Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 42 (2005)

Linear Advice for Randomized Logarithmic Space

Lance Fortnow
Department of Computer Science
University of Chicago
Chicago, IL 60637
fortnow@cs.uchicago.edu

Adam R. Klivans*
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712
klivans@cs.utexas.edu

May 5, 2005

Abstract

We show that $\mathsf{RL} \subseteq \mathsf{L}/O(n)$, i.e., any language computable in randomized logarithmic space can be computed in deterministic logarithmic space with a linear amount of non-uniform advice. To prove our result we use an ultra-low space walk on the Gabber-Galil expander graph due to Gutfreund and Viola.

*Work done while at TTI-Chicago.

ISSN 1433-8092

1 Introduction

The question of whether RL, randomized logarithmic space, can be simulated in L, deterministic logarithmic space, remains a central challenge in complexity-theory. The best known deterministic simulation of randomized logarithmic space is due to Saks and Zhou [16] who, building on seminal work due to Nisan [13], proved that $\mathsf{BPL} \subseteq \mathsf{L}^{3/2}$. Recently in a breakthrough result, Reingold [14] proved that the s-t connectivity problem on undirected graphs could be solved in L; this implies SL, the symmetric analogue of NL, equals L. The possibility of extending his techniques to prove $\mathsf{RL} = \mathsf{L}$ has subsequently been investigated by Reingold, Trevisan and Vadhan [15].

1.1 Randomness and Non-Uniformity

The relationship between randomness and non-uniformity is a topic of fundamental importance in complexity theory. Derandomizing complexity classes frequently involves a consideration of the smallest non-uniform complexity class or circuit family which contains a particular randomized class. Adleman's well known result on BPP [1], for example, shows that BPP can be computed by polynomial-size circuits. Goldreich and Wigderson [7] have recently proved an interesting relationship between the non-uniform complexity of RL and the existence of deterministic algorithms for RL which work on almost every input. More specifically they showed that if RL is computable in log-space with $o(n)$ bits of non-uniform advice and furthermore most of the advice strings are "good," (i.e. result in a correct simulation of the RL machine by the advice-taking deterministic logarithmic-space machine) then there exists a deterministic log-space simulation of RL which errs on a $o(1)$ fraction of inputs for every input length. In other words, finding a log-space simulation with $o(n)$ bits of non-uniform advice is a step towards showing that RL is "almost" in L.

1.2 Our Results

We prove that every language in RL can be computed in L with $O(n)$ bits of additional non-uniform advice:

Theorem 1 *Every language in* RL *can be computed by a deterministic, log-space Turing machine which receives $O(n)$ bits of non-uniform advice on a two-way read-only input tape.*

In Section 3, we state as a corollary of Nisan's well-known pseudo-random generator for space bounded computation the inclusion $\mathsf{RL} \subseteq \mathsf{L}/O(n \log n)$. What is more difficult is to show that $\mathsf{RL} \subseteq \mathsf{L}/O(n)$. To do this we use a non-standard, space-efficient walk on an expander graph when the edge labels are presented on a two-way read-only input tape. Such a walk was used by Gutfreund and Viola [8] in the context of building pseudorandom generators computable in AC^0:

Theorem 2 (Gutfreund-Viola) *There exists an $O(\log(n))$-space algorithm for taking a walk of length $O(n)$ on a particular constant-degree expander graph with $2^{O(n)}$ nodes if the algorithm has access to an initial vertex and edge labels describing the walk via a two-way read-only advice tape.*

We present a proof of Theorem 2 for completeness. Note that a naive implementation of a walk on a graph of size $2^{O(n)}$ would require $O(n)$ space just to remember the current vertex label. The main tool for using even less space is a Gabber-Galil graph [6] in conjunction with log-space algorithms for converting to and from Chinese Remainder Representations of integers [4].

We can then apply the above walk on an expander graph to amplify the success probability of the RL algorithm using only $O(n)$ random bits and $O(\log n)$ space. Using Adleman's trick [1] we can conclude that there must exist a good advice string of length $O(n)$ which works for all inputs.

1.3 Related Work

Bar-Yossef, Goldreich, and Wigderson [3] initiated a study of on-line, space-efficient generation of walks on expander graphs. Their work shows how to amplify a space S algorithm using r random bits with error probability $1/3$ to an $O(kS)$-space algorithm that uses $r+O(k)$ random bits and has error probability $\epsilon^{\Omega(k)}$ for any constant $\epsilon > 0$. Note, however, that we need $\epsilon < 2^{-n}$ and hence must take $k \geq n$. The space of their resulting algorithm will then be $\Omega(nS)$, which is much too large for our application here. Our savings comes from the fact that we have an initial vertex and the edge labels of a particular walk on an expander graph on an advice tape– we thus do not need to remember an initial vertex or the edge labels.

2 Preliminaries

Karp and Lipton [10] give a general definition of complexity classes with advice.

Definition 3 (Karp-Lipton) *For a complexity class $\mathcal{C}$, the class $\mathcal{C}/f(n)$ is the set of languages A such that there exists a language B in $\mathcal{C}$ and a sequence of strings $a_0, a_1, \ldots$, with $|a_n| = f(n)$ and for all x in A,*

$$x \in A \Leftrightarrow (x, a_{|x|}) \in B.$$

Our main result shows that for every language A in RL there is a constant c such that A is in L/cn. By Definition 3, the logarithmic space machine accepting B will have access to the advice as part of the 2-way read-only input tape.

To achieve our result we start with pseudorandom generators for space-bounded computations.

Definition 4 *A generator $G : \{0,1\}^m \to \{0,1\}^n$ is called a* pseudorandom generator for space(S) with parameter ϵ *if for every randomized space(S) algorithm A and every input y we have*

$$|\Pr(A(y) \textit{ accepts}) - \Pr(A(G(x)) \textit{ accepts})| \leq \epsilon$$

where y is chosen uniformly at random in $\{0,1\}^n$ and x uniformly in $\{0,1\}^m$.

We will also use expander graphs.

Definition 5 *An graph $G = (V, E)$ is an ϵ-expander if there exists a constant $\epsilon > 0$ such that for all U such that $|U| \leq |V|/2$, $|U \cup N(U)| \geq (1+\epsilon)|U|$ where $N(U)$ is the neighborhood of U. A set of graphs $\{G_1, G_2, \ldots\}$ is a family of constant-degree expander graphs if there are fixed constants d and ϵ such that for all n, G_n has n vertices, the degree of every vertex of G_n is d and G_n is an ϵ-expander.*

We will use the now well-known fact that explicitly constructible constant-degree expander graphs can be used to reduce the error of randomized decision procedures:

Theorem 6 ([5, 9]) *Let $L \in \mathsf{BPP}$ be decided by a probabilistic turing machine M using $r(n) = n^{O(1)}$ random bits. Then there exists a probabilistic polynomial-time algorithm for deciding L using $O(r(n) + t)$ random bits with error probability 2^{-t}. The algorithm chooses a random vertex of an expander graph and walks for t steps substituting the labels of the vertices in place of the truly random bits M would have used.*

For details on the history, constructions, spectral theory, and applications of expander graphs see Motwani and Raghavan [12] or the lecture notes for a course by Nati Linial and Avi Wigderson [11].

3 Starting Point: Nisan's Generator

We will use Nisan's well-known generator for space bounded computation [13] as the starting point for the proof of our main result:

Theorem 7 (Nisan) *For any R and S there exists a pseudorandom generator*

$$G : \{0,1\}^{O(S \log(R/S))} \to \{0,1\}^R$$

for space(S) with error parameter 2^{-S}. Furthermore, if the seed is written on a two-way read-only tape, the ith bit of the output of the generator can be computed in $O(S)$ space.

Corollary 8 *For any language A in RL there is a probabilistic algorithm solving L with one-sided error $1/n$ using logarithmic space and $O(\log^2 n)$ random bits on a 2-way read-only tape.*

Fix an RL language A. To prove the existence of a good advice string of length $O(n \log n)$ for A we first apply Corollary 8 to get a randomized algorithm using $O(\log^2 n)$ random bits that for any fixed input x in A results in an answer which is correct with probability at least $1 - 1/n$ and always rejects on inputs not in A.

Running the algorithm $2n/(\log n)$ times independently results in a randomized algorithm with error probability strictly less than 2^{-n}. Thus, by a union bound over all inputs, there must exist a sequence of n seeds to G which results in a correct classification of *any* input x.

Hence the total advice string is of length $O(n \log n)$. Now assume that a log-space machine is given access to this advice string on a read-only (multiple access) input tape. Since the ith bit of Nisan's generator is computable in log-space, we can carry out Adleman's trick in logarithmic space. Thus, as a corollary to Nisan's generator we have the following:

Corollary 9 $\mathsf{RL} \subseteq \mathsf{L}/O(n \log n)$.

We wish to reduce the size of the advice to $O(n)$ bits. Note that any attempt to construct an advice string using only Adleman's trick cannot hope to achieve an advice string of length less than n. This is because using k bits of randomness in a black-box fashion can only drive the error probability of the algorithm down to 2^{-k} (without derandomizing the algorithm altogether).

4 Space-Efficient Walks on Gabber-Galil Graphs

Consider a language $L \in \mathsf{RL}$ and its associated randomized Turing machine M. A now standard approach for derandomizing algorithms [5, 9] is to "re-cycle" random bits via a walk on a suitable expander graph. For example, if we have a pseudorandom generator G with seed length s, and a suitable constant degree expander graph E with 2^s vertices, we can use the following randomness-efficient algorithm to compute L:

- Associate each vertex of the graph with a seed of G (note each vertex has a label of length s).
- Use s random bits to choose an initial vertex of E and walk randomly (by choosing t random edge labels) for t more steps to select t more seeds for G.
- Simulate L using the output of the generator G instead of truly random bits. Do this t times independently and accept if any of the t simulations result in accept.

This algorithm uses $O(s+t)$ random bits and requires space $O(s)$ (plus the space required to compute a neighbor of the current vertex). Applying Theorem 6 we see that the error probability of this algorithm is at most 2^{-t}.

We would like to use the above algorithm where the vertices of an expander graph correspond to seeds for Nisan's pseudorandom generator. Unfortunately, to simulate languages in RL using the output of Nisan's generator the seed must be of size $\Omega(\log^2 n)$, which means that the above algorithm will use at least $\Omega(\log^2 n)$ space just to keep track of the current vertex. As such, we will have to use a very space-efficient method for traversing an expander graph due to Gutfreund and Viola [8]. The walk uses a result due to Gabber and Galil [6]:

Theorem 10 (Gabber-Galil) *Let Z_m be the integers modulo m and let E be a graph with vertex set $Z_m \times Z_m$ and edge relations $(x, y) \Rightarrow (x, y) \cup (x, x + y) \cup (x, x + y + 1) \cup (x + y, y) \cup (x + y + 1, y)$. (i.e. each vertex is connected to 5 other vertices via the above simple arithmetic operations mod m). Then E is an expander graph with m^2 vertices and degree* 5.

We know present the walk due to Gutfreund and Viola [8] and for completeness we include a proof of correctness.

Let $m = \Pi_{i=1}^{k} p_i$ where each p_i is a distinct prime. Then we can view each vertex in the above expander graph via the Chinese remainder theorem as $(a_1, \ldots, a_k) \times (b_1, \ldots, b_k)$ where each $a_i, b_i \in Z_{p_i}$. Since we are interested in representing seeds of length $O(\log^2 n)$, we can think of m as the product of $O(\log n)$ primes, each of bit-length $O(\log n)$.

The idea is that to walk on this graph, we need only keep track of an index, and two residues $a, b \in Z_p$ for p a prime of bit-length at most $O(\log n)$. That is to say, we will store a *residue* of the Chinese Remainder Representation of m, rather than the integer m itself, and we can update the current residue during each step of the walk in log-space:

Lemma 11 *Let E be the Gabber-Galil graph $Z_m \times Z_m$ as above where $m = \Pi_{i=1}^{a \log n} p_i$, the product of $a \log n$ primes (for sufficiently large a) each of bit length at most $O(\log n)$. There exists an $O(\log n)$ space algorithm W such that on input s, a starting vertex of E, a sequence of edge labels $t = (t_1, \ldots, t_\ell)$, and $1 \leq i \leq a \log n$, a residue index, W outputs the ith residues of the two integers representing the vertex reached by starting at s and traversing the edge labels indicated by t.*

Proof: Assume that s can be represented via the Chinese Remainder Theorem as $(a_1, \ldots, a_\ell) \times (b_1, \ldots, a_\ell)$ where each a_i, b_i has bit-length $O(\log n)$. The edge relations of the Gabber-Galil graph involve only an addition and may be carried out component-wise. For example, if we are currently storing a_i and b_i and the next edge label t_j is equal to 1, then the new components we store are a_i and $a_i + b_i \mod p_i$. Hence we need only remember the two current residues, a prime p_i, and the number of steps we have already taken. This requires $O(\log n)$ space. ▮

Thus, although we cannot store an entire vertex of our expander graph, we can compute residues of vertices explored by a random walk on the graph. Unfortunately, Nisan's generator requires a seed described by the original representation of vertices of the Gabber-Galil graph. As such we will require a space-efficient routine for computing the ith bit of an integer m if we are only given access to its residues modulo distinct primes. Following Gutfreund and Viola, we can apply a recent space-efficient algorithm for coverting to and from the Chinese Remainder Representation due to Chiu, Davida and Litow [4]:

Theorem 12 (Chiu-Davida-Litow) *Let $a_1, \ldots, a_\ell$ be the Chinese Remainder Representation of an integer m with respect to primes $p_1, \ldots, p_\ell$. There exists a log-space algorithm D such that on input $a_1, \ldots, a_\ell$, primes $p_1, \ldots, p_\ell$, and index i, D outputs the ith bit of the binary representation of the integer m.*

For a further discussion of space-efficient, uniform algorithms for arithmetic operations such as division and converting from the Chinese Remainder Representation see Allender et al. [2].

5 Putting It All Together

We can combine these space-efficient tools to prove our main result:

Theorem 13 $\mathsf{RL} \subseteq \mathsf{L}/O(n)$

Proof:

From the discussion at the beginning of Section 4, we know that for polynomial-time advice taking Turing machines simulating RL, for every input length n there must exist a good advice string $A(n)$ of length $O(n)$ consisting of an initial vertex on a suitable Gabber-Galil expander graph with $n^{O(\log n)}$ vertices and a sequence of $2n$ edge labels. Let us assume that the vertices of the graph equal $Z_m \times Z_m$ where m is a product of $O(\log n)$ primes $p_1, \ldots, p_k$ each of bit length $O(\log n)$ (such primes are guaranteed to exist by the Prime Number Theorem). Augment our advice string $A(n)$ with primes $p_1, \ldots, p_k$. Call this advice string $A'(n)$.

Our claim is that $A'(n)$ is a good advice string for a log-space Turing machine M which computes the above simulation of RL using the following procedures:

1. Simulate the RL machine $2n$ times using the output from Nisan's generator on seeds corresponding to the vertices reached by the walk given on the advice string. If any simulation results in accept then accept.

2. When Nisan's generator requires a bit from the seed, walk on the Gabber-Galil graph to obtain the ith bit of the binary representation of that vertex.

3. When the ith bit of a vertex from the graph is required, apply the Chinese Remainder Representation algorithm given from Theorem 12 and use Lemma 11 to obtain any residue required by the CRR algorithm of Theorem 12.

Note that each procedure can be performed in log-space, and the entire algorithm is a composition of three log-space procedures. Therefore, the entire simulation carried out by M can be performed in log-space. We use the fact that the initial vertex and edge labels are written on a tape in a critical way: whenever the algorithm needs a bit of the ith vertex label from the walk on the graph, we can move our tape head back to the initial vertex and walk from the beginning for i steps.

▮

We note here that our space-efficient walk can be used on graphs with 2^n vertices (rather than $O(n^{\log n})$ vertices), as we can represent each vertex as the product of more than just $O(\log n)$ primes.

6 Connectivity for Expander Graphs in NC^1?

We conclude with the following question: is the undirected connectivity problem for expander graphs in NC^1? We can give an NC^1 algorithm for walking on the Gabber-Galil graphs used for our main theorem (assuming the edge labels are on an input tape). Is it possible that for a general expander graph connectivity is in RNC^1? Such a result would lead to the intriguing possibility that more general connectivity problems are not only in L but in NC^1.

7 Acknowledgments

Thanks to Jaikumar Radhakrishnan and Rahul Santhanam for helpful discussions. Thanks to Emanuele Viola for making us aware of his work with Danny Gutfreund.

References

[1] L. Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science*, pages 75–83. IEEE, New York, 1978.

[2] E. Allender, D. Barrington, and W. Hesse. Uniform circuits for division: Consequences and problems. In *Annual IEEE Conference on Computational Complexity*, volume 16, 2001.

[3] Z. Bar-Yossef, O. Goldreich, and A. Wigderson. Deterministic amplification of space-bounded probabilistic algorithms. In *Proceedings of the 14th IEEE Conference on Computational Complexity*, pages 188–199. IEEE, New York, 1999.

[4] A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC^1. *RAIRO - Theoretical Informatics and Applications*, 35:259–275, 2001.

[5] A. Cohen and A. Wigderson. Dispensers, deterministic amplification, and weak random sources (extended abstract). In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 14–25, Research Triangle Park, NC, October 1989. IEEE Computer Society Press.

[6] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, June 1981.

[7] O. Goldreich and A. Wigderson. Derandomization that is rarely wrong from short advice that is typically good. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, volume 2483 of *Lecture Notes in Computer Science*, pages 209–223. Springer, Berlin, 2002.

[8] D. Gutfreund and E. Viola. Fooling parity tests with parity gates. In *Proceedings of the 8th International Workshop on Randomization and Computation (RANDOM)*. 2004.

[9] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 248–253. IEEE, New York, 1989.

[10] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on the Theory of Computing*, pages 302–309. ACM, New York, 1980.

[11] N. Linial and A. Wigderson. Lecture notes on expander graphs and their applications. http://www.math.ias.edu/~boaz/ExpanderCourse/index.html, 2002.

[12] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

[13] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[14] O. Reingold. Undirected st-connectivity in log-space. In *Proceedings of the 36th ACM Symposium on the Theory of Computing*. ACM, New York, 2005. To appear.

[15] O. Reingold, L. Trevisan, and S. Vadhan. Pseudorandom walks in biregular graphs and the RL vs. L problem. Technical Report TR05-022, Electronic Colloquium on Computational Complexity, 2005.

[16] M. Saks and S. Zhou. $\mathrm{BP}_H\mathrm{SPACE(S)} \subseteq \mathrm{DPSPACE(S^{3/2})}$. *Journal of Computer and System Sciences*, 58(2):376–403, April 1999.