Undirected log-space connectivity

Srinidhi P V (2020701029)

Vivek Pareek (2021701001)

Problem Statement

- **<u>Decision problem</u>**: Given a graph G and two vertices s(*source*) and t(*target*) in G, we ask if there is a <u>path</u> from s to t in G
- **<u>Search problem</u>**: Given a graph G and two vertices s(source) and t(target) in G, find a path from s to t in G
- Focus on <u>undirected graphs</u> for this discussion
- Formally, we define the language associated with the decision problems as follows: $USTCONN = \{< G, s, t > |$ G is an undirected graph and t is reachable from s , for vertices s and t in $G\}$
- We already have polynomial-time algorithms – breadth-first-search and depth-first-search to answer the connectivity decision problem
  - **DFS**: Time complexity is O(|V| + |E|) and space complexity is O(|V|)
  - **BFS**: Time complexity is O(|V| + |E|) and space complexity is O(|V|)
- Our interest: Is there a log-space DTM (algorithm) that decides $USTCONN$? In other words, can we solve the reachability problem using a deterministic log-space algorithm?

Known Results (space complexity)

STCONN $\in$ NL

- It is a well-known fact that $STCONN$, generalized form of $USTCONN$, is in NL
- The non-deterministic log-space algorithm works as follows:
  - On input $[G, s, t]$, non-deterministically select one of the edges incident on s and traverse it
  - For any vertex we arrive at, non-deterministically select a edge incident on it and traverse it
  - Repeat the above step, till we reach t or no edges to traverse
- At any given point in execution, the algorithm must only keep note of vertex t and current vertex. Since vertex encoding takes $\log(n)$ bits, we get a non-deterministic log-space algorithm

USTCONN ∈ RL

- Suppose that for a given graph $H = (V, E)$ and two vertices $s, t \in V$ that are connected in H, the expected path length for a random walk from s to get to t is μ
- Construct a randomize algorithm as follows:
  - Starting from s, perform random walk for 2μ time steps
    - During any time-step of execution, if we arrive at vertex t, halt and accept input
    - Otherwise continue
  - Halt and reject
- If s and t are not connected, we are never going to find a path. So, the algorithm rejects no instances with 1 probability
- If s and t are connected by a path longer than 2μ, the algorithm rejects the input with a probability less than ½
- At any given point during the execution, the algorithm only must keep note of the target vertex and current vertex, both of which takes $\log(n)$ bits. Hence, this is a log-space algorithm

USTCONN $\in$ L

Expander Graphs

- A key combinatorial object used in the deterministic log-space solution
- A class of graphs that are well-connected and sparse
  - Well-connected implies that any random walk from s leads to t with high probability
  - Spareness implies that the graph does not have many edges
- Formally, we define a (d, k, a) expander graph of an undirected multi-graph of G as follows:
  - <u>Sparsity</u>: For some integer d, the degree of each vertex is exactly d
  - <u>Well-connectedness</u>: Any subset S of V with $|S| \leq k$, we define its complement S'. We say G is well-connected if the number of edges in the boundary of S ((s, s') with $s \in S$ and $s' \in S'$) is at most $a.|S|$
  - Generally, $k = \Omega(|V|)$ and $a = \Theta(d)$
- If graph G is well-connected then it becomes more easy/likely to find a path to the target vertex t

Spectral Expansion of Graphs

- Adjacency matrix – natural representation of graphs
- Spectral expansion gives you insights into graphs from a linear/matrix algebra perspective
- Idea behind spectral expansion:
  - Given a graph G, let M be the adjacency matrix. The normalized adjacency matrix is then $\frac{1}{d}M$
  - $\pi.M$ – probability distribution (in vector form) which gives the probability of ending up in vertex j after taking a random step (π a probability distribution across all vertices v in G)
  - $\pi.M^t$ gives a probability distribution across vertices in G. It tells us the probability of ending up in a vertex j after a t-length random walk
  - If we have π is a uniform distribution across vertices, then $\pi.M$ produces a uniform distribution
  - <u>Claim</u>: After taking polynomial number of random steps, $\pi.M^t$ converges to uniform distribution
  - $\frac{\|\pi_t - u\|}{\|\pi_{t-1} - u\|}$ provides a convergence ratio of a t-length walk to a t-1 length walk. Lower the value, faster the convergence to uniform distribution
- $\lambda(G) = \max_{\pi} \frac{\|\pi.M - u\|}{\|\pi - u\|}$ (a worst-case ratio) and spectral gap: $\gamma(G) = 1 - \lambda(G)$

Spectral Expansion

- A graph G has spectral expansion γ if $\gamma(G) \geq \gamma$.
- If G is a n-vertex, d-regular graph, then G is a $(\frac{n}{2}, \lambda(G))$ expander.

Random walk on expander

Intuition: Let us represent the probability distribution vector for vertices initially from source to be $p^{(0)}$.

Let A be the normalized adjacency matrix for the graph. After one random step we represent the probability distribution vector as $p^{(1)}$

$$p^{(1)} = \mathrm{A}p^{(0)}$$

Theorem : After k steps, we can represent the probability distribution as $p^{(k)} = A^{(k)}p^{(0)}$. Then for a (n, d, α) expander where $\alpha = \max(|\lambda_2|\ ,\ |\lambda_n|)$

$$\left(p^{(k)} - \frac{\vec{1}}{n}\right) \leq \alpha^k$$

This implies that a random walk in an expander reaches the stationary distribution exponentially in α . We can simplify the statement and say that in an expander, after some random steps (Here it would be O(log(n)) steps if $\alpha \leq \frac{1}{2}$ which will be our aim) the probability of being at any particular vertex is uniformly distributed among the vertices.

Effects of squaring

Squaring and it's effects : The squaring of a graph G^2 has this effect on the values of n, d and λ respectively

- number of vertices of G^2 = n
- degree of $G^2 = d^2$
- λ of $G^2 = \lambda^2$

Therefore the effect is that the number of vertices remain the same, the degree of the graph is squared and the value of λ is also squared.

This means that G^2 is a (n, d^2, λ^2) expander. As λ lies between [0,1], the value $\lambda^2 \leq \lambda$ and hence the spectral gap $(1 - \lambda^2)$ increases which means that the graphs expansion increases at the cost of increasing the degree.

Solving USTCONN using powering

Intuition for graph powering : We define the graph powering operation G^k to be the graph constructed when there is an edge between every 2 vertices which are at a distance of length k.

The representation for graph we will use is the rotation map. Note that here we will assume that the graph is a d-regular graph.

Rotation map : The rotation map representation of a graph G is given by $Rot_G(u, i_1) = (\text{v}, j_2)$, this means that if the edges for a vertex of a d-regular graph are enumerated in [1, d] then for $1 < i_1 <$ d and $1 < j_2 <$ d if we take edge i_1 from vertex u we land on v and from perspective of vertex v, the j_2^{th} edge lands us on vertex u.

Solving USTCON using powering

Suppose we have a graph G. The rotation map of graph G^2 given by Rot_{G^2}(u, (i_1, i_2)) can be computed using $Rot_G(u, i_1) = (w, j_2)$ and $Rot_G(w, i_2) = (v, j_1)$. So we can compute the rotation map of G^2 using the rotation map of G using some extra space for i_1, j_2 and i_2. Since we just need extra space for storing the edge index, the extra space we are using is log(Deg(G)).

$$\text{Space}(G^2) = \text{Space(G)} + \log(\text{Deg(G)})$$

We can perform this in iterative manner for n times and after each iteration we can check for USTCON condition. We have provided a solution for USTCON this way, but there is a problem here. The space required for G^n is :

$$\text{space}(G^n) = \text{space}(G^{2^{log(n)}})$$
$$\text{space}(G^{2^{log(n)}}) = \text{space}(G^{2^{log(n)-1}}) + \log(\text{Deg}(G^{2^{log(n)-1}}))$$
$$\text{space}(G^n) = \text{space(G)} + \text{space}(log(Deg(G))) + \text{space}(log(Deg(G^2))) + \ldots + \text{space}(log(Deg(G^{2^{log(n)-1}})))$$

What's wrong with this approach?

Though we have provided a solution for USTCONN, squaring operation also squares the degree and hence the space required is not O(log(n)), it cannot be upper bounded by logarithmic space. If we were able to keep the degree constant, then we could have a logarithmic space algorithm. This is where zig-zag product comes in, it keeps the degree constant but mildly decreases the gain we had with spectral gap after powering. So, Reingold's algorithm performs a combination of graph powering and zig-zag product to convert an arbitrary graph into an expander hence solving USTCONN in log-space.

Properties of zig-zag products

Rotation map representation of G ⓩ H given by $Rot_{G ⓩ H}$ is given by :

<u>Definition of zig zag product</u> : Let G be a (N, D, λ_1) expander and and H be a (D, d, λ_2) expander. Then G ⓩ H has following properties:

- V(G ⓩ H) = V(G)*V(H)
- Deg(G ⓩ H) = d^2
- λ = f(λ_1, λ_2)
- $Rot_{G ⓩ H}$((u, i), (a_1, a_2)) = ((v, j), (b_1, b_2)) if the following is satisfied : There exist (i', j') such that

  1. $Rot_H(i, a_1)$ = (i', b_2)
  2. $Rot_G(u, i')$ = (v, j')
  3. $Rot_H(j', a_2)$ = (j, b_1)

If we have d << D then the effect of zig-zag product is that the number of vertices increases, the degree decreases if d << D and if H is a good expander then spectral gap decreases but the decrease is mild.

The zig-zag step

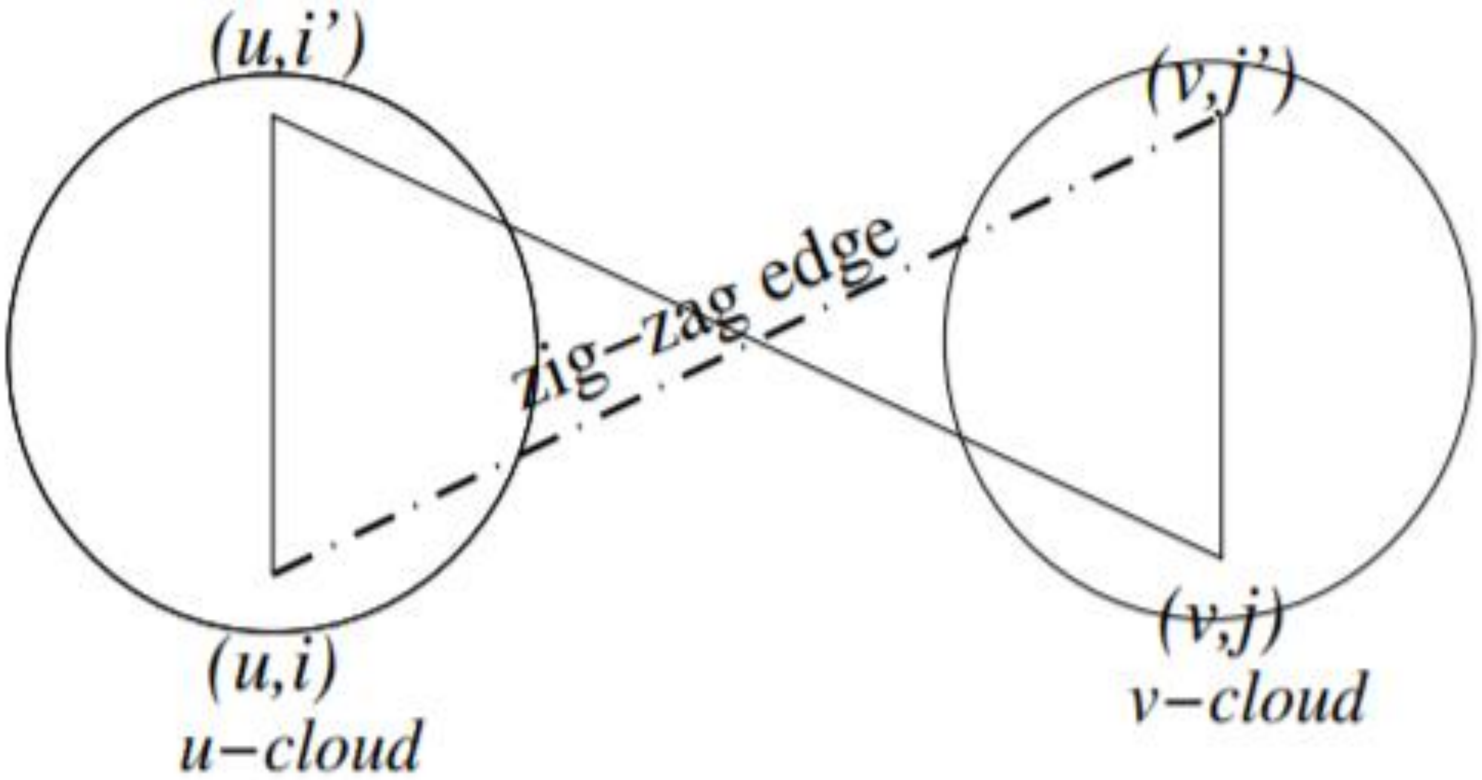

Figure 1: Zig-Zag Product

Theorem 1 : The value of λ i.e $f(\lambda_1, \lambda_2) \leq \lambda_1 + \lambda_2 + \lambda_2^2$

If H is a good expander such that $\lambda_2 \leq \frac{1}{2}$ then the value of $f(\lambda_1, \lambda_2)$ does increase but not by much w.r.t G. This implies that the spectral gap is decreased but the decrease is mild.

Lemma 1 : If G is a connected, D regular non bipartite graph on n vertices then

$$1 - \lambda \geq \frac{1}{DN^2}$$

This implies that the spectral gap of a D-regular, non-bipartite, connected graph on n-vertices, is lower bounded by inverse polynomial of number of vertices in the graph.

Lemma 2 : If we have $\lambda(H) \leq \frac{1}{2}$ then

$$1 - \lambda\ (G \ z\ H) \leq \frac{1}{3}\ (1-\lambda(G))$$

We saw previously that squaring increases spectral gap $(1 - \lambda)$ to $(1 - \lambda^2)$. By lemma 2, the spectral gap decreases by a factor of 3, which is a mild decrease. Reingold's algorithm applies powering and zig-zag product (say k iterations) to increase spectral gap such that we reach $\lambda_k \leq \frac{1}{2}$.

Reingold's algorithm

<ins>Theorem 2</ins> : Random n vertex d regular graph is an expander with high probability

<ins>Theorem 3</ins> : There is a d ∈ N such that for every n there is an (n, d, 1/2) expander that can be constructed deterministically.

Let H be a $(d^{16}, d, \frac{1}{2})$ expander. Such a graph H can be constructed deterministically (it's existence is guaranteed by Theorem 3) or using Theorem 2, we can pick random graphs until we get a graph satisfying the properties of H (this is an exhaustive search).

We also make the following assumptions about G :

- G is a non-bipartite regular graph. (We can convert our graph to follow this.)
- G is a (N, d^{16}, λ) expander. We will convert our graph G into a 3-regular graph by replacing each vertex in G by a cycle of length d. Then we will add self loops to each vertex to make the graph d^16 regular.
- G is connected. This assumption may not make sense at first because if G is connected, then there is no point of this algorithm as s-t are connected. This assumption actually means that the algorithm works on connected components of G. This will work as each component of G is also non-bipartite and d^{16} regular.

Reingold's algorithm

Reingold's Algorithm :

Input: G - d^{16} regular graph and 2 vertices s,t $\in$ V(G)

1. Set l to be the smallest integer such that $\left(1 - \frac{1}{d^{16}n^2}\right)^{2^l} \leq \frac{1}{2}$. Here l is O(log(n))
2. Set $G_0 \leftarrow$ G
3. for i = 1,...,l do $G_i \leftarrow (G_{i-1} ⓩ H)^8$
4. Check if s and t are connected in G_l by enumerating over all O(log(n)) paths and check if s and t are connected in one of them.

We cannot directly work on the graph G_l as the construction of G_l will take more than O(log(n)) space. We have to emulate steps on G_l on graph G using only logarithmic space.

Emulating G^l

```
def emulate_expander(G, H, t, u, i) :
"""
params:
    G: Input graph for USTCONN
    H: Graph to do zig-zag product with
    t: Round of expander construction
    u: Vertex to move out of
    i: Edge index
"""
    if t = 0:
        # No expander construction. Check G
        Traverse adjacency list of u in G
        v:= i-th neighbour of u in G
        j:= edge index such that (v,j) = u
        return (v, None, j, None)
    else:
        Interpret u as (a,x) in V(G(t-1)) x V(H)
        Interpret i as (a, b) in {1, .., d*d}
        x':= a-th neighbour of x in H
        a':= Index of x on the list of x' in H

        Interpret x as ({s1, ..,sk}) in {1,..,d*d}^8
        for i = 1, ..,8 do
            si' = emulate_expander(G, H, t-1, u, si)

        Interpret (s'8, ..,s'1) as y in {1, .., d^16}
        y':= b-th neighbour of y in H
        b':= Index of y in list of y' in H

    return (y, b', (s'8, ..,s'1), a')
```

Space complexity

We can see from the rotation map representation of zig-zag product that the space required for it is some constant. In our analysis of the space required for squaring, we can modify the equation for k = log(n) steps:

$$\text{space}(G^k) = \text{space}(G) + \sum_{i=1}^{log(k)-1} log(Deg(G^{2^i}))$$

Since we know that the extra space required for zig-zag product is constant, the overall space required for combination of powering and zig-zag is some constant i.e O(log(Deg(G))). Since Deg(G) is maintained constant at each iteration to be d^2

Since we maintain constant degree in our graph, we can approximate the above equation for combination of powering and zig-zag as

$$\text{space}(G_l) \approx \text{space}(G) + \sum_{i=1}^{log(n)-1} log(d^2) \approx O(\log(n).\log(d)) \approx O(\log(n))$$

Hence we have shown how we can emulate graph G_l on G using only O(log(n)) space.