Distance Oracles for Sparse Graphs

SPUR Final Paper, Summer 2014
Georgios Vlachos
Mentor: Adrian Vladu
Project suggested by Adrian Vladu

July 30, 2014

Abstract

Suppose you are given a large network, such as the Internet or the US road network. In many practical applications, one might be interested in querying shortest path distances between pairs of nodes of the network. However, attempting to implement a data structure that supports such operations will result either in a large query time, or large space requirement, both of which are undesirable. In order to overcome this barrier, better time vs. space trade-offs can be achieved by settling for data structures that return approximate distances.

More formally, let $G(V, E)$ be an undirected graph with nonnegative edge weights. A stretch k distance oracle is a compact data structure that can efficiently answer approximate distance queries between any pair of vertices: given a pair (u, v) it returns an estimate $\delta(u, v)$ satisfying $d(u, v) \leq \delta(u, v) \leq k \cdot d(u, v)$, where d is the shortest path metric in G. This data structure has been introduced by Thorup and Zwick [TZ01], who showed how to design stretch $2k-1$ distance oracles, using space $O(|V|^{1+\frac{1}{k}})$ and constant query time.

Although their trade-off is optimal for general graphs, assuming the Girth Conjecture of Erdös, little is known about the regime of sparse graphs, which appear in most applications. Agarwal and Godfrey [AG13] presented one such result, which we improve upon.

To summarize, our contribution is three-fold:

- We improve on the distance oracles of Agarwal and Godfrey. They present stretch $1 + \frac{2}{t}$ distance oracles using space $O(\frac{n^2}{a})$ and query time $\widetilde{O}(a^t)$ [1]. Under the same space-time trade-off, we improve the stretch to $1 + \frac{1}{t}$.
- We introduce a new approach for clustering for *sparse* graphs, that we apply to known algorithms. For sparse graphs, there is a stretch 3 distance oracle that uses space $O(n^{2-2\epsilon})$ and has query time $\widetilde{O}(n^{\epsilon})$, for $\epsilon \in [0, \frac{1}{2}]$. Applying our clustering to the previous algorithm we obtain, for degree k expander graphs of expansion c, distance oracles with space $\widetilde{O}(n^{2-2\epsilon})$ and query time $O(n^{\epsilon - \log_k c(1-3\epsilon)})$ for $\epsilon \in [\frac{1}{4}, \frac{1}{3}]$.
- We reduce a class of set intersection problems to distance oracles of stretch less than $1 + \frac{1}{t}$, for all integers $t \geq 1$.

[1]We write $f(n) = \widetilde{O}(n)$ iff $f(n) = O(n \log^{\beta} n)$ for some constant β.

1 Introduction

In their seminal paper from 2001, Thorup and Zwick introduce *distance oracles*, a compact data structure that can report approximate distances between nodes in graphs [TZ01]. More specifically, given an undirected weighted graph $G = (V, E)$, we wish to create a data structure which, whenever queried for a pair of nodes (u, v), *efficiently* returns an approximate distance $\delta(u, v)$ satisfying $d(u, v) \leq \delta(u, v) \leq k \cdot d(u, v)$, where d is the shortest path metric in the graph, and k is an approximation factor which we will refer to as *stretch*.

Trivially, one could simply store all the $|V|^2$ pairwise distances in a matrix, and report distances in constant time. However, for practical purposes (road networks, routing on the internet), these matrices become prohibitively large. On the other side of the trade-off, we could simply store the graph, and query distances by running Dijkstra's algorithm; but in this case the query time becomes impractical.

It is thus necessary to relax the requirements of the problem, and trade the quality of the result and the query time in exchange for smaller storage space.

Thorup and Zwick managed to achieve, for all positive integers k, distance oracles of size $O(kn^{1+1/k})$ that return distances of stretch $2k - 1$ in time $O(k)$. They argued that their space-stretch trade-off is nearly optimal by showing that if a well known conjecture of Erdos (the Girth Conjecture) holds, then there are graphs with $\Theta(n^{1+1/k})$ edges for which any stretch k distance oracle needs space $\Omega(n^{1+1/k})$, regardless of the query time.

The idea introduced in their paper, and used repeatedly thereafter, is to uniformly sample a small subset of nodes $L \subseteq V$, which are designated as landmarks, and save all the distances between V and L. Then, for any (u, v), if v is closer to u than u's closest landmark, then we will have already saved their distance, otherwise we will use u's closest landmark for an approximation of the distance.

The lower bound proof of Thorup-Zwick uses a compressibility argument, which can only be used for graphs with many edges. So our work focuses in the regime of sparse graphs, where $m = O(n)$ and the above lower bound argument does not apply.

More recently, Patrascu and Roditty [PR10] managed to get stretch 2 with sub-quadratic space and constant query time, by taking advantage of graph sparsity. They also showed, using a widely believed conjucture about the hardness of the set intersection problem, that even for sparse graphs we can't hope for stretch less than 2 with constant query time.

Finally, Agarwal and Godfrey [AG13] managed to obtain stretch less than 2 by allowing large sublinear query times. In particular, for sparse graphs they achieve a three-way trade-off; more specifically, for any positive integers a and t, they design stretch $1 + \frac{2}{t+1}$ oracles, with space $O\left(\frac{n^2}{a}\right)$, and query time $\widetilde{O}(a^t)$.

2 Preliminaries

2.1 Definitions and notation

In this section we introduce the notation we will use throughout the paper. We are given as an input a graph $G(V, E)$, where V is the set of vertices and E is the set of edges. We will let $n = |V|$, and $m = |E|$. For our purposes, we assume that G is sparse, i.e. $m = O(n)$. We will denote by $d(u, v)$ the length of the shortest path from u to v, while $\delta(u, v)$ will be the approximation of $d(u, v)$ returned by our distance oracle. The stretch of a distance oracle is the constant k, characteristic to the oracle, for which $d(u, v) \leq \delta(u, v) \leq k\, d(u, v)$ In every algorithm, we will need to sample a set L of landmarks. For any vertex u, its closest vertex in L is denoted by l_u (ties broken arbitrarily). We will also need the notion of the ball B_u around any vertex u, that is $B_u = \{v | d(u, v) < d(u, l_u)\}$. The radius r_u of B_u is $d(u, l(u))$, while the vicinity V_u of B_u will be the set of vertices that have a neighbour in B_u and are not contained in it. For v on the vicinity of B_u, we define $d_u v$ to be the length of the shortest path from u to v if only edges from B_u are used. Finally, let $\deg_u$ be the degree of vertex u.

2.2 Review: Thorup-Zwick distance oracles for stretch 3

Theorem 2.1. *For any graph, there exists a data structure of size $O(n^{\frac{3}{2}})$ that can answer stretch 3. distance queries in time $O(1)$. [TZ01]*

Below we briefly describe the construction of the data structure, the querying algorithm, and the theoretical analysis.

Preprocessing:
Sample each vertex of the graph into the set L of landmarks with probability $n^{-\frac{1}{2}}$. We save the distances from every vertex to every landmark. We also save, for each vertex u, the distance to everything within B_u. All the calculated distances are saved in a hash table H, from where they can be retrieved in constant time. We also save for each u its closest landmark l_u.

Query d(u,v):
We first check if the distance is in H. If it is not, then we know that $v \notin B_u$, so $r_u \leq d(u, v)$. So our estimate will be $\delta(u, v) = r_u + d(v, l_u)$. By the triangle inequality,

$$d(u, v) \leq \delta(u, v) = d(u, l_u) + d(v, l_u) \leq d(u, l_u) + (d(u, l_u) + d(u, v)) \leq 3d(u, v)$$

so we get stretch 3.

Runtime Analysis:

$$E[|L|] = n^{\frac{1}{2}}, so\ E[|L \times V|] = n^{\frac{3}{2}}$$

$$E[|B_u|] = O(n^{\frac{1}{2}}), so\ E[\Sigma_u |B_u|] = O(n^{\frac{3}{2}})$$

To see why $E[|B_u|] = O(n^{\frac{1}{2}})$, order the nodes of V by their distances from u, breaking ties arbitrarily. The size of B_u is at most the position of the earliest landmark that appears in the sequence, which is a geometric random variable with mean $n^{\frac{1}{2}}$
So total expected space is $O(n^{\frac{3}{2}}) + O(n^{\frac{3}{2}}) = O(n^{\frac{3}{2}})$, and we can resample until we get space $O(n^{\frac{3}{2}})$.

Algorithm 1 Preprocess$(G(V, E), k)$

```
1: Uniformly sample $L \subseteq V$ with probability $n^{-1/2}$.
2: Store all the distances in $L \times V$.
3: for $v = 1 \to |V|$ do
4:     Store all the distances $d(u, v)$, $v \in B_u$
5: end for
```

Algorithm 2 Query(u, v)

```
1: if $d(u, v)$ has been stored then
2:     return $d(u, v)$
3: end if
4: return $d(u, l_u) + d(l_u, v)$
```

3 Results

In the first section, we improve the stretch less than 2 distance oracles of [AG13]. More specifically, under the same space-time trade-off that they get for stretch $1 + \frac{2}{t+1}$, we get stretch $1 + \frac{1}{t}$.

Next, we present stretch 2 and 3 algorithms first appearing in [AGHP11] and show how to improve the query time. In the case of expander graphs, we get a polynomial improvement.

Finally, in the spirit of [CP10], [PR10], we reduce some set intersection problems to the problem of creating distance oracles of stretch less than $1 + \frac{1}{t}$.

In the following sections, we will use Lemma 3.1 repeatedly. The distance oracles we present work for bounded degree graphs. But using this lemma we can transform any sparse graph to a graph of bounded degree with a linear number of extra dummy nodes, so it is enough to return queries on the new bounded degree graph, which asymptotically has the same size.

Lemma 3.1. *Given a sparse weighted graph $G(V, E)$, we can create a graph $G'(V', E')$ with $O(n)$ nodes, and degree $O(1)$, that includes the nodes of our original graph and preserves the distances between pairs of original nodes.*

Proof. Let $d = \lceil \frac{m}{n} \rceil = O(1)$. If a vertex u has $e > d$ neighbours, then we add a dummy vertex u', connect u and u' with an edge of weight 0 and move $d-1$ neighbours of u to u'. So if u is connected to v through an edge of weight w, moving v from u to u' means deleting edge (u, v) and adding an edge (u', v) of weight w. Clearly, the distance between any two nodes from our original graph is preserved. Now consider the sum $\Sigma_u e_u = O(n)$, where $e_u = 0$ if $\deg_u \leq d$ and $e_u = \deg_u - d$ otherwise. Each vertex has degree at most d iff the sum is 0. Since the sum decreases by at least 1 each time we break a vertex and it initially is $O(n)$, we will get $O(n)$ extra nodes and bounded degree in the end.

□

3.1 Improving Agarwal-Godfrey distance oracles

Theorem 3.2. *Given a sparse weighted graph $G(V, E)$ and integers $a \in \{1, \ldots, n\}$, $t \geq 1$, one can construct a stretch $1 + \frac{1}{t}$ distance oracle with space $O(\frac{n^2}{a})$ and query time $\widetilde{O}(a^t)$.*

This in an improvement over the algorithm of Agarwal-Godfrey, whose distance oracles have stretch $1 + \frac{2}{t+1}$, under the same space-time trade-off. In view of Lemma 3.1, it is enough to design an algorithm that works on bounded degree graphs.

3.1.1 Description of the algorithm

Preprocessing:
We construct L by independently including each vertex of the graph with probability $1/a$. From each vertex of L we run Dijkstra, so we get all distances in $L \times V$. We save the graph with the landmarks marked, and the distances from every node to every landmark.

Query:
Given two vertices u and v, we report an approximate distance by the following procedure. We start by growing the ball B_u around u, using Dijkstra. We throw the nodes of the ball, together with the vicinity of u, into a hash table augmented with their calculated distances from u. For points w on the vicinity of B_u, we use $d_u w$. Let us denote the saved distance by $\delta(u, w)$.
Next, from each point w on the vicinity of B_u, we grow the ball B_w. For each node z reached, including those in V_w, add it to the hash table with distance $\delta(u, w) + d(w, z)$, where $d(w, z)$ is calculated by the Dijkstra algorithm (for points on the vicinity, use $d_u v$ instead of $d(u, v)$). If z is already in the hash table, update its distance to the minimum of the old and the new one.
For $t-2$ more steps, grow balls from the vicinities of all the balls grown in the previous step, then update the distances from u. Repeat for v, so we have two hash tables H_u and H_v.

Set $\Delta = \infty$
For each node w in H_u:

If w is in H_v, update $\Delta = \min(\Delta, \delta(u, w) + \delta(v, w))$
Update $\Delta = \min(\Delta, \delta(u, w) + d(w, l(w)) + d(l(w), v)$
For each w in Hv:
Update $\Delta = \min(\Delta, \delta(v, w) + d(w, l(w)) + d(l(w), u))$
Return Δ

3.1.2 The algorithm

Algorithm 3 Preprocessing

```
1: Uniformly sample $L \subseteq V$ with probability $\frac{1}{a}$
2: Store all distance in $L \times V$
```

Algorithm 4 Query(u, v)

```
 1: Create empty hash table $H_u$
 2: Add $u$ to $H_u$ with $d(u, u) = 0$
 3: for $i = 1$ to $t$ do
 4:     for $w \in H_u$ do
 5:         Do Dijkstra from $w$ until $l_w$ is reached
 6:         For each node z explored by Dijkstra, add it to $H_u$ with distance
            $\delta(u, z) = \min(\delta(u, z), \delta(u, w) + d(z, w)$
 7:         For each node z in $V_z$, add it to $H_u$ with distance
            $\delta(u, z) = \min(\delta(u, z), \delta(u, w) + d_z w)$
 8:     end for
 9: end for
10: for $w \in H_u$ do
11:     if $w \in H_v$ then
12:         $\Delta = \min(\Delta, \delta(u, w) + \delta(v, w))$
13:     end if
14:     $\Delta = \min(\Delta, \delta(u, w) + d(w, l(w)) + d(l(w), v)$
15: end for
16: for $w \in H_v$ do
17:     $\Delta = \min(\Delta, \delta(v, w) + d(w, l(w)) + d(l(w), u))$
18: end for
19: return $\Delta$
```

3.1.3 Space-Time Analysis

All we save is $|L \times V|$ distances, for total expected space $O\left(\frac{n^2}{a}\right)$. We will resample until every ball has size $\widetilde{O}(a)$ and $|L| = O\left(\frac{n}{a}\right)$ (since $B_u > 2a \log n$ with probability $O\left(\frac{1}{n^2}\right)$, we can make sure both bounds hold by resampling). Because the degree is bounded, the vicinity of every ball contains $\widetilde{O}(a)$ points. So the size of our hash tables increases by $\widetilde{O}(a)$ at each step and in the end $|H_u|, |H_v| = \widetilde{O}(a^t)$. Searching each ball using Dijkstra takes $\widetilde{O}(a)$ so creating and comparing the hash tables takes time $\widetilde{O}(a^t)$.

3.1.4 Proof of Correctness

By the triangle inequality, every δ estimate is no less than the actual distance it estimates, so Δ can't be less than $d(u, v)$. Consider now a shortest path between u and v. When we grow B_u, we either set the actual distance $\delta(u, v) = d(u, v)$, or there is a point u_1 that lies on the intersection of the shortest path and the vicinity of B_u, for which $\delta(u_1, u) = d(u_1, u) \geq r_0$, where r_0 is the radius of B_u. Growing the ball around u_1, we again either get the actual distance from u_1 to v, which will give us the actual distance from u to v as u_1 lies on their shortest path, or we get $\delta(u_2, u) = d(u_2, u) \geq r_0 + r_1$, where r_1 is the radius of the ball around u_1. We continue like this for t steps in total, and we either get the exact distance between u and v or a sequence of points $u_0 = u, u_1, .., u_t$ on the shortest path, with their exact distances from u calculated and $d(u_{i+1}, u_i) \geq r_i$.
For v, we respectively get a sequence $v_0 = v, v_1.., v_t$ with $d(v_{i+1}, v_i \geq R_i)$, where R_i are the radii for the v_i. Consider now the paths from u to u_t and from v to v_t along the shortest path form u to v. If these two paths intersect, the algorithm returns the exact distance from u to v. Otherwise, the following inequality holds

$$r_0 + r_1 + .. + r_{t-1} + R_0 + R_1 + .. + R_{t-1} \leq d(u, v)$$

Without loss of generality, let r_i be the least of these radii. Then $r_i \leq \frac{d(u,v)}{2t}$.
Therefore

$$\begin{aligned} d(u, u_i) + d(u_i, l(u_i)) + d(l(u_i), v)) &\leq d(u, u_i) + d(u_i, l(u_i)) + d(u_i, l(u_i)) + d(u_i, v) \\ &= d(u, u_i) + d(u_i, v) + 2r_i \\ &= d(u, v) + 2r_i \leq d(u, v) + 2\frac{d(u, v)}{2t} \\ &= \left(1 + \frac{1}{t}\right) d(u, v) \end{aligned}$$

So $\Delta \leq \left(1 + \frac{1}{t}\right) d(u, v)$

3.1.5 Improvement for small a

Theorem 3.3. *Given a sparse weighted graph $G(V, E)$ and integers $a \in \{1, \ldots, n\}$, $t \geq 1$, one can construct a stretch $1 + \frac{1}{t+\frac{1}{2}}$ distance oracle with space $O\left(\frac{n^2}{a}\right) + \widetilde{O}(na^{t+1})$ and query time $\widetilde{O}(a^t)$. This improves on the previous algorithm for $a < n^{\frac{1}{t+2}}$.*

Proof. (sketch) We modify the algorithm a bit. Instead of creating the hash tables H_u, H_v upon query, we just save them at preprocessing. For each node u, we also save a hash table H'_u that contains the nodes of H_u explored in the first $t-1$ steps of the previous query algorithm. For the query, for each node of H'_u that is contained in H_v, update the Δ estimate as in the original algorithm. If H'_u and H_v don't intersect on the shortest path, we will get a point $z \in H'_u$ *or* H'_v on the shortest path with $r_z \leq \frac{d(u,v)}{2t+1}$. Updating Δ for every $z \in H'_u \cup H'_v$, we get the desired stretch. Since we only traverse hash tables H'_u, H'_v of total size $O(a^t)$, this is our runtime. □

Corollary 3.4. *For $t = 2, a = n^{0.25}$ we get stretch 1.4 for space $n^{1.75}$, time $n^{0.5}$.*

3.2 Stretch 3 oracles for sparse graphs; improved oracles for expanders

Theorem 3.5. *[AGHP11] For bounded degree weighted graphs, we can design stretch-3 distance oracles of space $O(\frac{n^2}{a^2})$ and query time $\widetilde{O}(a)$, where a takes values in the interval $(n^{\frac{1}{4}}, n^{\frac{1}{2}}]$.*

3.2.1 The algorithm

Algorithm 5 Preprocessing

```
1: Uniformly sample $L \subseteq U$ with probability $\frac{1}{a}$
2: Store all distances in $L \times L$
```

Algorithm 6 Query(u, v)

```
1: Run Dijkstra from $u$ until $l_u$ is reached, then throw the explored nodes augmented
   with their distance from $u$ into hash table $H_u$. Add all $w \in V_u$ to $H_u$ with distance
   $d_u w$.
2: Run Dijkstra from $v$ until $l_v$ is reached, then throw the explored nodes augmented
   with their distance from $v$ into hash table $H_v$. Add all $w \in V_v$ to $H_v$ with distance
   $d_v w$.
3: $\Delta = \infty$
4: for $w \in H_u$ do
5:     if $w \in H_v$ then
6:         $\Delta = \min(\Delta, d(u,w) + d(w,v))$
7:     end if
8: end for
9: $\Delta = \min(\Delta, d(u,l_u) + d(l_u,l_v) + d(l_v,v))$
10: return $\Delta$
```

3.2.2 Space-Time Analysis

We have $E(|L|) = O\left(\frac{n}{a}\right)$. We will resample until all balls have size $\widetilde{O}(a)$ and $|L| = O\left(\frac{n}{a}\right)$, so total space will be $|L \times L| = O\left(\frac{n^2}{a^2}\right)$ (can be done since $|B_u| > 2a\log n$ with probability $O\left(\frac{1}{n^2}\right)$ and $\log n \leq 2\log a$). Exploring B_u, B_v by Dijkstra takes $\widetilde{O}(a)$, while comparing the two balls takes time linear in their size, so total query time $\widetilde{O}(a)$.

3.2.3 Proof of correctness

By the triangle inequality, any approximation is at least the actual distance $d(u,v)$. If the balls intersect, then let s_{uv} be the shortest path connecting u, v. Let p_u be the node farthest from u in $s_{uv} \cap B_u$, p_v the closest to v in $s_{uv} \cap B_v$. If $p'_u \in V_u$ is next to p_u on s_{uv}, then $d_u p'_u = d(u, p'_u)$, since the shortest path from u to p'_u goes through p_u. Define p'_v similarly and so we have the exact distances from u to p'_u and from v to p'_v in our hash tables. If $s_{up'_u}$, $s_{vp'_v}$ intersect, then we will get the exact distance $d(u,v)$. Otherwise $d(u,l_u) + d(v,l_v) \leq d(u,v)$ so the final approximation will give

$$\begin{aligned} d(u,l_u) + d(l_u,l_v) + d(l_v,v) &\leq (d(u,l_u) + d(v,l_v) + d(u,v)) + (d(u,l_u) + d(v,l_v)) \\ &\leq 3d(u,v) \end{aligned}$$

Theorem 3.6. *Given an unweighted graph $G(V,E)$ of degree at most k and edge expansion c (for all $V' \subseteq V$ such that $|V'| \leq |V|/10$, $|\partial V'| \geq c|V'|$[2]), we can construct stretch 3 distance oracles with space $O(n^{2-2\epsilon})$ and query time $O(n^{\epsilon - \log_k c(1-3\epsilon)})$.*

Corollary 3.7. *Given a degree 5 expander with edge expansion 4, we can construct stretch 3 distance oracles with space $O(n^{1.4})$ and query time $O(n^{0.2139})$.*

3.2.4 The Algorithm

For this algorithm, let C_u be the set of all nodes within distance $\log_k n^{1-3\epsilon}$ from u.

Algorithm 7 Preprocessing

```
1: Uniformly sample $L \subseteq V$ with probability $\frac{1}{n^\epsilon}$
2: Store all distances in $L \times L$
3: Store all distances in $u \times C_u$
4: for $u \in V$ do
5:     for $v \in V$ do
6:         if $B_u \cap C_v \neq \emptyset$ then
7:             Store $d(u,v)$
8:         end if
9:     end for
10: end for
```

[2]Given $V' \subseteq V$, $\partial V' = \{(u,v) \in E : u \in V', v \notin V'\}$.

Algorithm 8 Query

```
1: Grow ball $B'_u$ or radius $r_u - \log_k n^\epsilon$ around $u$ using Dijkstra
2: if $v$ is found by Dijkstra then
3:     return $d(u, v)$ as found by Dijkstra
4: end if
5: for $z \in B'_u$ do
6:     if $B_v \cap C_z \neq \emptyset$ then
7:         return $d(u, z) + d(z, v)$
8:     end if
9: end for
10: return $d(u, l_u) + d(l_u, l_v) + d(l_v, v)$
```

3.2.5 Proof of correctness

Clearly, $\cup_{z \in B'_u} C_z$ covers B_u. So if $B_u \cap B_v \neq \emptyset$ and $v \notin B'_u$, then for any $z \in B'_u$, $d(u, z) < d(u, v)$, so line 7 of the query algorithm will give stretch-3 for any z for which $C_z \cap B_v \neq \emptyset$. If line 7 does not return an answer, then $B_u \cap B_v = \emptyset$ and line 10 will give stretch-3.

3.2.6 Space-Time Analysis

Resample until $|L \times L| = O\left(\frac{n^2}{a^2}\right)$ and all balls have size $\widetilde{O}(n^\epsilon)$.
Consider the clusters of radius $\log_k n^{1-3\epsilon}$ around every vertex. Then consider any ball B_u, of size $\widetilde{O}(n^\epsilon)$. The number of points at distance at most $\log_k n^{1-3\epsilon}$ from the ball are $\widetilde{O}(n^{\epsilon+1-3\epsilon}) = \widetilde{O}(n^{1-2\epsilon})$, so each ball intersects $\widetilde{O}(n^{1-2\epsilon})$ clusters. So saving all the ball-cluster intersections takes space $\widetilde{O}(n^{2-2\epsilon})$.
The query time is proportional to the size of B'_u. Increasing the radius of a ball by 1 incrases its volume by at least c, so since we have to increase $\log_k n^{1-3\epsilon}$ times to get B_u which has volume $\widetilde{O}(n^\epsilon)$, the volume of B'_u and thus the runtime is $\widetilde{O}\left(\frac{n^\epsilon}{c^{\log_k n^{1-3\epsilon}}}\right) = \widetilde{O}\left(\frac{n^\epsilon}{n^{(1-3\epsilon)\log_k c}}\right)$.

Theorem 3.8. *We can improve the query time in Theorem 3.6 by a factor of* $\log n$ *for* $a = n^\epsilon$ $\forall \epsilon \in (\frac{1}{4}, \frac{1}{3})$. *We can do the same in Theorem 3.9 for* $a = n^\epsilon$ $\forall \epsilon \in (\frac{1}{3}, \frac{1}{2})$.

Proof. (sketch)
We use the same clustering as above, but we make the radius of the clusters $\log_k n^{1-3\epsilon}$ and instead of saving the whole graph, for each u we store $O(\frac{|B_u|}{\log n})$ clusters that cover $B_u \cup V_u$ □

The above theorem gives an improvement for all sparse graphs, using a clustering that yields a polynomial improvement for the more specific case of expanders. Although the improvement we show for general sparse graphs is only by a logarithmic factor, we expect this to fare pretty well in practice. Notice that whenever the ball of one of the queried vertices has good expansion properties, the query time actually improves by a polynomial factor. Furthermore, it looks like the more general property that is required by the ball is only to have low diameter. In light of previous results concerning graph decompositions (see [CKR04]), we believe that there is much more structure to exploit in order to achieve provably better distance oracles. However, for now, we look forward to implementing this improvement and comparing it against the previously designed oracles.

3.3 Stretch 2 oracles for sparse graphs; improved oracles for expanders

Theorem 3.9. *[AGHP11] For bounded degree weighted graphs, we can design stretch-2 distance oracles of space $O(\frac{n^2}{a})$ and query time $O(a \log n)$, where a takes values in the interval $[1, n]$ (we use a similar approach to [AGHP11], who did this for $a \in [1, n^{\frac{1}{2}}]$).*

3.3.1 The algorithm

Algorithm 9 Preprocessing

```
1: Uniformly sample $L \subseteq V$ with probability $\frac{1}{a}$
2: Store all distances in $V \times L$
3: Store the graph
```

Algorithm 10 Query

```
1: Do Dijkstra from $u$ until $l_u$ is reached, then throw the explored nodes augmented with
   their distance from $u$ into hash table $H_u$. Add all $w \in V_u$ to $H_u$ with distance $d_u w$
2: Do Dijkstra from $v$ until $l_v$ is reached, then throw the explored nodes augmented with
   their distance from $v$ into hash table $H_v$. Add all $w \in V_v$ to $H_v$ with distance $d_v w$
3: $\Delta = \infty$
4: for $w \in U$ do
5:     if $w \in V$ then
6:         $\Delta = \min(\Delta, d(u, w) + d(w, v))$
7:     end if
8: end for
9: $\Delta = \min(\Delta, d(u, l_u) + d(l_u, v), d(u, l_v) + d(l_v, v))$
10: return $\Delta$
```

3.3.2 Space-Time Analysis

As before, we can resample until $|V \times L| = O\left(\frac{n^2}{a^2}\right)$ and $|B_u| = O(a \log n)\ \forall u$.

Exploring B_u, B_v by Dijkstra takes $O(a \log n)$, while comparing the two balls takes time linear in their size, so total query time $O(a \log n)$

3.3.3 Proof of correctness

By the triangle inequality, any approximation is at least the actual distance $d(u, v)$. If we don't get the exact distance on the shortest path, then as in the stretch 3 oracle proof, described in section 3.2.3, we get $d(u, l_u) + d(v, l_v) \leq d(u, v)$. In that case, without loss of generality let $d(u, l_u) \leq d(v, l_v) \Rightarrow 2d(u, l_u) \leq d(u, v)$ and the final approximation will give

$$d(u, l_u) + d(l_u, v) \leq d(u, l_u) + d(u, l_u) + d(u, v) \leq 2d(u, v)$$

Theorem 3.10. *Given an unweighted graph $G(V, E)$ with maximum degree k and edge expansion c (for all $V' \subseteq V$ such that $|V'| \leq |V|/10$, $|\partial V'| \geq c|V'|$), we can construct stretch 2 distance oracles with space $O(n^{2-\epsilon})$, query time $\widetilde{O}(n^{\epsilon - \log_k c(1-2\epsilon)})$, for any $\epsilon \in (\frac{1}{3}, \frac{1}{2})$.*

Proof. The algorithm uses exactly the same clustering as for the *stretch* 3 oracle, this time with clusters of radius $\log_k n^{1-2\epsilon}$. The algorithm and proof are almost identical, so we won't repeat them here. □

3.4 Lower Bounds via Set Intersection Hardness

Below we describe a folklore problem on static data structures, for which we make a conjecture concerning its space-query time tradeoff. While [CP10] already have a proof in the same flavor, we use this conjecture to show conditional lower bounds for distance oracles of stretch less than 2. We also introduce an additional version of Set Intersection (called Set Intersection by distance 2), which we show that is harder than Set Intersection and we reduce it to distance oracles of distance less than 1.5.

3.4.1 The Set Intersection Problem

We are given a ground set U and a family of k sets $S_1, .., S_k \subseteq U$, such that $|S_i| = n/k$ for all i. We wish to preprocess the input to create a data structure occupying space $\mathcal{S}$ that can answer queries of the type "is $S_i \cap S_j$ empty?" in time $\mathcal{T}$.

Below we give a conjecture concerning the space vs. query time tradeoff between $\mathcal{S}$ and $\mathcal{T}$. We believe that in some regimes, one can not do better than the naive solution. That is, either store all intersecting pairs of sets (or the complement), or save the sets and compare them on the fly when queried.

We believe that constructions such as the one below are the hardest instances for Set Intersection. Consider, for $r \in [\frac{1}{2}, 1)$ and small $\epsilon > 0$, an instance of Set Intersection with $|U| = n^{2-2r-2\epsilon}, k = n^r$. Its sets are constructed by uniformly sampling elements with probability $\frac{1}{n^{1-r-\epsilon}}$. In expectation, $|S_i| = n^{1-r-\epsilon}$. Furthermore, every two sets intersect with constant probability. So the expected number of set intersections is $\Theta(n^{2r+2\epsilon})$ (the size of the complement is the same). So the natural solution will require either space $\Omega(n^{2r+2\epsilon})$, or query time $\Omega(n^{1-r-\epsilon})$.

One should note that at the extremes of the tradeoff curve lie the naive query algorithm, which requires space $\mathcal{S} = \Theta(n)$ and query time $\mathcal{T} = O(n^{1-r})$, respectively the data structure that stores all the pairwise intersections, which requires space $\mathcal{S} = \Theta(n^{2r})$ and query time $\mathcal{T} = O(1)$.

Conjecture 3.11. *Any data structure for the Set Intersection Problem in the regime $k = \Theta(n^{r+\epsilon})$ using space $\mathcal{S} = O(n^{2r})$ requires query time $\mathcal{T} = \Omega(n^{1-r-\epsilon})$, $\forall r \in [\frac{1}{2}, 1)$, for all small $\epsilon > 0$.*

Next, we present a reduction of Set Intersection to distance oracles of stretch less than 2. We note that a similar result was given by [CP10]. However, we provide the specific bounds for distance oracles that are implied by our conjecture.

3.4.2 Reduction to distance oracles of stretch less than 2

Consider an instance of Set Intersection with k sets over a universe of size n. Create a graph with nodes $u_1, u_2, \ldots, u_n$ representing the elements of U, nodes $s_1, \ldots, s_k$ representing the sets $S_1, \ldots, S_k$. Connect s_i and u_m with an edge of length 1 iff $m \in S_i$.

Now, it is easy to see that sets S_i and S_j have a common element iff the distance of nodes s_i and s_j in the graph is 2, otherwise their distance will be at least 4. Since any distance oracle of stretch less than 2 can distinguish between distances 2 and 4, such a distance oracle can solve the set intersection problem.

Theorem 3.12. *Assuming Conjecture 3.11, any distance oracle of stretch less than 2 with space $O(n^{2r})$ requires query time at least $\Omega(n^{1-r-\epsilon})$, for $r \in [1/2, 1)$ and any constant $\epsilon > 0$.*

Proof. (sketch) Suppose we have a distance oracle of stretch less than 2 that uses space $O(n^{2r})$ and query time $o(n^{1-r-\epsilon})$ on sparse graphs of n nodes. Given the instance of the set intersection problem described before Conjecture 3.11, we can reduce it to the problem of returning distances of stretch less than 2 on a graph of $\max(n^{2-2r}, n^r)$ vertices and n edges. Adding dummy nodes, we get a sparse graph of n nodes. So using the distance oracle, we can get query time $o(n^{1-r-\epsilon})$ with space $O(n^{2r})$, which contradicts Conjecture 3.11. □

We compare the previous result against the upper bound given in Theorem 3.3. More specifically, the upper bound gives stretch 5/3 distance oracles, with space $O(n^{5/3})$ and query time $\widetilde{O}(n^{1/3})$. The following corollary shows that assuming the Set Intersection conjecture, this is not very far off from optimality.

Corollary 3.13. *Assuming Conjecture 3.11, any distance oracle with stretch less than 2, using space $O(n^{5/3})$ requires query time $\Omega(n^{1/6-\epsilon})$, for any constant $\epsilon > 0$.*

For the rest of the section, we introduce a new problem related in spirit with Set Intersection. We prove that it is harder than standard Set Intersection, and show that it also implies hardness for distance oracles with stretch less than 1.5. Our statements do not include specific quantitative bounds for the time being, since we are still trying to understand how hard this new problem actually is.

3.4.3 Set Intersection by distance 2

We are given a ground set U and sets $S_1, .., S_k$ over U. We wish to create a data structure that can answer the following query: Given S_i, S_j, does there exist an S_k with $S_i \cap S_k \neq \emptyset$ and $S_j \cap S_k \neq \emptyset$? In other words, check whether S_i and S_j intersect or have a common neighbour.

3.4.4 Set Intersection by distance 2 is harder than Set Intersection

Consider an instance of the set intersection problem: Ground set U with $|U| = n$, sets $S_1, .., S_m$ of total size P.

We can reduce it to an instance of the set intersection by distance 2 in the following way: For each set S_i, add a dummy element u_{n+i} to U and S_i and create a new set S'_i containing u_{n+i} alone. So we double the number of sets but leave the size of the universe unchanged.

Notice that $S_i \cap S_j \neq \emptyset$ iff S'_i and S_j have distance at most 2, so creating a data structure that answers distance 2 queries for the new sets also solves the previous problem.

Now suppose that we have a data structure that solves the set intersection by distance 2 problem using space $s(n, m, P)$, time $t(n, m, P)$. We showed that we can solve the set intersection problem using space $O(s(2n, 2m, 2P))$ and query time $O(t(2n, 2m, 2P))$. What is left is to show that $s(2n, 2m, 2P) = O(s(n, m, P))$ and $t(2n, 2m, 2P) = O(t(n, m, P))$. To show this, partition U into 4 sets of size $\frac{n}{2}$. Consider the $\binom{4}{2} = 6$ universes we can get as the union of any two of these sets. Let them be $U_k, k \in 1, .., 6$. Now S_i, S_j have distance at most 2 over U iff their restrictions to U_k have distance at most 2 over U_k for some k. We can check this using space $6O(s(n, 2m, 2P))$, time $6O(t(n, 2m, 2P))$, so $s(2n, 2m, 2P) = O(s(n, 2m, 2P))$, $t(2n, 2m, 2P) = O(t(n, 2m, 2P))$. Doing the same for m, P, we get $s(2n, 2m, 2P) = O(s(n, m, P))$ and $t(2n, 2m, 2P) = O(t(n, m, P))$.

3.4.5 Reduction to distance oracles of stretch less than 1.5

Suppose U is $\{1, 2, .., n\}$ and we are given sets $S_1, .., S_k$. Create a graph with nodes $u_1, u_2, .., u_n$ representing the elements of U, nodes $s_1, .., s_k$ representing the sets $S_1, .., S_k$.

Connect s_i, u_m with an edge of length 1 iff $m \in S_i$.

Now, it is easy to see that sets S_i, S_j have some common neighbour iff the distance of nodes s_i, s_j in the graph is at most 4, otherwise their distance will be at least 6. Since any distance oracle of stretch less than 1.5 can distinguish between distances 4 and 6, such a distance oracle can solve the set intersection problem by distance 2. So it can't have a better space-time trade-off than the optimal data structure that solves the set intersection problem.

In a similar fashion with Set Intersection by distance 2, we can consider a more general version, which uses distance t between sets. Just as above, it turns out that this can be used to prove conditional lower bounds for the space vs time tradeoff for distance oracles with stretch less than $1 + 1/t$, for any positive integer t.

3.4.6 Generalized Set Intersection Reduction

We can define set intersection by distance t similarly. The query asks whether two sets S_i, S_j are connected by a sequence of at most $t-1$ neighbors. This problem can be reduced to distance oracles of stretch less than $1 + \frac{1}{t}$. It can similarly be shown that the problem of set intersection by distance $t + 1$ is harder than set intersection by distance t.

4 Discussion

In this paper, we presented some new approaches to distance oracles for sparse graphs. We expect to extend these approaches and combine them with some other results we have in mind. More specifically, it looks like we can take much advantage of various graph decompositions in the spirit of [CKR04]. We already used a simple clustering scheme for expander graphs, where our approach improves the query time by a polynomial factor. We believe that this is a good heuristic that significantly improves a few algorithms in practice, so we aim to implement our approach and see how well it fares in practice. We support our claim by mentioning that our clustering performs well on expanders, and on graphs with very poor expansion. We hope that we can bridge the gap between the two by employing a more adaptive clustering scheme. For the sake of experimenting, graphs of the internet topology and road networks should provide useful evidence.

5 Acknowledgements

I would especially like to thank my mentor, Adrian Vladu, who patiently guided my research over the last weeks, for his enthusiasm and support. I would also like to thank professors Pavel Etingof and David Jerison for giving me the opportunity to participate in SPUR, and for taking the time to discuss my ideas.

References

[AG13] Rachit Agarwal and Philip Brighten Godfrey. Distance oracles for stretch less than 2. In *SODA*, volume 52, page 526, 2013. 1, 2, 4

[AGHP11] Rachit Agarwal, Philip Brighten Godfrey, and Sariel Har-Peled. Approximate distance queries and compact routing in sparse graphs. In *INFOCOM*, page 1754, 2011. 4, 8, 11

[CKR04] Gruia Călinescu, Howard J. Karloff, and Yuval Rabani. Approximation algorithms for the 0-extension problem. *SIAM J. Comput.*, 34(2):358–372, 2004. 11, 15

[CP10] Hagai Cohen and Ely Porat. On the hardness of distance oracle for sparse graph. In *CoRR*, 2010. 4, 12, 13

[PR10] Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. In *FOCS*, volume 52, page 815, 2010. 2, 4

[TZ01] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *STOC*, volume 52, page 183, 2001. 1, 2, 3

A Hash Tables

Hash tables are dynamic data structures for storing elements augmented with information. They support searches, insertions and deletions, all of which can be executed in constant amortized time. The space they require is asymptotically optima, since a hash table holding k elements at a specific time occupies $O(k)$ words of memory.

B Dijkstra's Algorithm

Dijkstra's algorithm is the textbook algorithm for calculating single source shortest paths in graphs with nonnegative edge weights. Starting at u, it explores the closest node to u not yet explored, adds it to the explored set and repeats. We can stop the algorithm once it reaches some node v, and the explored set will contain all nodes closer than this node. For bounded degree graphs, exploring a nodes costs $O(a \log a) = \widetilde{O}(a)$.

C Expander Graphs

We use expander graphs to benchmark our algorithms against. A family of (k, c) expanders is a family of sparse graphs of degree bounded above by some constant k, such that for all $V' \subseteq V, |V'| \leq |V|/10$, we have $|\partial V'| \geq c\,|V'|$. The parameter c denotes the *edge expansion* of a graph in the family.