princeton univ. F'23 cos 521: Advanced Algorithm Design

Lecture 17: Distance Oracles

Lecturer: *Huacheng Yu*

Lecture notes are based on Virginia Vassilevska Williams' lecture notes here: http://theory.stanford.edu/ virgi/cs267/lecture8.pdf

In this lecture, we will talk about *distance oracles*. A distance oracle is a data structure that stores (a compressed version of) a graph G supporting efficient (approximate) distance queries.

1 t-approximate distance orcales

A t-approximate distance orcale is a data structure that is computed from an (undirected) input graph $G = (V, E)$ by a preprocessing algorithm such that given any two vertices $x, y \in V$, a query algorithm can efficiently output an estimate D that is between $d(x, y)$ and $t \cdot d(x, y)$. The query algorithm may only access the distance oracle but not the input graph G.

We want to construct, in low preprocessing time, a small data structure with fast query time. One solution is to use the spanners from the last lecture. Recall that we proved the following.

Theorem 1. *For any G, there exists a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges.*

Thus, the distance oracle will have size $\tilde{O}(n^{1+1/k})$. But the spanner does not tell us how to compute the pairwise distances, we would need to run BFS or Dijkstra's algorithm at query. The query time is slow.

In this lecture, we will prove that the same space bound can be achieved with a much faster query time.

Theorem 2 (Thorup, Zwick'01)**.** *For all integer $k \geq 1$, there exists a $(2k-1)$-approximate distance oracle using $\tilde{O}(n^{1+1/k})$ space, $\tilde{O}(m \cdot n^{1/k})$ preprocessing time and $O(k)$ query time.*

2 3-approximate distance orcales

To begin with, we will first present a 3-approximate distance orcale ($k = 2$), which demonstrates the main ideas. Here, the goal is to construct a $n^{1.5}$-space data structure with constant query time.

The main idea is that we will find a small set A of "hubs" in the graph, and route through these hubs if the query pair is far. We will remember the (exact) distances to all hubs from each vertex, which is cheap as $|A|$ is small. We will also guarantee that there are not too many close pairs, so that their distances can be remembered exactly in the data structure. It turns out that a random set A works.

Preprocessing algorithms. We first sample a random set $A \subseteq V$ of size $O(\sqrt{n}\log n)$. Then let $p(v)$ be the closest vertex to v in A. For each v, we will store the exact distance $d(v,x)$ for x such that

- $d(v,x) < d(v,p(v))$, or
- $x \in A$

using a hash table. Denote by the set of x that we store the distance to v by $B(v)$.

Space. The preprocessing algorithm does not store distances for too many pairs of vertices.

Lemma 3. *For every v, there exists at most $O(\sqrt{n})$ x such that $d(v,x) < d(v,p(v))$ with high prob.*

Proof. Fix v, let N be the set of $\sqrt{n}$ vertices closest to v (break ties arbitrarily). By the same argument as in the spanner lecture, we have $\Pr[A \cap N = \emptyset] \leq 1/n^2$. On the other hand, when A and N intersect, there can be at most $\sqrt{n}$ x such that $d(v,x) < d(v,p(v))$. By union bound over v, the lemma holds. □

Therefore, we have $|B(v)| \leq O(\sqrt{n}\log n)$ with high probability, which implies that the space is $\tilde{O}(n^{1.5})$.

Preprocessing. By running Dijkstra's algorithm from every vertex $x \in A$, as well as from every vertex $v \in V$ and stopping when it hits any vertex in A, we can show that every edge is used to update the distance only when one endpoint is in $B(v)$ for the current v. Hence, the total running time is $\tilde{O}(m\sqrt{n})$. The details are omitted.

Query algorithm. Given a query pair (u,v), we use the following algorithm to estimate their distance.

```
1. if $v \in B(u)$
2.     return $d(u,v)$
3. return $D = d(u,p(u)) + d(p(u),v)$
```

If $v \in B(u)$, we return the exact distance. Otherwise, by the triangle inequality, we have $D \geq d(u,v)$, and

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

where the last inequality uses the fact that $v \notin B(u)$, and hence $d(u,v) \geq d(u,p(u))$.

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

The idea for general k is to have multiple levels of hubs. We sample a large set of level-1 hubs A_1, then sample a subset of them to be level-2 hubs A_2, etc. The preprocessing algorithm will only store the distances to the "close" hubs in each level i, i.e., the ones that are closer to the closest level $i+1$ hubs.

Preprocessing algorithm. More concretely, let $A_0 = V$ be the set of all vertices, and sample $A_i \subseteq A_{i-1}$ of size $O(|A_{i-1}| \cdot n^{-1/k} \log n) = \tilde{O}(n^{1-i/k})$. Then let $p_i(v)$ denote the closest vertex in A_i to v, and let $A_i(v)$ be the set of vertices in A_i that are closer to v than $p_{i+1}(v)$. Finally, we store for each v, the distance $d(v, x)$ for all x in

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

in a hash table.

By a similar argument to Lemma 3, we obtain the following bound on $|A_i(v)|$.

Lemma 4. *For each v and i, $|A_i(v)| \le \tilde{O}(n^{1/k})$ with high probability.*

We omit the proof here. The lemma implies that for each v, $|B(v)| \le \tilde{O}(k \cdot n^{1/k})$. The total size is at most $\tilde{O}(k \cdot n^{1+1/k})$ with high probability.

Query algorithm. To answer a query, we iteratively check if we can route through $p_i(v)$ or $p_i(u)$.

```
1. for $i = 1, \ldots, k$
2.     if $p_{i-1}(u) \in B(v)$, return $D = d(u, p_{i-1}(u)) + d(p_{i-1}(u), v)$
3.     if $p_{i-1}(v) \in B(u)$, return $D = d(u, p_{i-1}(v)) + d(p_{i-1}(v), v)$
```

Note that the algorithm will terminate when $i = k$, since $p_{k-1}(u) \in A_{k-1} \subseteq B(v)$.

The error bound is guaranteed by the following lemma. Intuitively, it says that when $p_{i-1}(u) \notin B(v)$ and $p_{i-1}(v) \notin B(u)$, u and v must be relatively far, thus, the distance to the closest level i hub is not too large compared to $d(u, v)$.

Lemma 5. *If the algorithm is in iteration i, it must have*

$$\begin{aligned} d(u, p_{i-1}(u)) &\le (i-1) \cdot d(u, v) \\ d(v, p_{i-1}(v)) &\le (i-1) \cdot d(u, v). \end{aligned} \tag{1}$$

Moreover, if $p_{i-1}(u) \in B(v)$ or $p_{i-1}(v) \in B(u)$, $D \le (2i-1) \cdot d(u, v)$. Otherwise, $d(u, p_i(u)) \le i \cdot d(u, v)$ and $d(v, p_i(v)) \le i \cdot d(u, v)$.

In particular, the lemma implies that $D \le (2k-1) \cdot d(u, v)$.

Proof. We prove (1) by induction. When $i = 1$, since $p_0(u) = u$ and $p_0(v) = v$. The inequalities hold.

Then if $p_{i-1}(u) \in B(v)$, the query algorithm returns $D = d(u, p_{i-1}(u)) + d(p_{i-1}(u), v) \le 2d(u, p_{i-1}(u)) + d(u, v) \le (2i-1)d(u, v)$. The $p_{i-1}(v) \in B(u)$ case is similar.

Otherwise, $d(v, p_{i-1}(u)) \ge d(v, p_i(v))$, and we have

$$d(v, p_i(v)) \le d(v, p_{i-1}(u)) \le d(u, v) + d(u, p_{i-1}(u)) \le i \cdot d(u, v).$$

It is similar to show that $d(u, p_i(u)) \le i \cdot d(u, v)$. This proves the lemma. □