6.890 Lecture 14 Distance oracles
Scribe: Charles Celerier, Nicole Wein **Date:** October 28, 2021

Given a graph G, a distance oracle is a data structure that stores a summary of G so that distance queries can be answered efficiently.

1 t-approximate distance oracles

A t-approximate distance oracle is defined by two algorithms:

- a preprocessing algorithm that takes as its input a graph $G = (V, E)$ (possibly with nonnegative edge weights) and returns a summary $S(G)$ of G stored in memory.
- a query algorithm that takes as its input two vertices $u, v \in V$ and returns an estimate $D(u, v)$ such that $d(u, v) \leq D(u, v) \leq t \cdot d(u, v)$ where $d(u, v)$ is the distance from u to v in G. The query algorithm does not have access to G, but only to the summary $S(G)$ of G stored in memory.

The quality of an approximate distance oracles is determined by four things:

- the approximation factor t,
- the storage space $|S(G)|$,
- the preprocessing time, and
- the query time.

We would like to minimize all of these quantitites.

A first idea for a t-approximate distance oracle is to use a t-spanner graph of G. The space usage would be small, and the quality of the distance estimates would be guaranteed, and the preprocessing time would be small. However, in order to compute each query, one may still need to run Dijkstra's algorithm (or BFS for unweighted graphs) on the spanner, and this would take time at least linear in the number of edges in the spanner. Last lecture we saw a $(2k-1)$-spanner on $n^{1+1/k}$ edges, so this implies a distance oracle with query time $\tilde{O}(n^{1+1/k})$. However, we would like the query time to be *constant.*

2 A lower bound on space usage

Although the above spanner does not achieve good query time, as we will show, it does achieve optimal space usage for any distance oracle with the same approximation factor, if we assume the widely believed Erdős girth conjecture.

Conjecture 1 (Erdős girth conjecture)**.** *Let $m_k(n)$ be the maximum number of edges in an n-node undirected unweighted graph of girth $\geq 2k + 2$. Then $m_k(n) = \Omega(n^{1+1/k})$.*

As a sidenote, recall that last lecture we showed that if an n-node graph has at most $2n^{1+1/k}$ edges then it has girth at most $2k$. The Erdős girth conjecture shows that this statement is asymptotically optimal in the sense that the conjecture says that there exist graphs on $Cn^{1+1/k}$ edges for some constant $C < 2$ with girth at least $2k + 2$.

Now we will prove that this above spanner achieves optimal space usage under this conjecture:

Theorem 2.1. *If $\overline{D}$ is a t-distance oracle for $t \leq 2k$, then for all n-node graphs G, the storage space $|S(G)|$ of $\overline{D}$ on G must be at least $m_k(n)$ bits.*

The consequence of this theorem is that the Erdős girth conjecture implies that the required amount of storage is at least $\Omega(n^{1+1/k})$. The amount of storage our spanner uses is $O(n^{1+1/k}\log n)$ where the log factor is simply for writing down the name of each edge. Intuitively, this theorem suggests that graphs with low girth cannot be "compressed" very well.

Proof of Theorem 2.1. Let G be a girth $\geq 2k+2$ graph on $m_k(n)$ edges and n nodes. Note that this implies G has $2^{m_k(n)}$ subgraphs. The main idea of this proof is to show that for any pair H_1, H_2 of subgraphs of G, any distance oracle must store different information for each one, that is, $S(H_1) \neq S(H_2)$. This provides a lower bound on the storage space.

Let H_1 and H_2 be subgraphs of G so that $H_1 \neq H_2$. Then, without loss of generality, let there be an edge (u, v) in H_1 that is not in H_2. Let D be a t-approximate distance oracle for $t < 2k+1$. We will consider what D stores in memory for *all* $2^{m_k(n)}$ subgraphs of G.

Let $D_{H_1}(u, v)$ and $D_{H_2}(u, v)$ be the answer that D gives for $d_{H_1}(u, v)$ and $d_{H_2}(u, v)$, respectively. Since $d_{H_1}(u, v) = 1$, we know that

$$1 \leq D_{H_1}(u, v) \leq t. \tag{1}$$

Furthermore, since G has no cycles of length at most $2k+1$, removing the edge (u, v) from G causes the distance between u and v to become at least $2k+1$. Thus, $d_{H_2}(u, v) \geq 2k+1$. Therefore, we have

$$D_{H_2}(u, v) \geq d_{H_2}(u, v) \geq 2k+1 > t. \tag{2}$$

Combining euqations 1 and 2, we have that $D_{H_1}(u, v) < D_{H_2}(u, v)$. Since the distance oracle uses the same query algorithm for both H_1 and H_2, and the only input to the query algorithm is the summary S of the input graph, this means that $S(H_1) \neq S(H_2)$.

Thus, for all $2^{m_k(n)}$ subgraphs of G, the information that D stores must be distinct. Then since $2^{m_k(n)}$ different bit strings cannot be represented by less than $m_k(n)$ bits, the storage space that D uses must be at least $m_k(n)$ bits. □

3 A $(2k-1)$-approximate distance oracle with constant query time

For the rest of the lecture we will show that there exist distance oracles with constant query time that still have optimal space usage. In particular, we will prove the following theorem.

Theorem 3.1. *For all integers $k \geq 2$, for any n-node undirected weighted graph, there exists a $(2k-1)$-approximate distance oracle using $\tilde{O}(n^{1+1/k})$ space, $\tilde{O}(mn^{1/k})$ time for preprocessing, and can answer queries in $O(1)$ time.*

It turns out that good distance oracles do not exist for directed graphs. It is a good exercise to think about why.

3.1 Warm-up: A 3-approximate distance oracle

As a warm-up to proving theorem 3.1, we will consider the $k = 2$ case, i.e. a 3-approximate distance oracle with $\tilde{O}(n^{3/2})$ space usage.

Construction

- For each $v \in V$, let $N_{\sqrt{n}}(v)$ be the set of the closest $\sqrt{n}$ nodes to v.
- Let A be a random subset of V of size $O(\sqrt{n}\log n)$ so that by the hitting set results from previous lectures, we have that A hits $N_{\sqrt{n}}(v)$ for all v with high probability. (We can also create A deterministically, given all the $N_{\sqrt{n}}(v)$.)
- For each vertex v, let $p(v)$ be the closest node to v that is in A.

- Let $A(v)$ to be the set of nodes that are closer to v than A is to v. That is, $A(v) = \{x \in V \mid d(v,x) < d(v,p(v))\}$.
- Let the *ball* (or *bunch*) $B(v)$ of v be $A \cup A(v)$ for all v.

For all $v \in V$ and for all $x \in B(v)$ we store $d(v,x)$ in a hash table. We also store for each v, $p(v)$.

Query algorithm If $v \in B(u)$, then the distance oracle returns $d(u,v)$ from memory by accessing the hash table of v. Otherwise, $v \notin B(u)$, so the distance oracle accesses $p(u)$ and returns $d(u,p(u)) + d(p(u),v)$. This is possible in constant time since by definition $p(u) \in A \subset B(x)$ for all $x \in V$.

Storage space We will show that we only store $n^{3/2}$ distances. To do this, we will show that each ball is of size at most $\tilde{O}(n^{1/2})$ with high probability.

Recall that $B(v) = A \cup A(v)$. Since $|A| = O(\sqrt{n}\log n)$, it suffices to show $|A(v)| \leq \sqrt{n}$. Because A hits the closest $\sqrt{n}$ nodes to v with high probability, we necessarily have that some node $a \in A$ is also in $N_{\sqrt{n}}(v)$. Thus, all nodes closer to v than a are also in $N_{\sqrt{n}}(v)$. By the definition of $A(v)$, this implies $A(v) \subset N_{\sqrt{n}}(v)$. Therefore, $|A(v)| \leq |N_{\sqrt{n}}(v)| = \sqrt{n}$.

Approximation factor We are only left with proving that the query responses satisfy

$$d(u,v) \leq D(u,v) \leq 3d(u,v). \tag{3}$$

First we note that if $v \in B(u)$, our query algorithm returns the exact distance. Thus, suppose $v \notin B(u)$. In this case our query algorithm returns $d(u,p(u)) + d(p(u),v)$. By the triangle inequality, $d(u,v) \leq d(u,p(u)) + d(p(u),v)$, so the first inequality in (3) holds.

Now we prove the second inequality. Since $v \notin B(u)$ we know that $v \notin A(u)$. Thus, by definition $d(u,v) \geq d(u,p(u))$. Applying this inequality along with the triangle inequality, we have:

$$\begin{aligned} D(u,v) &= d(u,p(u)) + d(p(u),v) \\ &\leq d(u,p(u)) + (d(p(u),u) + d(u,v)) \\ &= 2d(u,p(u)) + d(u,v) \\ &\leq 3d(u,v). \end{aligned}$$

4 Proof of Theorem 3.1

We will ignore preprocessing time for this lecture; you will prove it on the homework.

Construction Instead of taking a single sample A of nodes, we take many nested samples.

Let $A_0 = V$ and $A_k = \emptyset$. For $1 \leq i \leq k-1$, choose a random $A_i \subseteq A_{i-1}$ of size $\frac{|A_{i-1}|}{n^{1/k}} \log n \leq \tilde{O}(n^{1-i/k})$. Let $p_i(v)$ be the closest node to v in A_i, with the caveat that if $d(v,p_i(v)) = d(v,p_{i+1}(v))$ then we let $p_i(v) = p_{i+1}(v)$. Now we define the following sets for all v

$$A_i(v) = \{x \in A_i \mid d(v,x) < d(v,p_{i+1}(v))\}$$

and

$$B(v) = A_{k-1} \cup \left(\bigcup_{i=0}^{k-2} A_i(v) \right).$$

Then $\forall\, v \in V$, $\forall\, x \in B(v)$, store $d(v,x)$ in a hash table for v. Also store for each $v \in V$ and each $i \leq k-1$, $p_i(v)$.

Query algorithm

Algorithm 1: Query(u, v)

```
$w \leftarrow= v$
for $i = 0$ to $k - 1$ do
    if $w \in B(u)$ then
        return $d(u, w) + d(w, v)$;
    else // $w \notin B(u)$
        $w \leftarrow p_{i+1}(u)$;
        swap $u$ and $v$;
```

This algorithm looks strange at first glance. Below we will discuss the ideas behind what is happening here.

Storage space We will show that we only store $n^{1+1/k}$ distances. To do this, we will show that each ball is of size at most $\tilde{O}(n^{1/k})$ with high probability.

By definition, $|A_0| = n$, $|A_1| = \Theta(n^{1-1/k} \log n)$, and more generally, $|A_i| \leq O(n^{1-i/k} \log^i n)$, so $|A_{k-1}| \leq \tilde{O}(n^{1-(k-1)/k}) = \tilde{O}(n^{1/k})$.

It remains to show that for all i, $|A_i(v)| \leq \tilde{O}(n^{1/k})$. Let $T_{i,v}$ be the closest $n^{1/k}$ nodes of A_i to v. By construction, $A_{i+1} \subseteq A_i$ is random and has size $O(\frac{|A_i|}{n^{1/k}} \log n)$. This implies, again by the hitting set results discussed before, that with high probability for all v, A_{i+1} hits $T_{i,v}$ (i.e. there is a node in A_{i+1} that is also in $T_{i,v}$). By the definition of $A_i(v)$, similar to our proof for the $k = 2$ case, this implies $A_i(v) \subseteq T_{i,v}$. We can then conclude that $|A_i(v)| \leq |T_{i,v}| = n^{1/k}$.

Query time First notice that in iteration i of the for loop $w = p_i(v)$, although u and v swap at each iteration so w alternates between p_i of the original v and p_i of the original u. Thus, we return $d(u, p_i(v)) + d(p_i(v), v)$ (where possibly u and v have swapped from their original identities).

To show that the query algorithm runs in constant time, we need to show that the distance oracle has stored the distances $d(u, p_i(v))$ and $d(p_i(v), v)$. That is, we will show that

Claim 1. *For all* i, u, $p_i(u) \in B(u)$.

Proof. By induction. Observe that $p_{k-1}(u) \in A_{k-1} \subseteq B(u)$. Assume $p_{i+1}(u) \in B(u)$. If $p_i(u) = p_{i+1}(u)$, we are done. Otherwise, $d(p_i(u), u) < d(u, p_{i+1}(u))$ so $p_i(u) \in A_i(u) \subseteq B(u)$. □

Approximation factor Note that by the triangle inequality, we have

$$d(u, v) \leq d(u, w) + d(w, v) = D(u, v)$$

for all i. Hence any distance estimate returned will always be at least the real distance.

We will prove correctness by induction via Lemma 4.1 below. The following few paragraphs before the lemma statement describe the analysis for the first few levels. They are not strictly necessary for the proof, but may provide some idea of what is going on.

If $v \in B(u)$ the algorithm returns the exact distance. So assume otherwise. Then, $v \notin A_0(u)$, so $d(u, v) \geq d(u, p_1(u))$.

Now, the algorithm asks if $p_1(u) \in B(v)$. If so, the algorithm returns $D(u, v) = d(v, p_1(u)) + d(p_1(u), u)$. Applying the triangle inequality to the triangle composed of u, v, and $p_1(u)$, we have that since $d(u, v) \geq d(u, p_1(u))$, we have $d(v, p_1(u)) \leq 2d(u, v)$. Thus, $D(u, v) \leq 3d(u, v)$.

On the other hand, if $p_1(u) \notin B(v)$, then $p_1(u) \notin A_1(v)$ so $d(v, p_1(u)) \geq d(v, p_2(v))$. Since $d(v, p_1(u)) \leq 2d(u, v)$, we have that $d(v, p_2(v)) \leq 2d(u, v)$. Now, the algorithm asks if $p_2(u) \in B(v)$. If so, the algorithm returns $D(u, v) = d(u, p_2(v)) + d(p_2(v), v) \leq d(u, v) + 2d(v, p_2(v))$. Now, we have that since $d(v, p_2(v)) \leq 2d(u, v)$, we have that $D(u, v) \leq 5d(u, v)$.

Now we give our lemma that gives us an inductive proof of the approximation quality.

Lemma 4.1. *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 $d(u,p_i(u)) \leq i \cdot d(u,v)$*

Proof of Lemma 4.1. Note that the initial condition is trivially true when $i = 1$ as $w = v$ and $d(w,v) = 0$.

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 $w \notin B_{i-1}(u)$,

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

□

Lastly, we note that the algorithm will return an estimate $D(u,v) = d(u,w) + d(w,v)$ in some iteration. To see this, notice that $p_{k-1}(u), p_{k-1}(v) \in A_{k-1} \subset B(u) \cap B(v)$. Hence if no value was returned by iteration k, in iteration k we'll have $w = p_{k-1}(v)$ and $w \in B(u)$.

Since the worst possible return is in the k^{th} iteration, we get the our desired approximation factor by induction via Lemma 4.1.