Deterministic Space-Bounded Graph Connectivity Algorithms

Jesper Jansson

1 Introduction

The *st-connectivity problem*, also referred to in the literature as the *st-reachability problem* or the *graph accessibility problem*, is the following:

> Given a directed graph $G = (V, E)$, and two vertices s and $t \in V$, is there a path from s to t in G?

If the edges are undirected (which is a special case of the above, as can be seen by interpreting each undirected edge as two directed edges, one in each direction), the problem is often called *undirected st-connectivity*.

Throughout this paper, we use the abbreviations STCON for the st-connectivity problem and USTCON for the undirected version of STCON.

First of all, note that when talking about the space complexity of an algorithm, we are concerned about the size of the work space it uses, i.e., the space that the (read-only) input itself occupies is not counted against the algorithm. Next, since we will measure and compare the space used by different algorithms as compared to the size of their input, we should specify what the input looks like. Suppose that G is represented as a string over the alphabet **{0,1,{,},,}** as follows. The vertices in V are numbered and encoded in binary. The string that represents G is simply a listing of V followed by a list of the edges in E; for example, if $G = (\{1,2,3,4\}, \{\{1,3\},\{1,4\},\{4,2\}\})$ then the representation of G would be $\{1,10,11,100\},\{\{1,11\},\{1,100\},\{100,10\}\}$. If $|V| = n$ and the length of the representation is r, then $\log r = O(\log n)$ and $\log n = O(\log r)$ since $n \leq r \leq cn^2$ for some constant c. What this means is that we don't need to worry so much about the exact representation of the given graph G when working with asymptotic space bounds; we can just assume that the input size is measured in terms of the number of vertices and edges in G. As is standard in the literature, we let $n = |V|$ and $m = |E|$.

STCON can be solved deterministically in polynomial (in n) time and space by depth-first or breadth-first search techniques. In the worst case, any algorithm might have to go through all vertices and edges of G to find a path from s to t, so the lower bound on the time required to solve the problem is linear and can't be pushed below this. However, an interesting question is whether there exist methods that always use much less than linear space.

2 A Quick Review of Complexity Classes

2.1 Space Complexity Classes

$DSPACE(S(n))$ is the class of languages accepted by deterministic Turing machines running in space $O(S(n))$. We sometimes use the notation $L^{\alpha} = DSPACE((\log n)^{\alpha})$ for $\alpha > 0$.

Similarly, $NSPACE(S(n))$ is the class of languages accepted by nondeterministic Turing machines running in space $O(S(n))$, and $NL^{\alpha} = NSPACE((\log n)^{\alpha})$ for $\alpha > 0$.

$SSPACE(S(n))$ is the class of languages accepted by symmetric Turing machines (see Section 3.4) running in space $O(S(n))$. SL is the class of symmetric nondeterministic logspace computations.

The relationship between these complexity classes is $L \subseteq SL \subseteq NL$.

2.2 Probabilistic Complexity Classes

RL = The class of languages accepted by probabilistic Turing machines with one-sided error running in $O(\log n)$ space.

RLP = The class of languages accepted by probabilistic Turing machines with one-sided error running in $O(\log n)$ space and polynomial expected time.

$ZPLP$ = The class of languages accepted by probabilistic Turing machines with zero-sided error running in $O(\log n)$ space and polynomial expected time.

Observe that $RLP \cap co{-}RLP = ZPLP$.

2.3 Miscellaneous Complexity Classes

SC = "Steve's Class" = $DTISP$(poly, polylog)

where $DTISP(T(n), S(n))$ = The class of languages accepted by deterministic Turing machines running in time $T(n)$ and using space $S(n)$.

3 History

3.1 Savitch (1970)

Walter Savitch [19] invented the first deterministic algorithm for STCON that runs in sublinear space. It uses $O(\log^2 n)$ space and $n^{\Theta(\log n)}$ time. See also Section 3.2.

The idea behind Savitch's algorithm is to use the work space as a stack for activation records that are created recursively. Define the answer to the subproblem $\mathrm{PATH}(s, t, i)$ to be *true* if there is a path of length $\leq 2^i$ from s to t in G, and *false* if there is not. A recursive solution for $\mathrm{PATH}(x, y, i)$ is obtained by evaluating $\mathrm{PATH}(x, z, i-1)$ AND $\mathrm{PATH}(z, y, i-1)$ for all vertices $z \in V$. Each level of recursion pushes an activation record (x, y, i) on the stack; since the size of an activation record is bounded by $3 \log n$ and there are no more than $\log n$ recursion levels (and thus no more than $\log n$ activations records on the stack at any given time), the total space needed is in $O(\log^2 n)$.

Remark: Several of the latest papers about space-bounded connectivity that quote Savitch's article [2, 16, 17, 18] contain the same bibliographical error in their reference lists...

3.2 Jones (1975 & 1977)

Jones [10, 11] showed that STCON is complete for NL with respect to logspace reductions. (Given any language in NL, accepted in $O(\log n)$ space by some nondeterministic Turing machine M with one final state and one accepting configuration, to decide if M accepts an input string x is the same thing as answering the question "Does the initial configuration of M corresponding to input x lead to the accepting configuration of M?". Clearly, this is equivalent to "Is there a path in the configuration graph for M from the node representing the initial configuration corresponding to input x to the node representing the accepting configuration?", which is an instance of STCON.)

This result immediately implies that $NL \subseteq L^2$, and is implicit in Savitch's article.

3.3 Aleliunas, Karp, Lipton, Lovász, and Rackoff (1979)

A probabilistic logspace algorithm for USTCON was given by Aleliunas, Karp, Lipton, Lovász, and Rackoff in [1]. In terms of complexity classes, they showed that USTCON $\in RLP$.

A *universal traversal sequence* is a deterministic instruction sequence for walking around in a graph which guarantees that every vertex is visited eventually if the graph is connected. The authors proved that a universal traversal sequence of polynomial length exists, so that a random walk can be simulated, thereby getting their probabilistic algorithm. They showed that $SL \subseteq RL$ (before the class SL had been invented!), but weren't able to deterministically generate short universal traversal sequences in logarithmic space.

3.4 Lewis and Papadimitriou (1982)

Symmetric Turing machines are a kind of Turing machines with limited nondeterministic power, which were introduced by Lewis and Papadimitriou in [14] in an attempt to formalize the notion of symmetric computations. Informally, symmetric computations are reversible in an approximative manner. A symmetric computation can be divided into locally deterministic subcomputations (*segments*) between *special configurations* at which all nondeterminism takes place, such that the only special configuration reachable from a backwards computation beginning somewhere in the middle of a segment is the special configuration that started that segment. Note that backwards computations from the middle of a segment are not necessarily deterministic, but the first special configuration that will be reached is uniquely determined. Thus, the segments are locally deterministic in the forwards direction and have limited nondeterminism in the backwards direction. Also, if there is a way to get from special configuration A_1 to special configuration A_2 in the forwards direction, then it must be possible to get from A_2 to A_1 in the forwards direction (again, not necessarily passing through the same set of configurations). See Figure 1.

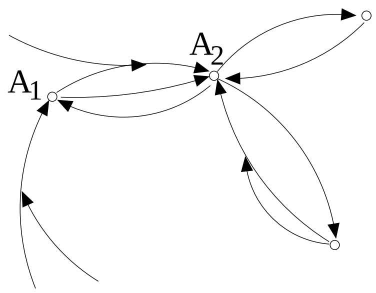

Figure 1: Symmetric computation: All nondeterminism is gathered at special configurations, marked by circles. The arcs represent locally deterministic subcomputations passing through non-special configurations.

An important theorem shows that USTCON is SL-complete under logspace reductions. For the proof, the authors construct a nondeterministic Turing machine M that accepts USTCON in $O(\log n)$ space, and invoke a lemma to ensure that M can be converted to a symmetric Turing machine. Then the theorem follows as any language accepted in logarithmic space by a symmetric Turing machine is logspace reducible to USTCON (because of the properties of symmetric computations discussed above, the configuration graph can be viewed an undirected graph whose nodes correspond to the special configurations). The significance of the theorem is twofold: (1) any result of the type USTCON $\in X$ implies that $SL \subseteq X$; and (2) it suggests that USTCON might be "easier" than STCON. Most efforts since then have been concentrated on USTCON rather than STCON.

3.5 Borodin, Cook, Dymond, Ruzzo, and Tompa (1989)

Borodin, Cook, Dymond, Ruzzo, and Tompa [5] carried the results of Aleliunas *et al.* one step further. They showed that USTCON $\in ZPLP$ by proving that $\overline{\text{USTCON}}$ (the complement of USTCON) also lies in RLP, and so their main result was an $O(\log n)$ space, errorless probabilistic algorithm for USTCON with polynomial expected running time.

3.6 Nisan (1990 & 1992)

Nisan [15] proved that a universal traversal sequence can be constructed in $O(\log^2 n)$ space using a pseudorandom generator. As a side effect, this implies that the algorithm of Aleliunas *et al.* can be derandomized.

In another article [16], Nisan proved that $RL \subseteq SC$, which yields a deterministic $O(\log^2 n)$ space, polynomial time algorithm for USTCON. (Here, the exponent hidden behind the word 'polynomial' is something like 45...) In general, any problem computable in randomized logspace (*i.e.*, any problem in RL) has a deterministic algorithm that runs in polynomial time and $O(\log^2 n)$ space, by using Nisan's pseudorandom generator to simulate the randomness. (Assign values deterministically to most of the input bits of the pseudorandom generator, and then try all possible assignments to the $O(\log n)$ remaining bits.)

3.7 Barnes and Ruzzo (1991)

The first deterministic algorithms for USTCON running in both sublinear space and polynomial time were provided by Barnes and Ruzzo [4]. (This was before Nisan's 1992 paper [16].) A deterministic time-space tradeoff for USTCON was given.

3.8 Nisan, Szemeredi, and Wigderson (1992)

Nisan, Szemeredi, and Wigderson [17] described a deterministic algorithm for USTCON that only uses $O(\log^{3/2} n)$ space, finally improving on Savitch's deterministic space bound after more than 20 years!

The central procedure of the algorithm is a graph shrinking procedure called SHRINK. It takes a graph G as input, and outputs a smaller graph G' in which s' and t' are connected if and only if s and t were connected in G. The algorithm uses short universal traversal sequences from all vertices in the graph to visit large neighborhoods and then pairwise independent sampling (which can be derandomized) in order to create the smaller graph.

This algorithm is treated in detail in Section 5.1.

3.9 Saks and Zhou (1995)

An *off-line randomized algorithm* is a randomized algorithm that when started with input x, computes the number of random bits it is going to need for the computation, asks the random source for a string y of this length, stores y in read-only memory, and then proceeds deterministically.

Saks and Zhou [18] used off-line randomized algorithms together with some new techniques for lowering the number of required random bits to obtain the following result.

> Any language that can be recognized by a randomized algorithm (possibly with two-sided error) that runs in space S and expected time $2^{O(S)}$ can be recognized by a deterministic algorithm running in space $S^{3/2}$.

In particular, $RL \subseteq L^{3/2}$.

Saks and Zhou exploited the fact that the repeated squaring problem for matrices is closely related to the problem of derandomizing space-bounded randomized algorithms.

3.10 Armoni, Ta-Shma, Wigderson, and Zhou (1997)

The best deterministic space bound for USTCON so far is by Armoni, Ta-Shma, Wigderson, and Zhou in [2]: $O(\log^{4/3} n)$.

The authors merged several ideas from the previous papers, and modified the way that neighbors in the graph are visited in the pseudorandom walk.

See Section 5.2 for a summary of the algorithm.

3.11 Barnes, Buss, Ruzzo, and Schieber (1998)

Barnes, Buss, Ruzzo, and Schieber [3] went back to attack STCON, and found the first deterministic, sublinear space algorithm running in polynomial time. To be precise, it uses $n/2^{\Theta(\sqrt{\log n})}$ space.

The algorithm consists of two subalgorithms with different time-space tradeoffs. The first one does a modified breadth-first search to compute a subtree of the breadth-first search tree rooted at s, only keeping vertices from certain levels; the second one is a recursive subroutine that locates short paths between the stored levels. When combining the two subalgorithms, the authors optimize the total running time by using the short paths algorithm to answer many short path queries at once.

4 A Parallel Development

A natural generalization of USTCON is the problem of computing the connected components in a given undirected graph, hereafter referred to as CCOMP. A lot of research has been done on how to efficiently solve CCOMP when parallel processors are available and the EREW PRAM model is assumed; the difference is that instead of trying to minimize the deterministic space (and running time) required for USTCON, people try to minimize the running time (and number of processors) needed for CCOMP. It is well known that logspace can be simulated by an EREW PRAM running in logarithmic time and with a polynomial number of processors [13], so if STCON $\in L$ then we could solve CCOMP on an EREW PRAM in logarithmic time. Note that the converse does not have to be true; CCOMP might be solvable on an EREW PRAM in $O(\log n)$ time even if $L \neq NL$. The recent result by Chong, Han, and Lam [6] (see Section 4.6) shows that, in fact, CCOMP *is* solvable on an EREW PRAM in $O(\log n)$ time, which settles a long-standing open question.

Most of the parallel algorithms for CCOMP are based on the following "hook-and-contract" scheme, which in its unmodified form takes $O(\log^2 n)$ time to run on an EREW PRAM.

repeat
1. Partition the graph into a set of components by letting each vertex hook up to some neighbor, if it has any.
2. Contract every component into a representative vertex.

until every connected component of G has contracted to one vertex

4.1 Shiloach and Vishkin (1982)

For the CRCW PRAM, there exists an $O(\log n)$ time deterministic algorithm that runs on $n + m$ processors, as was shown by Shiloach and Vishkin [20]. However, to simulate a CRCW algorithm on an EREW PRAM slows down the running time by a factor of $\log n$, so this result is comparable to Savitch's (see Section 3.1).

4.2 Cole and Vishkin (1986)

Cole and Vishkin [8] showed that there is a parallel algorithm for CCOMP running in $O(\log n)$ time and using $\frac{(n+m)\,\alpha(n,m)}{\log n}$ processors, where $\alpha(n, m)$ is the inverse of Ackermann's function.

4.3 Johnson and Metaxas (1991)

A CREW PRAM algorithm for CCOMP that uses only $O(\log^{3/2} n)$ time and $n + m$ processors was given by Johnson and Metaxas [9]. The algorithm can be modified to run on an EREW PRAM with the same bounds.

4.4 Karger, Nisan, and Parnas (1992)

Karger, Nisan, and Parnas [12] published a randomized EREW algorithm which uses $O(\log n)$ time and $\frac{n^{1+\epsilon}+m}{\log n}$ processors, for any fixed $\epsilon > 0$. A deterministic version of the algorithm runs in $O(\log^{3/2} n)$ time with $n+m$ processors.

4.5 Chong and Lam (1995)

Chong and Lam [7] presented a deterministic EREW PRAM algorithm for CCOMP that instead of forcing the components to hook and contract uniformly, uses a new way of hooking, along with a generalized "growth control" schedule, to execute in $O(\log n \log\log n)$ time with $n+m$ processors.

4.6 Chong, Han, and Lam (1999)

Chong, Han, and Lam [6] came up with a deterministic EREW PRAM algorithm that uses $n+m$ processors and runs in $O(\log n)$ time, thereby beating all the previous algorithms. The algorithm actually finds a minimum spanning forest on a given weighted, undirected graph, which is a problem that is at least as hard as CCOMP.

The novel idea is that concurrent processes are used to find sets of good edges in a multi-threaded manner. There are $\lfloor \log n \rfloor$ iteration stages in the algorithm; these are organized so that the computations required to obtain the set of edges for stage i start long before the computations for stage $i-1$ are finished. Thus, once stage $i-1$ is taken care of, stage i is handled quickly, and the total running time becomes $O(\log n)$.

5 Description of Some Significant Results

5.1 USTCON is solvable deterministically in $O(\log^{3/2} n)$ space

This section describes the deterministic, $O(\log^{3/2} n)$ space algorithm for USTCON by Nisan, Szemeredi, and Wigderson [17] mentioned earlier (Section 3.8).

The main idea, illustrated in Figure 2, is to:

- Construct connected clusters in G.
- Associate clusters in G with vertices in a new, smaller graph G'.
- Let two vertices in G' be connected iff $\exists$ an edge in G between corresponding clusters.
- Recursively compute $G', G'', \ldots$ until the answer to "Are s and t connected?" is obvious.

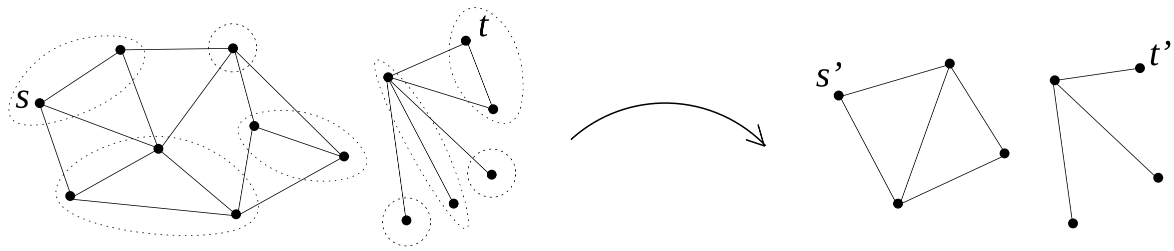

Figure 2: s is connected to t in G if and only if s' is connected to t' in G'.

The construction of G' from G is realized in a procedure called SHRINK, i.e., $G' = \text{SHRINK}(G)$. Storing intermediate results explicitly would take too much space; instead, they are recomputed many times. (Running time is sacrificed for space.) Therefore, the algorithm is not really going to be executed in the linear fashion outlined here, but in a zigzag pattern between recursion levels. Nevertheless, to keep the description simple, we focus on one recursion level, keeping in mind that each reference to an intermediate result is implemented by calls to previous levels.

SHRINK(G):

1. $\forall v \in V$, find $N(v) \subseteq V$ by walking along a k-universal traversal sequence σ starting at v.
2. - If $t \in N(s)$ or $s \in N(t)$, then output ("Connected") and halt.
   - Else, if $N(s) = Comp(s)$ or $N(t) = Comp(t)$, then output ("Not connected") and halt.
3. Choose $L \subseteq V$ according to a pairwise independent distribution such that $\forall v \in V$, $v \in L$ with probability $\frac{1}{6f}$. Then, $\forall v \in V$, define $rep_L(v)$:
   - If $N(v) = Comp(v)$, then $rep_L(v) = 0$.
   - Else, if $N(v) \cap L = \{\}$, then $rep_L(v) = v$.
   - Else, $rep_L(v) =$ lowest numbered vertex in $N(v) \cap L$.
4. $V' = \{rep_L(v) \mid v \in V\}$
5. Connect V' in accordance with the structure of G. $\Rightarrow E'$

Output: $G' = (V', E')$

Comments

1. $\sigma \in \{1,2,3\}^*$ is a *k-universal traversal sequence* if for every connected 3-regular k-vertex graph, every labeling of it, and every vertex v, the walk starting from v following σ visits all vertices. Note that any undirected graph can be turned into a labeled, 3-regular graph; see Figure 3.

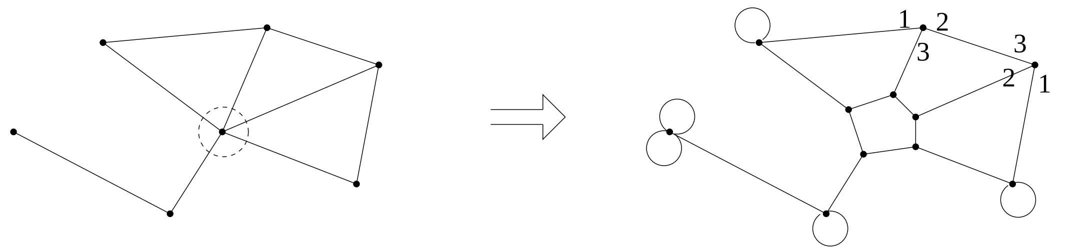

Figure 3: An undirected graph can be transformed into a 3-regular graph by replacing any vertex of degree $d > 3$ by d vertices in a ring, and adding self-loops to vertices of degree < 3. A labeling is then obtained by numbering the three exits from each vertex by {1,2,3} arbitrarily.

   $N(v)$, the *neighborhood* of v, is defined as the set of vertices actually encountered in the walk starting at v, plus all immediate neighbors of these vertices. The bigger the value of $|N(v)|$, the better; however, computing larger neighborhoods wastes more work space as longer universal traversal sequences must be generated. We need to find a compromise between the size of the neighborhoods and the amount of space used. The universal traversal sequence will be dimensioned to visit all nodes in a k-vertex graph, where $k < n$.

2. $Comp(v)$ is the set of vertices that are in the same connected component as v.
   This is how to check if $N(v) = Comp(v)$:
   Go through all vertices $u \in N(v)$; for each u, check that for all its neighbors w, $w \in N(v)$.

3. f is the *shrinking parameter*, which will be specified later on.

   Number the vertices of G, assigning '1' to s and '2' to t.
   L is a set of potential representative vertices, chosen according to a pairwise independent distribution. Only $O(\log n)$ bits are needed to index L in a probability space, so derandomizing the choosing of L is done by cycling through all possibilities in some fixed order, until either some choice yields a small enough V', or all possibilities have been tested. (Using a field F_q of size poly(n), each $L(a,b)$ is defined by two elements $a, b \in F_q$. Then, for every vertex v, let $v \in L(a,b)$ if and only if $a \cdot v + b \in \{1, \ldots, \frac{q}{6f}\}$.)

4. V' is the set of representative vertices (every representative vertex corresponds to a cluster). Hopefully, the neighborhoods will be large so that there will be many intersections with L, causing $|V'|$ to be small.

5. If $\exists$ edge (u,v) in E such that $rep_L(u) = u', rep_L(v) = v'$, and $u' \neq v'$, then $(u', v') \in E'$.

Analysis

If $|\,\text{SHRINK}(G)\,| \leq \frac{|V|}{f}$ for some fixed f and SHRINK(G) is computable in $s(n)$ space, then USTCON $\in DSPACE(s(n) \cdot \log_f(n))$, where $\log_f(n)$ is the worst-case recursion depth. We want the clusters to be larger than just some constant, since if all we could say was $f =$ constant, then we would get the weak result USTCON $\in DSPACE(\log^2 n)$, which we already know from Savitch's Theorem...

Let us first examine the neighborhoods obtained by the algorithm. Nisan's Theorem, together with Lemma 1, ensures us that they can be computed efficiently and that they are large:

Theorem (Nisan [15]) *For every k there exists a k-universal traversal sequence σ of length $k^{O(\log k)}$. Each entry σ_i of these sequences can be computed in $O(\log^2 k)$ space.*

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

(The proof of Lemma 1 is a simple proof by contradiction, and can be found in the article.)

Letting $k = 6^4 f^4$, we have $|N(v)| \geq 36f^2$ (or $N(v) = Comp(v)$) for every v.

This is used in the proof of the next lemma:

Lemma 2 *If L is chosen at random such that every vertex is in L with probability $\frac{1}{6f}$ and the choices are pairwise independent, then $Pr[|V'| \leq \frac{|V|}{f}] > 0$.*

(To prove Lemma 2, the authors distinguish between two types of vertices in V': $V' = L \cup B$, with $L \cap B = \{\}$, where $B = \{v \mid N(v) \cap L = \{\}$ and $|N(v)| \geq 36f^2\}$, and use the Markov and Chebyshev inequalities to show that $Pr[|L| \leq \frac{|V|}{2f}] \geq \frac{2}{3}$ and $Pr[|B| \leq \frac{|V|}{2f}] \geq \frac{2}{3}$.)

Derandomizing the selection of L then leads to a probabilistic proof that V' satisfies $|V'| \leq \frac{|V|}{f}$.

Now we are ready to assign a value to f and finish off the analysis.

Let $f = 2^{\sqrt{\log n}}$. Then $\log k = \log(6^4 f^4) = O(\log f) = O(\sqrt{\log n})$, and

$$s(n) = O(\text{number of bits for a pointer in the graph} + \text{space to compute the steps of } \sigma) = \\ = O(\log n + \log^2 k) = O(\log n).$$

Also, we have $\log_f(n) = \frac{\log n}{\log f} = \sqrt{\log n}$.

This implies that USTCON $\in DSPACE(s(n) \cdot \log_f(n)) = DSPACE(\log^{3/2} n)$.

5.2 USTCON is solvable deterministically in $O(\log^{4/3} n)$ space

This section describes the deterministic, $O(\log^{4/3} n)$ space algorithm for USTCON by Armoni, Ta-Shma, Wigderson, and Zhou [2] mentioned earlier (Section 3.10).

The algorithm is based on the shrinking scheme of Nisan, Szemeredi, and Wigderson (see the previous section of this paper). Armoni *et al.* observe that using Nisan's construction for universal traversal sequences to find neighborhoods only guarantees clusters of size $O(2^{\sqrt{\log n}})$, which implies $O(\sqrt{\log n})$ recursion levels, so they replace it. A key idea is to use a long random string h as input to all recursion levels, as well as short independent random strings $\overline{q}$ at different recursion levels, so that some random bits can be reused, thus saving space.

A graph G can be represented by its *adjacency matrix*, also denoted by G. In the adjacency matrix, $G[i, j]$ is equal to 1 if (i, j) is an edge in G, and 0 otherwise. A *neighborhood matrix* N of G is a matrix of the same dimension as G, in which all entries are real numbers between 0 and 1, such that $N[i, j] \neq 0$ if and only if i and j are connected in G. In the descriptions below, matrices are used to represent graphs and their neighborhoods.

The algorithm is built from several procedures; an overview of the dependencies between them is shown in Figure 4. Each procedure will be accounted for briefly.

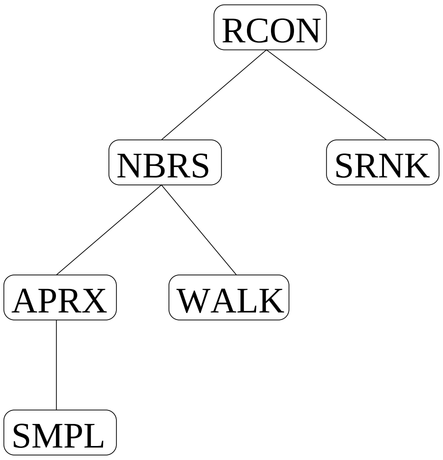

Figure 4: Components of the algorithm.

APRX and SMPL are general procedures for approximating a function in limited space, and may very well be useful for solving other problems than USTCON.

RCON

RCON is the main procedure. It relies upon NBRS to compute neighborhoods, and then SRNK to condense the graphs into smaller ones.
The same long random string h is passed to NBRS in every call, whereas each short random string $q(i)$ is passed only once.

Input: (G, s, t)

1. Compute parameters a, d, m, r.
   $a =$ Accuracy parameter passed to NBRS to control the accuracy of approximations
   $d =$ Perturbation parameter
   $m =$ Determines the shrinking parameter f via $f = 2^{m-1}$ (f is called k in the paper)
   $r =$ Number of recursive levels $= \log n/(m-1)$
2. Request off-line random input $h \in \{0,1\}^{O(m^2)}$ and $\overline{q} = [q(1), \ldots, q(r)] \in (\{0,1\}^d)^r$.
3. Let $G_0 = G$ and compute the sequence $[G_i]_{i=1}^r$ defined by $G_i = \text{SRNK}(G_{i-1}, s, t, N_i)$, where $N_i = \text{NBRS}(G_{i-1}, m, a, d; h, q(i))$.

Output: 1 if s and t are connected in G_r, and 0 otherwise.

NBRS

By applying APRX to WALK, the requirement "most h's are good for all vertices" is relaxed to "for every vertex, most h's are good".
It turns out that with high probability, NBRS still computes a neighborhood matrix in which every row contains enough non-zero entries.

Input: G, m, a, d
Off-line random input: $h \in \{0,1\}^{O(m^2)}$ and $q \in \{0,1\}^d$
Output: APRX(WALK,$(G, m), a, d; h, q$)

APRX

APRX is used to approximate $\mathbf{E}_y[B(x;y)] = \frac{1}{2^l} \sum_{y \in \{0,1\}^l} B(x;y)$, where $B : \{0,1\}^l \to [0,1]$, on given input x, to a given accuracy. APRX has about the same random bit complexity and processing space complexity as the function B, and with high probability, the outcome is independent of which h is used.

Input: B, x, a, d
Off-line random input: $h \in \{0,1\}^{3R(|x|)}$ and $q \in \{0,1\}^d$ ($R(|x|)$ is B's random bit complexity)
Output: $PerT(\text{SMPL}(B, x, a; h), q \cdot 2^{-a}, a - d)$, where $PerT$ performs a random (depending on q) perturbation and truncation on each entry of the matrix to eliminate dependence on h.

SMPL

The naive way to estimate $\mathbf{E}_y[B(x;y)]$ for a fixed x to a given accuracy is to evaluate B at a lot of independently chosen random sample points y. This uses way too many random bits, so instead, SMPL employs a special case of Zuckerman's randomness-efficient oblivious sampler, built from improved extractors, to stretch the long random string h into an even longer pseudorandom string $A(h)$, which is then used as the source of y's.

Whenever B needs a bit of y, the sampler A is run on h to get the bit, since storing all of y would take too much space.

Input: B, x, a
Off-line random input: $h \in \{0,1\}^{3l}$, where $l = 3R(|x|)$

1. Define $p = \lfloor (2l \cdot 2^a)^{c_0} \rfloor$, where c_0 is an absolute constant.
2. Let $A : \{0,1\}^{3l} \to \{0,1\}^{pl}$ be a $(\frac{1}{2^a}, \frac{1}{2^l})$-oblivious sampler.
3. $Sum =$ The K x K matrix with all 0 entries, where $K =$ the dimension of $B(x;y)$.
4. For $k = 1$ to p do $Sum = Sum + B(x; y = A(h)[k])$.

Output: $\frac{1}{p} Sum$

WALK

WALK finds a neighborhood matrix, and is designed so that each row of $\mathbf{E}_h[\text{WALK}(G, m; h)]$ has more than f entries that are greater than $\frac{1}{n}$ to assure stability against small perturbations.

F is a special pseudorandom generator used to dictate a walk on G with the property that if the connected component of the starting vertex has size c, then for all but $\frac{1}{c^5}$ of the possible h's, the probability (over x) that the walk visits a good vertex is greater than zero.

Input: G, m
Off-line random input: $h \in (\{0,1\}^{2p})^{\log l}$, where $p = O(m)$ and $l = O(f^3)$.

For every pair of vertices v_1, v_2 in G do

1. Set $result[v_1, v_2] = 0$.
2. For every $x \in \{0,1\}^p$ do
   if the walk on G that starts from v_1 and proceeds according to $F(x,h)$ passes through v_2 or a neighbor of v_2, then $result[v_1, v_2] = 1$.

Output: $result$

SRNK

SRNK is identical to steps 2. – 5. in Nisan, Szemeredi, and Wigderson's SHRINK procedure described in the previous section, with one adjustment: $N(v)$ is found by examining the entries of row v in the NBRS-matrix.

Analysis

The following two lemmas are central to the space complexity analysis of RCON.

Lemma 1 *RCON has random bit complexity $O(m^2+dr)$ and space complexity $O(r(a+\log n))$.*

(Proving Lemma 1 is done by showing that the space used when computing $\mathrm{SRNK}(G_{i-1}, s, t, N_i)$, where $N_i = \mathrm{NBRS}(G_{i-1}, m, a, d; h, q(i))$, is bounded by $O(a + \log n)$ for each i. The fact that RCON has the random bit complexity stated is immediate from RCON's description.)

Lemma 2 *If $a - d \geq \log n + 1$, then*
Pr[RCON($G, s, t; h, \overline{q}$) computes connectivity] $\geq 1 - rn^2(2^{-3m^2} + 2^{-d+1})$.

(Lemma 2 is proved by upper bounding the probability over randomly chosen $(h, \overline{q})$ that RCON is unsuccessful, i.e., by summing over i the probabilities that $\mathrm{NBRS}(G_{i-1}, m, a, d; h, q(i))$ either (1) contains rows with less than 2^{m-1} non-zero entries; or (2) depends on h.

The reason why RCON is successful with high probability lies in the application of APRX to WALK, which reduces the influence from any particular choice of h. Another lemma says that the approximation process doesn't mess things up; the neighborhood matrices are shown to be resilient to small perturbations, so the resulting neighborhoods will still be large.)

In addition, it is required that $m \leq \log n$ and $a = O(m^{3/2})$ are satisfied.

Carefully choosing values for the parameters, the authors find that setting
$a = 7 \log n$, $d = 4 \log n$, $m = \log^{2/3} n$, and thus $r = \Theta(\log^{1/3} n)$ gives an algorithm that

- Takes $O(\log^{4/3} n)$ random bits and runs in $O(\log^{4/3} n)$ space.
- Computes connectivity with probability $\geq 1 - \frac{1}{n}$.

Finally, since the algorithm is trivially derandomized (test all values for the $O(\log^{4/3} n)$ random bits by enumeration, reusing the work space), this shows that USTCON $\in DSPACE(\log^{4/3} n)$.

6 Conclusion

6.1 Summary of Important Results

The following table summarizes the currently best known results for STCON and USTCON:

STCON	**USTCON**
$\in L^2$ [19]	$\in L^{4/3}$ [2]
NL-complete under logspace reductions. [10, 11]	SL-complete under logspace reductions. [14]
Solvable deterministically in sublinear space and polynomial time. [3]	Solvable deterministically in polylogarithmic space and polynomial time. [16]

CCOMP: Solvable in $O(\log n)$ time on an EREW PRAM, using $n + m$ processors. [6]

6.2 Open Questions and Discussion

- MAJOR OPEN QUESTION: *Is $L = NL$?*

  Unfortunately, it does not seem as if anybody will settle this one for quite some time... Since STCON is NL-complete, $L = NL$ if and only if STCON is solvable by a deterministic logspace algorithm, so for somebody trying to prove $L \neq NL$, showing that STCON admits no such algorithm might be the way to go.

  The same comments hold for the analogous $L = SL$ question, but with USTCON being the problem that may separate the two classes.

- *Is USTCON complete for NL under logspace reductions?*

  If it is, then any result like USTCON $\in X$ would directly imply that $NL \subseteq X$. Specifically, finding a deterministic logspace algorithm for USTCON would then prove that $L = NL$.

- *What is the exact deterministic space complexity of STCON and USTCON?*

  Even if there is no deterministic logspace algorithm for USTCON, the current space bound is probably going to be improved in the future. The fraction $\frac{4}{3}$ in the exponent does not seem like a natural bound, perhaps $1 + \epsilon$ (with $\epsilon > 0$ being an arbitrarily small constant) is closer to the truth?

- *Why is there a such a big difference between the best results for STCON and USTCON?*

  Some algorithmic graph theory problems (e.g., constructing minimum spanning forests) appear to admit more efficient solutions for undirected graphs than for directed graphs. The seemingly fundamental difference between directed and undirected graphs is the symmetry inherent to undirected graphs. Most algorithms for USTCON make heavy use of the fact that the given graph is undirected, and therefore symmetric in the sense that if vertex a is connected to vertex b, then b is connected to a, and a and b can be replaced by a single vertex with the combined connectivity properties of a and b. The existing algorithms for directed graphs cannot do this, and instead have to enumerate all possible paths somehow, using as little space as possible.

Of course, the primary reason for the large gap could just be that people have put a lot more work into USTCON.

- *Are there any deterministic algorithms for STCON that use polylogarithmic space and run in polynomial time?*

  The approach by Barnes, Buss, Ruzzo, and Schieber [3] looks promising for further research. Their modified breadth-first search algorithm divides the graph into pieces which are subsequently treated by their short paths algorithm, but there is nothing saying that this partitioning the vertices of the graph is optimal; in other words, other methods may be a lot better. Also, since the short paths algorithm itself is a very general algorithm for directed connectivity (Savitch's algorithm is a special case of it, for example), any small improvement might give much stronger results right away.

- *What are some untried ideas for solving USTCON?*

  An idea that may or may not work is to substitute the concept of connected clusters in the algorithms in Section 5 with a more general notion of equivalence classes of vertices. Two vertices in the same equivalence class would have to belong to the same connected component, but not necessarily be immediate neighbors. The shrinking would proceed as before, so that at each level, some sufficiently large number of vertices in each equivalence class would collapse to a single representative vertex.

  This is similar to the new approach taken by Barnes, Buss, Ruzzo, and Schieber [3] for STCON, but in the case of undirected graphs, one might be able to use fancy tricks (fancier than breadth-first search, anyway!) such as letting every jth vertex encountered by a special traversal sequence starting at v belong to v's equivalence class, and so on. Again, some clever way of finding paths between different equivalence classes might be needed (or maybe not, since the graph is undirected).

- *Can the successful ideas behind the parallel algorithms for CCOMP be employed to solve USTCON in less than $O(\log^{4/3} n)$ deterministic space?*

  Needless to say, the two problems are closely related. The basic hook-and-contract scheme for CCOMP described in Section 4 resembles the graph shrinking method for USTCON. They both depend on the adequacy of the method used for choosing representative vertices in each iteration, so that the number of iterations will be small. Intuitively, what keeps a time- and processor-efficient CCOMP algorithm from being translated straightaway into a space-efficient USTCON algorithm is the fact that each processor must be able to point to some vertex in the graph, which would take a total of $O((n+m)\log n)$ space for $n+m$ processors working simultaneously.

  Historically, the best known results for the time complexity for CCOMP on the EREW PRAM have been comparable to the best known results for the space complexity of USTCON. Now that the quest for a time-optimal EREW PRAM algorithm for CCOMP has been completed, people hope for progress in the USTCON area. If history is allowed to repeat itself, the next advancement to look forward to should be an $O(\log n \log\log n)$ deterministic space USTCON algorithm...

References

[1] R. Aleliunas, R. Karp, R. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proc. 20th IEEE Symposium on Foundations of Computer Science (FOCS'79)*, pp. 218–223, 1979.

[2] R. Armoni, A. Ta-Shma, A. Wigderson, and S. Zhou. $SL \subseteq L^{4/3}$. In *Proc. 29th ACM Symposium on Theory of Computing (STOC'97)*, pp. 230–239, 1997.

[3] G. Barnes, J. Buss, W. Ruzzo, and B. Schieber. A Sublinear Space, Polynomial Time Algorithm for Directed s-t Connectivity. *SIAM Journal of Computing*, Vol. 27, No. 5, pp. 1273–1282, 1998.

[4] G. Barnes and W. Ruzzo. Deterministic Algorithms for Undirected s-t Connectivity Using Polynomial Time and Sublinear Space. In *Proc. 23rd ACM Symposium on Theory of Computing (STOC'91)*, pp. 43–53, 1991.

[5] A. Borodin, S. Cook, P. Dymond, W. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal of Computing*, Vol. 18, No. 3, pp. 559–578, 1989.

[6] K.W. Chong, Y. Han, and T.W. Lam. On the parallel time complexity of undirected connectivity and minimum spanning trees. To appear in *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, 1999.

[7] K.W. Chong and T.W. Lam. Finding Connected Components in $O(\log n \log\log n)$ time on the EREW PRAM. *Journal of Algorithms*, Vol. 18, No. 3, pp. 378–402, 1995.

[8] R. Cole and U. Vishkin. Approximate and exact parallel scheduling with applications to list, tree, and graph problems. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS'86)*, pp. 478–491, 1986.

[9] D. Johnson and P. Metaxas. Connected Components in $O(\lg^{3/2}|V|)$ Parallel Time for the CREW PRAM. In *Proc. 32nd IEEE Symposium on Foundations of Computer Science (FOCS'91)*, pp. 688–697, 1991.

[10] N. Jones. Space-Bounded Reducibility among Combinatorial Problems. *Journal of Computer and System Sciences*, Vol. 11, No. 1, pp. 68–85, 1975.

[11] N. Jones. Corrigendum: Space-Bounded Reducibility among Combinatorial Problems *Journal of Computer and System Sciences*, Vol. 15, No. 2, p. 241, 1977.

[12] D. Karger, N. Nisan, and M. Parnas. Fast Connected Components Algorithm for the EREW PRAM. In *Proc. 4th ACM Symposium on Parallel Algorithms and Architectures*, pp. 373-381, 1992.

[13] R. Karp and V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. In *Handbook of Theoretical Computer Science*, Vol. A, J. van Leeuwen (Ed.), pp. 869–932, 1990.

[14] H. Lewis and C. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, Vol. 19, No. 2, pp. 161–187, 1982.

[15] N. Nisan. Pseudorandom Generators for Space-Bounded Computation. In *Proc. 22nd ACM Symposium on Theory of Computing (STOC'90)*, pp. 204–212, 1990.

[16] N. Nisan. $RL \subseteq SC$. In *Proc. 24th ACM Symposium on Theory of Computing (STOC'92)*, pp. 619–623, 1992.

[17] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected Connectivity in $O(\log^{1.5} n)$ Space. In *Proc. 33th IEEE Symposium on Foundations of Computer Science (FOCS'92)*, pp. 24–29, 1992.

[18] M. Saks and S. Zhou. $RSPACE(S) \subseteq DSPACE(S^{\frac{3}{2}})$. In *Proc. 36th IEEE Symposium on Foundations of Computer Science (FOCS'95)*, pp. 344–353, 1995.

[19] W. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, Vol. 4, No. 2, pp. 177–192, 1970.

[20] Y. Shiloach and U. Vishkin. An $O(\log n)$ Parallel Connectivity Algorithm. *Journal of Algorithms*, Vol. 3, No. 1, pp. 57–67, 1982.