$RL \subseteq SC$

Noam Nisan *

March 1, 1995

Abstract

We show that any randomized Logspace algorithm (running in polynomial time, with bounded two-sided error) can be simulated deterministically in polynomial time and $O(\log^2 n)$ space. This puts RL in SC, "Steve's Class". In particular, we get a polynomial time $O(\log^2 n)$ space algorithm for the st-connectivity problem on undirected graphs.

1 Introduction

Given a (directed) graph G and two vertices in it s and t, the st-connectivity (or reachability) problem is to decide whether there exists a directed path from s to t in G. The st-connectivity problem in directed graphs is one of the most basic problems in computer science. Using BFS or DFS, st-connectivity can be solved in polynomial (actually even linear) time and linear space. On the other hand, Savitch's [Sav70] theorem gives a $O(\log^2 n)$-space algorithm for it, but which requires super-polynomial time. It is a long standing open problem whether there exists an algorithm for st-connectivity that combines both features: runs in polynomial time and poly-logarithmic space.

For st-connectivity in *undirected* graphs two more types of algorithms are known. In [AKL*79] a *randomized* Logspace (and polynomial time) algorithm is given. A zero-error version of this type of algorithm is given in [BCD*89]. As for deterministic algorithms, Barnes and Ruzzo [BR91] recently presented an algorithm for undirected st-connectivity that runs in polynomial time and n^ϵ space (for any fixed $\epsilon > 0$).

*Dept. of CS, Hebrew University, Jerusalem. Supported by BSF 89-00126 and by a Wolfson research award.

In this paper we present further progress on this problem, obtaining an exponential improvement (in space).

Theorem 1: *There exists a deterministic algorithm for the undirected st-connectivity problem that runs in polynomial time and $O(\log^2 n)$ space.*

Unfortunately, the running time of the algorithm is a high order polynomial (something like n^{45}). We do not know how to improve the running time significantly.

The algorithm is obtained by derandomizing the randomized algorithm of [AKL*79]. In fact, the derandomization result is very general. We prove that any problem that can be computed in randomized Logspace has a *deterministic* algorithm that runs in polynomial time and $O(\log^2 n)$ space. In terms of complexity classes we can state our result as:

Theorem 2:

$$BPL \subseteq SC.$$

For all (space constructible) space bounds $S(n) = \Omega(\log n)$,

$$BPSPACE(S(n)) \subseteq DTISP(2^{O(S(n))}, S^2(n)).$$

Here $BPSPACE(S)$ is the class of languages that can be recognized by a randomized space-S Turing machine which on every input outputs the correct answer with probability of at least 2/3 and which never loops[1]. BPL is the class $BPSPACE(\log n)$. $DTISP(T(n), S(n))$ is the class of problems that can be solved by deterministic Turing machines running in time $T(n)$ and space $S(n)$. SC, "Steve's Class", is $DTISP(poly, polylog)$. Clearly the same inclusion applies to the more commonly used one-sided error version of BPL, i.e. to RL.

We can state what is essentially an equivalent result in purely algorithmic terms.

Theorem 3: *There exists a deterministic algorithm for the following problem:*
Input: *An n by n transition probability matrix M, an integer t, and a rational ϵ. (t and ϵ are given in unary.)*
Output: *A matrix A such that $\|A - M^t\| \leq \epsilon$.*

[1] For an overview of the subtleties involved in the definition of randomized space bounded algorithms see e.g. [BCD*89, Nis93]. We follow the notation of [Nis93]. In the notation of [BCD*89] we are talking about $BPLP$.

The algorithm runs in time poly(N) *and space* $O(\log^2 N)$*, where* $N = n^2 + t + \epsilon^{-1}$ *is the input size.*

The simulation of randomized algorithms is based upon a careful usage of the pseudorandom generators of [Nis92]. The main idea of the simulation is to deterministically assign values to most of the input bits of the generator, leaving only $O(\log n)$ unassigned input bits. After this is done in an appropriate way, the simulation is concluded by going over all the possible assignments to the unassigned bits. For each such assignment we simulate the randomized algorithm using the output bits of the generator instead of random coin flips.

The values to the input bits are assigned sequentially, in groups of size $O(\log n)$ bits each. Each time we choose an assignment satisfying a certain "goodness" condition. These conditions are defined so as to insure that the probability distribution induced by a random choice of values to the unassigned bits behaves "similarly" to the probability distribution induced by a random choice of all the input bits to the generator. Here "behaves similarly" means that our simulated random-Logspace machine accepts with approximately the same probability.

Section 2 introduces our notation and goes over necessary preliminaries. In Section 3 we reduce the problem to simulating a finite state machine. In section 4 we describe the simulation itself.

2 Notation and Preliminaries

2.1 Finite State Machines

We model a BPL-machine running on a specific input as a Finite State Machine (FSM). The states of the FSM are all the (polynomially many) configurations of the RL machine and the alphabet is $\{0, 1\}$. Our aim is to approximate the probability of reaching an accepting state in the FSM starting from the initial state and following random choices at every step.

Definition 1

- *Let* Q *be a FSM,* i *and* j *be states of* Q *and* $x \in \{0, 1\}^*$*. We say that* x *takes* i *to* j *(in* Q*) if* Q*, when started at state* i *and fed the string* x*, ends at state* j*.*
- *Let* D *be a probability distribution on* $\{0, 1\}^*$*, and* Q *a FSM with* n *states. We denote by* $Q(D)$ *the* n *by* n *matrix such that*

$$Q(D)[i, j] = Pr_{x \in D}[x \text{ takes } i \text{ to } j \text{ in } Q].$$

- *For an integer t we denote by U_t the uniform distribution on $\{0,1\}^t$.*

Our aim in thus to find a good approximation of $Q(U_t)$, where t is polynomial in the size of Q.

2.2 Universal Hashing

Our generators are based upon universal hash functions [CW79]. Formally, let H be a set of functions $h : \{0,1\}^n \to \{0,1\}^m$.

Definition 2 *(Carter-Wegman) H is called a universal family of hash functions if for any $x_1 \neq x_2 \in \{0,1\}^n$ and $y_1, y_2 \in \{0,1\}^m$ we have that*

$$Pr_{h\in H}[h(x_1) = y_1 \text{ and } h(x_2) = y_2] \;=\; 2^{-2m}.$$

It is only important for this paper that it is possible to efficiently give small universal families of hash functions. I.e. such that each $h \in H$ can be represented by at most $O(n+m)$ bits, and such that computing $h(x)$ given the representations of h and x is efficient. An example of such a family is the set of all linear functions over $GF(2^n)$. We will require the following lemma from [Nis92]:

Lemma 1 *Let $A \subset \{0,1\}^m$, $B \subset \{0,1\}^m$, H be a universal family of hash functions $h : \{0,1\}^m \to \{0,1\}^m$, and $\epsilon > 0$, then*

$$Pr_{h\in H}[\,|Pr_{x\in\{0,1\}^m}[x \in A \text{ and } h(x) \in B] \;-\; \rho(A)\rho(B)| \;\geq\; \epsilon\,] \;\leq\; \frac{1}{2^m \epsilon^2},$$

where $\rho(A) = |A|/2^m$ and $\rho(B) = |B|/2^m$.

2.3 Approximations

It will be convenient to measure the distance between the effects of two distributions D_1, D_2 on a FSM Q by the $norm_1$ of the difference $Q(D_1) - Q(D_2)$. For a vector $x \in \mathcal{R}^n$ we define $\|x\| = \sum |x_i|$, and for a real n by n matrix M we define

$$\|M\| = \sup_{0\neq x\in\mathcal{R}^n} \frac{\|xM\|}{\|x\|}.$$

All norms appearing in this paper are these 1-norms. The following facts are standard:

- $\|M + N\| \leq \|M\| + \|N\|$.

- $\|MN\| \leq \|M\|\|N\|$.
- $\|M\| = \max_i(\sum_j |M[i,j]|)$.
- If M is a transition probability matrix, i.e. all entries are non negative, and the sum of entries in each row is 1, then $\|M\| = 1$.
- If all entries of M are bounded in absolute value by ϵ then $\|M\| \leq n\epsilon$.

3 The Problem to Solve

In section 4 we will exhibit an algorithm for the following problem:
INPUT: An n-state FSM Q; an integer t given in unary; a rational ϵ given in unary.
OUTPUT: An integer $t' \geq t$ and an n by n matrix A that satisfies $\|A - Q(U_{t'})\| \leq \epsilon$.
Note: It is more natural to ask for an approximation to $Q(U_t)$ itself. This can also be done but it slightly complicates the exposition, and thus we consider the problem as stated.

The running time of this algorithm is $O(n^{24}\epsilon^{-6}t^7 \log^3 N)$, where N is the total input size.

Let us now derive theorems 1,2, and 3 from this algorithm.

Proof: (of theorem 2) We will prove the simulation of BPL in SC. The simulation of $BPSPACE(S)$ for general space bounds S clearly follows.

Given an BPL machine T and an input x to it, we will reduce (in Logspace) the problem of simulating T on x to solving our problem: we first build a FSM Q representing the possible configurations of T and the transitions between them, given input x. It is easy to verify, and well known, that for any fixed x, T can indeed only have a polynomial number of configurations. We then set $t = n$ to be the number of states of Q, and, say, $\epsilon = 0.1$. We assume w.l.o.g. that once T reaches an accepting or rejecting state, it stays there, and that it never loops. In this case, the machine always stays stationary after the t'th step. The probability that T accepts is given in our notation by

$$\sum_{accepting\ states\ j} Q(U_{t'})[1,j]$$

for any $t' \geq t$ where 1 is the initial state of Q. This probability we can approximate within ϵ using A, the output of the algorithm, since

$$\sum_{accepting\ states\ j} (Q(U_{t'})[1,j] - A[1,j]) \leq \|A - Q(U_{t'})\| \leq \epsilon.$$

This approximation suffices to decide whether the BPL machine accepted or rejected with probability of at least $2/3$ since $1/3 + \epsilon < 2/3 - \epsilon$. □

Proof: (of theorem 3) Given the matrix M we build (in Logspace) a FSM Q such that for all $t' \geq tk$, $Q(U_{t'})$ contains a submatrix A such that $\|A - M^t\| \leq \epsilon$, where $k = O(\log N)$ is specified below. Q will have $(t+1) \cdot n$ "main" states $\{< j, i >: 1 \leq j \leq n, 0 \leq i \leq t\}$, and some more "intermediate" states which will be described below. All the states of the form $< j, t >$ will be stationary under both 0 and 1 inputs. Each state of the form $< j, i >$ for $0 \leq i < t$ will have a binary tree composed of intermediate vertices connecting it to states of the form $< j', i+1 >$. This binary tree will be a balanced tree of depth exactly k. The leaves of the tree are connected to states $< j', i+1 >$ so that the probability of reaching each state $< j', i+1 >$ from state $< j, i >$ is equal to $M(j, j')$ to within an error of $\epsilon/(nt)$. This is clearly easily done if $2^k \geq nt/\epsilon$, which is how k is chosen. It is now easy to see that the submatrix of $Q(U_{t'})$ corresponding to rows of the form $< j, 0 >$ and columns of the form $< j, t >$ gives exactly the matrix A desired. □

Proof: (of theorem 1): This follows directly from theorem 2, and the fact that *st*-connectivity is in RL [AKL*79]. □

4 The Simulation

4.1 The Generator

Our simulation uses the pseudorandom generator of [Nis92].

Definition 3 *For an integer m and every integer $k \geq 0$ we define a function*

$$G_k : \{0,1\}^m \times H^k \to \{0,1\}^{m2^k},$$

where H is a universal family of hash functions $h : \{0,1\}^m \to \{0,1\}^m$. G_k is defined recursively by

$$G_0(x) = x$$

and

$$G_k(x, h_1, \ldots, h_k) =$$

$$G_{k-1}(x, h_1, \ldots, h_{k-1}) \circ G_{k-1}(h_k(x), h_1, \ldots, h_{k-1}).$$

Here $\circ$ means concatenation of the two strings.

The trick in the algorithm will be to choose $h_1, \ldots, h_k$ carefully and then obtain a distribution on $\{0,1\}^{m2^k}$ by choosing $x \in \{0,1\}^m$ at random.

Definition 4 *For $h_1, \ldots, h_k \in H$ denote by $D_{h_1,\ldots,h_k}$ the probability distribution of $G_k(x, h_1, \ldots, h_k)$ induced by choosing x uniformly at random in $\{0,1\}^m$. Also denote $M_{h_1,\ldots,h_k} = Q(D_{h_1,\ldots,h_k})$ (here Q is the fixed FSM which was given as input).*

Our aim will be to choose $h_1, \ldots, h_k$ such that $M_{h_1,\ldots,h_k}$ approximates $Q(U_{m2^k})$ well. The functions will be chosen one after another satisfying the following "goodness" conditions.

Definition 5 *A function $h_k \in H$ is called ϵ-good for $h_1, \ldots, h_{k-1}$ if $\|M^2_{h_1,\ldots,h_{k-1}} - M_{h_1,\ldots,h_k}\| \le \epsilon$.*

(For the case $k = 1$ we define $M_{null} = Q(U_m)$.)

4.2 The Algorithm

4.2.1 The Main Loop

1. Define constants: $K = \log_2 t$ (rounded up); $\delta = \epsilon/(2^K)$; $m = 1 + 7\log_2 n - 2\log_2 \delta$ (rounded up); $t' = m2^K$. (The choice of m fixes a family H of hash function $h : \{0,1\}^m \to \{0,1\}^m$, where $|H| = 2^{O(m)}$.)

2. For $k = 1 \ldots K$ do

   (a) Use procedure "Find" to find h_k which is δ-good for $h_1, \ldots, h_{k-1}$.

3. For all states i, j of Q do

   (a) Use procedure "Compute" to compute $A[i,j] = M_{h_1 \ldots h_K}[i,j]$.

   (b) Output $A[i,j]$.

4.2.2 Procedure Find

INPUT: Hash functions $h_1, \ldots, h_{k-1}$; rational number $\delta > 0$.
OUTPUT: Hash function h_k such that h_k is δ-good for $h_1, \ldots, h_{k-1}$.
ALGORITHM (exhaustive search):

1. For all $h \in H$ do
   (a) For all states i, j of Q do
      i. Use procedure "Compute" to compute $p_1 = M_{h_1,\ldots,h_{k-1},h}[i,j]$.
      ii. Use $2n$ calls to procedure "Compute" to compute
      $$p_2 = \sum_{states\ l} M_{h_1,\ldots,h_{k-1}}[i,l] M_{h_1,\ldots,h_{k-1}}[l,j].$$
      iii. If $|p_1 - p_2| > \delta/n$ Then exit inner loop and goto next value of h.
   (b) Return $h_k = h$.

4.2.3 Procedure Compute

INPUT: Hash functions $h_1, \ldots, h_k$; states i, j.
OUTPUT: The value of $M_{h_1,\ldots,h_k}[i,j]$.
ALGORITHM:

1. count := 0;
2. For all $x \in \{0,1\}^m$ do
   (a) Simulate Q starting from i and running on the string $G_k(x, h_1, \ldots, h_k)$.
   (b) If the state reached is j Then count := count+1.
3. Return $M_{h_1,\ldots,h_k}[i,j] = count/2^m$.

Comment: Since we cannot store the whole value of $G_k(x, h_1, \ldots, h_k)$, we implement step 2a by computing the bits of $G_k(x, h_1, \ldots, h_k)$ one by one, as we need them for the walk on Q. The reader may verify that, in fact, these bits can be naturally computed in groups of m bits at a time.

4.2.4 Complexity Analysis

SPACE: The only place where more than $O(\log n)$ space is used, is the space needed to store the values of $h_1, \ldots, h_k$. Note that there are about $O(\log t)$ such functions and each of these can be represented by $O(m) = O(\log n + \log t + \log \epsilon^{-1})$ bits.
TIME: Procedure Find is called $K = O(\log t)$ times. In each call there is a loop over all $h \in H$. The size of H (using e.g. linear functions over $GF(2^m)$) is 2^{2m}. For each $h \in H$ we make $O(n^3)$ calls to procedure Compute. Procedure compute requires computing G_k 2^m times. Each computation of G_k requires time $O(m^2 2^k)$ (assuming a naive implementation in $O(m^2)$ time for each hashing operation). As $m = 7\log_2 n + 2\log_2 t + 2\log_2 \epsilon^{-1} + O(1)$, we get that the total running time is $O(n^{24}\epsilon^{-6}t^7\log^3 N)$ (where N is the total input size).

4.3 Correctness

We will need two lemmas.

Lemma 2 *Procedure Find always returns a function h_k which is δ-good for $h_1, \ldots, h_{k-1}$.*

Lemma 3 *If for every $1 \leq r \leq k$, h_r is δ-good for $h_1, \ldots, h_{r-1}$ then $\|M_{h_1,\ldots,h_k} - Q(U_{m2^k})\| \leq (2^k - 1)\delta$.*

Once these lemmas are known we get

Theorem 1 *The algorithm above outputs a matrix A such that $\|A - Q(U_{t'})\| \leq \epsilon$.*

Proof: (of Theorem) By lemma 2 we know that for all $1 \leq k \leq K$, h_k is δ-good for $h_1, \ldots, h_{k-1}$. By lemma 3 we get that $A = M_{h_1,\ldots,h_K}$ satisfies $\|A - Q(U_{m2^K})\| \leq (2^K - 1)\delta$. But $t' = m2^K$ and $(2^K - 1)\delta \leq \epsilon$. □

Proof: (of lemma 2) The proof follows from the following claim:
Claim: For any sequence of functions $h_1, \ldots, h_{k-1}$ there exists a function $h \in H$ such that for every i and j:

$$|\sum_l M_{h_1,\ldots,h_{k-1}}[i,l] M_{h_1,\ldots,h_{k-1}}[l,j] \; - \; M_{h_1,\ldots,h_{k-1},h}[i,j]| \leq \delta/n.$$

The claim implies the lemma since $\sum_l M_{h_1,\ldots,h_{k-1}}[i,l]M_{h_1,\ldots,h_{k-1}}[l,j]$ is exactly the i,j'th entry of $M^2_{h_1,\ldots,h_{k-1}}$. The difference matrix $M_{h_1,\ldots,h_{k-1},h} - M^2_{h_1,\ldots,h_{k-1}}$ thus has all of its entries bounded in absolute value by δ/n, which implies that its norm is bounded by δ.

Proof (of the Claim): For every pair i,j define sets:

$$A_{ij} = \{x \in \{0,1\}^m | G_{k-1}(x, h_1, \ldots, h_{k-1}) \, takes \, i \, to \, j\}.$$

Using this notation we have for all i,j: $M_{h_1,\ldots,h_{k-1}}[i,j] = \rho(A_{ij})$, Where $\rho(A) = |A|/2^m$. Using the definition of G_k, we also have:

$$M_{h_1,\ldots,h_{k-1},h}[i,j] = \sum_l Pr_{x\in\{0,1\}^m}[x \in A_{il} \, and \, h(x) \in A_{lj}].$$

Fix a triplet of states i,j,l. By lemma 1, if h is chosen uniformly at random from H then with probability of at least $1 - \frac{n^4}{2^m\delta^2} \geq 1 - 1/(2n^3)$ the following holds:

$$|Pr_{x\in\{0,1\}^m}[x \in A_{il} \, and \, h(x) \in A_{lj}] - \rho(A_{il})\rho(A_{lj})| \leq \delta/n^2.$$

By an averaging argument it is true that for at least half the functions $h \in H$, the above inequality holds for *all* triplets of states i,j,l.

Adding up the first term over all l gives $M_{h_1,\ldots,h_{k-1},h}[i,j]$, while adding the second term up for all l gives $\sum_l M_{h_1,\ldots,h_{k-1}}[i,l]M_{h_1,\ldots,h_{k-1}}[l,j]$. The difference is thus bounded from above by $n\delta/n^2 = \delta/n$. $\square$

Proof: (of lemma 3) The proof is by induction on k. For $k = 0$ the statement is trivial. Assume it is true for $k-1$ and prove for k.

We can estimate

$$\|M_{h_1,\ldots,h_k} - Q(U_{m2^k})\| \leq \|M_{h_1,\ldots,h_k} - M^2_{h_1,\ldots,h_{k-1}}\| + \|M^2_{h_1,\ldots,h_{k-1}} - Q(U_{m2^k})\|.$$

Since h_k is δ-good for $h_1, \ldots, h_{k-1}$, the first summand is bounded from above by δ. We will bound the second summand from above by $(2^k - 2)\delta$, and that will complete the proof of the lemma.

First note that $Q(U_{m2^k}) = Q(U_{m2^{k-1}})^2$. We estimate $\|M^2_{h_1,\ldots,h_{k-1}} - Q(U_{m2^{k-1}})^2\|$ from above by

$$\|M_{h_1,\ldots,h_{k-1}}\|\|M_{h_1,\ldots,h_{k-1}} - Q(U_{m2^{k-1}})\| + \|M_{h_1,\ldots,h_{k-1}} - Q(U_{m2^{k-1}})\|\|Q(U_{m2^{k-1}})\|.$$

The induction hypothesis implies that $\|M_{h_1,\ldots,h_{k-1}} - Q(U_{m2^{k-1}})\| \leq (2^{k-1} - 1)\delta$. The norms of $M_{h_1,\ldots,h_{k-1}}$ and of $Q(U_{m2^{k-1}})$ are 1 since they both are transition probability matrices. The above expression is thus bounded from above by $1 \cdot (2^{k-1} - 1)\delta + (2^{k-1} - 1)\delta \cdot 1 \ = \ (2^k - 2)\delta$. $\square$

5 Acknowledgements

I would like to thank the anonymous referee for several suggestions improving the presentation of the paper.

References

[AKL*79] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal sequences and the complexity of maze problems. In 20^{th} *Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico*, 1979.

[BR91] G. Barnes, and W.L. Ruzzo. Deterministic algorithms for undirected $s - t$ connectivity using polynomial time and sublinear space. In *Proceedings of the* 23^{st} *Annual ACM Symposium on Theory of Computing*, 1991.

[BCD*89] A. Borodin, S.A. Cook, P.W. Dymond, W.L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM J. Comput.*, 18(3):559–578, 1989.

[CW79] L. Carter and M. Wegman. Universal hash functions. *J. Comp. and Syst. Sci.*, 18(2):143–154, 1979.

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[Nis93] N. Nisan. On read-once vs. multiple access to randomness in Logspace. *Theoretical Computer Science*, 107:135–144, 1993.

[Sav70] W.J. Savitch. Relationships between nondeterministic and deterministic space complexities. *J. Comp. and Syst. Sci.*, 4(2):177–192, 1970.