Undirected Connectivity in $O(\log^{1.5} n)$ Space[*]

Noam Nisan
CS Department
Hebrew University
Jerusalem, Israel

Endre Szemeredi
Department of Mathematics
Rutgers University
New Brunswick, N.J

Avi Wigderson
CS Department
Hebrew University
Jerusalem, Israel
and
Princeton University
Princeton, N.J.

Abstract

We present a deterministic algorithm for the connectivity problem on undirected graphs that runs in $O(\log^{1.5} n)$ space. Thus, the recursive doubling technique of Savich which requires $\Theta(\log^2 n)$ space is not optimal for this problem.

1 Introduction

The *st*-connectivity problem is one of the most basic graph problems, and has received much attention. Given a directed graph G and two vertices in it s and t, the problem is to determine whether there exists a directed path from s to t in G. Several linear time algorithms are known for this problem; they all require linear space. The best space complexity known for this problem is $O(\log^2 n)$, which is achieved by Savitch's algorithm [Sav70]. It is a long standing open problem to find a smaller space algorithm for connectivity. As connectivity is complete for the complexity class NL, such an algorithm would imply improved deterministic simulation of nondeterministic space. It has thus been conjectured that such algorithms cannot be found and that Savitch's algorithm is indeed optimal.

A special case of the connectivity problem which is of special interest is "undirected connectivity", i.e. the case where G is undirected. For this case a *randomized* Logspace algorithm was given in [AKL*79] (with a zero-error version in [BCD*89]). No *deterministic* algorithm that improved upon Savitch's space requirement was known, although improved time-space tradeoffs were given in [BR91, Nis91]. It has also been a long standing open problem to improve upon Savitch's algorithm even for undirected graphs.

In this paper we present such an algorithm. Our algorithm solves the connectivity problem on undirected graphs and uses only $O(\log^{1.5} n)$ space. The algorithm uses the universal sequences obtained from the pseudorandom generators of [Nis90], as well as pair-wise independent sampling. It is also possible to give a variant of our algorithm that does not rely on any pseudorandom generators and still beats Savitch's algorithm by requiring only $O(\log^2 n/\log\log n)$ space.

An immediate application is a EREW parallel algorithm for undirected connectivity that runs in $O(\log^{1.5} n)$ time and uses a polynomial number of processors. The same time bound was achieved in [JM91] in the weaker CREW model using only a linear number of processors. Recently, [KNP92] show how a variant of our algorithm can be implemented on an EREW PRAM using a linear number of processors.

2 The Algorithm

2.1 Top Level

The input to our algorithm is an undirected graph G over N vertices. It outputs whether or not vertex 1 is connected to vertex 2 in the graph.

The main procedure we will use is procedure *shrink*. This procedure accepts as input a graph G over n vertices and a parameter n'. Its output can take two forms. It may either output an answer whether vertex 1 is connected to vertex 2 in G or else it may output a graph G' over n' vertices with the property that vertex 1 is connected to vertex 2 in G' if and only if vertex 1 is connected to vertex 2 in G. Procedure shrink requires $O(\log n + (\log(n/n'))^2)$ space.

[*]The first and last authors were partially supported by the Wolfson Research Awards

Let us now see how we solve the connectivity problem on a graph G with N vertices using procedure shrink. We can view procedure shrink as answering queries of the form $(u,v) \in E(G')$?, where its input is given by queries of the form $(u,v) \in E(G)$?. We will thus use procedure shrink recursively. For $i = 0 ... \sqrt{\log N}$ define $n_i = 2^{i\sqrt{\log N}}$. The i'th recursion level of procedure shrink will work on a graph G_i over n_i vertices, and its output will be the graph G_{i-1} over n_{i-1} vertices. It will answer queries of the $(i-1)$'st recursion level regarding G_{i-1} by running procedure shrink on the graph G_i (using queries which are, in turn, answered by the $i+1$'st recursion level etc.). The base of the recursion is the $\sqrt{\log N}$'th level where we have the original graph G.

Notice that (1) G_0 is trivial so the first recursion level tells us whether vertex 1 is connected to vertex 2; (2) The recursion depth is only $\sqrt{\log N}$; (3) Each recursion level requires $O(\log N)$ space (since $\log(n_i/n_{i-1}) = \sqrt{\log N}$).

2.2 Procedure Shrink: Overview

Procedure shrink accepts a graph G over n vertices, and must output a graph G' which is smaller than G by a factor of $f = n/n'$ and still captures the connectivity of G. The Outline of the algorithm is as follows:

1. For each vertex v we find a set $N(v)$ of vertices that are connected to v in G. This set is of size polynomial in f. (Unless v's connected component is smaller than that – these cases are immediately discarded).

2. A subset R of the vertices of G is chosen. R is of size $n' = n/f$ and it satisfies the condition that for every vertex v it intersects $N(v)$. R will be the vertex set of G'.

3. For each vertex v we choose a representative vertex $rep(v) \in R \cap N(v)$. The edges of G' are now all the edges of the form $(rep(u), rep(v))$ where (u,v) is an edge of G.

The first stage is implemented by walking in G, starting form v, along a universal sequence for graphs of size $poly(f)$. We use the universal sequences of Nisan [Nis90] which can be generated in space $O(\log^2 f)$.

The second stage is implemented by using pair-wise independence, where each vertex decides at random to be in R with probability $O(1/f)$. This takes care of intersecting almost all $N(v)$'s, and we have to add only a small number of extra vertices from the unsatisfied $N(v)$'s. Again by trying all possibilities from the pair-wise independent space this stage is done deterministically.

Remark: as is usual when working in sub-linear space we cannot store any of the intermediate results (such as $N(v)$ or R) in our limited working storage. One should thus view this outline as describing procedures, each one implemented using calls to the previous one.

2.3 Details of Procedure Shrink

2.3.1 Computing $N(v)$

$N(v)$ is computed by walking along a universal sequence. We will use universal sequences for 3-regular graphs as defined in [AKL*79].

Definition 1 *A 3-regular graph G is called labeled if the three edges adjacent to each vertex are numbered by some permutation of $1, 2, 3$. For a vertex v in G and a string $\sigma \in \{1,2,3\}^*$, a walk on G starting from v and following the directions of σ is naturally defined.*

We say that σ is an m-universal traversal sequence if for every connected 3-regular m-vertex graph G, every labeling of it and every vertex v, the walk on G starting from v following σ visits all vertices of G.

In [AKL*79] it is shown that polynomial length universal sequences exist. The best explicit construction known is due to [Nis90]:

Theorem 1 *For every m there exists an m-universal sequence σ of length $m^{O(\log m)}$. Each entry σ_i of these sequences can be computed in $O(\log^2 m)$ space.*

Let $\sigma \in \{1,2,3\}^*$. We wish to walk on a non-regular graph G according to σ. To do this we imagine each vertex of degree $d > 3$ as composed of d pseudo-vertices connected to each other by a cycle. Each of these pseudo-vertices has degree 3: two edges being part of the cycle, and one edge going outward, being one of the edges of the original vertex. We can now naturally use σ to perform a walk on the 3-regular graph composed of these pseudo-vertices.

Remark: One can easily extend the definition and constructions of universal sequences to non-regular graphs. If this is done then this slight complication of "converting" a non-regular graph to a 3-regular one can be eliminated.

Fix $m = 6^4 f^4$, and let $\sigma \in \{1,2,3\}^*$ be an m-universal sequence. For every vertex v we will walk on G, starting from v, according to σ. Define $N(v)$ to be the set of vertices encountered in this walk, as well as their immediate neighbours.

For the computation of each step of the walk we need only remember the current pseudo-vertex and generate one entry from the sequence. Thus each step and hence enumerating $N(v)$ can be done in space $O(\log n + \log^2 f)$.

It is clear that $N(v)$ is a subset of the connected component of v. We will show that $N(v)$ is also rather large, unless it equals to the whole connected componnet of v (denoted $Comp(v)$).

Lemma 1 *If σ is an m-universal sequence, then either $N(v) = Comp(v)$ or $|N(v)| \geq \sqrt{m}$.*

Using our choice of m we get that for every vertex v, $|N(v)| \geq 36f^2$.

2.3.2 Representatives

We first choose a subset L of the vertices to be possible representatives. L will be chosen from a pairwise independent distribution, as to satisfy a certain condition given below. Vertices 1 and 2 will then be added to it. L will be represented succintly using $O(\log n)$ bits by its index in the probability space. Once L is chosen we define $rep_L(v)$ for every vertex v as follows:

1. If $N(v) = Comp(v)$ then $rep_L(v) = 0$. (No representative needed if component is too small.)
2. Else, if $N(v) \cap L = \emptyset$ then $rep_L(v) = v$.
3. Else (the "usual" case), $rep_L(v)$ is the smallest labeled vertex in $N(v) \cap L$.

The final set of representatives will be $R_L = \{rep_L(v)\}$ where v ranges over all vertices of G. I.e. R_L will contain some vertices from L as well as those vertices v that have no L-member in $N(v)$ even though $N(v) \neq Comp(v)$.

The condition that L must satisfy is that $|R_L| \leq n' = n/f$. The fact that such a set can be found in a pairwise independent distribution space is ensured by the following lemma.

Lemma 2 *If L is chosen at random such that every vertex is in L with probability $1/(6f)$ and the choices are pair-wise independent, then $Pr[|R_L| \leq n'] > 0$.*

Let us briefly discuss the algorithmic aspects. Let us first see that $rep_L(v)$ can be computed form v and L in space $O(\log n + \log^2 f)$. Cases 2 and 3 are easy as we need only enumerate the vertices of $N(v)$. Checking whether $N(v) = Comp(v)$ is done by going over all vertices $u \in N(v)$; for each u we check that for all its neighbours w it is also true that $w \in N(v)$.

Choosing L is done as follows. We use a space of polynomial (in n) size satisfying the conditions of the lemma, and such that computing for a vertex v whether $v \in L$ is easy given the $O(\log n)$ bits which are the index of L in the space. (Such spaces are well known, e.g., take a field F_q of size poly(n); each L is defined by two elements $a, b \in F_q$; and $v \in L_{a,b}$ if $av + b \in \{1...q/(6f)\}$.) We go over all possibilities of L in the space (in a fixed order). For each we construct R_L, and check whether its size is small enough. If so we stick with this (first encountered) L.

2.3.3 The Algorithm

1. If $N(1) = Comp(1)$ or $N(2) = Comp(2)$ Then
   (a) If $2 \in N(1)$ Then Output("Connected") and Halt.
   (b) Else Output("Not Connected") and Halt.
2. Choose L as described in the previous subsection.
3. If $rep_L(2) = 1$ Then Output("Connected") and Halt.
4. The vertex set of G' is R_L.
5. An edge (u, v) is an edge of G' if there exists an edge (u', v') of G such that $rep_L(u') = u$ and $rep_L(v') = v$.

Remark: Formally one should rename the vertices of R_L to have the names $1...n'$. This is easily done by renaming v to be its rank in R_L.

It is clear that in those cases that this algorithm actually gives an answer it is correct. Correctness when a graph G' is produced is ensured by:

Lemma 3 *If the previous algorithm outputs a graph G' then Vertex 1 is connected to vertex 2 in G if and only if vertex 1 is connected to vertex 2 in G'.*

3 Proofs of correctness

3.1 Proof of lemma 1

Assume that $N(v)$ does not satisfy the lemma. Let $N'(v)$ be the subset of vertices actually visited by the walk. ($N(v)$, by definition, includes also the neighbours of vertices in $N'(v)$).

As $N(v)$ is contained in $Comp(v)$, but is not equal to all of it, there must be a vertex $u \in Comp(v) - N(v)$

which is adjacent to a vertex $w \in N(v)$. It is clear that $w \notin N'(v)$. Now consider the induced graph on $N(v)$, and consider the graph of pseudo-vertices composing the vertices of this graph. This is a connected 3-regular graph. Its size is at most m since $|N(v)| \leq \sqrt{m}$ and each vertex is composed of at most $\sqrt{m}$ pseudo-vertices. However, a walk from v according to σ on this graph will behave exactly as the walk from v performed in G, and thus will not visit w. This contradicts σ being a universal sequence for 3-regular m-vertex graphs.

3.2 Proof of lemma 2

R_L is composed of two types of vertices: those in L and those vertices v such that satisfy both $N(v) \cap L = \emptyset$ and $|N(v)| \geq 36f^2$. Call the second set of vertices B. We will show that (w.h.p.) these two subsets are small.

Claim 1: With probability of at least 2/3, $|L| \leq n'/2$.
Claim 2: With probability of at lest 2/3, $|B| \leq n'/2$.

It follows that with probability of at least 1/3 the size of both sets combined is at most n'.

Proof of Claim 1: The expected size of L is $n/(6f) = n'/6$. Using the Markov inequality, the probability that $|L|$ is at least 3 times its expected value is at most 1/3.

Proof of Claim 2: Fix a vertex v such that $|N(v)| \geq 36f^2$. We assume w.l.o.g. that actually $|N(v)| = 36f^2$. We will first show that

$$Pr_L[v \in B] = Pr_L[N(v) \cap L = \emptyset] \leq 1/(6f)$$

For each vertex $u \in N(v)$ we define a random variable X_u which is 1 if $u \in L$ and 0 otherwise. The variables X_u are pairwise independent. We also define a random variable $X = \sum_{u \in N(v)} X_u$. We have that $v \in B$ iff $X = 0$. We bound the probability that $X = 0$ using the Chebyshev inequality. The expected value of X is $E(X) = |N(v)|/(6f) = 6f$. The variance of X may be computed as if the X_u's were truly independent, and is $Var(X) = |N(v)| \cdot 1/(6f) \cdot (1 - 1/(6f)) \leq 6f$. Using the Chebyshev inequality,

$$Pr[X = 0] \leq Pr[|X - E(X)| \geq 6f] \leq$$

$$\leq (6f)^{-2} Var(X) \leq 1/(6f)$$

It follows that the expected size of B is at most $n/(6f) = n'/6$. Finally, using the Markov inequality, the probability that $|B|$ is more than 3 times its expected value is at most 1/3.

3.3 Proof of lemma 3

First note that $rep(1) = 1$ and $rep(2) = 2$ since we always put vertices 1 and 2 in L and we always take the smallest numbered vertex in $N(v) \cap L$ as $rep(v)$. (The algorithm has specifically ruled out the possibilities that $rep(1) = 0$ or $rep(2) = 0$ or $rep(2) = 1$.)

Assume that vertex 1 is connected to vertex 2 in G'. Notice that if u is a neighbour of v in G' then there is a path from u to v in G. The reason is that there exists an edge (u', v') in G such that $rep(u') = u$ and $rep(v') = v$. Since, in G, u' is connected to $u = rep(u')$ and v' is connected to $v = rep(v')$, we get that u is connected to v in G. It follows that if vertex 1 is connected to vertex 2 in G' they are also connected in G.

Conversely, assume that 1 is connected to 2 in G say thru the path $v_1, v_2, ..., v_k$. In this case we claim that 1 will be connected to 2 in G' thru the path $rep(v_1), rep(v_2), ..., rep(v_k)$. Indeed, notice that if v and u are neighbours in G then either $rep(u) = rep(v)$, or else $rep(u)$ is a neighbour of $rep(v)$ in G'.

4 Variants and Remarks

Small modifications of this algorithm, as well as using some of its ideas can give several other results. In particulr the following remarks should be made.

- It is possible to obatin an $O(\log^2 n / \log \log n)$ algorithm for connectivity without using any pseudorandom generators. This gives an elementary algorithm beating Savitch's bound. By taking, starting from a vertex v, all possible walks specified by $O(\log n)$ bits, one can ensure seeing at least $\log n / \log \log n$ vertices. This is so since it is sufficient to keep track of which outgoing edge was taken from each encountered vertex in the DFS, and untill a vertex of degree higher than $\log n$ is encountered (in which case we can stop), only $\log \log n$ bits are needed per vertex. Doing this enables one to get sets $N(v)$ of size $\log n / \log \log n$ (instead of size $2^{\sqrt{\log n}}$ using universal sequences). Procedure shrink can then be used to shrink the graph by a factor of $(\log n)^{\Theta(1)}$, getting a recursion depth of only $\log n / \log \log n$.

- It has been suggested by [KNP92] and by [ST92] that the usage of pair-wise independence can be eliminated. The following mechanism can be used to choose the set of representatives R instead: a vertex v is in R if $N(v)$ does not intersect $N(w)$

for all vertices w of lower rank (in a fixed ordering). As the chosen v's have pairwise disjoint $N(v)$'s, we have $|R| \leq n/f$ as long as $|N(v)| \geq f$ for every vertex v. Choosing a representative $rep(v)$ for every vertex v is now slightly more complicated as is is not always true that R intersects $N(v)$. However, the following simple loop easily finds a representative $rep(v) \in R$:

```
1. $u \leftarrow v$
2. while $u \notin R$ do
     – $u \leftarrow min(w)$ such that $N(u) \cap N(w) \neq \emptyset$.
3. $rep(v) \leftarrow u$
```

- For various specific families of graphs one can get an $O(\log n)$ space algorithm. For example, let $t(n)$ be the largest size of a graph for which we can test connectivity in $O(\log n)$ space (we have $t(n) \geq exp((\log n)^{2/3})$). It is possible to test connectivity on graphs of minimum degree $n/t(n)$ in Logspace. Simply we let $N(v)$ be the neighbour set of v, and one application of procedure shrink can reduce the graph size to $t(n)$.

- One can find neighbour sets $N(v)$ of size n^ϵ quite easily using BFS or DFS. This allows procedure shrink to shrink the graph by a factor of n^ϵ and achieve only a constant recursion depth. This yields a polynomial-time $O(n^\epsilon)$-space algorithm for connectivity. In fact this algorithm resembles that of [BR91].

- It is well known (see e.g. [KR90]) that LOGSPACE can be simulated by an EREW PRAM running in logarithmic time and polynomial number of processors. Thus it is not difficult to convert our algorithm to a parallel one that requires $O(\log^{3/2} n)$ time on an EREW PRAM, and uses a polynomial number of processors. This matches the time bound of the recent algorithm of [JM91], which works on the stronger CREW PRAM. We note, however, that they need only a linear number of processors, while in our algorithm the polynomial is quite hugh. In [KNP92] a linear-procesor variant of our EREW algorithm is obtained.

- All the algorithms above can be easily converted so as to produce the connected components of the input graph, as well as a path between every connected pair of vertices.

- Any improved construction of universal sequences may be plugged into our algorithm to obatin an improved algorithm for connectivity. In fact our algorithm may be seen to convert a wide variaty of algorithms for connectivity (e.g. some JAG algorithms [CR80]) to faster ones. This is done by using such an algorithm to find the sets $N(v)$.

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 20th *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 23st 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.

[BNS90] L. Babai, N. Nisan, and M. Szegedy. Multiparty Protocols and Logspace-hard pseudorandom sequences. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, 1990.

[CR80] S. Cook and C. Rackoff. Space Lower Bounds for Maze Threadability on Restricted Machines. In *SIAM J. Comput.*, 9(3): 636–652, 1980.

[JM91] D.B. Johnson, and P. Metaxas. Connected Components in $O(\log^{3/2} |V|)$ parallel time for the CREW PRAM. *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico*, 1991.

[KNP92] D. Karger, N. Nisan, and M. Parnas. Fast Connected Components Algorithms for the EREW PRAM. To appear in *SPAA*, 1992.

[KR90] R. M. Karp and V. Ramachandran. Parallel Algorithms for Shared-Memory Machines.

In *Handbook of Theoretical Computer Science*, Vol A, J. van Leeuwen Ed., 869–932, 1990.

[Nis90] N. Nisan. Pseudorandom generators for space-bounded computation. In *Proceedings of the* 22st *Annual ACM Symposium on Theory of Computing*, 1990.

[Nis91] N. Nisan. $RL \subseteq SC$. *Proceedings of the* 24th *Annual ACM Symposium on Theory of Computing*, 1992.

[ST92] R. K. Sinha and M. Tompa. Private communication.

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