All-Pairs Approximate Shortest Paths and Distance Oracle Preprocessing

Christian Sommer

Apple Inc., Cupertino, CA, USA
csommer@apple.com

Abstract

Given an undirected, unweighted graph G on n nodes, there is an $O(n^2 \text{poly} \log n)$-time algorithm that computes a data structure called *distance oracle* of size $O(n^{5/3} \text{poly} \log n)$ answering approximate distance queries in constant time. For nodes at distance d the distance estimate is between d and $2d + 1$.

This new distance oracle improves upon the oracles of Pătraşcu and Roditty (FOCS 2010), Abraham and Gavoille (DISC 2011), and Agarwal and Brighten Godfrey (PODC 2013) in terms of preprocessing time, and upon the oracle of Baswana and Sen (SODA 2004) in terms of stretch. The running time analysis is tight (up to logarithmic factors) due to a recent lower bound of Abboud and Bodwin (STOC 2016).

Techniques include dominating sets, sampling, balls, and spanners, and the main contribution lies in the way these techniques are combined. Perhaps the most interesting aspect from a technical point of view is the application of a spanner without incurring its constant additive stretch penalty.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases graph algorithms, data structures, approximate shortest paths, distance oracles, distance labels

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.55

1 Introduction

Given a graph $G = (V, E)$ on $n := |V|$ vertices, the *All-Pairs Shortest Path (APSP)* problem asks for pairwise distances $d(u, v)$ among all pairs of nodes $u, v \in V$. The fastest known algorithms to compute these $\binom{n}{2}$ distance values (in this paper, we restrict ourselves to undirected, unweighted graphs) use fast matrix multiplication [16, 18, 29, 34] or run in roughly cubic time [12, 20, 33, 37]. Depending on the number of edges $m := |E|$, computing n independent shortest-path trees (breadth-first search trees for unweighted graphs) in $O(mn)$ may be faster.

Algorithms for All-Pairs *Approximate* Shortest Paths (*APASP*) trade accuracy for speed. Their worst-case approximation guarantee is called *stretch*. Stretch (α, β) means that for any pair of nodes $u, v \in V$ at distance d the algorithm's estimate is between d and $\alpha \cdot d + \beta$ (typically, when $\beta > 0$, graphs are assumed to be unweighted or β depends on the largest edge weight).

Aingworth, Chekuri, Indyk, and Motwani [5] introduced a technique to handle high-degree nodes using *dominating sets*. They apply their technique to derive a $(1, 2)$-stretch algorithm that runs in time $\tilde{O}(n^{5/2})$ (as usual, $\tilde{O}(\cdot)$ hides logarithmic factors). Their dominating-set technique has been used in many subsequent algorithms. Dor, Halperin, and Zwick [17] improved the running time to $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$. Furthermore, they also gave a reduction, proving that an APASP algorithm with multiplicative stretch strictly less than 2

© Christian Sommer;
licensed under Creative Commons License CC-BY

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 55; pp. 55:1–55:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 A selection of related work on the all-pairs approximate shortest path problem and constant-time approximate distance oracles for unweighted graphs. Some results hold for weighted graphs as well, and some methods guarantee better bounds for sparse graphs. To simplify the comparison for dense graphs, the bounds in this table are for $m = n^2$ (except for the sub-quadratic algorithm of [6]). Stretch (α, β) means that for pairs at distance d the distance estimate is between d and $\alpha d + \beta$.

Stretch	Time $\tilde{O}(\cdot)$	Space $\tilde{O}(\cdot)$	Reference
(1,2)	$n^{5/2}$	n^2	[5]
(1,2)	$n^{7/3}$	n^2	[17]
(3,0)	n^2	n^2	[15]
(3,0)	$n^{5/2}$	$n^{3/2}$	[31]
(3,0)	n^2	$n^{3/2}$	[8, 9]
(3,10)	$n^{23/12} + m$	$n^{3/2}$	[6]
(2,1)	n^2	n^2	[10]
(2,1)	$n^{8/3}$	$n^{5/3}$	[7] (space implicit)
(2,3)	n^2	$n^{5/3}$	[7] (space implicit)
(2,1)	poly	$n^{5/3}$	[21]
(2,1)	n^2	$n^{5/3}$	Theorem 1

or constant additive stretch serves as an algorithm for boolean matrix multiplication. After this reduction, most research efforts have been focused on stretch 2 or higher. Cohen and Zwick [15] provided various tradeoffs such as an $\tilde{O}(n^2)$ algorithm with stretch $\alpha = 3$. Berman and Kasiviswanathan [10] further improved the stretch to $(2, 1)$. See also the survey by Zwick [38].

A *distance oracle* is a *compact* representation of the AP(A)SP matrix of a graph $G = (V, E)$. The main quantities of interest are the *construction time* (also called *preprocessing time*), the *space consumption*, the *query time*, and the *stretch*. Thorup and Zwick [31] coined the term *distance oracle*, and they also provided an oracle with constant query time, multiplicative stretch $\alpha = 3$, space $O(n^{3/2})$, and preprocessing time $\tilde{O}(mn^{1/2})$, as well as more general tradeoffs for all odd integers $\alpha > 3$, which are not discussed any further in this paper.[1] For $\alpha = 3$, their space bound is asymptotically optimal due to the existence of dense graphs with large girth [11, 24, 32]. The preprocessing time was subsequently improved in a line of work: Baswana and Sen [9] improved the preprocessing time to $O(n^2)$ for unweighted graphs, Baswana and Kavitha [8] obtained $O(n^2)$ preprocessing time for weighted graphs, and Baswana, Gaur, Sen, and Upadhyay [6] gave a sub-quadratic preprocessing algorithm for stretch $(3, 10)$. See also the survey by Sen [26].[2]

Is stretch $\alpha = 3$ the best we can expect of a distance oracle with sub-quadratic space? Pătraşcu and Roditty [21][3] provide an oracle with constant query time, stretch $(2, 1)$, space $O(n^{5/3})$, and polynomial preprocessing time. The tradeoff between stretch and space is asymptotically optimal assuming the hardness of set intersection [21, 22]. Less well known, the work of Baswana, Goyal, and Sen [7] contains a data structure that is remarkably similar

[1] For larger stretch values, Wulff-Nilsen [36] provides the fastest preprocessing times and Chechik [13, 14] provides the fastest query times; for a survey, see [27].

[2] Again for larger stretch values, Wulff-Nilsen [36] further improved the preprocessing times.

[3] See also Abraham and Gavoille [2] for a distributed distance oracle, also known as a *distance labeling scheme*, as well as Agarwal and Brighten Godfrey [4].

to the distance oracle in the result of Pătraşcu and Roditty [21], implicitly providing an $\tilde{O}(mn^{2/3})$ bound on the preprocessing time for $(2,1)$-approximate distance oracles.[4] They also provide an $\tilde{O}(n^2)$-time algorithm for a data structure with stretch $(2,3)$ instead of the optimal $(2,1)$. Again, the space bound is not stated explicitly as a distance oracle result. For an overview, see Table 1.

What is the fastest preprocessing time that can be achieved for distance oracles with multiplicative stretch $\alpha < 3$? We propose an algorithm that runs in quadratic time (up to polylogarithmic factors) and computes a $(2,1)$-approximate distance oracle. Analogous to the distance oracles by Abraham and Gavoille [2], the distance oracle can also be distributed as *distance labels*.

▶ **Theorem 1.** *Given an undirected, unweighted graph G on n nodes, there is an $\tilde{O}(n^2)$-time algorithm that computes a $(2,1)$-approximate distance oracle of size $\tilde{O}(n^{5/3})$ with query time $O(1)$.*

Previously known quadratic-time algorithms either produce data structures with quadratic space ([10, 15, 17], see also Section 2.4) or with worst-case stretch $(2,3)$ (see [7]). Probably the most interesting technical aspect of the result is the use of an additive spanner *free of charge*, i.e., without incurring its typical stretch penalty (the query time has a linear dependency on the additive stretch of the spanner). The argument works for any spanner with constant additive stretch, and hence is tight due to a recent lower bound by Abboud and Bodwin [1].

For general graphs, Theorem 1 seems hard to improve upon (modulo logarithmic factors) for two reasons (beyond the obvious reason that the algorithm needs to read the graph with potentially $\Omega(n^2)$ edges): the space-stretch tradeoff is essentially tight due to a set intersection lower bound [21, 22], and the preprocessing-stretch-query tradeoff is essentially tight as for distance oracles with quadratic preprocessing time and $\tilde{O}(1)$ query time (but independent of the space), any improvement of the multiplicative stretch below 2 would imply a (quasi-)quadratic algorithm for boolean matrix multiplication [3, 17] (the fastest known combinatorial algorithm runs in roughly cubic time [37]).

For sparse graphs, I am not aware of any arguments against even faster preprocessing time $\tilde{O}(m + n^{5/3})$ (and no additive $+1$ stretch if $m = \tilde{O}(n)$ [21]). As mentioned above, Baswana, Goyal, and Sen [7] implicitly prove an $\tilde{O}(mn^{2/3})$ bound on the preprocessing time (Theorem 5), which for $m = \tilde{O}(n)$ is asymptotically optimal. The optimal preprocessing time for m between $\tilde{\omega}(n)$ and $\tilde{o}(n^2)$ remains open.

The distance oracle of Theorem 1 roughly works as follows (the description omits $\tilde{O}$-notation and assumes familiarity with the techniques described in the preliminaries (Section 2)): short distances are exact by storing balls and their intersection (as in [2, 7, 21]); long distances are triangulations via *landmarks*. However, we cannot afford to compute exact distances for each node and all $n^{2/3}$ landmarks. Instead, we group landmarks by degree (using dominating sets), where paths from/to the 2^i landmarks with degree $n/2^i$ are computed in graphs with $n^2/2^i$ edges plus the edges of an $(1,6)$-spanner [35]. The spanner ensures that distances do not deteriorate too much. For each node v, we compute a set of *portals* by sparsifying its set of (up to $\log n$) nearest landmarks to include a landmark with degree $n/2^j$ if and only if it is closer to v than all landmarks with higher degree $n/2^i$ (i.e., for all $i < j$). At query time, a constant number of landmarks per node is sufficient to guarantee

[4] An important reference I unfortunately failed to include when writing [27].

stretch $(2, 1)$: the query algorithm triangulates via landmarks by increasing degree, and each subsequent landmark implies that the target distance increases by at least 1.

The overall framework is described in Section 3. The main part of the argument is formalized as a distance oracle for *heavy paths*, where a path is defined to be heavy if it passes through two consecutive high-degree nodes, see Section 4.

2 Preliminaries

2.1 Spanners

Spanners [23] are relatively sparse subgraphs with additional properties such as stretch bounds on shortest-path distances. A spanner is deemed to have *stretch* (α, β) if $d_H(u, v) \leqslant \alpha \cdot d_G(u, v) + \beta$. Note that, since H is a subgraph, $d_G(u, v) \leqslant d_H(u, v)$. While there are no known distance oracles or distance labeling schemes with constant additive stretch (and are unlikely to exist [19, 21, 28]), such spanners can be computed efficiently. The preprocessing algorithm in this paper invokes the following spanner construction as a subroutine.

▶ **Theorem 2** (Woodruff [35]). *Given an undirected, unweighted graph G on n nodes, there is an $\tilde{O}(n^2)$-time algorithm that computes a $(1, 6)$-spanner with $\tilde{O}(n^{4/3})$ edges.*

Recently, Abboud and Bodwin [1] proved that Theorem 2 is tight in a very strong way. Any spanner with constant additive stretch must have essentially $n^{4/3}$ edges. More precisely, they prove the following.

▶ **Theorem 3** (Abboud and Bodwin [1]). *For all $\epsilon > 0$ there exists a $\delta > 0$ and an infinite family of graphs $G = (V, E)$ such that for any subgraph $H = (V, E')$ with $|E'| = O(n^{4/3-\epsilon})$, there exist nodes $u, v \in V$ with $d_H(u, v) = d_G(u, v) + \Omega(n^{\delta})$.*

Their result illustrates the limitations of the construction in this paper. There is no better spanner than Woodruff's for our purposes (as long as we are indifferent to log factors in the $\tilde{O}(\cdot)$ notation).

2.2 Balls and Clusters

While long distances can be approximated well using triangulation via *landmarks*, many distance oracles handle short-range distances using *balls* and *clusters* [2, 21, 22, 31].

▶ **Definition 4** (Balls and Clusters). Given a graph $G = (V, E)$ and a set of landmarks $L \subseteq V$, the *ball* of a node v with respect to L is $B_L(v) := \{u : d(v, u) < d(v, L)\}$. The *cluster* of a node u with respect to L is $C_L(u) := \{v : d(v, u) < d(v, L)\}$.

Note that $v \in C_L(u)$ if and only if $u \in B_L(v)$.

For any $0 < \ell < n$, random sampling with probability ℓ/n yields ℓ landmarks (in expectation) and *average* ball and cluster sizes of $O(n/\ell)$. A sampling algorithm of Thorup and Zwick [30] computes a set of landmarks L such that no node cluster $C_L(u)$ is larger than $O(n/\ell)$ and $|L| = O(\ell \log n)$. The running time of their algorithm is bounded by $O((mn \log n)/\ell)$. Abraham and Gavoille [2] extend the Thorup-Zwick sampling algorithm to also guarantee worst-case bounds on balls $|B_L(v)| = O(n/\ell)$.

Roditty, Thorup, and Zwick [25] provide a deterministic algorithm (based on hitting sets) to select landmarks. They also claim (without proof due to lack of space) that their algorithm can be used to de-randomize the quadratic-time preprocessing algorithm of Baswana and Sen [9].

2.3 Stretch-2 Distance Oracles

In the following, we give a brief description of the distance oracles by [2, 4, 21] for unweighted graphs. Similar arguments were also used in the analysis for all-pairs approximate shortest path algorithms [7].

The preprocessing algorithm starts by selecting a set of landmarks L of size roughly $|L| = \tilde{O}(n^{2/3})$ such that $|B_L(u)|, |C_L(u)| = \tilde{O}(n^{1/3})$ for all $u \in V$ (see sampling algorithm above). Long-range distances (global queries) are answered by triangulation using the landmark closest to either of the query nodes. To prepare for those queries, the preprocessing algorithm computes distances for all pairs of nodes in $V \times L$. Short-range distances (local queries) are answered using *super balls* $S_L(\cdot)$. For each node u, the algorithm computes the distance to all nodes in its ball $B_L(u)$ as well as all nodes v whose balls $B_L(v)$ intersect with $B_L(u)$, i.e., $S_L(u) := B_L(u) \cup \bigcup_{w \in B_L(u)} C_L(w)$. Since the sizes of both $B_L(\cdot)$ and $C_L(\cdot)$ are bounded by $\tilde{O}(n^{1/3})$, super balls contain at most $|S_L(u)| = \tilde{O}(n^{2/3})$ nodes.

The query algorithm for source s and target t checks whether $t \in S_L(s)$ or $s \in S_L(t)$. If so, the exact distance has been pre-computed (a local query). Otherwise, let l_s and l_t denote the landmarks closest to s and t, respectively. The long-range distance $\min\{d(s, l_s) + d(l_s, t), d(s, l_t) + d(l_t, t)\}$ is returned.

For the stretch analysis, we need to bound the distance returned for global queries. Without loss of generality, let us assume $d(t, l_t) \leqslant d(s, l_s)$. Since $B_L(s)$ and $B_L(t)$ do not intersect, $d(s, t) + 1 \geqslant d(s, l_s) + d(t, l_t) \geqslant 2d(t, l_t)$. By the triangle inequality, $d(s, l_t) \leqslant d(s, t) + d(t, l_t)$, hence the distance estimate is at most $d(s, t) + 2d(t, l_t)$, which is bounded by $2d(s, t) + 1$.

Baswana, Goyal, and Sen [7, Theorem 5.1] provide a randomized preprocessing algorithm with expected running time $O(m^{2/3} n \log n + n^2)$. More generally, they prove the following.

▶ **Theorem 5** (Baswana, Goyal, and Sen [7]). *For any $p \in (0, 1)$, an undirected unweighted graph $G = (V, E)$ can be preprocessed in expected $O(m \log n + n^2 + (n/p^2) \log n + mnp \log n)$ time to build a data structure that can report $(2, 1)$-approximate distances in constant time.*

Instead of super balls, a so-called *overlap matrix* is pre-computed to decide whether two balls intersect $B_L(u) \cap B_L(v) \stackrel{?}{=} \emptyset$. The landmark set L is computed using the sampling algorithm of Thorup and Zwick [30] (see also Section 2.2) to ensure that balls and their intersections are not too large.

2.4 High-Degree Dominating Sets

Recall that a dominating set of a graph G is a subset of nodes $D \subseteq V(G)$ such that each node is in D or has a neighbor in D. Aingworth, Chekuri, Indyk, and Motwani [5] prove that for a set of high-degree nodes there is an algorithm that can find a small dominating set (the algorithm is a greedy algorithm with logarithmic approximation ratio).

▶ **Theorem 6** (Aingworth et al. [5]). *Let $G = (V, E)$ be an undirected graph with $n := |V|$ and $m := |E|$, and let $V_\delta := \{v \in V : \deg(v) \geqslant \delta\}$. There is an algorithm that finds a dominating set for V_δ with size $O((n \log n)/\delta)$ in time $O(m + n\delta)$.*

Building on this theorem, they derive an $\tilde{O}(n^{5/2})$-time algorithm computing all-pairs approximate shortest paths with stretch $(1, 2)$. The dominating-set technique has been used extensively in algorithms for APASP (and related algorithms). Dor, Halperin, and Zwick [17, Theorem 6.2] compute an $(1, 2)$-approximate *spanner* in time $\tilde{O}(n^2)$ as follows (Algorithm 1): for decreasing node degrees $(n/2^i)$, split the nodes into high-degree and low-degree nodes,

compute a dominating set for the high-degree nodes (as in Theorem 6), compute a BFS tree from each node in the dominating set in the subgraph with edges adjacent to at least one low-degree node, and return the union of all these BFS trees.

1: **for** $i \in \{0, 1, 2, 3, \ldots, \lceil \log_2 \sqrt{n} \rceil\}$ **do**
2: $\quad \delta_i = n/2^i$
3: $\quad V_i = \{v \in V : \deg(v) \geqslant \delta_i\}$, $E_i = \{uv \in E : \deg(u) < 2\delta_i \text{ or } \deg(v) < 2\delta_i\}$
4: $\quad D_i = \mathsf{dominate}(V_i)$ (as in Theorem 6)
5: $\quad$ **for** $p \in D_i$ **do**
6: $\quad\quad T_i(p) = \mathsf{bfs}_{(V,E_i)}(p)$ (where T_i denotes the edges of the breadth-first tree)
7: $\quad$ **end for**
8: **end for**
9: **return** $\bigcup_{i,l \in D_i} T_i(p)$

Algorithm 1: Spanner construction by Dor, Halperin, and Zwick [17, Figure 7].

Cohen and Zwick [15] and Baswana and Kavitha [8] also provide algorithms similar to Algorithm 1 with different parameters and analysis.

Yet another instantiation of the above algorithm and framework is due to Berman and Kasiviswanathan [10]. Their all-pairs approximate shortest path algorithm computes $(2, 1)$-approximate distances in time $O(n^2 \log^2 n)$. Their main loop is essentially as in Algorithm 1, Lines 1–8 for i up to $\lceil \log_2 n \rceil$. Each node v remembers its nearest landmark $l_i(v)$ (for each level i, and with arbitrary tie breaking). Distance estimates $\tilde{d}(s, t)$ are computed as the minimum over all i of the triangulation via the landmark of s or via the landmark of t. We apply and extend their argument in Section 4.2.2.

3 Proof of Theorem 1

We split the problem into a dense case and a sparse case, where sparse means $O(n^{4/3})$ edges. The overall distance oracle simply returns the minimum from both data structures.

As in most shortest-path algorithms using dominating sets (see Section 2.4), edges uv are classified by minimum degree of their adjacent vertices u and v:

▶ **Definition 7** (Edge Degree)**.** The *degree* of an edge $uv \in E(G)$, denoted by $\deg(uv)$, is defined as $\deg(uv) := \min_{u,v} \{\deg(u), \deg(v)\}$.

Let G_δ denote the subgraph of G induced by all edges uv for which $\deg(uv) \leqslant \delta$. Note that $|E(G_\delta)| = O(n \cdot \delta)$, since each edge contributes to the degree of two nodes. As degree threshold in our proof, let Δ be the smallest power of 2 greater than $n^{1/3}$.

For the sparse case, we run the $O(n^2 + m^{2/3} n \log n)$-time algorithm of Baswana, Goyal, and Sen [7] (Theorem 5) on G_Δ. For $m = O(n^{4/3})$, the running time is $O(n^2)$. As mentioned in the introduction, their data structure is similar to the distance oracles by Pătraşcu and Roditty [21] and Abraham and Gavoille [2]. See Section 2.3 for more details.

For the dense case, we compute a data structure for G called *Heavy Path Oracle*. It is essentially an approximate distance oracle but limited in the following way: it can only return *heavy* paths.

▶ **Definition 8** (Heavy Paths)**.** A path is called δ-*heavy* if and only if it contains an edge uv with degree $\deg(uv) \geqslant \delta$.

Quite naturally, and analogously to a shortest path, there is a shortest δ-heavy path between any pair of nodes $s, t \in V(G)$. Let the *shortest δ-heavy distance* between s and t be the length of a shortest δ-heavy path between s and t.

The distance oracle for the dense case handles those pairs correctly for which any shortest path is Δ-heavy. If no shortest path between s and t is Δ-heavy, any shortest path must be in G_Δ, and hence the distance oracle for the sparse case provides an accurate estimate.

The remainder of this paper is devoted to the proof of the following lemma, which concludes the proof of Theorem 1.

▶ **Lemma 9** (Heavy Path Oracle). *Given an undirected, unweighted graph G on n nodes, let Δ denote the smallest power of 2 greater than $n^{1/3}$. There is an $\tilde{O}(n^2)$-time algorithm that computes a data structure of size $\tilde{O}(n^{5/3})$ with query time $O(1)$ that returns $(2,1)$-approximations for shortest Δ-heavy distances.*

4 Heavy Path Oracle

We prove Lemma 9 by adapting the APASP algorithms of Dor, Halperin, and Zwick [17] and Berman and Kasiviswanathan [10].

As typical in distance oracles, each node $v \in V$ stores distances to a set of *landmarks*, chosen as the union of hierarchical dominating sets (details below). Furthermore, each node stores a constant-size subset of nearby landmarks called *portals*. At query time, triangulation happens via portals.

4.1 Preprocessing Algorithm

For a pseudo-code description, see Algorithm 2. Given $G = (V, E)$, the algorithm begins by computing a sparse $(1, 6)$-spanner $H = (V, S)$ as in Theorem 2. Recall that G_δ is defined to be the subgraph of G induced by all edges uv for which $\deg(uv) \leqslant \delta$. In the remainder of the preprocessing algorithm, each breadth-first search in a subgraph $G_\delta = (V, E_\delta)$ of $G = (V, E)$ shall also consider the edges of the spanner, i.e., the breadth-first search is executed in $G_\delta + H = (V, E_\delta \cup S)$.

Independent of the spanner, the edge set E is organized into $\lceil \log n^{2/3} \rceil$ hierarchical classes: for $\delta_i = n/2^i$, let $V_i := \{v \in V : \deg(v) \geqslant \delta_i\}$. Class E_i consists of all edges uv for which u or v has degree $< 2\delta_i$. The algorithm then finds a dominating set for V_i, denoted by L_i (the union of all L_i is the *landmark set*). For each node $\ell \in L_i$, the algorithm runs a breadth-first search in $(V, E_i \cup S)$, computing and storing distances $d_{(V,E_i \cup S)}(\ell, \cdot)$. Each node v remembers its nearest landmark as a *portal* $p_i(v)$ if and only if it is closer than all portals $p_j(v)$ on previous levels $j < i$. This selection of portals is essential for the query time and stretch analysis. Let $K \geqslant 12$ (twice the additive stretch of the spanner). For each node v, let $P(v)$ denote the set that contains the $K + 1$ portals closest to v.

4.1.1 Space and Running Time Analysis

We have $O(\log n)$ distance tables $L_i \times V$. The dominating sets are small $|L_i| \leqslant \tilde{O}(n^{2/3})$, hence the distance tables can be kept (and stored in a distributed way at the non-landmark node to also enable distance labels).

For each level i, each node v stores distances to all nodes of the dominating set L_i. This dominating set L_i has size $O(2^i \log n)$ due to Theorem 6. The last level has the largest dominating set of size $O(n^{2/3} \log n)$. The overall space consumption is thus bounded by $O(n^{5/3} \log^2 n)$.

.

```
1: (V, S) = Spanner(V, E) (as in Theorem 2)
2: for i ∈ {0, 1, 2, 3, ..., ⌈log_2 n^{2/3}⌉} do
3:    δ_i = n/2^i
4:    V_i = {v ∈ V : deg(v) ⩾ δ_i}
5:    L_i = DominatingSet(V_i) (as in Theorem 6)
6:    E_i = {uv ∈ E : deg(u) < 2δ_i or deg(v) < 2δ_i}
7:    for ℓ ∈ L_i do
8:       BreadthFirstSearch_{(V,E_i∪S)}(ℓ)
9:       for v ∈ V do
10:         store distance d_{(V,E_i∪S)}(v, ℓ) to landmark
11:         if d_{(V,E_i∪S)}(v, ℓ) < d_{(V,E_i∪S)}(v, ℓ_i(v)) then
12:            ℓ_i(v) = ℓ (remember nearest landmark per level)
13:         end if
14:      end for
15:   end for
16: end for
    Portal Selection
17: for v ∈ V do
18:   for i ∈ {0, 1, 2, 3, ..., ⌈log_2 n^{2/3}⌉} do
19:      if d_{(V,E_i∪S)}(v, ℓ_i(v)) < d_{(V,E_j∪S)}(v, ℓ_j(v)) for all j < i then
20:         store ℓ_i(v) as a candidate portal
21:      end if
22:   end for
23:   let P(v) be the first K + 1 portals closest to v
24: end for
```

Algorithm 2: Preprocessing of graph $G = (V, E)$

For each level i, the dominating set is computed in time at most $O(n^2)$ by Theorem 6. For each $\ell \in L_i$ the algorithm computes a breadth-first search in a graph with at most $|E_i| = O(n^2/2^i)$ plus $|S| = \tilde{O}(n^{4/3})$ edges (the latter are the edges of the $(1, 6)$-spanner). Hence, the running time per level is at most $O(n^2 + |L_i| \cdot (|E_i| + |S|)) = \tilde{O}(n^2)$.

Summing up over all levels, the running time is $\tilde{O}(n^2)$ plus the time to compute the spanner, which is $\tilde{O}(n^2)$ by Theorem 2, plus the time to select the portals per node, which is at most $O(n^2 \log n)$ (since they are already generated in sorted order).

4.2 Query Algorithm

Distance estimates $\tilde{d}(s, t)$ are computed as the minimum over all the triangulations via portals of s and t: $\min_{p \in P(s) \cup P(t)} \{\tilde{d}(s, p) + \tilde{d}(p, t)\}$, where $\tilde{d}(s, p)$ and $\tilde{d}(p, t)$ denote the distances stored at s and t, respectively, to/from landmark p.

4.2.1 Query Time

There are $K + 1$ portals per node, each triangulation requires two table lookups for landmark distances (one for $\tilde{d}(s, p)$ and the other one for $\tilde{d}(p, t)$), hence the query time is $O(K)$ (using perfect hashing). For the stretch analysis to work, we need $K \geqslant 12$ (twice the additive stretch of the spanner), hence the query time is constant.

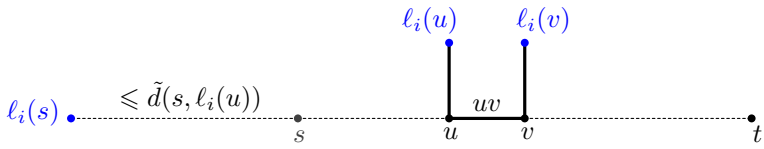

Figure 1 Worst-case stretch when triangulating via landmark $\ell_i(s)$. Distances are in $G = (V, E_i \cup S)$. Solid lines depict edges, dashed lines depict paths (which may have no, one, or multiple edge(s)). Edge uv is an edge on the shortest path with highest degree $\deg(uv)$. In this illustration, query node s is closer to uv than t. Since $\ell_i(s)$ is the nearest landmark for s, we have that $\tilde{d}(s, \ell_i(s)) \leqslant \tilde{d}(s, \ell_i(u))$.

4.2.2 Stretch Analysis

The stretch bound for the distance estimate $\tilde{d}(s,t)$ is derived as follows.

First, let us assume that, instead of triangulating via portals $P(\cdot)$ we were to triangulate via all closest landmarks of s and t, respectively. For each level i we compute the triangulation via the nearest landmark of s, i.e., $\tilde{d}(s, \ell_i(s)) + \tilde{d}(\ell_i(s), t)$, and via the nearest landmark of t, i.e., $\tilde{d}(s, \ell_i(t)) + \tilde{d}(\ell_i(t), t)$ and return the minimum.

The following argument is illustrated in Figure 1. Let E_i be the set that contains all the edges of the shortest s-t path and maximizes i. Let $uv \in E_i$ be an edge on this shortest path that is not in E_{i+1} (an edge between two high-degree nodes). The shortest-path length is $d(s,u) + 1 + d(v,t)$. Suppose $d(s,u) \leqslant d(v,t)$ (the other case is symmetric). Let us consider s and its nearest landmark $\ell_i(s)$. Since $u \in V_i$, u or its neighbor is in L_i, hence $\tilde{d}(s, \ell_i(s)) = d(s, \ell_i(s)) \leqslant d(s,u) + 1$. Triangulating via $\ell_i(s)$ increases the path length by at most $2d(s,u) + 2 \leqslant d(s,u) + d(v,t) + 2$, which yields stretch $(2,1)$. The same argument is also used by Berman and Kasiviswanathan [10].

However, we cannot afford to try all landmarks $\ell_i(\cdot)$ at query time since there may be logarithmically many for each query node. Instead, the preprocessing algorithm filters landmarks into a set of portals. For a node u, a landmark $\ell_i(u)$ is defined to *dominate* landmark $\ell_j(u)$ if $i < j$ and $\tilde{d}(u, \ell_i(u)) \leqslant \tilde{d}(u, \ell_j(u))$, which means that distances from $\ell_j(u)$ were computed in a subgraph of the one for which distances from $\ell_i(u)$ were computed. Let $D(u,i)$ denote the set of all landmarks $\ell_j(u)$ that dominate $\ell_i(u)$.

Again, let E_i be the set that contains all the edges of the shortest s-t path and maximizes i. We distinguish three cases. The first two cases are both fairly straightforward and could be combined. The most difficult case is the third, where query nodes are far away from the dense part of the graph, and hence also far away from the heavy edge uv.

- $\ell_i(s) \in P(s)$ and $\ell_i(t) \in P(t)$. The argument above applies since the query algorithm triangulates via both landmarks and computes a $(2,1)$-approximation.
- $\ell_i(s) \in P(s)$ or $D(s,i) \cap P(s) \neq \emptyset$, and the same holds for t. Let $\ell_j(s)$ be the landmark in $P(s)$ that dominates $\ell_i(s)$. When triangulating via $\ell_j(s)$, the stretch cannot increase since $d_{(V,E_j \cup S)}(s, \ell_j(s)) \leqslant d_{(V,E_i \cup S)}(s, \ell_i(s))$. In Figure 1, simply re-label $\ell_i(s)$ with $\ell_j(s)$. Since $E_i \subseteq E_j$ graph distances cannot increase. Analogous for t.
- Neither the landmark nor a dominating landmark are in the portal set. Now the spanner comes into play. At least one of the nodes must have had more than $K+1$ candidate portals (without loss of generality it is s), otherwise $\ell_i(s)$ or a landmark $\ell_j(s)$ dominating $\ell_i(s)$ would have to be in $P(s)$. The preprocessing algorithm truncated the portal list. Let $p^{(0)}(s), p^{(1)}(s), \ldots p^{(K)}(s)$ be the sequence of portals for s, ordered by increasing distance from s, and let $x := d(s, p^{(0)}(s))$. Each subsequent portal $p^{(j)}(s)$ increases the distance

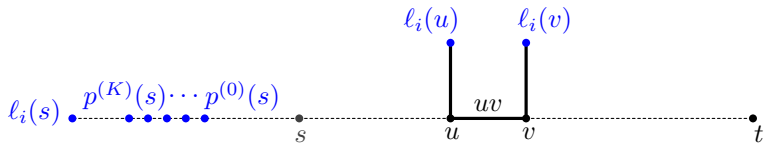

Figure 2 Worst-case stretch when triangulating via portal $p^0(s)$. There are $K+1$ portals closer than $\ell_i(s)$. When computing distances from/to these portals, the edges of the $(1,6)$-spanner were used, hence the distortion is bounded.

by at least 1, hence the last portal considered by the query algorithm is at distance at least $d(s, p^{(K)}(s)) \geqslant d(s, p^{(0)}(s)) + K$. Note that the best landmark for triangulation, $\ell_i(s)$, is even farther away, i.e., $d(s, \ell_i(s)) > d(s, p^{(K)}(s)) \geqslant d(s, p(0)(s)) + K = x + K$, otherwise it would have dominated $p^{(K)}(s)$. The shortest-path distance $d(s,t)$ is thus at least $x + K + y$, where $y := d(t, p^{(0)}(t))$. Without loss of generality, let us assume that s has a portal no farther than t has, i.e., $x \leqslant y$. Since the distance is long, we can safely triangulate using the first portal $p^{(0)}(s)$, which is relatively close to s. Due to always including the edges S of the $(1,6)$-spanner, the additive stretch when triangulating via $p^{(0)}(s)$ is at most $2(x+6)$, hence the estimate will be at most $(x + K + y) + 2(x + 6)$. Multiplicative stretch $(2,1)$ means the estimate can be $2(x + K + y) + 1$. Since $x \leqslant y$, the bound follows for $K \geqslant 12$. See Figure 2.

5 Conclusion

Distance oracles with stretch $(2,1)$ can be computed in quadratic time (modulo logarithmic factors) even for dense graphs. The main technical contribution is the use of a spanner without paying for its stretch penalty (the additive stretch of the spanner only affects the query time of the distance oracle). Previous attempts using a spanner increased the stretch to $(2,3)$, see [7].

The main difficulty lies in computing distances from/to landmarks. Since there are $n^{2/3}$ landmarks, we cannot afford to compute exact distances to all of them. Our algorithm allows constant additive distortion using a spanner. Other than pre-computing a spanner, the main bottleneck of the preprocessing algorithm in this paper is computing distances from $n^{2/3}$ nodes in a graph with $n^{4/3}$ edges. If there were a way to compute a spanner with additive stretch and fewer edges, there might be ways to push the preprocessing time below quadratic (if the number of edges m is sub-quadratic of course). However, Abboud and Bodwin [1] recently showed that Woodruff's spanner [35] is essentially as good as it gets for constant additive stretch. For stretch-3 distance oracles, there are sub-quadratic algorithms, increasing the stretch by a small additive constant [6]. For stretch-2 distance oracles, it seems like any further improvement to preprocessing times may have to find a way around landmarks or additive spanners.

Another open question is whether the result can be generalized to weighted graphs. The APASP algorithm of Berman and Kasiviswanathan [10] guarantees stretch $(2, w)$, where w denotes the largest edge length/weight on the shortest path. There are two main points where the algorithm and its analysis do not generalize: *a)* the additive spanner is insensitive to edge lengths, and *b)* each subsequent portal $p^{(i)}(\cdot)$ increases the distance by 1, which would not necessarily be true when edge lengths come into play.

Acknowledgments. Thanks to the anonymous reviewers for their feedback on earlier versions.

References

1. Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. In *48th ACM Symposium on Theory of Computing (STOC)*, 2016. To appear, available from: `http://arxiv.org/abs/1511.00700`.
2. Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In *25th International Symposium on Distributed Computing (DISC)*, pages 404–415, 2011. `doi:10.1007/978-3-642-24100-0_39`.
3. Rachit Agarwal. The space-stretch-time tradeoff in distance oracles. In *22nd European Symposium on Algorithms (ESA)*, pages 49–60, 2014. `doi:10.1007/978-3-662-44777-2_5`.
4. Rachit Agarwal and Philip Brighten Godfrey. Brief announcement: a simple stretch 2 distance oracle. In *32nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 110–112, 2013. `doi:10.1145/2484239.2484277`.
5. Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999. Announced at SODA 1996. `doi:10.1137/S0097539796303421`.
6. Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In *35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 609–621, 2008. `doi:10.1007/978-3-540-70575-8_50`.
7. Surender Baswana, Vishrut Goyal, and Sandeep Sen. All-pairs nearly 2-approximate shortest paths in $O(n^2 \text{poly} \log n)$ time. *Theoretical Computer Science*, 410(1):84–93, 2009. Announced at STACS 2005.
8. Surender Baswana and Telikepalli Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM Journal on Computing*, 39(7):2865–2896, 2010. Announced at FOCS 2006. `doi:10.1137/080737174`.
9. Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Transactions on Algorithms*, 2(4):557–577, 2006. Announced at SODA 2004. `doi:10.1145/1198513.1198518`.
10. Piotr Berman and Shiva Prasad Kasiviswanathan. Faster approximation of distances in graphs. In *10th International Workshop on Algorithms and Data Structures (WADS)*, pages 541–552, 2007.
11. William G. Brown. On graphs that do not contain a Thomsen graph. *Canadian Mathematical Bulletin*, 9:281–285, 1966.
12. Timothy M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In *26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 212–217, 2015.
13. Shiri Chechik. Approximate distance oracles with constant query time. In *46th ACM Symposium on Theory of Computing (STOC)*, pages 654–663, 2014. `doi:10.1145/2591796.2591801`.
14. Shiri Chechik. Approximate distance oracles with improved bounds. In *47th ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 2015. `doi:10.1145/2746539.2746562`.
15. Edith Cohen and Uri Zwick. All-pairs small-stretch paths. *Journal of Algorithms*, 38(2):335–353, 2001. Announced at SODA 1997. `doi:10.1006/jagm.2000.1117`.
16. Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. Announced at STOC 1987. `doi:10.1016/S0747-7171(08)80013-2`.

17 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000. Announced at FOCS 1996. doi:10.1137/S0097539797327908.

18 François Le Gall. Powers of tensors and fast matrix multiplication. In *39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

19 Cyril Gavoille and Christian Sommer. Sparse spanners vs. compact routing. In *23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 225–234, 2011. doi:10.1145/1989493.1989526.

20 Yijie Han and Tadao Takaoka. An $O(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. In *13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 131–141, 2012. doi:10.1007/978-3-642-31155-0_12.

21 Mihai Patrascu and Liam Roditty. Distance oracles beyond the Thorup-Zwick bound. *SIAM Journal on Computing*, 43(1):300–311, 2014. Announced at FOCS 2010. doi:10.1137/11084128X.

22 Mihai Patrascu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *53rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 738–747, 2012.

23 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.

24 Istvan Reiman. Über ein Problem von K. Zarankiewicz. *Acta Mathematica Academiae Scientiarum Hungarica*, 9:269–273, 1958.

25 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 261–272, 2005. doi:10.1007/11523468_22.

26 Sandeep Sen. Approximating shortest paths in graphs. In *3rd International Workshop on Algorithms and Computation (WALCOM)*, pages 32–43, 2009. doi:10.1007/978-3-642-00202-1_3.

27 Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46(4):45, 2014. doi:10.1145/2530531.

28 Christian Sommer, Elad Verbin, and Wei Yu. Distance oracles for sparse graphs. In *50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 703–712, 2009.

29 Andrew James Stothers. *On the Complexity of Matrix Multiplication.* PhD thesis, University of Edinburgh, 2010.

30 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *13th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, 2001. doi:10.1145/378580.378581.

31 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005. Announced at STOC 2001. doi:10.1145/1044731.1044732.

32 Rephael Wenger. Extremal graphs with no C^4's, C^6's, or C^{10}'s. *Journal of Combinatorial Theory, Series B*, 52:113–116, 1991.

33 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *46th ACM Symposium on Theory of Computing (STOC)*, pages 664–673, 2014. doi:10.1145/2591796.2591811.

34 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *44th Symposium on Theory of Computing (STOC)*, pages 887–898, 2012. doi:10.1145/2213977.2214056.

35 David P. Woodruff. Additive spanners in nearly quadratic time. In *37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 463–474, 2010. doi:10.1007/978-3-642-14165-2_40.

36 Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 202–208, 2012. URL: http://portal.acm.org/citation.cfm?id=2095134&CFID=63838676&CFTOKEN=79617016.

37 Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015)*, pages 1094–1105, 2015. doi:10.1007/978-3-662-47672-7_89.

38 Uri Zwick. Exact and approximate distances in graphs – A survey. In *9th European Symposium on Algorithms (ESA)*, pages 33–48, 2001. doi:10.1007/3-540-44676-1_3.