid T_1(n+1) =o(T_2(n))\}$. □ (Corollary 2)

Corollary 3: There is a set in $\{1^{\left.2^{\cdot^{\cdot^{\cdot^{2^n}}}}\right\}k\ 2's}\}$ for any k separating $NTIME(2^{c\sqrt{n}})$ from NP.

Proof : Apply Theorem 2.3. □ (Corollary 3)

Remark: Corollary 3 answers our Problem 3 listed in Section 2.1. Under the stronger assumption of $NTIME[2^{c\sqrt{n}}]\subseteq PSPACE$ (we can construct an oracle for which this containment holds), we can conclude the similar result like Theorem 2.1B. Note that one can easily construct an oracle B such that $NTIME^B[2^{c\sqrt{n}}]\subseteq PSPACE^B$. The oracle constructed in Theorem 2.2 tells us, with the old method, this is the best separation result we can get. Taking $k=3$, we have $f_3(n)=n^{logn}$. This implies there exists a subset of $\{1^{2^{2^{2^n}}}\}$ in $NTIME[n^{logn}]$-NP, which meets the limit put by the oracle in Theorem 2.2.

2.4. The case of NTIME versus SPACE

Now we solve Problem 1 listed in Section 2.1, by using a method different from the one used to prove Theorem 2.1B. We will prove a stronger version of Problem 1.

Theorem 2.4: Let $R(n)$ be time-constructible and for all $k \geq 1$

$$\lim_{n \to \infty} \frac{n^k}{R(n)} = 0$$

If for some time-constructible function f, $NTIME[R(n)]\,|_{1^{f(n)}} \subseteq PSPACE$, then for any monotonically increasing time-constructible $T(n)$, where $T(n) \geq 2^n$ and $2^{T^{-1}(n)}$ is computable in polynomial time in n,

$$\bigcup_{c \geq 1} NTIME[T(n)^c] \neq \bigcup_{c \geq 1} SPACE[T(n)^c].$$

Example: Under the conditions of Theorem 2.4, we have $NP \neq PSPACE$, $NE \neq ESPACE$, $NEE \neq EESPACE$, etc.

We need following lemma proved by [BGW].

Lemma 1[BGW] : Any TM running in time $T(n)$ can be simulated by a 3-tape nondeterministic TM running in time $T(n)$.

The idea of Lemma 1's proof: To simulate a k-tape machine by only three tapes, use one tape with k tracks [See HU]. The three tape machine first 'guess' the computation of the k tape machine: at time t, the k-tape machine is in state q_t, its head i is reading symbol a_t, and writing symbol b_t, for

$i=1, \cdots, k$. This information is recorded on one track per tape (head), ordered according to time. Then verify the correctness of the 'guessing' track by track. See also [SFM] for a short proof. □ (Lemma 1)

Proof of Theorem 2.4: Let $N_1, N_2, \cdots$ be a standard enumeration of 3-tape nondeterministic polynomial time bounded TM's such that each machine appears infinitely often (by attaching trivial prefixes). This enumerates all languages in NP.

Now we define a universal machine M running in $NTIME[R(n)]$ as follows: Let $g(n)=f(T(n))$. The input M works on is of the form $1^{g(i)}$ for some i, other inputs are rejected. M computes i first, then nondeterministically simulates N_i on $1^{g(i)}$ honestly. M accepts $1^{g(i)}$ iff N_i accepts $1^{g(i)}$ in less than $R(|x|)/C_i$ time, where C_i is the number of tape symbols in N_i. This simulation is possible by Lemma 1. Since for all k, $\frac{R(n)}{n^k} \rightarrow 0$, M accepts $1^{g(i)}$ iff N_i accepts $1^{g(i)}$ for large enough i.

By our definition it is clear that $L(M) \in NTIME[R(n)]$. And by the assumption $NTIME[R(n)]\,|_{1^{f(n)}} \subseteq PSPACE$ and the fact that $L(M) \subseteq \{1^{g(n)}\}$, we conclude that $L(M) \in PSPACE$. But then there is a fixed k such that $L(M) \in SPACE[n^k]$. So there is a machine $\hat{M}$ running in n^k space such that $L(M)=L(\hat{M})$. Notice that $\hat{M}$ is deterministic.

Now we do the diagonalization by using $\hat{M}$. Define the diagonal language $L=\{1^{g(m)} \mid N_m$ does not accept $1^{g(m)}$ in time $R[g(m)]/C_m\}$. Clearly L is not in NP because of the fact that $\frac{R(n)}{n^k} \rightarrow 0$ and that every machine is enumerated infinitely often. To show that $L \in PSPACE$, we design $\hat{M}$: for input $1^{g(m)}$, $\overline{M}$ simulates $\hat{M}$ on input $1^{g(m)}$, and $\overline{M}$ accepts iff $\hat{M}$ rejects. It is clear that $\overline{M}$ requires no more than n^{k+1} space, and $L = L(\overline{M})$.

Now define $L' = \{1^{T^{-1}(g(m))} \mid 1^{g(m)} \in L\}$. Since $L \in PSPACE-NP$, we have $L' \in SPACE[T(m)^c]-NTIME[T(m)^c]$ □ (Theorem 2.4)

Corollary: There is an oracle A under which,

$$\bigcup_{c>0} DTIME^A[T(n)^c] \neq \bigcup_{c>0} NTIME^A[T(n)^c] \neq \bigcup_{c>0} SPACE[T(n)^c],$$

for all time-constructible $T(n) > 2^n$ simultaneously.

Proof of Corollary: Construct oracle A

$NTIME^A[n^{logn}] \subseteq PSPACE^A$, and

$DTIME^A[n^{logn}] \subseteq NP^A$.

The construction of the oracle is standard, therefore we leave it to the interested reader. The key is to observe that $PSPACE$ computations can do an exhaustive search and NP computations can "guess". □ (Corollary)

2.5. The world of NTIME ∩ CoNTIME

In Section 2.1 the assumption $DTIME[n^{logn}] \subseteq NP$ was made. The assumption, as the following theorem shows, has a strong implication.

Theorem 2.5A: For any time-constructible function $R(n)$, $DTIME[R(n)] \subseteq NP$ implies $DTIME[R(n)] \subseteq NP \cap CoNP$.

Proof of Theorem 2.5*A*: Let $L \in DTIME[R(n)]$. Then $\bar{L} \in DTIME[R(n)]$. Therefore $\bar{L} \in NP$, and $L \in CoNP$. We conclude $DTIME[R(n)] \subseteq NP \cap CoNP$. □ (Theorem 2.5A)

A natural question to ask is whether assuming $NTIME[n^{\log n}] \cap CoNTIME[n^{\log n}] \subseteq NP$, we can prove

$$\bigcup_{c \geq 1} NTIME[T^c] \cap \bigcup_{c \geq 1} CoNTIME[T^c] \neq \bigcup_{c \geq 1} NTIME[T^c],$$

for time-constructible T.

This question is in turn closely related to the $NTIME \cap CoNTIME$ hierarchy since if there are very sparse sets in the intersection hierarchy, then the technique in the proof of Theorem 2.1B can be directly applied. Unfortunately, there has not been any nontrivial hierarchy obtained for the intersection classes. This question again relates to another classical question concerning the complete sets in $NTIME \cap CoNTIME$, as shown in the following theorem.

Theorem 2.5B: If there is a complete set (under linear time reduction) for $NTIME[t] \cap CoNTIME[t]$ then for $t'(n)$ satisfying $t(n)=o(t'(n))$, $NTIME[t] \cap CoNTIME[t]$-$NTIME[t'] \cap CoNTIME[t']$ contains arbitrarily sparse sets.

Proof of Theorem 2.5*B*: With the tool of the traditional diagonalization, this theorem is essentially trivial. If there is a complete set for

$NTIME[t] \cap CoNTIME[t]$, then there is a recursive representation of all languages in this class by complementary pairs [LLR, K1]. Then if $t(n) = o(t'(n))$, we can certainly do the diagonalization in time t' to obtain a language in $NTIME[t'] \cap CoNTIME[t']$, not in $NTIME[t] \cap CoNTIME[t]$. This is because we have the list of complementary pairs, and the nondeterministic simulation do not cost more time [BGW]. □ (Theorem 2.5B)

Remark: Notice that under the assumption that complete sets exist, the intersection hierarchy is tight. This is not the case for either deterministic time hierarchy or nondeterministic time hierarchy.

Concerning the complete sets in $NP \cap CoNP$, Sipser has constructed an oracle X such that $NP^X \cap CoNP^X$ does not have complete sets [S]. Here we present a much simpler and shorter construction for Sipser's oracle.

First we need some technical notation. For each language $L \in NP \cap CoNP$, we can use a ***complementary pair*** $(M, \overline{M})$ of nondeterministic polynomial-time machines to represent L, where $L(M) = L$ and $L(\overline{M}) = \overline{L}$. A class C is ***recursively presentable*** if there exists a r.e. set of machines $M_{i1}, M_{i2}, \cdots$, so that $C = \{L(M_{ij} \mid j \geq 1\}$. This is equivalent to say that some TM M enumerates all the languages in C.

Theorem 2.5C: (Sipser) There is a recursive oracle A such that $NP^A \cap CoNP^A$ does not have complete sets.

The following Lemma can be found in [LLR] and [K1].

Lemma: For any oracle A, if $NP^A \cap CoNP^A$ has complete sets then the sets in $NP^A \cap CoNP^A$ are recursively presentable by complementary pairs. $\Box$ (Lemma)

Proof of Theorem 2.5*C*: Let $M_1, M_2, \cdots$ be the list of all TM's. Let f be a polynomial time computable 1-1 onto mapping from $N\mathbf{x}N\mathbf{x}N$ to N, where $f(i,j,h) > i+j+h$ and N is the set of positive integers.

Construction of A:

Stage n:

Let $n = f(i,j,h)$. Assume M_i outputs j *th* pair of form $(M, \overline{M})$ at h *th* step, where M and $\overline{M}$ are of some standard polynomial time clocked form. If this is not true, skip this stage. Let p_{ij} denote the polynomial time clock. Fix smallest k such that no string of length $\geq k$ is in A or A' yet. Fix smallest $i_0 \geq k$ such that $i_0 = f(i,c,c)$ for some integer c. For $i \in \{k, k+1, ..., i_0 + p_{ij}(i_0)\}$ - $\{i_0\}$, add 1^i to A. Now simulate M and $\overline{M}$ on 1^{i_0}, relative to A.

(1) If M (resp. $\overline{M}$) accepts, find $0y$ $(1y)$ of length i_0 which is not queried on the accepting path of M $(\overline{M})$, and put it into A.

(2) If both M and $\overline{M}$ do not accept, then put 1^{i_0} into A' (which leaves $(M, \overline{M})$ a bad pair).

End of construction

Assertion: No M_i enumerates exactly the languages in $NP^A \cap CoNP^A$ by complementary polynomial machine pairs.

Proof: Suppose $\hat{M}_i$ does. Then some M_i does so with polynomial time clocked pairs. Define

$L_i = \{ 1^k \mid \text{for some } j,\ f(i,j,j)=k, \text{ and for some } y,\ |y|=k-1,\ 1y \in A \}$.

In the construction of A, case (2) can never happen to M_i since otherwise M_i would not enumerate only complementary pairs. But if only (1) can happen to M_i, our language L_i is in $NP^A \cap CoNP^A$ which is different from any language enumerated by M_i by the diagonal process of (1). This contradiction completes the proof. □ (Assertion)

By above Lemma, the proof is complete. □ (Theorem 2.5C)

CHAPTER 3

Lower Bounds by Kolmogorov-complexity

3.1. Introduction

Obtaining good lower bounds has been one of the most important issues in theoretical computer science. In this chapter we investigate the application of Kolmogorov-complexity (K-complexity) in proving lower bounds for computation. Several classical open questions are answered by means of K-complexity.

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. In this chapter, however, the beauty of simplicity and intuition is brought back in the lower bound proofs via a recently discovered tool, the K-complexity. We will demonstrate how K-complexity can be applied through our solutions to several old open problems. New techniques are introduced along with these proofs as well.

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]).

Informally, a string is random if it cannot be computed from a shorter string. More precisely, a finite string X is random if, the smallest TM that starts on empty input tape and prints the string X, is of size equal to or greater than $|X|$.

The main idea in the lower bound proofs using random strings is as follows. A random string is chosen to construct the input. Then if a machine does not run for a long enough time on this (particular!) input, we will find a way, from the information that 'carried away' from this string, to reconstruct our random string X. This gives us a short program to print X, implying that X is not random. The advantage of using K-complexity is that we can put our hand on a hard but not provably hard (Fact 3) input.

Formally,

Definition 3.1A: The *K-complexity* of a finite string X, written as $K(X)$, is the size of the smallest TM which, starting from empty input tape, prints X.

Remark: The *K-complexity* of a finite string X can also be defined to be the size of the smallest *nondeterministic* TM that *accepts* only X. It is easy to see that this definition is equivalent to Definition 3.1A up to a constant factor in machine size. In the application, we will take whichever convenient to use.

Definition 3.1B: The *K-complexity* of X, relative to string Y, written as $K(X \mid Y)$, is the length of smallest TM (deterministic or nondeterministic), with Y as its extra information, that *accepts* (or *prints*) only X.

Definition 3.1C: 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*'.

In Section 3.2.4, we first consider deterministic on-line computations. By *on-line* computation we mean the input tape can be read only from left to right (1-way input tape). A classical question, probably among the oldest in complexity theory, was how efficiently a one tape on-line TM could simulate a k tape on-line TM. In other words, whether more tapes were better than one, and how much better. The first result in this direction was obtained by Hartmanis and Stearns [HS] who showed that one tape TM can simulate k tape TM in quadratic time. This result is also true for nondeterministic machines. Then in 1963, Rabin [R] proved that two tapes are better than one tape for real time computations. In 1982 Paul established the first nonlinear lower bound: it requires $\Omega(n(logn)^{1/2})$ time to simulate two tapes by one tape for on-line computations [P2]. In 1983, Duris, Galil, Paul, and Reischuk [DGPR] obtained an $\Omega(nlogn)$ lower bound. We now settle this open problem by proving an $\Omega(n^2)$ tight lower bound. This lower bound meets the upper bound provided in [HLS]. This problem was independently solved by Maass [M], the author [L], and Vitanyi[V].

In Section 3.2.5, the nondeterministic case for a 1 tape on-line TM simulating a 2 stack on-line TM is considered. The best upper bound is again $O(n^2)$ by [HS]. The best lower bound was obtained in [DGPR]: $\Omega(nlogn)$. We provide an $\Omega(n^{1.5}/logn)$ lower bound for one tape nondeterministic on-line TM simulating two pushdown store on-line TM. This greatly improves the $\Omega(nlogn)$ lower bound of [DGPR]. Notice that this lower bound is not implied by the lower bounds in Section 3.2.6.

For the nondeterministic case of 1 tape versus 2 tapes, Maass [M] obtained an $\Omega(n^2/(logn)^2loglogn)$ lower bound. In Section 3.2.6, we improve this lower bound to $\Omega(n^2/lognloglogn)$ and prove that the tight $\Omega(n^2)$ lower bound cannot be obtained by the language used in [M]. We present a upper bound on an interesting combinatorial problem defined in [M], and we show that the actual language use in [M] and a language defined seven years ago in [F] can be accepted in $O(n^2loglogn/\sqrt{logn})$ time by a 1 tape on-line machine.

In Section 3.3.3, K-complexity is applied to give a lower bound on string-matching. The string-matching problem is defined 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. String-matching is a very important practical problem. It is extensively used, for example, in text-editing. Linear time algorithms for string-matching were developed by Boyer and Moore[BM], Cook [C2], and Knuth, Morris, and Pratt [KMP]. In 1981, Galil and Seiferas [GS] proved that 6 head two-way DFA (deterministic finite automata with 6 heads each can move to either direction independently) can do string-matching in linear time. They raised the question whether some k-head 1-way DFA (the k heads can only go from left to right), written as k-DFA, can do string-matching. We develop a set of new methods, settling the cases of $k=2$ and $k=3$. We show that no 3-DFA can accept the language $L=\{\#x\$y\cent \mid x \text{ is a substring of } y\}$. By one of the Lemmas developed here, we are also able to present a simple proof of 'k heads are better than $k-1$ heads for DFAs', first proved by Yao and Rivest [YR].

In Section 3.3.4, we prove that string-matching requires almost quadratic time for a k-head 2-way DFA with $k-1$ heads being able to see only endmarkers.

Our last lower bound by K-complexity, obtained in Section 3.3.5, shows that *checking* is easier than *generating* [W]. In [L2], we proved that one tape on-line nondeterministic TM equipped with a random number generator can simulate a two tape machine in less than quadratic time with any small error probability ϵ. The proof depends on the fact that on a single tape the string $\#x\#x\#$ can be accepted very fast, in $|x|log|x|$ time, probabilistically [F]. The nondeterminism above may be removed if we can show that with $\#x\#$ on a single tape, the string $\#x\#x\#$ can be generated faster than quadratic time probabilistically with error probability $\epsilon<1/2$. Unfortunately, we prove this is not possible: on a single tape, probabilistically moving a string (requiring quadratic time) is harder than probabilistically checking the equivalence of two strings. It is interesting to note that this result is true for neither deterministic nor nondeterministic machines.

As a by-product of K-complexity, all the lower bounds we prove are not only for the worst cases, but also for the average cases as well, because there are many random strings.

3.2. On one tape versus two

3.2.1. Summary

We obtain the following results for on-line computations,

(1) it requires $\Omega(n^2)$ time to simulate a 2 pushdown store machine deterministically by a 1 tape machine, this is settles an old open problem listed in [DGPR];

(2) it requires $\Omega(n^{1.5}/logn)$ time to nondeterministically simulate a 2 pushdown store machine by a 1 tape machine, this greatly improves an $\Omega(nlogn)$ lower bound of [DGPR];

(3) it requires $\Omega(n^2/lognloglogn)$ time to nondeterministically simulate a 2 tape machine by a 1 tape machine, this is a minor improvement of an $\Omega(n^2/log^2nloglogn)$ lower bound contained in [M]; and

(4) it is shown that the $\Omega(n^2)$ lower bound for the nondeterministic case cannot be obtained by the language (which is shown to be acceptable by a 1 tape machine in $O(n^2loglogn/\sqrt{logn})$ time) used in [M] or [F].

3.2.2. Background

In this Section we will 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-stack machine is like a usual pushdown automaton, but has k pushdown stores.

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 $\Omega(nloglogn)$ time. (They later improved this to $logn$.) The following open questions (open problems 1 and 7) were listed in [DGPR]: whether the gaps between the Hartmanis-Stearns [HS] $O(n^2)$ upper bound and the $\Omega(n(logn)^{1/2})$ lower bound in deterministic case and the $\Omega(nloglogn)$ lower bound in nondeterministic case for 1 tape simulating k

tapes (stacks) 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.

The first result was obtained independently by Maass [M], the author [L], and Vitanyi [V2]: it requires $\Omega(n^2)$ time to deterministically simulate 2 tapes (pushdown stores) with 1 tape. This provides the tight lower bound, settling the long-standing open question.

We also show that it requires $\Omega(n^{1.5}/logn)$ time to nondeterministically simulate 2 pushdown stores with 1 tape. This greatly improves the $nlogn$ bound of [DGPR]. This is the best result so far obtained for 1 tape versus 2 stacks in the nondeterministic case, and it is not implied by the results of [M] or next section.

For the case of 1 nondeterministic tape versus 2 tapes, a stronger result (but this does not imply the above result) is due to Maass' [M] who obtained an $\Omega(n^2/(logn)^2loglogn)$ lower bound via an ingenious language and a very nice combinatorial lemma. In Section 3.2.6 we first improve the Maass' lower bound to $n^2/lognloglogn$. Unlike Sections 3.2.4 and 3.2.5, this result is based on on Maass' proof. Surprisingly, we next show 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 less than quadratic time. Therefore the tight lower bound needs a new language.

Remark 3.2.2: As a corollary of the first result we can also obtain the following separation result, which shows that there is a language L accepted in linear time by a 1-tape nondeterministic on-line machine but requiring $\Omega(n^2)$ time for its deterministic version. It is well-known the general task for separating the deterministic time from nondeterministic time is very hard. One recent well known result by [PPST] states: $NTIME[n] \neq DTIME[n]$. Maass and Schorr [MS], and Gill [G] considered the class of TM's with a single tape which serves for both input and working space. For such TM's, Gill proved that there is a language L which can be accepted in nondeterministic $O(nlogn)$ time but not in deterministic $o(n^2)$ time. Maass and Schorr, using the idea of alternation, proved there is a language L which is in nondeterministic linear time but not in deterministic $o(n^{1.22})$ time. Notice that our 1-tape real time model is not as weak as one might think: it can simulate k counters for any k [V], and it can accept the palindrome language which requires $\Omega(n^2)$ time for above single tape machine even with the power of nondeterminism [H1, HU].

3.2.3. Jamming Lemma

In this section we give some more notation and present an important lemma that is needed for the later proofs.

Throughout this chapter variables X, Y, x_i, y_i ... denote strings in Σ^* for $\Sigma = \{0,1\}$.

For the rest of this section we will consider a 1-tape on-line machine M. We will call M's input tape head h_1, and its work tape head h_2.

Definition 3.2.3: 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 3.2.3*A*: 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, the total space ($|c.s.|$) we need is less than $d\,|M|+ logk_1+ ...+ logk_d$, with $\sum_{i=1}^{d} k_i = n$, which is less than $d\,|M|+ dlogn/d$ by a standard calculation (i.e. maximize the function).

*Remark*3.2.3*B*: 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 3.2.3A), 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 $x_1x_2 \cdots x_k\#\ldots$, 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 3.2.3C: 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 input $\{x_1,...,x_k\}-\{x_{i_1},...,x_{i_l}\}$ 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 3.2.3*D*: If M is nondeterministic, the above Jamming Lemma is still true except that we 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.

3.2.4. Deterministic case: $\Omega(n^2)$ tight lower bound for 1 tape simulating 2

Define language

$$L=\{x_1\$x_2\$ \cdots \$x_k\#y_1\$ \cdots \$y_l\#(1^{i_1},1^{j_1})(1^{i_2},1^{j_2})...(1^{i_s},1^{j_s}) \mid x_p=y_q$$

$$\text{where } p=i_1+...+i_t \text{ , } q=j_1+...+j_t \text{ for all } t=1,...,s\}.$$

Theorem 3.2.4: It requires $\Omega(n^2)$ time to deterministically simulate 2 pushdown stores by 1 tape.

The theorem will follow from Lemma 3.2.4 below. We shall be generous with the constants to make the proof easier.

Lemma 3.2.4. Any deterministic one-tape on-line TM accepting L requires $\Omega(n^2)$ time.

Proof of Lemma 3.2.4: Suppose instead that a 1-tape on-line deterministic machine M accepts L in $o(n^2)$ time. We will derive a contradiction by showing that a random string has a short program.

Let us assume that each tape square of M's work tape can be encoded by C_0 binary bits. We fix large n and C such that $C > 100C_0|M|$, $\frac{2n\log C}{C} < \frac{n}{48}$, $\frac{(\log C)n}{C^2} < \frac{n}{8C}$, and all formulas to follow are meaningful.

We first fix a string X of length n such that $K(X) \geq |X|$. Then break X into $k = n/C$ parts, each C long, and call them $x_1, x_2, \cdots, x_k$. Consider the partial input beginning $x_1\$x_2\$ \cdots \$x_k\#$ ... to M. If more than $k/2$ x_i's are mapped *onto* (See Definition 3.2.3) regions of sizes at least n/C^3, then M requires $O(n^2)$ time, a contradiction. Therefore we can assume that half x_i's are mapped *into* (See Definition 3.2.3) regions of sizes less than n/C^3 on the work tape. We order these x_i's by the left boundaries of the regions *onto* which they are mapped. Let x_c be the median.

The following is the idea of the proof. We consider two cases. In the first case we assume that many x_i's are jammed (mapped) into a small region R; that is, when h_1 (the input tape head) is reading them, h_2 (the work tape head) is always in this small region R. We shall then show that X is not random by the Jamming Lemma. In the second case, we assume there is no jamming region, and that the x_i's are spread evenly on the work tape. In that case, we will arrange the y_j's so that there will be many pairs (x_i, y_j)'s so that $x_i = y_j$ and x_i and y_j are mapped into regions that are far apart. For each of these pairs we will arrange the indices to make M match x_i against y_j.

Case 1 (jammed): Assume there exist k/C x_i's and a fixed region R of length n/C^2 on the work tape such that M maps these x_i's into R.

We will show that a short program can be constructed which accepts only X. Consider the two regions of length $|R|$ to the left and to the right of R on the work tape. Call them R_l and R_r, respectively. We find positions p_l in R_l and p_r in R_r with shortest c.s.'s in their respective regions. They both must be shorter than n/C^3, for if the shortest c.s. in either region is longer than n/C^3 then M uses O(n^2) time. Let region R_l' (R_r') be the portion of R_l (R_r) right (left) to p_l (p_r).

Now, with the information of

(1) less than $k - \frac{k}{C}$ x_i's that are not mapped into $R_l' R R_r'$,

(2) the two c.s.'s, and

(3) the region $R_l' R R_r'$,

we can construct a short program to check if a string Y is X by running M as follows. Check if $|Y|=|X|$. By Jamming Lemma construct the content of M's work tape at the time h_1 gets to first # sign; divide Y into k equal pieces and form $y_1\$ \cdots \$y_k$. Consider M with input ending simulating M from the time when h_1 reads first # sign. If M accepts, then we know $Y=X$.

To show above program is short we have to represent the c.s.'s and $k-(k/C)$ x_i's in a small space. For the c.s.'s, by Remark 3.2.3A, we can use $(n/C^3)(|M|+logC^3)$ space to write them (because h_1 is 1-way). For the x_i's, their contents and their relative positions are specified by the method of Remark 3.2.3B in less than $2n-(n/C)+(n/4C)$ space.

Now the length of this program is only the sum of:

(1) $n-(n/C)+(n/4C)$ for $k-(k/C)$ x_i's,

(2) $3n/C^2$ for 3 regions R_l', R, and R_r',

(3) $(n/C^3)(|M|+3logC)$ for the 2 c.s.'s, and

(4) $O(logn)$ for some counters.

The total is less than n for large n. Thus $K(X)<|X|$, a contradiction.

Case 2 (not jammed): Assume that for each fixed region R, with $|R|=n/C^2$, there are at most k/C x_i's mapped into R.

Fix a region of length n/C^2 that contains x_c; call it R_c. By a simple counting argument, we can see that there must be a set S_r containing $k/6$ x_i's, say $S_r=\{x_{i_1}, \cdots ,x_{i_{k/6}}\}$, completely mapped to the right of R_c, and a set

$S_l=\{x_{j_1}, \cdots, x_{j_{k/6}}\}$ completely mapped to the left of R_c. Without loss of generality, assume $i_1<i_2<\ldots<i_{k/6}$, and $j_1<j_2<\ldots<j_{k/6}$.

We now arrange the y_i's. Let y_1 be x_{i_1}, y_2 be x_{j_1}, y_3 be x_{i_2}, y_4 be x_{j_2}, and so forth. In general,

(*) $y_{2m}=x_{j_m}$ and $y_{2m-1}=x_{i_m}$, for $m=1,2, .., k/3$.

Our partial input to M is $x_1\$ \cdots \$x_k\#y_1\$ \cdots \$y_{k/3}\$$. If there do not exist $k/12$ pairs $y_{2i-1}\$y_{2i}\$$ such that while h_1 (the input head) reads them, h_2 (the work head) travels a distance less than $n/4C^2$, then M uses $O(n^2)$ time, a contradiction. Otherwise if we have $k/12$ such pairs, then by our ordering of the y_j's and a simple counting argument, we see that there will be a region R ($\subset R_c$) of length $n/4C^2$ and $k/24$ x_i's all from either S_r or S_l that are mapped into one side of R and whose corresponding y_j's (x_i *corresponds* to y_j if $x_i=y_j$) are mapped into the other side of R. Let the set of these x_i's be S_x, and the set of corresponding y_j's be S_y. We know that when h_1 reads anything in S_x, h_2 is at one side of R, and when h_1 reads anything in S_y, h_2 is at the other side of R. $|S_x|=|S_y|=k/24$. Let the indices in S_x be $a_1<a_2<\ldots<a_{k/24}$, and the indices in S_y be $b_1<b_2<\ldots<b_{k/24}$. By our previous arrangement we know $x_{a_i}=y_{b_i}$. Now we complete our input to M by appending

$$(**) \quad \#(1^{a_1},1^{b_1})(1^{a_2-a_1},1^{b_2-b_1}) \cdots (1^{a_{k/24}-\sum_{l=1}^{k/24-1} a_l},1^{b_{k/24}-\sum_{l=1}^{k/24-1} b_l}).$$

After M finishes its computation on (**), we find a position p in R which has the shortest c.s. Let $C_1 = n/(l\,|S_x|)$. If this c.s. is longer than n/C_1C^3, then M uses time $O(n^2)$, a contradiction. If it is less than n/C_1C^3, then again we construct a short program P to accept X by a 'cut and paste' argument. We shall first show how the program P is constructed, and then show that its length is short. Without loss of generality, assume that S_x is mapped into the left-hand side of R and S_y is mapped into the right. For input Y, program P first partitions Y into $y_1\$ \cdots \$y_k$ and compares relative parts with $\{x_1, \ldots, x_k\} - S_x$. To compare the x_i's in S_x, simulate M as follows. Put the elements of $\{x_1,\ldots,x_k\} - S_x$ into their correct places on the input tape, filling the places for x_i's in S_x arbitrarily. Arrange the y_l's as in (*), *i.e.*, y_{j_m} at $2m$th place and y_{i_m} at $2m$–1st place, for $m = 1, \cdots, k/3$. Therefore all the y_i's in $S_y (= S_x)$ (the parts not compared) will be put into the input. Adding string (**) above completes the input of M. Now using the c.s. at point p we run M such that h_2 stays in right-hand side (S_y's side) and never passes the point p. Whenever h_2 meets p, we check if the current ID matches the corresponding one in c.s. and if it does we then use the next ID in c.s. to continue. If in the course of the simulation process, M rejects or there is a mismatch (that is, when h_2 gets to p, M is not in the same state or h_1 position as is indicated in the c.s.), then $Y \neq X$. Otherwise $Y = X$. Notice that it is possible for M to accept (or reject) on the left-hand side (S_x's side) of point p, in this case if all IDs match our accepting c.s. then we know M accepts, since once h_2 passes point p it does not read anything in S_y and all other y_i's are 'good' ones (we have already checked them).

The length of the program P is the sum of:

(1) less than $n-(C|S_x|)+(C|S_x|)/4$ for the $n-|S_x|$ x_i's not in S_x, using the representation of Remark 3.2.3B;

(2) $(n/C_1C^3)(|M|+logC_1C^3)$ for the n/C_1C^3 long c.s. by Remark 3.2.3A;

(3) $(2n/C)logC$ (by a simple calculation), which is $<C|S_x|/4$ by the definition of C, for indices of the y_l's, because the odd and even y_i's can be specified by relative locations, respectively, for we ordered the x_{i_l}'s and x_{j_l}'s;

(4) $|S_x|log(n/|S_x|)<C|S_x|/8$ for the indices in (1); and

(5) several counters of size O($logn$);

The total is less than n for large n's. This again implies $K(X)<|X|$, a contradiction.

Case 1 and Case 2 complete the proof of Lemma 3.2.4. □ (Lemma 3.2.4)

Proof of Theorem 3.2.4: Obviously, L can be accepted in linear time by a 2 tape machine. To accept L in real time by a deterministic 2-stack machine, we can define a new padded language L_p from L,

$$L_p=\{x_k\$...\$x_1\#y_k\$...\$y_1\#(1^{i_1},1^{j_1},1^k)(1^{i_2},1^{j_2},1^k)...(1^{i_s},1^{j_s},1^k)\,|$$

For each triple (i_n,j_n,k) there is a pair (i_0,j_0) so that

$$\sum_{i=1}^{i_0-1}|x_i|=\sum_{t=1}^{n}i_t+(n-1)k$$

$$\sum_{j=1}^{j_0-1} |y_j| = \sum_{t=1}^{n} j_t + (n-1)k,\ x_{i_0} = y_{j_0}, \text{and } |x_{i_0}| = k\}$$

L_p can be accepted in real time by a deterministic 2-stack machine in an obvious way. We observe that the same lower bound proof (Lemma 3.2.4) for L works as well for L_p. Therefore the theorem is proved. □ (Theorem 3.2.4)

3.2.5. Nondeterministic case A: 1 tape versus 2 stacks

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

The theorem will follow from Lemma 3.2.5. We shall concentrate on explaining the ideas of the proof; the details can be found in the proof of Lemma 3.2.4.

Lemma 3.2.5: It requires $\Omega(n^{1.5}/logn)$ time to accept L' by any 1-tape nondeterministic on-line machine.

Proof of Lemma 3.2.5: Suppose a nondeterministic 1-tape M accepts L' in time $o(n^{1.5}/logn)$.

As in Lemma 3.2.4 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 $k = n^{1/2}/Clogn$. 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 $O(n^{1.5}/logn)$, 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 region 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 $\leq n/C^3$. Then as before we consider the two regions R_l and R_r of length $|R|$ neighboring to R. 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 $n^{0.5}/C^4logn$ then M runs in time $O(n^{1.5}/logn)$. If they are both shorter than $n^{0.5}/C^4logn$ 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. We then find X, as in case 1 of Lemma 3.2.4, by a short program, 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. We know $|R_0|\geq n/C^2$. 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 can reconstruct x_0 by the routine (cut&paste) method as in Lemma 3.2.4. We simulate M with the help of this short c.s., and the other x_i's ($i>0$). This contradicts the randomness of X. □ (Lemma 3.2.5)

Proof of Theorem 3.2.5: 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 #. 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 3.2.5, the proof is complete. □ (Theorem 3.2.5)

3.2.6. Nondeterministic case B: 1 tape versus 2 tapes

The first result for nondeterministic machines was obtained by [DG] who showed 2 tapes are better than 1. [DGPR] obtained the $\Omega(nlogn)$ lower bound.

Maass [M] obtained an almost tight $\Omega(\frac{n^2}{(logn)^2 loglogn})$ lower bound for one tape simulating two tapes for nondeterministic machines. Recently, the author was surprised by a paper 7 years ago by Freivalds [F] who claimed a result (Theorem 2 in [F], without proof) which practically implies a tight $\Omega(n^2)$ lower bound for our problem, although in a different context. 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 $*b_i{}^j*$ and they are of a fixed length.

The language, B, constructed in [F] is similar (but less complete) if we let $c(i)=a(i)b(i)$ and replace 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 $y \in \overline{B'}$.

In this section we will

(1) improve the [M]'s lower bound to $\Omega(\frac{n^2}{lognloglogn})$; and

(2) show that from neither the method of [M] nor the language of [F] can we obtain the tight $\Omega(n^2)$ lower bound. We will actually show that $\hat{L}$, L^*, and B can be accepted in $O(\frac{n^2 loglogn}{\sqrt{logn}})$ time by a 1 tape nondeterministic (deterministic for L^* and B) machine. An $n/\sqrt{logn}$ upper bound on Theorem 3.1 in [M] is obtained. This result suggests that a much tighter lower bound must be obtained by some other languages.

The results contained in this section represent a joint work of the author and Z. Zhang [LZ].

Unlike the results in previous 2 sections (which were obtained in parallel to those of [M] and [V2]), Theorem 3.2.6A, which improves the best lower bound of [M], is based on [M]'s approach. We assume the reader is familiar with [M].

Theorem 3.2.6A: It requires $\Omega(n^2/lognloglogn)$ time to nondeterministically simulate 2 tapes by 1 tape for on-line computation.

We will show that the language L^* (and $\hat{L}$) requires $\Omega(n^2/lognloglogn)$ time. 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 3.2.6A: (Maass) 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 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 will be given later in this section.

Notice that any sequence S in L^* satisfies the requirements in Lemma 3.2.6A. 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* 'deserts' on the work tape.

Lemma 3.2.6B: ('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 3.2.6A;

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

Proof of Lemma 3.2.6*B*: 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 $D_1, D_2, \ldots, D_{logn/C}$ for some constant C in the 'middle' of work tape such that (1), (2), and (3) above are satisfied. □ (Lemma 3.2.6B)

To prove Theorem 3.2.6A, we apply the proof of [M] for each desert D_i in Lemma 3.2.6B. Instead of using Theorem 3.1 of [M] we use Lemma 3.2.6A 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 3.2.6A 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 3.2.6A)

Theorem 3.2.6B: $\hat{L}$ (L^* and B) can be accepted in $O(\frac{n^2loglogn}{\sqrt{logn}})$ time by a 1-tape *nondeterministic* on-line machine.

Lemma 3.2.6C: 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) $s_1 = 2 * s_2 (mod k)$ or $s_1 = 2 * s_2 + 1 (mod k)$, 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) $|N| = O(k / \sqrt{log k})$, where $N = \{s_1 \in S_1 \mid s_1 R s_2 \text{ for some } s_2 \in S_2\}$. ($N$ is the set of elements in S_1 that are 'neighbors' of some elements in S_2.)

Remark: This gives an upper bound on Theorem 3.1 of [M] and Lemma 3.2.6A.

Proof of Lemma 3.2.6C: Let $bin(i)$ denote the binary representation of integer i, and $\#bin(i)$ denote the number of 1's in $bin(i)$. Without loss of generality, let l be odd. Define

$S_1 = \{s \mid \#bin(s) < l/2\}$, and

$S_2 = \{s \mid \#bin(s) > l/2\}$.

(Note: If l even, put half of those s's with $\#bin(s) = l/2$ into S_1 and the other half to S_2. This will not hurt the calculations below.) Obviously, $|S_1| = |S_2|$ and $S_1 \cap S_2 = \phi$. We now calculate $|N|$ in the following. Define

$N_1 = \{s_1 \in S_1 \mid$ for some $s_2 \in S_2$, s_1 and s_2 satisfy condition (1) in the lemma$\}$,

$N_2=\{s_1\in S_1 \mid \text{for some } s_2\in S_2,\ s_1+1=s_2\}$, and

$N_3=\{s_1\in S_1 \mid \text{for some } s_2\in S_2,\ s_2+1=s_1\}$.

It is clear that $|N|\le 2|N_1|+|N_2|+|N_3|$. Since rule (1) in the lemma can at most change 1 in $\#bin(i)$ for any i, N_1 contains only those s_1's in S_1 such that $\#bin(s_1)=l/2$. Therefore

$$|N_1|\le\binom{l/2}{l}=O\left(\frac{\sqrt{2\pi l}\,e^{-l}l^l}{\pi l e^{-l}(l/2)^l}\right)=O(2^l/\sqrt{l}).$$

Similarly, since $\#bin(s+1)\le\#bin(s)+1$, $|N_2|=O(2^l/\sqrt{l})$.

Consider N_3. For $s_2\in S_2$, if $\#bin(s_2)=\frac{l}{2}+i$ and $s_2+1\in S_1$, then the last $i+1$ bits of bin(s_2) must be 1's. Fix i, we have at most $\sum_{i=1}^{l/2}\binom{(l/2)-1}{l-i-1}$ such elements. So,

$$\begin{aligned}|N_3|&\le\sum_{i=1}^{l/2}\binom{(l/2)-1}{l-i-1},\\ &=\sum_{i=1}^{l/2}\binom{(l/2)-i}{l-i-1}\\ &\le\sum_{i=1}^{i/2}\binom{(l/2)-\lceil i/2\rceil}{l-i+1}\\ &=O\Big(\sum_{i=1}^{l/2}\frac{2^{l-i+1}}{\sqrt{l}}\Big)\\ &=O(2^l/\sqrt{l}).\end{aligned}$$

This concludes $|N|=O(k/\sqrt{logk})$. □ (Lemma 3.2.6C)

Remark: Assertion (c) in Lemma 3.2.6C is true for any partition as long as for all $s_1 \in S_1$ and $s_2 \in S_2$ we have $\#bin(s_1) \leq \#bin(s_2)$. It is this property, which will be simply referred as Lemma 3.2.6C, we actually use in the proof of Theorem 3.2.6B.

Proof of Theorem 3.2.6*B*: We design a 1-tape on-line nondeterministic machine to accept $\hat{L}$. According to (the remark of) Lemma 3.2.6C, we will distribute each sequence $\{b_i^j\}$ on a separate track of the work tape by the following rules:

(1) if $\#bin(b_p^j) < \#bin(b_q^j)$, then b_p^j is put on the left of b_q^j;

(2) if $\#bin(b_p^j) = \#bin(b_q^j)$, and $b_p^j < b_q^j$ then b_p^j is put on the left of b_q^j.

There is one more problem to be solved: M has to keep track which block it is reading and thereby decides where to put the block. It may 'drag' a counter under the work head, but this results an intolerable running time $O(\frac{n^2 logn}{\sqrt{logn}})$. The techniques for 'dragging' a counter that do not cause the $logn$ delay developed in [P3] and [F1] cannot be used here, since only one work tape is available.

Surprisingly, the major part of the counter never need to be moved at all. We divide the work tape into four tracks: track i, i=1,2,3,4. Each track is further divided into 4 subtracks: subtrack i, i=1,2,3,4. For each track, we will keep a static counter $C0$ (of length $logn$) which never moves, and we keep two other counters $C1$ and $C2$ of length $loglogn$ moving with the work tape head.

$C1$ holds the most recent value of the last *loglogn* bits of $C0$. $C2$ holds the number of 1's in the $C0$ (last *loglogn* bits are taken from $C1$). $C0$ is modified every time $C1$ overflows. The algorithm of M is outlined as below.

(1) M lays the following frame on each track ($C0=C1=C2=0$):

Subtrack 1: $\# \; 1 \; \# \; space_1 \; \# \; 2 \; \# \; space_2 \; \# \cdots \# \; logn \; \# \; space_{logn} \; \#$,

Subtrack 2: $C0$ (size *logn*),

Subtrack 3: $C1$ (size *loglogn*),

Subtrack 4: $C2$ (size *loglogn*).

The segment $\# \; m \; \# \; space_m \; \#$ on track j is explained as '$space_m$ will hold all blocks $b_i{}^j$'s such that $\#bin(i)=m$.' The size of each $space_m$ can be guessed. The size of each $\# \; m \; \#$ is *loglogn* where n can be guessed also.

(2) Sequence $\{b_i{}^j\}$ will be put in track j, where j can be found out by the finite control of M. At the time M is about to read $b_i{}^j$, it adds 1 to $C1$ in track j. If $C1$ overflows, M goes back to change $C0$ in track j, and then bring back a number, of length at most *loglogn*, which is the number of 1's decreased or increased in $C0$. (We do not bring along $C1$ and $C2$ with the work head in this process.) Then modify $C2$ in track j accordingly.

(3) Locate the work head to $\# \; l \; \#$ where l is equal to $C2$. Copy $b_i{}^j$ (whose length can be guessed) on to the first available place in $\# \; space_l \; \#$ in subtrack 1 of track j. In this process $C1$ and $C2$ are moved along with the work head.

(4) After the whole input is read, check if $b_i^1 = b_i^2 = b_i^3 = b_i^4$, for all i, in the obviously way.

The time complexity of this algorithm is analyzed as follows. Step (1) takes no more than $O(nlogn)$ time. In step (2), the overflow can happen at most $O(n/logn)$ times. Each time the overflow happens, modifying $C0$ and $C2$ requires at most $O(nloglogn)$ time. Thus step (2) takes $O(\frac{n^2loglogn}{logn})$ time. By Lemma 3.2.6C, step (3) takes $O(\frac{n^2loglogn}{\sqrt{logn}})$ time. Step (4) takes only $O(n)$ time. Thus the total time is $O(\frac{n^2loglogn}{\sqrt{logn}})$. The proof is therefore complete. □ (Theorem 3.2.6B)

Corollary: Language L^* (Note: B can be accepted even faster.) can be accepted by a 1 tape deterministic on-line machine in $O(n^2loglogn/\sqrt{logn})$ time. □ (Corollary)

3.3. Lower bounds on string-matching

3.3.1. Summary

A set of new techniques is developed to cope with an open question, due to Galil and Seiferas [GS], as to whether a k-head one-way DFA can perform string-matching. We answer the question for the case $k=3$ by showing that no three-head one-way DFA accepts the language $L=\{\#x\$y\cent|x$ is a substring of $y\}$. This improves the results obtained in [LY] and [L1].

Two other related lower bounds are also obtained. The first shows that two-way k-head DFA with $k-1$ heads being able to see only endmarkers requires $n^2/logn$ time to do string-matching. The second shows that probabilistic moving a string is harder than probabilistic matching two strings on one Turing machine tape.

3.3.2. Background

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. It is well known that the string-matching problem is very important in practice.

Since the publication of linear 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 noticed that a multi-head *one-way* deterministic finite automaton must operate in linear time. Motivated by this observation 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 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.

In this chapter we develop new techniques and settle the case of $k=3$ by showing that 3-DFA cannot do string-matching. We hope that the methods used here combined with that of [LY] will help in providing useful techniques for the general problem.

In addition, in Section 3.3.4 we study string-matching by 2-way k-head DFA with $k-1$ heads blind, and in Section 3.3.5, we study probabilistic matching and moving strings on one Turing machine tape.

A k-DFA M has a finite set Q of states, a subset A of Q of accepting states, k heads $h_1, h_2, \cdots, h_k$ (which may see each other), a transition function, and a one-way read-only input tape. We assume that the standard input to M is $\#pattern\$text\not{c}$, where $pattern, text \in \Sigma^*$ for the alphabet $\Sigma=\{0,1\}$. $\#$ and $\not{c}$ are the left and right endmarkers, respectively. Initially, all k heads are at the left-most tape position. At each step, depending on the current state and the ordered k-tuple of symbols seen by the heads $h_1, h_2, \cdots, h_k$, M changes state and moves some of the heads one position to the right. If no head is moved during $|Q|+1$ consecutive steps, we know that no head will ever move again. We can modify M so that whenever this situation occurs, all heads move to the right until they reach the right end-marker. Hence, we assume that on each input all heads will eventually reach the right end-marker, and M halts when this happens.

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. Clearly $1 \le i_j \le |I|$ for $1 \le j \le k$. The *initial ID* of M on input I is $(I,q_*,1,1,\cdots,1)$. The *computation* of M on I is the sequence of ID's reached by M on input I, starting from the initial ID. A *partial ID* of M on input I at step t is $(I',q,i_1,\cdots,i_k)$, where I' is the suffix of I which has not yet been read by some head(s) of M.

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. By k-NFA we shall denote the nondeterministic version of a k-DFA.

Let x and y be binary strings. We write $x=y$ if they are the same binary string. xy denotes the concatenation of x and y. $|x|$ denotes the length of x. Superscripts will be used to denote different occurrences of the same string. Subscripts will be used to denote different binary strings.

In the following, the word 'information' only means some binary strings. The 'amount of information' is the total length of the strings.

3.3.3. Three 1-way heads cannot do string-matching

Some more technical definitions and conventions are needed. 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 3.3.3.

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 $1^k X 1^{k'}$ where $Y=kXk'$. In the following, we always consider input of form $\# 1^k X 1^{k'} \$ text ¢$. 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 $s_1 s_2 \cdots s_6$ or $x_{uv...w1} x_{uv...w2} \dots x_{uv...w6}$, 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 3.3.3A The *text* in the input $\# 1^k X 1^{k'} \$ text ¢$ is *regular* if it is concatenated, no more than 1000 times, from the following blocks:

(1) $1^l 0 1^l 0 \cdots 1^l 0$, 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 3.3.3A)

Proposition 3.3.3: 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 3.3.3)

Proposition 3.3.3 is exactly the objective of Definition 3.3.3A. The proof which we have omitted here follows directly from this definition.

Definition 3.3.3B: 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 3.3.3A that appears in the *text*, $K(X'-x\,|X-x) \leq 50 log |X|$. □ (Definition 3.3.3B)

Definition 3.3.3C: 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 3.3.3C)

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 $a_i=1^mX1^m$ 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. We will try to construct 'bad' (but *easy*) inputs to fool the head out of *pattern*. With the goal of 'constructing an *easier text* than the *pattern*' in mind, we start our proof. Drawing some pictures while following the proof would facilitate its understanding.

Theorem 3.3.3: No 3-DFA accepts $L=\{\#a_0\$text\cent \mid a_0$ is a substring of *text*$\}$.

Proof of Theorem 3.3.3: 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, thus obtaining 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 $x_1x_2x_3x_4x_5x_6$ of equal length as assumed above (and so is each sub-string). We consider only inputs of form $\#1^kX1^{k'}\$text\cent$ to M. That is, $a_0=1^kX1^{k'}$. 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 $X1^m$. To understand it better, one may want to read **P** later when **P** is called.

Strategy P(x): Given $\#a_0\$text\not{c}$ 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 $\not{c}$);
      continue to simulate $M$ until $p(h_1)=b_i0$;
      $i:=i+1$
  until $S_1$ or $S_2$ or $S_3$=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 of the main theorem.

Lemma 1 (The Matching Lemma): Let the *text* be *regular* (Def. 3.3.3A) 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. 3.3.3B), 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: Lemma 1 is true for a k-NFA, for any k.

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. 3.3.3B) segment of X. For input $I = \# 1^k X 1^{k'} \$ Z 0 a_1 0 a_2 ... a_{l+2} 0 ¢$, where $a_i = 1^m X 1^m$, let Z be regular (Def. 3.3.3A) with no occurrence of a_0 and no more than l occurrences of s in it. If there is a time when $p(h_1) = (s$ 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.

Proof of Lemma 2: Divide s equally as $s = s_1 s_2 s_3$. Fix any a_i, the s in it is not matched by h_1. As in the Matching Lemma, for $s = s_1 s_2 s_3$ we can find s_1', s_2', s_3' such that each s_i' can be constructed by using $X-s$ and some $4m(l+2)$ long information, so that if we replace s by $\hat{s} = \hat{s}_1 \hat{s}_2 \hat{s}_3$ where $\hat{s}_i$ is either s_i or s_i', M will be in the same status as if s were not replaced by $\hat{s}$ at times $p(h_1) = \hat{s}_1$, or $\hat{s}_2$, or $\hat{s}_3$ of a_i. Notice that $s_i' \neq s_i$ because s_i' is not random relative to $X-s$.

Now we start to simulate M from the beginning. We will try to cheat M so that it fails to match some s_i and we can apply the Matching Lemma. Our strategy is as follows: Run M on I; for $i = 1,2,..,l+1$, do (1) or (2) in the following.

(1) If s_1 and s_2 in a_i are matched (by h_2 and h_3), change s_3 when this matching occurs; to s_3';

(2) If h_2 passes s_1 or s_2 without matching it, then interrupt M as soon as this happens. Keep the s in this a_i unchanged, change every s_j in $a_{l>i}$ to s_j', and halt.

Now if (2) is true at the ith step, then a_i is the only good occurrence of the pattern and $s_1 s_2$ of a_i is independent. But $s_1 s_2$ of a_i is never matched. If rule (1) is always true, then some s_j (which is independent) in a_{l+2} cannot be well-matched, clearly. In both cases, the Matching Lemma is violated. □ (Lemma 2)

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 ¢ sign, then Y is not random.

Proof of Lemma 3: The partial current *ID* (current *ID* without the part that no head can see any more) at time t can be specified by the following short information less than $|X|+2(|k|+|k'|)/3$ long. X plus $O(\log|Y|)$ information suffices for constructing the *text*. For specifying the positions of the 2 heads that are not in a_0, $O(\log|X|)$ suffices by assumption. For the third head, if it is not in a_0, then we simply specify it using information less than $|m|+\log|X|$ long. Suppose it is in a_0, then if it is in X of a_0 we specify k' by $|k'|$ information and the head's position by $\log|X|$ information. If it is in k' of a_0, then we only specify the distance from this head to the \$ sign using information of less than $|k'|$ long.

After specifying the partial input, we find the smallest i and j such that appending 01^iX1^j at the end (before ¢ sign) makes M accept. So we know $k=i$ and $k'=j$. This implies that Y is not random because we can reconstruct Y

with less than $|Y|$ information. $\square$ (Lemma 3)

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).

Proof of Lemma 4: We consider each h_i that passed s before time t. Let us consider the way h_i passes through s. We call a position p in s a *singular point* if while some head is staying at p, only the heads reading 1's move and some head moves more than $|M|+2log\ |X|$ steps. Obviously there can be only 2C singular points for each head because once the head runs into a singular point a whole block 1's must be crossed by some other head. Now we fix the largest segment $\bar{s}$ of s which does not contain a singular point. We know $|\bar{s}| \geq |s|/6C$. Since $\bar{s}$ is random relative to $Y-\bar{s}$ it follows that there must be many $\hat{s}$'s such that

$$(*) \qquad \hat{ID}_{p(h_i)=\bar{s}} = \hat{ID}_{p(h_i)=\hat{s}}, \text{ for all } h_i \text{ that passed } s \text{ before time } t.$$

where we used $\hat{ID}_{p(h_i)=x}$ to denote

$$(p(h_1), p(h_2), p(h_3), \text{ state of } M)$$

at the time when $p(h_i)=x$. Now we are going to find such an $\hat{s}$ without using too much information other than X. Particularly, we do not want to use $|m|$, or $|k|$, or $|k'|$ information other than X. We will exhaustively search for an $\hat{s}$

such that (*) is true. For each h_i in (*), we alphabetically generate $\hat{s}$ and run M from the time that h_i is at the beginning of $\bar{s}$ and $\hat{s}$ until we find an $\hat{s}$ such that (*) is true. To do this, if other 2 heads read only portions of X then we need only $O(log\,|X|)$ information to specify the starting and ending positions of the other 2 heads. But if some head runs into a block of 1^m or even 1^k or $1^{k'}$ of the pattern, then instead of using $|m|$ information by the obvious way, we only specify the number of 1's which were read when h_i went through $\bar{s}$. This is obviously short for we assumed $\bar{s}$ does not contain a singular point. Finally replacing $\bar{s}$ in s by $\hat{s}$ gives us the s' which satisfies the four conditions in the lemma. □ (Replacement Lemma)

Lemmas 4-1, 4-2, 4-3 are variants of the Replacement Lemma that are needed in the application. By the method of Lemma 1 and Lemma 4 we obtain Lemma 4-1 as below.

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.

Proof of Lemma 4–2: Lemma 4-1 cannot be applied directly since otherwise we cannot obtain an independent e. We 'slice' each s_i into $10*C*D$ pieces. For each s_i choose the piece v that was compared (See Definition 3.3.3C) with a shortest segment w_v of some s_1 and then apply Lemma 4-1 on v to get $v' \neq v$. Therefore $K(v' | X - s_1 + w_v) \leq 10 log |X|$. By a simple counting, w_v can be at most $|s_1|/10C$ long. After doing this $C-1$ times (for each $s_{i>1}$) there are $C-1$ segments, total $|s_1|/10$ long, of s_1 have been used for the construction of replacements. We conclude that there is a segment e, of s_1, such that $|e| = |s_1|/2C$ and e is independent. Finally we arbitrarily choose another segment of s_1, disjoint from e, to apply Lemma 4-1. □ (Lemma 4-2)

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.

Proof of Lemma 4–3: Canceling the last sentence in the Lemma 4-2's proof gives us the proof of Lemma 4-3. □ (Lemma 4-3)

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.

Proof of Lemma 5: Assume there are C occurrences of s in *text*. Append $a_1 0 a_2 0 ... a_{C+2} 0$ after *text* and before ¢, where $a_i = 1^m X 1^m$ for $i=1,..,C+2$. Continue to run M from time t and stop as soon as (1) or (2) below happens.

(1) $p(h_1)=a_{C+2}0$. Then no s_3 in any $a_{1\le i\le C+2}$ is matched. By the 2-head Lemma we are done since s_3 is independent and *text* is *easy*;

(2) s_3 of some $a_{1\le i\le C+2}$ is matched. Then cut away the un-read input. It is clear that, for those $a_{1\le i\le C+2}$ which are not canceled, there is a d, $1\le d\le 6$, such that s_{3d} in any a_i is not matched. By the Replacement Lemma we replace s_{3d} of every remaining a_i by some easy replacement $s_{3d}' \neq s_{3d}$ such that, (a) $K(s_{3d}' \,|X-s) < 10 log\,|X|$, and (b) after the replacement for $j>0$ when $p(h_1)=a_j 0$, M is in the same status as if no replacement occurred.

(2.1) If the matching was done by h_1 and h_2, then the conditions of Lemma 3 are satisfied and we are done.

(2.2) If (h_3,h_1) or (h_3,h_2) did the matching of s_3, then $p(h_3)>(s_2$ of $a_0)$. Append $a_{C+3}0 ... a_{2C+4}0$ after *text* and before ¢, where $a_i=1^m X 1^m$, for $C+3\le i\le 2C+4$. Now we go on simulating M and repeat above arguments (1) and (2) one more time, except using s_2 instead of s_3, and s_{2d} instead of s_{3d}. Notice that $s-s_{3d}$ is independent by (a) in (2). If we get back to case (1) or (2.1) then we are done. If we have case (2.2), then $p(h_3)>\$$ (instead of s_2 of a_0), the conditions of Lemma 3 are satisfied and we are done. □ (Lemma 5)

Lemma 6: Let the *text* in input $\#a_0\$text\cent$ 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) $s<p(h_1),p(h_2)<\cent$, (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.

Proof of Lemma 6: Continue to simulate M and stop as soon as one of the following cases happens.

(1) $p(h_3)>(1^k$ of $a_0)$. As soon as this happens, we replace s_6 of a_j by some $s_6' \neq s_6$ by Lemma 4-1. Since $s_1 \cdots s_5$ is independent, the conditions of Lemma 5 are satisfied, we are done.

(2) $p(h_1)=text$ (before $\cent$). Since $p(h_3)<(X$ of $a_0)$, s is not well-matched yet.

(*) We now apply $\mathbf{P}(s)$:

If S_1 is true, then $p(h_3)>(1^k$ of $a_0)$, we are done as in case (1);

If S_3 is true, then s cannot be well-matched for this input, which contradicts to the Matching Lemma;

If S_2 is true, then $\mathbf{P}$ appends $a_f=1^{C_1+l*C_2}X1^m$ to the input. We continue to run M until one of the following case first happens.

(2.1) If $p(h_3)>(1^k$ of $a_0)$. We delete last 1^m of a_f and we are done, as in (1).

(2.2) $p(h_1)=(X$ of $a_f)$.

(2.2.1) Some s_h of a_f is not matched. Notice that by assumption s_h of a_j is not well-matched. If h_2 and h_3 do not do any matching of X before h_2 reaches a_f, then Lemma 4-3 can be applied so that only a_j contains a_0. Then it is easy to see that there exists a substring e of s_h in a_j, of length at least $|s_h|/c$ for some constant c, that cannot be well-matched. This contradicts to the Matching Lemma. If h_2 and h_3 do some matching of X before h_2 reaches a_f, then at first step the matching happens, if h_1 is at ¢, we apply Lemma 4-3 so that only a_f could contain a_0. Then we can vary l to find k, by the method of Lemma 3, concluding Y is not random. If h_1 is not at ¢, we apply Lemma 4-2 and Lemma 3.

(2.2.2) If all s_j's of a_f are matched (by h_1 and h_2), then let $X=X_1sX_2$, change a_f to $1^m X_1 s_1 s_2 s_3$, and repeat (*)'s process.

Notice that (2.2.2) above can not happen more than a constant number of times, since *text* is *easy* and at least one full s has to be jumped over each time (2.2.2) is true. □ (Lemma 6)

Now we continue to prove Theorem 3.3.3. We construct an *easy text*. Let the partial input be

(A) $\#a_0\$a_1 0 \cdots,$

where $a_1=1^m X 1^m$. 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 $a_1'=1^m x_1 x_2$. Add $a_2=1^m X 1^m 0$ 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 $a_t=1^m X 1^m$ 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 $a_t=1^m x_1 x_2 x_3$.

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) $\ldots a_2 0 a_3 0 \ldots a_{i-1} 0 1^m x_{p1} \ldots x_{p4} 0 ¢$.

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 **P** adds $a_f = 1^{C_1 + l*C_2} X 1^m$ 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 $\cent$, we are done by Lemma 4-2 and Lemma 5. And if $p(h_1)=\cent$ 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 $1^m X 1^m$, 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.)

□ (Theorem 3.3.3)

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 the proofs 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.

3.3.4. String-matching by a 2-way k-DFA with k-1 heads blind

A 2-way k-DFA is just like a k-DFA but each head can go both directions. We assume that a 2-way k-DFA stops by entering a final state. 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 3.3.3, we prove a lower bound on the time to do string-matching required by a 2-way k-DFA with $k-1$ blind heads. And we will also give an upper bound for some simple matching problem. 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 3.3.4A: 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.

Proof of Theorem 3.3.4*A*: Suppose M does string-matching in $o(n^2/logn)$ time, where M is a 2-way k-DFA with $k-1$ blind heads. Let h_1 be the non-blind head and $h_2, \ldots, h_k$ be the $k-1$ blind heads. Fix a long enough random string X and consider input

$$\#X\$0^{|X|}X\cent$$

on the input tape. Define the crossing sequence (c.s.) of h_1 at a fixed position p of the input tape of M to be a sequence of $\overline{ID}$'s of form (state of M, positions of

$h_2, \ldots, h_k)$ which specifies the status of M when h_1 passes this position. Now consider $|X|$ c.s.'s under $0^{|X|}$. If each one is of length greater than $|X|/k^2 log|X|$, then M takes $O(n^2/logn)$ time, a contradiction. Otherwise there exists a c.s. of length less than $|X|/k^2 log|X|$. But then from this c.s., which can be described by less than $|X|/2$ information, we can reconstruct X by a short program as follows: For each X', form input $\#blank\$0^{|X|}X'¢$. Start to simulate M from the first $\overline{ID}$ of our short c.s. at position p, going only to the right. Each time h_1 runs back to the position p we match the current status of M against the next $\overline{ID}$ of the c.s. and if they match we take the following $\overline{ID}$ in the c.s. and continue to simulate M from it (Right turn only!); if they do not match, then $X' \neq X$, we interrupt the simulation. Thus if the simulation is finished correctly, we can conclude $X'=X$. □ (Theorem 3.3.4A)

One may wonder whether this *logn* factor can be canceled. For this type of inputs the answer is NO, as the next theorem demonstrates.

Theorem 3.3.4B: $\bar{L}=\{\#x\$yx¢\}$ 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 $L'=\{\#a_0\$a_1*a_2*\ldots*a_l¢ \mid a_0=a_i$ 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.

Proof of Theorem 3.3.4B: We sketch an M accepting $\bar{L}$ with one regular head h_1 and two blind heads h_2 and h_3. To match fast, M will match block by block with each block roughly $logn$ long, where n is the length of input. The algorithm for M follows.

Step_0: h_1,h_2,h_3 at # sign. h_2 (the counter) moves one step right.

Step_i:

Step_i. 1: h_1 moves right one step, and (h_2,h_3) double (times 2) the size of the counter held by h_2 (still held by h_2 after modification). h_2 moves right one more step (plus 1) if h_1 reads a one; If h_1 reads a zero h_2 does not move. If h_2 did not pass the ¢ sign, then repeat *Step_i.* 1, else go to *Step_i.* 2. {Remember a $logn$ long block in a counter held by h_2.}

Step_i. 2: h_2 holds the counter and h_3 goes to the $ sign. h_1 and h_3 move simultaneously to the left until h_1 reads # sign. h_1 and h_3 switch positions. Then h_1 goes to ¢ sign. h_1 and h_3 go backward at the same time until h_3 reads # sign. {This places h_1 in the corresponding matching position in the *text*.}

Step_i. 3: h_2 and h_3 decrease the counter, held by h_2, by half (divide by 2). If there is a remainder 1 then h_1 mush read a 1; If the division is even, then h_1 must read a 0. h_1 moves one step left. Repeat *Step_i.* 3 until the h_2 counter becomes zero. {matching a block}

Step_i. 4: h_1 goes back to the corresponding position for the next block in the pattern with the help of h_2 and h_3. To do this, remember current (after *Step_i.* 3) distance between h_1 and the ¢ sign. Send h_1 back to *pattern* with the same distance to the \$ sign. This is actually the position of h_1 before *Step_i* is started. Then repeat the process of *Step_i.* 1 once more.) Go to *Step_i*+ 1. {Re-set h_1 for next step $(i+1)$.}

end.

By standard calculation, *Step_i* is repeated no more than $|pattern|/logn$ times. For each repetition, *Step_i* requires $O(n)$ time. Therefore total is $O(n^2/log\ n)$. The correctness of this algorithm is clear. □ (Theorem 3.3.4B)

3.3.5. Probabilistic checking is easier than probabilistic generating

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 faster than the Strassen's or Pan's algorithm 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.

We will prove a lower bound which says a block cannot be moved faster than n^2 time even with the help of a random number generator. We follow [G]. A PTM is a TM equipped with a random number generator. A PTM 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.

It is a very interesting result by Freivalds [F] 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 3.3.5: 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.

Proof of Theorem 3.3.5: Assume M does the job in $o(n^2)$ time. We fix a random string x of enough length. Consider the crossing sequences (c.s.) at the # signs. Let the number of computations is T (each for one random sequence).

Since each computation uses $o(n^2)$ time, we have, in total, $o(n^2 T)$ elements of the c.s. at # signs. Let $\epsilon = 1/2 - \delta$. Then there must exist a position i at some # sign, such that the c.s.'s at i are short, say of length $< \frac{n}{10|M|}$, for $T' = (1-\delta)T$ computations, since otherwise the total number of c.s.'s for all computations is going to be $O(n^2 T)$, a contradiction.

Suppose we found above i. Then for those T' computations, the c.s.'s at position i are shorter than $\frac{n}{10|M|}$. Notice that each c.s. corresponds to one or more computations that cause this c.s. Among these T' c.s.'s (there might be a lot same c.s.'s), at least one c.s. corresponds to those computations where more than half of them produce correct output, since otherwise the error probability exceeds $(1-\delta)/2 = \epsilon$.

Fix above c.s. We can give a short program to produce the x as follows, we generate all possible random strings to run M, pick out those computations which match the c.s. at position i. The majority of these computations should output x at the $0^{|x|}$ position, and our short program will output this x. The information we need for this program is the c.s. of length $n/10|M|$, $logn$ for the #'s and 0's. Therefore the total space needed is less than n, contradicting to the randomness of x. □ (Theorem 3.3.5)

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 a n^2 lower bound for accepting the palindromes were proved long time ago by Hennie [H2].

CHAPTER 4

Discussion

This concluding chapter consists of remarks on several major open problems in this thesis.

Most tractable open problems leftover from this thesis are from Chapter 3. The only problems leftover from Chapter 2 are those problems that have both positive and negative solutions in the presence of some oracles. Whereas, there are several seemingly tractable, but important, open problems contained in Chapter 3.

One such problem is to characterize exactly the power of one nondeterministic tape. In Chapter 3, the gap between the $O(n^2)$ upper bound and the $\Omega(n^2/lognloglogn)$ lower bound is left open. Can we close the gap? Theorem 3.2.6B suggests that new language must be constructed. In [L3], the following theorem was proved.

Theorem: If a one tape nondeterministic machine is equipped with a random number generator, than it can simulate two tapes in less than quadratic time with error probability less than any fixed $\epsilon > 0$.

From this result and Theorem 3.2.6B, is it reasonable to suspect that the Hartmanis-Stearns $O(n^2)$ upper bound is not optimal, say, for a one tape nondeterministic machine simulating a two pushdown store machine?

Another problem which is more promising is to improve the $\Omega(n^{1.5}/logn)$ lower bound on one tape simulating two pushdown stores.

In Chapter 3, we developed some techniques for resolving the problem whether a k-head DFA can do string-matching. Although we have answered the question negatively for the cases of $k=2$ and $k=3$, the general question is still open. It would also be interesting to see a simple proof of Theorem 3.3.3.

Apart from the problems dealt with in the thesis, there are other important related problems. For example, to prove similar one tape versus two tape results for the case of *off-line* computation (where the input is two-way). No such results are known. Once two-way input is considered, the problems become very hard. Some related results for 2-way machines can be found in [DG] and [Ch].

CHAPTER 5

Appendix: Definitions and Notation

This appendix presents the major concepts and standard notation of computational complexity, used in this thesis. Other unusual terms are defined where they are used. We assume that the reader is familiar with the basic definitions presented in [HU].

The following abbreviations and definitions will be used. (N)TM stands for (non-)deterministic Turing machine with one 2-way input tape and some work tapes. Here we call a tape '2-way' if the head on the tape can go both directions, whereas we call a tape '1-way' if the head can only go from left to right. A k-tape *on-line* TM is a TM with one 1-way input tape and k (2-way) work tapes. k-tape *real time* TM is an on-line TM that runs only a constant number of steps for each input bit. By *on-line* (*real time*) *computation* we mean the computation made by an on-line (real time) TM. A DFA is a deterministic finite automaton. A PDA is a deterministic pushdown automaton. A k-DFA is a deterministic finite automaton with k reading heads on the 1-way input tape. k-NFA is the nondeterministic version of k-DFA. P is the class of languages accepted by a TM in polynomial time. NP is The class of languages accepted by a NTM in polynomial time. $PSPACE$ is the class of languages accepted by a TM or NTM in polynomial space. E (EE) is the class of languages accepted by a TM in (double) exponential time. NE (NEE) is the class of languages accepted by a NTM in (double) exponential time. $ESPACE$ ($EESPACE$) is the class of languages accepted by a TM or NTM in (double) exponential space. The classes EEE, $NEEE$ and so on are defined similarly. $DTIME[t(n)]$ ($NTIME[t(n)]$) stands for the class of languages accepted by a TM (NTM) in time $ct(n)$ for some constant c. Note, $DTIME[t(n)]=DTIME[ct(n)]$ for any constant c by the linear time speedup theorem of [HLS]. $DTIME[t^c(n)]$ is an abbreviation for $\bigcup_{d>0} DTIME[t^d(n)]$. $NTIME[t^c(n)]$ is an abbreviation for $\bigcup_{d>0} NTIME[t^d(n)]$. $CoNP$ is the class of languages whose complements are in NP. $CoNTIME[t(n)]$ is the class of languages whose complements are in $NTIME[t(n)]$. $CoNTIME[t^c(n)]$ stands for $\bigcup_{c>0} CoNTIME[t^c(n)]$.

We use $\overline{L}$ to denote the complement of language L, that is, Σ^*-L, where Σ is some finite alphabet under consideration. If M is a Turing machine, then $L(M)$ is the language accepted by M. If L_1 and L_2 are two languages, we say L_1 is ***polynomial-time reducible*** to L_2 if there is a polynomial-time computable function r so that $x \in L_1$ if and only if $r(x) \in L_2$. A language L is ***complete*** with respect to complexity class C if every language in C is polynomial-time reducible to L and L is in C.

We call a Turing machine M a $T(n)$ ***machine*** if it runs in time $T(n)$. Let C be a complexity class. We call a Turing machine M a C ***machine*** if it runs within the complexity bound of C. A list of machines $\{M_1, M_2, \cdots\}$ is a *standard enumeration* of C if

(1) for each language L in C there are infinite number of machines $\{M_{i_1}, M_{i_2}, \cdots\} \subseteq \{M_1, M_2, \cdots\}$, such that $L = L(M_{i_k})$ for $k = 1, 2, \cdots$;

(2) every M_i on the list is a C machine. (Notice that $L(M_i) \in C$ is not enough.)

A function $f(n) > n$ is *time-constructible* if some multitape TM on input 1^n can put down $f(n)$ markers on a work tape in time $f(n)$.

Let there be given two functions f and g. We write $g(n) = o(f(n))$ if $\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$. We write $g(n) = O(f(n))$ if there are constants C_1 and C_2 such that $C_1 f(n) < g(n) < C_2 f(n)$ for almost every n. If a and b are integers, we use $a \mid b$ to mean a divides b. The notation $\binom{a}{b}$ denotes the number of ways of choosing a objects from b objects.

Let $\Sigma = \{0,1\}$. Most languages in this thesis are subsets of Σ^*. If S is a set, then $|S|$ denotes the number of elements in S. Following the definition of [HIS], a set S is *sparse* if for each integer n there are only a polynomial number of strings of length n in S. Formally, S is *sparse* if there is a fixed c such that $|S \cap \Sigma^n| \leq n^c + c$. We call a set S a *tally* set if $S \subseteq \{1\}^*$.

An oracle machine M^A (with oracle A) has an extra write-only oracle tape and three special states named $q_?$, q_y, q_n. When the state of $q_?$ is entered, M^A asks if the string on the oracle tape is in oracle A. If it is in A, then state q_y is entered, otherwise state q_n is entered.

CHAPTER 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.

[BGS] T. Baker, J. Gill, and R. Solovay, Relativizations of the $P=?NP$ question, SIAM J. Comp., 4 (1975) pp. 161-173.

[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.

[CKS] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer, Alternation, J. ACM. 28, 1 (Jan. 1981) pp. 114-133.

[Ch] M. Chrobak, Variations on the technique of Duris and Galil, preprint, Institute of Mathematics, Polish Academy of Science (1984).

[C] S.A. Cook, The complexity of theorem proving procedures, Proc. ACM STOC (1971) pp. 151-158.

[C1] S.A. Cook, A hierarchy for nondeterministic time complexity, JCSS 7, 4 (Aug. 1973) pp. 343-353.

[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.

[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, Information Processing, 77 (1977) pp. 839-842.

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

[GJ] M.R. Garey and D.S. Johnson, Computers and intractability, a guide to the theory of NP-completeness, 1979, W.H. Freeman and Co., San Francisco, Calif.

[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.

[H] J. Hartmanis, On sparse sets in NP-P, Inf. Proc. Let. 2 (1983) pp. 55-60.

[H1] J. Hartmanis, Feasible computations and provable complexity properties, CBMS-NSF Regional Conference Series in Applied Mathematics 30, SIAM Monograph (1978).

[HLS] J. Hartmanis, P.M. Lewis II, and R.E. Stearns, Classification of computations by time and memory requirements, Proc. IFIP Congress 65, Spartan, N.Y. (1965) pp. 31-35.

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

[HIS] J. Hartmanis, N. Immerman, and V. Sewelson: On Sparse Sets in NP-P: EXPTIME vs. NEXPTIME, Proc. 15th ACM STOC (April 1983) pp. 382-391

[HY] J. Hartmanis and Y. Yesha, Computation times of NP sets of different densities, Proc. 10th ICALP, Lecture notes in computer science 154, Springer-Verlag (1983) pp. 319-330.

[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).

[K1] W. Kowalozyk, Some connections between presentability of complexity classes and the power of formal systems of reasoning, Proc. MFCS'84 (1984) pp. 364-369.

[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.

[LLR] L. Landweber, R. Lipton, and E. Robertson, On the structure of sets in *NP* and other complexity classes, Theor. Comp. Sci., 18 (1982) pp. 95-103.

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

[L1] M. Li, Lower bounds on string-matching, TR-84-636, Department of Computer Science, Cornell University (July 1984).

[L2] M. Li, On the power of one tape (I), manuscript (May 1984).

[L3] M. Li, Separating the nondeterministic time hierarchy by tally sets, manuscript (Jan. 1983).

[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).

[LZ] M. Li and Z. Zhang, On the power of one tape (II), draft (1984).

[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).

[MS] W. Maass and A. Schorr, Speedup of 1-tape Turing machines by bounded alternation, preprint (1983).

[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.

[PPST] W.J. Paul, N. Pippenger, E. Szemeredi, and W. Trotter, On determinism versus nondeterminism and related problems, Proc. 24th IEEE FOCS (1983) pp. 429-438.

[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.

[RS] C.W. Rackoff and J.I. Seiferas, Limitations on separating nondeterministic complexity classes, 10,4 SIAM J. of Comp. (1981) pp. 742-745.

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

[SFM] J.I. Seiferas, M.J. Fischer, and A.J. Meyer: Separating nondeterministic time classes. J.ACM, 25, 1, (1978) pp. 146-167.

[S] M. Sipser, On relativization and the existence of complete sets, 9th ICALP, Lecture Notes in Computer Science, Springer Verlag, Berlin (1982) pp. 523-531.

[V] P.M.B. Vitanyi, Real-time simulations of multicounters by oblivious one tape Turing machines, Proc. 14th ACM STOC, (1982) pp. 27-36.

[V1] P.M.B. Vitanyi, On the simulation of many storage heads by one, 10th ICALP, Lecture Notes in Computer Science 154, Springer Verlag, Berlin (1983) pp. 687-694

[V2] 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.

[Z] S. Zak, A Turing machine hierarchy, to appear in Theor. Comp. Sci.