CS369E: Expanders May 16, 2005

Lecture 8: Undirected Connectivity is in logspace

Lecturer: Prahladh Harsha *Scribe: Cynthia Dwork & Prahladh Harsha*

In the second half of today's lecture, we will discuss a deterministic logspace algorithm for undirected connectivity, a recent and beautiful result due to Omer Reingold [Rei]. In fact, Reingold's algorithm is one of the reasons this course is being offered this quarter.

8.1 Undirected S-T Connectivity

The undirected s-t connectivity problem is the problem of finding if there exists a path between two specified vertices in a given undirected graph. More formally, the problem is as follows:
Input: A undirected graph $G = (V, E)$ and two vertices $s, t \in V$ (s denotes source and t target).
Problem: Are s and t connected? I.e., does there exist a path in G from the source s to the target t?

$$\text{USTCONN} = \{\langle G, s, t\rangle \mid G\text{– undirected graph}, s, t \in V(G); s \text{ and } t \text{ are connected in } G\}.$$

Clearly, any of the standard search algorithms (depth-first-search, breadth-first-search etc.) solve USTCONN in linear time. Thus, the time complexity of USTCONN is well-understood. What we would be interested in today's lecture is the same complexity of USTCONN. It is to be noted that the standard search algorithms perform poorly with respect to space (this is because their implementation requires a stack or queue which in the worst case could be as large as the graph). The question we will concern ourselves is the following: Is USTCONN in Logspace. In other words, does there exist a deterministic logspace algorithm that can decide connectivity in an undirected graph. Reingold resolved this question positively and gave a logspace algorithm for USTCONN using the zig-zag product. Before that, we will briefly look at the history of USTCONN.

8.1.1 History of Space Complexity of USTCONN

USTCONN is in NL. In fact, the directed counterpart of USTCONNis the complete problem for non-deterministic logspace. In 1970, Savitch demonstrated [Sav] a simulation of a non-deterministic space S machine by a deterministic space S^2 machine. Thus, USTCONN $\in$ SPACE($\log^2 n$). In one of the initial lectures of this course on random walks (Lecture 3), we saw a randomized logspace algorithm for USTCONN due to Aleliunas et. al. [AKLLR]. Thus, USTCONN $\in$ RL (RL denotes randomized logspace). Saks and Zhou, in 1995, then showed that any randomized space S machine can be simulated by a deterministic space $S^{3/2}$ machine [SZ]. Putting both these results together, we have USTCONN $\in$ SPACE($\log^{3/2} n$). Later, in 1997, Armoni, Ta-Shma, Wigderson and Zhou improved this deterministic simulation to give a $O(\log^{4/3} n)$-space algorithm for USTCONN. The status of this problem has been open since then till it was resolved recently due to Reingold. Note that $\log n$ space is required to even index a vertex in the graph.

8.2 Savitch's Deterministic Simulation

To begin with, we will look at Savitch's algorithm for USTCONN. The main idea in Savitch's algorithm is that squaring improves connectivity. More formally, for any graph G, define G^{sq} to be the graph on the same set of vertices as G, but has an edge between any two vertices u, v in $V(G)$ if there exists a path of length at most two between u and v in the original graph G. Note this is not the same as the more natural G^2 which corresponds to the graph where there are as many edges between u and v as the number of walks of length exactly two between u and v in G. A simple observation reveals that if u and v are connected in G, then (u, v) is an edge in $G^{\mathrm{sq}^{\log n}}$. Savitch gave a $O(\log^2 n)$ algorithm that computes the graph $G^{\mathrm{sq}^{\log n}}$ from G, thus proving USTCONN $\in$ SPACE($\log^2 n$).

We will now perform Savitch's algorithm with the more natural G^2 instead of G^{sq}. Though, this will also solve USTCONN, this will not give us another proof of Savitch's Theorem. In fact, the space complexity of computing G^{2^n} is huge. However, this exercise will be illuminating and will lead us towards Reingold's algorithm.

For the purpose of this discussion, we will assume all graphs are regular. Before discussing the algorithm for G^{2^n}, we need to indicate which representation of the graph we use. For reasons that will become clear later, we will use the rotation map representation introduced while talking about the zig-zag product. If the graph G is d-regular, then the rotation map Rot_G is the permutation that maps $(u, i) \in V \times [d]$ to $(v, j) \in V \times [d]$ if the i^{th} edge from u leads to v and the label of this edge with respect to v is j. Note, $\mathrm{Rot}_G(\mathrm{Rot}_G(u, i)) = (u, i)$ for all (u, i). We can compute this representation from the adjacency matrix and list in $O(\log n)$ space.

Let us first calculate the space-complexity of computing the rotation map of H^2 given the rotation map of H. Let H be a d-regular graph, then H^2 is a d^2 regular graph. The edge labels of H^2 can be assumed to be from the set $[d]\times[d]$. How do we compute $\mathrm{Rot}_{H^2}(u, (i_1, i_2))$ efficiently in space? To start with we assume that the tape contains $(u, (i_1, i_2))$ and at the end of the computation, we would like this to be replaced by $\mathrm{Rot}_{H^2}(u, (i_1, i_2))$. For a graph G, let SPACE(G) denote the additional space required to replace the input (u, i) on the tape with $\mathrm{Rot}_G(u, i)$. We first compute $\mathrm{Rot}_H(u, i_1) = (w, j_2)$ and replace the tape contents $(u, (i_1, i_2))$ with $(w, (j_2, i_2))$. This requires an additional space for computing Rot_H, i.e., SPACE(H). In the second step, we then reuse this extra space to compute $\mathrm{Rot}_H(w, i_2) = (v, j_1)$ and replace the tape contents $(w, (j_2, i_2))$ with $(v, (j_2, j_1))$. Finally, we swap the indices j_1 and j_2 in the tape to obtain $(v, (j_1, j_2))$ which is in fact $\mathrm{Rot}_{H^2}(u, (i_1, i_2))$. A analysis of the above shows that

$$\mathrm{SPACE}(H^2) = \mathrm{SPACE}(H) + O(\log \deg(H)).$$

Performing the above procedure $O(\log n)$ times, we can compute the rotation map of $G^n = G^{2^{\log n}}$, thus solving USTCONN. The space complexity of this algorithm is given by the following:

$$\begin{aligned}
\text{SPACE}(G^n) &= \text{SPACE}(G^{2^{\log n}}) \\
&= \text{SPACE}(G^{2^{\log n-1}}) + O(\log \deg(G^{2^{\log n-1}})) \\
&\vdots \\
&= \text{SPACE}(G) + O(\log \deg G) + O(\log \deg G^2) + \cdots + O(\log \deg G^{2^{\log n-1}}) \\
&= \sum_{i=1}^{\log n-1} O(\log \deg G^{2^i})
\end{aligned}$$

The reason this is a bad algorithm for solving USTCONN is because the degree of G^{2^i} grows prohibitively large with i, in fact $\deg(G^{2^i}) = (\deg(G))^{2^i}$. If somehow we could keep the degree constant through out the process, then in fact the above procedure would give a $O(\log n)$ space algorithm for USTCONN. But this is not possible, squaring a graph will also square the degree. Is it possible to obtain the same effect as squaring without actually increasing the degree of the graph? The main advantage of squaring is that it improves the expansion of the graph (and thus the connectivity of the graph). The crucial idea in Reingold's algorithm is that the zig-zag product (discussed in the first half of today's lecture) can be used to decrease the degree without altering the expansion of the graph by too much. Reingold's algorithm thus alternates between squaring and zig-zag to improve the expansion of the graph (via squaring) while not increasing the degree.

8.3 Zig-Zag Product

We say that a graph G is a (N, d, λ)-graph if G is a d-regular graph on N vertices and the spectral expansion of G is at most λ. Let us quickly recall the definition of the zig-zag product.

Definition 8.1 *The zig-zag product between rotation map representations of two graphs G, a (N, D_1, λ_1)-graph and H, a (D_1, D_2, λ_2)-graph, is a rotation map representation of a graph, denoted by $G \text{ⓩ} H$. The graph $G \text{ⓩ} H$ and its rotation map are defined as below.*

1. *$G \text{ⓩ} H$ has ND_1 vertices.*
2. *$G \text{ⓩ} H$ is a D_2^2-regular graph.*
3. *$\text{Rot}_{G \text{ⓩ} H}((u, i), (a_1, a_2)) = ((v, j), (b_1, b_2))$ if the following is satisfied: There exist $i', j' \in [D_2]$ such that*
   - $\text{Rot}_H(i, a_1) = (i', b_2)$
   - $\text{Rot}_G(u, i') = (v, j')$
   - $\text{Rot}_H(j', a_2) = (j, b_1)$

We proved the following result in the earlier lecture on zig-zag products.

Theorem 8.2 *Suppose G is an (N_1, d_1, λ_1)-expander and H is a (d_1, d_2, λ_2)-expander. Then $G \circled{z} H$ is an $(N_1 d_1, d_2^2, f(\lambda_1, \lambda_2))$-expander, where $f(\lambda_1, \lambda_2) \le \lambda_1 + \lambda_2 + \lambda_2^2$.*

Note that the above result is useful only when both the graphs G and H have fairly good expansion to start with. For our case, all we know is that the original graph, if connected and non-bipartite, has spectral expansion bounded away from 1 by at least a inverse polynomial. In fact, we proved the following result in Lecture 3.

Lemma 8.3 *If G is a connected, d-regular, non-bipartite graph on n vertices, then*

$$1 - \lambda \ge \frac{1}{dn^2}.$$

The zig-zag product is useful even in this case as long as the other graph H has good spectral expansion. We will use the following result (which we will not prove in class) on zig-zag products for this purpose.

Lemma 8.4 *If $\lambda(H) \le 1/2$, then*

$$1 - \lambda(G \circled{z} H) \le \frac{1}{3}\left(1 - \lambda(G)\right).$$

We mention that the zig-zag product is not the only graph product that satisfies such properties. The replacement product would have sufficed for our purposes (see [MR1, MR2]). However, we use the zig-zag product since we are already familiar with it from the previous lecture. A proof of Lemma 8.4 can be found in [RVW] while a proof of a similar statement for the replacement product can be found in Martin and Randall [MR2].

8.4 Reingold's Algorithm

As mentioned in the previous sections, the main idea in Reingold's algorithm is to alternate squaring with zig-zag product (with a constant sized expander). Lemma 8.4 tells us the following: as long as H is a good expander (i.e., $\lambda(H) \le 1/2$), zig-zagging G with H only reduces the spectral gap (i.e., $1 - \lambda$) by a factor of three (3). This is good for us, since squaring improves the spectral gap $1 - \lambda$ from to $1 - \lambda^2$ while zig-zag product deteriorates the spectral gap from $1 - \lambda$ to $1 - \lambda/3$. Thus, we could alternate a couple of squarings with a zig-zag product to reduce the spectral gap by a factor of two while keeping the degree constant.

We are now ready to describe Reingold's algorithm.

let H be a $(d^{16}, d, 1/2)$-graph for some constant G. Such a graph H can be found either by exhaustive search or by using one of the expander constructions (described earlier in the course). For this section, we will assume that the input graph G for which we need to check (s, t) connectivity is a d^{16}-regular non-bipartite graph. We will later remove these restrictions on G.

Furthermore, we will assume G is a connected graph. Actually, this is a stupid assumption since if G were indeed connected, then there is nothing to prove. What we actually mean is the following: Reingold's algorithm works independently for each connected component of the graph and checks if t exists in the connected component that contains s. Since every component of G is d^{16}-regular, connected and non-bipartite, we have from Lemma 8.3 that $1 - \lambda(C) \ge 1/d^{16}n^2$, for all components C of G.

Checking connectivity on an expander We first argue that checking connectivity on a graph, each of whose connected components is an expander (i.e., $\lambda \leq 1/2$) can be done in logspace. This follows from the simple observation that in an expander, the distance between any two vertices in $O(\log n)$. Thus, it suffices to enumerate all $O(\log n)$ paths in the graph originating at s and check if any of them lead to t. This can be done in logspace. Thus, it suffices for us to convert G into another G' in logspace such that each connected component of G' is an expander (i.e., $\lambda \leq 1/2$) and furthermore, two vertices are connected in G iff they are connected in G' (i.e., the transformation does not alter the connectivity of the graph).

Reingold's Algorithm

Input: G – d^{16}-regular graph and two vertices $s, t \in V(G)$.

1. Set l to be the smallest integer such that $\left(1 - \frac{1}{d^{16}n^2}\right)^{2^l} \leq \frac{1}{2}$.
    Comment: l is $O(\log n)$
2. Set $G_0 \leftarrow G$.
3. For $i = 1, \ldots, l$ do, set $G_i \leftarrow (G_{i-1} \textcircled{z} H)^8$.
    Comment: (1) Each G_i is a d^{16}-regular graph.
    (2) Each connected component of G_l is an expander with spectral expansion at most $1/2$ (see Theorem 8.6)
4. Check if s and t are connected in G_l by enumerating over all $O(\log n)$ paths originating at s.

We first prove the comment in Step 3, which will suffice to prove the correctness of Reingold's algorithm. For this we need the following proposition.

Proposition 8.5 *For $i = 1, \ldots, l$, $\lambda(G_i) \leq \min\{\lambda^2(G_{i-1}), 1/2\}$.*

Proof: Since $G_i = (G \textcircled{z} H)^8$, we have from Lemma 8.4 that $\lambda(G_i) = \lambda^8(G_{i-1} \textcircled{z} H) \leq [1 - (1 - \lambda(G_{i-1}))/3]^8$. Now, consider the following two cases.

Case (i): $\lambda(G_{i-1}) \leq 1/2$. Then,

$$\lambda(G_i) = (\lambda(G_{i-1} \textcircled{z} H))^8 \leq \left(1 - \frac{1}{3} \cdot \frac{1}{2}\right)^8 = \left(\frac{5}{6}\right)^8 < \frac{1}{2}.$$

Case (ii): $\lambda(G_{i-1}) > 1/2$. In this case, we can by expansion check that

$$\left(1 - \frac{1}{3}(1 - x)\right)^4 \leq x, \qquad \text{for all } \frac{1}{2} \leq x \leq 1$$

Hence,

$$\lambda(G_i) = (\lambda(G_{i-1} \textcircled{z} H))^8 \leq \left(1 - \frac{1}{3}(1 - \lambda(G_{i-1}))\right)^8 \leq \lambda^2(G_{i-1}).$$

■

By our choice of l, we have the following theorem on the expansion of each connected component of G_l.

Theorem 8.6 *The spectral expansion of each connected component of G_l is at most $1/2$.*

Space Complexity of Reingold's Algorithm We noted before that each squaring operation requires an additional space of $O(\log \deg G)$. Similarly, it can be shown that each zig-zag product with H (of constant size) also requires additional space at most $O(\log \deg G)$. Since there are at most $O(\log n)$ squaring and zig-zag products (since $l = O(\log n)$) and the degree of all the graphs is at most d^{16}, a constant, the total space complexity of the algorithm is at most $O(\log n)$.

Handling non-regular bipartite graph To start with, we will convert the graph into a 3-regular graph by replacing each vertex of degree d greater than 3 by a cycle of size d and connecting each of the d neighbors of the vertex to the d distinct points on the circle. To convert the graph into a d^{16}-regular graph, we then add $d^{16} - 3$ self loops to each vertex. Note, the addition of self loops also makes the graph non-bipartite. Both these conversions can be effected in log space.

References

[AKLLR] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, László Lovász, Charles Rackoff: "Random Walks, Universal Traversal Sequences, and the Complexity of Maze Problems". FOCS 1979: 218–223

[ATWZ] Roy Armoni, Amnon Ta-Shma, Avi Wigderson, Shiyu Zhou: "An $O\left(\log^{4/3} n\right)$ space algorithm for (s,t) connectivity in undirected graphs". J. ACM 47(2): 294–311 (2000).

[MR1] Neal Madras, Dana Randall: "Factoring Graphs to Bound Mixing Rates". FOCS 1996: 194–203

[MR2] Russell A. Martin, Dana Randall: "Sampling Adsorbing Staircase Walks Using a New Markov Chain Decomposition Method". FOCS 2000: 492–502

[Rei] Omer Reingold: "Undirected ST-connectivity in log-space". STOC 2005: 376–385 (See also ECCC Tech Report TR04–094, 2004).

[RVW] Omer Reingold, Salil Vadhan, and Avi Wigderson: "Entropy Waves, The Zig-Zag Graph Product, and New Constant-Degree Expanders and Extractors", Annals of Math 155: 157–187, 2002.

[SZ] Michael E. Saks, Shiyu Zhou: "$\mathrm{BP}_H\mathrm{Space}(S) \subseteq \mathrm{DSPACE}(S^{3/2})$". J. Comput. Syst. Sci. 58(2): 376–403 (1999)

[Sav] Walter J. Savitch: "Relationships Between Nondeterministic and Deterministic Tape Complexities". J. Comput. Syst. Sci. 4(2): 177–192 (1970).