THEORY OF COMPUTING, Volume 18 (10), 2022, pp. 1–12
www.theoryofcomputing.org

Pseudorandom Bits and Lower Bounds for Randomized Turing Machines

Emanuele Viola*

Received August 5, 2019; Revised January 24, 2022; Published May 20, 2022

Abstract. We exhibit a pseudorandom generator with nearly quadratic stretch for randomized Turing machines, which have a one-way random tape and a two-way work tape. This is the first generator for this model. Its stretch is essentially the best possible given current lower bounds. We use the generator to prove a time lower bound in the above Turing machine model extended with a two-way read-only input tape. The lower bound is of the form $n^{1+\Omega(1)}$ and is for a function computable in linear time with two quantifier alternations. Previously lower bounds were not known even for functions computable in simply exponential time.

1

Turing's machines (TMs) are one of the most studied models of computation. More than fifty years ago, Hennie proved quadratic lower bounds [9] for solving simple problems on one-tape TMs. While this remains the best time lower bound for one-tape machines, lower bounds for stronger models have been established in many papers, including [23, 15, 16, 12, 24, 19, 28], some of which are discussed below.

*Supported by NSF CCF awards 1813930 and 2114116.

ACM Classification: F.2.3

AMS Classification: 68Q17

Key words and phrases: lower bound, Turing machine, pseudorandom generator, tape

 DOI: 10.4086/toc.2022.v018a010

In 1994, Impagliazzo, Nisan, and Wigderson constructed a pseudorandom generator [11] that *fools* one-tape TMs. More precisely they show how to efficiently map a seed of length $O(\sqrt{t})$ into a string of length t that cannot be distinguished from uniform by a one-tape Turing machine running in time t.

However, their result does not apply to *randomized* Turing machines, which can "toss coins." The model in [11] corresponds to randomized machines that begin by filling the tape with coin tosses in a one-way fashion, and then operate deterministically. By contrast, randomized machines can toss coins at any time during a computation. Equivalently, we can think of randomized machines as having an extra one-way tape with random bits on it. Note that in [11] the Turing machine receives the output of the generator on its (only) tape. A randomized machine instead receives it on a separate tape. This allows the machine to perform several tasks *in linear time*, such as copying the random bits on the work tape in reverse order, comparing the first half of the random bits with the second half, etc. All of these operations require *quadratic time* on a basic one-tape machine. And in fact, an instantiation of the [11] generator can be broken by a randomized TM (see Section 5).

In this paper we give the first generator for randomized TMs. Since there are several variants of TMs let us first define the model precisely and then state our result.

Definition 1.1. A randomized Turing machine (RTM) is a Turing machine with two tapes:
- A two-way, read-write work tape,
- and a one-way random tape.

Theorem 1.2. There are explicit pseudorandom generators with seed length $O(\sqrt{t}\log^3 t \log q)$ and error $\epsilon = (tq)^{-\sqrt{t}}$ that fool RTMs with q states running in time t. That is, no such RTM can tell whether its random tape is filled with uniform bits or with the output of the generator on a uniform random seed, except with advantage ϵ.

The seed length is, up to the polylogarithmic factors, the best possible short of proving super-quadratic lower bounds for TMs.

One motivation for constructing pseudorandom generators is proving lower bounds for stronger models. The RTM model above is the natural randomized extension of the basic one-tape machine model. Now we consider a different model to extend: Turing machines with a work tape and a two-way read-only input tape. This is one of the strongest models for which we can prove lower bounds. Time lower bounds of the form $n^{1+\Omega(1)}$ were shown in [16, 19, 28]. The question of extending these lower bounds to hold even for randomized machines was asked by several authors. In particular it was asked in [26, 27] (by the author), where lower bounds for an intermediate model are proved, and in a blog post [14] by Lipton.

In this paper we prove a lower bound for this randomized model. Again, let us first define the model.

Definition 1.3. An RTM2 is a Turing machine with three tapes:
- A two-way, read-write work tape,
- a one-way random tape, and
- a two-way, read-only input tape.

We prove a lower bound for a problem in $\Sigma_3\text{Time}(n)$ – linear time with two quantifier alternations. By standard completeness results the lower bound holds for $QSAT_3$ – the problem of deciding the validity of a formula with 2 quantifier alternations. This for example follows from the proof of Theorem 7 in [4]. Previously lower bounds were not known even for functions computable in exponential time $E = \text{Time}(2^{O(n)})$.

Theorem 1.4. There is a function computable in $\Sigma_3\text{Time}(n)$ which requires time $n^{1+\Omega(1)}$ on any RTM2 with bounded error probability.

The lower bound holds for general *two-sided* error. Moreover, as in previous work, see for example [19], it in fact holds even if the machine has *direct access* (a. k. a. random access) to the input. This means that there is a separate index tape and in one time step the machine can move the input head to the position written in binary on the index tape.

2 Overview of techniques

The proof of Theorem 1.2 relies on a new simulation of RTMs by a family of space-efficient *branching programs*, henceforth called *programs* for brevity. It is critical that each program in the family is allowed to read bits in a different order, so let us first define such programs.

Definition 2.1. A *(branching) program* with space s on t bits consists of a permutation π of the bits and a layered directed graph with $t+1$ layers. Each layer has 2^s nodes, except the first layer which has 1, and the last layer which has 2. Each node in layer $i \le t$ has two edges, labeled with 0 and 1, connecting to nodes in layer $i+1$. Nodes on layer $m+1$ are labeled with 0 and 1. On a t-bit input, the program follows the path corresponding to reading the input in a one-way fashion according to the permutation π: layer i reads input bit $\pi(i)$. It then outputs the label of the last node.

If M is an RTM we write $M(r)$ for the output of M when its random tape is initialized with r (and the work tape is blank). We can now state our simulation.

Theorem 2.2. Let M be an RTM running in time t and with q states. There is a family of $(tq)^{O(\sqrt{t})}$ programs P_α using space $O(\sqrt{t}\log tq)$ such that for any r:
- if $M(r) = 1$ then there is exactly one α such that $P_\alpha(r) = 1$,
- if $M(r) = 0$ then $P_\alpha(r) = 0$ for every α.

The string α in the simulation is used to guess short *crossing sequences* which partition the work tape into small blocks; the program then simulates the machine one block at the time. This idea goes back at least to the work by Maass and Schorr [16], see also the paper by van Melkebeek and Raz [19]. We show that it can be applied to RTMs if the random bits can be permuted. Here is where we use that the programs read bits in any order. This step is inspired by the works of Maass [15] and Kalyanasundaram and Schnitger [12] who give explicit functions (of the random bits) that are hard for RTMs. Our partition of the work tape has the additional property that it can be "verified" by the program, a property which is not apparent in previous simulations. This allows us to guarantee that $P_\alpha(r) = 1$ for at most one α, which is used in obtaining pseudorandom generators.

Given the simulation, we can use pseudorandom generators for programs that read bits in any order. However, we need a good dependence on the number of states and the error. Using a result from [7] (Corollary 20 and surrounding discussion) gives a pseudorandom generator for RTMs with seed length $t^{5/6+o(1)}$. This is sufficient for Theorem 1.4. Using the generator in the follow-up by Forbes and Kelley [5] we improve 5/6 to 1/2.

Proof of Theorem 1.2 from Theorem 2.2. The generator in [5], Corollary 4.3, fools branching programs on t bits using space s with error ϵ and seed length $O(\log(t2^s/\epsilon)\log^2 t) = O(s + \log(1/\epsilon))\log^2 t$. We set ϵ to $(tq)^{-O(\sqrt{t})}$ and recall $s = O(\sqrt{t}\log tq)$ to obtain seed length at most $O(\sqrt{t}\log^3 t \log q)$. To show correctness, let U be the uniform distribution and D the output distribution of the generator. We have

$$|\mathbb{E}[M(U)] - \mathbb{E}[M(D)]| = \left|\mathbb{E}\left[\sum_\alpha P_\alpha(U)\right] - \mathbb{E}\left[\sum_\alpha P_\alpha(D)\right]\right| \le (tq)^{O(\sqrt{t})}\epsilon = (tq)^{-\Omega(\sqrt{t})}$$

concluding the proof. □

The proof of Theorem 1.4 is in Section 4. It follows closely the proof of Theorem 6.2 in [27] which establishes a lower bound for a different model of Turing machines. Theorem 1.4 in this paper is to Theorem 6.2 in [27] as Theorem 1.2 in this paper is to the generator in [11]: Theorem 6.2 in [27] does not allow for a separate random-bit tape, but instead concerns Turing machines whose work tape is initially filled with random bits.

In turn, the proof from [27] relies on a number of other results in the literature, which are also required for the proof in this paper. We use the arguments mentioned earlier in [16, 19] to simulate TMs using little space and non-determinism. We then follow the lead of Diehl and van Melkebeek [4] who proved time-space lower bounds for randomized space-bounded computation. They used Nisan's pseudorandom generator [22] and the Sipser–Gacs–Lautemann [25, 13] simulation of BPP in the polynomial-time hierarchy. We use instead the pseudorandom generator given by Theorem 1.2. Here it is critical that the seed length has a good dependence on the number of states, since we will hard-wire the input to an RTM2 in the states. Turning to the Sipser–Gacs–Lautemann simulation, we note that this simulation (which has two quantifiers) is too slow for our purpose; we use the faster simulation in [27] (with three quantifiers). Finally, we rely on the machinery of time-space tradeoffs; for a survey of those see van Melkebeek [18] or Williams' thesis [29].

3 Proof of Theorem 2.2

The reader may want to refer to Table 1 for an example of an RTM computation and some of the parameters in this proof. For integers i and j we write $[i, j]$ for the set $\{i, i + 1, \dots, j\}$. Because the machine runs in time t it can only use $t + 1$ work cells. We identify these cells with $[1, t + 1]$. There are t *boundaries* between these cells. Boundary $i \in [t]$ is between work cell i and $i + 1$.

Tape cell	1	2	3	4	5	6	7	8	9
	$\star$1								
		H							
			H						
				H					
					H				
						H			
							$\star$3		
							$\star$3		
						H			
							H		
						H			
					H				
				H					
			H						
		H							
			H						
				H					
			H						
		H							
			$\star$2						
				H					
					$\star$3				
						H			
							H		
								H	
							H		
								H	
							H		
								H	
									$\star$3
block	1	2	2	2	3	3	3	3	3
	b_1			b_2					b_3

Table 1: Computation table of an RTM with 9 work tape cells reading 6 random bits. Each row corresponds to a different time stamp and shows the position of the head H on the work tape. The symbol $\star$ indicates when a random bit is read. We have three boundaries shown at the bottom. The “block” row shows the partition of work cells in blocks. The induced partition on the random-bit tape is 133233.

The string α. The string α contains $\sqrt{t}$ boundaries $b_1, b_2, \ldots, b_{\sqrt{t}} \in [t]$. These boundaries partition the work cells into $\sqrt{t}+1$ *blocks*, where block i are the cells $[b_{i-1}+1, b_i]$ with $b_0 = 0$ and $b_{\sqrt{t}+1} = t+1$. Each boundary b_i belongs to the set

$$B_i := [\sqrt{t}(i-1)+1, \sqrt{t}i].$$

This implies that each block has $\leq 2\sqrt{t}$ cells.

The string α also contains $\sqrt{t}$ *crossings* $c_1, c_2, \ldots, c_{\sqrt{t}}$. A crossing c is a tuple $(i \longrightarrow j, h, q)$ which means that the machine is crossing from block i to block j with the head on the random tape in position h and state q. The length of α is $\sqrt{t}O(\log tq)$ as required.

The property of P_α. We shall give programs P_α such that $P_\alpha(r) = 1$ if and only if

(1) $M(r) = 1$,

(2) for every i the boundary b_i is the one among B_i such that the computation $M(r)$ crosses it the least number of times, picking the smallest b_i in case of ties, and

(3) the crossings c_i are those induced by the b_i and the computation $M(r)$.

Concluding the proof assuming the property. Assuming we have P_α as above, we conclude the proof as follows. First, we need to verify that there are boundaries with respect to which the computation has $\leq \sqrt{t}$ crossings. This would be true even if we picked the b_i in a progression $b_i = \sqrt{t}(i-1)+j$: Because different values of j give disjoint crossings, and the total number of crossings is at most the computation time t, there is a value $j \in [1, \sqrt{t}]$ for which such b_i induce $\leq \sqrt{t}$ crossings. More so it holds for our definition of b_i.

Also, there cannot be $\alpha \neq \alpha'$ such that $P_\alpha(r) = P_{\alpha'}(r) = 1$. This is true because α is uniquely specified by the computation $M(r)$.

Designing the P_α. It remains to exhibit the programs P_α. The crossings $c_i = (k_i \longrightarrow k_{i+1}, h_i, q_i)$ induce a partition of the random tape in at most $\sqrt{t}+1$ intervals. Interval i is $[h_{i-1}+1, h_i]$ corresponding to the computation in block k_i from crossing $i-1$ to i, where crossing 0 is the beginning of the computation and $h_0 = 0$, and $h_{\sqrt{t}+1} = t$. We use the convention that $[x+1, x]$ is the empty interval, which may arise if the machine goes from one crossing to the next without reading random bits. These intervals can have any length.

The program simulates the machine one block at the time, reusing space across the blocks. The program will read the random bits in the following order. First it reads all the random bits in the intervals corresponding to work-tape block 1, then all the intervals corresponding to work-tape block 2, and so on.

For each block i, the program goes through the crossings involving block i in order and simulates the machine starting when a crossing enters block i until it leaves it. While doing so it verifies that the crossings leaving the block are correct assuming those entering it are. If not the program aborts and outputs zero. For this the program just needs memory for the work cells in one block, and the state of the machine, overall $O(\sqrt{t}+\log q)$ bits. Moreover, the program computes the number of crossings for each boundary in the block i it simulates. This takes additional $O(\sqrt{t}\log t)$ bits. At the end of the simulation of one block, the program can use this information to verify that the boundaries match the intended values. Recall that block i is $[b_{i-1}+1, b_i]$ and that each $b_i \in B_i$. The computation on block i can verify "one side" of the requirements for both b_{i-1} and b_i. Specifically, the program verifies that for each $b' \in B_{i-1}$ that is bigger than b_{i-1} the number of crossings at b' is at least as big as the number of crossings at b_{i-1}, and that for each $b' \in B_i$ that is smaller than b_i the number of crossings at b' is strictly bigger than the number of crossings at b_i. If one of these checks does not pass the program aborts and outputs zero. If all checks pass for every block then the b_i are correct.

Overall the space required by the program is $O(\sqrt{t}+\log q)+O(\sqrt{t}\log t) = O(\sqrt{t}\log tq)$.

4 Proof of Theorem 1.4

In this section we present the proof of Theorem 1.4. The proof involves several different computational models, so we need a definition.

Definition 4.1. We denote by $\mathrm{RTM2Time}(t)$ the class of problems that can be solved by an RTM2 machine running in time t.

We denote by $\mathrm{TiSp}(s,t)$ the class of problems that can be solved in time t and simultaneously space s. The precise machine model is not important here – we can assume a RAM machine.

For a complexity class C and a function b we denote by $\exists^b C$ the class of problems that can be solved in C with an $\exists$ quantifier on b bits: $\exists^b C = \{L' : \exists L \in C \text{ such that } x \in L' \Leftrightarrow \exists y \in \{0,1\}^{b(|x|)} : (x,y) \in L\}$. We use the corresponding definition for the $\forall$ and probabilistic BP quantifiers.

Towards a contradiction, assume that $\Sigma_3\mathrm{Time}(n) \subseteq \mathrm{RTM2Time}\left(n^{1+o(1)}\right)$. Let $m := n^2$. We derive the following contradiction as in [27]:

$$\begin{aligned}
\Sigma_3\mathrm{Time}(m) &\subseteq \mathrm{RTM2Time}\left(m^{1+o(1)}\right) && (4.1)\\
&\subseteq \mathrm{BP}^{m^{1-\Omega(1)}}\exists^{m^{1-\Omega(1)}}\mathrm{TiSp}\left(\mathrm{poly}(m), m^{1-\Omega(1)}\right) && (4.2)\\
&\subseteq \exists^{m^{1-\Omega(1)}}\forall^{m^{1-\Omega(1)}}\exists^{m^{1-\Omega(1)}}\mathrm{TiSp}\left(\mathrm{poly}(m), m^{1-\Omega(1)}\right) && (4.3)\\
&\subseteq \Sigma_{O(1)}\mathrm{Time}\left(m^{1-\Omega(1)}\right) && (4.4)\\
&\subseteq \Sigma_3\mathrm{Time}(o(m)) && (4.5)\\
&\quad\ \text{contradiction.} && (4.6)
\end{aligned}$$

The only inclusion that requires a new proof is (4.2). Before providing that proof we review why the other inclusions hold.

(4.1) is by padding.

(4.3) is by the time-efficient simulation of probabilistic time with three alternations, proved in [27] (Theorem 1.3).

(4.4) follows by the fact, usually attributed to [21], that one can trade alternations for time in sublinear-space computations. The required inclusion can be found in Section 3.2 in [17], or in [6].

(4.5) follows by noting that the assumption $\Sigma_3\mathrm{Time}(n) \subseteq \mathrm{RTM2Time}\left(n^{1+o(1)}\right)$ plus the simulation in [27] (Theorem 1.3) imply that $\Sigma_3\mathrm{Time}(n) \subseteq \Pi_3\mathrm{Time}\left(n^{1+o(1)}\right)$. This allows us to efficiently collapse the polynomial hierarchy at the third level.

The contradiction in (4.6) is with the time hierarchy, see for example Section 3.1 in [17].

4.1 Proving (4.2)

In this subsection we give the proof of inclusion (4.2). First we need that the generator is computable in linear space and polynomial time.

Claim 4.2. *Given an index i to an output bit, and a seed, we can compute the output bit i of the generator in Theorem 1.2 in space $t^{1-\Omega(1)}$ and simultaneously polynomial time.*

Proof. [Sketch] The generator in [5], Corollary 4.3, involves computing small-bias generators [20, 2] and bounded-independence generators [3, 1]. Those generators are in fact computable even in logarithmic space, see Theorem 14 in [8]. The rest of the computation amounts to taking bit-wise ANDs and XORs of strings, which can be done efficiently in small space. □

We can now prove Inclusion (4.2). We obtain the following variant of Claim 6.3 in [27]. For an RTM2 machine M we write $M(x;r)$ for the output of M on input x and random bits r. Recall that $m = m(n) = n^2$.

Claim 4.3. *Let M be an RTM2 machine with $O(1)$ states running in time $m^{1+o(1)}$. Let $G : \{0,1\}^{m^{1-\Omega(1)}} \to \{0,1\}^{m^{1+\Omega(1)}}$ be the generator from Claim 4.2. Then the language $\{(x;\sigma) : M(x;G(\sigma)) = 1\}$ is in $\exists^{m^{1-\Omega(1)}}\mathrm{TiSp}\left(\mathrm{poly}(m), m^{1-\Omega(1)}\right)$. In particular,*

$$\mathrm{RTM2Time}\left(m^{1+o(1)}\right) \subseteq \mathrm{BP}^{m^{1-\Omega(1)}}\exists^{m^{1-\Omega(1)}}\mathrm{TiSp}\left(\mathrm{poly}(m), m^{1-\Omega(1)}\right).$$

Proof. The "in particular" part follows from the first part by the correctness of the generator, using $m^{1-\Omega(1)}$ uniform bits as seed for the generator. Here note that we hardwire the input x by multiplying the number of states by $|x|$, and think of the resulting machine as an RTM.

To prove the first part, denote by τ the work tape of M. Let us divide τ in consecutive blocks where the first block is of size d and all the others are of size, say, $\sqrt{m}$. The parameter d will be specified later. Note d completely specifies this subdivision in blocks. For fixed d, similarly to before let us define a *crossing* to be a tuple

$$(a \to a', hi, hr, q)$$

where a is an index to a block in τ, $a' = a \pm 1$, hi is the position of the head on the input tape, hr is the position of the head on the random tape, and q is the state. A crossing $(a \to a', hi, hr, q)$

corresponds to M crossing the boundary of block a towards block a' with state q and input-tape and random-tape head positions hi and hr (the position of the head on the work tape is determined by a and a').

Since different values of $d \in \{0, 1, \ldots, \sqrt{m} - 1\}$ give rise to disjoint sets of blocks, there is $d < b$ such that the number of crossings induced by the computation $M(x; G(\sigma))$ is at most $t/\sqrt{m} \leq m^{0.51}$.

Simulation: We simulate the machine M as follows: First we guess a shift d and a sequence of $t/\sqrt{m}$ crossings. Then we check consistency of the computation one block at the time. For every block number i, we use $\sqrt{m}$ bits of space to simulate the machine on that block. First we initialize those bits to 0. Then we scan, in order, the list of guessed crossings. Whenever we encounter a crossing of the form $(a \to i, hi, hr, q)$ we do the following. We move the input head to hi and we initialize a counter r to hr. We simulate M starting in state q until it crosses the block boundary towards block a'. Whenever M asks for a random bit, we reply with the r-th output bit of $G(\sigma)$ and increase r by one. Then we look at the next crossing $(\alpha \to \alpha', hi', hr', q')$ in the guessed list and check that $\alpha = i$, $\alpha' = a'$ and $r = hr'$ and that hi' and q' match too. We continue in this way until the list is over. Finally, without loss of generality we can check that the last crossing has the accepting state of M.

Complexity: Note that each crossing can be specified by $O(\log m)$ bits, and therefore we guess at most $m^{0.51}O(\log m) = m^{1-\Omega(1)}$ bits total. The rest of the computation can be done simultaneously in time $\text{poly}(m)$ and space $m^{1-\Omega(1)}$. To see this, note that the generator is computable in these resources by Claim 4.2, while the rest of the simulation only uses space for one block, which takes $\sqrt{m}$ bits, plus $O(\log m)$ bits for various counters. □

5 RTM breaks INW

In this section we show that an instantiation of the generator in [11] for Turing machines does not fool RTMs. Let s be a parameter and G be a graph on vertex set $\{0, 1\}^s$. The basic step of the [11] generator is showing the pseudorandomness of the distribution (x, y, z), where (x, z) is a uniform edge in G and y is a uniform string in $\{0, 1\}^s$. The strength of the generator is related to the expansion of G. A simple graph with nearly maximum expansion is the one where (x, z) is an edge when the bit-wise XOR v of x and z has zero inner product modulo 2, that is $v_1v_2 \oplus v_3v_4 \oplus \cdots \oplus v_{s-1}v_s = 0$. This corresponds to the distribution $(x, y, x \oplus v)$ where x and y are uniform and v is a uniform string with zero inner product modulo 2. (It can be verified directly that the distribution v has *bias* $2^{-\Omega(s)}$ in the sense of Naor and Naor [20], and it is folklore that this equals the normalized second largest eigenvalue in G, which is a well-known algebraic measure of expansion, see [10].)

We now claim that an RTM can distinguish this distribution from uniform in time $O(s)$. The RTM first copies x on the work tape, then moves the work-tape head back to the beginning. Then by XORing corresponding random bits and work-tape bits the RTM can recover the bits of v and compute the inner product of v. Note we use one-way access to the random tape, but two-way access to the work tape.

By contrast, a one-tape TM cannot distinguish this distribution from uniform unless it runs in time $\Omega(s^2)$ [11].

Acknowledgments. We thank the anonymous referees for the detailed and helpful feedback.

References

[1] NOGA ALON, LÁSZLÓ BABAI, AND ALON ITAI: A fast and simple randomized algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986. [doi:10.1016/0196-6774(86)90019-2] 8

[2] NOGA ALON, ODED GOLDREICH, JOHAN HÅSTAD, AND RENÉ PERALTA: Simple constructions of almost k-wise independent random variables. *Random Struct. Algor.*, 3(3):289–304, 1992. [doi:10.1002/rsa.3240030308] 8

[3] BENNY CHOR AND ODED GOLDREICH: On the power of two-point based sampling. *J. Complexity*, 5(1):96–106, 1989. [doi:10.1016/0885-064X(89)90015-0] 8

[4] SCOTT DIEHL AND DIETER VAN MELKEBEEK: Time-space lower bounds for the polynomial-time hierarchy on randomized machines. *SIAM J. Comput.*, 36(3):563–594, 2006. [doi:10.1137/050642228] 3, 4

[5] MICHAEL A. FORBES AND ZANDER KELLEY: Pseudorandom generators for read-once branching programs, in any order. In *Proc. 59th FOCS*, pp. 946–955. IEEE Comp. Soc., 2018. [doi:10.1109/FOCS.2018.00093] 4, 8

[6] LANCE FORTNOW, RICHARD LIPTON, DIETER VAN MELKEBEEK, AND ANASTASIOS VIGLAS: Time-space lower bounds for satisfiability. *J. ACM*, 52(6):835–865, 2005. [doi:10.1145/1101821.1101822] 8

[7] ELAD HARAMATY, CHIN HO LEE, AND EMANUELE VIOLA: Bounded independence plus noise fools products. *SIAM J. Comput.*, 47(2):493–523, 2018. Preliminary version in CCC'17. [doi:10.1137/17M1129088] 4

[8] ALEXANDER HEALY AND EMANUELE VIOLA: Constant-depth circuits for arithmetic in finite fields of characteristic two. In *Proc. 23rd Symp. Theoret. Aspects of Comp. Sci. (STACS'06)*, pp. 672–683. Springer, 2006. [doi:10.1007/11672142_55] 8

[9] FREDERICK C. HENNIE: One-tape, off-line Turing machine computations. *Inform. Control*, 8(6):553–578, 1965. [doi:10.1016/S0019-9958(65)90399-2] 1

[10] SHLOMO HOORY, NATHAN LINIAL, AND AVI WIGDERSON: Expander graphs and their applications. *Bull. AMS*, 43(4):439–561, 2006. [doi:10.1090/S0273-0979-06-01126-8] 9

[11] RUSSELL IMPAGLIAZZO, NOAM NISAN, AND AVI WIGDERSON: Pseudorandomness for network algorithms. In *Proc. 26th STOC*, pp. 356–364. ACM Press, 1994. [doi:10.1145/195058.195190] 2, 4, 9, 10

[12] BALA KALYANASUNDARAM AND GEORG SCHNITGER: The probabilistic communication complexity of set intersection. *SIAM J. Discr. Math.*, 5(4):545–557, 1992. [doi:10.1137/0405044] 1, 3

[13] CLEMENS LAUTEMANN: BPP and the polynomial hierarchy. *Inform. Process. Lett.*, 17(4):215–217, 1983. [doi:10.1016/0020-0190(83)90044-3] 4

[14] RICHARD LIPTON: Old ideas and lower bounds on SAT. *Gödel's lost letter and P=NP*, 2010. Author's website. 2

[15] WOLFGANG MAASS: Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines. *Trans. AMS*, 292(2):675–693, 1985. [doi:10.2307/2000238] 1, 3

[16] WOLFGANG MAASS AND AMIR SCHORR: Speed-up of Turing machines with one work tape and a two-way input tape. *SIAM J. Comput.*, 16(1):195–202, 1987. [doi:10.1137/0216016] 1, 2, 3, 4

[17] DIETER VAN MELKEBEEK: Time-space lower bounds for NP-complete problems. *Current Trends Theor. Comp. Sci.*, pp. 265–291, 2004. [doi:10.1142/9789812562494_0015] 8

[18] DIETER VAN MELKEBEEK: A survey of lower bounds for satisfiability and related problems. *Found. Trends Theor. Comp. Sci.*, 2(3):197–303, 2006. [doi:10.1561/0400000012] 4

[19] DIETER VAN MELKEBEEK AND RAN RAZ: A time lower bound for satisfiability. *Theoret. Comput. Sci.*, 348(2–3):311–320, 2005. [doi:10.1016/j.tcs.2005.09.020] 1, 2, 3, 4

[20] JOSEPH NAOR AND MONI NAOR: Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993. [doi:10.1137/0222053] 8, 9

[21] VALERY A. NEPOMNYASHCHII: Rudimentary predicates and Turing calculations. *Dokl. Akad. Nauk SSSR (Russian)*, 195(2):282–284, 1970. MathNet.ru (Russian), English: Soviet Math. Dokl. 11(6), 1970, pp. 1462–1465. 8

[22] NOAM NISAN: Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. [doi:10.1007/BF01305237] 4

[23] WOLFGANG J. PAUL, NICHOLAS PIPPENGER, ENDRE SZEMERÉDI, AND WILLIAM T. TROTTER: On determinism versus non-determinism and related problems (preliminary version). In *Proc. 24th FOCS*, pp. 429–438. IEEE Comp. Soc., 1983. [doi:10.1109/SFCS.1983.39] 1

[24] RAHUL SANTHANAM: On separators, segregators and time versus space. In *Proc. 16th IEEE Conf. on Comput. Complexity (CCC'01)*, pp. 286–294. IEEE Comp. Soc., 2001. [doi:10.1109/CCC.2001.933895] 1

[25] MICHAEL SIPSER: A complexity theoretic approach to randomness. In *Proc. 15th STOC*, pp. 330–335. ACM Press, 1983. [doi:10.1145/800061.808762] 4

[26] EMANUELE VIOLA: *The Complexity of Hardness Amplification and Derandomization*. Ph. D. thesis, Harvard Univ., 2006. Author's website. 2

[27] EMANUELE VIOLA: On approximate majority and probabilistic time. *Comput. Complexity*, 18(3):337–375, 2009. Preliminary version in CCC'07. [doi:10.1007/s00037-009-0267-3] 2, 4, 7, 8

[28] RYAN WILLIAMS: Inductive time-space lower bounds for SAT and related problems. *Comput. Complexity*, 15(4):433–470, 2006. [doi:10.1007/s00037-007-0221-1] 1, 2

[29] RYAN WILLIAMS: *Algorithms and Resource Requirements for Fundamental Problems*. Ph. D. thesis, Carnegie Mellon Univ., 2007. Author's website. 4

AUTHOR

Emanuele Viola
Associate professor
Khoury College of Computer Sciences
Northeastern University
Boston, MA
viola@ccs.neu.edu
http://www.ccs.neu.edu/~viola

ABOUT THE AUTHOR

EMANUELE VIOLA, called "Manu" by friends and colleagues, has been thinking about these problems (on and off) for at least 15 years.