Reingold's algorithm

November 13, 2026

1 Reingold's algorithm for connectivity in Logspace

We need to check whether in an undirected graph, there is a path which exists from a given vertex s to another given vertex t. Algorithms such as depth-first search which takes linear time in the size of the graph unfortunately use linear memory to store the paths being explored. Reingold's algorithm shows that we can solve undirected connectivity in Logspace - *i.e.* using only logarithmic memory in the size of the graph.

The main idea is to first notice that connectivity in expander graphs is quite easy to check : expanders have very small diameter, hence by taking a short number of random steps from s, with positive probability, we can reach t.

Now, we have to accomplish three things - first, how do we deal with arbitrary graphs which are not expanders? second, can we derandomize the algorithm? third, can we keep track of the random walks in logspace? Reingold's algorithm accomplishes all three.

We use the approach in Arora-Barak, which uses the replacement product.

2 The algorithm

The main idea of the algorithm is to transform the given graph G into another graph G' - this transformation itself must be logspace computable - such that every connected component in G' is an expander, but vertices in G which were not connected by any path in G will stay unconnected in G'.

Add more self-loops to ensure that the input G has degree d^{50} for some large d sufficient to ensure that there is $(d^50, d/2, 0.01)$-expander graph H. We store all of H in memory - this takes $O(1)$ space since H is fixed.

We now perform a recursive construction. Let $G_0 = G$ and for $k \geq 1$, define $G_k = (G \circ_R H)^{50}$, where $\circ_R$ is the replacement product.

Inductively, assume that G_{k-1} is an N_{k-1}-vertex graph with degree d^{50}. We can show that G_j is a (d^50N_{k-1})-vertex graph with degree d. Note also that replacement product and the graph product preserves connectivity exactly in the transformed vertices. We have the following lemma.

Lemma 2.1. *For every $k \geq 0$, every connected component in G_k is an $(d^50k, d^20, 1-\epsilon)$-expander, where $\epsilon = \min\{1/20, 1.5^k/(12n^2)\}$ and n denotes the number of vertices in $G = G_0$.*

Thus, $G_{10\log n}$ is an expander with $\lambda(G_{10\log n}) \leq 1 - \frac{1}{20}$. Now, we enumerate over all paths in $G_{10\log n}$ that start at s and has length $\ell = \mathrm{O}(\log n)$, and see if any of them hit t. We can perform this enumeration by simply concatenating edge labels for paths of that length, and checking whether any of them reach t.

We now argue that this can be done in logspace. The main question is about performing the replacement product in logspace.

We show that we can take a single step in $G_{k-1} \circ_R H$ in logspace. We can then repeat the same process 50 times to take a step in $(G_{k-1} \circ_R H)^{50}$.

A vertex in $G_{k-1} \circ_R H$ is a pair $\langle u, v\rangle$ of vertices where $u \in V(G_{k-1})$ and $v \in V(H)$. The index of a neighbor of $\langle u, v\rangle$ is a pair $\langle b, i\rangle$ where b is either 0 or 1, and $1 \leq i \leq d/2$. If $b = 0$, then the $\langle b, i\rangle$th neighbor of $\langle u, v\rangle$ is a vertex $\langle u, v'\rangle$ where v' is the ith neighbor of v in H - *i.e.* this is an "internal edge" inside H. Otherwise, this neighbor is $\langle u', v'\rangle$ denoting the result of applying the rotation map of G_{k-1} to $\langle u, v\rangle$. Thus there is a recursive algorithm to compute the rotation map of G_k. Let s_k denote the space needed to compute the rotation map of G_k. Then it is possible to show that $s_k = s_{k-1} + O(1)$, showing that $s_{10\log n} = O(\log n)$.