

# 1 Introduction

When designing algorithms, there are certain properties of the algorithm that we aim to optimize for. These include:

- Time
- Space/Memory
- Parallelism
- Power
- Loss/Accuracy
- Determinism / Randomness
- Communication

The main goal of this class will be focusing on the Space / Memory, Determinism / Randomness, and Communication complexity of algorithms.

## 1.1 Memory Efficiency

We define the memory usage of an algorithm based off how much *extra memory* it requires given an input of size  $x$ . We say that the algorithm has read access to the input  $x$  and Read/Write access to any extra space it may need to run.

# 2 s-t Connectivity

The s-t connectivity problem is defined as follows:

- Input:  $G = (V, E), s, t$  where  $s$  is the source node and  $t$  is the destination node.
- Output:  $\begin{cases} \text{YES if } s, t \text{ are connected.} \\ \text{NO otherwise.} \end{cases}$

The trivial solution to this problem is to perform a BFS or DFS from  $s$  to see if  $t$  is reachable. This requires a graph representation which is typically assumed to be one of the following:

- **Adjacency List:** Each vertex has a corresponding list of vertices that it shares an edge with.

- **Adjacency Matrix:** An  $(|V| = n)^2$  sized matrix denoted as  $A$  where  $A[i, j] = 1$  if there is a directed edge from vertex  $i$  to vertex  $j$ .

## 2.1 Basic algorithm

The basic algorithm using BFS is as follows

---

### Algorithm 1 Graph Reachability

---

```

1: Flag[u] ← 0  ∀u
2: Flag[s] ← 0
3: Q ← [s]
4: while Q ≠ ∅ do
5:   remove v from Q
6:   for each neighbor u of v do
7:     if Flag[u] = 0 then
8:       add u to Q
9:     end if
10:  end for
11:  Flag[v] ← 1
12: end while
13: if Flag[t] = 1 then
14:   output YES
15: else
16:   output NO
17: end if

```

---

The running time of this algorithm is  $O(|V| + |E|)$ . The Memory usage is dependent on the Flag array which requires  $|V|$  extra bits. (Ask why this is the bound since the algorithm needs  $\log(|V|)$  bits to represent nodes).

## 2.2 Memory Efficient s-t Connectivity

Questions were raised wondering if there was a more memory efficient version of BFS that solved this problem and if so, what is the least memory needed to solve s-t connectivity.

**Theorem 2.1** (1980s). *There is a randomized algorithm with  $5 \log |v|$  bits of additional memory to solve s – t connectivity problem that runs in  $O(n^3)$  runtime.*

### Random Walk Algorithm

- Counter  $\leftarrow 0$
- $v \leftarrow s$
- **while** Counter  $\leq T$ :
  - **if**  $v = t$  **then return YES**
  - **else**
    - \*  $v \leftarrow$  a random neighbor of  $v$
    - \* Counter  $\leftarrow$  Counter + 1
- **return NO**

The bits needed are for current vertex  $v(\log n)$ , Counter  $(\log T)$ , Bits needed for random sampling  $(\log n)$ .

The above algorithm uses the following result:

**Theorem 2.2.** *For all graphs, if  $s, t$  are connected:*

$$\mathbb{E}[\text{Time random walk from } s \text{ hits } t] = O(n^3)$$

### 2.3 Notes and History

The above algorithm and proof led to the question of whether or not we can get a **logspace** algorithm for directed s.t. connectivity **without randomness**?

The reason that this question holds significance in complexity theory is its implications on computational ability. s-t Connectivity is a problem that is *NL* (Nondeterministic Logarithmic Space) Complete. This has the following implication:

**Theorem 2.3.** *If directed  $s - t$  connectivity can be solved with  $O(\log |V|)$  bits of memory (and no randomness)  $\implies$  Any randomized algorithm can be made deterministic at the expense of a constant-factor increase in memory (and polynomial slow-up in time).*

In 2005, Omer Reingold proved that  $s.t$  connectivity on undirected graphs can be solved with  $O(\log |v|)$  extra memory. This is the first big hit for this class.

### 3 Spectral Graph Theory

To construct an algorithm that solves *s.t* connectivity, we use an adjacency matrix. We also need the following definitions.

**Definition 3.1.** We say a graph is  $D$ -regular if all the vertices have the same degree  $D$ .

**Definition 3.2.** A Normalized Adjacency Graph, denoted as  $M_a = \frac{A_G}{D}$ . This means the sums of the matrix add to 1.

( I don't really know how these relate, ask professor next lecture )

**Theorem 3.1** (SGT Theorem 1). *If  $G$  is regular. Then:*

1.  $1$  is an eigenvalue of  $M_G$
2.  $G$  is connected  $\iff$  eigenvalue of  $1$  is unique

*Proof.* (1) To demonstrate the first part of the theorem, we simply need to show a vector that, when multiplied by  $M_G$ , remains the same. Consider the  $\vec{1}$  (All 1's vector). Since the matrix is normalized, the sum of all entries of a particular row is

$$\sum_{i=1}^d \frac{1}{d} = 1$$

Thus,  $\vec{1}$  multiplied by this matrix must result in a 1 for every row which gives us back the 1 vector  $\square$

We prove the second part of the theorem in two parts.

1. All eigenvalues are less than or equal to 1
2. If the graph is disconnected, the eigenvalue 1 appears at least twice in the eigenvalues of the matrix

*Proof.* (2a) To demonstrate that all eigenvalues are less than or equal to 1, consider  $v_{i*}$ , the largest entry in  $v$  in absolute value. We will show that this

value cannot increase as a result of the matrix multiplication.

$$\begin{aligned}
\lambda v_{i*} &= \sum_{j=1}^n M[i, j] \cdot v_j \\
|\lambda| \cdot |v_{i*}| &= \left| \sum_{j=1}^n M[i, j] \cdot v_j \right| \\
&\leq \sum_{j=1}^n M[i, j] \cdot |v_j| \\
&\leq \sum_{j=1}^n M[i, j] \cdot |v_{i*}| \\
&= |v_{i*}| \cdot \sum_{j=1}^n M[i, j] \\
&= |v_{i*}| \cdot 1
\end{aligned}$$

Here we see that based on the normalized rows, the max entry in the vector cannot increase in magnitude. This implies that the absolute value of  $\lambda$  must be less than or equal to 1.  $\square$

To prove that a disconnected graph has at least 2 eigenvectors with eigenvalue 1, consider the structure of a matrix that represents a disconnected graph. This would be a **block structure** meaning that it would contain two squares in the matrix that correspond to the connected components of the graph. In this case, the vectors of only 1's for those particular entries would result in the same vector when multiplied by that matrix. (Look at notes for this better intuition).