Graph Connectivity and its applications to Space-Bounded Computation

Bryce Sandlund

Abstract

We discuss the problem of st-connectivity in undirected and directed graphs. In particular, we focus on the result of Reingold that st-connectivity on an undirected graph is solvable in $O(\log n)$ space on a deterministic turing machine. Furthermore, we discuss the implications this may have to the more general problem of st-connectivity on a directed graph and the associated question of RL vs. L.

1 Introduction

One question in the study of derandomization is the tradeoff between two basic computational resources: space and randomness. That is, does randomness in computation allow for more efficient use of space? Central to this line of research is the question if RL = L; that is, if the class of languages decidable by a randomized turing machine with one-sided error in logarithmic space, and L, the class of languages decidable by a deterministic turing machine in logarithmic space, are the same. This has widely been conjectured to be true, in addition to the inclusion of BPL, the class of languages decidable by a randomized turing machine with two-sided error, into L. If this is the case, this implies that randomness does not add more power in the context of logarithmic space computations.

Progress in this area was made in 2004 when Reingold [1] showed that SL, the class of languages decidable by a non-deterministic, symmetric turing machine with logarithmic space [4], equals L. The inclusion SL $\subset$ L was made when the problem USTCON, st-connectivity in an undirected graph, known to be complete for SL, was shown to be solvable in deterministic logarithmic space. Note that a simple randomized algorithm for USTCON is to take a random walk starting at vertex s. The authors of [3] show that after polynomially many steps, with probability $\geq 2/3$, the node t has been visited. In general, it was previously known that L $\subset$ SL $\subset$ RL $\subset$ NL $\subset$ L^2, where NL is the class of languages solvable by a general non-deterministic turing machine in logarithmic space.

If the algorithm of [1] could be generalized to work in the case of directed graphs, known to be complete for the class NL, we would in fact have NL = L, and so RL = L would follow as a simple corollary. Note that at the moment, the inclusion RL $\subset$ NL has not been made, and that the above randomized algorithm fails for STCON, since a random walk starting at s may only visit t with exponentially small probability.

In [2], the authors attempt this adaptation and show that st-connectivity on a directed, eulerian graph is computable in deterministic log-space. Moreover, they show that it is possible to recover the st-path on a eulerian directed graph in logarithmic space. They use this to construct a "pseudorandom" walk generator on regular and consistently labelled directed graphs, and show that if the approach can be generalized to all regular directed graphs, then RL = L. To this end, the barrier behind showing RL = L can be reduced to the problem of finding a consistent labelling of a directed graph in deterministic log-space.

In this report, we study the algorithm given by Reingold [1] to show that undirected st-connectivity is in L. We then explore the results given in [2]. In the final section, intuition, ideas, and possible progress on solving the RL=L is given from the author's own work.

1.1 Background

We note a few observations about the classes L, SL, and NL and their respective complete problems. First, define the following two problems:

Definition 1.1.1 (STCONDAG)**.** We denote by STCONDAG the directed st-connectivity problem where the graph G is restricted to a directed, acyclic graph.

Definition 1.1.2 (STCONFOREST)**.** We denote by STCONFOREST the problem of st-connectivity where the graph G is a forest; that is, it is undirected, and it is the disjoint union of some number of trees.

It seems natural that these problems would be share many of the same properties as STCON and USTCON. Indeed:

Lemma 1.1.3. *STCONDAG is NL-Complete under* $\leq_m^{\log}$ *reduction.*

Proof. We first note that because STCON is NL-Complete, and STCONDAG is a restriction of STCON, it is clear STCONDAG is in NL. Now, recall the reduction from NL to STCON. We noted that for a machine that uses $s(n)$ space, its configuration graph is of size $2^{O(s(n))}$. Now, we know that if a machine halts, it must run in time the size of the configuration space, so again $2^{O(s(n))}$, otherwise, it would be repeating states. Combining these ideas, we can represent each state in the configuration space as a pair $(state, timestep)$. All state transistions are present, except that in each transition we must go from $(state, timestep)$ to $(state', timestep+1)$. The cost of making this modification blows up the graph to size $2^{O(s(n))} \cdot 2^{O(s(n))} = 2^{O(s(n)}$. We therefore see the blowup does not matter, and the resulting configuration graph will not contain any cycles. Furthermore, this transformation is still computable in log-space. From this point the traditional reduction from NL to STCON can be followed. □

However, since it is still true that STCON is NL-Complete, this implies that there is no difference in the problems of STCONDAG and STCON in computational difficulty, which implies a more direct way to show Lemma 1.1.3. In fact, there is one:

Proof. Given a directed graph G of size n and two vertices s and t, first create a graph G' by making n copies of every vertex u in G as the new vertex (u, i) in G', where i goes from 1 to n for the respective copies. Then, represent each edge (u, v) in G' by the transition $(u, i) \mapsto (v, i + 1)$. Since any path from s to t is of length at most n, it then follows that a path from s to t exists in G if and only if it exists in G'. Furthermore, since i is always increasing in (u, i), it is clear G' is a DAG. Finally, the mapping can be computed in logarithmic space. Then, since STCON is complete for NL, it follows that STCONDAG is also complete for NL. □

This shows an interesting observation, that is, cycles do not matter for STCON. Now, consider the problem STCONFOREST; a similar application for undirected graphs. We note the following:

Lemma 1.1.4. *STCONFOREST* $\in L$.

Proof. Although the result by Reingold [1] shows STCONFOREST is in L, there is a simpler maze-traversal algorithm that we note here. Starting at s, keep track of the current node u and the most recently visited node r. When a vertex u is reached, take the edge that points to the next lexicographical neighbor after r. If no neighbors of u are lexicographically larger than r, wrap back around and take the smallest lexicographical neighbor. Since G is a tree, you only ever visit a node v for a second time from the same edge you most recently took to leave it. Because of this, it follows that you will explore all neighbors of u. Since this will happen for every node in the tree, after traversing each edge twice you will have visited t if and only if a path from s to t exists. We note that this relies on an ordering of the vertices in G. However, since each vertex of G is necessarily described by some number of bits, this ordering can be assumed by comparing the bit representations of each vertex of G. Furthermore, since keeping track of u and v requires only $O(\log n)$ bits, the algorithm runs in $O(\log n)$ space. □

In our understanding, the above algorithm has been known for some time. If it was complete for SL, then the result of Reingold would not have been the first to show SL = L. Unfortunately, we are not sure why the direct argument from Lemma 1.1.3 cannot be used to show STCONFOREST is SL-Hard. In particular, it may be that STCONFOREST is not in SL, because the direct approach listed above is not obviously computable on an SL machine, but the algorithm certainly can be run on a deterministic log-space machine, and as stated in [4], $DSPACE(s(n)) \subset SSPACE(s(n))$. We do note, however, that the algorithm described above does not obviously generalize to general undirected graphs, nor does it seem to work for any directed graph, so it seems reasonable to think STCONFOREST is easier than the classes SL or NL. However, this means that there is likely an error in the algorithm of Lemma 1.1.4, the direct reduction in the first proof of Lemma 1.1.3, or some other complication when dealing with symmetric turing machines.

2 Undirected st-connectivity in Log-Space

The basic intuition behind Reingold's [1] algorithm for solving USTCON in deterministic log-space is quite simple. The graph is transformed from a general undirected graph to a constant degree expander, with the invariant that connectivity is preserved in the new graph. Since expander graphs have logarithmic diameter, after exploring all paths of length $O(\log n)$ from s, if t is not found, s is not connected to t, and obviously if t is found, s and t are connected. The author shows this transformation can be computed in log-space, and furthermore that the search procedure can be conducted in logarithmic space.

Along this latter point, we give the following analysis. If each vertex has constant degree D, it takes only $O(\log D)$ bits to store a counter at each depth of the search procedure. Since D is a constant, this is just $O(1)$. Furthermore, since the depth is $O(\log n)$, it is then clear you only need $O(\log n)$ space to exhaust all paths of length $O(\log n)$ from s.

We will now argue step by step how to transform G into a constant degree expander. The first step of graph transformation involves turning G into a regular graph of constant degree D. One very simple way to do this is to make each edge into two vertices. Then for some fixed vertex u in G, for all edges (u, v) in G, connect the vertices created on the u side by this procedure together in a cycle. The resulting graph G' is regular of degree 3. Since we will later require D to be an arbitrary constant, not necessarily just 3, we then add self loops to the vertices of G' until all vertices have the desired degree.

Now, to convert the regular graph G' into a constant degree expander, there are two variables we will have to balance: the degree of the graph and the expansion properties of the graph. To measure the latter, a good metric is the spectral gap of G'. The spectral gap of a graph G is defined as $1 - \lambda(G)$, where $\lambda(G)$ is the second largest eigenvalue in the adjacency matrix of G, which, for a regular graph is necessarily less than 1. Useful to this algorithm, the spectral gap of a regular graph G is at least $1/poly$. More explicitly, we have:

Lemma 2.0.5. *For every D-regular, connected, non-bipartite graph G of size n it holds that:*

$$\lambda(G) \leq 1 - 1/(Dn^2)$$

Furthermore, if we can remove the dependence on n from $\lambda(G)$, effectively making it a constant, we have:

Lemma 2.0.6. *Given a regular graph G of degree D on n vertices, if $\lambda(G) = c$ for some c independent of n, then there exists a path of length $O(\log n)$ from any two vertices s and t in G.*

One way to decrease $\lambda(G)$ and thus increase the spectral gap is by taking powers of G. Formally, the cth power of G is defined by the graph on G where edges are replaced by all paths of length c in G. In this context, multiple edges and self-loops are allowed. Note that this operation can be computed by simply taking the adjacency matrix to the cth power. In this respect, it then follows:

Lemma 2.0.7. *If G is a D-regular graph with second largest eigenvalue λ, then G^t is a D^t regular graph with second largeset eigenvalue λ^t.*

While taking the power of a graph helps its expansionary properties, taking only graph powers would result in a graph with larger than constant degree. To combat this, we then would like a complementary procedure that reduces the degree of G without harming $\lambda(G)$ too much. Two such procedures are given below:

Definition 2.0.8 (Replacement Product)**.** Given a D-regular graph G on n vertices and a d-regular graph H on D vertices, the replacement product of G and H can be constructed by first replacing each vertex of G with H. In this sense, the vertices of H now represent the edges incident on a vertex v of G. Following this intuition, for each edge (u, v) of G, connect the corresponding vertex of H_u with H_v.

Definition 2.0.9 (Zig-Zag Product)**.** Given a D-regular graph G on n vertices and a d-regular graph H on D vertices, the replacement product of G and H can be constructed by first taking the replacement product of G and H. Then, enumerate all paths of length 3 that consist of an edge in H, followed by an edge in G, and then by another edge in H. All paths of this form will define the edges in the zig-zag product of G and H.

Note that in the replacement product, the resulting graph has degree $d+1$. Furthermore, in the zig-zag product, the path of length 3 involves choosing d vertices for the first step, a single vertex for the second, and another d for the third. It then follows the resulting graph has degree d^2.

The author of [1] notes that both the replacement product and the zig-zag product should be enough to show undirected st-connectivity in log-space, but he mentions that the analysis for the zig-zag product is simpler. With this in mind, we note how $\lambda(G)$ changes when you take the zig-zag product of G with a graph H, which will be specified later:

Theorem 2.0.10. *If G is a D-regular graph on n vertices with second largest eigenvalue λ, and H is a d-regular graph on D vertices with second largest eigenvalue λ, then the zig zag product of G and H is a d^2-regular graph on $n * D$ vertices with second largest eigenvalue equal to:*

$$f(\lambda, \alpha) = \frac{1}{2}(1-\alpha^2)\lambda + \frac{1}{2}\sqrt{(1-\alpha^2)^2\lambda^2 + 4\alpha^2}$$

Since $\lambda < 1$, after some algebraic manipulation we have the following corollary:

Corollary 2.0.11. *It follows the second largest eighenvalue of the zig zag product of G and H satisfies:*

$$f(\lambda, \alpha) \leq 1 - \frac{1}{2}(1-\alpha^2)(1-\lambda)$$

Thus, if $\lambda(H)$ is some constant, we see $\lambda(G)$ decreases only be a constant factor. This is accomplished by selecting H to be an expander with size D^{16}, degree D, and $\lambda(H) = 1/2$. The author sites relevant literature to show an expander described as above exists (and thus can be found in a brute-force simulation) for some constant D.

The final step in the algorithm is an explicit calculation to create the constant degree expander G. This is noted in the following transformation:

- Set $\ell = 2\lceil \log Dn^2 \rceil$.
- Set $G_0 = G$, and recursively define $G_i = (G_{i-1}\ ZZ\ H)^8$, where ZZ denotes the zig-zag product.

Theorem 2.0.12. *We then have $\lambda(G_\ell) \leq 1/2$, with G_ℓ constant degree.*

The remainder of the argument relies on showing the transformation can be computed in $O(\log n)$ space. Finally, the author shows that the path from s to t can also be recovered in $O(\log n)$ space. This has further applications as explained in the following section.

2.1 Universal Traversal Sequences and RL vs. L

In a follow up work, and also stated in [1], Reingold et. al state that the above algorithm can be generalized to work on directed eulerian graphs [2]. Here, a directed eulerian graph is defined as a directed graph with in-degree equal to out-degree at every vertex u. Somewhat unobviously is the following lemma:

Lemma 2.1.1. *STCON on a (directed) eulerian graph is $\leq_m^{\log}$ to USTCON*

The reasoning for the above is intuitive. In a eulerian graph, a eulerian circuit exists such that you can visit each edge exactly once. Therefore, a path exists such that you can visit every node in a connected component. Because of this, the edge directions are insignficant, and can be solved by USTCON.

Less obvious is how to reconstruct the path taken by the log-space algorithm described in the previous section. Nevertheless, Reingold et. al [3] show that it can be done.

Before continuing, we define the following:

Definition 2.1.2 (Universal Traversal Sequence)**.** A universal traversal sequence, defined on a D-regular graph of size n, is a sequence of labels in the set $\{1, ..., n\}$ such that for every graph G of size n, for every labeling of its edges, and for every start vertex, the sequence visits every vertex in G.

Polynomially long universal traversal sequences have been shown to exist in regular graphs. However, to be of use for a connectivity algorithm, such a sequence must also be constructible in $O(\log n)$. Reingold et. al [2] show that their algorithm for eulerian (regular) st-connectivity implies a universal traversal sequence constructible in space $O(\log n)$, given that the graph is consistently labelled. This requires another definition:

Definition 2.1.3. A d-regular digraph G is consistently labelled if for every vertex $u \in G$, the edges incident to u, both incoming and outgoing, are labelled from the 1 to d with no duplicates. Each edge in G can have only one such label.

In [2], the authors show that if their approach to a pseudorandom walk generator to produce the universal traversal sequences can be generalized to handle arbitrary edge labels, then L = RL. They do so by showing a new complete promise problem for RL. This problem takes as input a *regular* directed graph, and determines whether a random walk approaches a stationary distribution with some given dependence on s and t. This shows that regularity can be assumed for the directed graph case, but the consistent labelling remains a barrier to show RL = L.

In our opinion, this is the best direction for future research to try to show RL = L. Universal traversal sequences are a combinatorial object that may be able to solved, or shown to be hard.

References

[1] Omer Reingold. "Undirected Connectivity in Log-Space," in *Journal of the ACM (JACM)*. Volume 55 Issue 4, September 2008.

[2] O. Reingold, L. Trevisan, S. Vadhan. "Pseudorandom Walks on Regular Digraphs and the RL vs. L Problem," in *Symposium on the Theory of Computation, STOC*. 2006.

[3] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, and C. Rackoff. "Random walks, universal traversal sequences, and the complexity of maze problems," in *20th Annual Symposium on Foundations of Computer Science*, pages 218-223, Oct. 1970.

[4] H. R. Lewis. C. H. Papadimitriou. "Symmetric space-bounded computation," in *Proceedings of the Seventh International Colloquium on Autamata, Languages and Programming*. Lecture Notes in Computer Science 85, pp. 347-384, 1980.