institute *for* logic, language *and* computation

TAO JIANG, JOEL I. SEIFERAS,
PAUL M.B. VITÁNYI

Two heads are Better than Two Tapes

CT-94-05, *received*: April 1994

ILLC Research Report and Technical Notes Series
Series editor: Dick de Jongh

Computation and Complexity Theory (CT) Series, ISSN: 0928-3323

Institute for Logic, Language and Computation (ILLC)
University of Amsterdam
Plantage Muidergracht 24
NL-1018 TV Amsterdam
The Netherlands
e-mail: illc@fwi.uva.nl

Two Heads are Better than Two Tapes

TAO JIANG, JOEL I. SEIFERAS, AND PAUL M. B. VITÁNYI

McMaster University
University of Rochester
CWI and Universiteit van Amsterdam

March 10, 1994

ABSTRACT. We show that a Turing machine with two single-head one-dimensional tapes cannot recognize the set $\{ x2x' \mid x \in \{0,1\}^* \text{ and } x' \text{ is a prefix of } x \}$ in real time, although it can do so with three tapes, two two-dimensional tapes, or one two-head tape, or in linear time with just one tape. In particular, this settles the longstanding conjecture that a two-head Turing machine can recognize more languages in real time if its heads are on the *same* one-dimensional tape than if they are on *separate* one-dimensional tapes.

1. INTRODUCTION

The obvious structural parameters for the storage tapes of a Turing machine include the number of tapes, the dimension of the tapes, and the number of heads on each tape. It is natural to conjecture that a deficiency in any such parameter is significant and cannot be fully compensated for by advantages in the others. For the most part, this has indeed turned out to be the case, although the proofs have been disproportionately difficult [Ra63, He66, Gr77, Aa74, PSS81, Pa82, DGPR84, Ma85, LV88, LLV92, MSST90, PSSN90].

The case of deficiency in the number of heads allowed on each tape has turned out to be the most delicate, because it involves a surprise: A larger number of single-head tapes *can* compensate for the absence of multihead tapes [FMR72, LS81]. For example, four single-head tapes suffice for general simulation of a two-head tape unit, without any time loss at all [LS81]. The remaining question is just what, if anything, *is* the advantage of multihead tapes.

The simplest version of the question is whether a two-head tape is more powerful than two single-head tapes. In the case of multidimensional tapes, Paul has shown that it is [Pa84]. His proof involves using the two-head tape to write, and occasionally to retrieve parts of, algorithmically incompressible bit patterns. Because the diameter of the pattern (and hence the retrieval times) can be kept much smaller

Key words and phrases. two-head tape, multihead tape, buffer, queue, heads vs. tapes, multitape Turing machine, real-time simulation, on-line simulation, lower bound, Kolmogorov complexity, overlap.

The first author was supported in part by NSERC Operating Grant OGP0046613. The third author was supported in part by the European Union through NeuroCOLT ESPRIT Working Group Number 8556, and by NWO through NFI Project ALADDIN under Contract number NF 62-376.

We thank Wolfgang Maass for discussions that contributed to an early version of the Anti-Holography Lemma in 1985. We thank Ming Li for other valuable discussions, and for first bringing the problem addressed here to the attention of the first author.

This report will appear in the Proceedings of STOC '94, the Twenty-Sixth Annual ACM Symposium on Theory of Computing.

Typeset by $\mathcal{AMS}$-TEX

than its volume, no fast simulator would ever have time to perform any significant revision or copying of its representation of the bit pattern. On *one*-dimensional tapes, however, retrievals take time that is *not* small compared to the volume of data, and we cannot so easily focus on a nearly static representation of the data. We need some more subtle way to rule out all (possibly very obscure) copying methods that a two-tape machine might employ to keep up with its mission of fast simulation. Our argument below does finally get a handle on this elusive "copying" issue, making use of a lemma formulated more than ten years ago with this goal already in mind [Vi84, Far-Out Lemma below].

Our specific result is that no Turing machine with just two single-head one-dimensional storage tapes can recognize the following language L in real time:[1]

$$L = \{\, x2x' \mid x \in \{0,1\}^* \text{ and } x' \text{ is a prefix of } x \,\}.$$

With a two-head tape, a Turing machine can easily recognize L in real time.

Our result incidentally gives us a tight bound on the number of single-head tapes needed to recognize L in real time, since three *do* suffice [FMR72]. Thus L is another example of a language with "number-of-tapes complexity" 3, rather different from the one first given by Aanderaa [Aa74, PSS81]. (For the latter, even a two-head tape, even if enhanced by instantaneous head-to-head jumps and allowed to operate probabilistically, was not enough [PSSN90].)

2. Overlap

Part of our strategy will be to find within any computation a sufficiently long *sub*computation that is sufficiently well behaved for the rest of our analysis. The behavior we seek involves limitations on repeated access to storage locations, or "overlap" [Aa74, PSSN90].

Our overlap lemma is purely combinatorial, and does not depend at all on the *nature* of our computations or the "storage locations" corresponding to their steps. The use of computational terminology would only obscure the lemma's formulation and proof, so we avoid it.

An *overlap event* in a sequence $S = \ell_1, \ldots, \ell_T$ (of "storage locations", in our application) is a pair (i, j) with $1 \le i < j \le T$, and $\ell_i = \ell_j \notin \{\ell_{i+1}, \ldots, \ell_{j-1}\}$ ("visit and soonest revisit"). If $\omega_t(S)$ is the number of such overlap events "straddling" t (i.e., with $i \le t < j$), then the sequence's *internal overlap*, $\omega(S)$, is $\max\{\, \omega_t(S) \mid 1 \le t < T \,\}$. The *relative* internal overlap is $\omega(S)/T$.

(In our setting, we apply these definitions to the sequence of storage locations *shifted to* on the respective steps of a computation or subcomputation. Without loss of generality, we assume that a multihead or multitape machine shifts exactly one head on each step.)

Overlap Lemma. *Consider any $\delta < 1$ and any $\epsilon > 0$. Every sequence S (of length T, say) with "distinguished-position" density at least δ has a long contiguous subsequence, of length at least $\epsilon' T$ for some constant $\epsilon' > 0$ that depends only on δ and ϵ, with distinguished-position density still at least $\delta/2$, and with relative internal overlap less than ϵ.*

[1] *On-line recognition* requires a verdict for each input prefix before the next input symbol is read, and *real-time recognition* is on-line recognition with some constant delay bound on the number of steps between the reading of successive input symbols. Note that even a *single*-tape Turing machine can recognize L on-line in *cumulative* linear time; but this involves an unbounded (linear-time) delay to "rewind" after reading the symbol 2. In cumulative linear time, in fact, *general* on-line simulation of a two-head one-dimensional tape is possible using just two single-head tapes [St70]; so real time is a stronger notion of "without time loss". (There is an analogous linear-time simulation for two-dimensional tapes [ST89], but the question is open for higher dimensions.)

Proof. Without loss of generality, assume T is a power of 2 that is large in terms of δ and ϵ. We consider only the sequence's two halves, four quarters, eight eighths, etc. Of these, we seek many with sufficient distinguished-position density (at least $\delta/2$) and with internal overlap accounted for by distinct overlap events, planning then to use the fact that each item in S can serve as the second component of at most one overlap event.

Within each candidate subsequence S', we can select a particular straddle point t for which $\omega(S') = \omega_t(S')$, and then we can designate the $\omega(S')$ overlap events within S' that straddle position t as the ones we consider counting. The designated overlap events in S' can be shared by another interval only if that interval includes the corresponding selected straddle point t.

We consider the candidate sequences in order of decreasing length (i.e., halves, then quarters, then eighths, etc.). At each partitioning level, at least fraction $\delta/2$ of the subsequences must have distinguished-position density at least $\delta/2$. (Otherwise, we cannot possibly have the guaranteed total δT distinguished positions in the subsequences on that level, since $(\delta/2)\cdot 1 + (1-\delta/2)\cdot \delta/2 < \delta$.) Among these, we can get distinct countable overlap from

$\lceil(\delta/2)2\rceil = \lceil\delta\rceil$ halves,
$\lceil(\delta/2)4\rceil - \lceil(\delta/2)2\rceil = \lceil 2\delta\rceil - \lceil\delta\rceil$ quarters,
$\lceil(\delta/2)8\rceil - \lceil(\delta/2)4\rceil = \lceil 4\delta\rceil - \lceil 2\delta\rceil$ eighths,
$\lceil(\delta/2)16\rceil - \lceil(\delta/2)8\rceil = \lceil 8\delta\rceil - \lceil 4\delta\rceil$ sixteenths,
etc.

Until we find one of these sequences that has relative internal overlap less than ϵ, this accounts for at least about $\epsilon(\delta/4)T$ distinct overlap events at each level, and hence for more than T distinct overlap events after about $4/(\epsilon\delta)$ levels. This is impossible, so we must find the desired low-overlap sequence at one of these levels. □

3. Kolmogorov Complexity

A key to the tractability of our arguments (and most of the recent ones we have cited [Pa82, Pa84, PSS81, DGPR84, Ma85, LV88, LLV92, PSSN90, Vi84]) is the use of "incompressible data". Input strings that involve such data tend to be the hardest and least subject to special handling.

We define incompressibility in terms of Kolmogorov's robust notion of descriptional complexity [Ko65]. Informally, the Kolmogorov complexity $K(x)$ of a binary string x is the length of the shortest binary program (for a fixed reference universal machine) that prints x as its only output and then halts. A string x is *incompressible* if $K(x)$ is at least $|x|$, the approximate length of a program that simply includes all of x literally. Similarly, a string x is "*nearly*" incompressible if $K(x)$ is "almost as large as" $|x|$.

The appropriate standard for "almost as large" above can depend on the application, but it is usually "$K(x) \geq |x| - c\log|x|$", for some fixed constant c. For our proofs below the constant c might actually have to *depend* on some of the other constants, a possible source of confusion; a more absolute standard such as "$K(x) \geq |x| - \sqrt{|x|}$" should always be good enough, however.

Similarly, the *conditional* Kolmogorov complexity of x with respect to y, denoted by $K(x|y)$, is the length of the shortest program that, *with extra information* y, prints x. And a string x is incompressible or nearly incompressible *relative to* y if $K(x|y)$ is large in the appropriate sense. If, at the opposite extreme, $K(x|y)$ is so small that $|x| - K(x|y)$ is "almost as large as" $|x|$, then we say that y *codes* x [CTPR85].

There are a few well-known facts about these notions that we will use freely, sometimes only implicitly. Proofs and elaboration, when they are not sufficiently obvious, can be found in the literature [especially LV93]. The simplest is that, both absolutely and relative to any fixed string y, there are incompressible strings of every length, and that *most* strings are nearly incompressible, by *any* standard. Another easy one is that significantly long subwords of an incompressible string are themselves nearly incompressible, even relative to the rest of the string. More striking is Kolmogorov and Levin's "symmetry of information" [ZL70]: $K(x) - K(x|y)$ is very nearly equal to $K(y) - K(y|x)$ (up to an additive term that is logarithmic in the Kolmogorov complexity of the binary encoding of the pair (x, y)); i.e., y is always approximately as helpful in describing x as vice versa! All these facts can be relativized or further relativized; for example, symmetry of information also holds in the presence of help from any fixed string z:

$$K(x \mid z) - K(x|y \mid z) \approx K(y \mid z) - K(y|x \mid z).$$

4. Strategy

For the sake of argument, suppose some two-tape Turing machine M does recognize $\{\, x2x' \mid x \in \{0,1\}^* \text{ and } x' \text{ is a prefix of } x \,\}$ in real time. Once a binary string $x \in \{0,1\}^*$ has been read by M, the contents of M's tapes tend to serve as *a very redundant representation of prefixes of* x, because M has to be prepared to retrieve them at any time. (Our problem and this observation were motivation for Chung, Tarjan, Paul, and Reischuk's investigation of "robust codings of strings by pairs of strings" [CTPR85].) One way around this is for M to keep one or the other of its tapes' heads stationed at some stored record of a long prefix of x, as "insurance". The early real-time multihead simulations of buffers [FMR72] do follow this strategy, but we show that a machine with only two tapes will not be able to afford always to use one for insurance: There will always be a significant subcomputation in which the heads on both tapes "keep moving", even "essentially monotonically". Under these circumstances, in fact, we will be able to use part of *the computation itself*, rather than the combination of the two tapes' contents, as the very redundant representation, to contradict the following lemma, which we prove later.

Anti-Holography Lemma. *Consider any C, and consider any x that is long in terms of C, and that is nearly incompressible.*[2] *Suppose $y = y_1 y_2 \cdots y_k$ is a "representation" with the following properties:*

(1) $|y| \leq C|x|$;
(2) *For each ℓ, x's prefix of length $\ell|x|/k$ is coded by $y_{i+1} \cdots y_{i+\ell}$ for each $i \leq k - \ell$.*

Then k is bounded by some constant that depends only on C.

For a T-step subcomputation by M to serve as a representation y that *contradicts* this lemma, we need the following:

(1) A nearly incompressible prefix x of length at least $|y|/C = \Theta(T/C)$ was read before the subcomputation.
(2) There is a parse of the subcomputation into a large number k of pieces so that each prefix of x of length $\ell|x|/k$ is coded in every contiguous sequence of ℓ pieces.
(3) k is (too) large in terms of C.

We accomplish these things by finding a subcomputation that has a spatially monotonic "matching" that is both long and so well separated spatially that needed information on tape contents cannot be spread over many pieces of the subcomputation.

The first step is to define and find "a large matching". In a two-tape or two-head computation or subcomputation, a monotonic sequence of time instants is a *matching* if neither head scans the same tape square at more than one of the time instants. We prove the following lemma later on.

[2] We need $K(x) > \delta|x|$ for some fraction δ that is determined by C, so certainly the usual $K(x) > |x| - \mathcal{O}(\log|x|)$ will be enough if x is long.

Large-Matching Lemma. *If a two-tape Turing machine recognizes*

$$\{\, x2x' \mid x \in \{0,1\}^* \text{ and } x' \text{ is a prefix of } x \,\}$$

in real time, then its computation on an incompressible binary input of length n includes a matching of length $\Omega(n)$. (The implicit constant does depend on the machine.)

In a two-tape or two-head computation or subcomputation, a matching is (spatially) *monotonic* if, for each of the two heads, the spatial order of the corresponding sequence of tape squares being scanned at the specified time instants is strictly left-to-right or strictly right-to-left. The *minimum separation* of a monotonic matching is the least distance between successive tape squares in either corresponding sequence of tape squares.

Monotonization Lemma. *Suppose $\epsilon > 0$ is small in terms of $\delta > 0$. If a two-tape (sub)computation of length T has a matching of length at least δT and internal overlap less than ϵT, then the computation has a monotonic submatching of length $\Omega(\delta/\epsilon)$ and minimum separation $\Omega(\epsilon T)$. (The implicit constants here really are constant, not depending even on the machine; for use below, let c denote the smaller of them.)*

Proof. Without loss of generality, assume T is large in terms of δ and ϵ. Parse the computation into about $\delta/(2\epsilon)$ subcomputations, each including a matching of length at least $2\epsilon T$. Each subcomputation involves a contiguous set of at least $2\epsilon T$ distinct tape squares on each tape. The sets from successive subcomputations touch or intersect, but the overlap bound limits their intersection to less than ϵT tape squares. If we omit every second subcomputation's set, therefore, we get a spatially monotonic sequence of about $\delta/(4\epsilon)$ *non*intersecting sets on each tape. If we further omit every second *remaining* set, then we get a monotonic sequence of about $\delta/(8\epsilon)$ sets on each tape, with successive sets separated by at least $2\epsilon T$ tape squares. To get the desired submatching, simply include one matching-time instant from each of the $\delta/(8\epsilon)$ remaining subcomputations. □

5. Careful Argument

Now let us put together the whole argument, taking care to introduce the "constants" M (and d), δ, ϵ, and ϵ' in an appropriate order, all before the input length n and the particular input string x_0 on which we focus. Each of these values is allowed to depend on earlier ones, but not on later ones.

For the sake of argument, suppose some two-tape Turing machine M does recognize the language $\{\, x2x' \mid x \in \{0,1\}^* \text{ and } x' \text{ is a prefix of } x \,\}$ in real time, say with delay bound d. Citing the Large-Matching Lemma, take $\delta > 0$ small enough so that M's computation on any incompressible input string $x \in \{0,1\}^*$ includes a matching of length at least $\delta|x|$. Let $\epsilon > 0$ be small in terms of d, δ, and M; and let ϵ' be small in terms of d, δ, and ϵ. Let n be large in terms of *all* these constants, and let x_0 be any incompressible string of n bits.

Split the computation by M on input x_0 into an initial subcomputation and a final subcomputation, each including a matching of length $\lfloor \delta n/2 \rfloor$. The number of *steps* in each of these subcomputations will lie between $\lfloor \delta n/2 \rfloor$ and dn. Therefore, the initial one will involve a prefix of x_0 of length at least $(1/d)(\delta n/2) = n\delta/(2d)$, and the final one will have "match density" at least $(\delta n/2)/(dn) = \delta/(2d)$.

Applying the Overlap Lemma to the final subcomputation above, we obtain a subcomputation of some length $T \geq \epsilon' n$, with match density at least $\delta/(4d)$ and relative internal overlap less than ϵ, provided ϵ' was chosen small enough in terms of d, δ, and ϵ. Then applying the Monotonization Lemma, we obtain within this subcomputation a monotonic submatching of minimum separation at least $c\epsilon T$, and of length $2k+1$, where $2k+1$ is either $\lceil c(\delta/(4d))/\epsilon \rceil$ or $\lceil c(\delta/(4d))/\epsilon \rceil - 1$. If ϵ was chosen small, then k will be large. Note that $k\epsilon$ is approximately equal to a constant $c\delta/(8d)$ that depends only on M.

To obtain the desired contradiction to the Anti-Holography Lemma, take y to be a complete record of the T-step subcomputation obtained above, including the symbols scanned and written by each head on each step. To obtain y_1, y_2, ..., y_k, split this record at every second one of the time instants corresponding to the matching of length $2k+1$, starting with the third and ending with the third-to-last. Take x to be x_0's prefix of length $kc\epsilon T/(2d)$. Since $\delta n/(2d)$ exceeds this length (assuming we chose our constants appropriately), all of x was already read during the initial subcomputation above, and hence before the beginning of the subcomputation described by y. Note that, for some constant D that depends only on M,

$$|y| \le DT = \frac{2dD}{kc\epsilon}|x| \approx \frac{16d^2D}{c^2\delta}|x|,$$

and that k is large (in fact, *too* large for the Anti-Holography Lemma) in terms of the constant $C = 16d^2D/(c^2\delta)$, assuming we chose ϵ small enough.

To see that x's prefix of length $\ell|x|/k$ is coded by $y_{i+1}\dots y_{i+\ell}$ (for each appropriate ℓ and i), suppose we interrupt M with "the command to begin retrieval" (i.e., with the symbol 2) at the $(2i+\ell+1)$st of the time instants corresponding to the matching of length $2k+1$. Since M must be able to check the prefix of length $\ell|x|/k$ by reading only the information within distance $d\ell|x|/k = \ell c\epsilon T/2$ of its heads, that prefix must be coded by that information. Since this distance in each direction is just $\ell/2$ times the minimum separation of the matching, and since the matching is monotonic, the same information is available within the subcomputation record y, between the matching's time instants $2i+\ell+1-\lceil \ell/2 \rceil$ and $2i+\ell+1+\lceil \ell/2 \rceil$. Since $y_{i+1}\dots y_{i+\ell}$ runs from the matching's time instant $2i+1 \le 2i+\ell+1-\lceil \ell/2 \rceil$ to the matching's time instant $2i+2\ell+1 \ge 2i+\ell+1+\lceil \ell/2 \rceil$, it too codes the desired prefix.

6. Proof of Anti-Holography Lemma

Without loss of generality, assume k is equal to 2^e for some integer exponent e.[3] Then the target constant can be 2^{2C-1}. Again without loss of generality, assume k is *at most* this target constant *times two*.[4] Finally, without loss of generality, assume that $|x| = n$ is divisible by k, with $x = x_1 \dots x_k$ and $|x_i| = n/k$ for every i.[5]

To obtain short descriptions of y, we abbreviate many of its subwords in terms of just a few prefixes of x, using the symmetry of information. For each $j \le e$, and for $j = e-1$ in particular, this will yield

$$K(y|x_1 \dots x_{2^j}) \le |y| - (1 + j/2)n + \mathcal{O}(\log n).$$

Unless k is smaller than 2^{2C-1}, $e-1$ will be so large that this will imply that $x_1 \dots x_{2^{e-1}}$ codes y. Since y in turn codes all of $x = x_1 \dots x_{2^e}$ this will mean that the first half of x codes the whole string, contradicting the incompressibility assumption for x.

By induction on j ($j = 0, 1, \dots, e$), we actually prove "more local" bounds that *imply* the ones above: For each appropriate i ($i = 0, 1, \dots, k - 2^j$),

$$K(y_{i+1} \dots y_{i+2^j} | x_1 \dots x_{2^j}) \le |y_{i+1} \dots y_{i+2^j}| - 2^j(1 + j/2)n/k + \mathcal{O}(\log n).$$

[3]If it is *not*, then just reduce it until it *is*.

[4]Otherwise, pair up y_i's to reduce k by factors of 2 until it *is*.

[5]If x's length is *not* divisible by k, then just discard at most its last 2^{2C-1} bits, until its length *is* divisible by k.

Both the base case and the induction step are applications of an intuitively clear corollary, the Further-Abbreviation Lemma below, of the symmetry of information. For the base case, we apply the lemma with y' equal to y_{i+1}, x' equal to the null string, and x'' equal to x_1, to get the desired bound on $K(y'|x'')$:

$$\begin{aligned} K(y'|x'') &\leq K(y') - K(x'') + \mathcal{O}(\log n) \\ &\leq |y'| - n/k + \mathcal{O}(\log n). \end{aligned}$$

For the induction step, we let $y'' = y_{i+1} \dots y_{i+2^j}$ and $y''' = y_{i+2^j+1} \dots y_{i+2^{j+1}}$, and apply the lemma with y' equal to $y''y'''$, x' equal to $x_1 \dots x_{2^j}$, and x'' equal to $x_{2^j+1} \dots x_{2^{j+1}}$, to get the desired bound on $K(y'|x'x'')$:

$$\begin{aligned} K(y'|x'x'') &\leq K(y'|x') - K(x'') + \mathcal{O}(\log n) \\ &\leq K(y''|x') + K(y'''|x') - K(x'') + \mathcal{O}(\log n) \\ &\leq |y''| + |y'''| - 2 \cdot 2^j(1 + j/2)n/k - 2^j n/k + \mathcal{O}(\log n) \\ &= |y''| + |y'''| - 2^{j+1}(1 + (j+1)/2)n/k + \mathcal{O}(\log n). \quad \Box \end{aligned}$$

Further-Abbreviation Lemma. *Assume y', x', and x'' are strings of length $\Theta(n)$, with*

$$K(x''|y') = \mathcal{O}(\log n)$$

and

$$K(x''|x') = K(x'') - \mathcal{O}(\log n).$$

(I.e., y' codes x'', which is nearly incompressible relative to x'.) Then

$$K(y'|x'x'') \leq K(y'|x') - K(x'') + \mathcal{O}(\log n).$$

Proof. Let $d(u|v)$ denote a shortest description of u in terms of v, so that $|d(u|v)| = K(u|v)$. Then

$$\begin{aligned} K(y'|x'x'') &\leq K(d(y'|x')|x'x'') + \mathcal{O}(\log n) \\ &\leq K(d(y'|x')|x'' \mid x') + \mathcal{O}(\log n) \\ &\leq K(d(y'|x') \mid x') - K(x'' \mid x') + K(x''|d(y'|x') \mid x') + \mathcal{O}(\log n) \\ &\leq K(y'|x') - K(x''|x') + K(x''|y') + \mathcal{O}(\log n) \\ &\leq K(y'|x') - K(x'') + \mathcal{O}(\log n). \quad \Box \end{aligned}$$

7. Proof of Large-Matching Lemma

Our proof of the Large-Matching Lemma is based on an earlier theorem of Vitányi:

Far-Out Lemma [Vi84]. *If a two-tape Turing machine recognizes*

$$\{\, x2x' \mid x \in \{0,1\}^* \text{ and } x' \text{ is a prefix of } x \,\}$$

in real time, then its "worst-case closest head position"[6] *on incompressible inputs $x \in \{0,1\}^n$ is $\Omega(n)$.*

In other words, incompressible binary data is guaranteed at some point to drive *both* heads of such a machine *simultaneously* far from their original positions. By

[6]If $p_i(t)$ denotes the net displacement of head i at time t, then the "worst-case closest head position" is $\max_t \min_i p_i(t)$.

the continuity of sequential access, of course, this means that the heads actually spend *long intervals* of time simultaneously far from their original positions; and this is the fact that we exploit.

We actually show that even any *two-head* Turing machine (with both heads on the *same* one-dimensional tape) that recognizes our language and that satisfies the conclusion of the Far-Out Lemma also satisfies the desired conclusion of the Large-Matching Lemma. This simplifies the exposition, since we have only one tape to talk about. Note that the "matching" notion does make sense even when both heads are on the same tape.

As earlier, let us take explicit care to introduce our "constants" in an appropriate order. Consider any two-head Turing machine M alleged to recognize

$$\{\, x2x' \mid x \in \{0,1\}^* \text{ and } x' \text{ is a prefix of } x \,\}$$

in real time, say with delay bound d, and that satisfies the conclusion of the Far-Out Lemma. Let c be small enough to serve as the implicit constant in that conclusion. Let ϵ be small in terms of M and c; let δ be small in terms of M, c, and ϵ; let n be large in terms of M, c, ϵ, and δ; and let x be an incompressible string of n bits. Exploiting the conclusion of the Far-Out Lemma, parse x into three pieces, $x = uvw$, such that uv leaves both heads at least cn tape squares from where they started and the length of u is $\lfloor cn/(3d) \rfloor = \Theta(n)$.

Consider M's computation on $uv2u$. The first u must be read before either head gets as far as even $cn/3$ tape squares from where it started, but the second u must be read while neither head gets *closer* than $2cn/3$ tape squares to where it started. During its subcomputation on v, therefore, it seems that M must somehow "copy" its representation of u across the intervening $cn/3$ tape squares. We show that this process has to involve a matching larger than δn.

For the sake of argument, suppose there is *not* a matching larger than δn. Then there must be a *maximal* matching of size only $m \leq \delta n$. We will select some correspondingly small "interface" through which a description of u must pass. That interface will involve some rarely crossed boundary at distance between $cn/3$ and $2cn/3$ from the heads' starting position, and some other rarely crossed boundaries that tightly isolate the $2m$ tape squares involved in the matching. Since there are $2cn/3 - cn/3$ candidates for the former, we can select one that is crossed only a constant number (bounded in terms of d and c) of times. We will refer to the tape squares on the respective sides of this selected *central boundary* as *close* and *far*. By the following purely combinatorial lemma, we can tightly isolate the matched tape squares with at most $4m$ additional boundaries, each of which is crossed only a constant number (bounded in terms of d, c, and our "tightness criterion" ϵ) of times.

Tight-Isolation Lemma. *Consider a finite sequence S of nonnegative numbers, the first and last of which are 0. Let some of the separating "commas" be specially designated—call them "semicolons". For each threshold $\ell \geq 0$, let S_ℓ be the subsequence consisting of the items[7] that are reachable from the semicolons via items that exceed ℓ (and that themselves exceed ℓ). Then, for each $\epsilon > 0$, there is some ℓ such that $\ell|S_\ell| < \epsilon \sum S$, where $\sum S$ denotes the sum of the entire sequence S and ℓ is bounded by some constant that depends only on ϵ.*

Proof. Let $T = \sum S$, and let $k = 2\lceil 2/\epsilon \rceil$. Since $2T/k \leq \epsilon T/2 < \epsilon T$, let us aim for $\ell|S_\ell| \leq 2T/k$. If no ℓ in $\{\, k^i \mid 0 \leq i \leq k \,\}$ were to work, then we would have

$$2T/k < k^i|S_{k^i}| < T$$

[7]Note that the number of such *items* can be small even if the number of semicolons is large. For ℓ large enough, in fact, $|S_\ell|$ will be 0.

for every i. But this would lead to the contradiction

$$\begin{aligned} T &> \sum_{i=0}^{k} k^i(|S_{k^i}| - |S_{k^{i+1}}|) \\ &> \sum_{i=0}^{k}(2T/k - T/k) \\ &= (k+1)T/k. \quad \square \end{aligned}$$

In our application, the numbers are the lengths of the crossing sequences associated with the boundaries between tape squares, their sum is at most dn, and the semicolons are the matched tape squares. We obtain our desired "isolation neighborhoods" from the at-most-$2m$ contiguous neighborhoods that comprise S_ℓ[8] by adding one item at each end of each neighborhood. (This might cause neighborhoods to combine.) This adds at most $4m$ items to S_ℓ and results in nonempty isolation neighborhoods whose boundary items are at most ℓ.

Actually, the picture is clearer if we select our central boundary *after* we select the isolation neighborhoods. Assuming ϵ and δ are chosen appropriately small, this lets us select a boundary not included in any of the isolation neighborhoods. (There are at most $4m + |S_\ell| \leq 2\delta n + \epsilon dn \leq cn/6$ boundaries (half the original number of candidates) to avoid.)

Finally, we use our suggested interface to give a description of u in terms of v that is too short—say shorter than $|u|/2 \approx cn/(6d)$. (We could substitute a description this short for u in x to contradict the incompressiblity of x.) We claim we can reconstruct u from M, v, the length of u, and the following information about the subcomputation of M while reading the v part of input uv:

(1) The sequence of all $\mathcal{O}(m)$ selected boundary locations.
(2) The sequence of all $\mathcal{O}(m)$ crossings of these selected boundaries, and their times (implicitly or explicitly including the corresponding input positions).
(3) The following information for each close-to-far crossing, and for the end of the subcomputation:
   - M's control state and head positions.
   - The full content of every isolation neighborhood.
(4) The following information for each crossing *out of* an isolation neighborhood:
   - The full content of that isolation neighborhood.
   - The full content of the isolation neighborhood in which the other head remains[9]—*provided* that there has been a new crossing into that neighborhood since the previous time such information was given for it.

To determine u, it suffices to reconstruct enough of M's configuration after its computation on input uv so that we can check which additional input string $2u'$ of length $1 + |u|$ leads to acceptance. The far tape contents suffice for this.

Our reconstruction strategy is mostly to simulate M step-by-step, starting with the first close-to-far crossing. Toward this end, we strive to maintain the contents of any currently scanned close isolation neighborhood and of the entire far side. We temporarily *suspend* step-by-step simulation whenever a head shifts onto a *close* tape square not in any isolation neighborhood, and we aim to *resume* suspended step-by-step simulation whenever a head shifts onto a *far* tape square not in any

[8]To include all the semicolons, some of these "contiguous neighborhoods" might have to be the *empty* neighborhoods of the semicolons.

[9]The other head must remain in *some* isolation neighborhood—otherwise, the matching could be enlarged.

isolation neighborhood. Because our matching is maximal, such a far tape square is *not* scanned at the time of suspension, and hence also not at any time before the desired resumption. It follows that the information for the needed updates is indeed available, so that resumption is indeed possible. Similarly, any necessary updates are possible if the step-by-step simulation happens to be suspended when the subcomputation ends.

It remains only to show that $|u|/2$ bits suffice for our description of u in terms of v. For each of the sequences in (1) and (2), the trick is to give only the first number explicitly, and then to give the sequence of successive differences. The length of this encoding is $\mathcal{O}(m\log(n/m)) = \mathcal{O}(n\log(1/\delta)/(1/\delta))$, which can be limited to a small fraction of $|u|/2 \approx cn/(6d)$ by choosing δ small enough. For (4), note that that the contents of each isolation neighborhood is given at most once for each of the ℓ crossings into and out of the neighborhood. For (3) and (4), therefore, straightforward encoding requires only $\mathcal{O}(\log n + \ell(m + |S_\ell|)) = \mathcal{O}(\log n + \ell\delta n + \epsilon dn)$ bits, where the implicit constant is bounded in terms of d and c. This can be limited to another small fraction of $|u|/2$ by choosing ϵ small enough, δ small enough, and n large enough. For the remaining information, M, $|u|$, and a description of this whole discussion, we need only $\mathcal{O}(\log n)$ bits, which can be limited to a final small fraction of $|u|/2$ by choosing n large enough. □

8. Further Discussion and Remaining Questions

In retrospect, our contribution has been a constraint on how a Turing machine with only two storage heads can recognize L in real time. Even if the two heads are on the same one-dimensional tape, such a Turing machine cannot recognize L in real time unless it violates the conclusion of Vitányi's Far-Out Lemma [Vi84 and above]. Only in the latter, which we only *cite*, do we ever really exploit an assumption that the two heads are on separate tapes.

Our result rules out general real-time simulation of a two-head tape unit using only a pair of single-head tapes. It remains to be checked whether the result readily extends to *probabilistic* simulation [PSSN90]. A more difficult extension might rule out simulation using *three* single-head tapes, yielding a tight result; but this would require a more difficult witness language. Perhaps allowing the "back" head of the defining two-head machine also to move and store random data, but much more slowly than the "front" head, would let us combine our arguments with those of Aanderaa [Aa74, PSS81, Pa82]. A slightly weaker possibility might be to show that *two* single-head tapes *and a pushdown store* do not suffice, and a slightly stronger one might be to show that even *three* single-head tapes and a pushdown store do not suffice.

It might be even more difficult to rule out general real-time simulation of a two-head one-dimensional tape unit using two or three *higher-dimensional* single-head tapes. In fact our language L *can* be recognized in real time by a Turing machine with just two such two-dimensional tapes—the idea is to strive to maintain the n bits of data within an $\mathcal{O}(\sqrt{n})$ radius on both tapes, along with $\mathcal{O}(\sqrt{n})$ strategically placed *copies* of the first $\mathcal{O}(\sqrt{n})$ bits, to serve as insurance *alternatives* at the same time that the array of their left ends provides a convenient area for temporary collection of data and for copying data between the tapes.

The implications for real-time simulation of one-dimensional tape units with *more than* two heads remain to be investigated. For example, how does a three-head tape compare with three single-head tapes or with one single-head tape and one two-head tape? How tight is the known bound of $4h - 4$ single-head tapes for real-time simulation of one h-head (one-dimensional) tape [LS81]? Perhaps the many-heads setting is the right one for a first proof that even an *extra* head is not enough to compensate for the loss of sharing; e.g., can a 1000-head tape be simulated in real time by 1001 single-head tapes, or by 1000 single-head tapes and a pushdown store?

Finally, does any of this lead to more general insight into the heads or tapes requirements for arbitrary computational tasks? I.e., when asked about some computational task, can we tightly estimate the structure of the sequential storage that suffices for the task?

References

[Aa74] S. O. Aanderaa, *On k-tape versus $(k-1)$-tape real time computation*, Complexity of Computation (SIAM-AMS Proceedings **7**) (R. M. Karp, ed.), American Mathematical Society, Providence, Rhode Island, 1974, pp. 75–96.

[CTPR85] F. R. K. Chung, R. E. Tarjan, W. J. Paul, and R. Reischuk, *Coding strings by pairs of strings*, SIAM Journal on Discrete Mathematics **6**, 3 (July, 1985), 445–461.

[DGPR84] P. Ďuriš, Z. Galil, W. J. Paul, and R. Reischuk, *Two nonlinear lower bounds for on-line computations*, Information and Control **60**, 1–3 (January–March, 1984), 1–11.

[FMR72] P. C. Fischer, A. R. Meyer, and A. L. Rosenberg, *Real-time simulation of multihead tape units*, Journal of the Association for Computing Machinery **19**, 4 (October, 1972), 590–607.

[Gr77] D. Yu. Grigoriev, *Imbedding theorems for Turing machines of different dimensions and Kolmogorov's algorithms*, Soviet Mathematics **18**, 3 (May–June, 1977), 588–592.

[He66] F. C. Hennie, *On-line Turing machine computations*, IEEE Transactions on Electronic Computers **EC-15**, 1 (February, 1966), 35–44.

[Ko65] A. N. Kolmogorov, *Three approaches to the quantitative definition of information*, Problems of Information Transmission **1**, 1 (January–March, 1965), 1–7.

[LLV92] M. Li, L. Longpré, and P. M. B. Vitányi, *The power of the queue*, SIAM Journal on Computing **21**, 4 (August, 1992), 697–712.

[LS81] B. L. Leong and J. I. Seiferas, *New real-time simulations of multihead tape units*, Journal of the Association for Computing Machinery **28**, 1 (January, 1981), 166–180.

[LV88] M. Li and P. M. B. Vitányi, *Tape versus queue and stacks: the lower bounds*, Information and Computation **78**, 1 (July, 1988), 56–85.

[LV93] M. Li and P. M. B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, New York, 1993.

[Ma85] W. Maass, *Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines*, Transactions of the American Mathematical Society **292**, 2 (December, 1985), 675–693.

[MSST90] W. Maass, G. Schnitger, E. Szemerédi, and G. Turán, *Two tapes versus one for off-line Turing machines*, Technical Report CS-90-30 (June, 1990), The Pennsylvania State University, University Park, Pennsylvania.

[Pa82] W. J. Paul, *On-line simulation of $k+1$ tapes by k tapes requires nonlinear time*, Information and Control **53**, 1–2 (April–May, 1982), 1–8.

[Pa84] W. J. Paul, *On heads versus tapes*, Theoretical Computer Science **28**, 1–2 (January, 1984), 1–12.

[PSS81] W. J. Paul, J. I. Seiferas, and J. Simon, *An information-theoretic approach to time bounds for on-line computation*, Journal of Computer and System Sciences **23**, 2 (October, 1981), 108–126.

[PSSN90] R. Paturi, J. I. Seiferas, J. Simon, and R. E. Newman-Wolfe, *Milking the Aanderaa argument*, Information and Computation **88**, 1 (September, 1990), 88–104.

[Ra63] M. O. Rabin, *Real time computation*, Israel Journal of Mathematics **1**, 4 (December, 1963), 203–211.

[ST89] W. Schnitzlein and H.-J. Stoß, *Linear-time simulation of multihead Turing machines*, Information and Computation **81**, 3 (June, 1989), 353–363.

[St70] H.-J. Stoß, *k-Band-Simulation von k-Kopf-Turing-Maschinen*, Computing **6**, 3 (1970), 309–317. (German)

[Vi84] P. M. B. Vitányi, *On two-tape real-time computation and queues*, Journal of Computer and System Sciences **29**, 3 (December, 1984), 303–311.

[ZL70] A. K. Zvonkin and L. A. Levin, *The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms*, Russian Mathematical Surveys **25**, 6 (November–December, 1970), 83–124.

Department of Computer Science, McMaster University, Hamilton, Ontario L8S 4K1, Canada

E-mail address: jiang@maccs.mcmaster.ca

Computer Science Department, University of Rochester, Rochester, New York 14627-0226, U. S. A.
E-mail address: joel@cs.rochester.edu

Centre for Mathematics and Computer Science (CWI), Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
E-mail address: paulv@cwi.nl