1 From Last Lecture

We created an efficient $(2k-1)$-distance oracle that uses $\tilde{O}(k \cdot n^{1+1/k})$ space and answers a query in $O(k)$ time. Recall that we constructed sets A_i for $0 \leq i \leq k-1$ by setting $A_0 = V$ and sampling A_i from A_{i-1}. Then, for any vertex $v \in V$, we defined $p_i(v)$ to be the closest node in A_i to v. Finally, we defined $B_i(v) = \{x \in A_i | d(v,x) < d(v, p_{i+1}(v))\}$ for $0 \leq i \leq k-2$ and $B(v) = A_{k-1} \cup \left(\bigcup_{i=0}^{k-2} B_i(v) \right)$. We proved that $|B(v)| = \tilde{O}(k \cdot n^{1/k})$ and, for all i, $p_i(v) \in B(v)$. For the query algorithm, we store for all $v \in V$ and $x \in B(v)$ $d(v,x)$.

Algorithm 1: Query(u,v)

```
w ← p_0(v) = v;
for i = 1 → k do
    //w = p_{i-1}(v) ∈ B(v);
    if w ∈ B(u) then
        return d(u,w) + d(w,v);
    else
        w ← p_i(u);
        swap u and v;
```

2 Proof of $(2k-1)$-approximation

Note that in iteration k, we have $p_{k-1}(v) \in A_{k-1} \subseteq B(u) \cap B(v)$ by construction. Thus, Query(u,v) will always return some estimate of $d(u,v)$. We know that, for any w, $d(u,w) + d(w,v) \geq d(u,v)$ by the triangle inequality. We will now show that the returned estimate $D(u,v) = d(u,w) + d(w,v) \leq (2k-1)d(u,v)$.

Lemma 2.1. *If in iteration i, for $w = p_{i-1}(v)$, we have $d(w,v) \leq (i-1) \cdot d(u,v)$, then:*

- $d(u,w) + d(w,v) \leq (2i-1) \cdot d(u,v)$
- *if $w \notin B(u)$, then $d(u, p_i(u)) \leq i \cdot d(u,v)$*

Note that the initial condition is trivially true when $i = 1$ as $w = v$ and $d(w,v) = 0$. Then at each iteration we either return (when $w \in B(u)$) or guarantee the condition for the next iteration. Since the worst possible return is in the k^{th} iteration, we get the our desired approximation factor by induction.

Proof of Lemma 2.1. Suppose $d(w,v) \leq (i-1) \cdot d(u,v)$.

$$\begin{aligned} d(u,w) + d(w,v) &\leq d(u,v) + d(v,w) + d(v,w) \\ &\leq d(u,v) + 2(i-1) \cdot d(u,v) \\ &= (2i-1) \cdot d(u,v) \end{aligned}$$

Furthermore, assume that $w \notin B(u)$. Note that $w = p_{i-1}(v) \in A_{i-1}$ and by definition $B_{i-1}(u) = \{x \in A_{i-1} | d(u,x) < d(u, p_i(u))\} \subseteq B(u)$. Hence since $w \notin B_{i-1}(u)$,

$$\begin{aligned} d(u, p_i(u)) &\leq d(u, w) \\ &\leq d(u, v) + d(v, w) \\ &\leq d(u, v) + (i-1) \cdot d(u, v) \\ &\leq i \cdot d(u, v) \end{aligned}$$

□

3 A $(4k-3)$-approximation

If you do not swap u and v at the end of each iteration, and set $w \leftarrow p_i(v)$ at each iteration, you get a $4k-3$ approximation. This is useful in contexts where you only have local information (as we will see in Compact Routing).

Lemma 3.1. *Let i be the smallest i such that $p_i(v) \in B(u)$. Then $D(u, v) := d(u, p_i(v)) + d(p_i(v), v) \leq (4k-3) \cdot d(u, v)$.*

Proof. We start by showing that $d(v, p_i(v)) \leq 2i \cdot d(u, v)$. This is a proof by induction; we will show that for all $j \leq i$, $d(v, p_j(v)) \leq 2j \cdot d(u, v)$. The inequality is trivially true for $j = 0$ as $p_0(v) = v$ and $d(v, p_j(v)) = 0$. Suppose it is true for $j < i$. Then for $j + 1$:

$$\begin{aligned} d(v, p_{j+1}(v)) &\leq d(v, p_{j+1}(u)) \\ &\leq d(v, u) + d(u, p_{j+1}(u)) \\ &\leq d(v, u) + d(u, p_j(v)) \\ &\leq d(v, u) + d(u, v) + d(v, p_j(v)) \\ &\leq 2(j+1) \cdot d(v, u). \end{aligned}$$

The first line follows from the definition of $p_{j+1}(v)$ and since $p_{j+1}(u) \in A_{j+1}$. The second and fourth lines follow from the triangle inequality. The third line follows from the definition i: we know that for all $j < i$, $p_j(v) \notin B(u)$, which implies $d(u, p_j(v)) \geq d(u, p_{j+1}(u))$. Finally, the last line follows from the inductive hypothesis.

Now we use following two equations:

$$d(v, p_i(v)) \leq 2i \cdot d(u, v), \tag{1}$$

$$d(u, p_i(v)) \leq (2i+1) \cdot d(u, v). \tag{2}$$

Note that we just showed (1) by induction and (2) directly follows from (1) and the triangle inequality. We thus have:

$$\begin{aligned} D(u, v) &= d(u, p_i(v)) + d(p_i(v), v) \\ &\leq (4i+1) \cdot d(u, v) \\ &\leq (4(k-1)+1) \cdot d(u, v) \\ &\leq (4k-3) \cdot d(u, v). \end{aligned}$$

□

4 Compact Routing

A common application of distance oracles is compact routing, which we describe now. We have a graph $G = (V, E)$ and every node $v \in V$ has a routing table R_v. Each node receives packets that arrive with a header of information, including $L(u)$ - the address of the destination node u. The node then looks at its routing table R_v and decides which neighbor to send the packet to.

We want to design a method that stores small R_v and $L(u)$ for each node, while achieving short (i.e. close to optimal) paths for each packet.

Let's consider a first attempt. We will show the full construction in the next lecture. We will compute the distance oracle as before. For each vertex $v \in V$, R_v will store $p_i(v)$ for all i and, for all $x \in B(v)$, the next node in the shortest path from v to x. And for each vertex u, the label will be $L(u) = \{u, p_0(u), ..., p_{k-1}(u)\}$. We thus have $|R_v| \sim |B(v)| \sim \tilde{O}(kn^{1/k})$ and $|L(u)| \sim k \log n$, both of which are pretty good!

We now discuss how to decide which node to route an incoming packet to. Suppose node u gets packet $L(v)$. We run the algorithm without swapping as described in the previous section (i.e. we try each $p_i(v)$ and check if they are in $B(u)$). More formally:

Algorithm 2: NextNode$_u(v)$

```
for i = 0 → k − 1 do
    if p_i(v) ∈ B(u) then
        Send packet to next node on shortest path from u to p_i(v);
```

NOTE this algorithm only specifies how to send a packet to a node that is $p_i(v)$. It breaks down once we are there - more details in the next lecture!

This gives a $4k - 3$ approximation on the shortest path! We can achieve a $2k - 1$ approximation with a concept called "hand-shaking" to essentially simulate the first algorithm.

Handshaking. Suppose that a node u wants to send a packet to a node v. The sender u knows the address $L(v)$ of v and also its own bunch $B(u)$ but would like to compute the lowest j such that $p_j(u) \in B(v)$ and the lowest j' such that $p_{j'}(v) \in B(u)$. If it has both of these, it can route along a $(2k - 1)$-approximate shortest path. Note that u can compute the lowest j' such that $p_{j'}(v) \in B(u)$ since it has both $L(v)$ and $B(u)$, but it does not know how to compute the lowest j such that $p_j(u) \in B(v)$ since it does not know $B(v)$. The handshaking process just asks v for this $p_j(u)$ as follows.

The sender u first sends a small packet to v containing $L(u)$ along a $4k - 3$-approximate path. This packet asks v to compute the lowest j such that $p_j(u) \in B(v)$. The destination v sends this $p_j(u)$ back to u (or even just j). Then u compares j and j' (where $p_{j'}(v)$ is its own computed value), and sends to $p_j(u)$ if $j < j'$ and to $p_{j'}(v)$ otherwise.

Handshaking is especially useful if u is going to send many packets to v- only one small packet is sent along a $4k - 3$ approximate path, and all others are sent along $2k - 1$-approximate paths. The initial long path becomes negligible.

It is an open problem whether one can do $2k - 1$-approximate compact routing without handshaking. Chechik recently showed that $4k - 3$ is not optimal, and that one can do $3.68k$-approximate compact routing without handshaking.