Open access • Journal Article • DOI:10.1137/080737174

Faster Algorithms for All-pairs Approximate Shortest Paths in Undirected Graphs — Source link

Surender Baswana, Telikepalli Kavitha

Institutions: Indian Institute of Technology Kanpur, Indian Institute of Science

Published on: 01 May 2010 - SIAM Journal on Computing (Society for Industrial and Applied Mathematics)

FASTER ALGORITHMS FOR ALL-PAIRS APPROXIMATE SHORTEST PATHS IN UNDIRECTED GRAPHS[*]

SURENDER BASWANA[†] AND TELIKEPALLI KAVITHA[‡]

Abstract. Let $G = (V, E)$ be a *weighted* undirected graph having non-negative edge weights. An estimate $\hat{\delta}(u, v)$ of the actual distance $\delta(u, v)$ between $u, v \in V$ is said to be of stretch t iff $\delta(u, v) \leq \hat{\delta}(u, v) \leq t \cdot \delta(u, v)$. Computing all-pairs small stretch distances efficiently (both in terms of time and space) is a well-studied problem in graph algorithms.

We present a simple, novel and generic scheme for all-pairs approximate shortest paths. Using this scheme and some new ideas and tools, we design faster algorithms for all-pairs t-stretch distances for a whole range of stretch t, and also answer an open question posed by Thorup and Zwick in their seminal paper [Approximate Distance Oracles, Journal of ACM, 52(1), 2005, pp 1-24].

Key words. shortest path, distance, approximate distance, oracle, randomization.

AMS subject classifications. 05C12, 05C85, 68W05, 68W20, 68W25, 68W40

1. Introduction.

The all-pairs shortest paths (APSP) problem is undoubtedly one of the most fundamental algorithmic graph problems. The problem is defined as follows : *Preprocess a given graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges efficiently to build a data structure that can quickly answer a shortest path query or a distance query for any pair of vertices.* There are various algorithms for the APSP problem depending upon whether the graph is directed or undirected, edges are weighted or unweighted, weights are non-negative or negative. In its most generic version, that is, for a directed graph with real edge weights, the best known algorithm [13] for this problem requires $O(mn + n^2 \log \log n)$ time; note that for graphs with $m = \Theta(n^2)$, this algorithm has a running time of $\Theta(n^3)$ which matches that of the old and classical algorithm of Floyd and Warshal [9]. The existing lower bound on the time complexity of the APSP problem is the trivial $\Omega(n^2)$ lower bound. Researchers have striven for years to improve the time complexity of the APSP problem but with little success - the best known upper bound on the worst case time complexity of the APSP problem is $O(n^3/\log^2 n)$ due to Chan [6], which is *marginally* sub-cubic.

There exist sub-cubic ($O(n^{3-\epsilon})$ time) algorithms for the APSP problem for a few classes of graphs. All these algorithms employ the fast (sub-cubic) algorithm for matrix multiplication. The underlying intuition for taking this approach is the fact that computing all-pairs distances in a graph is related to computing $(min, +)$ product (called *distance* product) of matrices. Let ω be the exponent of matrix multiplication, i.e., the smallest constant for which matrix multiplication can be performed using $O(n^\omega)$ algebraic operations - additions, *subtractions*, and multiplications. The fastest known algorithm for matrix multiplication due to Coppersmith and Winograd [8] implies $\omega < 2.376$. For undirected unweighted graphs, Seidel [15] designed a very simple and elegant algorithm to solve the APSP problem in $\tilde{O}(n^\omega)$ [1] time. For undirected graphs with integer edge weights from $\{0, 1, ..., M\}$, Shoshan and Zwick [16]

[*]The results of the preliminary version of this paper appeared in FOCS 2006 [5].

[†]Computer Science and Engineering Department, IIT Kanpur, India; `sbaswana@cse.iitk.ac.in`. Part of this work was done while the author was a postdoctoral researcher at Max-Plank Institute for computer science, Saarbruecken, Germany.

[‡]Computer Science and Automation Department, Indian Institute of Science, Bangalore 560012, India; `kavitha@csa.iisc.ernet.in.`

[1]we shall use $\tilde{O}(f(n))$ to denote $O(f(n)\text{polylog}n)$, where $f(n)$ is poly(n).

designed an $\tilde{O}(Mn^{\omega})$ algorithm for the APSP problem; note the dependence of the time complexity on the absolute value of max edge weight. For graphs with real edge weights, researchers have recently explored the application of fast matrix multiplication algorithms for the APSP problem [6, 19]. For such graphs, Yuster [19] designed an algorithm that achieves sub-cubic running time provided the number of distinct weight edges emanating from each vertex is $O(n^{0.338})$.

Though all the algorithms mentioned above that work for reasonably wide classes of graphs are novel and insightful, designing a sub-cubic algorithm for the APSP problem for general graphs is yet a longstanding major open problem. Another parameter of the APSP problem that is sometimes as important as the time complexity is the space complexity. All the algorithms for the APSP problem achieve $\Theta(n^2)$ space. On the positive side, this upper bound on space complexity matches the lower bound too. But on the negative side, this quadratic bound is a major bottleneck for many applications, especially various large scale applications - consider a large network (for example Internet), the $n \times n$ size table for answering distance queries is much larger than the network itself which is already too large to be stored in RAM. These theoretical as well as practical motivations are enough to inspire researchers to explore ways to design efficient algorithms for the all-pairs *approximate* shortest paths (APASP) problem. The aim of this research is to design a sub-quadratic space data structure in sub-cubic (or even quadratic) time so that this data structure is capable of answering efficiently any distance query *approximately*.

First we formalize the notion of an algorithm that computes *approximate* shortest-paths/distances. As the name suggests, the distance reported by the algorithm between any given pair of vertices is not the exact distance, instead it may have some error. This error can be additive (surplus) or multiplicative (stretch). Let $\delta(u, v)$ denote the actual distance between vertices u and v in a given graph $G = (V, E)$. An algorithm is said to compute all-pairs t-approximate (*stretch*) distances for G if for any pair of vertices $u, v \in V$, the distance reported by it is at least $\delta(u, v)$ and at most $t\delta(u, v)$. In a similar manner, an algorithm is said to compute distance with *surplus* a if the distance reported is at least $\delta(u, v)$ and at most $\delta(u, v) + a$. In the last fifteen years, researchers have designed many novel combinatorial algorithms for the APASP problem for undirected graphs.

We shall now summarize the existing algorithms for the APASP problem. However, first we mention some simple lower bounds on the space and time complexity of the APASP problem. These bounds may be useful for a comparative evaluation of the upper bounds of the existing algorithms for APASP problem.

- An algorithm that computes all-pairs t-approximate distances with $t < 2$ can be used to compute the Boolean matrix product of two $n \times n$ Boolean matrices. Hence computing all-pairs distances with stretch less than 2 is as hard as Boolean matrix multiplication [10].
- Any data structure that is capable of answering a distance query with stretch less than 3 in constant time must occupy $\Omega(n^2)$ space in the worst case [18].

Cohen and Zwick [7] designed an $\tilde{O}(n^{3/2}m^{1/2})$ time algorithm for computing all-pairs 2-approximate distances. Cohen and Zwick [7] also designed an $\tilde{O}(n^{7/3})$ time algorithm for computing all-pairs 7/3-approximate distances. The space complexity of these algorithms is $\Theta(n^2)$ which is indeed optimal as seen from the above mentioned lower bound for stretch < 3. In the same paper [7], Cohen and Zwick also designed an $\tilde{O}(n^2)$ time algorithm for stretch 3, and the space required is $\Theta(n^2)$. These algorithms are nontrivial extensions of the algorithms by Dor et al. [10] for computing approximate distances in undirected unweighted graphs. For stretch ≥ 3, Thorup and Zwick [18] designed algorithms which form a milestone in the area of all-pairs approximate shortest paths. They showed that for any integer $k \geq 2$, an undirected weighted graph can be preprocessed in expected $O(kmn^{1/k})$ time to build a data structure of size $O(kn^{1+1/k})$. This data structure is capable of answering any distance query with a *stretch* $2k-1$ in just $O(k)$ time. The fact that the data structure is not storing all-pairs approximate distances explicitly, and yet it is capable of answering any distance query in *essentially* constant time is indeed amazing. Due to this feature, this data structure is called an *approximate distance oracle*. In addition, the space requirement of the oracle is essentially optimal, assuming a 1963 girth lower bound conjecture of Erdös [12]. With essentially *constant* query time (for small k) and optimal space, the only feature of the approximate distance oracle of Thorup and Zwick [18] which is not optimal is its preprocessing time, that is, the time needed to construct the oracle. The $O(kmn^{1/k})$ preprocessing time for $(2k-1)$-approximate distance oracle is indeed sub-cubic. The next natural limit to be achieved for the preprocessing time appears to be $O(n^2)$ since it matches the worst case input size. However, the current preprocessing time is larger than this limit especially when k is a small constant. For example, for graphs with $m = \Theta(n^2)$, the time taken to compute the 3-approximate distance oracle is $\Theta(n^{2.5})$.

At this point, the reader may also notice that there exist two algorithms for all-pairs 3-stretch distances. The first one is the 3-approximate distance oracle of Thorup and Zwick [18] and the second one is the 3-APASP algorithm of Cohen and Zwick [7]. It can be observed that 3-APASP algorithm of Cohen and Zwick [7] is preferred when time has to be optimized, and 3-approximate distance oracle of Thorup and Zwick [18] is preferred when space has to be optimized. Ideally, one would like an algorithm with the worst case preprocessing time of the former and the space requirement of the latter. Thorup and Zwick [18] posed the following open question : *is it possible to construct an all-pairs stretch 3 oracle that uses only $O(n^{3/2})$ space and $\tilde{O}(n^2)$ preprocessing time?* It is worth mentioning at this point that on studying the two algorithms [7, 18] (for stretch 3) the reader would realize that these algorithms and their underlying ideas appear to be quite different. This makes the open question nontrivial since it appears difficult to somehow unite the two algorithms to achieve the best of the two. A natural extension of the open question of Thorup and Zwick [18] is to improve the existing preprocessing time of the $(2k-1)$-approximate distance oracle to $O(n^2)$ in the worst case, for all values of k.

1.1. New Techniques and Contribution of this paper.

In this paper we answer the open question of Thorup and Zwick in its generality and also provide substantially faster algorithms for computing all-pairs t-approximate distances for various values of t in the range $[2, 3]$.

A generic scheme for APASP.

A simple idea to achieve sub-cubic running time for APASP is the following: Compute shortest paths from a large set of vertices in a sparse subgraph and from a small set of vertices in a dense subgraph. Most of the existing algorithms for APASP use this idea implicitly or explicitly to gain efficiency in running time. We present a simple and general scheme $\mathcal{A}(\mathcal{H})$, with input parameter $\mathcal{H}$ which is a hierarchy of $k+1$ subsets $S_0 \supseteq S_1 \supseteq \cdots \supseteq S_k$ of vertices of the given graph. We use random sampling to construct this hierarchy. For each set S_i, we define a subset of edges $E_{S_i} \subseteq E$ suitably which captures the above idea naturally as follows. If the set S_i is very *large*, the set E_{S_i} is very *sparse*, and vice versa. Our scheme executes Dijkstra's single source shortest paths algorithm from each vertex in S_i in the graph $(V, E_{S_{i+1}})$. This scheme is indeed simple and efficient, and it proves to be a generic and powerful technique for the APASP problem leading to improved algorithms for various values of stretch in the interval [2,3]. The new scheme also leads us to answer the open question of Thorup and Zwick [18]. The 3-approximate distance oracle of Thorup and Zwick[18], and the $\Theta(n^2 \log n)$ algorithm of Cohen and Zwick [7] for 3-APASP appear to be emerging from the new generic scheme $\mathcal{A}$ as can be seen from the following short summary of the scheme.

1. For a specific hierarchy $\mathcal{H}$ with $V = S_0 \supseteq S_1 \supseteq \cdots \supseteq S_k = \emptyset$ and $k = \log n$, the scheme $\mathcal{A}(\mathcal{H})$ turns out to be similar to the algorithm of Cohen and Zwick [7] for stretch 3. However, our resulting algorithm, combined with a more careful analysis, leads to stretch which is *almost* 2. In precise words, the distance reported between any pair of vertices $u, v \in V$ is at most $2\delta(u,v) + \mathbf{w}_{uv}$, where $\mathbf{w}_{uv}$ is the weight of the heaviest edge on the shortest path between u and v. In other words, the algorithm solves the $(2, \mathbf{w})$-APASP problem.
2. In a very natural manner, the scheme $\mathcal{A}(\mathcal{H})$ also leads to an $O(n^{3/2})$ space data structure capable of answering any 3-approximate distance query efficiently. The hierarchy $\mathcal{H}$ used here has the base set S_0 of size $O(\sqrt{n})$ which is formed by random sampling, and the set S_k is $\emptyset$. This data structure is parametrized such that its expected preprocessing time is $O(m\sqrt{n} + \min(m\sqrt{n}, kn^{2+\frac{1}{2k}}))$ and the query answering time is $O(k)$. For the special case when there are only two levels in the hierarchy $\mathcal{H}$, this data structure degenerates to the 3-approximate distance oracle of Thorup and Zwick [18]. Further, we improve a bottleneck in the construction of this data structure and this leads to an expected $O(\min(m\sqrt{n}, kn^{2+\frac{1}{2k}}))$ preprocessing time for the data structure. For $k = \log n$, our data structure achieves expected $O(\min(m\sqrt{n}, n^2 \log n))$ preprocessing time and $O(\log n)$ query answering time. This *almost* answers the open question posed by Thorup and Zwick [18].
3. The scheme $\mathcal{A}(\mathcal{H})$ also forms the foundation of our faster algorithms for different values of stretch in the interval $[2,3]$. In particular, it provides faster algorithms for stretch 2 and $\frac{7}{3}$. See Table 1. For each stretch, we define the hierarchy $\mathcal{H}$ suitably such that $\mathcal{A}(\mathcal{H})$ directly or after some small augmentation leads to a faster algorithm for APASP for that particular value of stretch.

Quadratic time algorithm for approximate distance oracles.

The $(2k-1)$-approximate distance oracles of Thorup and Zwick [18] build a hierarchy of subsets of vertices : $V \supseteq S_1 \supseteq \cdots \supseteq S_{k-1} \supseteq S_k = \emptyset$. In order to achieve a sub-quadratic space data structure, for each vertex u and level $i < k$, the preprocessing algorithm computes distances to a small number of vertices from S_{i-1}, called ball. In particular, the ith level ball for u consists of all those vertices in S_{i-1} that are closer to u than its nearest vertex from S_i. Computing these balls efficiently turns out to require building of truncated shortest path trees from vertices of S_{i-1}. Thorup and Zwick [18] design a variant of Dijkstra's algorithm for this task which guarantees expected $O(mn^{1/k})$ time to compute the balls at a level. This computation time is super-quadratic for dense graphs. In order to achieve a quadratic bound on the expected time for computing balls, therefore, the goal is essentially to break the dependency of the computation time on the number of edges of the given graph. In order to achieve this goal, we design an algorithm that has the following key feature. The expected time of this algorithm for computing balls is independent of the degree of a vertex. In fact, the computation time depends on the size of a ball which, due to underlying randomization, depends upon n only. It is due to this unique feature that the underlying technique of this algorithm might have applications in other algorithms as well. This algorithm leads to expected $O(\min(n^2, mn^{1/k}))$ time for computing the balls at any level. Using this improved algorithm and graph spanners, we present a $(2k-1)$-approximate distance oracle with expected $O(\min(kmn^{1/k}, n^2))$ preprocessing time for any $k > 2$. The query time is $O(k)$ for all $k > 2$ and the space requirement is $O(kn^{1+1/k})$ which is away from the conjectured lower bound only by a factor of k.

Stretch	Query Time	Space	Preprocessing Time	Reference
1	$O(1)$	$O(n^2)$	$O(mn + n^2 \log\log n))$	[13]
$1+\epsilon$	$O(1)$	$O(n^2)$	$\tilde{O}(n^{2.376})$	[21]
Algorithms for APASP with stretch < 3				
2	$O(1)$	$O(n^2)$	$O(n^{3/2}\sqrt{m}\log n)$	[7]
			$O((m\sqrt{n} + n^2)\log n)$	**this paper**
$(2, \mathbf{w})$	$O(1)$	$O(n^2)$	$O(n^2 \log n)$	**this paper**
$7/3$	$O(1)$	$O(n^2)$	$O(n^{7/3}\log n)$	[7]
			$O((m^{2/3}n + n^2)\log n)$	**this paper**
Algorithms for APASP with stretch ≥ 3				
3	$O(1)$	$O(n^2)$	$O(n^2 \log n)$	[7]
	$O(1)$	$O(n^{3/2})$	$O(m\sqrt{n})$	[18]
	$O(k)$	$O(n^{3/2})$	$O(\min(m\sqrt{n}, kn^{2+\frac{1}{2k}}))$	**this paper**
$2k-1$ (for $k > 2$)	$O(k)$	$O(kn^{1+1/k})$	$O(kmn^{1/k})$	[18]
			$O(\min(n^2, kmn^{1/k}))$	**this paper**

Table 1.1
Algorithms for approximate distances in weighted undirected graphs

1.2. Related Work. For the APASP problem in undirected unweighted graphs, many results, in particular, results with additive error (surplus) are known. Aingworth et al. [1] designed a simple and elegant $\tilde{O}(n^{5/2})$ algorithm for finding all distances with an additive error of at most 2 in an unweighted undirected graph. Their work [1] has significantly inspired the research for designing simple and combinatorial algorithms (without matrix multiplication) for all-pairs approximate shortest paths. Dor et al. [10] improved and extended the algorithms of Aingworth et al. [1] - their algorithm requires $O(kn^{2-\frac{1}{k}}m^{\frac{1}{k}}\text{polylog}n)$ time for finding distances with surplus $2(k-1)$ for all pair of vertices in unweighted undirected graphs.

There also exist algorithms for APASP in unweighted graphs that have multiplicative error as well as additive error simultaneously and achieve *close* to quadratic running time. Elkin [11] designed such an algorithm - Given an undirected unweighted graph and arbitrarily small constants $\zeta, \epsilon, \rho > 0$, there is an algorithm that requires $O(mn^\rho + n^{2+\zeta})$ time, and for any pair of vertices $u, v \in V$, reports distance $\hat{\delta}(u, v)$ satisfying the inequality :

$$\delta(u, v) \leq \hat{\delta}(u, v) \leq (1 + \epsilon)\delta(u, v) + \beta$$

where β is a function of ζ, ϵ, ρ. If the two vertices $u, v \in V$ are separated by a sufficiently long distance in the graph, the ratio $\frac{\hat{\delta}(u,v)}{\delta(u,v)}$ ensured by Elkin's algorithm is quite close to $(1+\epsilon)$, but this ratio will be quite huge for short paths since β depends on ζ as $(1/\zeta)^{\log 1/\zeta}$, depends inverse exponentially on ρ and inverse polynomially on ϵ. Baswana et al. [3] designed $\tilde{O}(n^2)$ time algorithms for all-pairs *nearly* 2-approximate distances - there is a constant additive error on top of the stretch 2. However, the techniques of neither of these two algorithms [3, 11] lead to similar results for weighted graphs. Baswana and Sen [4] showed that for unweighted graphs, a $(2k-1)$-approximate distance oracle of size $O(kn^{1+1/k})$ that returns $2k-1$ stretch distances in $O(k)$ time can be computed in expected $O(\min(n^2,\ kmn^{1/k}))$ time. Recently, Roditty, Thorup, and Zwick [14] showed that the results in [18] and [4] can be achieved deterministically also. For a more comprehensive survey of the algorithms and techniques for approximate shortest path problem and its exact variant, the reader should refer to the excellent survey paper by Zwick[20].

All the algorithms for reporting approximate distances that we mentioned above, including the new results of this paper, can be easily modified to report the corresponding approximate shortest path, and the time for doing so is proportional to the number of edges in the approximate shortest path.

Organization of the paper. In section 2, we introduce notations, definitions and lemmas to be used in the rest of the paper. This is followed by our scheme $\mathcal{A}(\mathcal{H})$ for APASP in section 3. In the same section, as an application of the scheme, we provide a simple $O(n^2 \log n)$ time algorithm for the $(2, \mathbf{w})$-APASP problem. In section 4, we describe how the same scheme $\mathcal{A}$ leads in a natural manner to a new generic data structure for 3-approximate distance oracles. We describe a faster algorithm for computing *balls* in section 5. Applying the results of section 5 and using graph spanners, we describe a quadratic time construction for $(2k-1)$-approximate distance oracles for $k > 2$ in section 6. In section 7, we describe a slightly augmented scheme $\hat{\mathcal{A}}$ which leads to faster algorithms for stretch 2 and stretch $\frac{7}{3}$.

2. Preliminaries.

Let $G = (V, E)$ be an undirected graph on $n = |V|$ vertices and $m = |E|$ edges. Let $\mathbf{w} : E \to \mathbf{R}^+$ be a weight function on the edge set of G. Without loss of generality we can assume that there are no edges of weight zero in the given undirected graph. Otherwise, we may merge the endpoints of each such edge, and work with the modified graph. We shall now introduce notations that will be used throughout the paper.

- $E(v)$ denotes the set of edges incident on vertex v.
- $\delta(u, v)$ denotes distance from u to v in the original graph. Note that $\delta(u, v) = \delta(v, u)$ since the graph is undirected.
- $p_S(v)$ denotes the vertex from S that is nearest to v (break the tie arbitrarily if there are multiple vertices nearest to v). Note that $p_S(v) = v$ if $v \in S$. [when we have a chain of subsets $S_0 \supseteq S_1 \supseteq \cdots$, we shall use $p_i(v)$ to denote the vertex $p_{S_i}(v)$]
- $\delta(v, S)$ denotes the distance from v to $p_S(v)$. We define $\delta(v, S) = \infty$ if $S = \emptyset$.
- $E_S(v)$ denotes the set of edges incident on vertex v with weight less than $\delta(v, S)$. Note that $E_S(v) = \emptyset$ if $v \in S$.
- $E_S = \cup_{v \in V} E_S(v)$.

Given a set $S \subseteq V$, it is quite easy to compute $p_S(v)$ and $\delta(v, S)$ for all $v \in V$ in $O(m + n \log n)$ time as follows. Connect a dummy vertex to all the vertices of the set

S with edges of weight zero, and then execute Dijkstra's algorithm from the dummy vertex in this augmented graph.

Now we define the basic hierarchy of subsets of vertices formed by random sampling. This hierarchy, directly or after some slight alteration, will be used by all our APASP algorithms.

DEFINITION 2.1. *Given any $0 \le \alpha < \beta \le 1$, the hierarchy $\mathcal{R}(\alpha, \beta, k)$ consists of $k+1$ subsets $\{S_0, \ldots, S_k\}$ defined as follows. S_0 is called the base set of the hierarchy, and is formed by selecting each vertex $v \in V$ independently with probability $n^{-\alpha}$. Each set S_i, for $i > 0$, is formed by selecting each vertex from S_{i-1} independently with probability $n^{\frac{\alpha-\beta}{k}}$. The set S_k is called the top set of the hierarchy. In case $\beta = 1$, we define the top set $S_k = \emptyset$.*

Most of our algorithms will use the following observation which exploits the undirectedness of the given graph. We shall refer to this observation as the *triangle inequality* on distances.

OBSERVATION 2.1. *For any three vertices $x, y, z \in V$, it is always true that $\delta(x, z) \le \delta(y, x) + \delta(y, z)$.*

The following lemma will be used repeatedly in the rest of the paper.

LEMMA 2.2. *Given a graph $G = (V, E)$ and $S \subseteq V$, the following assertions are true.*

1. *For any two vertices $u, v \in V$, if $\delta(u, v) < \delta(u, p_S(u))$, then the subgraph $G_S = (V, E_S)$ preserves the exact distance between u and v.*
2. *The subgraph $(V, E_S \cup E(p_S(u)))$ preserves the exact distance between u and $p_S(u)$.*

Proof. Consider the shortest path $u(= v_0), v_1, \cdots, v_j(= v)$ between u and v. Since $\delta(u, p_S(u)) > \delta(u, v)$ and $v_0, \ldots, v_j$ are the intermediate vertices on the shortest path between u and v, it follows that

$$\delta(u, p_S(u)) > \delta(u, v_k) \quad \text{for } 0 \le k \le j. \tag{2.1}$$

All we need to show is that all the edges of the shortest path are present in E_S. We shall prove this assertion by contradiction. Let (v_i, v_{i+1}) be the first edge (from the side of u) on the path that is not present in E_S. Since this edge is not present in E_S, it follows from the definition of $E_S(v_i)$ that there is some vertex $x \in S$ such that $\delta(v_i, x) \le \mathbf{w}(v_i, v_{i+1})$. This fact when combined with Inequality (2.1) gives us

$$\begin{aligned} \delta(u, p_S(u)) &> \delta(u, v_{i+1}) \\ &= \delta(u, v_i) + \mathbf{w}(v_i, v_{i+1}) \quad \{\text{since } u \rightsquigarrow v_i \to v_{i+1} \rightsquigarrow v \text{ is the shortest path}\} \\ &\ge \delta(u, v_i) + \delta(v_i, x) \quad \ge \quad \delta(u, x). \end{aligned}$$

Thus, for the vertex u, the vertex $x \in S$ is closer than $p_S(u)$ (a contradiction!). Hence all edges on the shortest path between u and v are present in E_S. This proves the first assertion of the lemma.

The second assertion of the lemma is just a corollary of the first assertion. If $u(= v_0), v_1, v_2, \cdots, v_i, p_S(u)$ is the shortest path between u and $p_S(u)$, it follows from the first assertion that every edge of this path, excluding the last edge, must be present in $G_S = (V, E_S)$. The last edge $(v_i, p_S(u))$ is present in $E(p_S(u))$. Thus the subgraph $(V, E_S \cup E(p_S(u)))$ preserves the exact distance between u and $p_S(u)$. ▯

The following lemma bounds the expected size of E_S when S is formed by picking each vertex with probability q.

Lemma 2.3. *If the set $S \subseteq V$ is formed by selecting each vertex independently with probability q, then the expected size of the set E_S is $O(n/q)$.*

Proof. Let us calculate the expected number of edges in $E_S(v)$ for any $v \in V$. If the vertex v belongs to the set S, then $E_S(v) = \emptyset$. Let us now estimate $E_S(v)$ when $v \notin S$. Consider the sequence $\langle v_1, v_2, \ldots \rangle$ of neighbors of v arranged in non-decreasing order of the weight of the edges incident on v. An edge (v, v_i) will belong to $E_S(v)$ if none of $v_1, \ldots, v_i$ is present in S. Since each vertex is selected in S independently with probability q, $\mathbf{Pr}[(v, v_i) \in E_S(v)] \le (1-q)^i$. Using linearity of expectation, if $v \notin S$, the expected number of edges in $E_S(v)$ is at most $\sum_i (1-q)^i = O(1/q)$. Hence the expected number of edges in E_S is $O(n/q)$. □

The following lemma can be viewed as an extension of Lemma 2.3.

Lemma 2.4. *[18] Given a graph $G = (V, E)$ and a vertex u, let Y be a set formed by picking each vertex of a set $X \subseteq V$ independently with probability q. Then the expected number of vertices from the set X whose distance from v is less than $\delta(v, Y)$ is $O(1/q)$.*

3. $\mathcal{A}(\mathcal{H})$: a hierarchical scheme for APASP.

For a given weighted undirected graph $G = (V, E)$, let $\mathcal{H} = \{S_0, \cdots, S_k\}$ be a hierarchy of subsets of vertices with $S_i \supseteq S_{i+1}$ for all $i < k$. The construction and description of the generic scheme $\mathcal{A}(\mathcal{H})$ for APASP is presented in Algorithm 1.

Algorithm 1: The construction and description of scheme $\mathcal{A}(\mathcal{H})$

```
input  : $G = (V, E)$, and hierarchy $\mathcal{H} = \{S_0, \cdots, S_k\}$ of subsets of vertices
         with $S_i \supseteq S_{i+1} \forall i < k$.
output : A table $d[\cdot, \cdot]$ storing approximate distances between all pairs
         $(s, v) \in S_0 \times V$.
foreach $i \in [0, k-1]$ do
    foreach $u \in V$ do compute $\delta(u, S_i)$ and $p_i(u)$, $d[p_i(u), u] \leftarrow \delta(u, S_i)$;
    foreach $s \in S_i$ do
        run Dijkstra's algorithm from $s$ in the subgraph $(V, E_{S_{i+1}} \cup E(s))$ and
        update the entries of row $d[s, \_]$ accordingly;
```

As we shall show in the following subsection, the scheme $\mathcal{A}$ implicitly stores approximate distances even from vertices not belonging to the hierarchy $\mathcal{H}$. To provide a better insight into the scheme, we now provide a visual description of the scheme $\mathcal{A}(\mathcal{H})$ from the perspective of a vertex $u \in V$. This visual description will be quite helpful in the analysis of the algorithms based on the scheme. Visualize the vertices arranged in a 2-d plane according to their distance from u. For each $0 \le i < k$, draw a circle centered at u and passing through $p_i(u)$. Thus there are k concentric circles around v. It can be observed from Algorithm 1 that the scheme performs Dijkstra's algorithm from $p_i(u)$ in $E_{S_{i+1}} \cup E(p_i(u))$ for each $i < k$. It follows from Lemma 2.2 that all the edges of the sub-graph induced by the vertices which lie completely inside $(i+1)$th circle will be preserved in the set $E_{S_{i+1}}$. Therefore, approximate information about the shortest paths from u to other vertices in the graph (in particular, to the the vertices lying inside $i+1$th circle) is implicitly computed by $p_i(u)$. However, this observation alone will not be able to guarantee a small bound on the worst case stretch for every $\delta(u, v)$. For this purpose, keeping the hierarchy of subsets S_i's proves to be helpful. It ensures that for any arbitrary vertex v, there will be some suitable

$i \in [0, k-1]$ such that the distance from $p_i(u)$ to v will approximate the distance $\delta(u, v)$ with a *small* stretch even in the worst case. This intuitive idea is formalized in Theorems 3.2 and 4.1 which in turn form the foundation of our algorithms for $(2, \mathbf{w})$-APASP and 3-approximate distance oracle.

As mentioned in the introduction, the scheme $\mathcal{A}(\mathcal{H})$ provides faster algorithms the APASP problem with various values of stretch by making a suitable choice of the hierarchy $\mathcal{H}$. To highlight these facts, we specifically give a description of the hierarchies that provide faster algorithms for APASP with stretch $(2, \mathbf{w})$, APASP with stretch 2, and 3-approximate distance oracle.

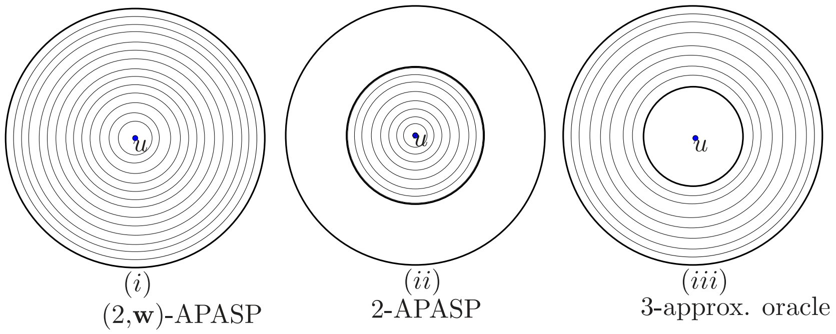

FIG. 3.1. *The scheme $\mathcal{A}(\mathcal{H})$ from perspective of u for various stretches*

- For $(2, \mathbf{w})$-APASP, we form the hierarchy $\mathcal{H}$ with base set $S_0 = V$, top set $S_k = \emptyset$ and choose $k = \log n$. In other words, $\mathcal{H} = \mathcal{R}(0, 1, \log n)$. See Figure 3.1$(i)$ for a better understanding of the scheme from the perspective of a vertex u. The scheme $\mathcal{A}(\mathcal{H})$ guarantees the following crucial property (Theorem 3.2).
  For any two vertices $u, v \in S_0$, the scheme implicitly stores a $(2, \mathbf{w})$-approximate distance between them.
  This property directly leads to an $O(n^2 \log n)$ time algorithm for $(2, \mathbf{w})$-APASP.
- For the 3-approximate distance oracle, we form the hierarchy $\mathcal{H}$ with the base set equal to a random sample of $\sqrt{n}$ vertices, and the top set is empty set. In other words, $\mathcal{H} = \mathcal{R}(\frac{1}{2}, 1, k)$. See Figure 3.1$(iii)$ for a better understanding of the scheme from the perspective of a vertex u. Here we exploit the following property (Theorem 4.1) of the scheme $\mathcal{A}(\mathcal{H})$.
  Let $S_0 \subset V$. If $\delta(u, v) \geq \delta(u, p_0(u))$ then the scheme $\mathcal{A}(\mathcal{H})$ implicitly stores 3-approximate distance between u and v.
- For APASP with stretch 2, we keep the base set $S_0 = V$, but choose the top set S_k as a random sample of size $O(\sqrt{n})$ instead of an empty set. In other words, $\mathcal{H} = \mathcal{R}(0, \frac{1}{2}, \log n)$. See Figure 3.1$(ii)$ for a better understanding of the scheme from the perspective of a vertex u. We employ a suitable augmentation of this scheme $\mathcal{A}(\mathcal{H})$ in our algorithms for APASP with stretch 2 and stretch 7/3.

The following lemma bounds the time and space requirement for the scheme $\mathcal{A}(\mathcal{H})$ for the hierarchy $\mathcal{H} = \mathcal{R}(\alpha, \beta, k)$.

LEMMA 3.1. *For a given graph $G = (V, E)$, fractions $\beta > \alpha > 0$, and an integer k, the scheme $\mathcal{A}(\mathcal{R}(\alpha, \beta, k))$ has expected $O(\min(kn^{2+\frac{\beta-\alpha}{k}},\ mn^{1-\alpha}))$ preprocessing time and $O(nk + n^{2-\alpha})$ space.*

Proof. Computation of the scheme $\mathcal{A}(\mathcal{H})$ involves k iterations, where the ith iteration executes Dijkstra's algorithm from each vertex $v \in S_{i-1}$ in the graph $G_i = (V, E_{S_i} \cup E(v))$. From the randomization employed in the construction of the hierarchy $\mathcal{H}$, it follows that the expected running time of the ith iteration is bounded by

$$\sum_{v \in V} (\deg(v) + \mathbf{E}[X_v |\ v \in S_{i-1}]) \cdot \mathbf{Pr}[v \in S_{i-1}] \tag{3.1}$$

where X_v is the random variable denoting the number of edges of the set E_{S_i} excluding those that are incident on v. From the definition of E_{S_i}, it is easy to observe that the conditional expectation $\mathbf{E}[X_v |\ v \in S_{i-1}]$ is upper bounded by the unconditional expectation X_v. Furthermore $\mathbf{E}[X_v]$ is bounded by $\mathbf{E}[E_{S_i}]$. Applying Lemma 2.3, we can thus conclude that

$$\mathbf{E}[X_v |\ v \in S_{i-1}] \le \min\left(\frac{n}{n^{-\alpha + \frac{(\alpha-\beta)i}{k}}},\ m - \deg(v)\right)$$

Hence the expected running time of the ith iteration is

$$\sum_{v \in V} \min\left(\frac{n}{n^{-\alpha + \frac{(\alpha-\beta)i}{k}}} + \deg(v),\ m\right) \cdot \mathbf{Pr}[v \in S_{i-1}]$$

$$< \min\left(\frac{2n}{n^{-\alpha + \frac{(\alpha-\beta)i}{k}}},\ m\right) \sum_{v \in V} \mathbf{Pr}[v \in S_{i-1}]$$
$$= \min\left(\frac{2n}{n^{-\alpha + \frac{\alpha-\beta}{k} i}},\ m\right) n^{1-\alpha+\frac{(\alpha-\beta)(i-1)}{k}} \quad = \quad \min(2n^{2+\frac{\beta-\alpha}{k}}, mn^{1-\alpha+\frac{(\alpha-\beta)(i-1)}{k}}).$$

Since $\beta > \alpha$, so for the k iterations, the total computation time spent in constructing $\mathcal{A}(\mathcal{H})$ is of the order of $\min(kn^{2+\frac{\beta-\alpha}{k}}, mn^{1-\alpha})$. □

We now present our algorithm for $(2, \mathbf{w})$-APASP based on the above scheme in the following subsection.

3.1. All pairs $(2, \mathbf{w})$-approximate distances. Let $\mathcal{H} = \{S_0, \cdots, S_k\}$ be a hierarchy of subsets of vertices. We now prove a crucial property of $\mathcal{A}(\mathcal{H})$.

THEOREM 3.2. *Let u and v be any two vertices in a given graph $G = (V, E)$, and let $\mathbf{w}_{uv}$ be the weight of the maximum weight edge on a shortest path between u and v. Given the scheme $\mathcal{A}(\mathcal{H})$ with $\mathcal{H} = \mathcal{R}(0, 1, k)$, we have*

$$\min_{0 \le j < k} (d[p_j(u), u] + d[p_j(u), v],\quad d[p_j(v), v] + d[p_j(v), u]) \quad \le \quad 2\delta(u, v) + \mathbf{w}_{uv}.$$

Proof. Let $i < k$ be such that $u \in S_i$ but $u \notin S_{i+1}$. Note that such a unique $i < k$ must exist since $S_0 = V$ and $S_k = \emptyset$. Now consider a shortest path P_{uv} between u and v. If the entire path P_{uv} is present in $E_{S_{i+1}} \cup E(u)$, then (the length of) this path will be computed exactly by the scheme $\mathcal{A}(\mathcal{H})$ when we run Dijkstra's algorithm from u in the graph $(V, E_{S_{i+1}} \cup E(u))$. Otherwise let $i + 1 \le j < k$ be such that $P_{uv} \nsubseteq E_{S_j}$ but $P_{uv} \subseteq E_{S_{j+1}}$. Such a j must exist because $E_{S_k} = E$. Since the path $P_{uv} \nsubseteq E_{S_j}$, there are one or more edges in P_{uv} that are not present in E_{S_j}. See Figure 3.2.

Traversing the path P_{uv} in the direction from u to v, let (x, y) be the first such edge. Since $(x, y) \notin E_{S_j}$, so as a corollary to Lemma 2.2, we have $\delta(u, p_j(u)) \le \delta(u, y)$

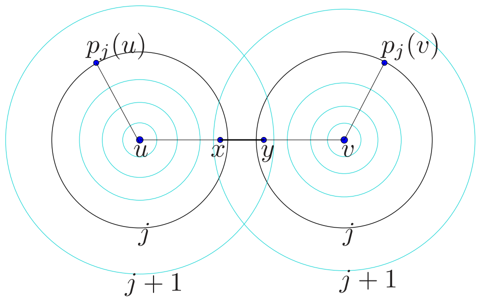

FIG. 3.2. *analyzing the scheme $\mathcal{A}$ for $(2,\mathbf{w})$-APASP*

and $\delta(v, p_j(v)) \leq \delta(v, x)$. Since the shortest path P_{uv} is of the form $u \rightsquigarrow x \rightarrow y \rightsquigarrow v$, adding the above two inequalities, we get

$$\delta(u, p_j(u)) + \delta(v, p_j(v)) \leq \delta(u, y) + \delta(x, v) = \delta(u, v) + \mathbf{w}(x, y)$$

Without loss of generality assume that $\delta(u, p_j(u)) \leq \delta(v, p_j(v))$. Then we have

$$2\delta(u, p_j(u)) \leq \delta(u, v) + \mathbf{w}(x, y). \tag{3.2}$$

Note that during the construction of scheme $\mathcal{A}(\mathcal{H})$, we execute Dijkstra's algorithm from $p_j(u)$ in the graph $(V, E_{S_{j+1}} \cup E(p_j(u)))$. Now recall from the definition of j that $P_{uv} \subseteq E_{S_{j+1}}$. Moreover, since $E_{S_j} \subseteq E_{S_{j+1}}$, it follows from Lemma 2.2 that the shortest path from $p_j(u)$ to u is also present in the graph $(V, E_{S_{j+1}} \cup E(p_j(u)))$. Using these observations and applying the triangle inequality, the approximate distance $d[p_j(u), v]$ between $p_j(u)$ and v as computed by Dijkstra's algorithm in the graph $(V, E_{S_{j+1}} \cup E(p_j(u)))$ can be bounded from above as follows.

$$d[p_j(u), v] \leq \delta(p_j(u), u) + \delta(u, v). \tag{3.3}$$

Also note that $d[p_j(u), u] = \delta(p_j(u), u)$. Now using Inequalities (3.2) and (3.3), we can bound $d[p_j(u), u] + d[p_j(u), v]$ as follows.

$$\begin{aligned} d[p_j(u), u] + d[p_j(u), v] &\leq \delta(p_j(u), u) + \delta(p_j(u), u) + \delta(u, v) \\ &= 2\delta(u, p_j(u)) + \delta(u, v) \\ &\leq \delta(u, v) + \mathbf{w}(x, y) + \delta(u, v) \quad \{\text{using Inequality (3.2)}\} \\ &= 2\delta(u, v) + \mathbf{w}(x, y). \end{aligned}$$

So if $\mathbf{w}_{uv}$ is the weight of the heaviest edge on the shortest path between u and v, then the distance $d[p_j(u), u] + d[p_j(u), v]$ is bounded by $2\delta(u, v) + \mathbf{w}_{uv}$. □

Algorithm 2 solves $(2, \mathbf{w})$-APASP by constructing $\mathcal{A}(\mathcal{H})$ on the hierarchy $\mathcal{H} = \mathcal{R}(0, 1, \log n)$ and refining the table $d[\cdot, \cdot]$ as suggested by Theorem 3.2.

The following theorem can be concluded from Lemma 3.1 and Algorithm 2.

THEOREM 3.3. *A given undirected weighted graph on n vertices can be preprocessed in expected $O(n^2 \log n)$ time to build an $n \times n$ table which stores all-pairs $(2, \mathbf{w})$-approximate distances.*

Algorithm 2: Algorithm for $(2, \mathbf{w})$-APASP problem in G.

```
$\mathcal{H} \leftarrow \mathcal{R}(0, 1, k)$ with $k = \log n$;
Let $d$ be the $n \times n$ table constructed using the scheme $\mathcal{A}(\mathcal{H})$;
foreach $u, v \in V$ do
    $d[u, v] \leftarrow \min_{0 \le i < k}\{d[p_i(u), u] + d[p_i(u), v], \quad d[p_i(v), v] + d[p_i(v), u]\}$
```

3.1.1. Comparison of $(2, \mathbf{w})$-APASP algorithm with the 3-APASP algorithm of [7]. Cohen and Zwick [7] designed an $O(n^2 \log n)$ time algorithm to compute all-pairs stretch 3 distances. At a superficial level, this algorithm of Cohen and Zwick [7] bears some similarity with our algorithm for $(2, \mathbf{w})$-APASP. In particular, the algorithm of Cohen and Zwick [7] also constructs a $k = \log n$ level hierarchy of subsets of vertices. However, there is a subtle difference in the design of the two algorithms. The reader may soon realize that this subtle difference makes the two algorithms incomparable in some sense. Both the algorithms are based on the following intuitive principle. To estimate the distance between any two vertices u and v, the distance information stored at other vertices in the graph can prove to be helpful. However the way this principle gets exploited is quite different in the two algorithms. Consider the task of computing approximate distances from a vertex u. Our algorithm employs only a *few* vertices, namely $\{p_j(u), j \ge 0\}$, but makes use of the distance information stored at them *completely.* In particular, it uses the distance from $p_i(u)$ to u as well as the distance from $p_i(u)$ to v to approximate $\delta(u, v)$. However, the algorithm of Cohen and Zwick [7] employs a significantly large number of vertices, but makes use of the distance information stored at them in a *partial* manner only as can be seen from the following summary of their algorithm.

The algorithm of Cohen and Zwick [7] first computes distances from vertices of set S_{k-1}, then distances from vertices of S_{k-2}, and so on. This order is very crucial in the algorithm of Cohen and Zwick [7]. Let $u \in S_i$. To compute approximate distances from u to all other vertices, the algorithm of Cohen and Zwick [7] does the following. Let $\hat{\delta}(w, v)$ be the estimated distance already computed by each $w \in S_{i+1}$. From u a new edge (u, w) of weight $\hat{\delta}(u, w)$ is added to each $w \in S_{i+1}$ in a sub-graph which is *similar* to $E_{S_{i+1}}$, and distances from u are then computed in this subgraph by running Dijkstra's algorithm. In this way, for approximating the distance from $u \in S_i$ to any vertex $v \in V$, this algorithm $\underline{\text{does not}}$ employ the estimated distances from vertices of S_{i+1} to v.

We would like to mention an additional point about the 3-APASP algorithm of Cohen and Zwick [7]. If $u \in S_i$, then the approximate distance $\hat{\delta}(u, v)$ between u and v that the algorithm computes satisfies the following inequality.

$$\delta(u, v) \le \hat{\delta}(u, v) \le \delta(u, v) + 2\mathbf{w}(\mathcal{E}_i)$$

where $\mathcal{E}_i$ is a set of at most i edges appearing on the shortest path between u and v.

In this manner, the two algorithms are incomparable, and neither of them is superior to the other in an absolute sense. However, a simple observation is that it is possible to design an $O(n^2 \log n)$ algorithm that achieves the best of both the algorithms.

4. All-pairs stretch 3 distances in quadratic time and sub-quadratic space.

In this section we present an $O(n^{3/2})$ size data structure that answers any distance query with stretch 3. This data structure is parametrized such that its expected preprocessing time is $O(\min(m\sqrt{n}, kn^{2+\frac{1}{2k}}))$ and query time is $O(k)$ for any $1 \le k \le \log n$.

The foundation of our data structure is again the scheme $\mathcal{A}(\mathcal{H})$. Recall from the previous section that the scheme $\mathcal{A}(\mathcal{H})$ for the hierarchy $\mathcal{H} = \mathcal{R}(0,1,k)$ implicitly stores $(2,\mathbf{w})$-approximate distances in a table that occupies $\Theta(n^2)$ space. In order to improve the space requirement, the base set S_0 of the hierarchy has to be quite small in size (see Lemma 3.1). But how can such a scheme be used for retrieving approximate distance between any two vertices not belonging to S_0 ? Here we employ another important property (Theorem 4.1) of the scheme $\mathcal{A}(\mathcal{H})$, which implies that even a small sized set S_0 would suffice to keep all-pairs 3-approximate distances.

THEOREM 4.1. *Consider the scheme $\mathcal{A}(\mathcal{H})$ built for the hierarchy $\mathcal{H} = \{S_0, ..., S_k\}$ where S_0 is a proper subset of V and $S_k = \emptyset$. If u and v are any two vertices in the graph such that $\delta(u,v) \ge \delta(u,S_0)$, then there exists some $0 \le j < k$ such that*

$$d[p_j(u), u] + d[p_j(u), v] \le 3\delta(u,v).$$

Proof. Consider the set of vertices $\{p_j(u) | 0 \le j < k\}$. It follows from the construction of S_j's that $\delta(u, p_j(u)) \le \delta(u, p_{j+1}(u))$. Since $\delta(u,S_0) \le \delta(u,v)$ and $\delta(u,S_k) = \infty$, there must be some unique $j < k$, such that

$$\delta(u, p_j(u)) \le \delta(u,v) < \delta(u, p_{j+1}(u)). \tag{4.1}$$

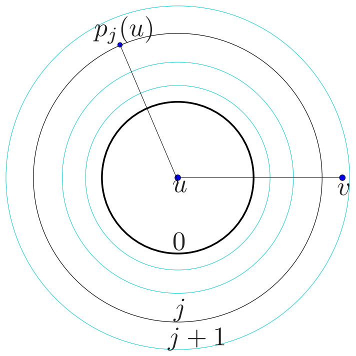

FIG. 4.1. *Analyzing the scheme $\mathcal{A}$ for 3-approximate distance oracle*

See Figure 4.1. Recall that during the construction of $\mathcal{A}(\mathcal{H})$ we compute distances from the vertex $p_j(u)$ to all the vertices in the graph $(V, E_{S_{j+1}} \cup E(p_j(u)))$. Since $\delta(u, p_j(u)) < \delta(u, p_{j+1}(u))$, it follows from Lemma 2.2 that the shortest path between $p_j(u)$ and u is preserved in the graph $(V, E_{S_{j+1}})$. It follows from the definition of j and Lemma 2.2 that the shortest path between u and v is also preserved in the graph $(V, E_{S_{j+1}} \cup E(p_j(u)))$. Using these observations and applying the triangle inequality, the distance $d[p_j(u), v]$ between $p_j(u)$ and v in the graph $(V, E_{S_{j+1}} \cup E(p_j(u)))$ is bounded from above by $\delta(p_j(u), u) + \delta(u,v)$. Combining this inequality with Inequality (4.1), and noting that $d[p_j(u), u] = \delta(u, p_j(u))$, we conclude that

$$d[p_j(u), u] + d[p_j(u), v] \le 3\delta(u,v).$$

$\square$

It follows from Theorem 4.1 that using the scheme $\mathcal{A}(\mathcal{H})$ even with a small size base set S_0, we can retrieve 3-approximate distances from any vertex u to all those vertices v for which $\delta(u,v) \geq \delta(u,S_0)$. Let us call the complement set which is the set of all those vertices whose distance from u is *less* than $\delta(u,S_0)$ as $ball(u,V,S_0)$. This is defined formally as follows.

Definition 4.2. *[18] For a vertex u in a graph $G=(V,E)$, and subsets X,Y of vertices, the set $ball(u,X,Y)$ of vertices is defined as : $ball(u,X,Y) = \{x \in X \mid \delta(u,x) < \delta(u,Y)\}$.*

In order to complete our data structure, it would suffice if we compute for each vertex u the distances to all the vertices of $ball(u,V,S_0)$ and keep these vertices and their distances from u in a hash table, say $B(u)$. Let $\mathcal{D}$ be the data structure thus formed. There are two issues regarding this data structure which need to be addressed now. Firstly, in order to achieve sub-quadratic size of $\mathcal{D}$, we will need a sub-linear bound on the size of $ball(u,V,S_0)$. Secondly, we need an efficient mechanism to compute $ball(u,V,S_0)$ for all $u \in V$. For the sub-quadratic bound on the size, a simple random sampling idea works: if we select each vertex independently with probability q to form the base set S_0, then it follows from Lemma 2.4 that the expected size of $ball(u,V,S_0)$ will be $O(1/q)$. Also note that the expected size of S_0 will be $O(nq)$. Thus the expected space taken by the data structure will be $O(n^2q + n/q)$ which is minimized at $O(n^{3/2})$ for $q = 1/\sqrt{n}$. We will rebuild the data structure if the space of the data structure exceeds twice the expected value. It follows from Markov's Inequality that the expected number of times we need to rebuild the data structure is $O(1)$. This leads to the $O(n^{3/2})$ size data structure. Combining all these ideas, Algorithms 3 and 4 describe respectively the preprocessing of the data structure $\mathcal{D}$ and the way any distance query is answered by it. The reader may notice that similar to the $(2,\mathbf{w})$-APASP algorithm of previous section, the new algorithm is also using the scheme $\mathcal{A}(\mathcal{H})$, but for a different hierarchy $\mathcal{H} = \mathcal{R}(\frac{1}{2},1,k)$.

Algorithm 3: $\mathcal{D}$: a data structure for all-pairs 3-approximate distances

```
$\mathcal{H} \leftarrow \mathcal{R}(\frac{1}{2},1,k)$;
Construct the scheme $\mathcal{A}(\mathcal{H})$;
foreach $u \in V \setminus S_0$ do
    compute vertices of $ball(u,V,S_0)$ and their distances from $u$;
    construct a hash table $B(u)$ storing $ball(u,V,S_0)$.
```

Algorithm 4: Reporting approximate distance between u and v using $\mathcal{D}$

```
if $v \in B(u)$ then
    report the exact distance $\delta(u,v)$
else
    report $\min_{0 \leq i < k}\{d[p_i(u),u] + d[p_i(u),v]\}$ using $\mathcal{A}(\mathcal{H})$.
```

It follows from Lemma 3.1 that the expected time for constructing the scheme $\mathcal{A}(\mathcal{H})$ is $O(\min(m\sqrt{n}, kn^{2+\frac{1}{2k}}))$. The expected time for computing $ball(u,V,S_0)$ for all u is $O(m\sqrt{n})$ as shown in [18]. Thus the expected preprocessing time for the data structure $\mathcal{D}$ is $O(\min(m\sqrt{n}, kn^{2+\frac{1}{2k}}) + m\sqrt{n})$. The query answering time is $O(k)$. We remark that for the hierarchy $\mathcal{H} = \{S_0, \emptyset\}$, the data structure $\mathcal{D}$ becomes the

3-approximate distance oracle of Thorup and Zwick [18] with expected $O(m\sqrt{n})$ preprocessing time. In other words, the 3-approximate distance oracle of Thorup and Zwick is just one special case of the generic data structure $\mathcal{D}$ described above. We further improve the preprocessing time of $\mathcal{D}$. To achieve this goal, in the following section, we describe a modified Dijkstra's algorithm that allows us to construct $ball(u, V, S_0), \forall u \in V$ in expected $O(\min(m\sqrt{n}, n^2))$ time (see Lemma 5.6). This leads to an upper bound of $O(\min(m\sqrt{n}, kn^{2+\frac{1}{2k}}))$ on the preprocessing time of $\mathcal{D}$. We can thus state the following theorem.

THEOREM 4.3. *Any undirected graph on n vertices and m edges can be preprocessed in expected $O(\min(m\sqrt{n},\ kn^{2+\frac{1}{2k}}))$ time to build a data structure that occupies $O(n^{3/2})$ space and answers any distance query with stretch 3 in $O(k)$ time.*

Note that by choosing different values of the parameter k, the data structure $\mathcal{D}$ offers a trade off between query answering time and preprocessing time. For $k = \log n$, we state the following corollary answering the open question of Thorup and Zwick [18].

COROLLARY 4.4. *Any undirected graph on n vertices and m edges can be preprocessed in expected $O(\min(m\sqrt{n}, n^2 \log n))$ time to build a data structure of $O(n^{3/2})$ size which can answer any distance query with stretch 3 in $O(\log n)$ time.*

5. Efficient computation of balls.

In this section we shall describe a faster algorithm for computing $ball(v, X, Y)$ for all vertices $v \in V$ and any given sets $X, Y \subseteq V$. The reader may observe that an $O(mn)$ time algorithm is obvious for this task. However, Thorup and Zwick [18] presented an improved algorithm for this task which achieves sub-cubic running time when the sets X and Y are formed by a suitable random sampling. Our algorithm, which we describe below, can be viewed as a careful refinement of the algorithm of Thorup and Zwick [18]. The starting point is a novel construct called *cluster* which was introduced by Thorup and Zwick [18].

DEFINITION 5.1. *For any two subsets $X, Y \subseteq V$ and a vertex $x \in X$, a* cluster *$C(x, X, Y)$ around x is the set $\{v \in V \mid \delta(v, x) < \delta(v, Y)\}$.*

It is easy to see that $v \in C(x, X, Y)$ if and only if $x \in ball(v, X, Y)$. In other words, balls and clusters are *inverses* of each other. So computing the clusters $C(x, X, Y)$ for all $x \in X$ would suffice to compute $ball(v, X, Y)$ for all $v \in V$. However, computing clusters turns out to be an easier task due to its shortest-paths tree like structure as mentioned in the following simple lemma.

LEMMA 5.2. *[18] For any two subsets $X, Y \subseteq V$ and $x \in X$, if a vertex $v \in V$ is present in $C(x, X, Y)$, then every vertex u lying on the shortest path between x and v is also present in $C(x, X, Y)$.*

Proof. The proof is by contradiction. Given that v belongs to $C(x, X, Y)$, let u be a vertex lying on a shortest path between v and x. The vertex u would not belong to $C(x, X, Y)$ if and only if there is some vertex $w \in Y$ such that $\delta(u, w) \leq \delta(u, x)$. On the other hand, using triangle inequality it follows that $\delta(v, w) \leq \delta(v, u) + \delta(u, w)$. Therefore, $\delta(u, w) \leq \delta(u, x)$ would imply that $\delta(v, w) \leq \delta(v, x)$. In other words, for the vertex v, the vertex $w \in Y$ is not farther than the vertex x. Hence v does not belong to $C(x, X, Y)$ (a contradiction !). $\square$

Lemma 5.2 implies that $C(x, X, Y)$ appears as a *truncated* subtree in the shortest path tree rooted at x. As a result $C(x, X, Y)$ can be computed using a variant of Dijkstra's algorithm as shown by Thorup and Zwick [18]. In order to achieve improved running time, the algorithm in [18] ensures that only those edges are processed whose at least one endpoint belongs to the set $C(x, X, Y)$. This clever restriction ensures a sub-cubic bound on the total time of construction of all clusters (and hence balls) at any level in the approximate distance oracle of Thorup and Zwick [18]. However, it is not good enough to guarantee a quadratic time. Here we present an algorithm which can be viewed as a modified Dijkstra's algorithm to compute $C(x, X, Y)$ and it processes relatively fewer edges of the graph. In particular, this new algorithm will process an edge $(u, v) \in E(u)$ only if

1. u belongs to $C(x, X, Y)$ *and*
2. $\delta(x, u) + \mathbf{w}(u, v) < \delta(v, Y)$, that is, the path $x \rightsquigarrow u \rightarrow v$ is a *witness* for v to be a member of $C(x, X, Y)$.
   (note that there may be more than one witness for the fact that v is in $C(x, X, Y)$.)

In order to avoid processing of those edges that do not satisfy the two conditions mentioned above, our algorithm uses some simple and clever ideas. Before starting the computation of clusters using our modified Dijkstra's algorithm, we build an additional data structure at each vertex $u \in V$ as follows. For each neighbor v of a vertex u, let $\Delta_u(v)$ be defined as $\Delta_u(v) = \delta(v, Y) - \mathbf{w}(u, v)$. For each vertex $u \in V$, we build a binary heap $\mathbf{H}(u)$ storing all its neighbors according to their Δ_u values. Note that $\mathbf{H}(u)$ will be a max-heap with the root storing the neighbor of u with the highest Δ_u value. A binary heap can be built in time of the order of the number of its elements [9]. Therefore, this task will take a total of $O(m)$ time. We would like to state the following observation on binary heaps that holds due to the binary tree like structure implicit in it. We shall use it during the analysis of our modified Dijkstra's algorithm.

Observation 5.1. *Given a binary max-heap and an element x, we can compute all the elements present in it which are greater than x in $O(n_x)$ time where n_x is the number of elements in the heap with values greater than x.*

We now state a simple observation about shortest paths in a graph with positive edge weights. We shall use this observation to prove correctness of our algorithm.

Observation 5.2. *Let $y, z \in V$ be any two vertices such that $\delta(x, y) < \delta(x, z)$. If y and z appear on a shortest path from x to some vertex v, then y must appear before z on this path.*

5.1. A modified Dijkstra's algorithm for computing clusters.

Now we present a new algorithm, namely Modified_Dijkstra(x, Y), that computes all the vertices of a cluster $C(x, X, Y)$. In a manner similar to the original Dijkstra's algorithm, the modified Dijkstra's algorithm begins with a priority queue on labels $\mathcal{L}[v], v \in V$ that are initialized to ∞ for all $v \in V$ before the execution of Modified_Dijkstra(x, Y) for each $x \in X$.

Algorithm 5: Modified_Dijkstra(x, Y), where $x \in X \setminus Y$

```
$\mathcal{L}[x] \longleftarrow 0$;
while $min(\mathcal{L}) \neq \infty$ do
    $u \longleftarrow$ extract_min($\mathcal{L}$);
    Compute all the vertices $v$ from Heap $\mathbf{H}(u)$ for whom $\Delta_u(v) > \mathcal{L}[u]$, and
    store them in a list $A(u)$;
    foreach $v \in A(u)$ do
        $\mathcal{L}[v] \leftarrow \min\{\mathcal{L}[v],\ \mathcal{L}[u] + \mathbf{w}(u, v)\}$
```

We shall now prove a number of key points regarding Modified_Dijkstra(x, Y)

algorithm. These points will prove that this algorithm indeed computes cluster $C(x, X, Y)$, and in addition, they will prove to be very crucial for analyzing its running time. Suppose the set $C(x, X, Y)$ contains j vertices. Without loss of generality, assume that there are no two vertices at the same distance from x in the graph. Consider the sequence $\langle x(= v_1), v_2, \cdots v_j \rangle$ of vertices of $C(x, X, Y)$ arranged in the increasing order of their distances from x. Let Π^i_{xz} be the set of paths from x to z, each of length $< \delta(z, Y)$ such that all the intermediate vertices belong to the set $\{v_1, \cdots, v_i\}$. Let $d(x, z, i)$ denote the length of the shortest path from the set Π^i_{xz}.

Proposition 5.3. *Consider ith iteration of the* **while** *loop executed in Modified_Dijkstra(x, Y). The following assertions hold true.*

$\mathcal{P}_i(1)$. *During ith iteration, v_i is extracted from the queue on $\mathcal{L}$, and $\mathcal{L}[v_i] = \delta(x, v_i)$.*

$\mathcal{P}_i(2)$. *Only those edges $(v_i, v) \in E$ are processed that satisfy the following inequality:*

$$\delta(x, v_i) + \mathbf{w}(v_i, v) < \delta(v, Y)$$

$\mathcal{P}_i(3)$. *At the end of ith iteration, for each $v \in V$, we have $\mathcal{L}[v] = d(x, v, i)$.*

Proof. We provide a proof by induction on i. The initialization of $\mathcal{L}$ ensures that the base case holds trivially. Let the assertions hold true for the $(i-1)$th iteration as well. Now consider the ith iteration.

We shall first prove $\mathcal{P}_i(1)$. It follows from the induction hypothesis that $\mathcal{L}[v_i] = d(x, v_i, i-1)$. Let π_{xv_i} be the shortest path from x to v_i in the given graph. It follows from Lemma 5.2 that all the intermediate vertices of this path belong to the set $\{v_1, \cdots, v_{i-1}\}$. So $d(x, v_i, i-1) = \delta(x, v_i)$. Using the induction hypothesis, the vertices $v_1, \cdots, v_{i-1}$ have already been extracted from the priority queue (on $\mathcal{L}$) of vertices. Also note that for each $\ell > i$, the label $\mathcal{L}[v_\ell]$ is greater than $\delta(x, v_i)$. This can be explained as follows. First it follows from the induction hypothesis that $\mathcal{L}[v_\ell] = d(x, v_\ell, i-1)$ which is at least $\delta(x, v_\ell)$ by definition. Now recall than $\delta(x, v_\ell)$ is greater than $\delta(x, v_i)$ using the assumption that we made at the beginning without loss of generality. So $\mathcal{L}[v_i] = \delta(x, v_i)$ is the smallest label in the priority queue at the beginning of the ith iteration of the **while** loop of Modified_Dijkstra(x, Y). Hence the vertex v_i gets extracted from the priority queue during the ith iteration and $\mathcal{L}(v_i) = \delta(x, v_i)$. So $\mathcal{P}_i(1)$ holds.

To prove assertion $\mathcal{P}_i(2)$, let us analyse the computation performed in the ith iteration after v_i is extracted from the priority queue. We compute the list $A(v_i)$ of all those neighbors v of v_i for whom $\Delta_{v_i}(v) > \mathcal{L}[v_i]$. From the definition of $\Delta_{v_i}(v)$, and the validity of $\mathcal{P}_i(1)$, it can be seen that $\mathcal{P}_i(2)$ holds.

In order to show that $\mathcal{P}_i(3)$ holds at the end of the ith iteration, we shall prove that $\mathcal{L}[v] = d(x, v, i)$ for every vertex v. Consider the set Π^i_{xv}. It follows from the definition that $\Pi^i_{xv} \supseteq \Pi^{i-1}_{xv}$. There are the following two cases : $\Pi^i_{xv} \supset \Pi^{i-1}_{xv}$ or $\Pi^i_{xv} = \Pi^{i-1}_{xv}$. To prove $\mathcal{L}[v] = d(x, v, i)$, we shall analyze these two cases separately. Note that $\Pi^i_{xv} \supset \Pi^{i-1}_{xv}$ if and only if there are one or more paths from x to v of length $< \delta(v, Y)$ which pass through v_i with intermediate vertices from set $\{v_1, \cdots, v_{i-1}\}$. Let π^i_{xv} be the shortest such path. It follows from Observation 5.2 that vertex v_i must appear as the last intermediate vertex on this path. Hence π^i_{xv} will be of the form $x \rightsquigarrow v_i \rightarrow v$. Therefore, $\Pi^i_{xv} \supset \Pi^{i-1}_{xv}$ iff (v_i, v) is an edge and $\delta(x, v_i) + \mathbf{w}(v_i, v) < \delta(v, Y)$. In this case, $d(x, v, i) = \min(d(x, v, i-1),\ \delta(x, v_i) + \mathbf{w}(v_i, v))$, and the reader may verify that the ith iteration updates $\mathcal{L}[v]$ to $d(x, v, i)$. Conversely $\Pi^i_{xv} = \Pi^{i-1}_{xv}$ iff either v is not a neighbor of v_i or $\delta(x, v_i) + \mathbf{w}(v_i, v) \geq \delta(v, Y)$. If either of the two conditions holds, then the ith iteration does not update $\mathcal{L}[v]$. But $\Pi^i_{xv} = \Pi^{i-1}_{xv}$ implies $d(x, v, i) = d(x, v, i-1)$. So in this case the induction hypothesis $\mathcal{P}_{i-1}$ itself ensures that $\mathcal{L}[v] = d(x, v, i-1) = d(x, v, i)$. ∎

Based on Proposition 5.3, we can conclude the following theorem.

THEOREM 5.4. *The algorithm* Modified_Dijkstra(x, Y) *extracts exactly the vertices of the set* $C(x, X, Y)$ *and assigns them their correct distances from* x. *During this computation, it processes an edge* (u, v) *if and only if* $x \in ball(u, X, Y)$ *and the path* $x \overset{\pi_{xu}}{\rightsquigarrow} u \to v$ *is shorter than* $\delta(v, Y)$, *where* π_{xu} *is the* x-u *shortest path.*

5.2. Analyzing the running time of Modified_Dijkstra(x, Y)**.** The time complexity of Modified_Dijkstra(x, Y) is governed by the computation performed in various iterations of its **while** loop. As mentioned in Theorem 5.4, each execution of this loop extracts a vertex from the priority queue which belongs to $C(x, X, Y)$ only. For each $u \in C(x, X, Y)$ extracted, the following computational tasks are executed: computation of the list $A(u)$, followed by updating $\mathcal{L}[v]$ for each $v \in A(u)$. The latter operation is actually a `decrease_key` operation on v. Now recall that $\mathbf{H}(u)$ is a binary max-heap. Therefore, it follows from Observation 5.1 that it takes $O(|A(u)|)$ time to compute $A(u)$ from $\mathbf{H}(u)$. Hence in order to assess the time complexity, we can conclude that Modified_Dijkstra(x, Y) algorithm performs only the following computational tasks. For each $u \in C(x, X, Y)$, one `extract_min` operation, and $|A(u)|$ `decrease_key` operations on the priority queue storing labels $\mathcal{L}[v], v \in V$. If we use a Fibonacci heap to implement this priority queue, it takes $O(\log n)$ time for each `extract_min` operation and $O(1)$ time for each `decrease_key` operation. Let us assign the cost of processing an edge (u, v) to the vertex v. Theorem 5.4 states that if (u, v) is processed, then the path $x \overset{\pi_{xu}}{\rightsquigarrow} u \to v$ is shorter than $\delta(v, Y)$. This implies that $x \in ball(v, X, Y)$ as well as $(u, v) \in E_Y(v)$. Thus a vertex v will be charged $O(|E_Y(v)| + \log n)$ amount of computation cost during Modified_Dijkstra(x, Y) if $x \in ball(v, X, Y)$, and nil otherwise.

THEOREM 5.5. *Given a graph* $G = (V, E)$, *and sets* X, Y *of vertices, we can compute* $ball(v, X, Y)$ *for all* $v \in V$ *by running* Modified_Dijkstra(x, Y) *on each* $x \in X \setminus Y$. *During this computation, each vertex* $v \in V$ *will be charged* $O(|E_Y(v)| + \log n)$ *computation cost once by every vertex in* $ball(v, X, Y)$.

The following lemma is an application of Theorem 5.5, with $X = V$ and Y as a random sample of V.

LEMMA 5.6. *Let* S *contain each vertex of the set* V *independently with probability* $q = 1/\sqrt{n}$. *Then we can compute* $ball(v, V, S), \forall v \in V \setminus S$ *in expected* $O(\min(n^2,\ m\sqrt{n}))$ *time.*

Proof. We compute $ball(v, V, S), \forall v \in V \setminus S$ by executing Modified_Dijkstra(x, S) on each $x \in V$. Note that we also get the distance from v to every $x \in ball(v, V, S)$ from this computation. Let us refer to Theorem 5.5 for analyzing the total computation cost (time) incurred in this task. It follows from Theorem 5.5 that $O(|E_S(v)| + \log n)$ computation cost will be charged to v by each vertex in $ball(v, V, S)$. So the expected cost charged to v is $O(\mathbf{E}[(|E_S(v)| + \log n) \cdot |ball(v, V, S)|])$. Observe that $|E_S(v)| \le \deg(v)$ and $\mathbf{E}[|ball(v, V, S)|] = 1/q = \sqrt{n}$ using Lemma 2.4. Hence the cost charged to v is bounded by $O(\deg(v)\sqrt{n})$ (ignoring the additive $\sqrt{n}\log n$ term).

We shall now provide a tighter analysis showing that expected cost charged to any vertex is bounded by $O(n)$, irrespective of the large degree that the vertex v may have. First note that all the neighbors of v corresponding to the edges of $E_S(v)$ are present in $ball(v, V, S)$. So $|E_S(v)| \le |ball(v, V, S)|$ always holds. Hence the expected cost charged to v is bounded from above by $\mathbf{E}[|ball(v, V, S)|^2]$. To calculate $\mathbf{E}[|ball(v, V, S)|^2]$, consider the sequence $v_1, v_2, v_3, \ldots, v_{n-1}$ of vertices of the given graph arranged in non-decreasing order of their distances from v. It can be observed that $|ball(v, V, S)| = i$ if none of $v_1, \cdots, v_i$ is selected in S, but v_{i+1} is selected in

S. The probability associated with this event is $(1-q)^i q$. Hence we can bound $\mathbf{E}[|ball(v,V,S)|^2]$ as follows:

$$\begin{aligned}
\mathbf{E}[|ball(v,V,S)|^2] &= \sum_{i=1}^{i=n-1} i^2(1-q)^i q \quad \le \quad q\sum_{i=1}^{i=\infty} i^2(1-q)^i \\
&= q\frac{(1+(1-q))(1-q)}{(1-(1-q))^3} \qquad \left\{\text{using } \sum_{i\ge 1} i^2x^i = \frac{(1+x)x}{(1-x)^3}\right\} \\
&\le q\frac{2}{q^3} = \frac{2}{q^2} = 2n \qquad \{\text{since } q = 1/\sqrt{n}\}
\end{aligned}$$

Hence the expected cost charged to vertex v is $O(\min(n,\ \deg(v)\sqrt{n}))$. So the expected cost incurred in the computation of $ball(v,V,S), \forall v \in V \setminus S$ is $O(\min(n^2,\ m\sqrt{n}))$. ∎

6. A $(\mathbf{2k-1})$-approximate distance oracle in expected $\mathbf{O}(n^2)$ time. In this section we describe the construction of a $(2k-1)$-approximate distance oracle for $k > 2$ which achieves expected $O(\min(kmn^{1/k}, n^2))$ preprocessing time and $O(k)$ query time. It is essentially the same as the oracle construction of Thorup and Zwick [18] except that we use a faster preprocessing algorithm for building the data structure, and a slightly more involved analysis of the query answering algorithm.

We first provide a brief overview of the $(2k-1)$-approximate distance oracle of Thorup and Zwick [18]. Its construction employs the hierarchy $\mathcal{R}(0,1,k)$ of subsets of vertices. The oracle keeps a hash table $B(v)$ for every vertex $v \in V$. The hash table $B(v)$ stores all the vertices of the set $\cup_{i\le k} ball(v, S_{i-1}, S_i)$ and their distances from v. Recall that in the hierarchy $\mathcal{R}(0,1,k)$, the subset S_i is formed by selecting each vertex of the set S_{i-1} independently with probability $n^{-1/k}$. So it follows from Lemma 2.4 that the expected size of $ball(v, S_{i-1}, S_i)$ is at most $O(n^{1/k})$. Hence, the hash table $B(v)$ occupies expected $O(kn^{1/k})$ space, and so the expected size of the approximate distance oracle is $O(kn^{1+1/k})$.

It can be observed that the main task involved in the preprocessing of the $(2k-1)$-approximate distance oracle is to compute, for each vertex v, distances to vertices of k ball centered around it, that is, $ball(v, S_{i-1}, S_i)$ for $1 \le i \le k$. Recall that for any $i \le k$, computing $ball(v, S_{i-1}, S_i)$ for all $v \in V$ turns out to be equivalent to computing clusters $C(w, S_{i-1}, S_i)$ for all $w \in S_{i-1}$. In the previous section, we designed a Modified_Dijkstra algorithm which computes these clusters efficiently. As a careful reader might have noticed, the key idea underlying Modified_Dijkstra algorithm is that it processes only those edges whose both the endpoints are present in $C(w, S_{i-1}, S_i)$ which is actually a *truncated* shortest path tree rooted at w provided $S_i \ne \emptyset$. This feature and the randomization underlying the hierarchy $\mathcal{R}(0,1,k)$ will help Modified_Dijkstra algorithm achieve improved preprocessing time for computing distances from each $v \in V$ to vertices belonging to its inner $k-1$ balls, that is, $\cup_{i<k} ball(v, S_{i-1}, S_i)$. However, it does not provide any improvement in the preprocessing time for the kth balls. This is because a cluster $C(w, S_{k-1}, S_k)$ for $w \in S_{k-1}$ corresponds to a full shortest path tree rooted at w since $S_k = \emptyset$. Therefore, computing the kth ball around each vertex amounts to computing a full shortest path tree from each $w \in S_{k-1}$, and there is no way to accomplish this task in less than expected $O(mn^{1/k})$ time since $\mathbf{E}[|S_{k-1}|] = O(n^{1/k})$. This preprocessing time could be $\Theta(n^{2+1/k})$ for dense graphs. To overcome this hurdle, we first compute a subgraph which is *sparse* and still preserves approximate distances pairwise. Then we execute Dijkstra's algorithm from each $w \in S_{k-1}$ in this subgraph to compute approximate

distance to each $v \in V$. As a result, each vertex $v \in V$ will store only *approximate* distances to vertices of its kth ball. This is contrary to the approximate distance oracle of Thorup and Zwick [18]. However, interestingly, the same query algorithm of Thorup and Zwick [18], but with a more careful analysis, still guarantees $(2k-1)$ stretch for each query.

6.1. Preprocessing algorithm. The new preprocessing algorithm employs a sparse subgraph whose edge set also contains a 3-spanner. A 3-spanner of a weighted graph is a subgraph such that the distance between any two vertices $u, v \in V$ in the subgraph is at most $3\delta(u,v)$. We use the following result concerning 3-spanners.

THEOREM 6.1. *[5] An undirected weighted graph on n vertices and m edges can be processed in expected $O(m)$ time to compute its 3-spanner which has $O(min(m, n^{3/2}))$ edges.*

Algorithm 6: Faster construction of a $(2k-1)$-approximate distance oracle

```
Let $S_0, \dots, S_k$ be the respective $k+1$ subsets of the hierarchy $\mathcal{R}(0,1,k)$;
foreach $i \in [1, k-1]$ and $v \in V$ do  compute $p_i(v)$ and $\delta(v, p_i(v))$;
foreach $i \in [1, k-1]$ and $w \in S_{i-1} \backslash S_i$ /* computing inner $k-1$ balls */
do
    run Modified_Dijkstra($w, S_{i-1}, S_i$) to compute $C(w, S_{i-1}, S_i)$
Compute a 3-spanner $(V, \mathcal{E}_S)$ of the given graph $G$;
foreach $x \in S_{k-1}$                              /* computing $k$th ball */
do
    Run Dijkstra's algorithm from $x$ in the subgraph $(V, E_{S_{k-1}} \cup E(x) \cup \mathcal{E}_S)$
```

Consider the approximate distance oracle computed by the preprocessing Algorithm 6. For the hash table $B(v)$ associated with v in this oracle, we shall use $d[v,x]$ to denote the (approximate/exact) distance between v and x. The following lemma holds directly from Algorithm 6.

LEMMA 6.2. *If $w \in B(v)$, then*

$$d[v,w] = \begin{cases} \delta(v,w) & \text{if } w \in S_i \textit{ for some } i < k-1 \\ \hat{\delta}(v,w) & \text{if } w \in S_{k-1} \end{cases}$$

where $\hat{\delta}(v,w)$ is the distance between v and w in the graph $(V, E_{S_{k-1}} \cup E(w) \cup \mathcal{E}_S)$.

Let us analyse the time complexity of the new preprocessing algorithm. First consider the task involving the computation of shortest-path trees from each $x \in S_{k-1}$ in the subgraph $(V, E_{S_{k-1}} \cup E(x) \cup \mathcal{E}_S)$. Note that a vertex $v \in V$ belongs to S_{k-1} with probability $n^{-(1-1/k)}$, so it follows from Lemma 2.3 that expected size of $E_{S_{k-1}}$ is $O(n^{2-1/k})$. Using Theorem 6.1, it follows that the 3-spanner $\mathcal{E}_S$ will have $O(n^{3/2})$ edges and can be computed in expected $O(m)$ time. Hence the expected number of edges in the subgraph $(V, E_{S_{k-1}} \cup E(x) \cup \mathcal{E}_S)$ will be $O(\min(m,\ n^{2-1/k}))$. The expected size of S_{k-1} will be $O(n^{1/k})$, and it has negative correlation with $|E_{S_{k-1}}|$. Therefore, it can be shown using elementary probability that it will take expected $O(\min(mn^{1/k},\ n^2))$ time to compute distances between each vertex v and each vertex of S_{k-1}. In the following lemma we provide a tighter bound on the expected time required to compute $ball(v, S_{i-1}, S_i), \forall v \in V$ for a given $i < k$. The proof of this lemma is a generalization of Lemma 5.6 with a few more technical details, therefore, to maintain continuity of the flow at this point, its proof is deferred to the end of this section.

Lemma 6.3. *For a given $i < k$, it takes expected $O(\min(n^{\frac{k+i+1}{k}},\ mn^{1/k}))$ time to compute $ball(v, S_{i-1}, S_i), \forall v \in V$.*

It follows from Lemma 6.3 and the preceding discussion that the expected preprocessing time of the approximate distance oracle is $O(\min(n^2,\ kmn^{1/k}))$. As discussed earlier, the expected size of the oracle is $O(kn^{1+1/k})$. We rebuild the oracle if its size exceeds twice this bound, and using Markov's inequality, the probability of this event will be at most 1/2. So the expected number of times the oracle has to be rebuilt till we get an oracle of size $O(kn^{1+1/k})$ will be $O(1)$. Thus an $O(n^{1+1/k})$ size approximate distance oracle can be built in expected $O(\min(n^2,\ kmn^{1/k}))$ time. The only issue that needs to be addressed is the analysis of the query algorithm.

6.2. Reporting approximate distance with stretch at most $(2k-1)$. We present below the query answering algorithm of Thorup and Zwick [18]. The analysis

Algorithm 7: $\mathcal{Q}(u, v)$: Reporting approximate distance between u and v

```
$i \leftarrow 1$;
while $p_{i-1}(u) \notin B(v)$ do
    swap$(u, v)$;
    $i \leftarrow i + 1$;
return    $d[u, p_{i-1}(u)] + d[v, p_{i-1}(u)]$;
```

of the above query answering algorithm is slightly different due to the slight dissimilarity in the construction of the new approximate distance oracle. First, we provide the underlying intuition of the query answering procedure before we formally analyse it.

Let u and v be any two vertices whose approximate distance is to be reported. In order to explain the query answering procedure, we introduce the following notation.

For a pair of vertices u and v, a vertex w is said to be *t-near* to u if $\delta(u, w) \leq t\delta(u, v)$. It follows from the simple triangle inequality that if a vertex is t-near to u then it is $(t+1)$-near to v also.

The entire query answering process is to search for a vertex which is t-near to u for some $t < k$, and whose distance to both u and v is known. This search is performed iteratively as follows : the ith iteration begins with a vertex $w \in S_{i-1}$ which is $(i-1)$-near to u (and hence i-near to v) and its distance from u is known. It is determined whether or not its distance to v is known. We do so by checking whether or not $w \in B(v)$. Note that $w \notin B(v)$ if and only if $w \notin ball(v, S_{i-1}, S_i)$. Using Definition 4.2, therefore, it follows that $w \notin B(v)$ if and only if, for the vertex v, the vertex $p_i(v)$ is equidistant or nearer than the vertex w. But this would imply that $p_i(v)$ is also i-near to v, and note that its distance from v is known. Therefore, if the distance $\delta(v, w)$ is not known, we continue in the next iteration with vertex $w = p_i(v)$, and swap the roles of u and v. In this way we proceed gradually, searching over the sets $S_0, \ldots, S_{k-1}$. We are bound to find such a vertex within at most k iterations since the (approximate) distance from S_{k-1} is known to all the vertices.

Based on the discussion above and the query answering algorithm $\mathcal{Q}(u, v)$, the following lemma holds.

LEMMA 6.4. *[18] At the beginning of the ith iteration of the* **while** *loop,* $\delta(u, p_{i-1}(u)) \leq (i-1)\delta(u,v)$.

The following theorem provides a guarantee of $2k-1$ on the worst case stretch for any distance query.

THEOREM 6.5. *The algorithm $\mathcal{Q}(u,v)$ reports in $O(k)$ time a $(2k-1)$-approximate distance between u and v, for any $k > 2$.*

Proof. The query answering algorithm performs iterations of the **while** loop until the condition of the loop fails. As mentioned above, there will be at most $(k-1)$ successful iterations before the condition of the loop fails. Let the condition fail at the beginning of the ith iteration. So the following inequality follows from Lemma 6.4 when we exit the **while** loop:

$$\delta(u, p_{i-1}(u)) \leq (i-1)\delta(u,v). \tag{6.1}$$

We analyze the distance reported by the oracle on the basis of the final value of i.
Case 1 : $i < k$.
In this case, using Lemma 6.2, the reported distance is $\delta(u, p_{i-1}(u)) + \delta(v, p_{i-1}(u))$, which using the triangle inequality is at most $2\delta(u, p_{i-1}(u)) + \delta(u,v)$. Using Inequality (6.1), this distance is at most $(2i-1)\delta(u,v)$.
Case 2 : $i = k$.
Note that each vertex u stores its exact distance to $p_i(u)$ for each $i < k$. So, it follows from Lemma 6.2 that the distance reported will be $\delta(u, p_{k-1}(u)) + \hat{\delta}(v, p_{k-1}(u))$. Let us analyze the following two sub-cases as shown in Figure 6.1. The analysis will exploit the equality $\hat{\delta}(u, p_{k-1}(u)) = \delta(u, p_{k-1}(u))$ which is implied by Lemma 2.2.

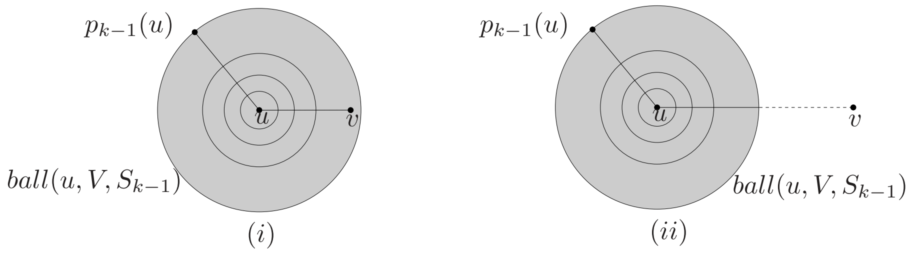

FIG. 6.1. *The two sub-cases depending upon whether $v \in ball(u,V,S_{k-1})$.*

In sub-case (i), the vertex v belongs to $ball(u,V,S_{k-1})$. Hence $\delta(u,v) < \delta(u, p_{k-1}(u))$. So it follows from Lemma 2.2 that $\hat{\delta}(v,u) = \delta(v,u)$. So using the triangle inequality,

$$\hat{\delta}(v, p_{k-1}(u)) \leq \hat{\delta}(v,u) + \hat{\delta}(u, p_{k-1}(u)) \leq \delta(v,u) + \delta(u, p_{k-1}(u)).$$

Inequality (6.1) implies that $\delta(u, p_{k-1}(u))$ is at most $(k-1)\delta(u,v)$, hence the distance reported by the oracle in this sub-case is at most $(2k-1)\delta(u,v)$.

In sub-case (ii), the vertex v does not belong to $ball(u,V,S_{k-1})$. (This is the main point where our analysis differs from that of Thorup and Zwick [18].) It follows that $\delta(u, p_{k-1}(u)) \leq \delta(u,v)$. Since $(V, \mathcal{E}_S)$ is a 3-spanner, $\hat{\delta}(v,u) \leq 3\delta(v,u)$. Combining these two inequalities and using the triangle inequality, it follows that

$$\hat{\delta}(v, p_{k-1}(u)) \leq \hat{\delta}(v,u) + \hat{\delta}(u, p_{k-1}(u)) \leq 3\delta(v,u) + \delta(u, p_{k-1}(u)) \leq 4\delta(v,u).$$

Hence, in this sub-case, the distance reported by the oracle will be at most $5\delta(u,v)$. For every $k > 2$, this stretch, which is 5, is always bounded by $(2k-1)$. ◻

We can conclude with the following theorem now.

THEOREM 6.6. *An undirected weighted graph on n vertices and m edges can be preprocessed in expected $O(\min(n^2,\ kmn^{1/k}))$ time to compute a $(2k-1)$-approximate distance oracle of size $O(kn^{1+1/k})$, for any integer $k > 2$.*

6.3. Proof of Lemma 6.3. The proof will employ a couple of observations concerning $ball(v, X, Y)$ when the sets X and Y are formed by suitable random sampling. These observations are formalized in the two lemmas which we prove first. Here, we introduce an additional notation $\mathcal{S}_u^v$ defined for any two vertices u, v as follows.
$\mathcal{S}_u^v$: the set of vertices in the graph that lie within distance $\delta(u, v)$ from v.

The first lemma gives a bound on $\mathbf{Pr}[u \in ball(v, X, Y)]$ in terms of $\mathbf{Pr}[u \in X]$ and $\mathbf{Pr}[u \in ball(v, V, Y)]$.

LEMMA 6.7. *Let X be formed by selecting each vertex from V independently with probability q, and let Y be formed by selecting each vertex of X independently with probability q'. For any $u, v \in V$,*

$$\mathbf{Pr}[u \in ball(v, X, Y)] \leq \mathbf{Pr}[u \in X] \cdot \mathbf{Pr}[u \in ball(v, V, Y)]$$

Proof. It follows from Definition 4.2 that vertex u belongs to $ball(v, X, Y)$ if and only if $u \in X$ and none of the vertices of set $\mathcal{S}_u^v$ is present in Y. Now whether or not a vertex belongs to sets X or Y is independent of any other vertices. Because of this independence incorporated in selecting vertices to form sets X and Y, we can observe that

$$\begin{aligned}\mathbf{Pr}[u \in ball(v, X, Y)] &= \mathbf{Pr}[u \notin Y \wedge u \in X] \cdot \prod_{w \in \mathcal{S}_u^v \setminus \{u\}} \mathbf{Pr}[w \notin Y] \\ &= \mathbf{Pr}[u \in X] \cdot \mathbf{Pr}[u \notin Y | u \in X] \cdot \prod_{w \in \mathcal{S}_u^v \setminus \{u\}} \mathbf{Pr}[w \notin Y]\end{aligned}$$

Now note that $\mathbf{Pr}[u \notin Y | u \in X] = (1 - q') \leq (1 - qq') = \mathbf{Pr}[u \notin Y]$. Hence

$$\begin{aligned}\mathbf{Pr}[u \in ball(v, X, Y)] &\leq \mathbf{Pr}[u \in X] \cdot \mathbf{Pr}[u \notin Y] \cdot \prod_{w \in \mathcal{S}_u^v \setminus \{u\}} \mathbf{Pr}[w \notin Y] \\ &= \mathbf{Pr}[u \in X] \cdot \prod_{w \in \mathcal{S}_u^v} \mathbf{Pr}[w \notin Y] \\ &= \mathbf{Pr}[u \in X] \cdot \mathbf{Pr}[u \in ball(v, V, Y)]\end{aligned}$$

□

Now we prove the following lemma that essentially states that conditioned on the event $u \in ball(v, V, Y)$, the probability that some vertex u' also belongs to $ball(v, V, Y)$ is independent of whether or not u was present in X.

LEMMA 6.8. *Let X be a set formed by selecting each vertex of the given graph independently with some probability, and let Y be formed by selecting each vertex from X independently with some probability. For any $u, u', v \in V$,*

$$\mathbf{Pr}[u' \in ball(v, V, Y) | u \in ball(v, X, Y)] = \mathbf{Pr}[u' \in ball(v, V, Y) | u \in ball(v, V, Y)]$$

Proof. First note that $u \in ball(v, X, Y)$ if and only if $u \in X$ and $u \in ball(v, V, Y)$. So if $u' \in \mathcal{S}^v_u$, then $\mathbf{Pr}[u' \in ball(v, V, Y)|u \in ball(v, X, Y)]$ as well as $\mathbf{Pr}[u' \in ball(v, V, Y)|u \in ball(v, V, Y)]$ are 1. Let us now analyze the case when $u' \notin \mathcal{S}^v_u$. First note that $u \in ball(v, X, Y)$ implies that none of the vertices of the set $\mathcal{S}^v_u$ is selected in Y. So conditioned on the event "$u \in ball(v, X, Y)$", the vertex u' will belong to $ball(v, V, Y)$ iff none of the vertices from $\mathcal{S}^v_{u'} \setminus \mathcal{S}^v_u$ is present in set Y. The latter event is independent of $u \in X$ since a vertex from set $V \setminus \{u\}$ is selected in Y independent of whether or not $u \in X$. Hence,

$$\begin{aligned}\mathbf{Pr}[u' \in ball(v, V, Y)|u \in ball(v, X, Y)] &= \prod_{w \in \mathcal{S}^v_{u'} \setminus \mathcal{S}^v_u} \mathbf{Pr}[w \notin Y] \\ &= \mathbf{Pr}[u' \in ball(v, V, Y)|u \in ball(v, V, Y)]\end{aligned}$$

□

Based on Lemma 6.8, the following corollary can be stated.

COROLLARY 6.9. *Let X be formed by selecting each vertex from V independently with some probability, and let Y be formed by selecting each vertex of X independently with some probability. For any vertices $u, v \in V$,*

$$\mathbf{E}[|ball(v, V, Y)| \mid u \in ball(v, X, Y)] = \mathbf{E}[|ball(v, V, Y)| \mid u \in ball(v, V, Y)].$$

We are now ready to prove Lemma 6.3. We compute $ball(v, S_{i-1}, S_i), \forall v \in V$ by executing Modified_Dijkstra(x, S_i) on each $x \in S_{i-1} \setminus S_i$. Theorem 5.5 implies that it suffices to show the following. The expected computation cost charged to a vertex v during the executions of Modified_Dijkstra(x, S_i) for all $x \in S_{i-1} \setminus S_i$ is bounded by $O(\min(n^{\frac{i+1}{k}},\ \deg(v)n^{1/k}))$. Let W_v denote the random variable for the computation cost charged to v. A vertex will charge $E_{S_i}(v)$ to vertex v iff it belongs to $ball(v, S_{i-1}, S_i)$ and 0 otherwise. So the expected cost charged to v is

$$\mathbf{E}[W_v] = \mathbf{E}[|E_{S_i}(v)| \cdot |ball(v, S_{i-1}, S_i)|]. \tag{6.2}$$

Since $|E_{S_i}(v)| \le \deg(v)$, we see that $\mathbf{E}[W_v]$ is bounded by $\deg(v)\mathbf{E}[|ball(v, S_{i-1}, S_i)|]$. It follows from Lemma 2.4 that $\mathbf{E}[|ball(v, S_{i-1}, S_i)|] = n^{1/k}$. Hence the expected cost charged to v is bounded by $O(\deg(v)n^{1/k})$. Now we shall show that it is bounded by $O(n^{\frac{i+1}{k}})$ as well.

Note that all the neighbors of v corresponding to the edges $E_{S_i}(v)$ are present in $ball(v, V, S_i)$, hence $|E_{S_i}(v)| \le |ball(v, V, S_i)|$ always. Using this bound and Equation (6.2), we bound $\mathbf{E}[W_v]$ as follows.

$$\begin{aligned}\mathbf{E}[W_v] &\le \mathbf{E}[|ball(v, V, S_i)| \cdot |ball(v, S_{i-1}, S_i)|] \\ &= \sum_{\forall v' \in V} \mathbf{E}[|ball(v, V, S_i)| \ \mid \ v' \in ball(v, S_{i-1}, S_i)]\ \mathbf{Pr}[v' \in ball(v, S_{i-1}, S_i] \\ &\le \sum_{\forall v' \in V} \mathbf{E}[|ball(v, V, S_i)| \ \mid \ v' \in ball(v, S_{i-1}, S_i)] n^{\frac{-i+1}{k}} \mathbf{Pr}[v' \in ball(v, V, S_i)] \\ &= n^{\frac{-i+1}{k}} \sum_{\forall v' \in V} \mathbf{E}[|ball(v, V, S_i)| \ \mid \ v' \in ball(v, V, S_i)] \mathbf{Pr}[v' \in ball(v, V, S_i)] \\ &= n^{\frac{-i+1}{k}} \mathbf{E}[|ball(v, V, S_i)|^2]\end{aligned}$$

The first equality follows from linearity of expectation. The second inequality above follows from Lemma 6.7 and the third inequality uses Corollary 6.9. Using arguments similar to those used in Lemma 5.6, it follows that $\mathbf{E}[|ball(v, V, S_i)|^2] = 1/q^2$ with $q = n^{\frac{-i}{k}}$. So $\mathbf{E}[W_v] \leq n^{\frac{i+1}{k}}$ as well.

7. All Pairs Approximate Shortest Paths with Stretch < 3.

In this section we present a new hierarchical scheme for the APASP problem, based on which we present algorithms to compute all-pairs distance estimates whose stretch is less than 3. This scheme is obtained by a suitable augmentation of the scheme $\mathcal{A}$, and will be denoted by $\hat{\mathcal{A}}$.

7.1. An augmented hierarchical scheme $\hat{\mathcal{A}}$ for APASP.

Recall that for the case of $(2, \mathbf{w})$-APASP and the 3-approximate distance oracle, the scheme $\mathcal{A}$ is applied on the randomized hierarchy $\mathcal{R}(\alpha, \beta, k)$ of subsets of vertices with suitable values of α and β. In order to design efficient algorithms for stretch less than 3, we shall use $\hat{\mathcal{A}}$ on a similar randomized hierarchy $\hat{\mathcal{R}}(0, \beta, k)$. This hierarchy of subsets of vertices is obtained by a small modification of the hierarchy $\mathcal{R}(0, \beta, k)$. The following lemma due to Thorup and Zwick [17] plays an important role here.

LEMMA 7.1. *[17] Given a weighted graph $G = (V, E)$ and $0 < q < 1$, we can compute a set $S \subset V$ of size $O(nq \log n)$ in expected time $O(m/q \log n)$ such that $|C(x, V, S)| = O(1/q)$ for each $x \in V$.*

We now define $\hat{\mathcal{R}}(0, \beta, k)$ as follows.

DEFINITION 7.2. *Let $S_0, \ldots, S_k$ be the respective $k+1$ vertex sets of the hierarchy $\mathcal{R}(0, \beta, k)$. Let $S \subset V$ be a set of size $O(n^{1-\beta})$ such that $|C(x, V, S)| = O(n^{\beta} \log n)$ for each $x \in V$. (We can use Lemma 7.1 to construct it.)*

$$\text{For each } i \in [0, k]: \qquad S_i \longleftarrow S_i \cup S$$

The hierarchy of augmented sets $\{S_i\}$ defines $\hat{\mathcal{R}}(0, \beta, k)$.

The augmented scheme $\hat{\mathcal{A}}$ when applied on the hierarchy $\hat{\mathcal{R}}(0, \beta, k)$ will compute an $n \times n$ table d storing approximate distances. Our algorithms for 2-APASP and 7/3-APASP will refine this table further. The table d is initialized as the adjacency matrix of the given graph. That is,

$$\forall x, y \in V \quad d[x, y] \longleftarrow \begin{cases} \mathbf{w}(x, y) & \text{if } (x, y) \in E \\ \infty & \text{otherwise} \end{cases}$$

The construction of this scheme is described in Algorithm 8.

The edge set $E_{S_{i+1}} \cup d[s, \_]$ refers to edges in the set $E_{S_{i+1}}$ along with edges (s, x) (of weight $d[s, x]$) to each $x \in V$. The reason for introducing such additional *weighted* edges from s is that, if s is already aware of a certain distance estimate $d[s, x]$ to vertex x, then we would like to use this estimate to refine the distance estimates from s to other vertices y.

The augmented scheme $\hat{\mathcal{A}}$ performs two tasks of distance computation (shown shaded in Algorithm 8) in addition to the tasks performed by the scheme $\mathcal{A}$. The first task is the computation of $ball(u, V, S_k)$ wherein we compute distances from u to all vertices in $ball(u, V, S_k)$. In the second task, we compute approximate distances from $p_i(u)$ to even those vertices y that may lie outside $ball(u, V, S_k)$ but have at least one neighbor in $ball(u, V, S_k)$. We can thus state the following observation which plays a crucial role in our algorithms for 2-APASP and 7/3-APASP described in the following subsections.

Algorithm 8: Augmented scheme $\hat{\mathcal{A}}$ for a hierarchy $\hat{\mathcal{R}}(0,\beta,k)$

```
foreach $i \in [0,k]$ and $u \in V$ do
    compute $p_i(u)$;    $d[u,p_i(u)] \longleftarrow \delta(u,p_i(u))$;
foreach $u \in V \setminus S_k$ do
    compute $ball(u,V,S_k)$;
    foreach $x \in ball(u,V,S_k)$ and $0 \le i \le k$ do
        foreach neighbor $y$ of $x$ do
            $d[p_i(u),y] \longleftarrow \min \begin{cases} d[p_i(u),y] \\ d[u,p_i(u)] + d[u,x] + \mathbf{w}(x,y) \end{cases}$
foreach $i \in [0,k-1]$ and $s \in S_i$ do
    run Dijkstra's algorithm from $s$ in the subgraph $(V, E_{S_{i+1}} \cup d[s,\_])$ and
    update $d[s,\_]$ accordingly.
```

Observation 7.1. *Let y be a vertex and P_{uy} be a shortest path from u to y. If y or a neighbor of y lying on the path P_{uy} belongs to $ball(u,V,S_k)$, then for any $0 \le i \le k$, it follows from Algorithm 8 that $d[p_i(u),y] \le \delta(u,p_i(u)) + \delta(u,y)$.*

Theorem 7.3. *Given a weighted graph $G=(V,E)$, let d be the approximate distance matrix computed by the augmented scheme $\hat{\mathcal{A}}$ for hierarchy $\hat{\mathcal{R}}(0,\beta,k)$. Let u and v be any two vertices. If $\delta(u,p_k(u)) + \delta(v,p_k(v)) > \delta(u,v)$, then the following assertion holds:*

$$\min_{0\le i<k}\{d[u,p_i(u)] + d[p_i(u),v], d[v,p_i(v)] + d[p_i(v),u]\} \le 2\delta(u,v).$$

Proof. Let $i<k$ be such that $\delta(u,p_i(u))+\delta(v,p_i(v)) \le \delta(u,v)$ but $\delta(u,p_{i+1}(u))+\delta(v,p_{i+1}(v)) > \delta(u,v)$. Such a unique i must exist since $\delta(u,p_0(u)) = \delta(v,p_0(v)) = 0$ and it is given that $\delta(u,p_k(u)) + \delta(v,p_k(v)) > \delta(u,v)$. Without loss of generality assume that $\delta(u,p_i(u)) \le \delta(v,p_i(v))$. Since $d[u,p_i(u)]$ is set to $\delta(u,p_i(u))$, so $2d[u,p_i(u)] \le \delta(u,v)$ holds. Now consider the shortest path P_{uv} between u and v in the original graph. Let y be the first vertex on this path in the direction from u to v such that $\delta(u,y) \ge \delta(u,p_{i+1}(u))$. Since $i+1 \le k$, the vertex y fulfills the conditions of Lemma 7.1, hence

$$d[p_i(u),y] \le \delta(u,p_i(u)) + \delta(u,y). \tag{7.1}$$

Also note that $\delta(v,y) < \delta(v,p_{i+1}(v))$. This is because $\delta(u,y) \ge \delta(u,p_{i+1}(u))$ and $\delta(u,p_{i+1}(u))+\delta(v,p_{i+1}(v)) > \delta(u,v)$. So it follows from Lemma 2.2 that the subpath P_{yv} is present in $E_{S_{i+1}}$. Therefore, at the end of Algorithm 8, when we execute Dijkstra's algorithm from $p_i(u)$ in the subgraph $(V, E_{S_{i+1}} \cup d[p_i(u)])$, the approximate distance $d[p_i(u),v]$ that is computed is at most

$$\begin{aligned} d[p_i(u),v] &\le d[p_i(u),y] + \delta(y,v) \\ &\le \delta(u,p_i(u)) + \delta(u,y) + \delta(y,v) \quad \{\text{using Equation (7.1)}\} \\ &= \delta(u,p_i(u)) + \delta(u,v). \end{aligned}$$

Since we assumed, without loss of generality, in the beginning that $2d[u,p_i(u)] \le \delta(u,v)$, it follows that

$$d[u,p_i(u)] + d[p_i(u),v] \le 2d[u,p_i(u)] + \delta(u,v) \le 2\delta(u,v).$$

□

We now analyze the time complexity for constructing the augmented scheme $\hat{\mathcal{A}}$ for $\hat{\mathcal{R}}(0, \beta, k)$. The task of computing the set $S \subset V$ such that cluster $C(x, V, S)$ at each $x \in V$ has $O(n^\beta \log n)$ vertices can be accomplished in expected $O(mn^\beta)$ time as shown by Thorup and Zwick [17]. The construction of $ball(u, V, S_k)$ for all v will be performed in expected $O(mn^\beta)$ time (apply Lemma 5.6 with $q = n^{-\beta}$). The remaining computational task of the shaded portion of Algorithm 8 will be performed in $O(\sum_{x \in V} |C(x, V, S_k)| deg(x))$ time, which is $\tilde{O}(mn^\beta)$. The last step of Algorithm 8 involves the execution of Dijkstra's algorithm from vertices of the hierarchy and will take expected $O(n^2 \log n)$ time using Lemma 3.1. Combined together, the total expected time complexity of $\hat{\mathcal{A}}$ for $\hat{\mathcal{R}}(0, \beta, k)$ is $\tilde{O}(mn^\beta + n^2)$.

Lemma 7.4. *The preprocessing time of the scheme $\hat{\mathcal{A}}$ built for $\hat{\mathcal{R}}(0, \beta, k)$ is $\tilde{O}(mn^\beta + n^2)$.*

In our algorithm for 7/3-APASP, we shall use one more observation regarding the augmented scheme. Recall that we approximate the distance from $p_i(u)$ to those vertices y that have a neighbor in $ball(u, V, S_k)$. In a similar fashion it is possible to approximate the distance from $p_k(x)$, where x belongs to $ball(u, V, S_k)$, to neighbors of u. For this we have to replace the innermost **for** loop of the shaded portion of Algorithm 8 by the following **for** loop:

```
for each neighbor $y$ of $u$ do
        $d[p_k(x), y] \longleftarrow \min(d[p_k(x), y], d[p_k(x), x] + d[u, x] + \mathbf{w}(u, y))$
```

This additional task can be accomplished in expected $O(|ball(u, V, S_k)| deg(u))$ time per vertex u, which amounts to a total expected $O(mn^\beta)$ time only. Thus this task does not increase the asymptotic time complexity of Algorithm 8. We can now state the following important observation which follows from this additional task.

Observation 7.2. *For each vertex w belonging to $ball(z, V, S_k)$ and any neighbor y of z, the augmented scheme $\hat{\mathcal{A}}$ ensures that $d[p_k(w), y]$ is bounded by $\delta(p_k(w), w) + \delta(w, z) + \mathbf{w}(z, y)$.*

7.2. Algorithm for all-pairs 2-approximate shortest paths. Theorem 7.3 immediately leads to Algorithm 9 for computing all-pairs stretch 2 distances.

Algorithm 9: An algorithm for 2-approximate distances.

```
Compute the scheme $\hat{\mathcal{A}}(\mathcal{H})$ for $\mathcal{H} = \hat{\mathcal{R}}(0, \frac{1}{2}, k)$ with $k = \log n$;
$S_{k+1} \longleftarrow \emptyset$;
foreach $s \in S_k$ do
    run Dijkstra's algorithm from $s$ in the graph $(V, E)$
foreach $u, v \in V$ do
    if $v \notin ball(u, V, S_k)$ and $u \notin ball(v, V, S_k)$ then
        $d[u, v] \leftarrow \min_{0 \le i \le k}\{d[u, p_i(u)] + d[p_i(u), v],\quad d[v, p_i(v)] + d[p_i(v), u]\}$
```

Lemma 7.5. *The table d computed by Algorithm 9 satisfies: $d[u, v] \le 2\delta(u, v)$, for all $(u, v) \in V \times V$.*

Proof. Let u and v be any two vertices in the graph $G = (V, E)$. The augmented scheme computes $ball(x, V, S_k)$ at each vertex $x \in V$. So if $v \in ball(u, V, S_k)$ or $u \in ball(v, V, S_k)$, the exact u-v distance is stored in $d[u, v]$. Otherwise there are two cases.

Case 1: $\delta(u,p_k(u))+\delta(v,p_k(v))>\delta(u,v)$.

It follows from Theorem 7.3 that

$$\min_{0\le i<k}\{d[u,p_i(u)]+d[p_i(u),v],\ d[v,p_i(v)]+d[p_i(v),u]\}\le 2\delta(u,v).$$

Hence $d[u,v]$ is at most $2\delta(u,v)$.

Case 2 : $\delta(u,p_k(u))+\delta(v,p_k(v))\le\delta(u,v)$.

Let us assume, without loss of generality, that $\delta(u,p_k(u))\le\delta(v,p_k(v))$. Since $d[u,p_k(u)]$ is set to $\delta(u,p_k(u))$ by the scheme $\hat{\mathcal{A}}$, it follows that $2d[u,p_k(u)]\le\delta(u,v)$ holds for Case 2. Note that we perform Dijkstra's algorithm in the original graph from every vertex of set S_k. Hence $d[p_k(u),v]=\delta(p_k(u),v)$, and applying the triangle inequality it follows that $d[p_k(u),v]\le\delta(u,p_k(u))+\delta(u,v)$. This inequality and the inequality $2d[u,p_k(u)]\le\delta(u,v)$ lead to

$$d[u,p_k(u)]+d[p_k(u),v]\le 2d[u,p_k(u)]+\delta(u,v)\le 2\delta(u,v).$$

□

The algorithm employs the augmented scheme $\hat{\mathcal{A}}$ for hierarchy $\hat{\mathcal{R}}(0,\frac{1}{2},k)$ with $k=\log n$. This will take $\tilde{O}(m\sqrt{n}+n^2)$ time using Lemma 7.4. It is easy to observe that all the remaining tasks of the algorithm will take expected $O(m\sqrt{n}+n^2\log n)$ time. Hence the overall time complexity of the algorithm is $\tilde{O}(m\sqrt{n}+n^2)$. We have thus proved the following theorem.

THEOREM 7.6. *A given undirected weighted graph on n vertices can be preprocessed in expected $\tilde{O}(m\sqrt{n}+n^2)$ time to build an $n\times n$ table that stores all-pairs 2-approximate distances.*

7.3. Algorithm for all-pairs 7/3-approximate shortest paths. Algorithm 10 computes all-pairs 7/3-approximate shortest paths for a given weighted undirected graph $G=(V,E)$. Its running time is $\tilde{O}(m^{2/3}n+n^2)$ which improves the $\tilde{O}(n^{7/3})$ time algorithm by Cohen and Zwick [7].

The algorithm can be viewed as an extension of the 2-APASP algorithm of the previous section with the objective of achieving a better running time at the expense of a slightly larger stretch.

Algorithm 10: The algorithm for all-pairs 7/3-stretch distances

```
Let $\beta,\gamma$ be two positive numbers to be fixed later on;
Compute the scheme $\hat{\mathcal{A}}$ for $\hat{\mathcal{R}}(0,\beta,k)$ with $k=\log n$;
Let $S_{k+1}$ be formed by selecting each $v\in S_k$ with prob. $n^{-\gamma}$,   and $S_{k+2}\leftarrow\emptyset$;
foreach $i\in\{k,k+1\}$ and $s\in S_i$ do
    run Dijkstra's algorithm with source $s$ in the subgraph $(V,E_{S_{i+1}}\cup d[s,\_])$
foreach $u\in V$ do
    foreach $s\in S_k$ that is present in ball$(u,V,S_{k+1})$ and $v\in V$ do
        $d[u,v]\leftarrow\min(d[s,v]+d[s,u],\ d[u,v])$
foreach $u,v\in V$ do
    $d[u,v]\leftarrow\min_{0\le i\le k+1}\{d[u,p_i(u)]+d[p_i(u),v],\ d[v,p_i(v)]+d[p_i(v),u],\ d[u,v]\}$
foreach $u,v\in V$ do $d[u,v]\leftarrow\min(d[u,v],d[v,u])$
```

Running Time Analysis. The computation of scheme $\hat{\mathcal{A}}(\hat{\mathcal{R}}(0,\beta,k))$ takes $\tilde{O}(mn^\beta + n^2)$ (from Theorem 7.4). The first **for** loop would have expected $O(mn^{1-\gamma-\beta} + n^{2+\gamma})$ running time. The second **for** loop would have expected $O(n^{2+\gamma})$ running time. The third **for** loop takes expected $O(n^2 \log n)$ time. In order to minimize the sum of all these quantities, we set

$$mn^\beta = mn^{1-\gamma-\beta} = n^{2+\gamma}$$

This gives $n^{3\beta}m = n^3$. Hence the expected time complexity of Algorithm 10 is $\tilde{O}(m^{2/3}n + n^2)$.

7.3.1. Bounding the stretch. Consider any two vertices $u, v \in V$. The reader may easily notice that apart from having a different value of β, Algorithm 10 has essentially all the ingredients of the algorithm for 2-APASP. In particular, the first and the third **for** loops do essentially the same computational tasks except that the hierarchy of vertices is a little different - one more layer of vertices S_{k+1} has been added to it. Due to this similarity, it is easy to observe that Algorithm 10 computes a 2-approximate distance between u and v if either of the following conditions hold :

- $\delta(u, p_k(u)) + \delta(v, p_k(v)) > \delta(u,v)$, that is, if $ball(u, V, S_k)$ and $ball(v, V, S_k)$ *intersect*.
- $\delta(u, p_{k+1}(u)) + \delta(v, p_{k+1}(v)) \le \delta(u,v)$, that is, if $ball(u, V, S_{k+1})$ and $ball(v, V, S_{k+1})$ do not *intersect*.

So the only nontrivial case that needs to be addressed is when $ball(u, V, S_k)$ and $ball(v, V, S_k)$ do not intersect but $ball(u, V, S_{k+1})$ and $ball(v, V, S_{k+1})$ do intersect. This case has been described in Figure 7.1.

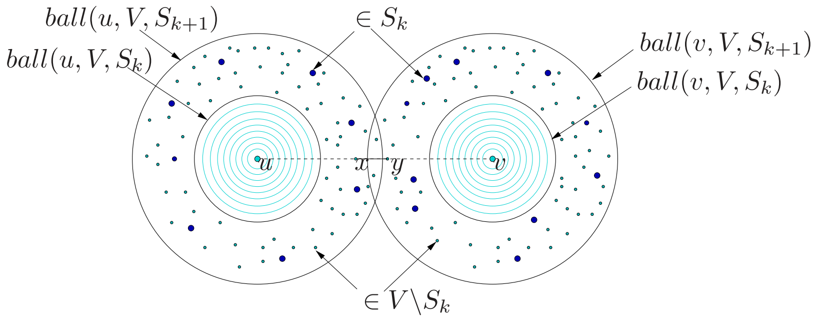

FIG. 7.1. *analyzing the nontrivial case for 7/3-APASP algorithm*

Let y be the first vertex on u-v path from the side of u that belongs to $ball(v, V, S_{k+1})$, and let x be its predecessor. Furthermore assume that neither x belongs to $ball(y, V, S_{k+1})$ nor y belongs to $ball(x, V, S_{k+1})$. Otherwise the entire shortest path between u and v is present in the edge set $E_{S_{k+1}}$ and all these edges are present in the graph on which we run Dijkstra's algorithm from vertices of S_k. This yields: $d[p_k(u), v] \le \delta(u,v) + \delta(u, p_k(u))$ and $d[p_k(v), u] \le \delta(u,v) + \delta(v, p_k(v))$. Hence $d[u,v] \le \delta(u,v) + 2\min\{\delta(u, p_k(u)), \delta(v, p_k(v))\}$, which is bounded by $2\delta(u,v)$ because $ball(u, V, S_k)$ and $ball(v, V, S_k)$ do not intersect.

Let us assume without loss of generality that $\delta(u,x) \le \delta(v,y)$. We now state one simple lemma that will be very useful in our analysis.

LEMMA 7.7. *If $u \notin ball(x, V, S_k)$ but $p_k(x) \in ball(u, V, S_{k+1})$, then $d[u,v] \le 2\delta(u,v)$.*

Proof. It follows from Observation 7.1 that after computing the augmented scheme $\hat{\mathcal{A}}$, $d[p_k(x), y] \leq \delta(p_k(x), x) + \mathbf{w}(x, y)$. Also observe that the entire shortest path between y and v is present in $E_{S_{k+1}}$. Therefore, at the end of the first **for** loop of Algorithm 10, the following bound holds for $d[p_k(x), v]$:

$$d[p_k(x), v] \leq \delta(p_k(x), x) + \delta(x, v) \tag{7.2}$$

It is given that $p_k(x) \in ball(u, V, S_{k+1})$. So it follows from Lemma 2.2 that the entire shortest path from $p_k(x)$ to u is also present in the subgraph $E_{S_{k+1}}$. Therefore, at the end of the first **for** loop, $d[p_k(x), u]$ is equal to $\delta(p_k(x), u)$. Furthermore, using the triangle inequality, it follows that $\delta(p_k(x), u)$ is bounded by $\delta(p_k(x), x) + \delta(u, x)$. This bound and Inequality 7.2 help $p_k(x)$ refine the estimate on $d[u, v]$ quite effectively during the second **for** loop of Algorithm 10 as follows.

$$\begin{align} d[u, v] &\leq d[p_k(x), u] + d[p_k(x), v] \tag{7.3}\\ &\leq 2\delta(p_k(x), x) + \delta(u, x) + \delta(x, v) \tag{7.4}\\ &= 2\delta(p_k(x), x) + \delta(u, v). \tag{7.5}\end{align}$$

Since it is given that $u \notin ball(x, V, S_k)$, so $\delta(x, p_k(x)) \leq \delta(u, x)$. Moreover, $\delta(x, p_k(x)) \leq \delta(x, y)$ since $y \notin ball(x, V, S_{k+1})$. Hence $2\delta(p_k(x), x)$ is bounded by $\delta(u, x) + \delta(x, y)$ which is bounded by $\delta(u, v)$. Hence it follows from Equation (7.5) that $d[u, v]$ is bounded by $2\delta(u, v)$. □

THEOREM 7.8. *The table d returned by Algorithm 10 satisfies :* $d[u, v] \leq 7/3\delta(u, v)$.

Proof. We split the proof into two cases.

Case 1 : $u \notin ball(x, V, S_k)$.

If $p_k(x)$ belongs to $ball(u, V, S_{k+1})$, it follows from Lemma 7.7 that $d[u, v]$ is bounded by $2\delta(u, v)$. So let us consider the case when $p_k(x)$ does not belong to $ball(u, V, S_{k+1})$. Since as a part of the algorithm, we execute Dijkstra's algorithm from $p_{k+1}(u)$ in the entire graph, at the end of the third **for** loop, we have the following bound on $d[u, v]$.

$$d[u, v] \leq \delta(u, v) + 2\delta(p_{k+1}(u), u)$$

We shall now bound this value by $7/3\delta(u, v)$ as follows. Since $p_k(x) \notin ball(u, V, S_{k+1})$, we have $\delta(u, p_{k+1}(u)) \leq \delta(u, x) + \delta(x, p_k(x))$. Thus we have

$$\begin{align} \delta(u, v) + 2\delta(p_{k+1}(u), u) &\leq \delta(u, v) + 2\delta(u, x) + 2\delta(x, p_k(x))\\ &\leq \delta(u, v) + \delta(u, x) + \delta(v, y) + \delta(x, y) + \delta(x, p_k(x))\\ &= 2\delta(u, v) + \delta(x, p_k(x)).\end{align}$$

In the above inequalities we bounded one occurrence of $\delta(u, x)$ by $\delta(v, y)$ which was our assumption and we bounded one occurrence of $\delta(x, p_k(x))$ by $\delta(x, y)$ which holds because $y \notin ball(x, V, S_{k+1})$. Now note that $\delta(x, p_k(x))$ is bounded by $\delta(u, x)$ (and hence by $\delta(v, y)$ also) since $u \notin ball(x, V, S_k)$. Combining these inequalities together, it follows that $\delta(x, p_k(x)) \leq \delta(u, v)/3$. Thus $d[u, v]$ is at most $7/3\delta(u, v)$.

Case 2 : $u \in ball(x, V, S_k)$.

Applying Observation 7.2 and using the fact that $u \rightsquigarrow x \rightarrow y$ is a shortest path, we can note that $d[p_k(u), y] \leq \delta(p_k(u), u) + \delta(u, y)$. Moreover all the edges of the shortest path between y and v are present in $E_{S_{k+1}}$ since $y \in ball(v, V, S_{k+1})$. Therefore, in the first **for** loop of Algorithm 10, when Dijkstra's algorithm is performed from $p_k(u)$ on the graph with $E_{S_{k+1}}$ edges, the approximate distance $d[p_k(u), v]$ computed is at most

$$d[p_k(u), v] \leq \delta(p_k(u), u)) + \delta(u, v).$$

This leads to the following upper bound on $d[u, v]$ at the end of the third **for** loop.

$$d[u, v] \leq \delta(u, v) + 2\delta(u, p_k(u)). \tag{7.6}$$

Analogously if v also belongs to $ball(y, V, S_k)$, then $d[v, u]$ is bounded by $\delta(u, v) + 2\delta(v, p_k(v))$ as well. Since $ball(u, V, S_k)$ and $ball(v, V, S_k)$ do not intersect, the minimum of $2\delta(v, p_k(v))$ and $2\delta(u, p_k(u))$ will be $\delta(u, v)$. Hence $\min(d[u, v], d[v, u])$ at the end of the algorithm will be bounded by $2\delta(u, v)$ in this case.

So let us consider the case when v does not belong to $ball(y, V, S_k)$. In this case if $p_k(y)$ belongs to $ball(v, V, S_{k+1})$, we again get $d[v, u] \leq 2\delta(u, v)$ because of Lemma 7.7. So the only situation left is when $p_k(y)$ does not belong to $ball(v, V, S_{k+1})$. Now observe that the inequality $d[u, v] \leq \delta(u, v) + 2\delta(v, p_{k+1}(v))$ holds due to the execution of Dijkstra's algorithm from $p_{k+1}(v)$ in the given graph during the first **for** loop of the algorithm. This inequality and the fact that $p_k(y) \notin ball(v, V, S_{k+1})$ leads to the following Inequality.

$$d[v, u] \leq \delta(u, v) + 2\delta(y, p_k(y)) + 2\delta(v, y) \tag{7.7}$$

The right hand side of Inequality (7.7) is further bounded by $\delta(u, v) + 4\delta(v, y)$ since $v \notin ball(y, V, S_k)$. Hence

$$d[v, u] \leq \delta(u, v) + 4\delta(v, y). \tag{7.8}$$

Combining Inequalities (7.6) and (7.8), it follows that just before the fourth loop of the algorithm,

$$\begin{aligned}
2d[u, v] + d[v, u] &\leq 2(\delta(u, v) + 2\delta(u, p_k(u))) + (\delta(u, v) + 4\delta(v, y)) \\
&= 3\delta(u, v) + 4(\delta(u, p_k(u)) + \delta(v, y)) \\
&\leq 3\delta(u, v) + 4(\delta(u, y) + \delta(v, y)) \quad \{\text{since } y \notin ball(u, V, S_{k+1})\} \\
&= 3\delta(u, v) + 4\delta(u, v) = 7\delta(u, v)
\end{aligned}$$

Hence $\min(d[u, v], d[v, u])$ is bounded by $7/3\delta(u, v)$. Since the fourth loop of the algorithm sets $d[u, v]$ to $\min(d[u, v], d[v, u])$, it follows that $d[u, v] \leq 7/3\delta(u, v)$ holds at the end of the algorithm. ☐

8. Conclusion.

In this paper, we presented faster algorithms for all-pairs approximate shortest paths in weighted undirected graphs. One of the main result presented in this paper is the generic scheme $\mathcal{A}$ for approximate shortest paths. In addition to achieve faster algorithms for all-pairs approximate shortest paths with stretch $2, (2, \mathbf{w}), \frac{7}{3}$. It also plays crucial role in designing a new algorithm for stretch 3 which has the best features of the previous two algorithms [18] and [7] simultaneously. These two algorithms happened to be two degenerate cases of the new algorithm. Another important result is an $O(n^2)$ time algorithm to construct $(2k-1)$-approximate distance oracles, thus answering the open question of Thorup and Zwick [18] for all $k > 1$. Achieving the upper bound of $O(n^2)$ on the preprocessing time for approximate distance oracles is a significant result since it matches the worst case input size

- $m = \Theta(n^2)$. However, it would be interesting to break this quadratic barrier for all graphs with $m = o(n^2)$. Recently some progress has been made in this direction for unweighted graphs at the expense of small additive error [2]. It would be quite interesting to know if similar results can be achieved for weighted graphs as well.

Acknowledgments. The authors are grateful to Sandeep Sen and anonymous referee for their valuable suggestions and comments. The first author is very thankful to Max-Planck Institute for computer science, Saarbruecken, Germany for providing an excellent and inspirational research atmosphere. It was during his stay there that the major part of this research work was done.

REFERENCES

[1] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths(without matrix multiplication). *SIAM Journal on Computing*, 28:1167–1181, 1999.

[2] Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In *Proceedings of 35th International Colloquium on Automata, Languages, and Programming*, volume 5125 of *LNCS*, pages 609–621. Springer, 2008.

[3] Surender Baswana, Vishrut Goyal, and Sandeep Sen. All-pairs nearly 2-approximate shortest paths in $O(n^2 \text{polylog} n)$ time. In *Proceedings of 22nd Annual Symposium on Theoretical Aspect of Computer Science*, volume 3404 of *LNCS*, pages 666–679. Springer, 2005.

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

[5] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures and Algorithms*, 30:532–563, 2007.

[6] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *Proceedings of 39th Annual ACM Symposium on Theory of Computing*, pages 590–598, 2007.

[7] Edith Cohen and Uri Zwick. All-pairs small stretch paths. *Journal of Algorithms*, 38:335–353, 2001.

[8] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[9] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. In *Introduction to Algorithms*. The MIT Press, 1990.

[10] Dorit Dor, Shay Halperin, and Uri Zwick. All pairs almost shortest paths. *Siam Journal on Computing*, 29:1740–1759, 2000.

[11] Michael Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1:282–323, 2005.

[12] Paul Erdős. Extremal problems in graph theory. In *Theory of Graphs and its Applications (Proc. Sympos. Smolenice,1963)*, pages 29–36, Publ. House Czechoslovak Acad. Sci., Prague, 1964.

[13] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312:47–74, 2004.

[14] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic construction of approximate distance oracles and spanners. In *Proceedings of 32nd International Colloquium on Automata, Languagaes and Programming*, volume 3580 of *LNCS*, pages 261–272. Springer, 2005.

[15] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51:400–403, 1995.

[16] A. Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 605–615, 1999.

[17] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of 13th ACM Symposium on Parallel Algorithms and Architecture*, pages 1–10, 2001.

[18] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of Association of Computing Machinery*, 52:1–24, 2005.

[19] Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted apsp. In *Proceedings of 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 950–957, 2009.

[20] Uri Zwick. Exact and approximate distances in graphs - a survey. In *Proceedings of the 9th Annual European Symposium on Algorithms*, volume 2161 of *LNCS*, pages 33–48. Springer, 2001.

[21] Uri Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of Association of Computing Machinery*, 49:289–317, 2002.