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

Surender Baswana[1], Vishrut Goyal[2], and Sandeep Sen[3]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany.
sbaswana@mpi-sb.mpg.de
[2] Persistent Systems Private Limited, Pune, India.
vishrut_goyal@persistent.co.in
[3] Department of Computer Science and Engineering, I.I.T. Kharagpur, India.
ssen@cse.iitkgp.ernet.in

Abstract. Let $G(V, E)$ be an unweighted undirected graph on $|V| = n$ vertices. Let $\delta(u, v)$ denote the shortest 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 randomized algorithms for computing all-pairs nearly 2-approximate distances. The first algorithm takes 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 $O(n^2 \log^{3/2})$ time, and for any $u, v \in V$ reports distance bounded by $2\delta(u, v) + 3$.

This paper also presents the first expected $O(n^2)$ time algorithm to compute all-pairs 3-approximate distances.

1 Introduction

All-pairs shortest path problem is undoubtedly one of the most fundamental algorithmic graph problem. 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 real edge-weights, the best known algorithm [6] 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 $O(n^3 \sqrt{\log \log n} / \log n)$ due to Zwick [10], which is marginally sub-cubic.

In the recent past, there has been a growing interest in designing efficient (sub-cubic running time) and simple algorithms for all-pairs *approximate* shortest-paths, and successful attempts have been made for undirected graphs. Zwick [9] provides an excellent survey on algorithms for computing approximate shortest

paths. An algorithm is said to compute all-pairs t-approximate distances, if for any pair of vertices $u, v \in V$, the distance $\delta^*(u, v)$ reported by the algorithm satisfies $1 \leq \frac{\delta^*(u,v)}{\delta(u,v)} \leq t$. In the following paragraph, we provide a very brief summary of the current state-of-the-art algorithms for all-pairs t-approximate shortest paths.

Cohen and Zwick [2], building upon the work of Dor et al. [3], designed an algorithm that given any undirected weighted graph with n vertices and m edges, computes all-pairs 2-approximate shortest paths in $O(n^{3/2}\sqrt{m})$ time and all-pairs 3-approximate shortest paths in just $O(n^2 \log n)$ time. For unweighted graphs, given arbitrarily small $\zeta, \epsilon, \rho > 0$, Elkin [4] designed 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 :

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

where β is a function of ζ, ϵ, ρ. If the two vertices $u, v \in V$ are separated by sufficiently long distances in the graph, the stretch $\frac{\delta^*(u,v)}{\delta(u,v)}$ ensured by Elkin's algorithm is quite close to $(1 + \epsilon)$. But the stretch factor may 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. Thorup and Zwick [7] introduced a remarkable data-structure called approximate distance oracle, that requires sub-cubic preprocessing time and sub-quadratic space, and yet answers an approximate-distance query in constant time (hence the name oracle). For a given integer $k \geq 2$, the space of the approximate distance oracle is $O(kn^{1+1/k})$ and it reports any $(2k - 1)$-approximate distance query in $O(k)$ time. The preprocessing time for $(2k - 1)$-approximate distance oracle is $O(mn^{1/k})$ which has been improved to $O(\min(mn^{1/k}, n^2 \log n))$ for unweighted graphs in [1]. Thorup and Zwick [7] also show that for any $t < 3$, a data-structure that answers any t-approximate distance query in constant time must occupy $\theta(n^2)$ space. This implies a lower bound of $\Omega(n^2)$ on space as well as on time complexity of any algorithm that answers any 2-approximate distance query in constant time. As mentioned above, the algorithm of Cohen and Zwick [2] establishes an upper bound of $O(n^{3/2}\sqrt{m})$ on time complexity of all-pairs 2-approximate shortest path problem.

1.1 Our Contribution

As an important contribution of this paper, we show that we can, in time $O(n^2 \operatorname{polylog} n)$, compute all-pairs *nearly* 2-approximate shortest paths for unweighted undirected graphs.

1. We first design a data-structure that, given any $u, v \in V$, 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. The expected preprocessing time required to build this data-structure is $O(m^{2/3}n \log n + n^2)$. In this way, our new algorithm, at the expense of introducing an additive error of just one unit, achieves a significant improvement in the running time over

the previous best algorithm [2] for all-pairs 2-approximate distances. The improvement is by a factor of at-least $n^{1/6}$ for the range $m > n^{3/2}$, whereas for $m < n^{3/2}$, the new algorithm takes expected $O(n^2)$ time.

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

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

   As would become clear subsequently from the paper, the additive error shows up only in some restricted worst case only. In general, the algorithm will behave very much like a 2-approximate shortest path algorithm.
3. As an additional and final contribution, this paper shows that it takes expected $O(n^2)$ time to compute 3-approximate distance oracle of size $O(n^{3/2})$.

Without any modifications, all our data-structures for reporting approximate distances can also be used to report approximate shortest-paths in optimal time.

2 A New Scheme for 2-Approximate Shortest Paths

Let $G(V, E)$ be an unweighted graph. The basic construct of our scheme is a *restricted* breadth-first-search (BFS) tree defined as Ball as follows.

Definition 1. *For a vertex u and a set $R \subset V$ of vertices, $Ball(u, R)$ denotes the set of vertices of the graph, such that the distance from u to these vertices is less than the distance from u to the nearest vertex of the set R.*

New scheme for approximate distance
Let $R \subset V$ be a set of vertices. Let n_u denote the vertex from the set R nearest to u. 1. **Global distance information** For each vertex $s \in R$, keep a BFS tree storing distance to all the vertices in the graph. 2. **Local distance information** For each vertex $u \in V \backslash R$, compute distance to all the vertices of $Ball(u, R)$ 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) or not.

The above scheme may appear similar to 3-approximate distance oracle of Thorup and Zwick [7] except the third step. It is this step that proves to be crucial in achieving 2-approximate distances.

Now we shall describe how our scheme can be used to answer a distance query with stretch 2.

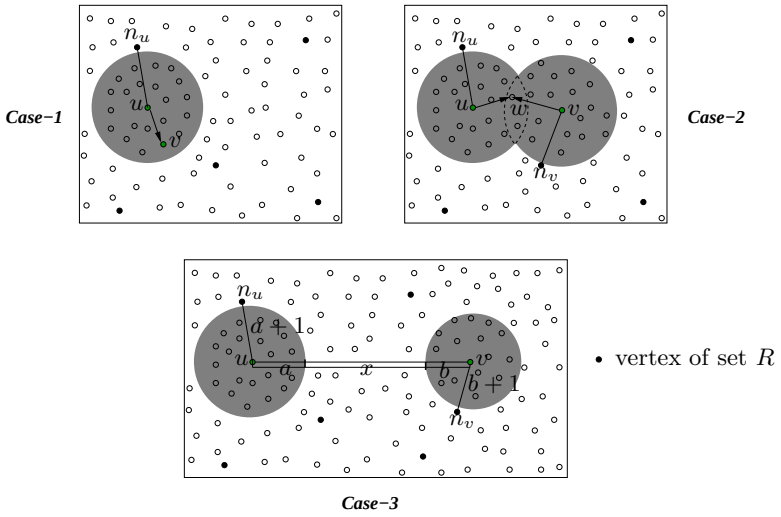

Fig. 1. Three cases in reporting distance between u and v

Answering distance query using new scheme

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

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

  report $\delta(u, v)$

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

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

- Else :

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

The query Procedure $\mathcal{Q}(u, v)$ explores all the three possible cases in the fixed order. In the first two cases, we manage to report distance using only the local distance information stored at vertices u and v. In the final and the third case, when the two Balls are non-overlapping, we use the global distance information stored at n_u and n_v.

Lemma 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 $Ball(u, R)$ and $Ball(v, R)$ respectively. The approximation factor associated with the distance reported by $\mathcal{Q}(u, v)$ depends

on which of the three steps, we report the distance at. So we analyze the three cases as follows:

Case 1 : The distance is reported in the first step of $\mathcal{Q}(u, v)$.
In this case, either u or v lie in the Ball of the other. Without loss of generality, let us assume that v lies in $Ball(u, R)$ (see Fig. 1, *Case-1*). Here we report the *exact* distance between u and v.

Case 2 : The distance is reported in the second step of $\mathcal{Q}(u, v)$.
Note that 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 $Ball(u, R)$ and $Ball(v, R)$. In other words, $\delta(u, v)$ is more than a and b. (see Fig. 1, *Case-2*).

Let w be a vertex lying in both $Ball(u, R)$ and $Ball(v, R)$. Clearly, $\delta(u, w) \leq a$ and $\delta(v, w) \leq b$. Therefore, the distance reported in this step is bounded by $a + b$, which is no more than $2\delta(u, v)$ 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, $Ball(u, R)$ and $Ball(v, R)$ are separated by distance $x \geq 1$. 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 $Ball(u, R)$ 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 $Ball(v, R)$ and has length b. Hence, the distance between u and v is $a+x+b$ for some $x \geq 1$. (see Fig. 1, *Case-3*).

In the third step, we report the 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 bounded by $2a + x + b + 1$. Similarly, analyzing the path from n_v to u passing through v, we can observe that $\delta(n_v, u)$ is bounded by $2b + x + a + 1$. Therefore, the distance reported by $\mathcal{Q}(u, v)$ is bounded as follows.

$$
\begin{aligned}
\min((\delta(u, n_u) + \delta(n_u, v)) \ , \ (\delta(v, n_v) + \delta(n_v, u))) & \\
&\leq \min(3a + x + b + 2, 3b + x + a + 2) \\
&= \min(3a + b, 3b + a) + x + 2 \\
&= 3b + a + x + 2 \quad \{\text{wlog assume that } a \geq b\} \\
&\leq 2a + 2b + x + 2 \quad \{\text{since } a \geq b\} \\
&\leq 2(a + x + b) + 1 \quad \{\text{since } x \geq 1\} \\
&= 2\delta(u, v) + 1
\end{aligned}
$$

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.

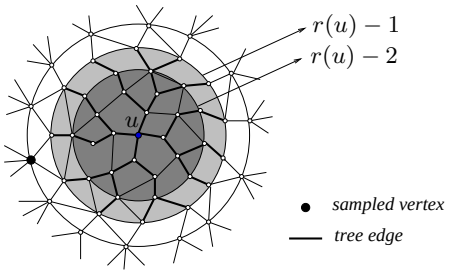

Fig. 2. To compute $Ball(u, R_p)$, we need to explore adjacency list of vertices lying in inner-shaded shell (of radius $r(u) - 2$) only.

3 Efficient Sub-Routines for Realization of the New Scheme

3.1 An Efficient Algorithm for Computing Balls

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

Let $r(u)$ denote the distance from u to n_u. It follows from Definition 1 that the vertices of $Ball(u, R_p)$ and their distance from u can be computed by building a BFS tree at u up to level (distance) $r(u)-1$. Therefore, prior to the computation of $Ball(u, R_p)$, we compute $r(u)$. In fact, the following procedure shows that it requires just a single BFS traversal to compute $r(u)$ and n_u, for all $u \in V \backslash R_p$.

Add a dummy vertex y to the given graph and connect it to all the vertices of set R_p. Compute a full BFS tree rooted at y. If a vertex u lies at level ℓ (hence at distance ℓ from y) in this BFS tree, it is at distance $\ell - 1$ from n_u, that is, $r(u) = \ell - 1$. In addition to $r(u)$, we can also compute n_u for each vertex u from this BFS tree.

If $r(u) = 1$, then $Ball(u, R_p)$ consists of vertex u only and we are done. For the case when $r(u) \geq 2$, we build a BFS tree upto level $r(u) - 1$ to compute $Ball(u, R_p)$, 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 $v_1(= u), v_2, \cdots, v_n$ be the sequence of vertices of the given graph arranged in non-decreasing order of their distance from u. Note that computing BFS tree upto level $r(u) - 1$ requires exploring the adjacency list of vertices up to level $r(u) - 2$ only (see Fig. 2). Therefore, in the computation of $Ball(u, R_p)$, we shall explore adjacency list of v_i 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 sampled independently). Following our charging scheme mentioned above, exploring adjacency list of vertex v_i would contribute $O(d'(v_i))$ to the computation time of $Ball(u, R_p)$, where $d'(v_i)$ is the number of edges incident on v_i from vertices $\{v_j | j > i\}$. So the expected cost of computing $Ball(u, R_p)$ is

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

Theorem 1. *Given an 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 $Ball(u, R_p)$, for all $u \in V \backslash R_p$ in expected time $O(m + \frac{n}{p^2})$.*

In a similar way, it can be shown that the expected number of vertices in $Ball(u, R_p)$ is $O(1/p)$ (observe that for vertex v_i to belong to $Ball(u, R_p)$, the event $\mathcal{E}_1^i$ must happen).

Lemma 2. *Given an unweighted graph $G(V, E)$, a uniformly random sample $R \subset V$ of size $\sqrt{n}$ induces a bound of $\sqrt{n}$ on expected size of $Ball(u, R)$.*

It follows from Definition 1 that $Ball(u, X) \subset Ball(u, Y)$, for all $Y \subset X \subset V$. Therefore, our algorithm would require less time to compute $Ball(u, X)$ than to compute $Ball(u, Y)$. So we can state the following corollary based on Theorem 1.

Corollary 1. *Given an 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 $R \supset R_p$, it takes expected $O(m + \frac{n}{p^2})$ time to compute $Ball(u, R)$, for all $u \in V \backslash R$.*

3.2 Computing Overlap Matrix $\mathcal{O}$

To determine, for a pair of vertices $u, v \in V$, whether there exists a vertex common to both $Ball(u, R)$ and $Ball(v, R)$, we keep a matrix $\mathcal{O}$ such that $\mathcal{O}[u, v]$ is null if $Ball(u, R) \cap Ball(v, R) = \emptyset$, otherwise $\mathcal{O}[u, v]$ stores a vertex that belongs to both the Balls. To build the matrix $\mathcal{O}$ efficiently, we form the following sets,

$$C(v, R) = \{u \in V | v \in Ball(u, R)\}.$$

It is easy to observe that we can form the sets $\{C(v, R) | v \in V\}$ by a single scan of the sets $\{Ball(u, R) | u \in V\}$. 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 ∈ V\R do
    For each u ∈ C(v, R) do
        For each w ∈ C(v, R) do
            O[u, w] ← 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 compute the overlap matrix $\mathcal{O}$ in $O(n/p^2 + n^2)$ time (which also matches the time required to compute Balls), we would need a set $R \subset V$ which would ensure that $|C(v,R)| = O(1/p)$, for all $v \in V$. We shall employ the random sampling scheme given by Thorup and Zwick [8] to compute the desired sample R as follows.

Procedure for computing the sample set R

```
Procedure sample(G, p) {
    R = ∅; V' = V;
    While (V' ≠ ∅)
        Add a uniform sample of size np from V' to R;
        For every u ∈ V\R do
            Compute Ball(u, R);
        For every v ∈ V\R do
            C(v, R) ← {u ∈ V | v ∈ Ball(u, R)};
        V' ← {v ∈ V | |C(v, R)| > 4/p};
    Return R;
}
```

For the first iteration, R is a uniform sample from V, that is, $R = R_p$. So using Theorem 1, the first iteration requires expected $O(m + n/p^2)$ time. Note that in each subsequent iteration, the sample R only grows. Hence it follows from Corollary 1 that each subsequent iteration will also require expected $O(m+n/p^2)$ time. It is shown in [8] that in every iteration, the size of V' decreases by a factor of 2 with probability at-least $1/2$. Hence after expected $\log n$ iterations, V' would be reduced to $\emptyset$. Thus, the expected size of the final sample R would be $np \log n$ and $C(v,R)$ will be bounded by $O(\frac{1}{p})$, for each $v \in V$.

Theorem 2. *Given an 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 $Ball(u,R), \forall u \in V \backslash R$.*
- *It takes $O(n^2 + \frac{n}{p^2})$ time to build the overlap matrix $\mathcal{O}$.*

4 Algorithms for Nearly 2-Approximate Shortest Paths

4.1 Algorithm I

Our first algorithm for computing nearly 2-approximate distances is a realization of the scheme mentioned in Sect. 2.

Algorithm I

Preprocessing

Let R be the set of vertices as defined by Theorem 2.

1. For each $u \in V \backslash R$, compute $Ball(u, R)$.
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 $u, v \in V$

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

Computing BFS tree from vertices of the set R requires $O(m|R|) = O(mnp \log n)$ expected time. Hence applying Theorem 2, it follows that the total expected time for preprocessing the graph in Algorithm I is given by

$$m \log n + n^2 + \frac{n}{p^2} \log n + mnp \log n = n^2 + m^{2/3} n \log n \qquad \{ \text{ for } p = \tfrac{1}{\sqrt[3]{m}} \}.$$

Theorem 3. *An unweighted graph $G(V, E)$ can be preprocessed in $O(m^{2/3} n \log n + n^2)$ expected time to output a data-structure of size $O(n^2)$ that, given any $u, v \in V$, requires constant time to report distance $\delta^*(u, v)$ satisfying*

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

The previous best known algorithm [2] for computing 2-approximate distances requires $O(n^{3/2}\sqrt{m})$ running time. Thus, in the worst case, we have been able to improve the running time by a factor of $O(n^{1/6})$ at the expense of introducing an additive error of just one unit.

4.2 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 $(n^2 \operatorname{polylog} n)$, 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 2. *Given $\alpha \geq 1, \beta \geq 0$, a subgraph $G(V, E')$, $E' \subset E$ 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 u and v belong to non-overlapping Balls. In the analysis of this case, we partitioned the shortest path between u and v into three sub-paths (see Fig. 1, *Case-3*): the sub-paths of lengths a and b covered by $Ball(u, R)$ and $Ball(v, R)$ respectively, and the sub-path of length x lying between the two Balls and not covered by either Ball. 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 $Ball(u, R)$ are preserved in the spanner. To ensure these additional features, we shall employ the parameterized spanner introduced in [1].

Parameterized (2,1)-Spanner [1]

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

An $O(m)$ time algorithm has been presented in [1] to compute a parameterized $(2, 1)$-spanner for a given $X \subset V$ (as a parameter). To ensure that the spanner is a parameterized $(2, 1)$-spanner, the algorithm establishes the following lemma.

Lemma 3. *[1] 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 no more than 2.*

The feature (ii) of the parameterized $(2, 1)$-spanner suggests 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 lie within a Ball are preserved too. Now observe that the shortest path from n_u to u lies fully inside $Ball(u, R)$ 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.

Algorithm II

Preprocessing

Let R be the set of vertices as defined by Theorem 2.

1. For each $u \in V \backslash R$, compute $Ball(u, R)$.
2. Compute overlap matrix $\mathcal{O}$.
3. (a) Let $G(V, E')$ 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 $G(V, E' \cup E(v))$. ($E(v)$ denotes the edges incident on v in the original graph $G(V, E)$)

Reporting distance between $u, v \in V$

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

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 analyze 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 [1].

Lemma 4. *[1] 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 $O(|R|n + n/p)$.*

Lemma 4 implies that the size of $(2, 1)$-spanner with parameter R (as determined in Sect. 3.2) would be $O(n/p + nnp \log n)$. Hence the expected preprocessing time of Algorithm II will be of the order of

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

Theorem 4. *An unweighted graph $G(V, E)$ can be preprocessed in $O(n^2 \log^{3/2} n)$ expected time to output a data-structure of size $O(n^2)$ that, given any $u, v \in V$, requires constant time to report distance $\delta^*(u, v)$ satisfying*

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

5 Algorithm for 3-Approximate Shortest Paths

Algorithm *3-approx*

Preprocessing

Let R be a set formed by picking each vertex with probability $1/\sqrt{n}$.

1. For each $u \in V \backslash R$, compute $Ball(u, R)$.
2. (a) Let $G(V, E')$ be a parameterized $(2, 1)$-spanner with respect to R.
   (b) For each $v \in R$, compute a full BFS tree rooted at v in $G(V, E' \cup E(v))$.

Reporting distance between vertices $u, v \in V$

If $v \in Ball(u, R)$ or $u \in Ball(v, R)$,
report $\delta(u, v)$
Else
report **minimum** of $(\delta(u, n_u) + \delta_s(n_u, v))$ and $(\delta(v, n_v) + \delta_s(n_v, u))$
(note that $\delta_s(x, y)$ denotes distance between x and y in the underlying spanner.)

Lemma 5. *The Algorithm 3-approx reports all-pairs 3-approximate distance.*

Proof. Let neither $u \in Ball(v, R)$ nor $v \in Ball(u, R)$. We have two cases now.
Case 1: $\underline{Ball(u, R) = \{u\}}$
Let $v_0(= u), v_1, \cdots, v_l(= v)$ be the shortest path between u and v. Since $Ball(u, R)$ consists of vertex u only, u must be adjacent to n_u. Also by Lemma 3, there is a path of length at-most 2 units between n_u and v_1 in the parameterized spanner. Furthermore, the distance $\delta_s(v_1, v_l)$ between v_1 and v_l in the spanner is no greater than $3(l-1)$. Hence $\delta_s(n_u, v) \leq 3(l-1) + 2$. Since $\delta(u, n_u) = 1$, the distance reported by the Algorithm *3-approx* is no more than $3l$.

Case 2: $\underline{Ball(u, R) \neq \{u\}}$
Consider $Ball(v, R)$. Let its radius be $a \geq 0$. The vertex u lies outside this Ball. The shortest path from v to u can be visualized as consisting of two segments (see Fig. 3) : the sub-path $\mathcal{P}_{vw}$ of length a lying inside the Ball and the sub-path

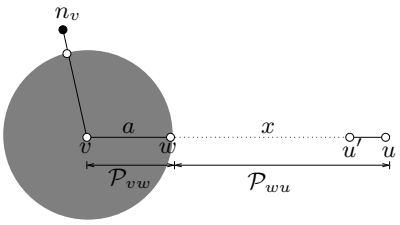

Fig. 3. Analyzing the path from v to u when $Ball(u, R) \neq \{u\}$

$\mathcal{P}_{wu}$ of length $x \geq 1$ outside the Ball. Let the length of path $\mathcal{P}_{wu}$ be stretched to x' in the parameterized spanner. Since the parameterized spanner preserves all-paths within a Ball, therefore, the distance $(\delta(v, n_v) + \delta_s(n_v, u))$ reported by the algorithm would be $3a + 2 + x'$. To ensure that this distance is no more than three times the actual distance $\delta(u, v) = a + x$, all we need to show is that $x' \leq 3x - 2$. Let $e(u', u)$ be the last edge of the path $\mathcal{P}_{wu}$. Now observe that $Ball(u, R) \neq \{u\}$ implies that $Ball(u, R)$ has radius at-least one. So the edge $e(u', u)$ must be present in the spanner (see feature (ii) of the parameterized spanner). Now the part of the path $\mathcal{P}_{wu}$ excluding the edge $e(u', u)$ is of length $x - 1$, and can't be stretched to more than $3(x-1)$ in the spanner. Hence $x' \leq 3(x-1) + 1 = 3x - 2$, and we are done.

The set R used in the Algorithm *3-approx* is a uniform random sample of $\sqrt{n}$ vertices. Hence using Theorem 1, it follows that a total of $O(n^2)$ expected time is required to compute $Ball(u, R), \forall u \in V \backslash R$. Also Lemma 4 implies that the number of edges in $(2, 1)$-spanner with parameter R is $O(n\sqrt{n})$. So the expected time required for building full BFS trees on all the vertices of R in the $(2, 1)$-spanner is $O(n^2)$. Hence the expected preprocessing time of the Algorithm *3-approx* is $O(n^2)$.

Space requirement : Keeping distance information from each $x \in R$ to all the other vertices requires a total of $O(n^{3/2})$ storage. For each vertex $u \in V \backslash R$, we can store the vertices belonging to $Ball(u, R)$ along with their distance from u in a 2-level hash table of [5] with optimal size $O(|Ball(u, R)|)$. Using this hash table, it can be determined whether $v \in Ball(u, R)$ or not in worst case constant time. So the space requirement of the data-structure of our algorithm (3-approximate distance oracle) given above is $O(n^{3/2})$, which is optimal as shown in [7].

Theorem 5. *An unweighted graph $G(V, E)$ can be preprocessed in expected $O(n^2)$ time to output a data-structure of size $O(n^{3/2})$ that can answer any 3-approximate distance query in constant time.*

6 Conclusion and Open Problems

Given an undirected unweighted graph $G(V, E)$ on $|V| = n$ vertices, we can compute all-pairs 3-approximate distances in $O(n^2)$ time. More importantly, we can compute nearly 2-approximate distances in $O(n^2 \operatorname{polylog} n)$ time. This upper bound is quite close to the $\Omega(n^2)$ lower bound on computing 2-approximate distances. It would be quite interesting to remove the additive error from our algorithm completely or to prove an $\Omega(n^{2+\epsilon})$ lower bound for computing all-pairs 2-approximate distances.

References

1. 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.
2. E. Cohen and U. Zwick. All-pairs small stretch paths. *Journal of Algorithms*, 38:335–353, 2001.
3. D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. *Siam Journal on Computing*, 29:1740–1759, 2000.
4. M. Elkin. Computing almost shortest paths. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 53–62, 2001.
5. M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $o(1)$ worst case time. *Journal of ACM*, 31:538–544, 1984.
6. S. Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312:47–74, 2004.
7. M. Thorup and U. Zwick. Approximate distance oracle. In *Proceedings of 33rd ACM Symposium on Theory of Computing (STOC)*, pages 183–192, 2001.

8. M. Thorup and U. Zwick. Compact routing schemes. In *13th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10, 2001.
9. 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.
10. U. Zwick. Slightly improved sub-cubic bound for computing all-pairs shortest paths. In *15th Annual International Symposium on Algorithms and Computation (ISAAC), to appear*, 2004.