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 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 of G stored in memory.

The quality of a t-approximate distance oracles is defined by its query time ($q(n)$), preprocessing time ($p(m, n)$), and storage space ($s(n)$). Intuitively, we want 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. 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. As we'll see later, it turns out that we can do better than this.

Before we talk about upper bounds, let's consider whether there is some limit to how small the summary can be for t-approximate distance oracles.

Theorem 1.1. *Let $m_k(n)$ be a lower bound on the number of edges in an n-node graph of girth $\geq 2k + 2$. Then any t-approximate distance oracle for $t < 2k + 1$ must use at least $m_k(n)$ bits of storage space.*

Recall the Erdős conjecture that $m_k(n) = \Omega(n^{1+1/k})$. If this conjecture is true (and it is, for small k), then we would immediately have that the required amount of storage is at least $\Omega(n^{1+1/k})$. This suggests that graphs with low girth cannot be "compressed" very well.

Thorup and Zwick showed that one can almost achieve this lower bound while also keeping constant query time.

Theorem 1.2 (Thorup, Zwick '01)**.** *For all integers $k \geq 1$, there exists a $(2k - 1)$-approximate distance oracle using $\tilde{O}(k \cdot n^{1+1/k})$ space, $\tilde{O}(mn^{1/k})$ time for preprocessing, and can answer queries in $O(k)$ time.*

The $O(k)$ query time has recently been improved by Chechik to $O(1)$ time, keeping the space usage roughly the same.

We will now prove the lower bound on the space usage of any t-approximate distance oracle. Our proof will show the lower bound for distance oracles for unweighted graphs, but the same lower bound also holds for weighted graphs since we can always represent unweighted graphs as weighted graphs for which all edges have weight 1.

Proof of Theorem 1.1. 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. 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 $t < 2k + 1$. If we define D_{H_1} and D_{H_2} to be the functions used by distance oracles on H_1 and H_2 for answering distance queries, then we can write the inequality

$$d(u, v) = 1 \leq D_{H_1}(u, v) \leq t < 2k + 1 \leq D_{H_2}(u, v).$$

The first and second inequalities are trivial since $(u, v) \in H_1$. The third inequality is more subtle. If this final inequality did not hold, then there would exist a cycle containing the edge (u, v) of length at most $2k + 1$ that uses the shortest path from u to v in H_2 (of length $\leq 2k$) and then traverses the edge from u to v in H_1. Of course, G has girth $\geq 2k + 2$, so the existence of such a length $2k + 1$ cycle is a contradiction. In particular, we have shown that $D_{H_1}(u, v) \neq D_{H_2}(u, v)$. Since the distance oracle uses the same query algorithm for both H_1 and H_2, and the query algorithm runs only differ on the storage spaces used, this means that different storage spaces must be used for H_1 and H_2 in any distance oracle! Since there are $2^{m_k(n)}$ subgraphs of G, we have that the storage space required is $\log 2^{m_k(n)} = m_k(n)$. □

2 An efficient 3-approximate distance oracle

We discuss the algorithm that achieves the result in Theorem 1.2.

Let us consider the special case of Theorem 1.2 for $k = 2$. Here, we want to find a 3-approximate distance oracle that uses $\tilde{O}(n^{1+1/2}) = n^{\tilde{1.5}}$ space.

The algorithm is as follows. We first let A be a random subset of V of size $O(\sqrt{n} \log n)$, and for each $v \in V$, $N_{\sqrt{n}}(v)$ be the set containing the closest $\sqrt{n}$ nodes to v. By the hitting set results we showed in previous lectures, we have that A hits $N_{\sqrt{n}}(v)$ for all v with high probability.

Now define $A(v) = \{x \in V \mid d(v, x) < d(v, p_A(v))\}$ where $p_A(v)$ is the closest node to v in A.

Notice that $A(v)$ are exactly the **nodes closer to v than all nodes of A**.

Finally, let $B(v) = A \cup A(v)$ for all v.

Then $\forall\ v \in V$, $\forall\ x \in B(v)$ we store $d(v, x)$ in a hash table. We also store for each v, $p_A(v)$.

Claim 1. $|B(v)| \leq O(\sqrt{n} \log n)$.

Proof. It suffices to show $|A(v)| \leq \sqrt{n}$ since by construction $|A| = O(\sqrt{n} \log n)$. 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, $|A(v)| \leq |N_{\sqrt{n}}(v)| = \sqrt{n}$. □

Claim 1 implies the storage space required by our distance oracle is $\tilde{O}(n \cdot \sqrt{n})$.

For the query time, consider a query (u, v). 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 $p_A(u)$ and returns $d(u, p_A(u)) + d(p_A(u), v)$. This is possible since $p_A(u) \in A \subset B(x)$ for all $x \in V$. Therefore, queries are responded to in constant time.

We are only left with proving that the query responses satisfy

$$d(u, v) \leq D(u, v) \leq 3d(u, v). \tag{1}$$

By the triangle inequality we have

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

This proves the first inequality in (1) holds. For the second, if $v \in B(u)$, the the query algorithm will return $D(u, v) = d(u, v)$. Otherwise, if $v \notin B(u)$, we again use the triangle inequality to find that

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

where $d(u, p(u)) \leq d(u, v)$ since $v \notin B(u)$. This proves the second inequality in (1) holds.

3 An efficient $(2k-1)$-approximate distance oracle

The algorithm for general k proceeds by instead of taking a single sample A of nodes, taking many related 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. Also, if $d(v, p_i(v)) = d(v, p_{i+1}(v))$ then 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 $i \leq k-1$, $p_i(v)$.

Claim 2. $\forall\ i, v,\ p_i(v) \in B(v)$.

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

Note that Claim 2 implies that we have stored in memory $d(v, p_i(v))$ for all i.

Claim 3. $\forall\ v,\ |B(v)| \leq \tilde{O}(k \cdot n^{1/k})$.

Proof. It suffices to show that for all i $|A_i(v)| \leq \tilde{O}(n^{1/k})$ since $|A_{k-1}| \leq \tilde{O}(n^{1-(k-1)/k}) = \tilde{O}(n^{1/k})$. Let $N_i(v)$ be the set containing 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 $N_i(v)$ (i.e. there is a node in A_{i+1} that is also in $N_i(v)$). By the definition of $A_i(v)$, similar to our proof for the $k = 2$ case, this immediately implies $A_i(v) \subseteq N_i(v)$. We can then conclude that $|A_i(v)| \leq |N_i(v)| = n^{1/k}$. □

Note that Claim 3 shows that the space required by the distance oracle is $\tilde{O}(k \cdot n^{1+1/k})$.

The distance responds to queries using the following algorithm:

Algorithm 1: Query$((u,v))$

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

The intuition here is that we want to find the smallest $p_i(u) \in B(u) \cap B(v)$.

First notice that in iteration i of the For loop, $w = p_{i-1}(v)$. Second, notice 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 $w \in B(u)$.

Also, 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.

Next class we will show that the estimate is also $\leq (2k-1)d(u,v)$. To do that it will suffice to prove the following lemma.

Lemma 3.1. *If the algorithm is in iteration i, then $d(v,w) \leq (i-1) \cdot d(u,v)$. Furthermore, if the algorithm is in iteration i and $w \in B(u)$, then we return a distance $\leq (2i-1) \cdot d(u,v)$. Otherwise, if $w \notin B(u)$, then $d(u,p_i(u)) \leq i \cdot d(u,v)$.*

Consider now how the lemma implies that $D(u,v) \leq (2k-1)d(u,v)$. This will be by induction. The base case is that before the For loop we have $d(v,w) = d(v,v) = 0 \leq 0 \cdot d(u,v)$. By induction we get that if the first iteration for which $w \in B(u)$ is j, then for all $i < j$ we get $d(u,p_i(u)) \leq i \cdot d(u,v)$. In particular, in the beginning of iteration j we get $d(v,w) \leq (j-1) \cdot d(u,v)$. Again by the lemma, we obtain that the algorithm returns $D(u,v) \leq (2j-1)d(u,v)$. Since we've shown that the algorithm returns an estimate at worst in iteration k, we get $D(u,v) \leq (2k-1)d(u,v)$.