COL751 - Lecture 5

1 Distance Oracle with stretch 3

In this lecture we will see construction of distance oracle of sub-quadratic size that for any query vertex pair (x, y), reports approximate distance between x and y in an input weighted graph G.

Before moving to the oracle construction details, it's important to note that none of the tools covered in the initial four lectures are adequate for computing an approximate distance oracle with constant query time. This is because tools till now used the idea of partitioning the vertex set V into subsets of low and high degrees, where edges incident to low-degree vertices could be counted for free. What if graph is already sparse with each vertex having just $O(1)$ edges? Spanners for such graphs are trivially the original graph itself; and we can even construct distance matrix in $O(n^2)$ time. However, the distance oracle problem is still non-trivial for sparse graphs.

We explore an alternative tool of working with truncated BFS trees. For any given vertex $x \in V$, the truncated BFS of x of size t - denoted as $BFS(x, t)$ - is the tree obtained by retaining the closest t vertices to x. The second idea that we will explore in this lecture is of hierarchical construction, this will be useful to obtain size stretch trade-offs for distance oracle problem.

Static Perfect Hashing In oracle constructions, one of the crucial data-structure used is hashing. Consider a scenario where we are given a set S of *items*, where each item is a (*key,value*) pair. Keys are from an arbitrary universe U. The task is to build a data structure that can efficiently answer the following query.

Search(k): For a given k, if the key k is present in S, return the value associated with the key k in S; otherwise return NULL.

The celebrated work by Fredman, Komlós and Szemerédi (1984) provides a data structure of size $O(|S|)$ with worst-case query time $O(1)$. The time to compute data-structure is $O(|S|)$ on expectation.[1]

Dynamic Perfect Hashing An extension to Static perfect hashing is the Dynamic perfect hashing, which will be used later in our course. Dietzfelbinger et al. (1994) showed that there exists a dynamic hashing algorithm in which membership queries always run in $O(1)$ worst-case time, the total space required is $O(|S|)$ (i.e. linear in number of items), and insertion and deletion takes $O(1)$ expected amortized time.

[1] We will not look into algorithm to compute hashing in this course, but will use static/dynamic perfect hashing as a black-box data-structures.

Oracle Construction Let us now see construction of distance oracle with multiplicative stretch 3.

- For each $x \in V$, we compute a (static) perfect hash-table for vertices in $BFS(x, \sqrt{n})$ so that membership in it can be verified in constant time; also we store in hash table the values $dist(x, y)$ for each y in $BFS(x, \sqrt{n})$.

- Next we compute a random subset R of vertices of size $O(\sqrt{n} \log n)$, and store a distance matrix for pairs in $R \times V$.

  Remark: With probability at least $1 - \frac{1}{n^2}$, the set R will have non-empty intersection with $BFS(x, \sqrt{n})$, for each $x \in V$. (For proof see lecture 2, Practice sheet 1).

- Finally, for each $x \in V$, we store a representative vertex r_x in $R \cap BFS(x, \sqrt{n})$.

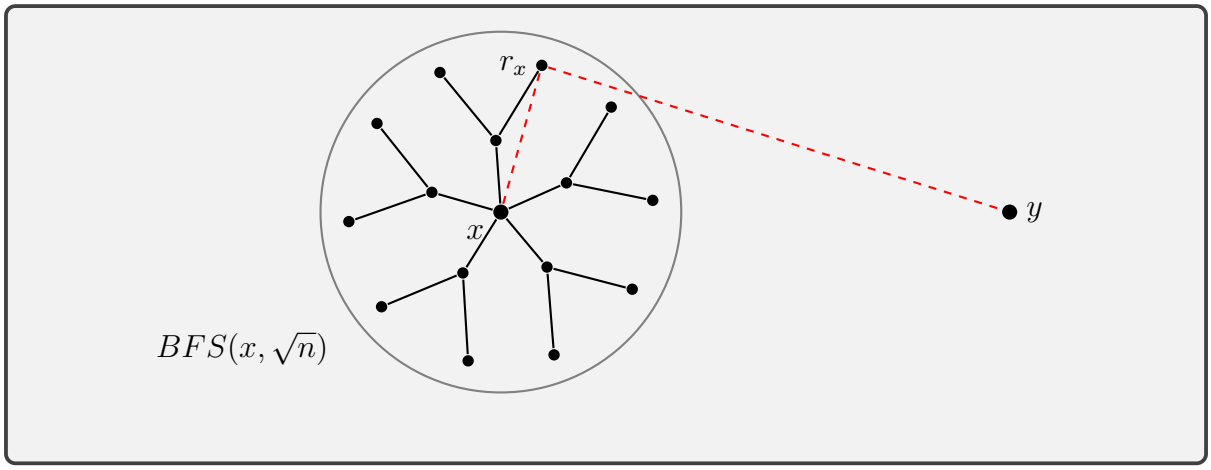
Query Oracle Consider a vertex pair (x, y). The steps to compute (x, y) approximate distance are:

1. Use hash-table of x to check in $O(1)$ time whether $y \in BFS(x, \sqrt{n})$. If yes, report the exact distance.

2. If $y \notin BFS(x, \sqrt{n})$, then we return $\widehat{d}(x, y)$ as

$$dist(x, r_x) + dist(r_x, y),$$

   where, r_x is representative vertex corresponding to node x lying in $R \cap BFS(x, \sqrt{n})$.



Lemma 1 *The multiplicative stretch of query oracle explained above is* 3.

Proof: If $y \in BFS(x, \sqrt{n})$, then we simply report exact distance using the hash table stored for x.

Let us next consider the case of $y \notin BFS(x, \sqrt{n})$. In this case observe that $dist(x, r_x)$ is at most $dist(x, y)$ as r_x lies in $BFS(x, \sqrt{n})$, whereas, y does not. Therefore,

$$\begin{aligned} \widehat{d}(x,y) &= dist(x,r_x) + dist(r_x,y) \\ &\leqslant dist(x,r_x) + dist(r_x,x) + dist(x,y) \quad \text{(by Triangle Inequality)} \\ &\leqslant 3 \cdot dist(x,y). \end{aligned}$$

This proves that the stretch of oracle is 3. □

Theorem 2 (Thorup, Zwick (STOC'01)) *For any n vertex weighted undirected graph $G = (V, E)$ we can compute in polynomial time an approximate distance oracle of size $O(n^{1.5} \log n)$ that for any query vertex pair $(x, y) \in V \times V$, reports 3-approximate (x, y) distance in $O(1)$ time.*

Remark This also provides an alternate construction of 3-spanner as we just need to store truncated BFS trees of $\sqrt{n}$ size for each each $x \in R$, and the shortest path tress rooted at vertices $r \in R$.

2 Distance oracle for $Z \times V$

Let us suppose we are only interested in querying 3-approximate distances between pairs in set $Z \times V$, for some subset Z of size k. Then, we can modify the earlier oracle construction as below.

- For each $x \in Z$, we compute a perfect hash-table for vertices in $BFS(x, n \log n/\sqrt{k})$ so that membership in it can be verified in constant time; also we store in hash table the values $dist(x, y)$ for each y in $BFS(x, n \log n/\sqrt{k})$.

- Next we compute a random subset R of vertices of size $O(\sqrt{k})$, and store a distance matrix for pairs in $R \times V$.

  Remark: With probability at least $1 - \frac{1}{n^2}$, the set R will have non-empty intersection with $BFS(x, n \log n/\sqrt{k})$, for each $x \in Z$.

- Finally, for each $x \in Z$, we store a representative vertex r_x in $R \cap BFS(x, n \log n/\sqrt{k})$.

The query process of our oracle is same as earlier, and the size of our oracle is $O(n\sqrt{k} \log n)$.

Theorem 3 *For any n vertex weighted undirected graph $G = (V, E)$ and any subset Z of vertices of size k we can compute in polynomial time an approximate distance oracle of size $O(n\sqrt{k} \log n)$ that for any query vertex pair $(x, y) \in Z \times V$, reports 3-approximate (x, y) distance in $O(1)$ time.*

3 Distance Oracle of $O(n^{4/3})$ size

We now see how to use the idea of hierarchical computation to obtain a distance oracle of $O(n^{4/3})$ size.

- For each $x \in V$, we compute a hash-table for vertices in $BFS(x, n^{1/3} \log n)$ so that membership in it can be verified in constant time; also we store in hash table the values $dist(x, y)$ for each y in $BFS(x, n^{1/3} \log n)$.

- Next we compute a random subset R of vertices of size $O(n^{2/3})$.

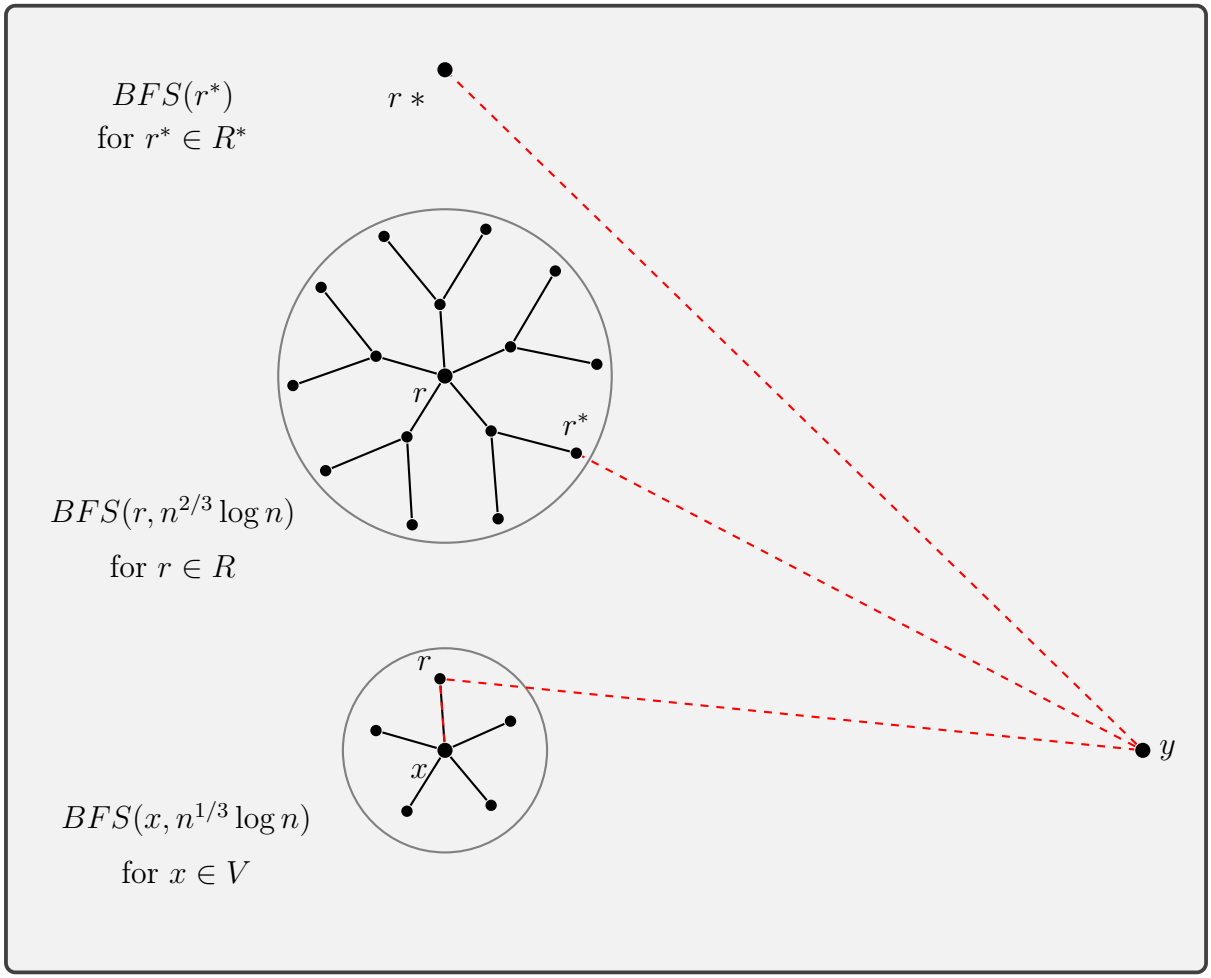
  – For each $r \in R$, we compute a hash-table for vertices in $BFS(r, n^{2/3} \log n)$; also we store in hash table the values $dist(r, y)$ for $y \in BFS(r, n^{2/3} \log n)$.

  – For each $x \in V$, we store a representative vertex r_x in $R \cap BFS(x, n^{1/3} \log n)$.

- Finally, we compute a random subset R^* of vertices of size $O(n^{1/3})$.

  – We store the entire distance matrix $R^* \times V$.

  – For each $r \in R$, we store a representative vertex r^* in $R^* \cap BFS(r, n^{2/3} \log n)$.



Lemma 4 *The total space taken by our data-structure is* $O(n^{4/3} \log n)$.

Query Oracle Consider a vertex pair (x, y). The steps to compute (x, y) approximate distance are:

1. Use hash-table of x to check in $O(1)$ time whether $y \in BFS(x, n^{1/3} \log n)$. If yes, report the exact distance.

2. If $y \notin BFS(x, n^{1/3} \log n)$, then we compute a vertex $r \in R \cap BFS(x, n^{1/3} \log n)$.

3. Next use hash-table of r to check in $O(1)$ time whether $y \in BFS(r, n^{2/3} \log n)$. If yes, report the distance as $dist(x, r) + dist(r, y)$.

4. If $y \notin BFS(x, n^{1/3} \log n)$, then we compute a vertex $r^* \in R^* \cap BFS(r, n^{r/3} \log n)$.

5. Finally, we report $dist(x, r) + dist(r, r^*) + dist(r^*, y)$

Homework 1 Prove that stretch of oracle is 7.

Homework 2 Extend this idea to compute a distance oracle of size $O(kn^{1+1/k} \log n)$, stretch O(function of k), and $O(k)$ worst case query time.

Extra Reading For distance oracle constructions of size $O(kn^{1+1/k} \log n)$, but with improved stretch see here and here.

4 Oracle of stretch 5, and size $O(n^{4/3})$

We slightly modify the construction in Section 1 to get an improved distance oracle of size $O(n^{4/3})$. The construction is as follows:

- For each $x \in V$, we compute a hash-table for vertices in $BFS(x, n^{1/3} \log n)$; also we store in hash table the values $dist(x, y)$ for each y in $BFS(x, n^{1/3} \log n)$.

- Next we compute a random subset R of vertices of size $O(n^{2/3})$, and store a distance matrix for pairs in $R \times R$.

- Finally, for each $x \in V$, we store a representative vertex r_x in $R \cap BFS(x, n^{1/3} \log n)$.

Query Oracle Consider a vertex pair (x, y). The steps to compute (x, y) approximate distance are:

1. Use hash-table of x to check in $O(1)$ time whether $y \in BFS(x, n^{1/3} \log n)$. If yes, report the exact distance.

2. Use hash-table of y to check in $O(1)$ time whether $x \in BFS(y, n^{1/3} \log n)$. If yes, report the exact distance.

3. If $y \notin BFS(x, n^{1/3}\log n)$ and $x \notin BFS(y, n^{1/3}\log n)$, then we return $\widehat{d}(x,y)$ as

$$dist(x, r_x) + dist(r_x, r_y) + dist(r_y, y),$$

where, $r_x \in R$ is representative vertex corresponding to node x, and $r_y \in R$ is representative vertex corresponding to node y.

Lemma 5 *The multiplicative stretch of query oracle explained above is* 5.

Proof: If $y \in BFS(x, n^{1/3}\log n)$ or $x \in BFS(y, n^{1/3}\log n)$, then we simply report exact distance.

Let us next consider the case that $y \notin BFS(x, n^{1/3}\log n)$ and $x \notin BFS(y, n^{1/3}\log n)$.

In this case observe that $dist(y, r_y)$, $dist(x, r_x)$ is at most $dist(x, y)$. Therefore,

$$\begin{aligned}
\widehat{d}(x,y) &= dist(x, r_x) + dist(r_x, r_y) + dist(r_y, y) \\
&\leqslant 2 \cdot dist(x, r_x) \; + \; dist(x, y) \; + \; 2 \cdot dist(r_y, y) \;\; \text{(by Triangle Inequality)} \\
&\leqslant 5 \cdot dist(x, y).
\end{aligned}$$

This proves the claim. □

Theorem 6 (Thorup, Zwick (STOC'01)) *For any n vertex weighted undirected graph $G = (V, E)$ we can compute in polynomial time an approximate distance oracle of size $O(n^{4/3}\log n)$ that for any query vertex pair $(x, y) \in V \times V$, reports 5-approximate (x, y) distance in $O(1)$ time.*