Approximate Distance Queries in Sparse Graphs

Rachit Agarwal, P. Brighten Godfrey and Sariel Har-Peled
University of Illinois at Urbana-Champaign, USA
{agarwa16, pbg, sariel}@illinois.edu

***Abstract*—An approximate distance query data structure is a compact representation of a graph. It allows querying for approximate shortest paths between any pair of vertices in the graph. Thorup and Zwick proved that any approximate data structure that allows retrieval of stretch $2k-1$ paths in constant time must require space $\Omega(n^{1+1/k})$. The hard cases that enforce their lower bound are, however, rather dense graphs with average degree $\Omega(n^{1/k})$.**

We present data structures that, for sparse graphs, substantially break the Thorup-Zwick lower bound barrier, albeit at the expense of slightly higher query time. For instance, for the realistic scenario of a graph with n vertices and average degree $\Theta(\log n)$, the data structure of Thorup and Zwick requires $O(n^{3/2})$ space and constant query time for stretch 3 paths. Special cases of our data structures can retrieve stretch 2 paths with $\tilde{O}(n^{3/2})$ space and stretch 3 paths with $\tilde{O}(n)$ space, albeit at the cost of $\tilde{O}(\sqrt{n})$ query time.

Moreover, supported by large-scale simulations on graphs including the AS-level Internet graph, we argue that the stretch-2 scheme would be simple and efficient to implement in practice as a distributed compact routing protocol.

I. Introduction

An *approximate distance query* (ADQ) data structure is a compact representation of a graph G that allows retrieval of approximate distance between any two vertices in the graph. The fundamental trade-off in constructing an ADQ structure is between space (size of the data structure) and stretch (the ratio of the distance returned by the data structure and the actual shortest distance between the two vertices). For general graphs, the optimal[1] space/stretch trade-off was achieved by Thorup and Zwick [1]: their ADQ structure, for any graph with n vertices and for any integer $k \geq 2$, is of size $O(kn^{1+1/k})$ and returns paths with stretch $(2k-1)$ in time $O(k)$. However, the hard instances for the matching lower bound are rather dense graphs, with average degree $\Omega(n^{1/k})$. The lower bound essentially states that there exist specially constructed graphs that are *incompressible*: the best bound it can prove only implies that the size of the data structure is lower bounded by the number of edges in the graph.

Thus, classic ADQ results may be quite far from optimal for the case of sparse graphs, i.e., graphs with low average degree Δ. And this case is in fact of key interest since real-world graphs are sparse, with degrees much closer to logarithmic than polynomial in n. For instance, letting $\Delta = c \log_2 n$, empirically, $c \simeq 0.6$ for the AS-level map of the Internet [2], $c \simeq 0.4$ for the router-level map of the Internet [2], and $c \simeq 1.34, 0.65, 1.21, 5.10, 29.9$ for social networks Cyworld, Testimonial, Orkut, MySpace and Facebook, respectively [3,4]. Very recently, Pătraşcu and Roditty [5] obtained improvements on the Thorup-Zwick scheme for stretch 2 (see §II), but no other improvements are known for general sparse graphs.

This paper presents algorithms that, for sparse graphs, substantially break the classic space/stretch trade-off barrier, albeit at the cost of increased query time. Moreover, we demonstrate that our approach allows a surprisingly large fraction of source-destination pairs to retrieve *shortest* paths (99.98% in the AS-level map of the Internet), and argue that our approach could be easily implemented as a distributed routing scheme.

More specifically, we introduce two new ADQ schemes which respectively improve stretch and space in comparison with the data structure of Thorup and Zwick [1]. For weighted undirected graphs with n vertices and average degree Δ, we achieve the following constructions:

- For $0 < \varepsilon \leq 1/2$, a data structure of size $O(n^{2-\varepsilon}\Delta^{1-\varepsilon})$ that returns stretch 2 paths in $O((n\Delta)^{\varepsilon})$ time.
- For $0 < \varepsilon < 1$ and any positive integer k, a data structure of size $O((n\Delta)^{(1+1/k)(1-\varepsilon)})$ that returns stretch $(4k-1)$ paths in $O((n\Delta)^{\varepsilon})$ time.

For example, in dense graphs, obtaining stretch 3 requires space $O(n^{3/2})$ and constant query time [1]; allowing greater query time cannot help stretch or space. But when $\Delta = \Theta(\log n)$, special cases of our two results yield schemes for:

- stretch 2, space $\tilde{O}(n^{3/2})$, and query time $\tilde{O}(\sqrt{n})$; and
- stretch 3, space $\tilde{O}(n)$, and query time $\tilde{O}(\sqrt{n})$.

Other than the result of [5] at stretch 2, the above data structures are the first to improve the space/stretch trade-off of Thorup and Zwick [1] for sparse graphs. Our reduction in space is rather surprising: for $\varepsilon = 1/(k+1)$, we have a data structure that is *linear in the size of the graph*. In fact, for $\varepsilon > 1/(k+1)$, our scheme stores the graph itself plus some additional state that can actually be *sublinear* in the size of the graph.

These results have important practical applications. A line of work on *compact routing* [6] has applied ADQ techniques to routing in networks, where routers should require limited state yet forward packets along short paths. Recent work has shown how these compact routing tables can be constructed using distributed protocols [7]–[9], and as discussed above, the networks in which these protocols might be applied are sparse. We describe how our stretch-2 scheme can be implemented in a distributed way similar to [8] – with the addition of a surprisingly lightweight end-to-end exchange of less than 5 KB (at most 4 packets) and a small amount of processing in order to set up a new end-to-end connection.

[1]For $k = 4$ & $k \geq 6$, the lower bound [1] relies on a conjecture of Erdős.

We complement our theoretical results with extensive simulations on various graph topologies including the AS-level map of the Internet. Interestingly, we find that in the Internet AS-level topology, our stretch-2 scheme finds *shortest* paths for 99.98% of the source-destination pairs – compared with 34.4% using [1].

In summary, our results represent a step towards characterizing the space/stretch/time trade-off for approximate distance queries in sparse graphs, and yield a simple, practical way to improve stretch in compact routing protocols.

Roadmap. We start our discussion with related work in §II. The notations and definitions used in the paper are described in §III. We give an overview of our schemes in §IV. In §V, we prove that average-degree-bounded graphs are no harder than maximum-degree-bounded graphs (for precise definitions, see §III). This allows us to restrict our attention to maximum-degree-bounded graphs in the rest of the paper. We present our low-stretch and low-state schemes in §VI and §VII respectively. §VIII describes how to implement our stretch-2 scheme in a distributed environment. We present simulation results in §IX and conclude the paper with some open problems in §X.

II. Related Work

In this section, we discuss the known lower and upper bounds for the space/stretch trade-off in the approximate distance query problem for the regime of sparse graphs.

Lower Bounds. For general graphs, Thorup and Zwick [1] showed (subject to a conjecture of Erdős) that achieving (integer) stretch $(2k-1)$ requires $\Omega(kn^{1+1/k})$ space. Their proof is information-theoretic, essentially showing that for any constant stretch, there exist graphs that require storing as many bits as the number of edges in the graph. For example, proving that stretch 3 requires $\Omega(n^{3/2})$ space uses a graph with $\Theta(n^{3/2})$ edges. There is no hope of this proof technique being helpful in the sparse case; for example with $\Delta = \Theta(\log n)$, this technique will only show that achieving any stretch value (even exact shortest paths) requires $\Omega(n \log n)$ bits, which is entirely acceptable.

In fact, a data structure of $\tilde{\Theta}(n\Delta)$ bits *can* permit retrieval of shortest paths, simply by storing the original graph and running Dijkstra's algorithm for each query. Of course, this takes time $\tilde{O}(n\Delta)$ per query. Thus, in the context of designing ADQ data structures, the cases of dense and sparse graphs are quite different. In the dense case the key is to *compress* the graph while ensuring that sufficient information remains to return low-stretch distances. In the sparse case the graph need not be compressed, but the trade-off with *query time* becomes critical.

Very little is known about this trade-off space for sparse graphs. First, Sommer et al. [10] show that any data structure that returns stretch t paths in time α requires space $n^{1+\Omega(1/\alpha t)}$. For data structures with constant query time, this gives a lower bound of space $n^{1+\Omega(1/t)}$ for any stretch t. However, if we allow $\Omega(\log n)$ query time, their result implies a trivial lower bound of $\Omega(n \log n)$ for any constant stretch. Second, Pătraşcu and Roditty [5] prove that if a widely believed conjecture about the hardness of set intersection queries holds, then retrieving stretch 2 paths in constant time requires a data structure of size $\Omega(n\sqrt{n\Delta})$. For the case of $\Omega(\log n)$ query time, as in our schemes, no non-trivial lower bounds are known.

Upper Bounds. Independently and concurrently to our work, Pătraşcu and Roditty [5] obtained, for sparse graphs, the only known improvement over the general (dense graph) case. Their data structure returns stretch 2 paths in constant time and requires $O(\Delta^{1/3}n^{5/3})$ space (for general graphs, this would require $\Theta(n^2)$ space [11]). In comparison, a special case of our first data structure returns stretch 2 paths with space $O(\Delta^{1/2}n^{3/2})$ at the expense of $O(\sqrt{n\Delta})$ higher query time. For general sparse graphs, no other results are known; however, for specific classes of sparse graphs, improvements are possible in stretch ([12] achieves stretch 2 for Erdős-Renyi graphs) and in space ([13,14] for power-law graphs). For more discussion on results for specific classes of sparse graphs, see [15].

III. Notations and Definitions

Throughout the paper, we let $G = (V, E)$ be a connected, undirected graph with $n = |V|$ vertices and $m = |E|$ edges. Unless mentioned otherwise, G is assumed to be weighted with each edge assigned a non-negative weight.

For any vertex $v \in V$, we denote by $N(v)$ the set of all the neighbors of v. For any set $V' \subset V$, we denote by $N(V')$ the set of all the neighbors of vertices in V'. We let $\deg(v)$ denote the number of neighbors of vertex v, *i.e.*, $\deg(v) = |N(v)|$. The graph is said to be maximum-degree-bounded (or, Δ-degree bounded) if for all vertices $v \in V$, $\deg(v) \leq \Delta$. We say that the graph is average-degree-bounded graph (or, has average degree Δ) if $2m/n \leq \Delta$.

For any pair of vertices $u, v \in V$, let $d(u, v)$ be the length of the shortest path between u and v in G and let $\delta(u, v)$ be the length of the path retrieved using the data structure. The data structure is said to return stretch t paths if for every pair of vertices $u, v \in V$, $d(u, v) \leq \delta(u, v) \leq t \cdot d(u, v)$.

Let $L \subset V$ be a set of vertices. For any vertex $v \in V$, we denote by $\ell(v)$ the nearest neighbor of v in L (i.e., the vertex $a \in L$ that minimizes $d(v, a)$, ties broken arbitrarily). We define the **ball of** v, $B(v)$, as the set of all vertices $w \in V$ for which $d(v, w) \leq d(v, \ell(v))$ and the **vicinity** of v as $\Gamma(v) = B(v) \cup N(B(v))$.

IV. Overview of our schemes

Our data structure for stretch 2 is conceptually similar to the data structure of Thorup and Zwick [1]. For a given graph, they construct a set of vertices, known as *landmarks*, such that each vertex has a landmark vertex in its ball. The data structure, stores for each vertex, the shortest distance to each vertex in its ball and to its closest landmark; the landmarks store distances to all vertices in the graph. This allows the source to return exact distances to destinations in its ball and stretch 3 distances via its landmark for all other destinations.

Intuitively, the cases that attain worst-case stretch in their data structure are the ones when the destination is *close* to one of the farthest vertices in the ball of the source. For such source-destination pairs, we exploit the idea of *vicinity intersection*: we are able to show that there is some vertex in the graph that is in the *vicinity* of both the source and the destination. If the vicinities of the source and the destination do not have any *intersection*, they must be *far away*. In such a case, we show that a path via the landmark node returns a path with stretch 2. Indeed, we need to store the vicinities of the nodes for our data structure; by exploiting the properties of sparse graphs, we are able to show that this does not increase the space requirements significantly.

The above data structure is of large space since it stores (a) shortest paths from the landmark vertices to all other vertices in the graph; and (b) the vicinities of each and every vertex in the graph. To overcome the first requirement, our second data structure requires to store the exact distances between all pairs of landmarks. This requires significantly lesser space; for instance, in a graph with n vertices, storing shortest paths between every pair of $\sqrt{n}$ landmark vertices requires space at most linear in the size of the graph. To overcome the second requirement, our second data structure computes the vicinities *on the fly*; we show that for sparse graphs, this can be done in sublinear time. If the vicinities of the two vertices intersect, exact distance is returned. If not, a low stretch path can be retrieved by combining the distances from the vertices to their respective landmarks and the distance between the landmarks.

V. Average-degree Bounded Graphs are no harder than Maximum-degree Bounded Graphs

In this section, we show that in the context of designing ADQ data structures, average-degree-bounded graphs are no harder than maximum-degree-bounded graphs. In particular, assume that we have a data structure $\mathscr{D}$ that is of size $O(S)$ and returns stretch-s paths in $O(T)$ time for any Δ-*degree bounded* graph on n vertices, where S and T are functions of n, Δ and s. We show that $\mathscr{D}$ can be used to build a data structure of size $O(S)$ that returns stretch-s paths on a graph with *average degree* Δ in at most $O(T)$ time.

Let $G=(V,E)$ be a graph with average degree Δ. Given G, we will first create a Δ-degree bounded graph $G_\Delta = (V_\Delta, E_\Delta)$. Then, we show how $\mathscr{D}$ can be used on G_Δ to return stretch-s paths on G.

The Reduction. For each vertex $v \in V$, create $\alpha_v = \lceil \deg(v)/\Delta \rceil$ vertices $v_1, v_2, \ldots, v_{\alpha_v}$ in V_Δ. For each edge $e=(u,v) \in E$, if $\deg(u) \leq \Delta$ and $\deg(v) \leq \Delta$, create an edge $e=(u_1,v_1)$ in E_Δ. For each vertex $v \in V$, we arbitrarily distribute $N(v)$ in G to the vertices corresponding to v in G_Δ such that for $i=1,2,\ldots,(\alpha_v-1)$, $|N(v) \cap N(v_i)| = \Delta$ and $|N(v) \cap N(v_{\alpha_v})| = (\deg(v) - (\alpha_v - 1)\cdot\Delta)$. Finally, for each pair v_i, v_{i+1}, we create an edge in E_Δ of weight 0.

In order to answer an approximate distance query for any pair of vertices $u, v \in V$, we use $\mathscr{D}$ to answer approximate distance queries between $u_1, v_1 \in V_\Delta$ in G_Δ and let the length of the path returned by the data structure be δ'. We output the distance δ' as an approximate distance for the pair of vertices in G.

State and Query Time. We first prove that asymptotically, the size of the data structure and the query time are not increased due to the reduction (under the assumption that S and T are functions of $|V|$, Δ and s only). Since, for any fixed s, S and T depend only on the number of vertices in the graph and the upper bound on the degree of vertices, it suffices to prove the following claim:

Claim 5.1: G_Δ is a Δ-degree bounded graph with $O(n)$ vertices.

Proof: The degree boundedness is trivial from the construction. We prove the claim regarding number of vertices. Let $V_s \subset V$ be the set of vertices such that for all $v \in V_s$, $\deg(v) > \Delta$. The number of edges incident on the vertices in V_s is at most $O(n\Delta)$ (the size of edge set E in G). For any Δ of these edges, we have created at most 1 extra vertex in V_Δ. Hence, the total number of vertices that we could have possibly created in V_Δ is $O(n)$. ■

Stretch. Consider any pair of vertices $u, v \in V$ at distance d in G. It is trivial that in G_Δ, the distance between the vertices $u_1, v_1 \in V_\Delta$ is $d' = d$. The data structure $\mathscr{D}$ returns a path of distance at most $\delta' = s \times d' = s \times d$, which is of stretch s.

In §VI, we discuss how this reduction can be intuitively interpreted when it is incorporated into the algorithm of the next section, so that the algorithm runs directly on G rather than G_Δ. In the rest of the paper, we restrict our attention to Δ-degree bounded graphs only.

VI. ReAR: Reduced Approximation Ratio approximate distance queries

In this section, we construct a data structure that, for any $0 < \varepsilon \leq 1/2$, is of size $O(n^{2-\varepsilon}\Delta^{1-\varepsilon})$ and returns stretch-2 paths for any Δ-degree bounded graph in at $O((n\Delta)^\varepsilon)$ time. The discussion in §V then immediately implies a data structure of size $O(n^{2-\varepsilon}\Delta^{1-\varepsilon})$ that returns stretch-2 paths in $O((n\Delta)^\varepsilon)$ time for all graphs with $O(n\Delta)$ edges.

A. Constructing the data structure

Let $G=(V,E)$ be a Δ-degree bounded graph. Fix some $0 < \epsilon \leq 1/2$. Our construction begins by sampling each vertex independently at random with probability $n^{-\varepsilon}\Delta^{1-\varepsilon}$, creating a set L of sampled vertices. Our data structure stores, for each vertex in L, a hash table containing shortest distance to every other vertex in G. Furthermore, the data structure stores for each vertex $v \in V \setminus L$:

- $\ell(v)$ and the "ball radius" r_v defined as $r_v = d(v, \ell(v))$.
- a hash table holding shortest distance to each vertex in $\Gamma(v) = B(v) \cup N(B(v))$.

Size of the data structure. Note that $E[|L|] = n^{1-\varepsilon}\Delta^{1-\varepsilon}$, and hence, storing shortest distances from vertices in L to all vertices in the graph requires $O(n^{2-\varepsilon}\Delta^{1-\varepsilon})$ space. Moreover, using a standard argument as in [1, Lemma 3.2], we get that $E[|B(v)|] = O(n^\varepsilon \Delta^{-(1-\varepsilon)})$. Since the graph is

Δ-degree bounded, $|N(B(v))| = O(n^{\varepsilon}\Delta^{\varepsilon})$ and hence, the data structure requires $O(n^{1+\varepsilon}\Delta^{\varepsilon})$ space corresponding to each vertex in $V\backslash L$ (storing shortest distance to $\ell(v)$ and r_v requires $O(1)$ space for each vertex in $V\backslash L$). Hence, the total size of the data structure is $O(n^{2-\varepsilon}\Delta^{1-\varepsilon})$.

As in [1,5], our construction of ADQ data structure is randomized. The space is bounded in expectation while the query algorithm below is deterministic with no errors. Given this guarantee, a data structure that gives the same stretch guarantee and has the same size *with high probability* can be constructed using a Las Vegas algorithm.

B. Answering distance queries

The query algorithm takes the two vertices u and v whose distance is to be estimated. If $v \in \Gamma(u)$ or $u \in \Gamma(v)$, the algorithm directly reads the distance between u and v from the hash table maintained at u or v respectively. If $v \notin \Gamma(u)$ and $u \notin \Gamma(v)$, the algorithm performs a *vicinity intersection check*: it queries each of the vertices $w \in \Gamma(u)$ and checks if $w \in \Gamma(v)$. If there is no such vertex w, the algorithm queries u and v for their vicinity radii r_u and r_v. If $r_u < r_v$, the algorithm returns $d(u,\ell(u)) + d(\ell(u),v)$ else it returns $d(v,\ell(v)) + d(\ell(v),u)$. The algorithm QUERY-2(u, v) for distance query between vertices u and v is shown in Fig. 1.

Analysis of the query answering algorithm. We obtain an upper bound of 2 on the stretch of the distance between the vertices returned by QUERY-2(u,v). In order to complete the proof, we will use the following lemma.

Lemma 6.1 (Vicinity Intersection Lemma): If $d(u,v) < r_u + r_v$, there exists a vertex $x \in V$ such that $x \in \Gamma(u) \cap \Gamma(v)$.

Proof. The lemma is trivially true if $d(u,v) < r_u$. So, consider the case when $r_u \le d(u,v) < r_u + r_v$. Let $P = (u, v_1, v_2, \ldots, v_k, v)$ denote the shortest path between u and v in G. Note that if $k < 2$, v_1 is a neighbor of both u and v, making the lemma true. So, consider the cases when $k \ge 2$.

Let $i_1 = \max\{i : v_i \in B(u)\}$ and $i_2 = \min\{j : v_j \in B(v)\}$. If $i_2 \le i_1 + 1$, we are done since $\Gamma(u) = B(u) \cup N(B(u))$. Consider the case when $i_2 > i_1 + 1$ and let $i_1 < i' < i_2$ be some index between i_1 and i_2. Note that $v_{i'}$ is neither in $B(u)$, nor in $B(v)$. Hence, we have that $d(u, v_{i'}) \ge r_u$ and $d(v, v_{i'}) \ge r_v$, which implies $d(u,v) = d(u,v_{i'}) + d(v_{i'},v) \ge r_u + r_v$, contradicting the assumption of the lemma. □

Claim 6.2: If $d(u,v) < r_u + r_v$, the algorithm QUERY-2(u,v) returns exact distance between u and v in $O(n^{\varepsilon}\Delta^{\varepsilon})$ time.

Proof: Let u and v be two vertices for which the distance query has to be performed. If $d(u,v) < r_u + r_v$, then, by Lemma 6.1 we have that there must be at least one vertex $x \in \Gamma(u)$, such that $x \in \Gamma(u) \cap \Gamma(v)$. The algorithm reads the "exact" distance $d(u,x)$ from the hash-table maintained at vertex u and the "exact" distance $d(v,x)$ from the hash-table maintained at vertex v. From the proof of vicinity intersection lemma, we note that among all such vertices $x \in \Gamma(u) \cap \Gamma(v)$, there must be at least one vertex which lies on the shortest path between u and v, and this vertex minimizes the "min-d" variable returned by the algorithm resulting in a distance estimate of stretch 1. Since this requires querying all the vertices in $\Gamma(u)$ (or $\Gamma(v)$), the query time is $O(n^{\varepsilon}\Delta^{\varepsilon})$. ■

For the case when $d(u,v) \ge r_u + r_v$, we show that our scheme results in a stretch at most 2.

Theorem 6.3: If $d(u,v) \ge r_u + r_v$, the algorithm QUERY-2(u,v) returns, in the worst case, a distance estimate of stretch-2 between u and v in $O(n^{\varepsilon}\Delta^{\varepsilon})$ time.

Proof: When $d(u,v) \ge r_u + r_v$, the distance estimate returned by the scheme is $\delta(u,v) = d(u,\ell(u)) + d(\ell(u),v)$. By the triangle inequality, $d(\ell(u),v) \le d(\ell(u),u) + d(u,v)$. Hence, $\delta(u,v) \le 2 \cdot d(u,\ell(u)) + d(u,v)$. Since $d(u,\ell(u)) = r_u$, we get $\delta(u,v) \le 2 \cdot r_u + d(u,v)$. Similarly, we can prove that $\delta(u,v) \le 2 \cdot r_v + d(u,v)$. Hence, we get that $\delta(u,v) \le 2 \cdot \min\{r_u, r_v\} + d(u,v)$.

Without loss of generality, assume that $r_u \le r_v$. Then, the condition in the lemma implies that $d(u,v) \ge 2 \cdot r_u$. The above discussion implies that the distance estimate returned is at most $2 \cdot r_u + d(u,v)$ giving a stretch of at most $2 \cdot d(u,v)/d(u,v) \le 2$. ■

C. Discussion

Note that the above results also imply a data structure for unweighted graphs that returns stretch-2 paths in subquadratic space at the expense of higher query time. In fact, if unweighted graphs were the main concern, a simple modification of our scheme gives us a data structure that, for any $0 < \varepsilon \le 1/2$ is of size $O(n^{2-\varepsilon})$ and given any two vertices u and v at distance d returns a distance of at most $(2d+1)$ in $O(n^{\varepsilon})$ time. The only known $o(n^2)$ data structure for such approximation ratio is again due to the recent result of Pătraşcu and Roditty [5] that requires $O(n^{5/3})$ space and takes constant query time. See [15] for detailed discussion.

Implications of the average-to-max-degree-bound reduction. The results in this section combined with the reduction of §V immediately give us a data structure of size $O(n^{2-\varepsilon}\Delta^{1-\varepsilon})$, which for any graph with at most $O(n\Delta)$ edges, returns stretch-2 paths in $O(n^{\varepsilon}\Delta^{\varepsilon})$ time. However, we can actually incorporate the reduction into the algorithm in a simple way that yields intuition and eases implementation.

Specifically, let G be the graph with average degree Δ. The reduction implies that each vertex v in G which has degree $\deg(v) > \Delta$ effectively "emulates" $\lceil \deg(v)/\Delta \rceil$ vertices in G_{Δ}. Now consider constructing the data structure presented in this section. While sampling vertices for the generating set L, the vertex v is now sampled with probability $n^{-\varepsilon}\Delta^{1-\varepsilon} \times \lceil \deg(v)/\Delta \rceil$, *i.e.*, with probability that is proportional to the degree of v. Moreover, due to Claim 5.1, the size of $B(v)$ remains unchanged asymptotically ($B(v)$ and hence $\Gamma(v)$ may change, but not their size). Thus, the implications of the reduction on the original graph are simple: just sample each vertex v in the graph with probability $n^{-\varepsilon}\Delta^{1-\varepsilon} \times \lceil \deg(v)/\Delta \rceil$ rather than probability $n^{-\varepsilon}\Delta^{1-\varepsilon}$. *In other words, rather than sampling vertices uniform-randomly, they are sampled with probability proportional to their degree.*

```
QUERY-2(u, v):

If v ∈ Γ(u)
   return d(u, v)
Else if u ∈ Γ(v)
   return d(v, u)
Else if VICINITY-INTERSECTION(u, v) < ∞
   return VICINITY-INTERSECTION(u, v)
Else if (r_u < r_v)
   return d(u, ℓ(u)) + d(ℓ(u), v)
Else
   return d(v, ℓ(v)) + d(ℓ(v), u)
```

```
VICINITY-INTERSECTION(u, v):

min-d ← ∞
For each w ∈ Γ(u)
   If d(u, w) + d(v, w) < min-d
        min-d ← d(u, w) + d(v, w)
return min-d
```

Fig. 1. Answering approximate distance queries with stretch-2. $d(x, y)$ is the exact distance from x to y, stored in a hash table at x.

An optimization. Although the worst-case stretch for REAR is 2, we can apply simple heuristics to improve the stretch in practice. In particular, we can further exploit the vicinity intersection process in our query algorithm. In QUERY-2, when the source u queries each of the vertices $w \in \Gamma(u)$, it could actually query the distance from w to the destination v. Note that each $w \in \Gamma(u)$ has a finite distance path to the destination through $\ell(w)$ that could be retrieved in constant time. The approximate distance query can then be answered by the data structure as the minimum of all the distances retrieved by querying the vertices in the vicinity of the source. This heuristic may result in improved stretch in practice without any asymptotic increase in the query time (see §VIII for implementation details). Indeed, we show in §IX that this heuristic increases the number of source-destination pairs that retrieve shortest paths by almost 25%.

VII. ReS: Reduced Space approximate distance queries

In this section, we give a data structure that, for any $0 < \varepsilon < 1$, is of size $O((n\Delta)^{(1+1/k)(1-\varepsilon)})$ and returns stretch-$(4k-1)$ paths in $O(n^{\varepsilon}\Delta^{\varepsilon})$ time for any graph with $O(n\Delta)$ edges.

A. Constructing the data structure

Let $G = (V, E)$ be a Δ-degree bounded graph. Fix some $0 < \varepsilon < 1$ and some integer $k > 0$. Our construction of data structure $\mathscr{D}$ begins by sampling each vertex independently at random with probability $n^{-\varepsilon}\Delta^{1-\varepsilon}$, creating a set L of sampled vertices. We now create a complete graph G' with vertices in L as the vertex set and for each pair $l_1, l_2 \in L$, the weight of the edge (l_1, l_2) being the shortest path between l_1 and l_2 in G. We run the approximate distance oracle from Thorup and Zwick on G' to construct a data structure $\mathscr{D}'$ that stores $(2k-1)$-approximate shortest paths between every pair of vertices in L. Furthermore, the data structure stores, for each node $v \in V$, the nearest neighbor (the node $a \in L$ that minimizes $d(v, a)$, ties broken arbitrarily) of v in L and the shortest distance to $\ell(v)$. $\mathscr{D}$ stores $\mathscr{D}'$ as a sub-data structure. Furthermore, $\mathscr{D}$ also stores the entire graph.

Size of the data structure. Note that $E[|L|] = (n\Delta)^{1-\varepsilon}$ and hence, the size of the data structure $\mathscr{D}'$ is $O((n\Delta)^{(1+1/k)(1-\varepsilon)})$. For $\epsilon < 1/(k+1)$, the size of $\mathscr{D}'$ asymptotically dominates the size of the graph and hence, the size of the data structure $\mathscr{D}$ is $O((n\Delta)^{(1+1/k)(1-\varepsilon)})$. For $\varepsilon > 1/(k+1)$, the size of the graph dominates the size of $\mathscr{D}$, resulting in size linear in the size of the graph.

B. Answering distance queries

For any node $v \in V$, we denote by r_v the "ball radius" $d(v, \ell(v))$. Let QUERYTZ(u, v) be the query algorithm for the Thorup-Zwick scheme that returns $(2k-1)$-approximate distances between nodes u and v.

Suppose we perform an approximate distance query for a pair of nodes $u, v \in V$. The two nodes u and v run any shortest-path algorithm that stops when u (respectively, v) has computed $\Gamma(u)$ (respectively $\Gamma(v)$) and shortest distances to nodes in $\Gamma(u)$ (respectively $\Gamma(v)$). This can be done since the graph is stored in the data structure. Both u and v temporarily store this information in a hash table. If $v \in \Gamma(u)$ or $u \in \Gamma(v)$, the algorithm directly reads the distance between u and v from the hash table. If none of the nodes has the other node in its hash table, the algorithm checks for "vicinity intersection", *i.e.*, for each node $x \in \Gamma(u)$, the algorithm checks if $x \in \Gamma(v)$. If such a node is found, the algorithm returns the sum of the shortest distances from u to x and from v to x. If no such x exists, the algorithm returns $d(u, \ell(u)) + \text{QUERYTZ}(u, v) + d(v, \ell(v))$. Finally, the hash tables are deleted from nodes u and v. The algorithm QUERY(u, v) for distance query between nodes u and v is shown in Fig. 2.

Analysis of the query answering algorithm. In terms of query time, we note that running the shortest-path algorithm requires at most $O(n^{\varepsilon}\Delta^{\varepsilon})$ time for a Δ-degree bounded graph. For stretch analysis, similar to Claim 6.2, one can prove that if $d(u, v) < r_u + r_v$, the algorithm QUERY(u, v) returns exact shortest distances in $O(n^{\varepsilon}\Delta^{\varepsilon})$ time. For the case when $d(u, v) \geq r_u + r_v$, we show that our scheme results in a stretch at most $(4k-1)$.

Theorem 7.1: If $d(u, v) \geq r_u + r_v$, the algorithm QUERY(u, v) returns, in the worst case, distance estimate of stretch-$(4k-1)$ between u and v in $O(n^{\varepsilon}\Delta^{\varepsilon})$ time.

Proof: When $d(u, v) \geq r_u + r_v$, the distance estimate returned by the scheme is $\delta(u, v) = d(u, \ell(u)) +$

```
QUERY(u, v):

Compute Γ(u)
Compute Γ(v)
If v ∈ Γ(u)
   return d(u, v)
Else if u ∈ Γ(v)
   return d(v, u)
Else if VICINITY-INTERSECTION(u, v) < ∞
   return VICINITY-INTERSECTION(u, v)
Else
   return d(u, ℓ(u)) + QUERYTZ(u, v) + d(ℓ(v), v)
```

Fig. 2. Answering approximate distance queries with stretch-3. $d(x, y)$ is the exact distance from x to y stored in a hash table at x. The VICINITY-INTERSECTION(u, v) algorithm is same as shown in Fig. 1. QUERYTZ(u, v) is the query algorithm for the Thorup-Zwick scheme that, for any integer $k > 0$, returns $(2k-1)$-approximate distances between nodes u and v.

QUERYTZ$(u, v) + d(\ell(v), v)$. Since QUERYTZ(u, v) returns $(2k-1)$-approximate distances, we have that $\delta(u, v) \leq d(u, \ell(u)) + (2k-1)d(\ell(u), \ell(v)) + d(\ell(v), v)$. By the triangle inequality, $d(\ell(u), \ell(v)) \leq d(\ell(u), u) + d(u, v) + d(v, \ell(v))$. Hence, $\delta(u, v) \leq 2k \cdot d(u, \ell(u)) + (2k-1)d(u, v) + 2k \cdot d(v, \ell(v))$. Since $d(u, \ell(u)) = r_u$ and $d(u, \ell(v)) = r_v$, we get $\delta(u, v) \leq 2k \cdot r_u + (2k-1)d(u, v) + 2k \cdot r_v = 2k(r_u + r_v) + (2k-1)d(u, v)$. Using the condition of the theorem, we get $\delta(u, v) \leq 2k \cdot d(u, v) + (2k-1)d(u, v) = (4k-1) \cdot d(u, v)$, which we set out to prove. ■

C. Discussion

Perhaps the most interesting result that follows from the above construction is for $\varepsilon = 1/(k+1)$. This gives us a data structure that is of size linear in the size of the graph and answers $(4k-1)$-approximate distance queries in time $O((n\Delta)^{1/(k+1)})$ for all graphs with $O(n\Delta)$ edges; no such construction was previously known. Moreover, the above data structure can be easily modified to get another data structure that returns stretch $(4k-1)$ paths in $O(n^{\varepsilon}\Delta)$ time and has size $O(n^{(1+1/k)(1-\varepsilon)})$: independent of the density of the graph.

The optimization. An optimization similar to that in §VI is again possible for the algorithm described in this section. During the vicinity check, when the source u queries each of the vertices $w \in \Gamma(u)$, it could actually query for its distance to $\ell(v)$ and combine this with $d(u, w)$ and $d(v, \ell(v))$ to retrieve the distance from the source to the destination vertex. Again, the approximate distance query can then be answered by the data structure as the minimum of all the distances retrieved by querying the vertices in the vicinity of the source resulting in improved stretch in practice without any asymptotic increase in the query time.

VIII. APPLICATION TO NETWORK ROUTING

Work on compact routing has applied the traditional results from approximate distance query problem [1] to network routing problems in order to find short paths while using little memory at routers. These solutions have been proposed as centralized algorithms [6] and more recently as distributed protocols for wireless sensor networks [7], the Internet [8] and peer-to-peer networks [9]. In this section, we discuss a surprisingly lightweight scheme that can be incorporated in distributed routing protocol implementations of the Thorup-Zwick (TZ) scheme, [8] for instance, to get a distributed routing protocol for our stretch-2 data structure.

TZ scheme and REAR**.** REAR can be incorporated into the the proposed distributed adaptations [7]–[9] of the TZ scheme with minimal changes. This is due to the fact that the construction in REAR, in concept, is similar to the TZ scheme: both schemes construct a set L of vertices and each vertex v stores a corresponding nearest neighbor $\ell(v)$ and certain vertices in its neighborhood. The first difference between REAR and the TZ scheme is that the set L is sampled proportional to node degree rather than uniform-randomly. Second, REAR differs from TZ in terms of the information stored in the data structure: for any vertex v, while TZ only requires storing the ball $B(v)$, REAR stores $\Gamma(v)$. Both modifications are easy changes to the distributed protocols of [7]–[9]. Third, to route from s to d, REAR allows s to set up an initial connection to d by using the TZ algorithm. This initial connection gives a path of stretch 3, via an essentially unmodified proof of [1,6]. The final task is to improve the stretch from 3 to 2.

Implementing vicinity intersection. REAR requires the source and the destination to perform a vicinity intersection in order to guarantee a path with stretch 2 (see Lemma 6.1). We show how vicinity intersection can be implemented in practice with a surprisingly lightweight exchange of very few packets. Recall, from the discussion above, that the initial connection gives the source a path to the destination with stretch 3. The source can then send the list of vertices in its vicinity to the destination using this path. For the router-level map of the Internet measured by CAIDA [2], which consists of $n = 192,244$ routers and has average degree $\Delta \simeq 0.4 \log_2 n$, this requires the source to transfer approximately 4.64 kilobytes of data ($4 \cdot \sqrt{n\Delta}$ bytes, since IPv4 addresses are 4 bytes). On today's Internet, packets are generally allowed to be at least 1500 bytes long, so this would take just four packets; with jumbo frames [16], it would take just one. The destination can then perform a vicinity intersection, which requires $\tilde{O}(\sqrt{n\Delta})$ time asymptotically but using the above numbers requires less than 1161 hash table lookups which is fast in practice.[2] Upon executing the vicinity intersection, the destination informs the source whether the vicinities intersect or not. If they do, it can inform the source of the vertex (or vertices) at which vicinities intersect. This requires at most one packet which can be routed from the destination through the source via a stretch-3 path. The source-destination pair, after sending five or fewer packets, now have a route with stretch 2.

[2]If the destination is a server, this could be a burden; but note that we could just as easily flip the protocol around so the source does the computation.

In practice, this is likely to be efficient even for relatively short-lived connections. For much larger networks, of course, the exchange of vicinity information would require more bandwidth and computation; but since a stretch-3 path is available immediately, the reduction to stretch 2 can be treated as an optimization for longer flows in order to amortize the overhead.

Probing and Shortcutting. The protocol for implementing vicinity intersection discussed above does not exploit the optimization discussed in §VI for heuristically improving the stretch for the retrieved paths. We discuss the implementation aspects related to the optimization. Implementing the optimization in practice leads to a process, which we call *probing and shortcutting* (P&S). P&S requires the source vertex to *probe* the vertices in its vicinity for improving stretch. We argue that this can be achieved with an extremely low overhead probing scheme. Once the source vertex finds a vertex in its vicinity that provides a better stretch, the source can conveniently switch the traffic through the *shortcut* path. We only discuss the probing mechanism, since shortcutting can be implemented easily in practice (note that the destination is oblivious to the shortcutting mechanism and hence, P&S does not require any handshaking mechanism).

For the probing mechanism, assume that the source opens an initial connection to a destination. The source, every 10^{th} packet, can probe a vertex in its vicinity (the question on deciding an appropriate order of probing the vertices in vicinity is discussed below) requesting the length of the route available from this vertex to the destination. These packets can be extremely small compared to the other data packets, leading to an extremely small overhead in terms of bandwidth consumed (just a fraction 0.1 more packets that are of negligible size compared to the data packets). Since the source-destination connections that account for most of the bandwidth sent on the networks are very long [17,18], we believe it is reasonable to amortize the cost of the probing over the lifetime of the connection.

In terms of the order of probing, we consider two heuristics. *Farthest-first*, in which the source probes the vertices that are the *boundary* vertices of its vicinity; and, *closest-first*, in which the source performs probing starting with the closest vertices (its neighbors). We show, through evaluations, that the former performs better than the latter.

IX. Evaluation Results

In this section, we evaluate the performance of ReAR and ReS schemes on large-scale synthetic and realistic topologies. We first present our methodology, followed by a summary of the evaluation results and conclude with a detailed discussion on the results.

A. Methodology

Schemes. We evaluate three schemes: the stretch-3 TZ scheme, ReAR: the stretch-2 scheme from §VI with $\varepsilon = 1/2$, and ReS: the stretch-3 scheme (for $k = 1$) from §VII with $\varepsilon = 1/2$. Furthermore, we evaluate ReAR and ReS schemes with and without the P&S optimization discussed in earlier sections. For the TZ scheme, we sampled each vertex (for set L) with probability $\sqrt{\log n/n}$. For ReAR and ReS, each vertex was sampled with probability $\sqrt{n \log n} \times \deg(v)/\log^2 n$. All the constants in the big-O notation were set to be 1.

Simulator. We wrote a static simulator to simulate the above schemes. Hence, from the perspective of application to distributed compact routing protocols, the results presented in this section assume a static network topology and give post-convergence results only. As outlined in §VIII, a distributed implementation of our stretch-2 scheme is a straightforward extension of past work, but we leave a full dynamic evaluation to future work. Our static simulator allows us to evaluate the schemes at much larger scale.

Topologies. We present evaluation results for three topologies. (1) $G(n,m)$ random graphs, *i.e.*, $n = 16384$ nodes with m uniform-random edges, with m set so that the average degree is 6, (2) geometric random graphs with $n = 16384$ nodes with average degree 6, and (3) a 33,014 node AS-level map of the Internet (referred to as the Internet graph in this section) [2].

For $G(n,m)$ graphs and the Internet graph, link weights are 1; for geometric random graphs, a link's weight is the Euclidean distance between the position of its two vertices. For $G(n,m)$ graphs and for geometric random graphs, we generated 10 different topologies with the same parameters and our results are the average of evaluations of these topologies. For geometric random graphs, we sampled a set of "source" vertices and evaluated the performance of the schemes from these sources to all the destinations. We found [15] that sampling 1/4 of the nodes as sources provided accurate results.

B. Results and Discussions

Stretch comparison with the TZ-scheme. Fig. 3 shows the performance of the three schemes for various graph topologies (TZ is the original TZ scheme, TZ-d scheme is discussed below). The most notable result of this evaluation is that ReAR allows retrieval of exact shortest paths for nearly all source-destination pairs: 98.94% in the $G(n,m)$ graph, and 99.98% in the Internet graph. Though $G(n,m)$ graphs and the Internet graph have highly different structures, these graphs have a common feature: for nearly all source-destination pairs, the two vicinities intersect, thus providing a shortest path. In the $G(n,m)$ graph (in which 96.2% source-destination pairs have intersecting vicinities), this occurs since, with high probability, the diameter of the graph is roughly at most twice the vicinity radius. In the Internet graph (in which 96.8% source-destination pairs have intersecting vicinities), vicinity intersection likely occurs at the "core" networks of the Internet. Since TZ scheme does not exploit the vicinity intersection, its performance is significantly worse than our schemes (only 34.4% of the source-destination pairs retrieved shortest paths).

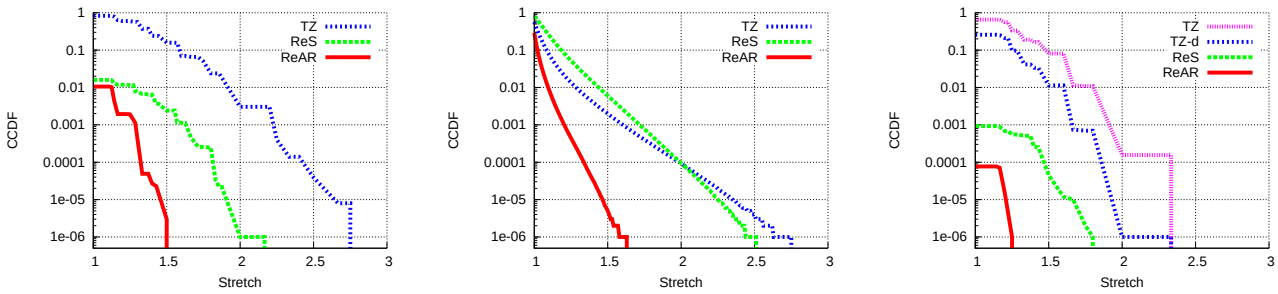

Fig. 3. Complementary CDF of Stretch in $G(n,m)$ random graph (left), geometric random graph (middle) and Internet graph (right).

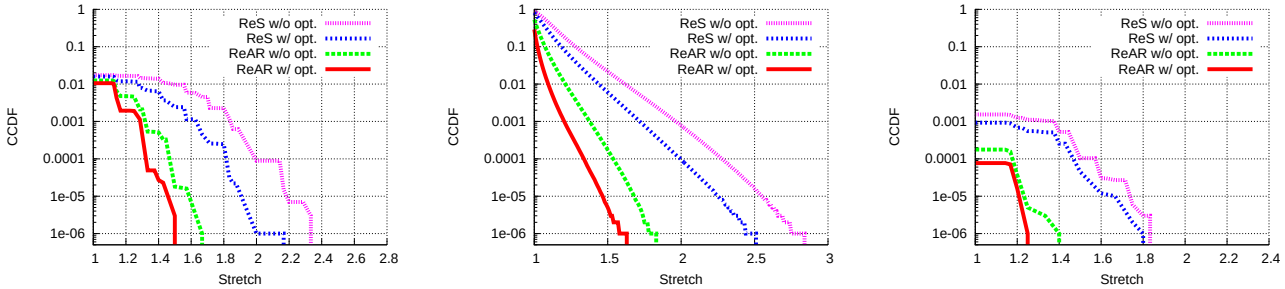

Fig. 4. Complementary CDF of Stretch (REAR and RES) in $G(n,m)$ random graph (left), geometric random graph (middle) and Internet graph (right).

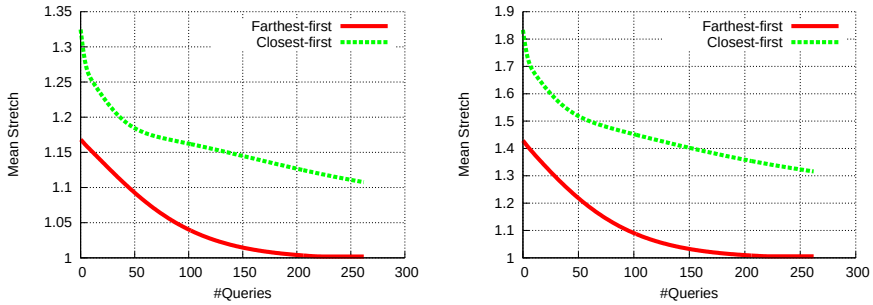

Fig. 5. Mean stretch versus query time for REAR (left) and RES (right) for 16,384 node $G(n,m)$ graph with average degree 6.

Another explanation to this surprising difference between the two schemes may be due to the set L – our scheme requires generating set L considering vertex degrees, while TZ scheme simply uniformly-randomly selects vertices for set L. We, hence, evaluated a modified version of the TZ scheme that uses the same set L as used by our schemes (see TZ-d in Fig. 3). Although this improves the performance of the TZ scheme (74.2% of the source-destination pairs now retrieve shortest paths), it is still much worse than the REAR and RES schemes. We, hence, believe that the high performance of our schemes is indeed due to the vicinity intersection idea.

For geometric random graphs, REAR allows retrieval of shortest paths only for 70.7% of the source-destination pairs in comparison to 42.9% for the TZ scheme; indeed, only 4.8% of the source-destination pairs have intersecting vicinities. However, REAR consistently performs better than the TZ-scheme, which in turn performs better than RES. Finally, while the TZ-scheme performs better than RES on an average, the worst-case stretch for the TZ-scheme is consistently much worse than RES. We believe that this is due to the optimization P&S, where many source-destination pairs could significantly improve their stretch due to short-cutting.

Stretch comparison of REAR and RES. We now compare REAR and RES specifically; the results for various graph topologies are shown in Fig. 4. We note that, as expected, REAR consistently performs better than RES, even without the P&S optimization. However, the more interesting observation is that the P&S optimization is much more effective in RES. In particular, we note that the tail of RES without the P&S optimization is significantly reduced when the optimization is used.

For $G(n,m)$ graphs, the stretch for 99% of the source-destination pairs is less than 1.15 using REAR. For RES, this is almost 1.3 (optimized version) and 1.5 (unoptimized version).

The case of geometric random graphs is rather interesting: first, we observe that not many source-destination pairs have intersecting vicinities, otherwise RES without the P&S optimization would not have achieved such a low fraction

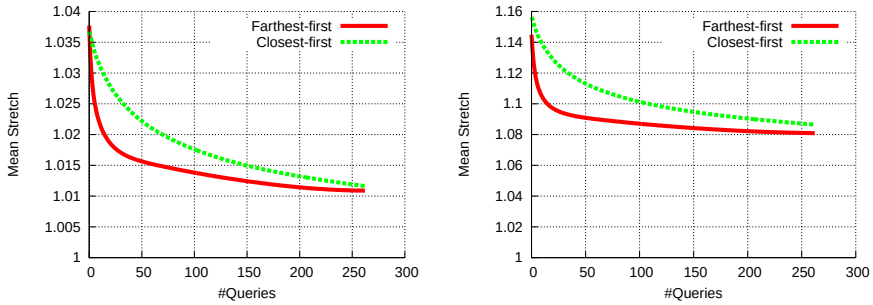

Fig. 6. Mean stretch versus query time for REAR (left) and RES (right) for 16,384 node geometric random graph with average degree 6.

of source-destination pairs retrieving shortest paths (only around 11%). Despite this, REAR performs surprisingly well: almost 48% of the source-destination pairs retrieve shortest paths without the P&S optimization and almost 71% retrieve shortest paths with the P&S optimization.

Stretch versus Query Time. For $G(n,m)$ graphs, Fig. 5 shows the variation of *mean stretch* – averaged over all source-destination pairs – with the number of queries for REAR and RES schemes, for the farthest-first and closest-first heuristics discussed in §VIII. We see a clear trend of "diminishing returns" where a few initial queries significantly reduce the stretch compared to no queries, after which the improvement is minimal. Based on the results, we conclude that in general, the farthest-first heuristic performs better in terms of the stretch with smaller query time. For the same two heuristics for stretch versus query time, Fig. 6 shows the results for REAR and RES schemes for the geometric random graph; we note that it is significantly better to start querying with the farthest nodes in the vicinity. Since the vicinities of most source-destination pairs intersect (and if they intersect, they do at least at one of the farthest nodes), queries starting from the farthest nodes achieved an improved stretch (quickly!). In terms of stretch versus query time, the results for the Internet graph were very similar to that of $G(n,m)$ graphs.

X. CONCLUSIONS

This paper presented algorithms which significantly improve the space and stretch of approximate distance query schemes in the realistic case of sparse graphs, and argued that our increased query time is reasonable in practice. Allowing increased query time to improve the space/stretch trade-off brings up several interesting open problems:

- Can the query time of our schemes be reduced? In other words, can one design a data structure of size $O(n\sqrt{n\Delta})$ that returns stretch-2 paths in $o(\sqrt{n\Delta})$ time?
- We sketched a distributed implementation of our REAR scheme, but not RES. While it seems significantly more challenging, a distributed version of RES could have significant implications in practice: One could achieve stretch 3 with constant amount of storage at nodes in the network.
- The most intriguing problem is to compute lower bounds for data structures that take $\Omega(\log n)$ query time and return constant stretch paths. The holy grail of the approximate distance query problem for sparse graphs is whether one can design a data structure of size $O(m)$ that yields constant stretch paths in $O(\text{polylog}(n))$ time. This would be a very significant result.

ACKNOWLEDGMENTS

The authors would like to thank Chandra Chekuri and Matthew Caesar for helpful discussions. This work was supported by National Science Foundation grants CNS 10-17069 and CCF-0915984.

REFERENCES

[1] M. Thorup and U. Zwick, "Approximate distance oracles," *Journal of the ACM*, vol. 52, no. 1, pp. 1–24, January 2005.
[2] The Cooperative Association for Internet Data Analysis, 2009. [Online]. Available: http://www.caida.org/home/
[3] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong, "Analysis of topological characteristics of huge online social networking services," in *Proc. ACM WWW*, May 2007.
[4] http://www.facebook.com/press/info.php?statistics.
[5] M. Pătraşcu and L. Roditty, "Distance oracles beyond the Thorup-Zwick bound," in *Proc. IEEE FOCS*, October 2010.
[6] M. Thorup and U. Zwick, "Compact routing schemes," in *Proc. ACM SPAA*, July 2001.
[7] Y. Mao, F. Wang, L. Qiu, S. Lam, and J. Smith, "S4: Small state and small stretch routing protocol for large wireless sensor networks," in *Proc. USENIX NSDI*, April 2007.
[8] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy, "Scalable routing on flat names," in *Proc. ACM CoNEXT*, December 2010.
[9] B. A. Ford, *UIA: A Global Connectivity Architecture for Mobile Personal Devices*. Massachusetts Institute of Technology: PhD Thesis, September, 2008.
[10] C. Sommer, E. Verbin, and W. Yu, "Distance oracles for sparse graphs," in *Proc. IEEE FOCS*, October 2009.
[11] C. Gavoille and M. Gengler, "Space efficiency for routing schemes of stretch factor three," *Journal of Parallel and Distributed Computing*, vol. 61, no. 5, pp. 679–687, May 2001.
[12] M. Enachescu, M. Wang, and A. Goel, "Reducing maximum stretch in compact routing," in *Proc. IEEE INFOCOM*, April 2008.
[13] D. Krioukov, K. Fall, and X. Yang, "Compact routing on internet like graphs," in *Proc. IEEE INFOCOM*, March 2004.
[14] W. Chen, C. Sommer, S. Teng, and Y. Wang, "Compact routing in power-law graphs," in *Proc. DISC*, September 2009.
[15] R. Agarwal, P. B. Godfrey, and S. Har-Peled, "Approximate distance queries in sparse graphs," University of Illinois at Urbana-Champaign, Tech. Rep., April 2010.
[16] W. http://en.wikipedia.org/wiki/Jumboframe.
[17] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying elephant flows through periodically sampled packets," in *Proc. ACM IMC*, October 2004.
[18] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of internet flow rates," in *Proc. ACM SIGCOMM*, August 2002.