Distance Oracles for Stretch Less Than 2

Rachit Agarwal
Department of Computer Science,
University of Illinois at Urbana-Champaign
agarwa16@illinois.edu

P. Brighten Godfrey
Department of Computer Science,
University of Illinois at Urbana-Champaign
pbg@illinois.edu

Abstract

We present distance oracles for weighted undirected graphs that return distances of stretch less than 2. For the realistic case of sparse graphs, our distance oracles exhibit a smooth three-way trade-off between space, stretch and query time — a phenomenon that does not occur in dense graphs. In particular, for any positive integer t and for any $1 \leq \alpha \leq n$, our distance oracle is of size $O(m + n^2/\alpha)$ and returns distances of stretch at most $(1 + \frac{2}{t+1})$ in time $O((\alpha\mu)^t)$, where $\mu = 2m/n$ is the average degree of the graph. The query time can be further reduced to $O((\alpha + \mu)^t)$ at the expense of a small additive stretch.

1 Introduction

Distance oracles are compact data structures that can be efficiently queried to compute the distance between any given source-destination pair in a graph. A distance oracle is said to return stretch-s distances if, for a given pair of vertices at distance d, the returned distance δ satisfies $d \leq \delta \leq s \cdot d$. There is a trade-off between the size of the oracle and its stretch; this trade-off is now well understood for general undirected graphs. In particular, there exist distance oracles of size $\Theta(n^{1.5})$ that return distances of stretch 3 [17], and a lower bound of Thorup and Zwick [17] shows that oracles that return distances of stretch less than 3 must have size $\Omega(n^2)$. However, the hard instances used to prove this lower bound are extremely *dense graphs*: the proof shows that to achieve stretch less than 3, the size of the oracle must be lower bounded by the number of edges in a graph with $\Theta(n^2)$ edges. Essentially, the bound shows the existence of a dense enough graph that is incompressible.

Can lower stretch be achieved using sub-quadratic space in sparse[1] graphs? This question is both interesting and important for two reasons. First, far from being a narrow special case of the problem, sparse graphs are the most relevant case. Nearly all large real-world networks are sparse, including road networks [14], social networks [4], the router-level Internet graph [9] and the Autonomous System-level Internet graph [8], as well as networks like expander graphs that are important in many settings; see [2] for numerical examples of the sparsity of a number of networks. These include nearly all networks with real-world applications of distance oracles — in social networks [1, 13], personalized search [13, 18], network routing [2, 16], etc. In this sense, oracles that match the lower bound of [17] are optimal only for the obscure case of extremely dense graphs.

The second reason sparse graphs are interesting is that the mathematical structure of the question changes dramatically in the case of sparse graphs. Taking the extreme case of $m = \tilde{O}(n)$ edges, one can trivially produce an oracle of size $\tilde{O}(n)$ that returns exact shortest paths (stretch 1), simply by storing the input graph and running Dijkstra's algorithm on each query. This, however, takes $\tilde{O}(n)$ time per query. Thus, in the dense case, the focus is on retaining sufficient information to retrieve low stretch distances after a "lossy compression" of the graph. In the sparse case, the input graph can be stored in relatively little space, and the trade-off with *query time* becomes critical.

Relatively little is known about the space-stretch-time trade-off for sparse graphs. Pătraşcu and Roddity [11] designed a stretch-2, constant-time oracle of size $O(n^{4/3}m^{1/3})$. Agarwal *et al.* [2, 3] explored the trade-off between size and query time; their lowest-stretch oracle is of size $O(m + nm^{1-\varepsilon})$ and returns stretch-2 distances in $O(m^{\varepsilon})$ time. No non-trivial distance oracle that returns distances of stretch less than 2 in sparse *weighted* graphs is known.

In this paper, we present the first distance oracle that returns distances of stretch less than 2 for weighted graphs with $m = o(n^2)$ edges. For the realistic case of sparse graphs, our distance oracle achieves a smooth three-way trade-off between space, stretch and query time. Our main theorem is as follows.

Theorem 1.1. *Let G be any weighted graph with n vertices and $m = O(n\mu)$ edges with non-negative weights. Then, for any $1 \leq \alpha \leq n$ and any integer $t > 0$, we can construct a distance oracle of expected size $O(m+n^2/\alpha)$ and a query algorithm that returns distances of stretch $(1+\frac{2}{t+1})$ in expected time $O((\alpha\mu)^t + (\alpha\mu)^{t-1}\alpha \log \alpha)$.*

[1] We say that a graph is sparse if it has $m = o(n^2)$ edges.

For instance, consider graphs with $m = \tilde{O}(n)$ edges and fix the space to be $\tilde{O}(n^{7/4})$. Then our oracle retrieves stretch-2 distances in $O(n^{1/4})$ time and stretch-1.67 distances in $O(\sqrt{n})$ time.

More generally, the parameters α and t in our result give a smooth tradeoff between query time and stretch (for fixed t) or between query time and size (for fixed α). The theorem also implies that on sufficiently sparse graphs, it is possible to retrieve distances with stretch arbitrarily close to 1 in sub-quadratic space and sub-linear query time. The query time in Theorem 1.1 can be further reduced using a small additive stretch, that depends only on t. We describe this and other special-case results in §6.

Before delving deeper, we remark on what we believe differentiates our approach from previous work. The main focus of [2, 3, 11, 17] was on designing (elegant and compact) distance oracles to represent the input graph; their query algorithms for retrieving distances were rather straightforward. In contrast, the focus of our work is a query algorithm that allows us to achieve a more general space-stretch-time trade-off and retrieve distances of stretch less than 2; our distance oracle, on the other hand, is closely related to that of [2, 17] with a simple (yet powerful) modification to facilitate our query algorithm. In particular, our main tool is a query algorithm that recursively queries a stretch-3 distance oracle of size $O(m + n^2/\alpha)$; with each successive query, the stretch improves and the query time increases. The main challenge is to improve stretch without significantly increasing query time, which we accomplish by performing recursive queries in a structured fashion.

2 Related work

Lower bounds for distance oracles. For general weighted undirected graphs, Thorup and Zwick [17] proved that distance oracles that return distances of stretch 2 and 3 must require $\Omega(n^2)$ and $\Omega(n^{3/2})$ space. Their lower bounds, as discussed earlier, hold only for dense graphs and do not apply to our case.

Sommer *et al.* [15] proved that the size of stretch-s time-t distance oracles is lower bounded by $n^{1+\Omega(1/st)}$; that is, for (constant stretch and) constant query time, any oracle must have super-linear size for graphs with $m = \tilde{O}(n)$ edges. For graphs with $m > \tilde{\Omega}(n)$ and/or super-constant query time, their bound does not have any meaningful interpretation. Conditioned on a conjecture on hardness of set intersection queries, Pătraşcu and Roditty [11] strengthened their result for the special case of stretch-2 oracles by proving a lower bound of $\Omega(n\sqrt{m})$ on the size of oracles with constant query time.

There are reasons to believe that it may be hard to improve upon these lower bounds unconditionally [11], and realistically, upper bounds seem to be the only way to make progress on the problem. A particularly compelling scenario is the case of $\Omega(\log n)$ query time, like ours, for which no non-trivial lower bounds are known and it is conceivable that distance oracles with smaller stretch *and* size exist.

Upper bounds for weighted graphs. For general weighted undirected graphs, Thorup and Zwick [17] designed a distance oracle, that for any integer $k \geq 2$, is of size $O(kn^{1+1/k})$ and returns stretch-$(2k-1)$ distances in $O(k)$ time; the construction time of their oracle is $O(mn^{1/k})$. Subsequent research improved the construction time [5–7, 19] and the query time [10]. Designing oracles with reduced size and/or stretch turned out to be a much harder problem, precisely due to the above lower bounds. Indeed, these results may be quite far from optimal for the realistic case of sparse graphs.

For the case of sparse graphs, Pătraşcu and Roditty [11] considered the problem of designing distance oracles with constant query time: their oracle is of size $O(n^{4/3}m^{1/3})$ and returns stretch-2 distances. Agarwal *et al.* [2, 3] designed distance oracles with super-constant query time and explored the trade-off between size and query time. For instance, on graphs with $m = \tilde{O}(n)$ edges, a special case of the distance oracle in [2] returns stretch-2 distances using $\tilde{O}(n^{2-\varepsilon})$ space and $O(n^{\varepsilon})$ query time — for $\varepsilon = 0.5$, this requires less space but higher query time than [11].

Recently, Porat and Roditty [12] gave a stretch-less-than-2 oracle for the special case of *unweighted* graphs. We give a detailed comparison below but note that for sparse *weighted* graphs, no non-trivial oracle for stretch less than 2 is known; even for unweighted graphs, we significantly improve upon their results *for each point in the space-stretch-time trade-off space*.

Comparison with Porat-Roditty oracle [12]. Perhaps, the work most closely related to ours is that of Porat and Roditty: their oracle is of size $O(nm^{1-\varepsilon})$ and returns distances of stretch $\frac{1+2\varepsilon}{1-2\varepsilon}$ in time $O(m^{1-\varepsilon})$ for the special case of unweighted graphs.

There are three main aspects in which our oracle of Theorem 1.1 improves upon their oracle. First, we significantly improve upon their results *for each point in the space-stretch-time trade-off space*. For instance, consider graphs with $m = \tilde{O}(n)$ edges and let $\alpha = n^{\varepsilon}$. Then, by setting $\varepsilon = (1/(2t+4))$, our oracle has the same stretch and space as their oracle but requires $\sqrt{n} \cdot n^{t/(2t+4)}$ less query time — a polynomial reduction in the query time for each value of t. The improvement increases with the density of the graph.

Second, unlike their distance oracle, our oracle works for general *weighted* graphs. Finally, their distance oracle exhibits the space-stretch trade-off as in classical distance oracles for dense graphs [17]; once the stretch is fixed, the space and query time are fixed. Our oracle exhibits a more general three-way trade-off highlighting a fundamental difference between the dense and the sparse cases.

3 Overview of our technique

We start by giving a high level overview of our technique. To do so, let us briefly recall one of the most frequently used techniques to design distance oracles [2, 3, 11, 17]. Typically, the construction of distance oracles starts by selecting a subset of vertices L known as "landmark vertices". The oracle stores the distance from each vertex in L to each other vertex in the graph. Next, each vertex u is assigned a landmark vertex $\ell(u) \in L$; this is the vertex $\ell \in L$ that minimized the distance $d(u,\ell)$, ties broken arbitrarily. Finally, the notion of "balls" is used — the ball of any vertex u is the set of all vertices w for which the distance between u and w is strictly less than the distance between u and $\ell(u)$. The distance from each vertex u to each vertex in $B(u)$ is either stored in the oracle [2,3,11,17] or is computed on the fly [2,3].

In order to retrieve low stretch distances, a typical query algorithm works as follows. When queried for the distance between two vertices u, v, the exact distance is returned if $u \in B(v)$ or if $v \in B(u)$; if not, the distance $d(u,\ell(u)) + d(\ell(u), v)$ is returned. In the latter case, by triangle inequality, we get that the returned distance is at most $2 \cdot d(u,\ell(u)) + d(u,v)$, which is at most $2 \cdot d(u,\ell(u))$ more than the exact distance. Hence, the stretch is given by $1 + 2d(u,\ell(u))/d(u,v)$. For instance, if $d(u,\ell(u)) \approx d(u,v)$, we get roughly stretch 3 [17].

Our technique builds upon the above technique using two observations. First, given the above stretch-3 oracle, it may be possible to retrieve distances of lower stretch by carefully querying the oracle. More specifically, let u' be some vertex along the shortest path between u and v and suppose we know the exact distance $d(u,u')$. We can then query the oracle for distance between u' and v and return the distance $d(u,u') + \delta(u',v)$. As above, the exact distance is returned if $u' \in B(v)$ or $v \in B(u')$; if not, the returned distance is $d(u,u') + d(u',\ell(u')) + d(\ell(u'),v)$. Using triangle inequality, we get that the returned distance is at most $d(u,u') + 2 \cdot d(u',\ell(u')) + d(u',v) = 2 \cdot d(u',\ell(u')) + d(u,v)$. If $d(u',\ell(u')) < d(u,\ell(u))$, this in fact leads to a better stretch; finding such a vertex u' and computing the distance $d(u,u')$ takes some time, but leads to improved stretch.

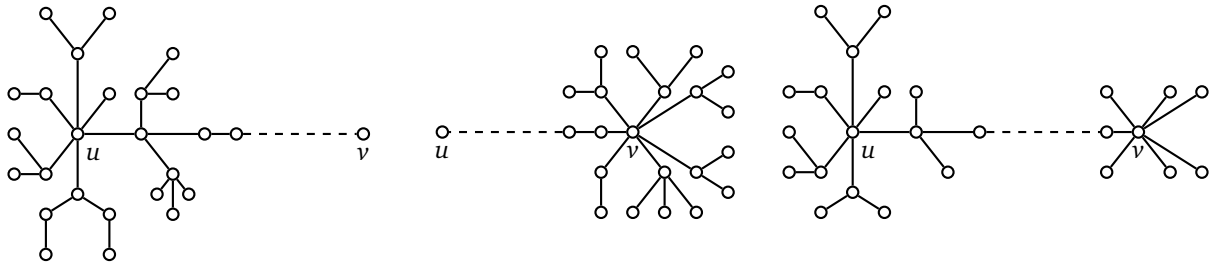
Our second observation is related to finding a good candidate vertex u' for querying the oracle. Recall that the stretch of the distance returned depends on $d(u',\ell(u'))$, or more precisely, on the ratio $d(u',\ell(u'))/d(u,v)$. The lower this ratio, the lower is the stretch of the distance returned. Hence, we not only want to find a vertex u' along the shortest path, but also a vertex with a small "ball radius" $d(u',\ell(u'))$. If we can find a vertex u' such that $d(u',\ell(u')) \leq d(u,v)/(t+1)$, we will get the desired bound on stretch. We will show that such a vertex always exists and can be found within the desired bound on the query time.

The main challenge in exploiting the above two observations is that of finding a good candidate vertex u' — one that lies along the shortest path between u and v and allows us to bound $d(u',\ell(u'))/d(u,v)$. Indeed, this information is not stored within the oracle and that is where we need to do most of the work. The rest of this section provides some low level details on *efficiently* finding such a vertex u'.

We start by noting that there may be multiple candidate vertices for u'; for instance, since $u \in B(u)$, there exists at least one such candidate vertex among the neighbors of vertices in $B(u)$[2]. To find this vertex, we grow a partial shortest path tree around the source u until all the neighbors of $B(u)$ have been explored (see Figure 1(a)); alternative algorithms for finding these candidate vertices may grow shortest path trees around v (as in Figure 1(b)) or even around both u and v (as in Figure 1(c)). Growing these shortest path trees contribute to the query time of our algorithm.

Here, we need to resolve the issue of the source and/or the destination having extremely dense neighborhoods — if $O(m)$ edges need to be explored to grow this (partial) shortest path tree, this may lead to $O(m)$ worst-case query time. In order to resolve this problem, we use a result from [2, 3], which shows that designing distance oracles for graphs with average degree μ is no harder than that for μ-degree bounded graphs. In particular, they present a technique that takes an oracle for μ-degree bounded graphs and converts it into an oracle for graphs with average degree μ with no loss in stretch and at most a constant factor increase in size and query time. This allows us to not worry about high-degree vertices by focusing on designing distance oracles for μ-degree bounded graphs.

[2]Note that at least one such vertex also exists among the neighbors of vertices in $B(v)$. Indeed, this vertex may be a better candidate for being the vertex u'. It turns out that our desired bound on stretch can be proved irrespective of which of these vertices is chosen as the vertex u' but it may be possible to achieve an improved bound on stretch (theoretically or empirically) by using a more sophisticated algorithm for selecting this vertex u'.



(a) Exploring the neighborhood of u in search of candidate vertex u'

(b) Exploring the neighborhood of v in search of candidate vertex u'

(c) Exploring the neighborhood of u and v in search of candidate vertex u'

Figure 1. Various possibilities of exploring the neighborhoods of the source and the destination in search of candidate vertex u'.

We now give the high-level idea of proving the desired bound of $(1+2/(t+1))$ on stretch. To achieve this bound, we use a recursive query algorithm. Once we have found a good candidate vertex u' among the neighbors of vertices in $B(u)$, we recurse; that is, we find a good candidate vertex u'' among the neighbors of vertices in $B(u')$ and so on. Once the depth of recursion has reached t, we are able to show that among all the candidate vertices explored during the recursive queries, we would have found a vertex w along the shortest path between u and v such that $d(w,\ell(w)) \leq d(u,v)/(t+1)$. As discussed earlier, this leads to the desired bound on stretch.

The space-stretch-time trade-off. Finally, we comment on the three-way trade-off between space, stretch and query time in our distance oracle.

For any fixed stretch, our distance oracles achieve the trade-off between space and query time by way of construction. Unlike the construction algorithms in [11, 17], the size of the landmark set L in our oracles is controlled by a parameter $1 \leq \alpha \leq n$. As we increase the size of L, the size of the oracle increases since it stores the distance from each vertex in L to each other vertex. On the other hand, as the size of L increases, fewer edges need to be explored while growing the shortest path trees in each recursive step, leading to a smaller query time. Hence, for any fixed stretch, we get a smooth space-time trade-off using the parameter α.

The other spectrum of the trade-off is achieved by using the recursive query algorithm — for a fixed size of the oracle, we get a trade-off between stretch and query time. Fix some $1 \leq \alpha \leq n$ and hence, the size of the oracle. Then, we get a stretch-time trade-off by controlling the depth of recursion — the lower the desired stretch, the higher the query time. This is due to the fact that we are simply querying the same data structure (recursively) and hence, the size of the oracle is fixed; the query algorithm simply allows us to trade-off query time for improved stretch.

4 A distance oracle for stretch 3

In this section, we construct a distance oracle that returns stretch-3 distances for any weighted graph. Our oracle is similar in spirit to the oracles of [2, 17] with some simple, yet powerful modifications. We give a randomized construction of the oracle; our query algorithm is deterministic and hence, using Chernoff bound, all the results hold with high probability with an extra logarithmic factor. In particular, we prove the following lemma:

Lemma 4.1. *Let G be any weighted graph with n vertices and $m = O(n\mu)$ edges with non-negative weights. Then, for any $1 \leq \alpha \leq n$, we can construct a distance oracle of expected size $O(m + n^2/\alpha)$ and a query algorithm that computes stretch-3 distances in expected time $O(\alpha\mu + \alpha \log \alpha)$.*

Let L and V' be a given subset of vertices. Then, for any vertex v, we define the following:

- **Nearest vertex in set L — $\ell(v)$:** the vertex $a \in L$ that minimizes $d(v,a)$, ties broken arbitrarily.
- **Neighbor set $N(V')$:** the set of all the neighbors of vertices in V'.
- **Ball of a vertex $B(v)$:** the set of vertices $w \in V$ for which $d(v,w) < d(v,\ell(v))$.
- **Ball radius r_v:** the distance from v to its nearest neighbor in L, that is, $d(v,\ell(v))$.
- **Vicinity of a vertex $\Gamma(v)$:** the set of vertices in $B(v) \cup N(B(v))$.
- **Distance-via-ball from v to w — $d'_v(w)$:** cost of the least-cost path from v to w such that all intermediate vertices on this path are contained in $B(v)$; that is:

$$d'_v(w) = \min_{x \in N(w) \cap B(v)} \{d(v,x) + \text{weight of edge}(x,w)\}$$

The following result from [2, 3] will be useful to succinctly describe our results:

Lemma 4.2 ([2, 3]). *Let G be any weighted undirected graph with n vertices, m edges and average degree $\mu = 2m/n$. Then, one can construct an equivalent graph with maximum degree $\lceil \mu + 2 \rceil$, such that the new graph has $2n$ vertices, $m + n$ edges, and has the same distances between any pair of vertices as the distance in the original graph between the corresponding vertices.*

The reduction that leads to the above result does introduce some new zero-weight edges in the graph. Hence, the above implies that as long as the stretch bound of the query algorithm does not depend on the edge weights, given a graph with average degree μ, we can replace it with a graph with maximum degree no more than $\lceil \mu + 2 \rceil$, and build the oracle on this new graph instead of the original graph. Using the above result, **we can henceforth focus on degree-bounded graphs**. To this end, let $G = (V, E)$ be a μ-degree bounded graph where each vertex has at most $\mu = 2m/n$ neighbors.

Constructing the oracle. Fix some $1 \leq \alpha \leq n$. The construction begins by creating a set of "landmark" vertices by sampling each vertex independently at random with probability $1/\alpha$. Denote by L the set of landmark vertices. The distance oracle stores, for each $v \in V$, a hash table storing the exact distance to each vertex in L; it also stores the nearest neighbor $\ell(v)$ and the ball radius r_v. In addition, the distance oracle stores the entire graph — for each vertex v, the set of edges (and their weights) incident on v.

Query algorithm. We now show how to retrieve stretch-3 distances from the above distance oracle. When queried for the distance between vertices u, v, the algorithm runs modified shortest-path algorithms from u and from v that stop once the distances to all vertices in $B(u)$ and $B(v)$, respectively, have been computed. These distances are stored in a hash table temporarily. To answer the query, the algorithm works in three steps: (1) it checks if $v \in B(u)$ or $u \in B(v)$ — if any of these is true, the exact distance is returned using the hash table; (2) if the first step is unsuccessful, it checks if $r_u = 0$ or $r_v = 0$ — if one of these is zero (say r_u), the algorithm returns $d(u, \ell(u)) + d(v, \ell(u))$, which is easily proved to be exact by using triangle inequality; and (3) if the first two checks are unsuccessful, the algorithm returns $d(u, \ell(u)) + d(v, \ell(u))$, which is of stretch 3 using an essentially unmodified proof of [17].

The above distance oracle and the query algorithm are similar to that of the construction of Thorup and Zwick [17] with three main differences. First, our query algorithm computes balls and corresponding distances to vertices in the ball *on the fly*; second, to allow computation of these balls and distances on the fly, the graph is stored within the oracle; and third, the sizes of the balls are controlled by the parameter α. It is this specific construction that allows us to use a recursive query algorithm to retrieve distances of lower stretch without increasing the size of the oracle. The bounds on size of the oracle and the query time are easily proved; see Appendix A.

Next, we claim two properties that the above distance oracle (and the query algorithm) guarantee:

Claim 4.3. *For any pair of vertices $u, v \in V$ and any $\beta > 1$, the above query algorithm either returns the exact distance, or there exists a vertex $w \in \Gamma(u) \setminus B(u)$ such that: $d(u, w) + \beta \cdot d(w, v) < \beta \cdot d(u, v)$.*

The proof of the above claim (see Appendix A) follows by noting that the algorithm returns the exact distance if $r_u = 0$. Next, we will need the following claim which shows that if the vicinities of a pair of vertices $u, v \in V$ do not intersect, we can compute a lower bound on the distance between u and v:

Claim 4.4. *For any pair of vertices $u, v \in V$, if $\Gamma(u) \cap \Gamma(v) = \emptyset$, we have that $d(u, v) \geq r_u + r_v$.*

Claim 4.4 has been explicitly used in [2, 11] for designing oracles of stretch 2 and larger. In fact, a stronger result from [3] show that the same lower bound on $d(u, v)$ holds even if $B(u) \cap \Gamma(v) \neq \emptyset$. For sake of completeness, we provide a proof in Appendix A.

We will extensively use the above two claims throughout the rest of the paper. In particular, we will query the distance oracle of Lemma 4.1 recursively in a structured fashion; in each recursive step, we will argue that when queried for distance between a pair of vertices $u, v \in V$, the query algorithm either returns the exact distance between u and v or we can find a vertex $w \in \Gamma(u)$ such that $d(x, v)$ is strictly less than $d(u, v)$ (using Claim 4.3). Once such a vertex w is found, we will use it along with Claim 4.4 to lower bound the distance between u and v.

To bound the query time of our algorithm, we will need the following claim (proof in Appendix A):

Claim 4.5. *Let $G = (V, E)$ be any weighted μ-degree bounded graph. Then, for any vertex v, given the distance oracle of Lemma 4.1 and given a hash table containing distances to each vertex in $B(v)$, a hash table containing distance-via-ball to each vertex in $\Gamma(v)$ can be constructed in $O(|B(v)| \cdot \mu) = O(\alpha\mu)$ time.*

Algorithm 1 Query (u, v, t): the query algorithm.

Compute $d(u, x)$ for each $x \in B(u)$ and compute $d(v, y)$ for each $y \in B(v)$
Compute $d'_u(x)$ for each $x \in \Gamma(u)$ and compute $d'_v(y)$ for each $y \in \Gamma(v)$
If $v \in B(u)$ or $u \in B(v)$
    return $d(u, v)$
If $r_u \geq r_v$
    $q_1 \leftarrow u$; $q_2 \leftarrow v$
Else
    $q_1 \leftarrow v$; $q_2 \leftarrow u$

If $t > 1$
    return $\min_{x \in \Gamma(q_1) \setminus B(q_1)} \left\{ d'_{q_1}(x) + \textsc{Query}(x, q_2, t-1) \right\}$
$\gamma_1 \leftarrow \infty$, $\gamma_2 \leftarrow \infty$
If $\Gamma(q_1) \cap \Gamma(q_2) \neq \emptyset$
    $\gamma_1 \leftarrow \min_{x \in \Gamma(q_1) \cap \Gamma(q_2)} \left\{ d'_{q_1}(x) + d'_{q_2}(x) \right\}$
$\gamma_2 \leftarrow d(q_2, \ell(q_2)) + d(q_1, \ell(q_2))$
return $\min\{\gamma_1, \gamma_2\}$

5 A recursive query algorithm

In this section, we present a recursive query algorithm and use it along with the distance oracle of Lemma 4.1 to prove Theorem 1.1. Recall that the query algorithm of Lemma 4.1 computes stretch-3 distances in $O(\alpha\mu)$ time. In order to compute distances with improved stretch, our query algorithm recursively queries the oracle; with each query, the stretch improves and the query time increases. The main challenge is to improve stretch without significantly increasing query time, which we accomplish by performing recursive queries in a carefully structured fashion.

The high level idea of the query algorithm is as follows (see Algorithm 1). The input to the algorithm is a pair of vertices $u, v \in V$ and a positive integer t that determines the depth of recursion (and hence, the desired stretch). This depth is specified *at the time of querying* and each query can have a different depth (and hence, stretch and query time guarantees).

In each recursive step, the algorithm executes as follows. Given a pair of vertices u, v and an integer t, the first two steps are executed similar to the query algorithm of Lemma 4.1. The difference lies in Step (3) — if none of the conditions in first two steps is satisfied, the algorithm checks if the desired depth of recursion is reached or not. **If the desired depth of recursion is not yet reached**, the algorithm selects one of the vertices out of u and v (we return, in a moment, to the question of how this selection is done); call the selected vertex q_1 and the other vertex q_2. The algorithm then recursively initiates multiple queries, each asking for the distance between q_2 and one of the vertices $x \in \Gamma(q_1)$; the result of each such query is added to the corresponding distance $d'_{q_1}(x)$ and the minimum of these distances is returned.

We now return to the question of how the algorithm selects the vertex q_1 (out of u and v) to initiate the next level of recursive queries. As discussed in §3, various strategies exist (for instance, three strategies of Figure 1); we discuss one of such strategies that recurses through the vertices in the vicinity of the vertex with larger ball radius. The intuition behind using this particular strategy is that by recursing through the vertex with larger ball radius, we may be able to get a better lower bound on the distance between the source and the destination (using Claim 4.4). Hence, we use ball radii to guide our selection of vertex q_1 out of u and v.

We describe this using an example: suppose that the query is performed on a source-destination pair (u, v) and suppose we always query the vertices in the vicinity of the source (u for this pair). Consider the shortest path between u and v shown in Figure 2. Starting with (u, v) as the source and the destination, if we want to get a good lower bound (via Claim 4.4) on the distance between the source and the destination, (one of) the best order(s) to proceed would be $(u, v) \to (x_1, v) \to (v, x_2) \to (x_4, x_2) \to (x_3, x_2)$; that is, in each step, search the vicinity of the vertex with larger ball radius. Hence, in each recursive step, we "swap" the source and the destination vertices depending on whose ball radius is larger. This leads to the desired bound on the stretch.

The last question to settle is the execution of the algorithm **when the desired depth of recursion has been reached**. If none of the pairs of vertices

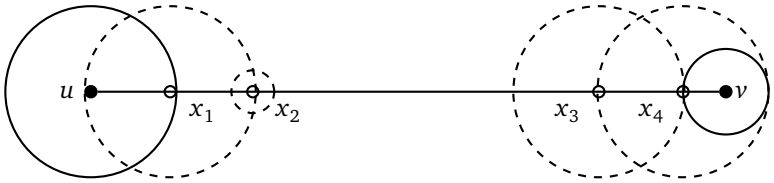


Figure 2. An illustration of the idea of "swapping" of vertices used in the query algorithm. The path shown is the shortest path between u and v; the circles around the vertices denote their balls.

queried during the execution of the algorithm satisfies the conditions for retrieving exact distances (line 3 and line 4 in Algorithm 1), the last query has to retrieve the distance via the landmark vertex of one of the vertices. Referring to Figure 2 again, once we query for distance between (x_2, x_3), which one should return the distance via its landmark vertex? Considering the worst-case scenario, since $d(x_2, \ell(x_2)) < d(x_3, \ell(x_3))$, it makes sense to retrieve the distance via the landmark of the vertex with smaller ball radius. With such a strict ordering of querying the vertices, we get the desired bound on the stretch for the distance retrieved by the query algorithm.

5.1 Formal Analysis of the query algorithm In the rest of the section, we assume that the query algorithm does not return the exact distance and terminates with some vertex returning the distance via its landmark vertex. We start with a simple observation:

Observation 5.1. *Each successive query contains a vertex from the previous query. Furthermore, the ball radius of the retained vertex is smaller than the ball radius of the dropped vertex.*

Suppose we perform a depth-t recursive query between vertices u and v. Let $x_1, x_2, \ldots, x_t, x_{t+1}$ be the set of vertices on the recursion path (in some arbitrary order) along which the final distance is returned. Note that the number of vertices in the set of queries is exactly one more than the depth of recursion. Without loss of generality, assume that the final query was performed on the pair of vertices x_t, x_{t+1} and x_{t+1} returns the distance via its landmark vertex $\ell(x_{t+1})$.

Then, we make the following claim, a proof of which is relatively straightforward using Observation 5.1:

Claim 5.2. *Let $x_1, x_2, \ldots, x_t, x_{t+1}$ be the set of vertices on the recursion path (in some arbitrary order) along which the final distance is returned and let x_{t+1} be the vertex that returns the distance in the last recursive call. Then, we have that: $r_{x_{t+1}} \leq r_{x_i}$, $\forall i \leq t$.*

Recall from the statement of Claim 4.4 that if the vicinities of two vertices do not intersect, it is possible to lower bound the exact distance between the two vertices. The following lemma generalizes the result of Claim 4.4 to the case of algorithm performing recursive queries with depth-t:

Lemma 5.3. *Let $x_1, x_2, \ldots, x_t, x_{t+1}$ be the set of vertices on the recursion path (in some arbitrary order) along which the final distance is returned and let x_{t+1} be the vertex that returns the distance in the last recursive call. Then, either the query algorithm returns the exact distance or the distance between the source and the destination is bounded by below as $d(u,v) \geq r_{x_1} + \cdots + r_{x_t} + r_{x_{t+1}} \geq (t+1) r_{x_{t+1}}$.*

Proof: Assume that the vertices on the recursion path along which the final distance is returned lie on the shortest path between u and v; indeed, if this were not the case, the stretch can only be smaller and the analysis will only give us a pessimistic bound. Consider the case when the query algorithm does not return the exact distance between u and v. Consider some query $\text{Query}(x, y, i)$ and without loss of generality, assume that the next query is $\text{Query}(x', y, i-1)$ for some $x' \in \Gamma(x) \backslash B(x)$. Then, since $x' \in \Gamma(x) \backslash B(x)$ and x' lies along the shortest path between x and y, we have that (recall, r_x is defined to be the radius of the ball $B(x)$)

$$d(x,y) = d(x,x') + d(x',y) \geq r_x + d(x',y)$$

Using the above expression on the pair of vertices along the recursion path for $\text{Query}(u, v, t)$ and assuming that the last query is performed on pair of vertices x_t, x_{t+1}, we get the following expression as a lower bound on the distance between u and v:

$$d(u,v) \geq r_{x_1} + r_{x_2} + \cdots + r_{x_{t-1}} + d(x_t, x_{t+1})$$

Let $P = (x_t, w_1, w_2, \ldots, x_{t+1})$ be the shortest path between x_t and x_{t+1} and let $w = w_{i_0}$ where $i_0 = \max\{i | w_{i-1} \in P \cap B(x_t)\}$. Then, if the query algorithm does not return the exact distance, we have that $w \notin \Gamma(x_{t+1})$ since otherwise we get the exact distance (using line 13 of Algorithm 1). Since $w \notin \Gamma(x_{t+1})$,

$$
\begin{aligned}
(1) \quad \delta(u,v) &= d(u,v) - d(x_t, x_{t+1}) + \delta(x_t, x_{t+1}) \\
(2) \quad &= d(u,v) - d(x_t, x_{t+1}) + d(x_{t+1}, \ell(x_{t+1})) + d(\ell(x_{t+1}), x_t) \\
(3) \quad &\leq d(u,v) - d(x_t, x_{t+1}) + d(x_{t+1}, \ell(x_{t+1})) + d(\ell(x_{t+1}), x_{t+1}) + d(x_{t+1}, x_t) \\
(4) \quad &= d(u,v) + 2 \cdot d(x_{t+1}, \ell(x_{t+1})) \\
(5) \quad &= d(u,v) + 2 \cdot r_{x_{t+1}}
\end{aligned}
$$

we get using Claim 4.4 that $d(x_t, x_{t+1}) \geq r_{x_t} + r_{x_{t+1}}$. Hence, we get the following lower bound on the distance between u and v:

$$d(u,v) \geq r_{x_1} + \cdots + r_{x_t} + r_{x_{t+1}}$$

Using the result from Claim 5.2 on the above expression gives us the desired bound. □

The final task is to provide an upper bound on the distance returned by the query algorithm; when combined with the lower bound on the exact distance between the source-destination pair, this will easily lead to a bound on the stretch. The following lemma suggests that the distance returned by the query algorithm with recursion depth t can not be much larger than the exact distance between u and v:

Lemma 5.4. *Let $x_1, x_2, \ldots, x_t, x_{t+1}$ be the set of vertices on the recursion path (in some arbitrary order) along which the final distance is returned and let x_{t+1} be the vertex that returns the distance in the last recursive call. Then, algorithm* Query(u, v, t) *returns distance that satisfies:*

$$\delta(u,v) \leq d(u,v) + 2 \cdot r_{x_{t+1}}$$

Proof: If the query algorithm returns the exact distance, the lemma trivially holds. Consider the case when such is not the case. We prove the lemma for the case when the vertices on the recursion path lie along the shortest path between u and v; as discussed earlier, if any other set of vertices return a shorter path, our analysis will only lead to a pessimistic bound on the retrieved distance.

First, we claim that assuming that the last query is on pair of vertices x_t, x_{t+1}, the distance returned by the query algorithm is given by $\delta(u,v) = (d(u,v) - d(x_t, x_{t+1})) + \delta(x_t, x_{t+1})$. To see this, recall our assumption that each of the vertices $x_1, x_2, \ldots, x_t, x_{t+1}$ lie along the shortest path. Hence, line 10 always adds up the *exact* distance in each recursive step; this follows by noting that if the vertices lie along the shortest path, then for any pair of vertices x_i, x_j, $d'_{x_i}(x_j) = d(x_i, x_j)$.

If x_{t+1} returns the distance via its landmark vertex, the distance returned by the distance oracle $\delta(u,v)$ is given by Eq. 1, or equivalently, by Eq. 2. The upper bound of Eq. 3, or equivalently, of Eq. 4 on the distance returned by the distance oracle follows using triangle inequality. Finally, the last simplification from Eq. 4 to Eq. 5 follows by the definition of ball radius, leading to the proof of the lemma.

□

We are now ready to prove the desired bounds on size of the oracle, the query time and the stretch of the distances returned by the query algorithm:

Proof of Theorem 1.1. The claim regarding the size of the distance oracle follows from Lemma 4.1. It remains to bound the query time and stretch. We start by proving the bound on query time. In the worst case, the distance is returned after t recursive calls of the query algorithm. First, we note that for any vertex v, $|B(v)| = O(\alpha)$ and since the graph is assumed to be μ-degree bounded, we have that $|\Gamma(v) \setminus B(v)| = O(\alpha\mu)$. Hence, for any vertex v, distances to vertices in the ball and distance-via-ball to vertices in the vicinity can be computed in time $O(\alpha\mu + \alpha \log \alpha)$ using results of Lemma 4.1 and Claim 4.5.

Also, using the bound on the number of vertices in $\Gamma(v) \setminus B(v)$, we have that in the i-th call, there are at most $O((\alpha\mu)^{(i-1)})$ vertices for which the condition of line 10 or line 13 is checked by the query algorithm; and each of these vertices have a vicinity of size $O(\alpha\mu)$. Hence, the time required to complete the i-th recursive query is $O((\alpha\mu)^{i-1} \cdot (\alpha\mu + \alpha \log \alpha))$. We now prove the bound on stretch. Using Lemma 5.4, we have that:

$$
\begin{aligned}
\delta(u,v) &\leq d(u,v) + 2 \cdot r_{x_{t+1}} \\
&\leq d(u,v) + 2 \cdot \frac{d(u,v)}{t+1} \qquad \text{(Lemma 5.3)} \\
&= \left(1 + \frac{2}{t+1}\right) d(u,v)
\end{aligned}
$$

□

6 Improving space-time trade-off

The distance oracle of Theorem 1.1 returns distances of stretch $1+2/(t+1)$ using $O(m+n^2/\alpha)$ space and $O((\alpha\mu)^t+(\alpha\mu)^{t-1}\alpha\log\alpha)$ query time. In this section, we show how to reduce the query time for the above oracle to $O((\alpha+\mu)^t)$ at the expense of a small additive stretch. We also show how to improve the space-time trade-off of Theorem 1.1 for the special case of $t=2$.

Theorem 6.1. *Let G be any weighted undirected graph with n vertices and $m=O(n\mu)$ edges and let w_{uv} be the weight of the heaviest edge along the shortest path between any pair of vertices u and v. For any integer $t>0$, denote by $\beta=2/(t+1)$. Then, for any $1\leq\alpha\leq n$ and any integer $t>0$, we can construct a distance oracle of size $O(m+n^2/\alpha)$ and a query algorithm that, when given two vertices u and v at distance d, returns a distance of at most $(1+\beta)d+(2-\beta)w_{uv}$ in time $O\left((\alpha+\mu)^t\right)$.*

For unweighted graphs with $m=\tilde{O}(n^{5/4})$ edges, for instance, the above theorem gives us a distance oracle of size $\tilde{O}(n^{7/4})$ that returns stretch-$(5/3,4/3)$ distances in $O(\sqrt{n})$ query time. This improves upon the space-time trade-off of Theorem 1.1 (which achieves the same space and query time only for graphs with $m=\tilde{O}(n)$ edges) at the expense of an additive stretch of $4/3$.

The distance oracle used by Theorem 6.1 is the same as in Lemma 4.1; it is our query algorithm (presented in Appendix B) that allows us to reduce the query time at the expense of a small additive stretch. The high level difference between the query algorithm of Theorem 1.1 and the one for additive stretch is that for any query between vertices u,v, it is no more necessary to recurse through vertices in the vicinity $\Gamma(u)\setminus B(u)$; it suffices to recurse on vertices in $B(u)\cup N(u)$ — that is, through vertices that are either in the ball or the neighbors of the source. Since balls are roughly a factor μ smaller than the vicinities, we achieve a reduced query time. The additive factor in stretch comes due to the fact that in comparison to the query algorithm of Theorem 1.1, the amount of progress that we make towards the destination in each subsequent query is now reduced by an amount equal to the weight of the edge along the shortest path that connects the vertex in $B(u)$ to its neighbor. We prove Theorem 6.1 in Appendix B.

It is possible to further improve the space-time trade-off in Theorem 1.1 and in Theorem 6.1 for the very special case of $t=2$ by using our query algorithms on the distance oracles of [11]:

Theorem 6.2. *Let G be any weighted undirected graph with n vertices and $m=O(n\mu)$ edges. Then, there exists a distance oracle of size $O(n^{4/3}m^{1/3})$ and a query algorithm that, in the worst case, returns a stretch-$5/3$ distance in time $O(n^{1/3}m^{1/3})$.*

Theorem 6.3. *Let G be any weighted undirected graph with n vertices and $m=O(n\mu)$ edges. Then, there exists a distance oracle of size $O(n^{5/3})$ and a query algorithm that, when given two vertices u and v at distance d, returns a distance of at most $5/3\times d+4/3\times w_{uv}$ in time $O(n^{2/3})$ where w_{uv} is the weight of the heaviest edge along the shortest path between u and v.*

The proofs for Theorem 6.2 and Theorem 6.3 use ideas similar to those of Theorem 1.1 and Theorem 6.1, respectively. For sake of completeness, we provide these proofs in Appendix C.

7 Open Problems

Pătraşcu and Roddity, in [11], raised the question of achievable stretch with subquadratic size distance oracles and polynomial, albeit sublinear, query times. Our paper partially answers their question, but many questions raised in [2,11] remain unresolved: can we reduce the query time without significant loss in size and/or stretch? Can we improve size and/or query time for stretch greater than 2? Can we derive lower bounds for some restricted cases, say $O(\text{polylog}(n))$ query time?

Allowing higher query time to reduce the size and/or stretch leads to several interesting possibilities in related problems:

- Distance oracles with constant query time can be used to design compact routing schemes [16]; the case of super-constant query time is no different — higher query time can often be taken care of by using lightweight handshaking schemes [2,3]. Can we design distributed compact routing schemes for our distance oracles without significantly stretching the path of the first packet?

- While it seems significantly more challenging, can this line of research lead to a $o(mn)$ time combinatorial algorithm for computing all-pair approximate shortest paths (APASP) for stretch less than 2? The only result known for computing APASP with stretch less than 2 is due to Zwick, which uses matrix multiplication [6,20].

Acknowledgments. The authors would like to thank the anonymous reviewers for their suggestions. We gratefully acknowledge the support of NSF grant CNS 10-17069.

References

[1] R. Agarwal, M. Caesar, P. B. Godfrey, and B. Y. Zhao. Shortest paths in less than a millisecond. *ACM SIGCOMM Workshop on Online Social Networks (WOSN)*, 2012.

[2] R. Agarwal, P. B. Godfrey, and S. Har-Peled. Approximate distance queries and compact routing in sparse graphs. *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 1754–1762, 2011.

[3] R. Agarwal, P. B. Godfrey, and S. Har-Peled. Faster approximate distance queries and compact routing in sparse graphs. http://arxiv.org/abs/1201.2703, 2012. ArXiv.

[4] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. *Proc. ACM International Conference on World Wide Web (WWW)*, 835–844, 2007.

[5] S. Baswana, A. Gaur, S. Sen, and J. Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. *Proc. International Colloquium on Automata, Languages and Programming (ICALP)*, 609–621, 2008.

[6] S. Baswana and T. Kavitha. Faster algorithms for approximate distance oracles and all-pair small stretch paths. *Proc. IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 591–602, 2006.

[7] S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Transactions on Algorithms* 2(4):557–577, 2006.

[8] CAIDA – The Cooperative Association for Internet Data Analysis. ⟨http://www.caida.org/home/⟩.

[9] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, M. Luckie, kc claffy, and C. Shannon. The ipv4 routed /24 as links dataset, November 2010.

[10] M. Mendel and A. Naor. Ramsey partitions and proximity data structures. *Journal of European Mathematical Society* 2(9):253–275, 2007.

[11] M. Pătraşcu and L. Roditty. Distance oracles beyond the Thorup-Zwick bound. *Proc. IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 815–823, 2010.

[12] E. Porat and L. Roditty. Preprocess, set, query! *Proc. European Conference on Algorithms (ESA)*, 603–614, 2011.

[13] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. *ACM Conference on Information and Knowledge Management (CIKM)*, 867–876, 2009.

[14] D. Schultes. *Route Planning in Road Networks*. PhD Thesis, University of Karlsruhe, February, 2008.

[15] C. Sommer, E. Verbin, and W. Yu. Distance oracles for sparse graphs. *Proc. IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 703–712, 2009.

[16] M. Thorup and U. Zwick. Compact routing schemes. *Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1–10, 2001.

[17] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM* 52(1):1–24, 2005.

[18] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. *Proc. ACM Conference on Information and Knowledge Management (CIKM)*, 563–572, 2007.

[19] C. Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 202–208, 2012.

[20] U. Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM* 49(3):289–317, 2002.

Appendix

A Proofs for §4

We start by providing the proof for Lemma 4.1; this requires proving the bound on size and the query time. The bound on stretch follows easily using triangle inequality and an essential unmodified proof from [17]. Recall that we are given a weighted undirected graph with n vertices and $m = O(n\mu)$ edges; and the graph is μ-degree bounded.

Proof of Lemma 4.1. Recall that each vertex in the graph is sampled uniform randomly with probability $1/\alpha$ for inclusion in set L. Hence, it follows that $E[|L|] = O(n/\alpha)$. Storing exact distances from each vertex in the graph to each vertex in L, hence, requires $O(n^2/\alpha)$ space, in expectation. In addition, each vertex stores the exact distance to each of its neighbors, requiring an additional $O(m)$ storage; storing $\ell(v)$ and r_v requires an additional $O(1)$ space. Hence, the size of the distance oracle is $O(m + n^2/\alpha)$, in expectation.

We now bound the query time. To start with, we note that since vertices in L are selected uniform randomly with probability $1/\alpha$, each vertex v has a ball of size $O(\alpha)$, in expectation; this is easily proved using an argument similar to [17, Lemma 3.2]. Computing the distance to each vertex in the ball requires time $O(\alpha\mu + \alpha\log\alpha)$ using the modified Dijkstra's algorithm presented in [17]. Once the distances to vertices in the ball are computed, the checks required by the query algorithms can be performed in $O(1)$ time, leading to the desired proof. □

Proof of Claim 4.3. Consider the case when the algorithm does not return the exact distance — that is, when $v \notin B(u)$ and $r_u > 0$. Let $P = (u, x_1, x_2, \ldots, v)$ be the shortest path between u and v. Let $i_0 = \max\{i | x_{i-1} \in P \cap B(u)\}$. Now consider the vertex $w =$

x_{i_0}. Clearly, $w \in \Gamma(u) \setminus B(u)$ since $x_{i_0-1} \in B(u)$ and $w \in N(x_{i_0-1})$; furthermore, $d(u,w) \geq r_u > 0$. The proof follows by noting that w lies along the shortest path from u to v and hence, $d(u,v) - d(w,v) = d(u,w)$. □

Proof of Claim 4.4. Let $P = (u, x_1, x_2, \ldots, v)$ be the shortest path between u and v. Consider the vertex $w = x_{i_0}$, where $i_0 = \max\{i | x_{i-1} \in P \cap B(u)\}$. By definition, $w \in \Gamma(u) \setminus B(u)$ and hence, $d(u,w) \geq r_u$. Furthermore, since $\Gamma(u) \cap \Gamma(v) = \emptyset$, we have that $w \notin \Gamma(v)$ and hence, $d(v,w) \geq r_v$. The proof follows using the fact that w is on the shortest path between u and v. □

Proof of Claim 4.5. We describe how to construct a hash table $\mathcal{H}$ that contains, for each vertex v, the distance-via-ball to all vertices $w \in \Gamma(v)$ in time $O(|B(v)| \cdot \mu)$. This assumes that the hash tables containing the vertices in the ball have already been constructed. Consider any vertex v; first, we copy each entry from the hash table containing distances to vertices in $B(v)$ into $\mathcal{H}$. Next, we iterate through each vertex $x \in B(v)$ and perform the following: for each neighbor $x' \in N(x)$, check if $x' \in B(v)$ — if yes, do nothing. If $x' \notin B(v)$, check if there is an entry for x' in $\mathcal{H}$. If no, create an entry with x' as the key and $(d(v,x) + \text{weight of edge}(x,x'))$ as the value. If there is already an entry, check if $(d(v,x) + \text{weight of edge}(x,x'))$ is less than the value corresponding to the key x' in $\mathcal{H}$; if yes, update the entry.

The above algorithm requires, for any vertex v, iterating through all vertices in $B(v)$ and their neighbors; since the graph is assumed to be a μ-degree bounded graph, computing distances-via-ball for any vertex v requires $O(|B(v)| \times \mu)$ time. Using proof of Lemma 4.1, this amounts to $O(\alpha\mu)$ time. □

B Proof of Theorem 6.1

In this section, we prove Theorem 6.1. The distance oracle used in the proof is the same as that of Lemma 4.1. Here, we give the query algorithm and then analyze the stretch and the query time. Since most of the results from §4 and §5 naturally follow for the case of additive stretch, we only focus on the differences that allow us to achieve an additive stretch. Recall from the discussion in §4 that it suffices to focus on $\mu = 2m/n$ degree bounded graphs.

B.1 Query algorithm.

The query algorithm for additive stretch is given in Algorithm 2. The algorithm is a slightly modified version of the query algorithm from §5.

B.2 Analysis.

Recall that our distance oracle for additive stretch is the one used in Lemma 4.1; hence, the bound on the size follows using the proof of Lemma 4.1. Moreover, recall from the proof of Lemma 4.1, that $E[|L|] = O(n^2/\alpha)$ and for any vertex $v \notin L$, $|B(v)| = O(\alpha)$ and $|\bar{\Gamma}(v)| = O(\alpha + \mu)$, in expectation.

Using the above easy observations, we now analyze the stretch and the query time for the above algorithm. We start with the following claim, which is akin to Claim 4.3:

Claim B.1. *For any $u, v \in V$ and any $\beta > 1$, the query algorithm of Lemma 4.1 either returns the exact distance, or there exists a vertex $w \in \bar{\Gamma}(u)$ such that: $d(u,w) + \beta \cdot d(w,v) < \beta \cdot d(u,v)$.*

Next, for the purposes of proving an additive stretch, we will need a simple modification in the statement of Claim 4.4, which we state below:

Claim B.2. *Let $u, v \in V$ and let $P = (u, x_1, x_2, \ldots, v)$ be the shortest path between u and v with the weight of the heaviest edge being w_{uv}. Furthermore, let $w = x_{i_0}$, where $i_0 = \max\{i | x_i \in P \cap \bar{\Gamma}(u)\}$. If $w \notin \bar{\Gamma}(v)$, $d(u,v) \geq r_u + r_v - w_{uv}$.*

Proof: Let $P = (u, x_1, x_2, \ldots, v)$ be the shortest path between u and v. Let $i_0 = \max\{i | x_i \in P \cap \bar{\Gamma}(u)\}$ and $w = x_{i_0}$ and $w' = x_{i_0+1}$. Then, since $w' \notin B(u)$, we have that $d(u,w') \geq r_u$ and $d(w,w') \leq w_{uv}$; this gives us that $d(u,w) = d(u,w') - d(w,w') \geq r_u - w_{uv}$. Furthermore, since $w \notin \bar{\Gamma}(v)$, we get that $d(v,w) \geq r_v$. The bound follows by noting that w lies on the shortest path between u and v and hence, $d(u,v) = d(u,w) + d(v,w) \geq r_u + r_v - w_{uv}$. □

Note that the query algorithm for additive stretch, similar to that of purely multiplicative stretch, iterates through vertices of the source — the vertex with larger ball radius. Hence, the results of Observation 5.1 and Claim 5.2 hold for the case of additive stretch query algorithm. More precisely, using the same notation as in §5, we can prove that $r_{x_{t+1}} \leq r_{x_i}$, for all $i \leq t$.

The rest of the proof is structured as for the case of purely multiplicative stretch — we first provide a lower bound on the exact distance between the source-destination pair using the depth of recursion; next, we provide an upper bound on the distance returned by the query algorithm and finally, use these two bounds to provide bounds on stretch.

Algorithm 2 QueryA (u, v, t): the distance query algorithm for additive stretch distance oracle. We denote by $\bar{\Gamma}(v)$ the set of vertices in $B(v) \cup N(v)$.

Compute $d(u,x)$ for each $x \in B(u)$ and compute $d(v,y)$ for each $y \in B(v)$
Compute $d'_u(x)$ for each $x \in \bar{\Gamma}(u)$ and compute $d'_v(y)$ for each $y \in \bar{\Gamma}(v)$
If $v \in B(u)$ or $u \in B(v)$
    return $d(u,v)$
If $r_u \geq r_v$
    $q_1 \leftarrow u$; $q_2 \leftarrow v$
Else
    $q_1 \leftarrow v$; $q_2 \leftarrow u$

If $t > 1$
    return $\min_{x \in \bar{\Gamma}(q_1)} \{d(q_1,x) + \textsc{Query}(x, q_2, t-1)\}$
$\gamma_1 \leftarrow \infty$, $\gamma_2 \leftarrow \infty$
If $\bar{\Gamma}(q_1) \cap \bar{\Gamma}(q_2) \neq \emptyset$
    $\gamma_1 \leftarrow \min_{x \in \bar{\Gamma}(q_1) \cap \bar{\Gamma}(q_2)} \{d(q_1,x) + d(q_2,x)\}$
$\gamma_2 \leftarrow d(q_2, \ell(q_2)) + d(q_1, \ell(q_2))$
return $\min\{\gamma_1, \gamma_2\}$

The central difference between the two query algorithm is in terms of the lower bound on the exact distance between the source-destination pair. We have the following lemma that allows us to prove the desired bound on the additive stretch. The proof to Lemma B.3 is essentially similar to that of Lemma 5.3 – the only difference is that we use the bound of Claim B.2 rather than Claim 4.4.

Lemma B.3. *Let $x_1, x_2, \ldots, x_t, x_{t+1}$ be the set of vertices on the recursion path (in some arbitrary order) along which the final distance is returned and let x_{t+1} be the vertex that returns the distance in the last recursive call. Then, either the query algorithm returns the exact distance or the distance between the source and the destination is bounded by below as:*

$$d(u,v) \geq (t+1) \cdot r_{x_{t+1}} - t \cdot w_{uv}$$

Proof: Assume that the vertices along the recursion path lie on the shortest path between u and v and that the query algorithm does not return the exact distance between u and v. Consider some query QueryA(x, y, i). Let $P = (x, x_1, x_2, \ldots, y)$ be the shortest path between x and y. Let $i_0 = \max\{i | x_i \in P \cap \bar{\Gamma}(x)\}$ and $x' = x_{i_0}$. Since x initiates a recursive call through all vertices in $\bar{\Gamma}(x)$, one such query must be initiated to x'; consider query QueryA$(x', y, i-1)$. Then, since $N(x') \notin \bar{\Gamma}(x)$, we have that $d(x,x') \geq r_x - w_{xy}$ (recall, r_x is defined to be the radius of the ball $B(x)$). Using the fact that x' lies on the shortest path between x and y, we get that:

$$d(x,y) = d(x,x') + d(x',y) \geq r_x - w_{xy} + d(x',y)$$

Using the above expression on the pair of vertices along the recursion path for QueryA(u, v, t) and assuming that the last query is performed on pair of vertices x_t, x_{t+1}, we get the following expression as a lower bound on the distance between u and v:

$$d(u,v) \geq r_{x_1} - w_{uv} + \cdots + r_{x_{t-1}} - w_{uv} + d(x_t, x_{t+1})$$

which is equivalent to

$$d(u,v) \geq r_{x_1} + r_{x_2} + \cdots + r_{x_{t-1}} - (t-1) w_{uv} + d(x_t, x_{t+1})$$

In the last step, we use Claim B.2 on x_t, x_{t+1} pair, giving us: $d(x_t, x_{t+1}) \geq r_{x_t} + r_{x_{t+1}} - w_{uv}$. Hence, we get the following lower bound on the distance between u and v:

$$d(u,v) \geq r_{x_1} + \cdots + r_{x_t} + r_{x_{t+1}} - t \cdot w_{uv}$$

Recall that the result of Claim 5.2 holds for the distance oracle and the query algorithm for additive stretch; using it on the above expression gives us the desired bound. □

The distance returned by the query algorithm for additive stretch can be bounded as follows:

Lemma B.4. *Let $x_1, x_2, \ldots, x_t, x_{t+1}$ be the set of vertices that were ever queried by the query algorithm (in some arbitrary order) and let x_{t+1} be the vertex that returns the distance via its landmark vertex $\ell(x_{t+1})$. Then, the algorithm* QueryA(u, v, t) *returns distance that satisfies:*

$$\delta(u,v) \leq d(u,v) + 2 \cdot r_{x_{t+1}}$$

The proof for the above lemma is the unmodified proof of Lemma 5.4 from §5.

Proof of Theorem 6.1. Using the oracle of Lemma 4.1, we get the bound on size. It remains to bound the stretch and query time; we start with the latter. In the worst case, the distance is returned after t recursive calls. In the i-th call, there are at most $O((\alpha+\mu)^{i-1})$ vertices for which line 10 or line 13 of the algorithm is executed; each of these vertices recurse through at most $O(\alpha+\mu)$ vertices. Hence, i-th call takes time $O\left((\alpha+\mu)^t\right)$.

We now prove the bound on stretch. Using Lemma B.4, we have that

$$\delta(u,v) \leq d(u,v) + 2 \cdot r_{x_{t+1}}$$

which using the bound from Lemma B.3, gives us:

$$\begin{aligned}
\delta(u,v) &\leq d(u,v) + 2 \cdot \frac{d(u,v) + t \cdot w_{uv}}{t+1} \\
&= \left(1 + \frac{2}{t+1}\right) d(u,v) + \frac{2t}{t+1} w_{uv} \\
&= \left(1 + \frac{2}{t+1}\right) d(u,v) + \left(2 - \frac{2}{t+1}\right) w_{uv}
\end{aligned}$$

which by setting $\beta = 2/(t+1)$, completes the proof. □

C Improving space-time trade-off for $t = 2$

We start by briefly describing the high level idea of the oracle of [11]. Their distance oracle, as in [17], constructs a set L of landmark vertices. Each landmark vertex stores the distance to each other vertex in the graph. Each vertex $v \in V \setminus L$ stores the distances to the vertices in its ball $B(v)$ and to their neighbors; that is, to the vertices in its vicinity $\Gamma(v)$. Furthermore, each vertex $v \in V \setminus L$ also stores distances to vertices u for which $\Gamma(u) \cap \Gamma(v) \neq \emptyset$; lets call this extended vicinity set and refer to it as $\Gamma'(v)$. When queried for distances between two vertices u and v, if $u \in \Gamma'(v)$ or if $v \in \Gamma'(u)$, the distance oracle returns the exact distance; if not, then the vertex with smaller ball radius returns the distance to the destination via its landmark vertex – this gives a bound of 2 on the stretch.

By using an elegant landmark selection algorithm, they are able to show that there exists a set of $O(n^{1/3}m^{1/3})$ landmark vertices such that every vertex in the graph has an extended vicinity set of size no more than $O(n^{1/3}m^{1/3})$. This leads to the size bound on the distance oracle – storing distances from landmark vertices to each other vertex in the graph requires $O(n^{4/3}m^{1/3})$ space, which is same as each vertex storing its extended vicinity set. Hence, the total size of the distance oracle is $O(n^{4/3}m^{1/3})$.

If we use the query algorithm of §5 on this distance oracle, the algorithm checks in time $O(n^{1/3}m^{1/3})$ whether there exists a vertex $x \in \Gamma'(u)$ such that $\Gamma'(x) \cap \Gamma'(v) \neq \emptyset$. If such an x is found, the exact distance will be returned. If no such x exists, the returned distance is minimum over all $x \in \Gamma'(u)$, $\delta(u,v) = d(u,x) + \delta(x,v)$. As earlier, we can restrict the analysis to all vertices x that lie along the shortest path between u and v (there must be some vertex in $\Gamma'(u)$ that lies along the shortest path, by definition of $\Gamma'(u)$), which gives us that the returned distance is upper bounded as $\delta(u,v) \leq d(u,v) + 2 \cdot r_x$, precisely as in the proofs of Lemma 5.4 and of Lemma B.4. The remaining task is to find a lower bound on $d(u,v)$, for which we have the following lemma:

Lemma C.1. *Let $P = (u, x_1, x_2, \ldots, v)$ be the shortest path between u and v and let $i_1 = \max\{i | x_i \in \Gamma'(u) \setminus B(u)\}$. If the final distance is returned via the landmark vertex of a vertex, $d(u,v) \geq r_u + r_{x_{i_1}} + r_v$.*

Proof: Let $i_1 = \max\{i | x_i \in \Gamma'(u) \setminus B(u)\}$ and let $i_2 = \max\{i | x_i \in \Gamma'(x_{i_1})\}$. Since $\Gamma'(x_{i_1}) \cap \Gamma'(v) = \emptyset$, we have that $x_{i_2} \notin \Gamma'(v)$. Hence, as in the proof of Lemma 4.4, we have that $d(x_{i_1}, v) \geq r_{x_{i_1}} + r_v$. Furthermore, since $d(u, r_{x_{i_1}}) \geq r_u$, we get the desired bound. □

Without loss of generality, assume that x_{i_1} is the vertex that returns the distance via its landmark vertex, then, we have that $d(u,v) \geq 3 \cdot r_{x_{i_1}}$, which using the upper bound from the discussion above leads to 5/3 being an upper bound on the stretch.

The bound on the query time follows from the fact that only one of the vertices out of u or v can explore the vertices in its extended vicinity leading to $O(n^{1/3}m^{1/3})$ query time. Exploring the vicinities further, as in our earlier algorithms, will lead to super linear query times, for which trivial oracles are known.

For the proof of Theorem 6.3, we note that everything but the lower bound on the distance between u and v remains the same. For the lower bound, we have the following lemma, the proof to which is exactly similar to that of Claim B.2:

Lemma C.2. *Let $P = (u, x_1, x_2, \ldots, v)$ be the shortest path between u and v and let $i_1 = \max\{i | x_i \in \Gamma'(u) \setminus B(u)\}$. If the query algorithm returns the distance via the landmark vertex of some vertex, $d(u,v) \geq r_u + r_{x_{i_1}} + r_v - 2w_{uv}$.*

Using this lemma, along with the upper bound on the returned distance, we get the desired bound on the stretch. The bounds on size and query time follow from the construction of the distance oracle.