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

July 26, 1995; revised January 16, 1997

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 Turing machines commonly used and studied in computer science have separate tapes for input/output and for storage, so that we can conveniently study both storage as a dynamic resource and the more complex storage structures required for efficient implementation of practical algorithms [HS65]. Early researchers [MRF67] asked specifically whether two-head storage is more powerful if both heads are on the *same* one-dimensional storage tape than if they are on *separate* one-dimensional tapes, an issue of whether *shared* sequential storage is more powerful than *separate* sequential storage. Our result settles the longstanding conjecture that it *is*.

In a broader context, there are several natural structural parameters for the storage tapes of a Turing machine. These 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

1991 *Mathematics Subject Classification.* Primary 68Q05, 68Q30; Secondary 68P20, 94A17, 68Q25, 03D15.

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. We thank Zvi Galil, Ken Regan, and the anonymous referee for helpful comments on the manuscript.

An earlier version of this report appeared in the Proceedings of STOC '94, the Twenty-Sixth Annual ACM Symposium on the Theory of Computing, pp. 668–675.

be the case, although the proofs have been disproportionately difficult [Ra63, He66, Gr77, Aa74, PSS81, Pa82, DGPR84, Ma85, LV88, LLV92, MSST93, 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 [MRF67, 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 "tapes" that are *multidimensional*, 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 a multidimensional pattern (and hence the retrieval times) can be kept much smaller 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 ordinary *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 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 the particular language L in real time, since three *do* suffice [MRF67, 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].)

Historically, multihead tapes were introduced in Hartmanis and Stearns' seminal paper [HS65], which outlined a *linear*-time[1] simulation of an h-head tape, using some larger number of ordinary single-head tapes. Stoß [St70] later reduced the number of single-head tapes to just h. Noting the existence of an easy *real*-time simulation in the *other* direction, Bečvář [Be65] explicitly raised the question of real-time simulation of an h-head tape using only single-head tapes. Meyer, Rosenberg, and Fischer devised the first such simulation [MRF67]; and others later reduced the number of tapes [FMR72, Be74, LS81], ultimately to just $4h-4$. We are the first to show that *this* number *cannot* always be reduced to just h, although both the extra power of multihead tapes and the more-than-two-tape complexity of the particular language L have been longstanding conjectures [FMR72, LS81, Vi84, Pa84].

The reader may have noticed that we use, and even mix, two apparently separate terminologies, that of Turing machines and language recognition, on the one hand,

[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.)

and that of storage structures and simulations on the other. These are actually closely related, however, the choice being only one of emphasis.

The main components of a "Turing machine", of course, are a finite-state controller and some sort of simple storage (usually tape) of unlimited capacity. (We assume any input or output is through other, separate peripheral devices to which the controller has access. In our case of on-line language recognition,[1] input symbols are consumed and produced sequentially, so that one-way input and output tapes would do.) When our focus is on the nature of the storage structure, we view it separately as a sequential machine (or "abstract storage unit") in its own right—one that responds in an on-line manner to input commands from some finite alphabet with appropriate output responses from some other finite alphabet. For some input commands, such as "report the symbol scanned by the second head", the precise response is significant; but, for others, such as "shift the first head left" or "write the symbol 1 with the second head", it amounts to a mere acknowledgment. The on-line requirement in this setting is just that each command's response must precede the next command. If a machine, possibly with a completely different storage structure of its own, efficiently produces the same output behavior, then it (or even its storage structure, if that is our emphasis) can be said to efficiently *simulate the storage structure.* The strongest time-efficiency goal is *real time*, requiring that there be some single bound d on the time delay between command receipt and response.

Note that several abstract storage units can be combined into one. The composite command alphabet is a disjoint union of the individual command alphabets. Thus, for example, a pair of ordinary one-head one-dimensional tapes can be viewed as a single abstract storage unit that happens to have the same command and response alphabets as a two-head one-dimensional tape unit. (There is no command to our two-head unit to report whether the two head positions coincide. Such a command is easy to simulate, however, so the issue is not important.)

An abstract storage unit with sufficiently atomic commands needs only a binary response alphabet $\{0, 1\}$. The command to "report the symbol scanned by the second head", for example, can be replaced by some constant number of binary commands of the form "report whether the symbol scanned by the second head is a". Simulation of such a storage unit amounts to what is usually called an *on-line language recognition problem* (recall our earlier footnote[1]), with 1 signalling "acceptance so far" and 0 signalling "rejection so far". It follows, for example, that a pair of single-head tapes can simulate a two-head tape in real time if and only if every language recognized in real time by a Turing machine with a two-head tape ("two-head Turing machine", for short) can be recognized in real time by a Turing machine with two single-head tapes ("two-tape Turing machine", for short), an issue that we resolve negatively in this paper.

2. Tools

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, which we call "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. Nor does it depend on the computational significance of the steps designated as "distinguished". The use of computational terminology would only obscure the lemma's formulation and proof, so we avoid it.

An *overlap event* in a sequence $S = \ell_1, \dots, \ell_T$ (of "storage locations", in our application) is a pair (i, j) of indices with $1 \le i < j \le T$ and $\ell_i = \ell_j \notin \{\ell_{i+1}, \dots, \ell_{j-1}\}$

(i.e., "visit and soonest revisit"). If $\omega_t(S)$ denotes the number of such overlap events "straddling" t (i.e., with $i \le t$ but $j \not\le t$), then the sequence's *internal overlap*, $\omega(S)$, is $\max\{\, \omega_t(S) \mid 1 \le t < T \,\}$. The *relative* internal overlap is $\omega(S)/T$.

Here is an example: In the sequence

$$S = \text{cow}, \text{pig}, \text{horse}, \text{pig}, \text{sheep}, \text{horse}, \text{pig},$$

the overlap events are $(2,4)$, $(4,7)$, and $(3,6)$. For t from 1 up to 6, the respective values of $\omega_t(S)$ are 0, 1, 2, 2, 2, and 1; so $\omega(S)$ is 2, and the relative internal overlap is $2/7$.

(In our setting below, we apply these definitions to the sequence of storage locations shifted to on the successive 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.)

The lemma we now formulate guarantees the existence of a contiguous subsequence that has "small" relative internal overlap (quantified using ε), but that is itself still "long" (quantified using ε'). The lemma additionally guarantees that the subsequence can include a quite fair share of a set of "distinguished positions" of our choice in the original sequence.

(Without the latter guarantee, the lemma is just a simple corollary (and the essence) of any of the "overlap lemmas" formulated in the work we have already cited [Aa74, PSSN90]. The "designated positions" in our setting will be the items in the sequence that correspond to a large "matching"—a notion we define later, especially motivated by computations involving two heads.)

Overlap Lemma. *Consider any $\delta < 1$ and any $\varepsilon > 0$. Every sequence S (of length T, say) with "distinguished-position" density at least δ has a long contiguous subsequence, of length at least $\varepsilon' T$ for some constant $\varepsilon' > 0$ that depends only on δ and ε, with distinguished-position density still at least $\delta/4$, and with relative internal overlap less than ε.*

Proof. Without loss of generality, assume T is large in terms of δ and ε. It suffices to assume T is a power of 2, and to aim for a subsequence with distinguished-position density still at least $\delta/2$ (better than our general goal of $\delta/4$). (If the given sequence does *not* have length that is a power of 2, then we can discard an appropriate prefix or suffix of length less than half the total length, to obtain such a sequence with distinguished-position density still at least $\delta/2$, and then work with that sequence instead.) 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 count distinct overlap from

$$\begin{aligned}\lceil(\delta/2)2\rceil = \lceil\delta\rceil &\geq \delta/2 - 1/2 \text{ halves,}\\ \lceil(\delta/2)4\rceil - \lceil(\delta/2)2\rceil = \lceil 2\delta\rceil - \lceil\delta\rceil &\geq \delta - 1/2 \text{ quarters,}\\ \lceil(\delta/2)8\rceil - \lceil(\delta/2)4\rceil = \lceil 4\delta\rceil - \lceil 2\delta\rceil &\geq 2\delta - 1/2 \text{ eighths,}\\ \lceil(\delta/2)16\rceil - \lceil(\delta/2)8\rceil = \lceil 8\delta\rceil - \lceil 4\delta\rceil &\geq 4\delta - 1/2 \text{ sixteenths,}\\ &\text{etc.}\end{aligned}$$

Unless we find one of these sequences that has relative internal overlap less than ε, this accounts, at the ith level, for at least

$$(2^{i-2}\delta - 1/2)(\varepsilon T/2^i) = \varepsilon\delta T/4 - \varepsilon T/2^{i+1}$$

distinct overlap events, and hence for more than T distinct overlap events after $\lceil(4+2\varepsilon)/(\varepsilon\delta)\rceil$ levels. This is impossible, so we must find the desired low-overlap sequence at one of these levels. □

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. This general approach is discussed in more detail elsewhere [PSS81, LV93].

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 context, a typical choice being "$K(x) \geq |x| - \mathcal{O}(\log|x|)$". The latter implicitly involves some constant, however, the careful choice of which might be an additional source of confusion in our many-parameter context. A less typical but more absolute standard such as "$K(x) \geq |x| - \sqrt{|x|}$" completely avoids the introduction of yet another constant.

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 substrings 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! (Admittedly, the word "helpful" can be misleading here—the result says nothing at all about the relative *computational* complexity of generating the two strings from each other.) 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:

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

(The meaning of "$K(x|y \mid z)$" is the same as that of "$K(x|w)$", where w is some standard binary encoding of the pair (y, z).)

3. Proof 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 had 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 would be 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 [MRF67, FMR72, Be74] do follow this strategy, but we show that a machine with only two tapes will not be able to afford always to use one in this way for insurance: There will have to be a significant subcomputation in which the heads on *both* tapes "keep moving", even "essentially monotonically"—essentially as they would for straightforward "copying". 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 constant C, and consider any binary string x that is long in terms of C, and that is nearly incompressible.*[2] *Suppose $y = y_1 y_2 \ldots y_k$ (each y_i a binary string) is a "representation" with the following properties:*

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

(See Figure 1.) Then k is bounded by some constant that depends only on C.

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

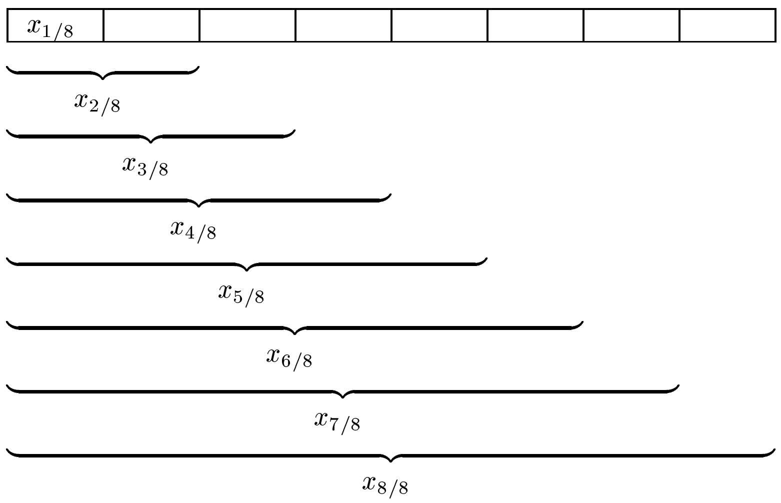
(1) A nearly incompressible input prefix x of length at least $|y|/C = \Theta(T/C)$ was read before the subcomputation.
(2) There is a parse of the subcomputation into a large number k of pieces so that each $x_{\ell/k}$ is coded in every contiguous sequence of ℓ pieces.
(3) k is (too) large in terms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

We accomplish these things by finding a subcomputation that has a spatially monotonic "matching" (a notion intended to model aspects of standard "copying") 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", and the second is to refine it in a suitable way. 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

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

The prefixes of x:



Some of the encoded prefix locations in y:

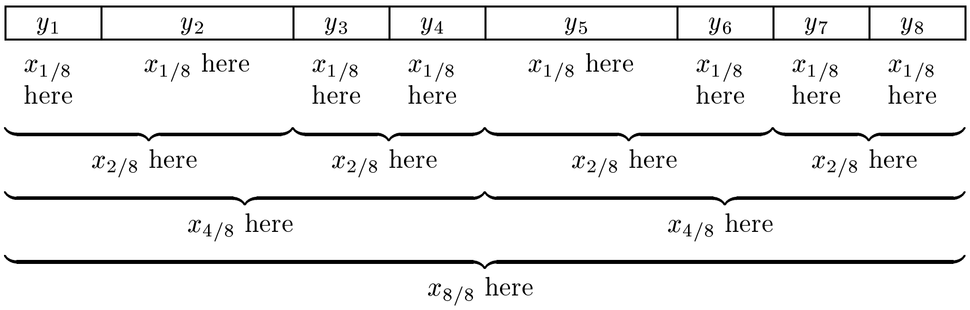


Figure 1. "Holography" example ($k = 8$)

tape square at more than one of the time instants. (So there is actually a separate one-to-one "matching" for each head, between the time instants and the tape squares scanned by that head at those times.) We prove the following lemma later on.

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.)

(Note that this lemma does *not* hold if the two heads can be on the same tape, since the straightforward real-time recognizer leaves one head completely stationary until the symbol 2 is reached.)

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. *If a multitape or multihead computation or subcomputation has a matching of length m and internal overlap at most p, then it has a monotonic submatching of length at least $\lceil m/\lceil 3p \rceil \rceil$ and minimum separation at least p.*

Proof. Just choose every $\lceil 3p \rceil$-th matching-time instant, starting with the first.

To see that this works, consider parsing the computation into $\lceil m/\lceil 3p \rceil \rceil$ subcomputations, each (except possibly the last) including a matching of length at least $3p$. Because the tapes are one-dimensional, each subcomputation involves a contiguous set, or "range", of at least $3p$ distinct tape squares visited by each head. Because of the overlap bound, these ranges overlap by no more than p, and each head begins and ends each subcomputation within p of opposite ends of its range for that subcomputation. (See Figure 2.) □

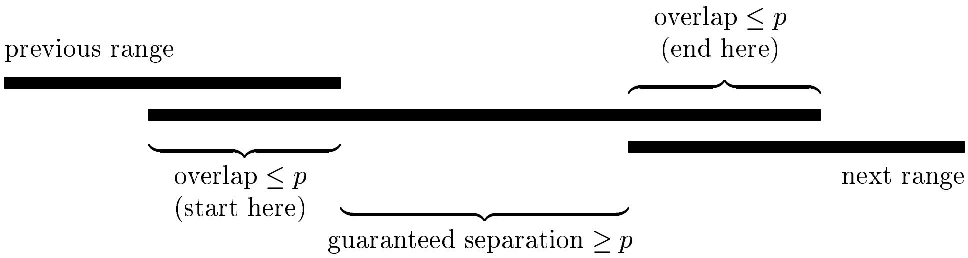


Figure 2. Range of a rightward head

4. Careful Argument

Now let us put together the whole argument, taking care to introduce the various "constants" in an appropriate order—M (and d), then δ, then ε, and finally ε'—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 sufficiently long incompressible input string $x \in \{0,1\}^*$ includes a matching of length at least $2\delta|x| + 2$. Let $\varepsilon > 0$ 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 at least δn. The number of *steps* in each of these subcomputations will lie between δn and dn. Therefore, the initial one will involve a prefix of x_0 of length at least $\delta n/d$, and the final one will have "match density" at least $(\delta n)/(dn) = \delta/d$.

Applying the Overlap Lemma to the final subcomputation above, we obtain a subcomputation of some length $T \geq \varepsilon' n$, 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 εT, and of length $2k+1$, where $2k+1$ is either $\lceil (\delta/(4d))T/\lceil 3\varepsilon T \rceil \rceil$ or $\lceil (\delta/(4d))T/\lceil 3\varepsilon T \rceil \rceil - 1$ (whichever is odd). If ε was chosen small, then k will be large. Note that $k\varepsilon$ is approximately equal to a constant $\delta/(24d)$ 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 $\lfloor k\varepsilon T/d\rfloor$. Since $\delta n/d$ exceeds this length (by a factor of at least about 24, since $T \le dn$), 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,

$$|y| \le DT \approx \frac{dD}{k\varepsilon}|x| \approx \frac{24d^2D}{\delta}|x|,$$

and that k is large (in fact, *too* large for the Anti-Holography Lemma) in terms of the constant $C = 24d^2D/\delta$, assuming we chose ε small enough.

To see that $x_{\ell/k}$ is coded by $y_{i+1}\ldots y_{i+\ell}$ (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 $2k+1$. (This is the middle match point in $y_{i+1}\ldots y_{i+\ell}$.) Since M must be able to check $x_{\ell/k}$ by reading only the information within distance $d\ell|x|/k \le \ell\varepsilon T$ of its heads, that prefix must be coded by that information. Since this distance in each direction is just ℓ 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 $2i+\ell+1-\ell$ and $2i+\ell+1+\ell$. Since $y_{i+1}\ldots y_{i+\ell}$ runs from the matching's time instant $2i+1$ to the matching's time instant $2i+2\ell+1$, therefore, it does code the desired prefix.

5. Proof of Anti-Holography Lemma

Recall from above the statement of the lemma (and Figure 1). 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 $x = x_1\ldots x_k$ and $|x_i| = n/k$ for every i.[5]

The key idea is obtain dramatically shortened descriptions of y in terms of prefixes of x. Since each prefix of x is coded in many disjoint substrings of y, we can use symmetry of information to efficiently abbreviate all those substrings in terms of the same one prefix of x. For each $j \le e$, and for $j = e-1$ in particular, this approach will yield

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

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

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

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

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.

[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.

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

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

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 $\big|d(u|v)\big| = K(u|v)$. Then

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

6. Proof of Large-Matching Lemma

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

Far-Out Lemma [Vi84][6]**.** *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"[7] *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] For a *sketch* of the proof, see the appendix below.

[7] 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. (Of course the obvious two-head machine, that does recognize our language in real time, does not satisfy the conclusion of *either* 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. (Without loss of generality, assume M's tape has a left end, where both heads start, and is unbounded to the right only.) 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. (See Figure 3.) 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.

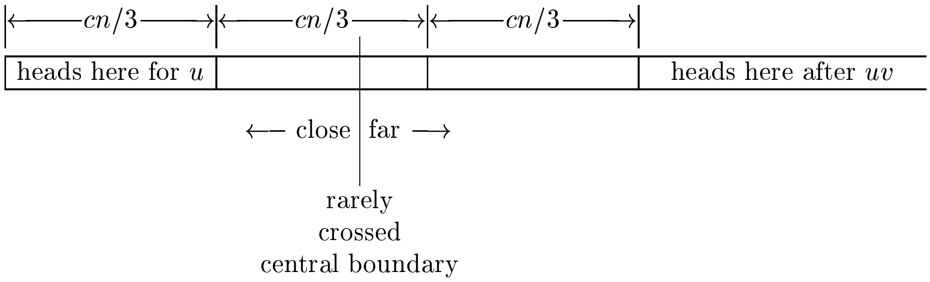


Figure 3. M's computation on uv

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 maximal matching. (Think of the latter tape squares as "shortcut terminals" through which we can "ship" information about u without a head carrying it. Maximality will insure that there is always at least one head on one of these terminals.) Since there are $2cn/3 - cn/3$ candidates for the former, we can select one that is crossed only a constant number of times (bounded in terms of d and c). We will refer to the tape squares on the left side of this selected *central boundary*, where the heads were when u was read, as *close*; and we will refer to the tape squares on the right side as *far*. (See Figure 3 again.) By the following purely combinatorial lemma, we can tightly isolate the $2m$ matched tape squares with at most $4m$ additional boundaries, each of which is crossed only a constant number of times (bounded in terms of d, c, and our "tightness criterion" ε).

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 (not necessarily contiguous) consisting of the items*[8] *that are reachable from the semicolons via items that exceed ℓ (and that themselves exceed ℓ). Then, for each $\varepsilon > 0$, there is some ℓ such that $\ell|S_\ell| < \varepsilon \sum S$, 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 $k = 2\lceil 2/\varepsilon \rceil$. Since $2T/k \leq \varepsilon T/2 < \varepsilon T$, 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

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

for every i. But this would lead to the contradiction

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

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[9] 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 or adjacent to any of the isolation neighborhoods. (There are at most $|S_\ell| + 4m \leq \varepsilon dn + 4\delta n \leq cn/6$ 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:

[8] 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.

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

- 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—*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. During step-by-step simulation, we 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 (at which time the other head must lie within some isolation neighborhood). We aim to *resume* suspended step-by-step simulation whenever a head shifts onto a *far* tape square not in any isolation neighborhood. Between a suspension and the desired resumption, the far tape contents are not scanned and do not change, except in the (at most) one scanned isolation neighborhood. Therefore, since the desired resumption is inaugurated by either a close-to-far crossing or a crossing out of that one isolation neighborhood, it follows that all the prerequisite information for resumption is indeed available at resumption time. Finally, since both heads are on the far side when the computation on uv ends, our step-by-step simulation is active at that point, and we do obtain the desired far tape contents.

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 (at most) ℓ crossings into and out of the neighborhood. For (3) and (4), therefore, straightforward encoding requires only $\mathcal{O}(\log n + \ell(m + |S_\ell|)) = \mathcal{O}(\log n + \ell\delta n + \varepsilon dn)$ 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. □

7. 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 (the first sublemma of) Vitányi's Far-Out Lemma (see Appendix below). Only in the latter 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 investigated whether the result extends to some notion of *probabilistic* real-time simulation [cf., PSSN90]. Another 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. Our particular 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? (Paul's results [Pa84] do answer such questions for tapes of higher dimension.) 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?

Appendix: A proof sketch for Vitányi's Far-Out Lemma

Suppose two-tape Turing machine M recognizes the language in real time. Without loss of generality, assume M's storage tape is only *semi*-infinite, and assume M writes only 0's and 1's. Let d be the delay of M.

Our ultimate goal is to show that both heads *simultaneously* range linearly far when the input is incompressible, but first we show that each one *separately* does so even when the input is just *nearly* incompressible. (The subsequent application is to significantly long *prefixes* of input strings that are not compressible at all.) It is only this part of the proof that requires the hypothesis that the two heads are on separate tapes. Like the proof of our Large-Matching Lemma above, this part is based on the "bottleneck" argument that Valiev [Va70] (and, independently, Meyer [Me71]) used to show that no *single*-tape Turing machine can accept the simpler language $\{\, x2x \mid x \in \{0,1\}^* \,\}$ in real time.

Suppose ε is small in terms of M and d, n is large in terms of all of the preceding, and x is of length n and nearly incompressible ($K(x) \geq n - \sqrt{n}$). We want to show that each head ranges farther than εn.

Suppose the head on one of the tapes, say the first, does *not* range farther than εn. Then the head on the *second* tape must certainly range farther than, say, $n/3$. (Otherwise, the total state after storage of x is a too-short description of x.) Let uvw be the parse of x with uv the shortest prefix of x that leaves M's second head at least $n/3$ tape squares out, and with $|u| = n/(9d)$, so that that same head gets no farther than $n/9$ tape squares out during input of u. On that head's tape, there must be a "bottleneck" boundary between $n/9$ and $2n/9$ tape squares out that gets crossed at most $9d$ times. (Cf., Figure 3 above.) Since all of u gets read when the second head is to the left of this bottleneck, it is possible to describe $x = uvw$ in terms of vw and the bottleneck's "crossing sequence", which should include, for each crossing, the step number and the "crossing state", which in turn should include the complete but relatively small contents of the *first* storage tape at the time of the crossing. The following information suffices:

(1) vw,
(2) a description of this discussion,
(3) a description of M,
(4) the value of n,
(5) the location of the bottleneck,
(6) the crossing sequence at the bottleneck.

If we provide vw as a literal suffix, then we can limit the length of this description to little more than $n-|u|$ bits, contradicting the near incompressibility of x. To recover u, we can use the information to determine enough of M's instantaneous description after reading uv (omitting from the instantaneous description only what is to the left of the bottleneck on the second tape) to then try each input continuation $2u'$ with $|u'| = n/(9d)$.

Finally, we return to our ultimate goal. Here is the idea: If the heads do *not* both go far out together, then they must *take turns*, so that some region gets crossed many times; abbreviate the symbols read while a head is in that region.

Suppose ε is small in terms of M and d (as above), ε_2 is small in terms of the preceding parameters (in particular, $\varepsilon_2 \ll \varepsilon$), ε_1 is small in terms of the now preceding parameters (in particular, $\varepsilon_1 \ll \varepsilon_2$), n is large in terms of all of the above, and x is of length n and incompressible. We want to show that both heads range farther than $\varepsilon_1 n$, simultaneously.

Suppose, to the contrary, that there is always at least one head within $\varepsilon_1 n$ tape squares of the origin. We count the crossings of the region from $\varepsilon_1 n$ to $\varepsilon_2 n$: It follows from our assumptions that a left-to-right crossing must occur between input symbol number $(d/\varepsilon)^i(\varepsilon_2 n/\varepsilon)$ and input symbol number $(d/\varepsilon)^{i+1}(\varepsilon_2 n/\varepsilon)$, for every i. (We use the fact that these input prefixes are themselves nearly incompressible.) By input symbol number n, therefore, the number of complete crossings (either direction) is at least $r = 2\log_{d/\varepsilon}(\varepsilon/\varepsilon_2)$ (which is large because ε_2 is so small).

There is a complication, however: There might also be *partial* crossings, involving fewer input symbols but additional overhead in the description we plan to give. To control this problem, we shrink the region slightly, replacing ε_1 and ε_2 with ε_1' and ε_2' from the first and last quarters, respectively, of the range $[\varepsilon_1, \varepsilon_2]$, chosen so that each of the boundaries $\varepsilon_1' n$ and $\varepsilon_2' n$ is crossed at most $R = 8d/\varepsilon_2$ times. This is possible, since $R(\varepsilon_2 - \varepsilon_1)n/4$ exceeds dn.

Finally, then, we formulate a description of the incompressible input that differs from the completely literal one as follows: We *eliminate* the input read while a head is in the range between $\varepsilon_1' n$ and $\varepsilon_2' n$, for a *savings* of at least $r(\varepsilon_2' - \varepsilon_1')n/d \geq r(\varepsilon_2 - \varepsilon_1)n/(2d)$ bits. We *add* descriptions of the crossing sequences at these two boundaries, including times, states, and the tape contents out to boundary $\varepsilon_1 n$ (the second term below), and also the full *final* contents of the tape squares *between* the two boundaries (the first term below), for a total *cost* of

$$\mathcal{O}((\varepsilon_2' - \varepsilon_1')n + R(\log n + \varepsilon_1 n)) = \mathcal{O}((\varepsilon_2 - \varepsilon_1)n + 8d(\log n + \varepsilon_1 n)/\varepsilon_2) = \mathcal{O}((\varepsilon_2 - \varepsilon_1)n)$$

bits, which can be kept significantly smaller than the savings. □

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.

[Be65] J. Bečvář, *Real-time and complexity problems in automata theory*, Kybernetika **1**, 6 (1965), 475–497.

[Be74] V. L. Bennison, *Saving tapes in the simulation of multihead Turing machines*, SIGACT News **6**, 2 (April, 1974), 23–26.

[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.

[HS65] J. Hartmanis and R. E. Stearns, *On the computational complexity of algorithms*, Transactions of the American Mathematical Society **117**, 5 (May, 1965), 285–306.

[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.

[Me71] A. R. Meyer, *An optimal time bound for a one tape on-line Turing machine computation*, unpublished manuscript (June, 1971, but earlier version already cited in 1967 [MRF67]).

[MRF67] A. R. Meyer, A. L. Rosenberg, and P. C. Fischer, *Turing machines with several read-write heads, preliminary report*, IEEE Conference Record of 1967 Eighth Annual Symposium on Switching and Automata Theory, IEEE Computer Society, Long Beach, California, 1967, pp. 117–127.

[MSST93] W. Maass, G. Schnitger, E. Szemerédi, and G. Turán, *Two tapes versus one for off-line Turing machines*, Computational Complexity **3**, 4 (1993), 392–401..

[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)

[Va70] M. K. Valiev, *Certain estimates of the time of computations on Turing machines with an input*, Cybernetics **6**, 6 (June, 1973), 734–741; translated from Kibernetika **6**, 6 (November–December, 1970), 26–32. (Russian)

[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