MIT/LCS/TM-303

THE POWER OF THE QUEUE

MING LI
LUC LONGPRE
PAUL M. B. VITANYI

APRIL 1986

The Power of the Queue

Ming Li

Department of Computer and Information Science
The Ohio State University, Columbus, OH 43210

Luc Longpré

Department of Computer Science
University of Washington, Seattle, WA 98195

Paul M.B. Vitányi

M.I.T. Laboratory for Computer Science
Cambridge, MA 02139
and
Centrum voor Wiskunde en Informatica
Amsterdam, The Netherlands

April, 1986

ABSTRACT

Queues, stacks (pushdown stores), and tapes are storage models which have direct applications in compiler design and the general design of algorithms. Whereas stacks (pushdown store or last-in-first-out storage) have been thoroughly investigated and are well understood, this is much less the case for queues (first-in-first-out storage). This paper contains a comprehensive study comparing queues to stacks and tapes. We address off-line machines with a one-way input, both deterministic and nondeterministic. The techniques rely on algorithmic information theory (Kolmogorov Complexity).

Keywords & Phrases : tape, stack, queue, pushdown stores, determinism, nondeterminism, off-line one-way input, time complexity, lower bound, upper bound, simulation, algorithmic information theory, Kolmogorov complexity

Note : This paper will be presented at the *Structure in Complexity Theory Conference,* to be held at the University of California, Berkeley, June 2-5, 1986.

The work of the third author was supported in part by the Office of Naval Research under Contract N00014-85-K-0168, by the Office of Army Research under Contract DAAG29-84-K-0058, by the National Science Foundation under Grant DCR-83-02391, and by the Defense Advanced Research Projects Agency (DARPA) under Grant N00014-83-K-0125.

1. Introduction

Queues, stacks (pushdown stores), and tapes are storage models which have direct applications in compiler design and the general design of algorithms. Whereas stacks (pushdown store or last-in-first-out storage) have been thoroughly investigated and are well understood, this is much less the case for queues (first-in-first-out storage). In this paper we present a comprehensive study comparing queues to stacks and tapes. We address off-line machines with a one-way input. In particular, 1 queue and 1 tape (or stack) are not comparable:

(1) Simulating 1 stack (and hence 1 tape) by 1 queue requires $\Omega(n^{4/3}/\log n)$ time in both the deterministic and the nondeterministic cases.

(2) Simulating 1 queue by 1 tape requires $\Omega(n^2)$ time in the deterministic case, and $\Omega(n^{4/3}/\log n)$ in the nondeterministic case;

We further compare the relative power between different numbers of queues:

(3) Nondeterministically simulating 2 queues (or 2 tapes) by 1 queue requires $\Omega(n^2/(\log^2 n \log\log n))$ time and deterministically simulating 2 queues (or 2 tapes) by 1 queue requires $\Omega(n^2)$ time. The second bound is tight. The first is almost tight.

(4) We also obtain the simulation results for queues: 2 nondeterministic queues (or 3 pushdown stores) can simulate k queues in linear time. One queue can simulate k queues in quadratic time.

It has been known for over twenty years that all multitape Turing machines can be simulated on-line by 2-tape Turing machines in time $O(n \log n)$ [HS2], and by 1-tape Turing machines in time $O(n^2)$ [HU]. Since then, many other models of computation have been introduced and compared. (See [Aa, DGPR, HS1, HS2, HU, LS, PSS, Pa, Vi2].) In addition to different storage mechanisms, real-time, on-line and off-line machines have been studied. An on-line machine is expected to give an answer after reading each prefix of the input. In this paper, we consider the off-line machines, where an answer is given only once the whole input has been read. We also use the one-way input convention, where the machine has a one-way input, a finite control and access to some storage.

The relative power of stacks and tapes is more or less well known. For example, for the nondeterministic case, we know that 1 stack $<$ 1 tape $<$ 2 stacks $<$ 3 stacks $=$ k stacks $=$ k tapes, where $A < B$ means that B can simulate A in linear time, while A cannot simulate B in linear time. In most of the cases, close lower and upper bounds are known for the simulation [Ma, Li1, Vi1, LV, Li2].

In this paper, we give a complete characterization of (off-line) queue machines. The main theorems show that one queue machines are not comparable to one stack or one tape machines, both deterministically and nondeterministically. We also compare the relative power of machines having different number of queues. We use Kolmogorov complexity techniques [Ko, Ch, So] to prove the theorems, together with some new techniques to enable us to deal with queues. The Kolmogorov complexity of a string x, $K(x)$, is the length of the shortest program printing the string x. By a simple counting argument, we know that there are strings x of each length such that $K(x) \geq |x|$. These strings are called *incompressible* or *K-random.*

In section 2, we introduce the jamming lemma which is used in further proofs. In section 3, we show that deterministically simulating a queue by a tape takes quadratic time (infinitely often). (For the lower bound on the simulation time of 1 queue by 1 tape in the nondeterministic case, see [Li3, LV].) In section 4, we have a lower bound for non-deterministically simulating a stack by a queue. In section 5, we present lower and upper bounds for simulating k queues by 2 queues or 2 queues by one queue.

2. The Jamming Lemma

In this section, we are concerned with one-tape and one-queue off-line TM's where the Turing machine has one 1-way input tape in addition to one work tape or one queue, each with one head. We will call the input tape head h_1 and work tape head or queue head h_2. We say that a poll occurs hen the head h_1 moves one cell. At any time t, $h_i(t)$ denotes the position of head h_i on its tape.

In the following lower bound proofs, the input will be separated into blocks. We will observe the behavior of the machine as the head polls the successive cells in a block. Although the definitions and the Jamming Lemma are stated with respect to one-tape TM's for simplicity, they also apply to one-queue machines where the work tape is replaced by a queue.

Definition 2.1: Let x_i be a block of input, and R be a tape segment on the storage tape. We say that M *maps* x_i *into* R if h_2 never leaves tape segment R while h_1 is reading x_i. We say M maps x_i *onto* R if h_2 traverses the *entire* tape segment R without leaving R while h_1 reads x_i.

Definition 2.2: A *crossing sequence* (c.s.) associated with the boundary between two contiguous work tape cells is a sequence of ID's of the form $(M(t), h_1(t))$, where $M(t)$ is the state of the machine at time t, for each time t when the machine crosses that boundary.

We prove an intuitively straightforward lemma for one-tape machines with one-way input. The lemma states that M cannot poll too many input symbols, with h_2 located on a given small tape segment bordered by short c.s.'s, without losing some information. Formally:

Jamming Lemma. *Let the input string start with $x\# = x_1 x_2 \cdots x_k \#$, with the x_i's blocks of equal length C. Let R be a segment of M's storage tape and let l be an integer such that M maps each block $x_{i_1}, \ldots, x_{i_l}$ (of the x_i's) into tape segment R. The contents of the storage tape of M, at time $t_\#$ when $h_1(t_\#) = |x\#|$ and $h_1(t_\# - 1) = |x|$, can be reconstructed by using only the blocks $x_{j_1} \cdots x_{j_{k-l}}$ which remain from $x_1 \cdots x_k$ after deleting blocks $x_{i_1}, \ldots, x_{i_l}$, the final contents of R, the two final c.s.'s on the left and right boundaries of R, a description of M and a description of this discussion.*

Remark 2.3: If we want to give a description of a sequence of different strings of variable length, we use self-delimiting strings, adding $O(\log n)$ bits for each string of length n.

Remark 2.4: Roughly speaking, if the number of missing bits $\sum_{j=1}^{l} | x_{i_j} |$ is greater than the number of added description bits then the Jamming Lemma implies that either $x = x_1 \cdots x_k$ is not incompressible or some information about x has been lost.

Proof of the Jamming Lemma. Let the two positions at the left boundary and the right boundary of R be l_R and r_R, respectively. We now simulate M. Put the blocks x_j of $x_{j_1} \cdots x_{j_{k-l}}$ in their correct positions on the input tape (as indicated by the h_1 values in the left and right c.s.'s). Run M with h_2 staying to the left of R. Whenever h_2 reaches point l_R, the left boundary of R, we interrupt M and check whether the current ID matches the next ID, say ID_i, in the c.s. at l_R. Subsequently, using ID_{i+1}, we skip the input up to and including $h_1(t_{i+1})$, adjust the state of M to $M(t_{i+1})$, and continue running M. After we have finished left of R, we do the same thing right of R. At the end we have determined the appropriate contents of M's tape, apart from the contents of R, at $t_{\#}$ (i.e., the time when h_1 reaches $\#$). Inscribing R with its final contents from the reconstruction description gives us M's storage tape contents at time $t_{\#}$. Notice that although there are many unknown x_i's, they are never polled since h_1 skips over them because h_2 never goes into R.

Remark 2.5: If M is nondeterministic, then we need to rephrase "contents of storage tape" by "legal contents of storage tape", which simply means that some computation path for the same input would create this storage tape contents.

3. Lower bound for simulating one queue by one tape

We present a tight lower time bound for deterministic simulation of one queue by one off-line tape with one-way input. (For a lower bound for the nondeterministic case, see [LV] or [Li3].)

Remark. Only in this section 3, $g(n) \in \Omega(f(n))$ means "there is a positive constant δ such that $g(n) \geq \delta f(n)$ infinitely often". Everywhere else the results hold for the stronger variant of Ω: "there exist a positive constant δ and a positive integer n_0 such that $g(n) \geq \delta f(n)$ for all $n \geq n_0$".

Let $\leq_{prefix}$ mean 'is a prefix of.' Let $\Sigma = \{0,1\} \times \{0,1,\overline{0},\overline{1},\epsilon\}$, where ϵ denotes the empty string, and consider the words over Σ of the form

$$(a_1,b_1)(a_2,b_2) \cdots (a_n,b_n) \tag{3.1}$$

such that for all i, $1 \leq i \leq n$,

$a_i \in \{0,1\}$ and $b_i \in \{0,1,\overline{0},\overline{1},\epsilon\}$

$b_1 b_2 \cdots b_i \leq_{prefix} \hat{a}_1 b_1 \hat{a}_2 b_2 \cdots \hat{a}_i b_i$,

where for any pair $(a,b) \in \Sigma$ we define $\hat{a}$ by

$\hat{a} = a$ if $b = \epsilon$

$\hat{a} = \overline{a}$ if $b \neq \epsilon$.

Remark. Words of this form constitute the witness language L_q below, which is real-time acceptable by a queue but which requires $\Omega(n^2)$ time for acceptance by a tape. Think of the sequence $a_1 a_2 \cdots a_n$ as the n-length sequence of bits to be stored consecutively in the queue, and the sequence $b_1 b_2 \cdots b_n$, of length m $(m \leq n)$, as the sequence of bits which are consecutively unstored from the queue. (Note, that while $a_i \neq \epsilon$ for all i, it is possible that $b_i = \epsilon$ for some i $(1 \leq i \leq n)$. That is, (a_i, b_i) specifies that ϵ be unstored.) For technical reasons in the proof below, we have to complicate this scheme. All of the prefix of $a_1 a_2 \cdots a_n$ which has been stored in the queue previously, needs to remain stored in the queue forever. Nonetheless, to force the queue to operate correctly we need to be able to unstore it. To combine both requirements, each pair (a_i, b_i) causes the queue not only to store a_i and to unstore b_i (possibly ϵ), but also to store b_i anew. Below we show that the scheme of barred and unbarred a_i's, related to whether or not the associated 'unstore' b_i's are ϵ or not, makes it possible to retrieve the complete sequence of a_i's, in the order they have been stored originally, from the queue contents at each instant.

The *witness* language L_q consists of all words satisfying (3.1). To aid intuition, we can view L_q as the language accepted by a queue Q as follows:

- Initially, Q is empty.
- For all $i \geq 1$, input command '(a_i, b_i)' to Q is interpreted by Q as '*if* $b_i = \epsilon$ *then* append a_i to the rear *else* append $\overline{a}_i$ to the rear; delete b_i up front; append b_i to the rear.' (Here 'action1;action2;action3' denotes the sequential execution of action1, action2 and action3.)
- A word $(a_1, b_1) \cdots (a_n, b_n)$ is accepted if the sequence of successive front items deleted in the actual computation by Q on this input is the sequence $b_1, \ldots, b_n$. All other words are rejected.

The properties of words of form (3.1) we need in the sequel are expressed in the following three lemmas.

Lemma 3.1: *For a word of the form (3.1),* $| \hat{a}_1 b_1 \hat{a}_2 b_2 \cdots \hat{a}_i b_i | - | b_1 b_2 \cdots b_i | = i$ *for all* i, $1 \leq i \leq n$.

Proof: Obvious.

Lemma 3.2: *For a word of the form (3.1) we can reconstruct* $a_1 a_2 \cdots a_n$ *from the* n*-length suffix of* $\hat{a}_1 b_1 \hat{a}_2 b_2 \cdots \hat{a}_n b_n$.

Proof: Let the n-length suffix be $x_1 x_2 \cdots x_n$ with $x_i \in \{0,1,\overline{0},\overline{1}\}$ $(1 \leq i \leq n)$. By (3.1) one of the following two cases must hold (note that the combination $x_{n-1} \in \{0,1\}$ *and* $x_n \in \{\overline{0},\overline{1}\}$ is impossible):

(a) Assume $x_n, x_{n-1} \in \{0,1\}$. Then $a_n = x_n$ and $b_n = \epsilon$ by (3.1). Consequently, $x_1 x_2 \cdots x_{n-1}$ is the $(n-1)$-length suffix of $\hat{a}_1 b_1 \hat{a}_2 b_2 \cdots \hat{a}_{n-1} b_{n-1}$ by (3.1).

(b) Assume $x_{n-1} \in \{\overline{0},\overline{1}\}$. Then $\overline{a}_n = x_{n-1}$ and $b_n = x_n$ by (3.1). Consequently, $x_n x_1 x_2 \cdots x_{n-2}$ is the $(n-1)$-length suffix of $\hat{a}_1 b_1 \hat{a}_2 b_2 \cdots \hat{a}_{n-1} b_{n-1}$ by (3.1). (Because b_n is the last *unstored* symbol which has been appended to the rear of the queue, it is the last symbol to have been unstored from the front of the queue. Therefore, to restore the queue contents just before (a_n, b_n) is processed, we delete suffix $a_n b_n$ from $x_1 x_2 \cdots x_n$ and prefix the remaining string with b_n.)

Iterating this reasoning n times we recover all of $a_1 a_2 \cdots a_n$. This proves the lemma.

Lemma 3.3: *For a word of the form (3.1) with* $|b_1 \cdots b_n| = m$, *we can reconstruct* $a_1 a_2 \cdots a_{m/2}$ *from* $b_1 \cdots b_n$.

Proof: Let

$$b_1 b_2 \cdots b_n = x_1 x_2 \cdots x_m, \;\; x_i \in \{0,1,\overline{0},\overline{1}\} \;(1 \le i \le m).$$

By (3.1) we have $\hat{a}_1 = x_1$.

(a) If $x_1 \in \{0,1\}$ then $a_1 = x_1$ and $b_1 = \epsilon$. Consequently, $x_2 \cdots x_m$ is the $(m-1)$-length prefix of $\hat{a}_2 b_2 \cdots \hat{a}_n b_n$ by (3.1).

(b) If $x_1 \in \{\overline{0},\overline{1}\}$ then $\overline{a}_1 = x_1$ and $b_1 = x_2$. Consequently, $x_3 \cdots x_m$ is the $(m-2)$-length prefix of $\hat{a}_2 b_2 \cdots \hat{a}_n b_n$ by (3.1).

Iterating this reasoning $m/2$ times we recover all of $a_1 a_2 \cdots a_{m/2}$. This proves the lemma.

Theorem 3.2. *It requires $\Omega(n^2)$ time to deterministically simulate one queue by one off-line tape with one-way input.*

Proof. (I). Assume, by way of contradiction, that an off-line deterministic 1-tape machine M with one-way input accepts L_q in time $T(n) \notin \Omega(n^2)$. We derive a contradiction by showing that then some incompressible string has too short a description. Without loss of generality, it can be assumed that M has a semi-infinite storage tape $[0,\infty)$ on which it writes only 0's and 1's, The positions at time t are denoted by $h_1(t)$ and $h_2(t)$. By t_i we denote the time when the ith input command is polled, i.e., $h_1(t_i) = i$ and $h_1(t_i - 1) = i-1$. Fix a constant C and the word length n as large as needed to derive the desired contradictions below and such that the formulas in the sequel are meaningful. Below we show that $T(m) > m^2/C^4$, for some m, $\sqrt{n}/C \le m \le n$, which contradicts the assumption and proves the theorem.

First, choose an incompressible string $x \in \{0,1\}*$ of length n. We consider the behavior of M on a fixed input prefix. This can be any string z such that $x = x_1 x_2 \cdots x_n$, $y = y_1 y_2 \cdots y_n$ and $z = (x_1, y_1)(x_2, y_2) \cdots (x_n, y_n)$, for some y such that z satisfies (3.1). Therefore, $z \in L_q$. If many polls occur while the head h_2 is in some small area, then we can show that x is not incompressible (Case 1). Otherwise, we choose particular y_i's, among the possibilities which remain under this constraint, so as to suit the argument in Case 2 below.

Case 1 (Jammed). Fix an integer m such that $\sqrt{n}/C \le m \le n$ (any such integer m will do) and consider the m-length prefix $z(m)$ of z. By (3.1), if z is in L_q then so is each prefix of z, so in particular $z(m) \in L_q$. Assume, by way of contradiction, that in the accepting computation on $z(m)$ at least $2m/C$ polls occur, with h_2 on a particular (m/C)-length tape segment $R = [a, a+m/C)$. Consider the two tape segments R_l and R_r of length $|R|/4$ left and right of R. Choose positions p_l in R_l and p_r in R_r with the shortest c.s.'s in their respective tape segments. These c.s.'s must both be shorter than m/C^2, for if the shortest c.s. in either tape segment is longer than m/C^2 then M uses $T(m) \geq m^2/4C^3$ time, which is a contradiction. (If $0 \leq a < m/4C$ then set $|R_l| = a$, so that $R_l R R_r \subset [0,\infty)$. Choose $p_l = 0$ and note that the length of the associated c.s. can be set to 0.) We show that a short program can be constructed which accepts only x. Let the bits of $x_1 \cdots x_m$ polled with h_2 outside tape segment $[p_l, p_r]$, concatenated in the order in which they occur in x, form a string u.

As explained below, we can construct a program to check if a string $x' \in \{0,1\}^*$ equals x, using as a description the values of n, m, C, a, the locations of p_l and p_r, the two c.s.'s at p_l and p_r, the self-delimiting version of u, the bits $x_{m+1} \cdots x_n$, the final contents of $[p_l, p_r]$ at time t_{m+1}, the state of M at time t_{m+1} and $h_2(t_{m+1})$.

This description of x requires no more than $n - \frac{m}{4C}$ bits, for sufficiently large C and n. However, this contradicts the incompressibility of x since $K(x) \geq n$ and $m \geq \sqrt{n}/C$.

To check whether a string x' equals x, check $|x'| = n$ and $x'_{m+1} \cdots x'_n = x_{m+1} \cdots x_n$. By the Jamming Lemma (using the above information as related to M's processing of the input) reconstruct the contents of M's storage tape at time t_{m+1}, after processing $z(m) = (x_1, y_1) \cdots (x_m, y_m)$. Simulate M from time t_{m+1} onwards on an input suffix

$$(0, y_{m+1})(0, y_{m+2}) \cdots (0, y_{2m}) \tag{3.2}$$

with $|y_{m+1} y_{m+2} \cdots y_{2m}| = m$, and such that M accepts for the chosen y_i's $(m+1 \leq i \leq 2m)$. It is easy to see from (3.1), that there is such a suffix (3.2) for which M accepts if $x'_1 x'_2 \cdots x'_m = x_1 x_2 \cdots x_m$. In that case $x' = x$, and by (3.1) and Lemma 3.1, $y_{m+1} y_{m+2} \cdots y_{2m}$ equals the m-length suffix of $\hat{x}_1 y_1 \cdots \hat{x}_m y_m$. By Lemma 3.2, we can retrieve $x_1 x_2 \cdots x_m$ from this suffix. Suppose, there is a $x' \neq x$ such that

$$z'(m) = (x'_1, y'_1)(x'_2, y'_2) \cdots (x'_m, y'_m) \tag{3.3}$$

matches the description above, and $z'(m)$ drives M into the same configuration at time t'_{m+1} of M's $(m+1)$th poll in its computation, as the configuration into which $z(m)$ drives M at time t_{m+1}. Consequently, the concatenation of (3.3) and (3.2) is also accepted by M. Note, that x' differs from x only in the first m bits, in particular in those bits polled with h_2 positioned in tape segment $[p_l, p_r]$. We can cut and paste the computations based on $z'(m)$ inside $[p_l, p_r]$ and based on $z(m)$ outside $[p_l, p_r]$, and still have M accept. The 'cut and paste' computation is accepting up to the $(m+1)$th poll because both computations satisfy the description above, and afterwards because the two computations are identical from the $(m+1)$st poll onwards. Let the resulting string composed in the obvious way from $x_1 \cdots x_m$ and $x'_1 \cdots x'_m$ be $\chi(m)=\chi_1 \cdots \chi_m$ with $\chi_i \in \{x_i, x'_i\}$ $(1 \le i \le m)$. Above we saw that we can retrieve $x_1 x_2 \cdots x_m$ from $y_{m+1} \cdots y_{2m}$, by Lemma 3.1 and Lemma 3.2. However, this contradicts the acceptance by M of the cut and paste computation based on $z(m)$ and $z'(m)$, because that entails the retrieval of $\chi(m) \ne x_1 x_2 \cdots x_m$ from $y_{m+1} \cdots y_{2m}$ by (3.1), Lemma 3.1 and Lemma 3.2.

Case 2 (Not jammed). Let n' be any integer such that $\sqrt{n}/C \le n' \le n$. Let $z(n')$ be the n'-length prefix of an n-length input z. By (3.1), if $z \in L_q$ then $z(n') \in L_q$. Assume, by way of contradiction, that in the accepting computation of M on $z(n')$ at most $2n'/C$ polls occur with h_2 on any particular (n'/C)-length tape segment $R = [a, a+n'/C)$.

We now define the particular input z we need. Let $x = x_1 x_2 \cdots x_n$ be as in Case 1. Determine the y_i's $(1 \le i \le n)$ in input $z = (x_1, y_1) \cdots (x_n, y_n)$ as follows.

(1) Let M start its computation with $y_1 = \epsilon$. So first (x_1, y_1) is polled.

(2) Let M continue its computation and suppose we have determined $(x_1, y_1) \cdots (x_{i-1}, y_{i-1})$ and M polls for the ith time. Let t_i be the time at which M polls (x_i, y_i). If $h_2(t_i) \in [0, n/4)$ then $y_i = \epsilon$, else $y_i \ne \epsilon$. In the latter case y_i is be determined uniquely from $(x_1, y_1) \cdots (x_{i-1}, y_{i-1})$ by using the relation $y_1 y_2 \cdots y_i \le_{prefix} \hat{x}_1 y_1 \cdots \hat{x}_{i-1} y_{i-1}$, that is, using (3.1) and the fact that $y_1 = \epsilon$ by (1).

We now fix a particular value m as determined by M's computation on z.

(a) By contradictory assumption (with $n' = n$), we have that $\le n/2$ polls occur on $[0, n/4)$ and $\ge n/2$ polls occur on $[n/2, \infty)$.

(b) Since $T(n) \notin \Omega(n^2)$, we have $h_2(t_i) \in [0, n/4)$, for all i $(1 \le i \le \sqrt{n})$.

Let $l(t)$ and $r(t)$ be the number of polls for (x_i, y_i)'s, with $h_2(t_i) \in [0, n/4)$ and $h_2(t_i) \in [n/4, \infty)$ $(1 \le t_i \le t)$, respectively. By (a) and (b) there is an integer m such that $l(t) > r(t)$, for $1 \le t < t_m$, $l(t_m) = r(t_m)$ and $\sqrt{n}/C \le m \le n$. This m is the *break even* length where the number of polls left and right of position $n/4$ on the tape is equal for the first time.

Claim 1. As a consequence of this definition of m and (1) and (2), it follows that $r(t_m) = |y_1 \cdots y_m| = m/2$ for input prefix

$$z(m) = (x_1, y_1) \cdots (x_m, y_m).$$

Since each prefix of z satisfies (3.1), we can retrieve $x_1 \cdots x_{m/4}$ from prefix $y_1 \cdots y_m$ of $\hat{x}_1 y_1 \cdots \hat{x}_m y_m$ by Lemma 3.3.

Claim 2. By definition, all y_i's in $y_1 \cdots y_m$, which are different from ϵ, are polled on $[n/4, \infty)$. Since $l(t_{m/4}) > r(t_{m/4})$, at most $m/8$ of the x_i's in $x_1 \cdots x_{m/4}$ are polled on $[n/4, \infty)$.

In the computation on the m-length prefix $z(m)$ of z, choose the point p with the shortest c.s. in $[n/4 - m/C, n/4)$. This c.s. is shorter than m/C^2; otherwise, the running time $T(m) > m^2/C^3$, which is a contradiction.

As explained below, we can construct a program to check if a string $x' \in \{0,1\}^*$ equals x, using as a description the values of n, m, the position of p, the c.s. at p, the string u of concatenated bits of $x_1 \cdots x_{m/4}$, polled with h_2 on $[p, \infty)$ and the string $x_{(m/4)+1} \cdots x_n$.

This description of x requires no more than $n - \frac{m}{16}$ bits, for sufficiently large C and n. However, this contradicts the incompressibility of x since $K(x) \geq n$ and $m \geq \sqrt{n}/C$.

To check whether a string x' equals x, check $|x'| = n$ and $x'_{(m/4)+1} \cdots x'_n = x_{(m/4)+1} \cdots x_n$. Let u' be the result of deleting the bits in x' in the same positions as the ones used to obtain u from x. These positions are determined by the crossing sequence at p. Check $u' = u$. If the test is negative then $x' \neq x$, else x' can only differ from x on positions where $x_1 \cdots x_{m/4}$'s bits are polled with h_2 on $[0, p)$. Run M on $z'(m)$, that is, the input constructed according to (1), (2), using the m-length prefix $x'_1 x'_2 \cdots x'_m$ of a candidate x'. Whenever h_2 crosses p we interrupt M and check if the current *ID* in the computation is consistent with the corresponding *ID* in the c.s. at p.

By construction everything matches up to the end of processing input $z'(m)$, and M accepts, if $x' = x$. Assume that $x' \neq x$ matches the description as well. Therefore, $x'_1 x'_2 \cdots x'_{m/4} \neq x_1 x_2 \cdots x_{m/4}$ and $x'_i = x_i$ for all i $(m/4 + 1 \leq i \leq n)$. Let the input $z'(m)$, based on $x'_1 x'_2 \cdots x'_m$ and constructed according to (1),(2), be

$$z'(m) = (x'_1, y'_1)(x'_2, y'_2) \cdots (x'_m, y'_m) .$$

Let the input based on $x_1 x_2 \cdots x_m$, constructed according to (1), (2), be

$$z(m) = (x_1, y_1)(x_2, y_2) \cdots (x_m, y_m) .$$

By assumption, x' and x differ only on the first $m/4$ bits, and then only on the bits that are polled left of p. Let the final accepting position of h_2 for M's computation on $z(m)$ be right of p. (If it is left of p interchange z and z' below.) Cut and paste the computations on $z(m)$ and $z'(m)$ such that M runs on input $z'(m)$ with h_2 left of position p, and M runs on input $z(m)$ with h_2 right of position p. Let $\varsigma(m)$ be the input composed in this way from $z'(m)$ and $z(m)$. By construction, the computation on $\varsigma(m)$ is also an accepting computation of M. Consequently, $\varsigma(m)$ satisfies (3.1). Then,

$$\varsigma(m) = (a_1, b_1)(a_2, b_2) \cdots (a_m, b_m)$$

with (a_i, b_i) is either (x_i, y_i) or (x'_i, y'_i) $(1 \leq i \leq m)$. Because both $z(m)$ and $z'(m)$ match the description, (x'_i, y'_i) is polled right of position p if and only if (x_i, y_i) is polled right of position p for all i, $1 \leq i \leq m$. Therefore, $y_i = \epsilon$ if and only if $y'_i = \epsilon$ if and only if (x_i, y_i) is polled left of position p if and only if (x'_i, y'_i) is polled left of position p for all i, $1 \leq i \leq m$. Consequently, the sequences of 'unstored' symbols unequal ϵ in the computations on $z(m)$, $z'(m)$ and $\varsigma(m)$ are equal, that is,

$$b_1 b_2 \cdots b_m = y_1 y_2 \cdots y_m \tag{3.4}$$

By assumption, (x_i, y_i) and (x'_i, y_i) are polled left of p and $x_i \neq x_i'$ for some i, $1 \leq i \leq m/4$. Therefore, since $a_i = x'_i$ if the ith poll occurs left of p, and $a_i = x_i$ if the ith poll occurs right of p $(1 \leq i \leq m/4)$, in M's computation on $\varsigma(m)$, we have $a_1 a_2 \cdots a_{m/4} \neq x_1 x_2 \cdots x_{m/4}$. Because $x_1 x_2 \cdots x_{m/4}$ is retrieved from $y_1 \cdots y_m$ by Claim 1, we retrieve $x_1 x_2 \cdots x_{m/4}$ from $b_1 b_2 \cdots b_m$ as well, by (3.4), using Lemma 3.3. However, for $\varsigma(m)$ to satisfy (3.1), we have to retrieve $a_1 a_2 \cdots a_{m/4}$ from $b_1 b_2 \cdots b_m$, by virtue of the construction of $\varsigma(m)$ and Claim 1. Consequently, $b_1 b_2 \cdots b_m$ is not a prefix of $\hat{a}_1 b_1 \hat{a}_2 b_2 \cdots \hat{a}_m b_m$ as required by (3.1). Hence, $\varsigma(m)$ does not satisfy (3.1), which is a contradiction.

Since $m \geq \sqrt{n}/C$, Cases 1 and 2 complete the proof of $T(n) \in \Omega(n^2)$.

(II). With the description of L_q we have already indicated how a queue recognizes this language in real-time.

The theorem follows from (I) and (II).

4. Lower bound for simulating one pushdown by one queue.

In this section, we show that it takes at least $\Omega(n^{4/3}/\log n)$ time for a one-way input one queue nondeterministic machine to recognize the language $\{w\#w^R : w \in \{0,1\}^*\}$.

Because this language can be recognized in linear time by a deterministic pushdown automaton, we can conclude that it takes at least $\Omega(n^{4/3}/\log n)$ time for a one queue nondeterministic machine to simulate a deterministic pushdown automaton.

The intuition behind the proof is that while the queue machine reads w, it has to store all the information in some sequential way on the queue. To compare this information with w^R, the machine will have to go through the queue too many times.

Let h_1 be the read-only head on the one-way input tape. We can view the queue as a tape with two heads h_2 and h_3. The head h_2 is a read-only, one-way head on the queue. The head h_3 is a write-only, one-way head on the queue. Each time something is put on the queue, h_3 writes and each time something is read from the queue, h_2 reads.

Theorem 4.1: A one-way input one queue nondeterministic machine takes time in $\Omega(n^{4/3}/\log n)$ to accept the language $\{w\#w^R : w \in \{0,1\}^*\}$.

Proof: Leading to a contradiction, we suppose that there is an algorithm to accept L in time $T(n)$ which is not in $\Omega(n^{4/3}/\log n)$.

Let $h_i(t)$ denote the position of head i at time t on its respective tape. At time t, the length of the queue is $h_3(t) - h_2(t)$, and the content of the tape between $h_2(t)$ and $h_3(t)$ is called the *actual queue*.

Let x be an incompressible string. We separate x into blocks: $x = x_0 x_1 x_2 \cdots x_m$. Each block x_i for $1 \leq i \leq m$ is separated into p subblocks: $x_i = x_{i1} x_{i2} x_{i3} \cdots x_{ip}$. For the proof of the theorem, we take $m = n^{1/3}$ and $p = n^{2/3}/k_1 \log n$, where k_1 is an appropriately chosen constant. Let $|x|=n$ and $|x_0|=n/2$. Each subblock will have the same length, $c\log n$. We look at any computation of the machine on input $x\#x^R$.

Claim 4.2: If $n/2\le h_1(t)\le 3n/2$, then the length of the queue at time t is at least $n/2-\log n$.

Proof: We know that $K(x_0)\ge n/2-k_2$ for some constant k_2. The result follows because x_0 can be described by the content of the queue, the state of the machine and $h_1(t)$.

Claim 4.3: Let t_j be the time step when the input head enters the block x_j. For at least half of the blocks x_j, we have that $h_2(t_{j+1})<h_3(t_j)$.

Proof: Otherwise the algorithm takes time $\Omega(n^{4/3}/\log n)$.

The machine needs to remember what it reads on the input and code it in some way on the queue. This notion will be captured by the influence relation defined below. What is written on the queue can be a coding of the input and of the rear of the queue. If $h_2(t_{j+1})\le h_3(t_j)$, then we have the nice property that a whole block from the input has to be coded sequentially on the queue, since the reading head on the queue doesn't reach where the coding has started. Let's call the blocks which satisfy this last claim the valid blocks.

Now, we define the influence relation. Let $c_1, c_2, \cdots c_p$ be the cells on the input tape. Let $d_1, d_2, \cdots d_q$ be the cells on the queue. We say that a cell d_j is *directly influenced* by a cell c_i if $h_1(t)=i$ at the time t when h_3 writes on d_j. Similarly, a cell d_j is *directly influenced* by a cell d_i if $h_2(t)=i$ at the time t when h_3 writes on d_j.

The *influence relation* is the transitive closure of the direct influence relation. We say that c_i (or d_i) influences d_j if there is a chain of direct influences from c_i to d_j. A block of cells influences a cell if and only if at least one of the cells in the block influences it.

The influence relation will allow us to talk about where the information can be stored on the queue. Notice that during the computation, the content of a cell may still be dependent on some other input cell even if that input cell has no influence on it, due to the finite control of the machine. This minor problem will not cause any trouble.

Claim 4.4: For any block x_j such that $h_2(t_{j+1})<h_3(t_j)$, we have that each cell in x_j influences a disjoint set of cells on the queue. Moreover, the regions influenced by these cells form an ordered sequence of regions on the actual queue at any later time.

Now we look at what happens when the input head h_1 reads the second part of the input. Let t_j' denote the time when the head h_1 enters the block x_j^R corresponding to x_j.

Claim 4.5: There is at least one valid block x_j such that $h_2(t_{j-1}')<h_3(t_j')$. (Remember that $x_{j-1}'^R$ follows $x_j'^R$.)

Proof: Otherwise the algorithm takes time $\Omega(n^{4/3}/\log n)$.

In the following two claims, we mention cycles and crossing sequences. A *cycle* is any span of time from time t to time t' such that $h_2(t')=h_3(t)$. The *crossing sequence* associated with the border between cell d_i and cell d_{i+1} is the list of states of the machine when any head goes from cell d_i to cell d_{i+1}. Because the tape is in fact a queue, the crossing sequence will have length 2.

Each block influences a series of regions on the tape, one for each cycle of the queue. The *crossing sequence* around a list of regions is the concatenation of the crossing sequences under the border of each region.

Claim 4.6: Throughout r cycles, starting at time t_0', the actual queue always has length at least $n^{2/3}-(r+k_4)\log n$, for some constant k_4.

Proof: Let x_i be the block provided by claim 4.5. Let x' be the string x where x_i has been deleted. Because the regions influenced by the x_{ij} are ordered on the actual queue, and the regions influenced by the $x_{ij}{}^R$ are in reverse order, there is only one contiguous region which can be influenced by both a subblock x_{ij} and its corresponding subblock $x_{ij}{}^R$. We call this region the overlapping region.

At any time after t_0', the string x can be totally described by x', the index of x_i, the actual queue, the crossing sequences around the overlapping region and the content of the regions that were overlapping at each cycle.

Claim 4.7: The machine makes $\Omega(n^{2/3}/\log n)$ cycles after t_0'.

Proof: The string x can be described by x', the index of x_i, the crossing sequence around the overlapping region and the content of the regions that were overlapping at each cycle. At each cycle, this information is of length $O(\log n)$, so it takes $n^{2/3}/\log n$ cycles to gather enough information. (At the end, we don't need the actual queue, so r has to be large to compensate.)

By the last two claims, the machine takes time in $\Omega(n^{4/3}/\log n)$.

5. Simulating more queues by less queues

In this section we study the power of queue machines with different number of queues. We first show that 2 queues are as good as k queues in the nondeterministic case. This motivates our research concerning a small number of queues. We also show that 1 queue can simulate k queues in quadratic time, deterministically or nondeterministically. We will provide tight, and almost tight, lower bounds for our simulations mentioned above.

5.1. Upper bounds

Theorem 5.1: Two pushdown stores can simulate one queue in linear time, both for deterministic and nondeterministic machines.

Proof: Let P be a two pushdown store machine with 2 pushdown stores $pd1$, $pd2$. To simulate a queue, every time the a symbol is pushed into the queue, P pushes the same symbol into $pd1$. If a symbol is taken from the queue, then P pops a (the same) symbol from $pd\,2$ if $pd\,2$ is not empty. If $pd\,2$ is empty then P first unloads the entire contents of $pd\,1$ into $pd\,2$ and then pops the top symbol from $pd\,2$. At the end of the input, P accepts iff the 1 queue machine accepts.

Theorem 5.2: Two queues can nondeterministically simulate k queues for any fixed k in linear time.

Proof: This is actually the same technique Book and Greibach [BG] used to prove the same theorem for tapes. The 2 queue machine guesses the computation of the k-queue machine computation and put this guess on 1 queue. Then use the other queue to simulate the computation of each of the k queues of the simulated machine and check its correctness against the guess on the first queue. We refer the reader to [BG] for the details. (This simulation takes $O(kn)$ time. But it can be improved to *real* time using the methods developed in [BG].)

Theorem 5.3: 3 pushdown stores can nondeterministically simulate k queues in linear time.

Proof: Combine ideas from the above 2 theorems. *I.e.* , guess the computation of the k queue machine, and put the guess into one pushdown store. Save this guess also to another pushdown store (but put a marker on the top). Then simulate a queue and check the correctness of the guess. (The simulation needs 2 pushdowns, one of the pushdowns has the guessed computation saved on the bottom.) After simulating one queue, retrieve the guessed content and again put it into 2 pushdowns. Repeat this process for each queue.

Notice that a strange phenomenon occurs here. When we have 1 queue and 1 pushdown store, 1 queue is better in the sense that 1 queue can accept all the *r.e.* languages but 1 pushdown cannot. However, when we have more pushdown stores, more pushdown stores seems to be better than queues because they are more efficient.

Theorem 5.4: One queue can simulate k queues in quadratic time, both deterministically and nondeterministically.

Proof: Here, use some basic simulation schemes.

This also relates to a interesting problem of whether "2 heads (on one tape) are better than 2 tapes (each with one single head)". Vitańyi [Vi3] showed that 2 tapes cannot simulate a queue in real time if one of the tape has only $o(n)$ cells to use. Our result here shows that 2 pushdowns can simulate a queue in linear time. It would be interesting to know whether this can be done in real time. The question of how to simulate k deterministic queues by 2 queues (like the Hennie Stearns simulation in the tape case [HS2]) remains open.

5.2. Lower bounds

We now prove optimal lower bounds for above simulations. We define the language L:

$$L = \{\, a \;\&\; b_0^1 b_1^1 \cdots b_k^1 \#$$
$$b_0^2 b_0^3 b_1^2 b_2^2 b_1^3 b_3^2 \cdots b_{2i}^2 b_i^3 b_{2i+1}^2 \cdots b_{k-1}^2 b_{(k-1)/2}^3 b_k^2$$

$$b_0^4 b_{(k+1)/2}^3 b_1^4 b_2^4 b_{(k+3)/2}^3 b_3^4 \cdots b_{2i \bmod (k+1)}^4 b_i^3 b_{(2i+1) \bmod (k+1)}^4 \cdots b_{k-1}^4 b_k^3 b_k^4$$

$n\&\ a$

$$\mid b_i^1 = b_i^2 = b_i^3 = b_i^4 \text{ for } i = 0, \ldots, k \text{ for any odd } k \}$$

The length of each b_i^j is a fixed constant C. The superscripts of b_i's are used only to facilitate later discussions. L can be considered as a modified version of a language used in [Ma]. We have added a string a on the both ends. The purpose of a is to prevent the queue from shrinking since if we choose a to be a long random string then before the second a is read the size of the queue has to be larger than $|a|$. We have to prevent the queue from shrinking because otherwise the crossing sequence argument would not work. In order to prove the lower bounds for queues new techniques, in addition to those used in [Ma,LV], are required.

Theorem 5.5: Simulating two deterministic queues by one nondeterministic queue requires $\Omega(n^2/\log^2 n \ \log\log n)$ time.

Proof: We will show that L defined above requires $\Omega(n^2/\log^2 n \ \log\log n)$ time on a one-queue nondeterministic machine (always with an extra 1-way input tape). Since L can obviously be accepted by a two-queue deterministic machine in linear time, the theorem will follow.

Now, aiming at a contradiction, assume that a one-queue machine M accepts L in $o(n^2/\log^2 n \ \log\log n)$ time. Only for the notational convenience, we think the queue of M as a *circular tape* with just *one queue head*, which combines the push head and the pop head. The head moves clockwise uniformly. The circular tape can augment (insert a tape square) or shrink (delete a tape square) at constant cost in order to mimic a queue. We call it *Queue* and write $|Queue(t)|$ to denote its length at time t. Name the input head h_1 and the queue head h_2. Initially the *Queue* is a point, a degenerated ring.

Choose a large n and a $C > 10|M| + 10$ so that all the subsequent formulas make sense. Choose an incompressible string $X \in \{0,1\}^{2n}$. Let $X = X' X''$ where $|X'| = |X''|$. Equally divide X'' into $k+1 = n/C \log\log n$ parts, $X'' = x_0 x_2 \cdots x_k$, each $C \log\log n$ long. Consider a word $w \in L$ where $a = X'$ and $b_i^j = x_i$ for $1 \le j \le 5$ and $0 \le i \le k$. Fix a shortest accepting path, P, of M on w.

Consider only the path P. Let $t_\&$ be the time when h_1 reaches the first $\&$, $t'_\&$ be the time h_1 reaches the second $\&$, and $t_\#$ be the time when h_1 reaches $\#$.

Claim 1. $|Queue(t_\&)| \ge n/2$.

Proof. If not, we can conclude that $K(X) < |X|$ as follows. For every Y such that $|Y| = |X|$, let $Y = a' \ y_0 \cdots y_k$. Replace the last a after the $\&$ sign in w by a'. Using the (short) description of the queue, start to simulate M from time $t_\&$. By a standard argument, $Y = X$ iff M accepts. Therefore $K(X) < |X|$, a contradiction.

By a similar argument as in Claim 1, we derive Claim 2.

Claim 2. $|\,Queue\,(t\,)\,| \geq n\,/2$ for every $t_{\&} \leq t \leq t'_{\&}$.

Claim 3. The crossing sequence from time $t_{\&}$ to $t'_{\&}$ is shorter than $\dfrac{n}{C^{10}\log^2 n \ \text{loglog}\ n}$.

Proof. Follows directly from Claim 2.

If the *Queue* cells which are scanned or created by h_2, while h_1 is scanning b_i^j, are in $Q=\{q_1, \cdots, q_u\}$, then we say that b_i^j is *mapped into* Q. Notice that b_i^j is at first mapped into a set, Q, of consecutive sequence of cells. But, different from a regular Turing machine tape, Q may become disconnected because other queue squares can be inserted later. We say that b_i^j is *sequentially mapped* if, while h_1 scans b_i^j, h_2 did not scan any *Queue* cell twice (leave and re-enter), that is, h_2 did not make a round trip on *Queue*. We say that b_i^j is *majorly mapped into* Q if b_i^j is sequentially mapped, and there are two substrings, u and v, of b_i^j which are mapped into Q and $|\,u\,| + |\,v\,| \geq \dfrac{|\,b_i^j\,|}{2} - 1$.

Remark. According to above definition, a b_i^j can be majorly mapped into two disjoint sets.

Claim 4. At time $t_{\#}$, *Queue* can be cut into two segments, S_1 and S_2, such that

(1) $S_1 \cap S_2 = \phi$ and $S_1 \cup S_2 = Queue$;

(2) $k\,/4$ $b_i^{\,1}$'s, say $b_{i_1}^1, \cdots, b_{i_{k/4}}^1$, are majorly mapped into S_1, and $k\,/4$ $b_i^{\,1}$'s, say $b_{j_1}^1, \cdots, b_{j_{k/4}}^1$, are majorly mapped into S_2. $\{b_{i_1}^1, \cdots, b_{i_{k/4}}^1\} \cap \{b_{j_1}^1, \cdots, b_{j_{k/4}}^1\} = \phi$.

(3) $|\,S_1\,|,\ |\,S_2\,| \geq n\,/C^2$;

Proof. In our proof only properties (1) and (2) are used, (3) is stated for the sake of completeness. We will only give proofs of (1) and (2). The proof of (3) is very similar to the last part of this proof and we leave the proof of (3) to the interested readers.

First we show that $\dfrac{99k}{100}$ $b_i^{\,1}$ are sequentially mapped. By Claim 2, for $t_{\&} \leq t \leq t'_{\&}$, we always have $|\,Queue\,(t\,)\,| \geq n\,/2$. Therefore, if after time $t_{\&}$ more than $k\,/100$ $b_i^{\,1}$'s are not sequentially mapped, then on each of them M must spend at least $n\,/2$ time to go around the *Queue*. Altogether M would be spending $\Omega(n^2/\text{loglog}\ n\,)$ time, a contradiction. Hence at least $99k\,/100$ $b_i^{\,1}$'s are sequentially mapped.

Now we can easily choose two points $p\,,q$ on *Queue* to cut *Queue* into two parts S_1 and S_2 such that (1) and (2) in the claim are true.

>From now on, we will always consider the *Queue* to be partitioned as S_1 and S_2. The sizes of S_1 and S_2 may increase or decrease. If anything is inserted in the intersection point of S_1 and S_2 then it does not matter in which set we place the inserted *Queue* cell.

The next claim is a simple generalization of a theorem proved by Maass in [M] (Theorem 3.1). The proof of this claim is a simple reworking of the proof in [M].

Claim 5. Let S be a sequence of numbers from $\{0, \ldots, k-1\}$, where $k=2^l$ for some l. Assume that every number $b \in \{0, \ldots, k-1\}$ is somewhere in S adjacent to the number $2b \pmod k$ and $2b \pmod k + 1$. Then for every partition of $\{0, \ldots, k-1\}$ into two sets G and R such that $S=G \cup R$ and $|G|, |R| > k/4$ there are at least $k/(c \log k)$ (for some fixed c) elements of G that occur somewhere in S adjacent to a number from R.

A $k/\sqrt{\log k}$ upper bound corresponding to the lower bound in this lemma is contained in [Li]. A more general, but weaker, upper bound can be found in [Kl]. Notice that any sequence S in L satisfies the requirements in Claim 5.

Claim 6. At time $t'_{\&}$, the b_i's between # and the second & are mapped into *Queue* in the following way: either

(1) a set, $\overline{S}_1$, of $k/c \log k$ b_j's, which belong to $\{b_{j_1}^1, \cdots, b_{j_{k/4}}^1\}$, are mapped into S_1; or

(2) a set, $\overline{S}_2$, of $k/c \log k$ b_i's, which belong to $\{b_{i_1}^1, \cdots, b_{i_{k/4}}^1\}$, are mapped into S_2.

Where $c << C$ is a small constant as used in Claim 5.

Proof. By Claim 3 we can assume that from time $t_{\#}$ to time $t'_{\&}$, h_2 made less than $\frac{k}{C^2 \log k}$ round trips on *Queue*. Therefore by the nature of the queue, only $\frac{2k}{C^2 \log k}$ b_i^j's can be mapped into both S_1 and S_2. Also since h_2 can alternate between S_1 and S_2 less than $\frac{2k}{C^2 \log k}$ times, we complete the proof by applying Claim 5.

Without loss of generality, assume that (1) of Claim 6 is true.

Claim 7. Let t_{end} be the time M ends. Either

(a) There exists a time $t'_{\&} \le t_0 \le t_{end}$ such that $|Queue(t_0)| \le \frac{n}{C^{10} \log n}$ and the crossing sequence from $t'_{\&}$ to t_0 is shorter than $\frac{n}{C^{10} \log n \log\log n}$; or

(b) >From time $t'_{\&}$ to time t_{end} the length of the crossing sequence is shorter than $\frac{n}{C^{10} \log n \log\log n}$.

Proof. If (a) and (b) are both false, then M spends $\Omega(\frac{n^2}{\log^2 n \log\log n})$ time, a contradiction.

By Claim 3 the crossing sequence is shorter than $\frac{n}{C^{10} \log^2 n \log\log n}$ before time $t'_{\&}$. Record this crossing sequence. For every j,k, if $b_j^k \in \overline{S}_1$, then b_j^1 is majorly mapped into S_2. Let u_j, v_j be the substrings of b_j^1 such that

(i) $|u_j| + |v_j| \ge \frac{|b_j^1|}{2} - 1$ and

(ii) u_j, v_j are mapped into S_2.

Let $S_{uv}=\{u_j,v_j \mid b_j^k \in \overline{S}_1$ for some $k>1\}$. Notice that $\sum_{u_j,v_j \in S_{uv}} (\,|u_j|+|v_j|\,) \geq \frac{n}{3C\log n}$.

Now we describe a program which reconstructs X with less than $|X|$ information. Consider every Y such that $|Y|=|X|$ and $Y=a\, y_0 \cdots y_k$ for some $y_0 \cdots y_k$.

(1) Check if Y is the same as X at positions other than those places occupied by $u_l, v_l \in S_{uv}$.

(2) If (1) is true, then construct the input, w_Y, as before except with x_i replaced by y_i for $i=0,1,\cdots,k$.

(3) Run M following path P up to time $t_{\&}$.

(4) We distinguish between two cases according to Claim 7.

Case 1. (b) of Claim 7 is true. Record the crossing sequence from $t'_{\&}$ to t_{end}. Continue to run M from $t'_{\&}$ to t_{end} such that h_2 never goes into S_2. Whenever h_2 reaches the border of S_2 it matches the current *ID* with the crossing sequence. If they match M *jumps* over S_2 and, using the next *ID* on the other side of S_2 to start from, \$M\$ continues until time t_{end}.

Case 2. (a) of Claim 7 is true. Record the crossing sequence from time $t'_{\&}$ to time t_0 and the contents of S_2 at time t_0. Simulate M, with h_2 staying outside of S_2, from time $t'_{\&}$ to time t_0 similar to Case 1. At time t_0, M puts the (short) contents of S_2 in the position of S_2 and then finishes the computation in the normal way.

(5) By the end M accepts iff $Y=X$. Notice that since M is nondeterministic, by "accept" we mean that there is an accepting path.

Now the information we used in this program is only the following.

(1) $X - S_{uv}$, plus the information to describe the relative locations of $b_j \in \overline{S}_1$ in X and the relative locations of u_j, vnn_j in $b_j \in \overline{S}_1$. Using the coding method described in the previous part of the paper, this would require at most

$$|X| - \frac{n}{2c\log n} + \frac{n}{\frac{C}{3}c\log n} + o\left(\frac{n}{\log n}\right),$$

where the second term is for the u_l, v_l in S_{uv}, the third term is for the information to describe the relative positions of $b_j \in \overline{S}_1$, and the last term is for the information needed to describe the relative positions of u_l, v_l in each b_l

(2) Description of the crossing sequence, of length less than $\frac{n}{C^9\log n\ \log\log n}$, around S_2. Again by the method used in previous part of this paper, this requires at most

$$\frac{n}{C^8\log n\ \log\log n} \text{ bits.}$$

(3) Description of the contents of S_1 at time t_0 when (a) of Claim 6 is true. But $|\mathit{Queue}(t_0)| \leq \frac{n}{C^{10}\log n}$.

(4) Extra $O(\log n)$ bits to describe the program discussed above.

The total is less than $|X|$. Therefore $K(X) < |X|$, a contradiction.

Corollary. Simulating two deterministic tapes by one nondeterministic queue requires $\Omega(n^2/\log^2 n \ \log\log n)$.

Proof. Since L can also be accepted by a two tape Turing machine in linear time.

Theorem 5.6: It requires $\Omega(n^2)$ time to simulate two deterministic queues by one deterministic queue.

Proof Idea. Define a language L_1 as follows. (Below, $a, x_i, y_i \in \{0,1\}^*$.)

$$L_1 = \{ a \ \& \ x_1\$x_2\$ \cdots \$x_k\#y_1\$ \cdots \$y_l\#(1^{i_1},1^{j_1})(1^{i_2},1^{j_2})\ldots(1^{i_s},1^{j_s}) \ \& \ a \mid$$
$$x_p = y_q \ \& \ (p = i_1 + \ldots + i_t, \ q = j_1 + \ldots + j_t) \ \& \ 1 \leq t \leq s \ \}.$$

L_1 can be accepted by a two queue deterministic machine in linear time. But using the techniques in Theorem 1 and in the proof of one deterministic Turing machine tape requiring square time for this language (See [LV]), it can be shown that L_1 requires $\Omega(n^2)$ for a one queue deterministic machine. We omit the proof.

Remark. The above lower bounds are similar to the case of one tape vs two tapes [Ma,LV]. However, the proofs require special techniques to handle the queues. Still we do not have a lower bound as good as in the nondeterministic tape case [LV,GKS]. We feel that some improvement should be possible.

References

[Aa] Aanderaa, S.O., "On k-tape versus (k-1)-tape real-time computation," in *Complexity of Computation,* ed. R.M. Karp, SIAM-AMS Proceedings, vol. 7, pp. 75-96, American Math. Society, Providence, R.I., 1974.

[BGW]
Book, R., S. Greibach, and B. Wegbreit, "Time- and tape-bound Turing acceptors and AFL's," *J. Computer and System Sciences,* vol. 4, pp. 606-621, 1970.

[Ch] Chaitin, G.J., "Algorithmic Information Theory," *IBM J. Res. Dev.,* vol. 21, pp. 350-359, 1977.

[DGPR]
Duris, P., Z. Galil, W. Paul, and R. Reischuk, "Two nonlinear lower bounds for on-line computations," *Information and Control,* vol. 60, pp. 1-11, 1984.

[GKS]Galil, Z., R. Kannan, E. Szemeredi, "On nontrivial separators for k-page graphs and simulations by nondeterministic one-tape Turing machines," in Proceedings 18th Annual ACM Symposium on Theory of Computing, 1986.

[HS1] Hartmanis, J. and R.E. Stearns, "On the computational complexity of algorithms," *Trans. Amer. Math. Soc.*, vol. 117, pp. 285-306, 1969.

[HS2] Hennie, F.C. and R.E. Stearns, "Two tape simulation of multitape Turing machines," *J. Ass. Comp. Mach.*, vol. 4, pp. 533-546, 1966.

[HU] Hopcroft, J.E. and J.D. Ullman, *Formal Languages and their Relations to Automata*, Addison-Wesley, 1969.

[Kl] Klawe, M., "Limitations on explicit construction of expanding graphs," *SIAM J. Comp.*, vol. 13, no. 4, pp. 156-166, 1984.

[Ko] Kolmogorov, A.N., "Three approaches to the quantitative definition of information," *Problems in Information Transmission*, vol. 1, no. 1, pp. 1-7, 1965.

[Li1] Li, M., "Simulating two pushdowns by one tape in O(n**1.5 (log n)**0.5) time," 26th Annual IEEE Symposium on the Foundations of Computer Science, 1985.

[Li2] Li, M., "Lower Bounds in Computational Complexity," Ph.D. Thesis, Report TR-85-663, Computer Science Department, Cornell University, march 1985.

[Li3] Li, M., "Lower bounds by Kolmogorov-complexity", 12th ICALP, Lecture Notes in Computer Science, 194, pp. 383-393, 1985.

[LV] Li, M. and P.M.B. Vitanyi, "Tape versus queue and stacks: The lower bounds," Submitted for publication.

[LS] Leong, B.L. and J.I. Seiferas, "New real-time simulations of mul- tihead tape units," *J. Ass. Comp. Mach.*, vol. 28, pp. 166-180, 1981.

[Ma] Maass, W., "Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines," *Trans. Amer. Math. Soc.*, 292,2, pp. 675-693, 1985. (Preliminary Version "Quadratic lower bounds for deterministic and nondeterministic one-tape Turing machines," pp 401-408 in Proceedings 16th ACM Symposium on Theory of Computing, 1984.)

[PSS] Paul, W.J., J.I. Seiferas, and J. Simon, "An information theoretic approach to time bounds for on-line computation," *J. Computer and System Sciences*, vol. 23, pp. 108-126, 1981.

[Pa] Paul, W.J., "On-line simulation of k+1 tapes by k tapes requires nonlinear time," *Information and Control*, pp. 1-8, 1982.

[So] Solomonov, R., *Information and Control*, vol. 7, pp. 1-22, 1964.

[Vi1] Vitányi, P.M.B., "One queue or two pushdown stores take square time on a one-head tape unit," Computer Science Technical Report CS-R8406, CWI, Amsterdam, March 1984.

[Vi2] Vitányi, P.M.B., "An N**1.618 lower bound on the time to simulate one queue or two pushdown stores by one tape," *Information Processing Letters*, vol. 21, pp. 147-152, 1985.

[Vi3] Vitányi, P.M.B., "On two-tape real-time computation and queues," *J. Computer and System Sciences*, vol. 29, pp. 303-311, 1984.