All-Pairs Nearly 2-Approximate Shortest Paths in $O(n^2 \operatorname{polylog} n)$ Time[*]

Surender Baswana [†]
Department of Computer Science & Engineering
I.I.T. Kanpur, India.
Email : sbaswana@cse.iitk.ac.in

Vishrut Goyal [‡]
Persistent Systems Private Limited,
Pune, India.
Email : vishrut_goyal@persistent.co.in

Sandeep Sen
Department of Computer Science & Engineering,
I.I.T. Delhi, India.
Email : ssen@cse.iitd.ernet.in

February 28, 2008

Abstract

Let $G = (V, E)$ be an unweighted undirected graph on $|V| = n$ vertices and $|E| = m$ edges. Let $\delta(u, v)$ denote the distance between vertices $u, v \in V$. An algorithm is said to compute all-pairs t-approximate shortest-paths/distances, for some $t \geq 1$, if for each pair of vertices $u, v \in V$, the path/distance reported by the algorithm is not longer/greater than $t \cdot \delta(u, v)$.

This paper presents two extremely simple randomized algorithms for computing all-pairs *nearly* 2-approximate distances. The first algorithm requires expected $O(m^{2/3} n \log n + n^2)$ time, and for any $u, v \in V$ reports distance no greater than $2\delta(u, v) + 1$. Our second algorithm requires expected $O(n^2 \log^{3/2} n)$ time, and for any $u, v \in V$ reports distance bounded by $2\delta(u, v) + 3$.

1 Introduction

The all-pairs shortest paths (APSP) problem is undoubtedly one of the most fundamental algorithmic graph problems. Given a graph $G = (V, E)$ on $n(= |V|)$ vertices and $m(= |E|)$ edges, the problem requires computation of shortest-paths/distances between each pair of vertices. There are various versions of this problem depending on whether the graph is directed or undirected, edges are weighted or unweighted, weights are positive or negative. In its most generic version, that is, for directed graph with

[*]Preliminary version of this result appeared in 22nd Symposium on Theoretical Aspects of Computer Science (STACS), 2005

[†]the work was done while the author was a postdoctoral researcher at Max-Planck Institute for computer science, 66123 Saarbrücken, Germany.

[‡]the work was done while the author was a masters student in the department of computer science at I.I.T. Delhi, India.

real edge-weights, the best known algorithm [14] for this problem requires $O(mn + n^2 \log\log n)$ time. However, 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. The best known upper bound on the time complexity of this problem is $O(n^3/\log^2 n)$ due to Chan [7], which is marginally subcubic. The existing lower bound on the time complexity of APSP is the trivial $\Omega(n^2)$ lower bound.

There exist subcubic algorithms for APSP problem if the edges weights are integers in a finite range. All these algorithms employ fast (subcubic) 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. For the usual algebraic, i.e., $(+, \times)$ product of two matrices, Strassen [17] gave the first subcubic algorithm, and many faster algorithms followed this algorithm. 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 [11] implies $\omega < 2.376$. For undirected unweighted graphs, Seidel gave a very simple and elegant algorithm to solve APSP in $\tilde{O}(n^\omega)$ time. In fact he showed that APASP in undirected unweighted graphs is harder than Boolean matrix multiplication by at most a polylogarithmic factor. For APSP in undirected graphs with edges weights from $\{0, 1, ..., M\}$, Shoshan and Zwick [16] desiged an $\tilde{O}(Mn^\omega)$ algorithm. For unweighted directed graphs, the first truly sub-cubic algorithm was designed by Alon et al. [2], which was improved subsequently by Takaoka [18], and most recently by Zwick [22]. For directed graphs with weights from the range $\{-1, 0, 1\}$, the algorithm by Zwick achieves $O(n^{2.575})$ running time. This algorithm works for integer edge weights in the range $\{-M, \ldots, M\}$ as well, and achieves subcubic running time provided $M < n^{3-\omega}$.

An algorithm for APSP problem which is based on matrix multiplication is undoubtedly very important because it breaks the cubic barrier in the time complexity of the fundamental problem of APSP. However, there is a natural question as to whether it is possible to design a sub-cubic algorithm for APSP problem that does not resort to any fast matrix multiplication subroutines. This question becomes even more significant given the wide practicality of APSP problem and the fact that all the existing algorithms for fast matrix multiplication are notoriously impractical. Motivated by this question, a number of sub-cubic algorithms have been designed in the last ten years that are based on very simple and novel combinatorial ideas, but compute $approximate$, instead of exact, shortest paths.

Algorithms for all-pairs approximate shortest paths : As the name suggests, an algorithm for approximate shortest-paths (or distances) would report paths (distances) which are longer than the actual shortest-paths (distances). The error associated with the distance could be either additive (*surplus*) or multiplicative (*stretch*). An algorithm is said to compute all-pairs *surplus-k* distances for some $k \geq 0$, if for any pair of vertices $u, v \in V$, the distance reported is at least $\delta(u, v)$ and at most $\delta(u, v) + k$. Likewise an algorithm is said to compute all-pairs *stretch-t* distances for a given $t \geq 1$, if for any pair of vertices $u, v \in V$ the distance reported is at least $\delta(u, v)$ and at most $t\delta(u, v)$. An interesting theoretical question is to find the possible trade-offs between the time complexity of such an algorithm and stretch/surplus of the distance it guarantee.

Consider the naive approach of computing APSP in an undirected unweighted graph where we build shortest path trees on all the vertices. The total running time of this algorithm will be $O(mn)$. The subcubic algorithms for approximate shortest-paths are based on the simple observation that minimizing the first term m (the number of edges) and/or the second term n (the number of vertices) in the expression $O(mn)$ would lead to subcubic running time. This objective may sound very simple; However, achieving this objective with a small bound on approximation factors (surplus or stretch) is quite nontrivial, and is achieved using simple but ingenious techniques by the existing algorithms.

Minimizing the term m in the running time $O(mn)$ could be achieved if we perform shortest path computation on a sparse subgraph. Since we require a bound on the approximation factor, it would be necessary that the subgraph preserves all-pairs distances approximately. Such a graph is called *spanner*. The challenge here is to compute the spanners with small stretch (or surplus) and still having a small size. There are a number of algorithms [3, 8, 9] that take this *spanner based* approach for computing approximate shortest paths with stretch 3 or more. Recently Elkin [13] also presented an algorithm based on this approach that achieves a tradeoff between the running time, stretch, and the surplus. In precise words, he gave the following result. Given an undirected unweighted graph and arbitrarily small constants $\zeta, \epsilon, \rho > 0$, there exists an algorithm that requires $O(mn^{\rho} + n^{2+\zeta})$ time, and for any pair of vertices $u, v \in V$, reports distance $\delta^*(u, v)$ satisfying the inequality :

$$\delta(u, v) \le \delta^*(u, v) \le (1 + \epsilon)\delta(u, v) + \beta$$

where β is a function of ζ, ϵ, ρ. If the two vertices $u, v \in V$ are separated by sufficiently long distances in the graph, the ratio $\frac{\delta^*(u,v)}{\delta(u,v)}$ 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.

Minimizing the term n in the running time $O(mn)$ could be achieved if we perform shortest path computation on a few *special* vertices only. This approach alone is not able to compute approximate shortest paths, so it is combined with computation of shortest path trees on the remaining vertices in a sparse subgraph defined in some *suitable* way by the special vertices. Quite often, this 2-level approach can be easily generalized to k-levels, for any integer $k > 2$. There is a family of algorithms based on this *hierarchical* approach. First such algorithm was designed by Aingworth et al. [1]. Their algorithm computes all-pairs surplus-2 distances in $O(n^{2.5} \operatorname{polylog} n)$. Dor et al. [12], improved this algorithm to achieve a running time of $O(\min(n^{3/2}\sqrt{m}, n^{7/3}) \operatorname{polylog} n)$. Note that for any two vertices, surplus-2 distance is also stretch-2 distance (but not vice versa) unless they are neighbors, which can be verified in constant time. Hence the algorithm by Dor et al. [12] that computes all-pairs surplus-2 distances can also report stretch-2 distances for any pair of vertices. They also generalized the algorithm to compute all-pairs surplus-$2k$ distances for arbitrary integer $k \ge 1$, and for $k = \log n$, they show that the resulting algorithm also computes all-pairs stretch 3 distances in $O(n^2 \operatorname{polylog} n)$ time.

In addition to the above mentioned work on undirected unweighted graphs, a lot of work has also been done on undirected weighted graphs including the milestone of *approximate distance oracles* by Thorup and Zwick [20]. This result achieves subcubic running time as well as optimal sub-quadratic space requirement for any stretch ≥ 3. For these and related algorithms for weighted graphs [10, 15], we refer the reader to an excellent and comprehensive survey by Zwick [21].

1.1 Our contribution and organization of the paper

We address the problem of computing all-pairs 2-approximate distances. The stretch 2 is unique in that it is the smallest stretch one can aim to compute for approximate distances in time which is less than Boolean matrix multiplication. This is because any algorithm that computes all-pairs approximate distances with stretch strictly less than 2 can be adapted to compute multiplication of any two Boolean matrices in the same time [12]. As mentioned above, the best known algorithm for computing all-pairs stretch-2 distances in unweighted graphs [12] has a running time of $O(\min(n^{3/2}m^{1/2}, n^{7/3}) \operatorname{polylog} n)$. This bound indeed beats the existing best known bound of $O(n^{2.376})$ for Boolean matrix multiplication. However, the question is whether this time complexity of all-pairs 2-approximate distances can be further improved. Note that there exists $\Omega(n^2)$ lower bound on space as well as time complexity of the problem of all-pairs 2-approximate distances. The contribution of this paper is to show that we can in fact compute all-pairs *nearly* 2-approximate distances for unweighted undirected graphs in time $O(n^2 \operatorname{polylog} n)$. We would also like to remark that previously it was possible to compute only all-pairs stretch-3 distances or (surplus $O(\log n)$ distances) in this time [12].

1. We first design a data structure that, given any two vertices $u, v \in V$, requires constant time to report distance bounded by $2\delta(u,v)+1$, that is, an additive error of one unit over the 2-approximate distance. Employing efficient randomized subroutines, we show that the expected preprocessing time required to build this data structure is $O(m^{2/3} n \log n + n^2)$. At the cost of introducing a surplus of one unit, this new algorithm (Algorithm I) is strictly faster than the previous best algorithm whenever $m = o(n^2)$ [12]. Moreover, for the range $m < n^{3/2}$, the new algorithm requires expected $O(n^2)$ time.

   The algorithm falls under the category of algorithms for approximate shortest paths that employ *hierarchical* approach, and is based on a new scheme for all-pairs stretch-2 shortest paths. This scheme, using a new idea, builds upon the earlier work of Thorup and Zwick [19, 20] which deals with the computation of all pairs approximate distances with stretch ≥ 3.

2. We further reduce the expected preprocessing time to $O(n^2 \log^{3/2} n)$ at the expense of increasing the surplus to 3, that is, given any pair of vertices $u, v \in V$, the distance $\delta^*(u,v)$ reported by our second algorithm (Algorithm II) satisfies

$$\delta(u,v) \leq \delta^*(u,v) \leq 2\delta(u,v) + 3$$

   The algorithm employs the new scheme on top of a special kind of spanner in order to achieve faster time. In this way, this algorithm combines both the hierarchical approach and the spanner based approach. The main obstacle in the design of this algorithm is to employ a spanner that should be constructible in quadratic time and should not increase the stretch beyond 2. We can't employ any additive spanner since none of them is constructible in quadratic time. On the other hand, employing a (multiplicative) spanner with stretch t would increase the stretch of the approximate distance reported to $2t$, which is undesirable. We get around this problem in a way similar to [5].

   As would become clear subsequently from the paper, the additive error shows up only in some restricted worst case only. In general, the algorithms will behave very much like a 2-approximate shortest path algorithm.

Without any modifications, all our data structures for reporting approximate distance can also be used to report approximate shortest path in optimal time. After discussing the preliminary notations and lemmas in the following section, we present and analyze the new scheme to design efficient algorithms for stretch *nearly* 2 in Section 3. In order to realize this scheme, we present and analyze efficient randomized subroutines in section 4. Finally, we present our two algorithms in section 5.

2 Preliminaries

In an unweighted graph, shortest path tree at a vertex is the same as a BFS (breadth first search) tree rooted at that vertex, and such a tree can be constructed in $O(m)$ time. Now we present the notations, definitions and important Lemmas (most of them from [20]) to be used in the rest of the paper.

For a given undirected unweighted graph $G = (V, E)$, and a set $Y \subseteq V$, first we define the following notations.

- $\delta(u,v)$: the distance between vertex u and vertex v in the graph.
- $\delta(v,Y)$: $\min_{y \in Y} \delta(v,y)$

- $n_v(Y)$: the vertex from the set Y which is nearest to v, that is, at distance $\delta(v, Y)$ from v. If there are multiple vertices in set Y at distance $\delta(v, Y)$ from v, we break the tie arbitrarily to ensure a unique $n_v(Y)$. Moreover, if $v \in Y$, we define $n_v(Y) = v$. For conciseness, we shall use n_v to denote $n_v(Y)$ when the set Y is clear from the context.
- R_p : A subset formed by selecting each vertex from V independently with probability p.

Given a set $Y \subset V$, it is quite easy to compute $n_v(Y)$ and $\delta(v, Y)$ for all $v \in V$ in $O(m)$ time as follows. Connect a dummy vertex ω to all the vertices of set Y, and perform a BFS traversal on the graph starting from ω. We can summarize this observation as the following Lemma.

Lemma 2.1 *Given a set $Y \subset V$, it requires $O(m)$ time to compute $n_v(Y)$ and $\delta(v, Y)$ for all vertices $v \in V$.*

An important construct from [20] which we shall use is a *Ball* which is defined as follows.

Definition 2.1 *Given a graph $G = (V, E)$, a vertex $v \in V$, and two subsets of vertices X and Y, the set $Ball(v, X, Y)$ is defined in the following way*

$$Ball(v, X, Y) = \{x \in X | \delta(v, x) < \delta(v, Y)\}$$

In other words, $Ball(v, X, Y)$ consists of all those vertices of the set X whose distance from v is less than the distance between $n_v(Y)$ and v. The *Radius* $r(v)$ of $Ball(v, X, Y)$ is defined as $\delta(v, Y) - 1$. It may be noted that $Ball(v, X, \emptyset)$ is the set X itself, whereas $Ball(v, X, X) = \emptyset$. The following lemma from [20] suggests that if the set Y is a suitable random sample of the set X, the expected size of $Ball(u, X, Y)$ will be sublinear in $|X|$.

Lemma 2.2 *[20] Given a graph $G = (V, E)$, the expected size of $Ball(v, X, Y)$ is at most $1/p$ if the set Y is constructed by either of the following two sampling methods :*
(i) Y contains each vertex from set X independently with probability p.
(ii) Y is a uniformly random sample of size $p|X|$ from set X (where all subsets of size $p|X|$ are equally likely).

Proof: Consider the sequence $\langle x_1, x_2, \cdots \rangle$ of vertices of set X arranged in non-decreasing order of their distance from v.

It follows from Definition 2.1 that the vertex x_i will belong to $Ball(v, X, Y)$ only if none of $x_1, \cdots, x_i$ are selected for the set Y. So if the set Y is formed by selecting each vertex of set X independently with probability p, then $x_i \in Ball(v, X, Y)$ with probability at most $(1 - p)^i$. Hence using linearity of expectation, the expected number of vertices in $Ball(v, X, Y)$ is at most $\sum_i (1 - p)^i < 1/p$.

It also follows from Definition 2.1 that the size of $Ball(v, X, Y)$ is bounded by the index of the first vertex of set Y in the sequence $\langle x_1, x_2, \cdots \rangle$. It follows from elementary probability that if we select np numbers uniformly from $[1, n]$, then the expected value of the smallest number selected is $(n + 1)/(np + 1)$. So if we select a sample of size np uniformly from set X, then the expected size of $Ball(v, X, Y)$ is bounded by $(n + 1)/(np + 1) - 1 < 1/p$. □

Definition 2.2 *Given a graph $G = (V, E)$, and two subsets X, Y of vertices, we define* cluster $C(x, Y)$, *for any $x \in X$ as the set $\{v \in V | x \in Ball(v, X, Y)\}$.*

Clusters can be viewed as inverses of Balls. Given any two subsets X, Y of vertices, the following important equality between the size of Balls and clusters can be easily verified using Definition 2.2.

$$\sum_{x \in X} |C(x, Y)| = \sum_{v \in V} |Ball(v, X, Y)| \tag{1}$$

Let $R \subset V$.

1. **Global distance information**

   For each vertex $s \in R$, build a BFS tree to compute distance from s to every vertex in the graph.

2. **Local distance information**

   For each vertex $u \in V \backslash R$, compute distance to all the vertices of $Ball(u, V, R)$ and its nearest vertex n_u.

3. Keep a data structure to determine, in constant time, whether any two Balls overlap (share a common vertex).

Figure 1: New scheme for nearly 2-approximate distances

3 A New Scheme for nearly 2-Approximate distances

The new scheme is described in Figure 1. The scheme may appear similar to the 3-approximate distance oracle of Thorup and Zwick [20] except the third step. However, it is this step that proves to be crucial in achieving nearly 2 stretch. The query procedure $\mathcal{Q}(u, v)$ for reporting approximate distance between any $u, v \in V$ using this scheme is described in Figure 2. The query Procedure $\mathcal{Q}(u, v)$ explores the following three possible cases in the above fixed order. First, if any of the two vertices lie in the Ball rooted at the other, we report the exact distance between u and v. Otherwise, if the Balls rooted at the two vertices overlap, and let w be a vertex common to both the Balls, then we report the sum of the distance from u to w and distance from v to w. In both the cases, we manage to report distance using only local distance information stored at vertices u and v. The only case that is left is, when the two Balls are non-overlapping. In this case, we use the global distance information stored at n_u and n_v. Note that our scheme keeps track of the exact distance from n_u to v as well as from n_v to u. We compute the sum of distance from u to n_u and distance from n_u to v; likewise, we compute sum of distance from v to n_v and distance from n_v to u, and report the minimum of the two sums as an approximate distance between u and v.

Theorem 3.1 *Given a graph $G = (V, E)$ and any two vertices $u, v \in V$, the approximate distance between u and v as reported by the query procedure $\mathcal{Q}(u, v)$ is bounded by $2\delta(u, v) + 1$.*

Proof: Let a and b be the radii of $Ball(u, V, R)$ and $Ball(v, V, R)$ respectively. The approximation factor associated with the distance depends on the step of procedure $\mathcal{Q}(u, v)$ in which it (the approximate distance) is reported. We analyze the three cases as follows:

Case 1 : The distance is reported in the first step of $\mathcal{Q}(u, v)$.
Without loss of generality, let us assume that v lies in $Ball(u, V, R)$ (see Figure 3, Case-1). Here we report the *exact* distance between u and v.

Case 2 : The distance is reported in the second step of $\mathcal{Q}(u, v)$.
Since the query procedure failed to report the distance in the first step, therefore, the distance between u and v is more than the radius of $Ball(u, V, R)$ and $Ball(v, V, R)$. In other words, $\delta(u, v)$ is greater than a as well as b. (see Figure 3, Case-2). Let w be a vertex lying in both $Ball(u, V, R)$ and $Ball(v, V, R)$. Clearly, $\delta(u, w) \leq a$ and $\delta(v, w) \leq b$. Therefore, the distance reported in this step is bounded by $a + b$,

```
$\mathcal{Q}(u,v)$ : We answer a distance query between $u$ and $v$ in the following order.

  • If $v \in Ball(u,V,R)$ or $u \in Ball(v,V,R)$ :

      report $\delta(u,v)$

  • Else if $Ball(u,V,R)$ and $Ball(v,V,R)$ overlap :

      report $\delta(u,w) + \delta(v,w)$ for some $w \in Ball(u,V,R) \cap Ball(v,V,R)$

  • Else :

      report minimum of $(\delta(u,n_u) + \delta(n_u,v))$ and $(\delta(v,n_v) + \delta(n_v,u))$
```

Figure 2: Answering distance query using new scheme

which is at most $2\delta(u,v)$ as explained above.
Case 3 : The distance is reported in the third step of $\mathcal{Q}(u,v)$.
Since the query procedure failed to report the distance in the second step, $Ball(u,V,R)$ and $Ball(v,V,R)$ are separated by distance $x \geq 1$ (see Figure 3, Case-3). So the shortest path between u and v can be viewed as consisting of three sub-paths : the first subpath is the portion of the path lying inside $Ball(u,V,R)$ and has length a, the second sub-path is the portion of the path lying outside the two Balls and has length x, and the third sub-path is the portion of the path lying inside $Ball(v,V,R)$ and has length b. Hence, the distance between u and v is $a + x + b$ for some $x \geq 1$.

In the third step, we report minimum of $(\delta(u,n_u) + \delta(n_u,v))$ and $(\delta(v,n_v) + \delta(n_v,u))$. It can be noted that $\delta(u,n_u) = a + 1$ and $\delta(v,n_v) = b + 1$. Now considering the path from n_u to v passing through u, we can observe that $\delta(n_u,v)$ is at most $2a + x + b + 1$. Similarly, analyzing the path from n_v to u passing through v, we can observe that $\delta(n_v,u)$ is at most $2b + x + a + 1$. Therefore, the distance reported by $\mathcal{Q}(u,v)$ can be bounded from above as follows.

$$\begin{aligned}
\min((\delta(u,n_u) + \delta(n_u,v)) \quad &, \quad (\delta(v,n_v) + \delta(n_v,u))) \\
&\leq \min(3a + x + b + 2, 3b + x + a + 2) \\
&= \min(3a + b, 3b + a) + x + 2 \\
&= 3b + a + x + 2 \quad \{\text{without loss of generality assume } a \geq b\} \\
&\leq 2a + 2b + x + 2 \quad \{\text{since } a \geq b\} \\
&\leq 2(a + x + b) + 1 \quad \{\text{since } x \geq 1\} \\
&= 2\delta(u,v) + 1
\end{aligned}$$

Hence, the distance between u, v as reported by $\mathcal{Q}(u,v)$ is bounded by $2\delta(u,v) + 1$. □
Remark: It is worth noting that the distance between any two vertices $u, v \in V$, as reported by $\mathcal{Q}(u,v)$, is bounded by $2\delta(u,v)$ even in the Case 3, if at least one of the following conditions hold:
(i) $x > 1$, that is, the two Balls are separated by a path longer than one edge.
(ii) $a \neq b$, that is, the radii of the two Balls differs.

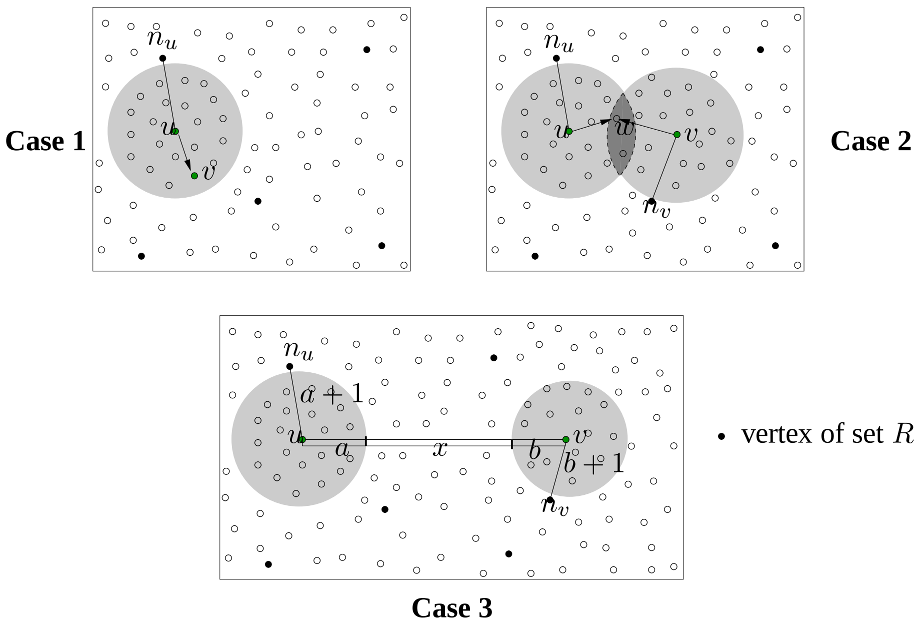

Figure 3: Three cases in reporting distance between u and v

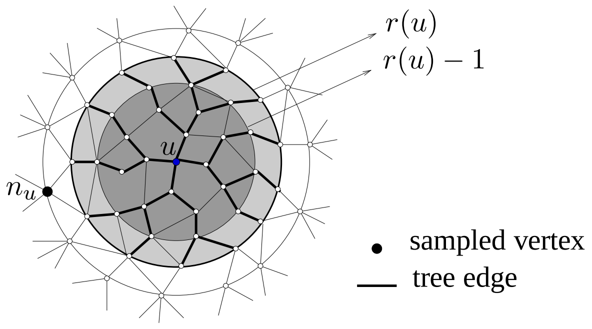

Figure 4: To compute $Ball(u, V, R_p)$, it suffices to explore adjacency list of vertices lying up to distance $r(u) - 1$ only.

4 Efficient Sub-Routines for Realization of the New Scheme

In order to design efficient algorithm based on our new scheme for all-pairs nearly 2-approximate distances, we shall now present sub-routines for efficient computation of Balls as well as efficient detection of overlap of any two Balls.

4.1 An Efficient Algorithm for Computing Balls

We shall now present an algorithm for computing $Ball(u, V, R_p)$, for all $u \in V \backslash R_p$. Recall that R_p is the set formed by selecting each vertex independently with probability $p < 1$.

It follows from Definition 2.1 that the vertices of $Ball(u, V, R_p)$ and their distances from u can be computed by building a BFS tree at u up to level (distance) equal to the radius $r(u)$ of the Ball. Therefore, prior to the computation of $Ball(u, V, R_p)$, we compute $r(u)$. It follows from Lemma 2.1 that we can compute radii of all Balls in $O(m)$ time by a single BFS traversal.

If $r(u) = 0$, $Ball(u, V, R_p)$ consists of vertex u only and we are done. For the case when $r(u) \geq 1$,

we build a BFS tree up to level $r(u)$ to compute $Ball(u, V, R_p)$, and the computation time required in doing so is of the order of the number of edges explored. Since the graph is undirected, an edge will be explored at most twice (once by each of its end-point), and we would charge the cost of exploring an edge to that end-point which explores it first. Let $\langle v_1(=u), v_2, \cdots, v_n\rangle$ be a sequence of vertices of the given graph arranged in non-decreasing order of their distance from u. Note that computing BFS tree up to level $r(u)$ requires exploring the adjacency list of vertices up to level $r(u) - 1$ only (see Figure 4). Therefore, in the computation of $Ball(u, V, R_p)$, we shall explore adjacency list of v_i only if the following two events happen:
$\mathcal{E}_1^i$: There is no vertex in the set $\{v_j | j < i\}$ which is selected in the sample R_p.
$\mathcal{E}_2^i$: There is no vertex from $\{v_j | j > i\}$ that is adjacent to v_i and also a sampled vertex.

The events $\mathcal{E}_1^i$ and $\mathcal{E}_2^i$ are independent since the vertices are selected independently to form R_p. Following our charging scheme mentioned above, exploring adjacency list of vertex v_i would contribute $O(d'(v_i))$ to the computation time of $Ball(u, V, R_p)$, where $d'(v_i)$ is the number of edges incident on v_i from vertices $\{v_j | j > i\}$. So the expected time for computing $Ball(u, V, R_p)$ is

$$\begin{aligned}\sum_{i=1}^{n}\left(\Pr(\mathcal{E}_1^i)\ \cdot\ \Pr(\mathcal{E}_2^i)\ \cdot\ d'(v_i)\right) &= \sum_{i=1}^{n}\left((1-p)^{i-1}(1-p)^{d'(v_i)}d'(v_i)\right)\\ &\le \sum_{i=1}^{n}\left((1-p)^{i-1}\sum_{j=1}^{d'(v_i)}(1-p)^{j-1}\right)\\ &\le \sum_{i=1}^{n}\left((1-p)^{i-1}\frac{1}{p}\right) \le \frac{1}{p}\sum_{i=1}^{n}(1-p)^{i-1} \le \frac{1}{p^2}.\end{aligned}$$

Combining this discussion with Lemma 2.1, we can state the following theorem.

Theorem 4.1 *Given an undirected unweighted graph $G = (V, E)$ and $p < 1$, let R_p be a set formed by selecting each vertex independently with probability p. There exists an algorithm for computing $Ball(u, V, R_p)$ for all $u \in V \backslash R_p$ in expected $O(m + \frac{n}{p^2})$ time.*

It follows from the proof of Theorem 4.1 that if we have a set $R \supseteq R_p$, the time required to compute $Ball(u, V, R)$ is not greater than the time required to compute $Ball(u, V, R_p)$ for any $u \in V$. So we can state the following corollary.

Corollary 4.1 *Given an undirected unweighted graph $G = (V, E)$ and $p < 1$, let R_p be a set formed by selecting each vertex independently with probability $p < 1$. For any set $R \supseteq R_p$, it takes expected $O(m + \frac{n}{p^2})$ time to compute $Ball(u, V, R)$, for all $u \in V \backslash R$.*

4.2 Computing Overlap Matrix $\mathcal{O}$

To determine for a pair of vertices $u, v \in V$, whether there exists a vertex common to both $Ball(u, V, R)$ and $Ball(v, V, R)$, we compute a matrix $\mathcal{O}$ such that $\mathcal{O}[u, v]$ is null if $Ball(u, V, R) \cap Ball(v, V, R) = \emptyset$, otherwise $\mathcal{O}[u, v]$ stores a vertex that belongs to both the Balls. To build the matrix $\mathcal{O}$ efficiently, we form the *clusters* $C(v, R)$ (see Definition 2.2). It is easy to observe that we can form sets $C(v, R), \forall v \in V$ by a single scan of sets $Ball(u, V, R), \forall u \in V$. Now once we have $C(v, R), \forall v \in V$, we can compute the matrix $\mathcal{O}$ as follows.

Algorithm for computing overlap matrix $\mathcal{O}$

```
For each $v \in V \backslash R$ do
    For each $u \in C(v, R)$ do
        For each $w \in C(v, R)$ do
            $\mathcal{O}[u, w] \leftarrow v$
```

```
Procedure **Augment**($V', R_p$) {
    $R \leftarrow R_p$;
    While ($V' \neq \emptyset$)
        Let $A$ be a uniform sample of size $np$ from $V'$
        $R \leftarrow A \cup R$
        For every $u \in V \backslash R$, compute $Ball(u, V, R)$;
        For every $v \in V \backslash R$ do
            $C(v, R) \leftarrow \{u \in V \mid v \in Ball(u, V, R)\}$;
        $V' \leftarrow \{v \in V' \,\Big|\, |C(v, R)| > 4/p\}$;
        End While ;
    Return $R$;
}
```

Figure 5: Augmenting the set R_p to ensure $|C(v, R)| = O(1/p),\ \forall v \in V$

The running time of the above algorithm for computing the overlap matrix $\mathcal{O}$ is of the order of $\sum_{v \in V} |C(v, R)|^2 + n^2$. In order to bound this running time by $O(n/p^2 + n^2)$ (which also matches the time required to compute Balls), we need to find a set $R \subset V$ which would ensure that $|C(v, R)| = O(1/p)$, for each $v \in V$. It should be noted that the equality 1 and Lemma 2.2 can't ensure $O(1/p)$ bound on the expected size of $C(v, R)$ if R is chosen to be R_p. Moreover, due to quadratic dependence of the running time (of the above algorithm) on $|C(v, R)|$, merely a bound on its expected size won't suffice; instead we would require the deviation in $|C(v, R)|$ to be small. We shall employ the algorithm by Thorup and Zwick [19] to build the desired set R such that $|C(v, R)| = O(1/p)$ for all vertices. Beginning with $R = R_p$, let V' be the set of *bad* vertices v such that $|C(v, R_p)| > 4/p$. The algorithm **Augment**(V', R_p), described in Figure 5, turns a fraction of bad vertices into *good* vertices (with cluster size $\leq 4/p$) iteratively by augmenting the set R. For every iteration in this algorithm, $R \supset R_p$, so it follows from Corollary 4.1 that each iteration will require expected $O(m + n/p^2)$ time. Furthermore, since the set R is augmented over the iterations, the value $|C(v, R)|$ will never increase for any vertex $v \in V$. Hence once a vertex becomes *good*, it will never become *bad* in future. Now for bounding the number of iterations, the following lemma will be crucial.

Lemma 4.1 *In each iteration, the number of bad vertices reduces by a factor of 2 with probability at least 1/2.*

Proof: Let V_i be the set of bad vertices (the set V') at the end of ith iteration. The iteration begins with augmenting the existing set R by a uniform sample of np vertices from V_i. The expected sum of sizes of the clusters around each vertex from V_i with respect to the augmented set R can be bounded as follows

$$\begin{aligned}
\mathbf{E}\left[\sum_{v \in V_i} |C(v, R)|\right] &= \mathbf{E}\left[\sum_{u \in V} |\, Ball(u, V_i, R)|\right] && \{\text{using Equality 1}\} \\
&\leq \mathbf{E}\left[\sum_{u \in V} |\, Ball(u, V_i, A)|\right] && \{\text{ since } A \subset R\ \} \\
&\leq n \cdot \frac{|V_i|}{pn} && \{\text{using Lemma 2.2 } (\textit{ii})\ \} \\
&= |V_i|/p
\end{aligned}$$

Algorithm I

Preprocessing

Let R be the set of vertices as defined by Theorem 4.2.

1. For each $u \in V \backslash R$, compute $Ball(u, V, R)$.
2. Compute overlap matrix $\mathcal{O}$.
3. For each $v \in R$, build a full BFS tree rooted at v in the graph.

Reporting distance between $u, v \in V$

$\mathcal{Q}(u, v)$

Figure 6: Algorithm I

In other words, starting with $|V_i|$ bad vertices in iteration i, the expected sum of sizes of all clusters around these vertices at the end of the iteration is $|V_i|/p$. Now, using Markov Inequality, the probability that this sum is bounded by $2|V_i|/p$ is at least 1/2, and in that case there can't be more than $|V_i|/2$ vertices in V_i with the cluster size greater than $4/p$. Therefore $\mathbf{Pr}[|V_{i+1}| \leq |V_i|/2] \geq 1/2$. $\square$

It follows from Lemma 4.1 that after expected $\log n$ iterations, V' would be reduced to $\emptyset$. Each iteration adds np vertices to the sample set R. Thus the expected size of the final sample R would be $np \log n$, and the terminating condition of the algorithm ensures that $C(v, R)$ is bounded by $O(\frac{1}{p})$, for each $v \in V$. Hence we can state the following Lemma.

Lemma 4.2 *Given an undirected unweighted graph $G = (V, E)$ and $p < 1$, we can compute a sample set R of expected size $np \log n$ such that $|C(v, R)|$ is bounded by $O(\frac{1}{p})$, for each $v \in V$.*

Combining Corollary 4.1 and Lemma 4.2, we can conclude the following theorem.

Theorem 4.2 *Given an undirected unweighted graph $G = (V, E)$ and $p < 1$, a set $R \subset V$ of size $O(np \log n)$ can be computed in expected $O(m \log n + \frac{n}{p^2} \log n)$ time ensuring that*
- *It takes a total of $O(m + n/p^2)$ time to compute $Ball(u, V, R), \forall u \in V \backslash R$.*
- *It takes $O(n^2 + \frac{n}{p^2})$ time to build the overlap matrix $\mathcal{O}$.*

5 Algorithms for Nearly 2-Approximate Shortest Paths

5.1 Algorithm I

Our first algorithm for computing nearly 2-approximate distances is a realization of the scheme mentioned in Section 2 and is described in Figure 5.1. Let us now analyse its time complexity. Computing BFS tree from vertices of the set R requires $O(m|R|) = O(mnp \log n)$ expected time. By applying Theorem 4.2, it follows that the total expected time for preprocessing the graph in Algorithm I is of the order of

$$O(m \log n + n^2 + \frac{n}{p^2} \log n + mnp \log n) = O(n^2 + m^{2/3} n \log n) \quad \{ \text{for } p = \frac{1}{\sqrt[3]{m}} \}.$$

Theorem 5.1 *An undirected unweighted graph $G = (V, E)$ can be preprocessed in expected $O(m^{2/3} n \log n + n^2)$ time to build a data structure of size $\Theta(n^2)$ which can report $\delta^*(u, v)$ in $O(1)$ time for any $u, v \in V$ such that*

$$\delta(u, v) \leq \delta^*(u, v) \leq 2\delta(u, v) + 1$$

5.2 Algorithm II

The preprocessing time of the first two steps in Algorithm I described above can be bounded by $O(n^2 \log n)$ with a suitable choice of p. The third step that computes BFS trees from vertices of set R requires $O(m|R|)$ time, which is certainly not $O(n^2 \log n)$ when the graph is dense. To improve its preprocessing time to $O(n^2 \operatorname{polylog} n)$, one idea is to perform BFS from R on a spanner (having $o(n^2)$ edges) of the original graph. A spanner is a subgraph that is sparse but still preserves approximate distance between vertices in the graph.

Definition 5.1 *Given $\alpha \geq 1, \beta \geq 0$, a subgraph (V, E'), $E' \subseteq E$ is said to be an (α, β)-spanner of $G = (V, E)$ if for each pair of vertices $u, v \in V$, the distance $\delta_s(u, v)$ in the spanner is bounded by $\alpha\delta(u, v) + \beta$.*

The sparsity of a spanner comes along with the stretching of the distances in the graph. So one has to be careful in employing an (α, β)-spanner (with $\alpha > 1$) in the third step, lest one should end up computing nearly 2α-approximate distances instead of nearly 2-approximate distances. To explore the possibility of using spanner in our algorithm, let us revisit our distance reporting scheme $\mathcal{Q}(u, v)$. The full BFS trees rooted at the vertices of set R serve to provide global distance information in the scheme $\mathcal{Q}(u, v)$, and they are required only when $Ball(u, V, R)$ and $Ball(v, V, R)$ are non-overlapping. In the analysis of this case, we partitioned the shortest path between u and v into three sub-paths (see Fig. 3,*Case-3*): the sub-paths of lengths a and b covered by $Ball(u, V, R)$ and $Ball(v, V, R)$ respectively, and the sub-path of length x lying between the two Balls and not covered by either of them. We showed that the distance $\delta^*(u, v)$ as reported by $\mathcal{Q}(u, v)$ is bounded by $2a + 2b + x + 2$. A comparison of this expression of $\delta^*(u, v)$ with $\delta(u, v) = a + x + b$ suggests that there is a possibility of stretching the uncovered sub-path (of length x) between the Balls by a factor of 2 and still keeping the distance reported to be nearly 2-approximate. So we may employ an (α, β)-spanner in the third step of our algorithm, provided α (the multiplicative stretch) is not greater than 2 and more importantly, for each vertex $u \in V \backslash R$, the shortest path from n_u to u as well as the shortest paths from u to all the vertices of $Ball(u, V, R)$ are preserved in the spanner. To ensure these additional features, we shall employ the parameterized spanner introduced in [5].

Definition 5.2 *Given a graph $G = (V, E)$, and a parameter $X \subset V$, a subgraph (V, E') is said to be a parameterized $(2, 1)$-spanner with respect to X if*
(i) (V, E') is a $(2, 1)$-spanner.
(ii) Every edge whose at least one endpoint is not adjacent to any vertex from the set X is surely present in the spanner too.

A parameterized $(2, 1)$-spanner for a given $X \subset V$ (as a parameter) can be constructed in $O(m + n)$ time [5]. To ensure that the spanner is a parameterized $(2, 1)$-spanner, the algorithm satisfies the following property.

Lemma 5.1 *[5] For an edge $e(u, v)$ not present in the spanner, there is a vertex $x \in X$ adjacent to u in the spanner such that there is a path from x to v in the spanner of length at most 2.*

```
Algorithm II

Preprocessing
Let $R$ be the set of vertices as defined by Theorem 4.2.

1. For each $u \in V \backslash R$, compute $Ball(u, V, R)$.
2. Compute overlap matrix $\mathcal{O}$.
3. (a) Let $(V, E')$ be a parameterized $(2,1)$-spanner with respect to $R$ for the given
       graph $G = (V, E)$.
   (b) For each $v \in R$, compute a full BFS tree rooted at $v$ in the graph $(V, E' \cup E(v))$.
       ($E(v)$ denotes the edges incident on $v$ in the original graph $G = (V, E)$)

Reporting distance between $u, v \in V$
$\mathcal{Q}(u, v)$
```

Figure 7: Algorithm II

Property (ii) of the parameterized $(2,1)$-spanner implies that if we choose R as the parameter, all the edges lying inside a Ball are present in the parameterized spanner, and hence all the shortest paths that completely lie within a Ball are also preserved. Now observe that the shortest path from n_u to u lies fully inside $Ball(u, V, R)$ except the first edge of this path which is incident on n_u. So to ensure that the shortest path from n_u to u is also preserved, it would suffice if we augment the spanner with all the edges in the original graph that are incident on n_u. The algorithm II is described in Figure 5.2.
From the discussion above, it follows that for any pair of vertices $u, v \in V$, the distance reported in Case-3 by $\mathcal{Q}(u, v)$ will be

$$\begin{aligned} \delta^*(u,v) &\leq 2(a+b) + (2x+1) + 2 \quad \{\text{since } x \text{ is stretched to } 2x+1\ \} \\ &= 2(a+x+b) + 3 \quad = \quad 2\delta(u,v) + 3. \end{aligned}$$

To analyse the running time of Algorithm II, observe that we perform BFS on a $(2,1)$-spanner. Therefore, a bound on the size of the spanner is required. We shall use the following lemma from [5].

Lemma 5.2 *[5] Let R_p be a set formed by selecting each vertex independently with probability p. For any set $R \supset R_p$, the expected size of parameterized $(2,1)$-spanner will be $O(|R|n + n/p)$.*

It follows from Lemma 4.2 that the expected size of R is $O(pn \log n)$. This fact in conjunction with the previous Lemma implies that the size of $(2,1)$-spanner is $O(n/p + n^2 p \log n)$. Hence using Theorem 4.2 the expected preprocessing time of Algorithm II is

$$O\left(m \log n + \frac{n}{p^2} \log n + \left(n^2 p \log n + \frac{n}{p}\right) np \log n\right) = O(n^2 \log^{\frac{3}{2}} n) \ \ \{\text{for } p = \frac{1}{\sqrt{n}\sqrt[4]{\log n}}\}$$

Theorem 5.2 *An undirected unweighted graph $G = (V, E)$ can be preprocessed in $O(n^2 \log^{3/2} n)$ expected time to build a data structure of size $\Theta(n^2)$ which can report $\delta^*(u, v)$ in constant time for any $u, v \in V$ such that*

$$\delta(u,v) \leq \delta^*(u,v) \leq 2\delta(u,v) + 3$$

6 Conclusion and Open Problems

Given an undirected unweighted graph $G = (V, E)$ on $|V| = n$ vertices, we can compute nearly 2-approximate distances in $O(n^2 \operatorname{polylog} n)$ time. A natural question is whether the same bound is achievable for undirected weighted graphs as well. Subsequent to the submission of this paper, Baswana and Kavitha [4], and Berman and Kasivishwanathan [6] independently answered this question in affirmative. They showed that an undirected weighted graph $G = (V, E)$ can be preprocessed in $O(n^2 \log n)$ time to compute δ^* such that for each pair $u, v \in V$

$$\delta(u, v) \leq \delta^*(u, v) \leq 2\delta(u, v) + w_{\max}(u, v)$$

where $w_{\max}(u, v)$ is the weight of the maximum weight edge on the shortest path between u and v.

References

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

[2] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all-pairs shortest paths problem. *Journal of Computer and System Sciences*, 54:255–262, 1997.

[3] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhod covers. *SIAM Journal on Computing*, 28:263–277, 1998.

[4] S. Baswana and T. Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 591–602, 2007.

[5] S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 271–280, 2004.

[6] P. Berman and S. P. Kasiviswanathan. Faster approximation of distances in graphs. In *Proceedings of 10th Workshop on Algorithms and Data Structures*, volume 4619 of *LNCS*, pages 541–552. Springer, 2007.

[7] T. 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.

[8] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28:210–236, 1999.

[9] E. Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *Journal of Association of Computing Machinery*, 47:132–166, 2000.

[10] E. Cohen and U. Zwick. All-pairs small stretch paths. *Journal of Algorithms*, 38:335–353, 2001.

[11] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[12] D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. *Siam Journal on Computing*, 29:1740–1759, 2000.

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

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

[15] L. Roditty, M. Thorup, and U. Zwick. Deterministic construction of approximate distance oracles and spanners. In *Proceedings of 32nd International Colloquim on Automata, Languages and Programming*, volume 3580 of *LNCS*, pages 261–272. Springer, 2005.

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

[17] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[18] T. Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20:309–318, 1998.

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

[20] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of Association of Computing Machinery*, 52:1–24, 2005.

[21] U. Zwick. Exact and approximate distances in graphs - a survey. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA)*, pages 33–48, 2001.

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