In this lecture we will continue the discussion of additive spanners, completing the description of $+4$-spanners. Then we will begin the topic of approximate distance oracles.

1 Additive 4-Spanners

In this section, a node in an $n$-node graph is high degree if it has degree greater than $n^{2/5}$ and is low degree otherwise.

**Theorem 1.1.** There exists an $\tilde{O}(n^{7/5})$ edge additive 4-spanner in any $n$-node $m$-edge $G = (V,E)$.

**Proof.** The algorithm for constructing a $+4$-spanner $H$ is as follows:

1. For every low degree $v \in V$ (degree $\leq n^{2/5}$) add $v$’s edges to $H$. This results in $\leq n \cdot n^{2/5} = n^{7/5}$ edges.

2. Let $S$ be a random subset of $V$ on $O(n^{2/5} \log n)$ nodes. For each $s \in S$, add the entire breadth-first-search tree rooted at $s$ into $H$. Here, we add $O(n^{2/5} \log n \cdot n) \in \tilde{O}(n^{7/5})$ edges because each BFS-tree has at most $n$ edges.

3. Let $T$ be a random sample on $O(n^{3/5} \log n)$ nodes. With high probability, $T$ hits the neighborhood of each high degree node. For each high degree node $v$, we pick one of its neighbors in $T$, $t(v)$ and we add $(v, t(v))$ to $H$ (adding $O(n)$ edges; $t(v)$ exists for each high degree $v$ since $T$ is a hitting set).

4. For each $t \in T$, we define $t$’s ball as the set of high degree vertices assigned to it:
   
   \[ B(t) = \{ v \in V | v \text{ high degree }, t = t(v) \}. \]

   We continue adding edges to $H$ with Algorithm 1.

```
Algorithm 1: Adding final edges to $H$

for each distinct $t, t' \in T$ do
  $P_{t,t'} = \{}$
  for each $u \in B(t), v \in B(t')$ do
    if shortest path $P$ from $u \rightarrow v$ has $< n^{1/5}$ high degree nodes then
      $P_{t,t'}$.insert($P$);
    if $P = \{}$ then
      break;
    $p =$ shortest path in $P_{t,t'}$;
    for each edge $e \in p$ do
      $H$.insert($e$);
```

The path added to $H$ corresponding to $t$ and $t'$ will be referred to as the $(t, t')$-linking path.

How many edges are added to $H$? There are $O(n^{6/5})$ pairs of $t, t'$. For each pair, we may add the edges from some path $P$ connecting $u$ and $v$ in $B(t)$ and $B(t')$ respectively. The only edges in $P$ not already in $H$ are those between two high degree nodes (by Step (1)), of which there are $\leq n^{1/5}$ in $P$. Thus each pair $t, t'$ adds $O(n^{1/5})$ edges to $H$. Summing over the $O(n^{6/5})$ pairs, $O(n^{7/5})$ edges are added.

---

1 The constant behind the big $O$ needs to be picked as sufficiently large for a hitting set argument appearing later to work.
We have constructed $H$ to be of size $\tilde{O}(n^{7/5})$. It remains to show that $H$ is an additive 4-spanner of $G$.

If a pair of vertices $u$ and $v$ in $G$ have a shortest path using no high degree nodes, then that path is in $H$ due to Step (1).

The following claim helps resolve the case where $u$ and $v$ have a shortest path using more than $n^{1/5}$ high degree nodes.

**Claim 1.** Let $P$ be a $uv$-shortest path for some $u, v \in V$. If $P$ has more than $n^{1/5}$ high degree nodes, then $P$ has at least $\Omega(n^{3/5})$ distinct neighbors.

**Proof.** Each node $x$ in $G$ can have at most 3 neighbors in $P$. Otherwise, $P$ could be shortened by taking a path through $x$, a contradiction since $P$ is a $uv$-shortest path for some $u, v \in V$. Since $P$ has more than $n^{1/5}$ high-degree nodes, each of which has at least $n^{2/5}$ neighbors, it follows that $P$ has more than $n^{3/5}$ edges incident on it. Each neighbor of $P$ can account for at most 3 of these edges, and hence $P$ must have at least $\Omega(n^{3/5})$ distinct neighbors.

It follows from Claim 1 that for every $u, v \in V$ with a shortest path hitting more than $n^{1/5}$ high degree nodes, $S$ hits a neighbor node of a shortest path from $u$ to $v$. In other words, such paths from $u$ to $v$ each have enough distinct neighbor nodes ($\Omega(n^{3/5})$) that $S$ forms a hitting set on them. Thus the edges added in Step (2) include a +2-approximation for a shortest path between such $u$ and $v$.

The only remaining case is $uv$-shortest paths hitting between $1$ and $n^{1/5}$ high degree nodes.

**Claim 2.** If a $uv$-shortest path hits between $1$ and $n^{1/5}$ high degree nodes, then after Step (4), $d_H(u, v) \leq d(u, v) + 4$.

**Proof.** Let $P$ be a shortest path from $u$ to $v$ hitting between $1$ and $n^{1/5}$ high degree nodes. Let $h_1$ be the first and $h_2$ be the last high degree nodes in $P$ (possibly not distinct). Let $t_1 = t(h_1)$ and $t_2 = t(h_2)$. Let $x$ and $y$ be the neighbors of $t_1$ and $t_2$ respectively connected by the $(t_1, t_2)$-linking path. Note that the $(t_1, t_2)$-linking path exists because there is at least one pair of elements of $B(t_1)$ and $B(t_2)$ connected by a shortest path using at most $n^{1/5}$ nodes, namely $h_1$ and $h_2$.

Since the subset of $P$ from $u$ to $h_1$ and from $h_2$ to $v$ is in $H$ (Step (1)), we have

$$d_H(u, v) \leq d(u, h_1) + d_H(h_1, h_2) + d(h_2, v).$$

Thus we need only show that $d_H(h_1, h_2) \leq d(h_2, h_2) + 4$.

Observe that

$$d_H(h_1, h_2) \leq d(h_1, x) + d(x, y) + d(y, h_2) = d(x, y) + 4.$$

Recall that $x$ and $y$ have the shortest path between them in $G$ of any pair of elements in $B(t_1)$ and $B(t_2)$, excluding paths with greater than $n^{1/5}$ high degree nodes. Therefore, $d(x, y) \leq d(h_1, h_2)$. Plugging this in yields

$$d_H(h_1, h_2) \leq d(h_1, h_2) + 4,$$

completing the proof. □
2 \( t \)-approximate distance oracles

Let \( G = (V, E) \) be an \( n \)-node \( m \)-edge graph (with possibly non-negative weights). In this section, such \( G \) are the inputs of \( t \)-approximate distance oracles.

A \( t \)-approximate distance oracle is is defined by its preprocessing algorithm and query algorithm:

**Preprocessing algorithm:** Given \( G \), produces a summary \( S(G) \) and stores it in memory.

**Query Algorithm:** Inputs \((u, v, S(G))\) where \(u, v \in V\) and outputs \(D(u, v)\) satisfying

\[
d(u, v) \leq D(u, v) \leq t \cdot d(u, v).
\]

A \( t \)-approximate distance oracle has three properties:

1. Preprocessing time \(T(m, n)\).
2. Query time (preferably constant) \(q(n)\).
3. Storage space \(|S(G)|\).

**Example 1.** Multiplicative spanner trees are an example of distance oracles. They can do well on properties (1) and (3), but do poorly on (2).

Eventually, we will prove the following theorem of Thorup and Zwick (not today though):

**Theorem 2.1.** For all \( k \geq 1 \), there is a \((2k - 1)\)-distance oracle with space \(O(k \cdot \tilde{n}^{1+1/k})\), query time \(O(k)\), and preprocessing time \(\tilde{O}(mn^{1/k})\).

First, however, we will consider the limitations of distance oracles in terms of space usage. Recall the following conjecture of Erdős:

**Conjecture 1.** For all \( k \) and large enough \( n \), there exists \( n \)-node graphs on \(\Omega(n^{1+1/k})\) edges with girth at least \(2k + 2\).

Combining this with the following cool result bounds how well a \( t \)-approximate oracle can do at space usage.

**Theorem 2.2.** Let \( m_k(n) \) be the minimum \# of edges such that for all \( k \), there exists an \( n \)-node graph on \( m_k(n) \) edges with girth at least \(2k + 2\). Then any \( t \)-approximate distance oracle algorithm with \( t < 2k + 1 \) must use at least \( m_k(n) \) bits of space for some \( n \)-node graph \( G \) for some \( n \).

**Proof.** Let \( D \) by be a \( t \)-approximate distance oracle with less than \( m_k(n) \) bits of storage space for some graph \( G \) having at least \( m_k(n) \) edges and girth at least \(2k + 2\). Observe that \( G \) has at least \( 2^{m_k(n)} \) subgraphs. If the distance oracle uses less than \( m_k(n) \) bits on \( n \) node graphs, then some two subgraphs of \( G \), \( H_1 \) and \( H_2 \), must use the same storage \( S(H_1) = S(H_2) \) (by the Pigeon Hole Principle).

Since \( H_1 \neq H_2 \), we can assume without loss of generality that some edge \((u, v)\) is in \( H_1 \) but not \( H_2 \). Thus \( D_{H_1}(u, v) = 1 \). But \( D_{H_2}(u, v) \geq 2k + 1 \) because \( G \) has girth at least \(2k + 2\). Indeed, if there were a path from \( u \) to \( v \) in \( H_2 \) of fewer than \(2k + 1\) nodes, then adding the edge \((u, v)\) would result in a cycle of size less than \(2k + 2\), a contradiction.

Since \( D \) is a \( t \)-approximate distance oracle and \( S(H_1) = S(H_2) \), it follows that a query of \((u, v, S(H_1))\) yields the same result as a query of \((u, v, S(H_2))\) which returns a value at least \(2k + 1\). This is a contradiction since \( t < 2k + 1 \), whereas a query of \((u, v, S(H_1))\) needs to return a value at most \( t \cdot 1 < 2k + 1 \), but we showed that the query needs to be at least \(2k + 1\).

It follows from Conjecture 1 and theorem 2.2, that the space usage in Theorem 2.1 is in some sense close to optimal, even for unweighted graphs.

We will now direct our attention to the special case of 3-approximate distance oracles (i.e., Theorem 2.1 for \( k = 2 \)).
Theorem 2.3. There is a 3-approximate distance oracle with

- Storage $|S(G)| \in \tilde{O}(n^{3/2})$;
- Query time $q(n) \in O(1)$;
- Preprocessing time $T(m, n) \in \tilde{O}(m \sqrt{n})$.

Proof. Preprocessing Phase: For all $v \in V$, let $T_v$ be the closest $\sqrt{n}$ nodes to $v$ (ties broken arbitrarily). Let $S$ be a hitting set for $\{T_v | v \in V\}$ of size $O(\sqrt{n} \log n)$. For all $v \in V$, let $p(v)$ be the closest node to $v$ in $S$ (ties broken arbitrarily). Let the ball of $v$ (not the same as defined in the previous section) be

$$B(v) = \{u \in V | d(v, u) < d(v, p(v))\}.$$ 

Moreover, let $\hat{B}(v) = B(v) \cup S$.

The preprocessing algorithm is Algorithm 2.

Algorithm 2: Preprocessing($G$)

```plaintext
foreach $v \in V$ do
    Store in $S(G)$ $p(v)$;
    foreach $u \in \hat{B}(v)$ do
        Store in $S(G)$ $d(u, v)$.
```

For each $v \in V$, $|\hat{B}(v)| = |B(v)| + |S|$. Since $s$ is among the $\sqrt{n}$ closest nodes to $v$, $|B(v)| \leq |T_v| \leq \sqrt{n}$. Thus $|\hat{B}(v)| \in O(\sqrt{n} \log n)$. Summing over $v \in V$, space usage is $|S(G)| \in \tilde{O}(n^{3/2})$.

Queries:

Algorithm 3: Query($u, v, S(G)$)

```plaintext
if $u \in \hat{B}(v)$ then
    return $d(u, v)$;
else
    Lookup $p(v)$ in $S(G)$;
    return $d(u, p(v)) + d(p(v), v)$;
```

Observe that if $u \in \hat{B}(v)$, then $d(u, v) \in S(G)$ and that otherwise, $d(u, p(v))$ and $d(p(v), v)$ are in $S(G)$. The following claim establishes that the query algorithm is correct.

Claim 3. If $u \notin \hat{B}(v)$, then $d(u, p(v)) + d(p(v), v) \leq 3 \cdot d(u, v)$.
Proof. Let $d_1 = d(u, v), d_2 = d(u, p(v)), d_3 = d(v, p(v))$ (See Figure 1). If $u \notin \hat{B}(v)$, then $d_1 \geq d_3$.

By the triangle inequality,
$$d_2 + d_3 \leq (d_1 + d_3) + d_3 = d_1 + 2d_3.$$

Since $u \notin \hat{B}(v)$, we have that $d_1 \geq d_3$. Thus
$$d_1 + 2d_3 \leq 3d_1,$$
completing the proof. □

Left to the reader: To show that preprocessing time is in $\tilde{O}(m\sqrt{n})$. □