Approximate Distance Oracles for Unweighted Graphs in $\tilde{O}(n^2)$ time

Surender Baswana* Sandeep Sen†

Abstract

Let $G(V, E)$ be an undirected weighted graph with $|V| = n, |E| = m$. Recently Thorup and Zwick introduced a remarkable data-structure that stores all-pairs approximate distance information implicitly in $o(n^2)$ space, and yet answers any approximate distance query in *constant* time. They named this data-structure *approximate distance oracle* because of this feature. Given an integer $k > 1$, a $(2k-1)$-approximate distance oracle requires $O(kn^{1+1/k})$ space and answers a $(2k-1)$-approximate distance query in $O(k)$ time. Thorup and Zwick showed that a $(2k-1)$-approximate distance oracle can be computed in $O(kmn^{1/k})$ time, and posed the following question : *Can $(2k-1)$-approximate distance oracle be computed in $\tilde{O}(n^2)$ time ?*

In this paper, we answer their question in affirmative for unweighted graphs. We present an algorithm that computes $(2k-1)$-approximate distance oracle for a given unweighted graph in $\tilde{O}(n^2)$ time. One of the new ideas used in the improved algorithm also leads to the first linear time algorithm for computing an optimal size $(2, 1)$-spanner of an unweighted graph.

1 Introduction

Computing all-pairs shortest paths (APSP) in a graph is one of the most fundamental algorithmic graph problem. There exist classical algorithms that require $\tilde{O}(mn)$ time for solving this problem, where m and n are respectively the number of edges and vertices in the given graph. There also exist algorithms based on fast matrix multiplication that achieve sub-cubic time. In particular, using the fastest known matrix multiplication algorithms [3]), the best bound for computing all-pairs shortest paths is $O(n^{2.575})$ [11]. However, there is still no combinatorial algorithm that could achieve $O(n^{3-\epsilon})$ running time for APSP problem.

In recent past, many simple combinatorial algorithms have been designed that compute all-pairs approximate shortest paths (APASP) for undirected graphs. These algorithms achieve significant improvement in the running time compared to those designed for APSP, but the distance reported has some additive or/and multiplicative error. An algorithm is said to compute all pairs α-approximate shortest paths with additive error β, if for each pair of vertices $u, v \in V$, the distance reported is bounded by $\alpha\delta(u, v)+\beta$, where $\delta(u, v)$ denotes the actual distance between u and v. For unweighted graphs, Dor et al. [4] developed an algorithm that requires $\tilde{O}(kn^{2-\frac{1}{k}}m^{\frac{1}{k}})$ time to compute all-pairs shortest paths with additive error $2(k-1)$. They also showed that all-pairs 3-approximate shortest paths can be computed in $\tilde{O}(n^2)$ time. Cohen and Zwick [2] later extended this result to weighted graphs. The output of all these algorithms is an $n \times n$ matrix that stores pairwise approximate distance for all the vertices explicitly to answer each distance query in constant time.

An equally important measure of efficiency of an algorithm is its space requirement. In the context of APASP problem, therefore, the following question arises : *Is there an implicit way of storing all-pairs approximate distance information that takes sub-quadratic space and still answers any query in $O(1)$ time ?* Thorup and Zwick [9] gave a novel algorithm that achieves simultaneous improvement in running time (sub-cubic) as well as space (sub-quadratic), and still answers any approximate distance query in constant time. For any $k \geq 1$, their algorithm runs in $O(kmn^{1/k})$ time to output a data-structure of size $O(kn^{1+1/k})$ that would answer any $(2k-1)$-approximate distance query in $O(k)$ time. They named the data-structure *approximate distance oracle*. The space requirements of these approximate distance oracles are optimal for $k = 1, 2, 3, 5$, and are conjectured to be optimal for any value of k.

Therefore, there are two efficient algorithms for computing all-pairs 3-approximate shortest paths. The first algorithm [2] runs in $\tilde{O}(n^2)$ time and outputs a matrix of size $\theta(n^2)$ that stores pairwise 3-approximate distances. The second algorithm [9] runs in $O(m\sqrt{n})$ time and computes a 3-approximate distance oracle of size $O(n^{3/2})$. Thorup and Zwick posed the following question : *Can a 3-approximate distance oracle of size $O(n^{3/2})$ be computed in $\tilde{O}(n^2)$ time?*

*Dept. of Comp. Sc. and Engg., I.I.T. Delhi, Hauz Khas, New Delhi. Work was supported in part by a fellowship from Infosys Technologies Ltd., Bangalore. Email : sbaswana@cse.iitd.ernet.in

†Dept. of Comp. Sc. and Engg., I.I.T. Delhi, Hauz Khas, New Delhi. Email : ssen@cse.iitd.ernet.in

1.1 Our contribution This paper answers the question of Thorup and Zwick in affirmative for unweighted graphs. In fact, for any $k \geq 1$, we design an algorithm that takes $O(\min(n^2 \ln n, kmn^{1/k})$ time to compute a $(2k-1)$-approximate distance oracle of size $O(kn^{1+1/k})$ for a given unweighted graph.

Another result of this paper is the first linear time algorithm for computing a $(2,1)$-spanner of (optimal) size $O(n^{3/2})$. For an unweighted graph $G(V,E)$, a sub-graph $G(V,E^*)$, $E^* \subset E$ is said to be an (α,β)-spanner if for each pair of vertices $u, v \in V$, the distance $\delta^*(u,v)$ between u and v in the sub-graph is at-most $(\alpha \cdot \delta(u,v) + \beta)$.

Previously existing linear time algorithms [1, 8] compute spanners of stretch three or more. Elkin and Peleg [6] presented algorithm for computing $(1+\epsilon,\beta)$ spanner, but with $O(n^{2+\mu})$ running time.

1.2 An overview of the previous algorithm and new techniques To give an intuition of the existing algorithm, and an exposition to the new techniques used, we first address 3-approximate distance oracle.

For a given undirected graph, storing distance information from each vertex to all the vertices requires $\theta(n^2)$ space. To achieve sub-quadratic space, the following simple idea comes to mind.

$\mathcal{I}$: *From each vertex, if we store distance information to a* small *number of vertices, can we still be able to report distance between any pair of vertices ?*

Thorup and Zwick [9] showed that the above idea can indeed be realized but at the expense of reporting approximate, instead of exact, distance as an answer to a distance query. Based on the idea $\mathcal{I}$, the scheme for computing 3-approximate distance oracle is outlined as follows.

1. Let S be a small set of $o(n)$ vertices. For each vertex $u \in V$, store the distances to all the vertices of the sample set S.

2. For each vertex $u \in V$, let n_u be the vertex nearest to u among all the vertices of set S, and let $N(u)$ be the set of all the vertices of the graph G that lie nearer to u than the vertex n_u. Store the vertices of set $N(u)$ along with their distance from u.

Let $u, v \in V$ be any two vertices whose intermediate distance is to be determined approximately. If either u or v belong to set S, we can report exact distance between the two. Otherwise also exact distance $\delta(u,v)$ will be reported if v lies in $N(u)$ or vice versa. The only case, that is left, is when neither $v \in N(u)$ nor $u \in N(v)$. In this case, we report $\delta(u,n_u) + \delta(v,n_u)$ as approximate distance between u and v. This distance is bounded by $3\delta(u,v)$ using triangle inequality as shown below (see Figure 1).

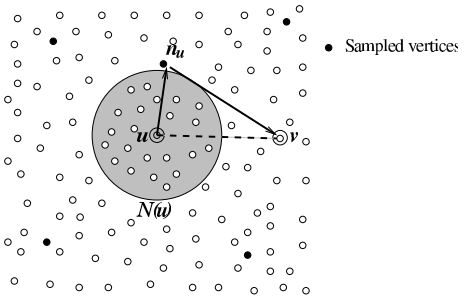

Figure 1: v is farther to u than n_u, bounding $\delta(n_u,v)$ using triangle inequality

$$\begin{aligned}
\delta(u,n_u) + \delta(v,n_u) &\leq \delta(u,n_u) + (\delta(v,u) + \delta(u,n_u)) \\
&= 2\delta(u,n_u) + \delta(u,v) \\
&\quad \{\text{since graph is undirected }\} \\
&\leq 2\delta(u,v) + \delta(u,v) = 3\delta(u,v) \\
&\quad \{\text{since } v \text{ is farther to } u \text{ than } n_u\}
\end{aligned}$$

The problem is to ensure a sub-quadratic bound on the size of this oracle. In particular, we have to show that the number of vertices in set $N(u)$ would be $o(n)$. Thorup and Zwick [9] show that if set S is a set of $\sqrt{n}$ vertices chosen uniformly from the graph, we can indeed get a bound of $O(\sqrt{n})$ on expected size of $N(u)$, and thus get a 3-approximate oracle of expected $O(n^{3/2})$ size.

The computation of 3-approximate distance oracle thus involves two tasks. The first task is to compute the vertices that form the set $N(u)$ and their distances from u, for each u. The second task is to compute distance from each vertex of set S to all the vertices in the graph. Thorup and Zwick [9] present $O(m\sqrt{n})$ time algorithms for each of the two tasks. It can be seen that the running time is not $\tilde{O}(n^2)$ for dense graphs.

We show that in addition to a uniformly chosen $\sqrt{n}$ vertices, if the set S also contains a dominating set for the vertices with degree more than $\sqrt{n}$, the first task can be performed in $\tilde{O}(n^2)$ time provided that the graph is unweighted.

The second task involves computation of shortest paths from each vertex of S to all the vertices. Note that the computation of shortest paths from a vertex to all the vertices takes $\theta(m)$ time, therefore it appears that the second task would require at-least $\theta(m|S|)$ time, which could be $\theta(n^{2.5})$ for dense graphs. To perform the second task in $O(n^2)$ time for dense graphs, one approach could be to first compute a sparse spanner of

the graph with at-most $n^{3/2}$ edges, and then compute distance from each $u \in S$ to all the vertices in this sparse spanner (instead of the original graph). This approach has been used by Elkin in his work [5]. However, from the description given above, to get a bound of $3\delta(u,v)$ on the distance reported between u and v, it is crucial that the distance between u and the vertex n_u must be preserved in the spanner. But sparseness of a spanner is achieved at the expense of stretching the distance between vertices. Therefore, in order to improve the running time of the second task so as to compute $(2k-1)$-approximate distance oracle in $\tilde{O}(n^2)$ time if we employ a typical sparse spanner, it would invariably increase the stretch of the oracle beyond $(2k-1)$.

Parameterized Spanner : We present a linear time algorithm that takes an unweighted graph G as input and a parameter $R \subset V$ to compute a sparse spanner of the graph G. The characteristic of this spanner is that each edge of the graph whose at-least one end-point is not adjacent to any vertex from set R is surely present in the spanner too. We use this property of the parameterized spanner cleverly to ensure that for the 3-approximate distance oracle, the distance between u and its nearest vertex from set S is preserved for each $u \in V$ in the spanner. We also ensure sparseness of the spanner by using random sampling to choose the parameter $R \subset V$. For computing $(2k-1)$-approximate distance oracle, we use a parameterized $(2,1)$-spanner of size $O(n^{\frac{2k-1}{k}})$ computed with a set of $O(n^{1/k})$ vertices as parameter.

2 A linear time algorithm for computing a parameterized $(2,1)$-spanner

Let $G(V,E)$ be the given undirected unweighted graph. For a sample $R \subset V$ of vertices, let V_R be the set of all those vertices that either belong to the set R or are adjacent to at-least one vertex from the set R. Let $G(V_R, E_R)$ denote the sub-graph induced by the vertices V_R, thus E_R consists of all those edges both of whose end-points belong to V_R. Let E'_R denote the remaining edges $E - E_R$. We first present an algorithm $\mathcal{A}(R)$ that computes a sparse $(2,1)$-spanner for the sub-graph $G(V_R, E_R)$.

Algorithm $\mathcal{A}(R)$ for $(2,1)$-spanner of $G(V_R, E_R)$
Input : sub-graph $G(V_R, E_R)$ for a sample $R \subset V$
Output : $G(V_R, E^*_R)$ as a $(2,1)$-spanner of $G(V_R, E_R)$

Initially $E^*_R = \emptyset$, and the algorithm adds edges to E^*_R in two steps.

1. *Forming clusters* :
   Partition the set V_R into subsets of vertices called *clusters* as follows. Each cluster will have exactly one vertex $x \in R$, called center of the cluster. Note that each vertex $v \in V_R - R$ has at-least one neighbor from set R, so assign the vertex v to the cluster centered at that neighbor. (Break the tie arbitrarily in case v has multiple neighbors from R). Add all the edges incident on vertices of R to the set E^*_R. Finally remove all the intra-cluster edges from set E_R, and pass the clustering of set V_R and the remaining edges, to the second step.

2. *Adding edges between vertices and clusters* :
   For each vertex $v \in V_R$, we group all its neighbors into their respective clusters. There will be at-most $|R|$ neighboring clusters of v. For each cluster adjacent to vertex v, we select an edge from the set of edges between (the vertices of) the cluster and v, and add it to the set E^*_R.

It is easy to verify that implementation of the two steps of the algorithm merely requires two traversals of the adjacency list of each vertex. Thus the running time of the algorithm is $O(m)$.

LEMMA 2.1. *The number of edges of the spanner $G(V_R, E^*_R)$ is bounded by $O(|R| \cdot |V_R|)$.*

LEMMA 2.2. *Let $e(u,v) \in E_R$ be an edge not present in the spanner-edges E^*_R. There is a one-edge or a two-edge path in the spanner (V_R, E^*_R) between the vertex u and the center of the cluster containing vertex v, and vice versa.*

Proof. There are two cases depending upon whether u and v belong to same cluster or different clusters.

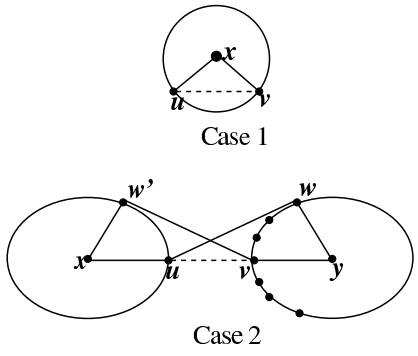

Figure 2: The two cases depending upon whether u, v belong to same/different clusters

Case 1 : *(u and v belong to same cluster)*
Let the cluster centered at $x \in R$ contains both the vertices u and v. Therefore, the edges $e(u,x)$ and

$e(v,x)$ are present in the spanner (refer to step 1.(a) of algorithm $\mathcal{A}(R)$. So there is one edge path from v to the center of the cluster containing u, and vice versa.
Case 2 : *(u and v belong to different clusters)*
Let x and y be the centers of the clusters containing u and v respectively. From step 2 of the algorithm, it follows that since u is adjacent to cluster centered at y, therefore, an edge between u and some vertex, say w of this cluster has been added to E^*_R. From step 1 of the algorithm, it follows that the edge $e(w,v)$ is also present in the spanner. So there is a two-edge path $u-w-y$ between u and y. In a similar way, there is a two-edge path $v-w'-x$ between v and the vertex x.

On the basis of the lemma given above, we shall now prove the following Lemma to show that the spanner $G(V_R, E^*_R)$ is indeed a $(2,1)$-spanner of $G(V_R, E_R)$.

LEMMA 2.3. *For each pair of vertices $u, v \in V_R$, if they are connected by some path in the sub-graph $G(V_R, E_R)$, the distance $\delta^*(u,v)$ between the two in the spanner $G(V_R, E^*_R)$ is related to their actual distance in $G(V_R, E_R)$ by the following inequality :* $\delta^*(u,v) \leq 2\delta(u,v)+1$

Proof. For a vertex $x \in V_R$, let $C(x)$ denote the center of the cluster to which the vertex x belongs.

Let $u, w \in V_R$ be any two vertices separated by a shortest path of length j in the sub-graph $G(V_R, E_R)$. Let $s_{uw} : \{u = v_0, v_1, \cdots, v_j = w\}$ be the sequence of vertices (as they appear) on the shortest path between u and w. Consider another sequence $c_{uw} : \{v_0, C(v_1), v_2, C(v_3), \cdots\}$ formed by replacing each vertex v_{2i+1} of the sequence s_{uw} by $C(v_{2i+1})$, for $i \leq j/2$.

It follows from Lemma 2.2 that for each edge $e(y,z) \in E_R$, there is a path between $C(y)$ and z of length at-most two in the spanner, and there is also a path between y and $C(z)$ of length at-most two in the spanner. Hence each pair of consecutive vertices in the sequence c_{uw} is connected by a path of length at-most two in the spanner. So there is a path of length at-most $2j$ between u and the last vertex of the sequence c_{uw} in the spanner. Note that the last vertex of the sequence c_{uw} is w or $C(w)$ depending on whether the path length j is even or odd. If the path length is even, it implies that $\delta^*(u,w) \leq 2j$. Otherwise (j is odd) note that either $C(w)$ is the vertex w itself or there is an edge between $C(w)$ and w in the spanner $G(V_R, E^*_R)$. Thus $\delta^*(u,v) \leq 2j+1$. Combining the two cases (even and odd j), we can conclude that $\delta^*(u,v) \leq 2\delta(u,v)+1$.

Consider the sub-graph $G(V, E'_R \cup E^*_R)$ formed by adding to the spanner computed by algorithm $\mathcal{A}(R)$, all the remaining vertices $V - V_R$ along with all their edges E'_R (recall that $E'_R = E - E_R$). Using arguments similar to Lemma 2.3, the following lemma shows that this sub-graph is indeed a $(2,1)$-spanner of the original graph $G(V,E)$.

LEMMA 2.4. *For a given $R \subset V$, if $G(V_R, E^*_R)$ is the $(2,1)$-spanner of sub-graph $G(V_R, E_R)$ reported by the algorithm $\mathcal{A}(R)$, then the sub-graph $G(V, E'_R \cup E^*_R)$ is also a $(2,1)$-spanner of the original graph $G(V,E)$.*

Proof. The proof is similar to that of Lemma 2.3. Let $C(x)$ denotes the vertex x if $x \notin V_R$, otherwise $C(x)$ denotes the center of the cluster containing vertex x.

Consider an edge $e(x,y) \in E$. Note that the set E'_R has all the edges whose at-least one end-point belongs to $V - V_R$. So if $e(x,y)$ is not present in set $E'_R \cup E^*_R$, it must be in the sub-graph $G(V_R, E_R)$, for whom $G(V_R, E^*_R)$ is the spanner. Therefore the statement of Lemma 2.2 can be restated as follows : *For each edge $e(x,y) \in E$, there is a path of length at-most two between $C(x)$ and y in the subgraph $G(V, E'_R \cup E^*_R)$, and vice versa.* To ensure that $\delta^*(u,w) \leq 2\delta(u,w)+1$ in the subgraph $G(V, E'_R \cup E^*_R)$, consider the sequence $s_{uw} : \{u = v_0, v_1, \cdots, v_j = w\}$ of vertices of the shortest path between u and v in the order they appear on it. Now consider another sequence $c_{uw} : \{v_0, C(v_1), v_2, C(v_3), \cdots\}$ formed by replacing each vertex v_{2i+1} of the sequence s_{uw} by $C(v_{2i+1})$, for $i \leq j/2$; and proceed exactly as in case of Lemma 2.3 to complete the proof.

For a given graph $G(V,E)$ and parameter R, henceforth we shall use G_R to denote the parameterized $(2,1)$-spanner $G(V, E'_R \cup E^*_R)$.

3 Preliminaries

In this section, we give definitions, notations and lemmas to be used in the context of our description of approximate distance oracles.

3.1 Ball around a vertex An important construct of the approximate distance oracles is $Ball(\cdot)$.
Definition : *For a vertex u, and subsets X, Y of vertices, $Ball(u, X, Y)$ is the set consisting of all those vertices of the set X that lie closer to u than any vertex from set Y. (see Figure 3)*

It follows from the definition given above that $Ball(u, X, \emptyset)$ is the set X itself, whereas $Ball(u, X, X) = \emptyset$. It can also be seen that the 3-approximate distance oracle outlined previously stores distance from each $u \in V$ to all the vertices of $Ball(u, V, S)$ and $Ball(u, S, \emptyset)$.

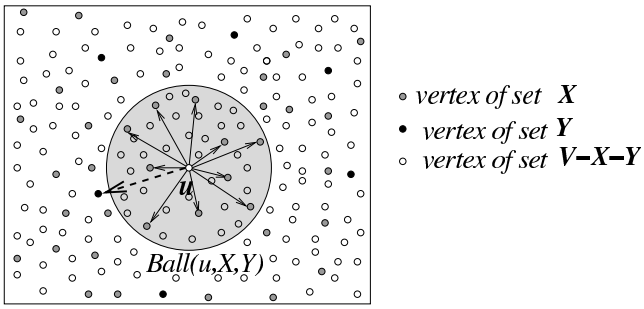

Figure 3: The vertices pointed by solid-arrows constitute $Ball(u, X, Y)$

LEMMA 3.1. *Given a graph $G(V,E)$, let the set Y be formed by picking each vertex of a set $X \subset V$ independently with probability γ. The expected size of $Ball(u,X,Y)$ is bounded by $1/\gamma$.*

Proof. Let $s(u) = \{v_1, v_2, \cdots, v_l\}$ be the sequence consisting of all the vertices of set X arranged in non-decreasing order of their distance from u. The vertex v_j would belong to $Ball(u,X,Y)$ if none of the vertices preceding it in the sequence $s(u)$ is picked to Y. Since each vertex is picked independently with probability γ, so the probability that v_j belongs to $Ball(u,X,Y)$ is $(1-\gamma)^{j-1}$. Thus the expected number of vertices in the set X that belong to $Ball(u,X,Y)$ is $\sum_{j=1}^{j=l}(1-\gamma)^{j-1} \leq \frac{1}{\gamma}$.

3.2 Notations and Lemmas In a given graph $G(V,E)$ and $k > 1$, let $\{\mathcal{R}_k^i | i \leq k\}$ denote a hierarchy of random vertex-sets defined as follows :

- $\mathcal{R}_k^1 = V$.
- $\mathcal{R}_k^{i+1}$ for $1 \leq i < k$ is formed by picking each vertex from the set $\mathcal{R}_k^i$ independently with probability $n^{-1/k}$.

For designing a $(2k-1)$-approximate distance oracle, the existing algorithm [9] computes $Ball(u, \mathcal{R}_k^i, \mathcal{R}_k^{i+1})$ for each $u \in V, 1 \leq i < k$. Asymptotically the time required in their computation is

$$\sum_{u \in V} |Ball(u, \mathcal{R}_k^i, \mathcal{R}_k^{i+1})| \deg(u) = n^{1/k} \sum_{u \in V} \deg(u)$$
$$\{\text{by lemma 3.1}\}$$

We show that the running time of the algorithm can be improved for unweighted graphs by using a slightly different hierarchy of vertex-sets $\{S_k^i | 1 \leq i \leq k\}$ as described below. The new hierarchy of vertex-sets ensures that a vertex with degree $n^{i/k} \ln n$ or more has a neighbor from set $\mathcal{S}_k^{i+1}$, i.e., the set S_k^{i+1} is a dominating set for the set of vertices with degree $n^{i/k} \ln n$ or more. As a result, $Ball(u, S_k^i, S_k^{i+1})$ would be just the singleton set $\{u\}$, and so for each vertex $u \in V$ with degree $n^{i/k} \ln n$ or more, the contribution in above equation would be reduced to $\deg(u)$ from $n^{1/k} \deg(u)$. Hence, the overall time required for computing $Ball(u, S_k^i, S_k^{i+1})$ for all $u \in V$ would become

$$\begin{aligned} & \sum_{\deg(u) > n^{i/k} \ln n} \deg(u) + n^{1/k} \sum_{\deg(u) < n^{i/k} \ln n} \deg(u) \\ < \; & m + \min(n^{(k+i+1)/k} \ln n, mn^{1/k}) \\ = \; & O(n^2 \ln n) \quad \forall i < k \end{aligned}$$

THEOREM 3.1. *Given an unweighted graph $G(V,E)$, a hierarchy of vertex-sets $\{S_k^i | i \leq k\}$ with $(V = S_k^1) \supset S_k^2 \supset \cdots \supset S_k^k$ can be formed in $O(m)$ time such that the following properties are true for all $i \leq k$:*
(i) $|S_k^i| = O(n^{\frac{k-i+1}{k}})$,
(ii) $Ball(u, V, S_k^{i+1}) = \{u\}$ if u has degree at-least $n^{i/k} \ln n$,
(iii) Expected size of $Ball(u, S_k^i, S_k^{i+1})$ is at-most $n^{1/k}$.

Proof. As a simple application of the greedy set-cover algorithm [10], it takes $O(m)$ time to find a dominating set $\mathcal{D} \subset V$ of $\frac{n}{r} \ln n$ size for the all the vertices with degree $\geq r$. Let D_k^i be a dominating set for all the vertices with degree $n^{i/k} \ln n$. It can be seen that $|D_k^i| = n^{1-i/k}$. We form a hierarchy of vertex-sets S_k^i as follows.

$$S_k^i = (\bigcup_{i-1 \leq j < k} D_k^j) \bigcup \mathcal{R}_k^i$$

It is easy to verify that $|S_k^i| = O(n^{\frac{k-i+1}{k}})$. Since $D_k^i \subset S_k^{i+1}$, therefore, $Ball(u, V, S_k^{i+1}) = \{u\}$ if u has degree at-least $n^{i/k} \ln n$. Also note that for any two subsets $A \subset B$, $Ball(u, X, A)$ is a subset of $Ball(u, X, B)$. Since $R_k^{i+1} \subset S_k^{i+1}$, therefore, the expected size of $Ball(u, S_k^i, S_k^{i+1})$ is at-most $n^{1/k}$.

For the family of vertex-sets $\mathcal{S}_k^i$ as given in Theorem 3.1, let $p^i(u), i > 1$ denote the vertex from S_k^i that is nearest to vertex u, and $p^1(u)$ denote the vertex u itself.

3.3 Preserving the distance from u to all the vertices of $Ball(u, V, S_k^k)$ in the parameterized $(2,1)$-spanner $G_{S_k^k}$ For a given unweighted graph $G(V,E)$, consider the parameterized $(2,1)$-spanner $G_{S_k^k}$ built using the set S_k^k as the parameter. Let $u = v_0, v_1, v_2, \cdots, v_{j-1}, v_j = v$ be the shortest path between u and $v \in Ball(u, V, S_k^k)$ in the graph $G(V,E)$. Note that none of the vertices $v_i, i < j$ have any neighbor from the set S_k^{k-1} (otherwise $v \notin Ball(u, V, S_k^k)$,

a contradiction !). Since the parameterized spanner would keep all those edges whose one of the end-point has no neighbor in the parameter vertex-set S_k^k, so all the edges incident on the vertices $v_i, i < j$ are present in the parameterized $(2,1)$-spanner $G_{S_k^k}$. Hence the shortest path from u to each $v \in Ball(u, V, S_k^k)$ is present in the $(2,1)$-spanner $G_{S_k^k}$ too. Also note that all the edges incident on each $x \in S_k^k$ are also kept in the parameterized spanner. Therefore, the following equality holds.

$$(3.1)\ \delta^*(u,v) = \delta(u,v) \quad \forall v \in Ball(u, V, S_k^k) \bigcup \{p^k(u)\}$$

4 $(2k-1)$-approximate distance oracle for unweighted graph

In this section we shall give the construction of a $(2k-1)$-approximate distance oracle which is also based on the idea $\mathcal{I}$, and can be viewed as a generalization of 3-approximate distance oracle.

The $(2k-1)$-approximate distance oracle is obtained as follows.

1. Let $S_k^1 \supset S_k^2 \supset \cdots S_k^k$ be the hierarchy of subsets of vertices as defined in Theorem 3.1.
2. For each vertex $u \in V$, store the distance from u to all the vertices of $\mathcal{S}_k^k$ in a hash table, denoted by $Ball^k(u)$ henceforth.
3. For each $u \in V$ and each $i < k$, store the vertices of $Ball(u, S_k^i, S_k^{i+1})$ along with their distance from u in a hash table.

   For sake of conciseness and without causing any ambiguity in notations, henceforth we shall use $Ball^i(u)$ to denote $Ball(u, S_k^i, S_k^{i+1})$ or the corresponding hash-table storing $Ball(u, S_k^i, S_k^{i+1})$ for $i < k$.

The collection of the hash-tables $Ball^i(u) : u \in V, i \leq k$ constitute the data-structure that will facilitate answering of any approximate distance query in constant time. Note that Instead of using typical hash table that achieves expected $O(1)$ query time, we implement each $Ball^i(u)$ as a 2-level hash table given by Fredman, Komlos and Szemeredi [7] of optimal size, so that it takes $O(1)$ worst case time to determine whether a vertex $v \in V$ belongs to the $Ball^i(u)$ or not, and report the distance $\delta(u,v)$ in case v belongs to $Ball^i(u)$.

To provide a better insight into the data-structure, Figure 4 depicts the set of vertices constituting $\{Ball^i(u) | i \leq k\}$.

To ensure the desired stretch-bound in the distance reported by $(2k-1)$-approximate distance oracle, Theorem 4.2 (main theorem of this paper) would play a central role.

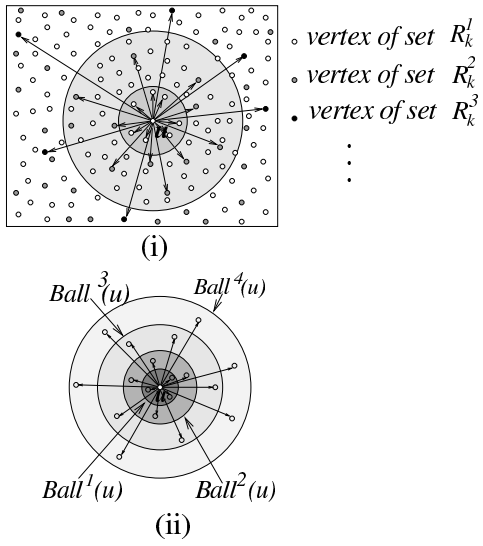

Figure 4: (i) Close description of $Ball^i(u), i < k$, (ii) hierarchy of balls around u

4.1 Reporting distance with stretch at-most $(2k-1)$

Given any two vertices $u, v \in V$ whose intermediate distance has to be determined approximately, we shall now present the procedure to find approximate distance between two vertices using the k-level data-structure described above.

Recall that $p^1(u) = u$ and $p^i(u)$ for $i > 1$ is the vertex from the set $\mathcal{S}_k^i$ that is nearest to u. Since $p^i(u) \in Ball^i(u)$ for each $u \in V$, so distance from each u to $p^i(u)$ is known for each $i \leq k$.

The query answering process performs at-most k search steps. In the first step, we search $Ball^1(u)$ for the vertex $p^1(v)$. If $p^1(v)$ is not present in $Ball^1(u)$, we move to the next level and in the second step we search $Ball^2(v)$ for vertex $p^2(u)$. We proceed in this way querying balls of u and v alternatively : In ith step, we search $Ball^i(x)$ for $p^i(y)$, where $(x = u, y = v)$ if i is odd, and $(x = v, y = u)$ otherwise. The search ends at ith step if $p^i(y)$ belongs to $Ball^i(x)$, and then we report $\delta(x, p^i(y)) + \delta(y, p^i(y)$ as an approximate distance between u and v.

Distance_Report(u,v)

$l \leftarrow 1$, $x \leftarrow u$,$y \leftarrow v$
While $(p^l(y) \notin Ball^l(x))$ **do**
    swap(x,y), $l \leftarrow l+1$
If $l < k$, return $\delta(y, p^l(y)) + \delta(x, p^l(y))$
Else return **min** of $(\delta^*(y, p^k(y)) + \delta^*(x, p^k(y)))$ and $(\delta^*(x, p^k(x)) + \delta^*(y, p^k(x)))$

Note that $p^k(y) \in \mathcal{S}_k^k$, and we store the distance from x to all the vertices of set $\mathcal{S}_k^k$ in $Ball^k(x)$. There-

fore, the 'while loop' of the distance reporting algorithm will execute at-most $k-1$ iterations, spending $O(1)$ time querying a hash table in each iteration.

In order to ensure that the above algorithm reports $(2k-1)$-approximate distance between $u, v \in V$, we first show that the following assertion holds :

$\mathcal{A}_i$: At the end of ith iteration of the *'while loop'*, $\delta(y, p^{i+1}(y)) \leq i\delta(u,v)$.

The assertion $\mathcal{A}_i$ can be proved using induction on i as follows. First note that the variables x and y take the values u and v alternatively during the *'while-loop'*. So $\delta(x,y) = \delta(u,v)$ always.

For the base case ($i = 0$), $p^1(y)$ is same as y, and y is v. So $\delta(y, p^1(y)) = 0$. Hence $\mathcal{A}_0$ is true. For the rest of the inductive proof, it suffices to show that if $\mathcal{A}_j$ is true, then after $(j+1)$th iteration $\mathcal{A}_{j+1}$ is also true. The proof is as follows.

We consider the case of 'even j', the arguments for the case of 'odd j' are similar. For even j, at the end of jth iteration, $\{x = u, y = v\}$, Thus $\mathcal{A}_j$ implies that at the end of jth iteration $\delta(v, p^{j+1}(v)) \leq j\delta(u,v)$. Consider the $(j+1)$th iteration. For the execution of $(j+1)$th iteration, the condition in the *'while-loop'* must have been true. Thus $p^{j+1}(v)$ does not belong to $Ball(u, \mathcal{R}_k^{j+1}, \mathcal{R}_k^{j+2})$. Hence by Definition of $Ball(\cdot)$, the vertex $p^{j+2}(u)$ must be lying closer to u than the vertex $p^{j+1}(v)$. So at the end of $(j+1)$th iteration, $\delta(y, p^{j+2}(y))$ can be bounded as follows

$$\begin{aligned}
\delta(y, p^{j+2}(y)) &= \delta(u, p^{j+2}(u)) \\
&\leq \delta(u, p^{j+1}(v)) \\
&\leq \delta(u,v) + \delta(v, p^{j+1}(v)) \\
&\quad \{\text{using triangle inequality}\} \\
&\leq \delta(u,v) + j\delta(u,v) \ \ \{\text{using } \mathcal{A}_j\ \} \\
&= (j+1)\delta(u,v)
\end{aligned}$$

Thus the assertion $\mathcal{A}_{j+1}$ holds.

We employ Theorem 4.2 and the observations of the parameterized $(2,1)$-spanner from previous section to prove the following theorem.

THEOREM 4.1. *The algorithm* Distance_Report(u,v) *reports* $(2k-1)$*-approximate distance between* u *and* v

Proof. Note that that the algorithm *Distance-Report(u,v)* would execute $(l-1)$ iterations of the *'while loop'*. If $l < k$, the distance reported is $\delta(y, p^l(y)) + \delta(x, p^l(y))$, which by triangle inequality is no more than $2\delta(y, p^l(y)) + \delta(x,y)$. Note that $\delta(x,y) = \delta(u,v)$, and $\delta(y, p^l(y)) \leq (l-1)\delta(u,v)$ as follows from assertion $\mathcal{A}_l$. Therefore, the distance reported is no more than $(2l-1)\delta(u,v) < (2k-1)\delta(u,v)$. If $l = k$, there are two cases.

Case 1 ($y \in Ball^k(x)$) : In this case, we bound $\delta^*(y, p^k(y)) + \delta^*(x, p^k(y))$ as follows.

$$\begin{aligned}
& \delta^*(y, p^k(y)) + \delta^*(x, p^k(y)) \\
= \ & \delta(y, p^k(y)) + \delta(x, p^k(y)) \ \{\text{using Equation 3.1}\} \\
\leq \ & 2\delta(y, p^k(y)) + \delta(x,y) \ \{\text{using triangle inequality}\} \\
\leq \ & 2(k-1)\delta(u,v) + \delta(u,v) \{\text{using assertion } \mathcal{A}_{k-1}\} \\
= \ & (2k-1)\delta(u,v)
\end{aligned}$$

Case 2 ($y \notin Ball^k(x)$) : Theorem 4.2 ensures that the distance reported is no more than $3\delta(x,y) = 3\delta(u,v) \leq (2k-1)\delta(u,v), \forall k > 1$.

The size of the set S_k^k is $O(n^{1/k})$, and the expected size of each $Ball(u, S_k^i, S_k^{i+1})$ is $n^{1/k}$ using Theorem 3.1 and Lemma 3.1. So the expected size of the distance oracle is $O(n^{1/k} \cdot n + (k-1) \cdot n \cdot n^{1/k}) = O(kn^{1+1/k})$.

4.2 Main Theorem

THEOREM 4.2. *For the* $(2,1)$*-spanner* $G_{S_k^k}$ *of unweighted graph* $G(V,E)$*, if* $v \notin Ball^k(u)$*, then either* $(\delta^*(u, p^k(u)) + \delta^*(v, p^k(u)))$ *or* $(\delta^*(v, p^k(v)) + \delta^*(u, p^k(v)))$ *is bounded by* $3\delta(u,v)$.

Proof. If either u is adjacent to $p^k(u)$ or v is adjacent to $p^k(v)$, we can prove the theorem as follows. Let v be adjacent to $p^k(v)$, and $v = v_0, v_1, v_2 \cdots, v_{j+1} = u$ be the shortest path of length $(j+1)$ between v and u in the original graph. It follows from Lemma 2.2 that there is a path from $p^k(v)$ to v_1 consisting of at-most two edges in the spanner. By definition of $(2,1)$-spanner, there is a path from v_1 to v_{j+1} in the spanner consisting of at-most $2j + [j > 0]$ edges. (Note that $[j > 0]$ is one if $j > 0$, and zero otherwise). Hence distance $\delta^*(v, p^k(v))$ between v and $p^k(v)$ in the spanner is no more than $2(j+1) + [j > 0]$. Also the edge $e(v, p^k(v))$ is present in the spanner. Hence, $\delta^*(v, p^k(v)) + \delta^*(u, p^k(v)) \leq 1 + 2(j+1) + [j > 0] \leq 3(j+1) = 3\delta(v,u)$.

If neither u is adjacent to $p^k(u)$ nor v is adjacent to $p^k(v)$, let us divide the shortest path P_{uv} between u and v in the graph $G(V,E)$ into three parts : the part P_{uw} lying in $Ball^k(u)$, the part $P_{w'v}$ lying inside $Ball^k(v)$, and the remaining part $P_{ww'}$ of the path that does not lie in either of the two balls. Let the subpaths $P_{uw}, P_{ww'}, P_{w'v}$ have lengths α, β, γ respectively. Note that both α and γ are greater or equal to 1 (see Figure 5). Now we bound the distance $\delta^*(u, p^k(u)) + \delta^*(v, p^k(u))$ as follows.

$$\begin{aligned}
& \delta^*(u, p^k(u)) + \delta^*(v, p^k(u)) \\
\leq \ & \delta^*(u, p^k(u)) + (\delta^*(v,u) + \delta^*(u, p^k(u))) \\
& \quad \{\text{by triangle inequality}\}
\end{aligned}$$

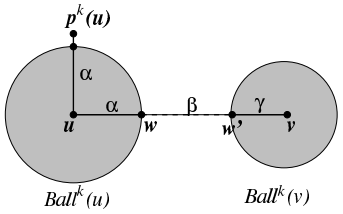

Figure 5: dividing the path $u - v$ into three parts.

$$
\begin{aligned}
&= 2\delta^*(u,p^k(u)) + (\delta^*(u,w) + \delta^*(w,w') + \delta^*(w',v)) \\
&= 2\delta(u,p^k) + \delta(u,w) + \delta^*(w,w') + \delta(w',v) \\
&\qquad \{\text{by Equation 3.1 }\} \\
&= 2(\delta(u,w)+1) + \delta(u,w) + \delta^*(w,w') + \delta(w',v) \\
&= 2(\alpha+1) + \alpha + (2\beta + [\beta > 0]) + \gamma \\
&\qquad \{\text{by defn. of } (2,1)\text{-spanner }\} \\
&= 3\alpha + 2\beta + \gamma + (2 + [\beta > 0]) \quad \leq \quad 3(\alpha+\beta+\gamma) \\
&= 3\delta(u,v)
\end{aligned}
$$

5 Computing $(2k-1)$-approximate distance oracle in $\tilde{O}(n^2)$ time

It follows from the description of the data-structure associated with approximate distance oracle that after forming the hierarchy of sets $\mathcal{S}_k^i$, that takes $O(m)$ time, all that is required is the computation of $p^i(u)$ and $Ball^i(u)$ for each u and $i \leq k$.

5.1 Computing $p^i(u)$ for each $u \in V$ Recall from definition itself that $p^i(u)$ is the vertex of the set S_k^i that is nearest to u. Hence, computing $p^i(u)$ for each $u \in V$ requires solving the following problem with $X = S_k^i, Y = V - X$.

Problem : *Given $X, Y \subset V$ in a graph $G = (V,E)$, with $X \cap Y = \emptyset$, compute the nearest vertex of set X for each vertex $y \in Y$.*

The above problem can be solved by running a single source shortest path algorithm on a modified graph as follows. Modify the original graph G by adding a dummy vertex s to the set V, and joining it to each vertex of the set X by an edge of zero weight. Let G' be the modified graph. It can be seen that the distance from a vertex y to its nearest vertex from set X is $\delta(s,y) - 1$. Moreover while Running the shortest path algorithm from the vertex s, if $e(s,x), x \in X$ is the edge leading to the shortest path from s to y, then x is the vertex from the set X that lies nearest to y. For an unweighted graph, single source shortest paths can be computed in $O(m)$ time by a variant of breadth-first search procedure starting from the source. we can thus state the following lemma.

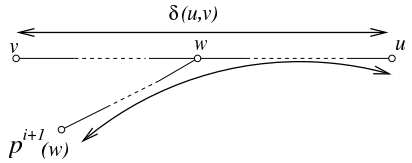

Figure 6: if w does not lie in $C(v, \mathcal{R}_k^{i+1})$, then $p^{i+1}(w)$ would lie closer to u than v.

LEMMA 5.1. *Given $X, Y \subset V$ in an unweighted graph $G(V,E)$, with $X \cap Y = \emptyset$, it takes $O(m)$ time to compute the nearest vertex of set X for each vertex $y \in Y$.*

5.2 Computing $Ball^i(u)$ efficiently In order to compute $Ball^i(u) = Ball(u, S_k^i, S_k^{i+1})$ efficiently for all the vertices, we first compute sets $\{C(v, S_k^{i+1}) | v \in S_k^i\}$ which are defined as follows :

Definition : *For a graph $G(V,E)$, and a set $X \subset V$, the set $C(v,X)$ consists of all those vertices of the graph for whom v lies closer than any vertex of set X.*

It follows from the definition above that $u \in C(v, S_k^{i+1})$ if and only if $v \in Ball^i(u)$. We shall make use of the following equality which is based on this observation.

$$(5.2) \quad \sum_{v \in S_k^i} \sum_{u \in C(v, S_k^{i+1})} \deg(u) = \sum_{u \in V} \sum_{v \in Ball^i(u)} \deg(u)$$

Given $\{C(v, S_k^{i+1}) | v \in S_k^i\}$, we can compute $\{Ball^i(u) | u \in V\}$ as follows.

For each $v \in S_k^i$ **do**
  For each $u \in C(v, S_k^{i+1})$ **do**
    $Ball^i(u) \longleftarrow Ball^i(u) \cup \{v\}$

Hence we can state the following Lemma.

LEMMA 5.2. *Given the family of sets $\{C(v, S_k^{i+1}) | v \in S_k^i\}$, we can compute $\{\{Ball^i(u) | u \in V)\}$ in time of the order of $\sum_{u \in V} |Ball^i(u)|$.*

The following property of the set $C(v, S_k^{i+1})$ will be used in its efficient computation.

LEMMA 5.3. *If $u \in C(v, S_k^{i+1})$, then all the vertices on the shortest path from v to u also belong to the set $C(v, S_k^{i+1})$.*

Proof. We give a proof by contradiction. Given that $u \in C(v, S_k^{i+1})$, let w be any vertex on the shortest path from v to u. If $w \notin C(v, S_k^{i+1})$, the vertex v doesn't lie closer to w than the vertex $p^{i+1}(w)$. See Figure 6. In other words $\delta(w,v) \geq \delta(w, p^{i+1}(w))$. Hence

$$
\begin{aligned}
\delta(u,v) &= \delta(u,w) + \delta(w,v) \\
&\geq \delta(u,w) + \delta(w, p^{i+1}(w)) \geq \delta(u, p^{i+1}(w))
\end{aligned}
$$

Thus v does not lie closer to u than $p^{i+1}(w)$ which is a vertex of set S_k^{i+1}. Hence by definition, $u \notin C(v, S_k^{i+1})$, thus a contradiction.

In the shortest path tree rooted at v, it follows from Lemma 5.3 that the entire cluster $C(v, S_k^{i+1})$ would appear as a sub-tree rooted at v. Also each vertex x belonging to this subtree (cluster) satisfies $\delta(v,x) < \delta(x, p^{i+1}(x))$. Based on these two observations, here follows an efficient algorithm that computes the set $C(v, S_k^{i+1})$. The algorithm performs a *restricted* shortest path computation from the vertex v, wherein we don't proceed along any vertex that does not belong to the set $C(v, S_k^{i+1})$.

Restricted shortest path algorithm : Note that the shortest path algorithm (Dijkstra's algorithm) starts with singleton tree $\{v\}$ and performs $n-1$ steps to grow the complete shortest path tree. Each vertex $x \in V \backslash \{v\}$ is assigned a label $L(x)$, which is infinity in the beginning, but eventually becomes the distance from v to x. Let V_i denotes the set of i nearest vertices from v. The algorithm maintains the following invariant at the end of lth step :
$\mathcal{I}(l)$: For all the vertices of the set V_l, the label $L(x) = \delta(v,x)$, and for every other vertex $y \in V \backslash V_l$, the label $L(y)$ is equal to the length of the shortest path from v to y that passes through vertices of V_l only.

During the $(j+1)$th step, we select the vertex, say w from set $V - V_j$ with least value of $L(\cdot)$. Since all edge-weights are positive, it follows from the invariant $\mathcal{I}(j)$ that $L(w) = \delta(w,v)$. Thus we add w to set V_j to get the set V_{j+1}. Now in order to satisfy the invariant $\mathcal{I}(j+1)$, we relax each edges $e(w,y)$ incident from w to a vertex $y \in V - V_{j+1}$ as follows : $L(y) \leftarrow \min\{L(y), L(w) + weight(w,y)\}$. It is easy to observe that this ensures the validity of the invariant $\mathcal{I}(j+1)$.

In the restricted shortest path algorithm, we will put the following restriction on relaxation of an edge $e(w,y)$: we relax the edge $e(w,y)$ only if $L(w) + weight(w,y)$ is less than $\delta(y, p^i(y))$. This will ensure that a vertex $y \notin C(v, \mathcal{R}_k^{i+1})$ will never be visited during the algorithm. The fact that the vertices of the cluster $C(v, \mathcal{R}_k^{i+1})$ form a sub-tree of the shortest path tree rooted at v ensures that the above restricted shortest path algorithm indeed finds all the vertices that form the cluster $C(v, \mathcal{R}_k^{i+1})$ along with their distance from v.

The running time of the above algorithm is dominated by the total number of edges relaxed, and each edge relaxation requires $O(1)$ time for unweighted graphs. (For unweighted graphs, Dijkstra's algorithm is just a variant of breadth-first search). So, the above algorithm runs in time of the order of $\deg(v) + \sum_{u \in C(v, \mathcal{R}_k^{i+1})} \deg(u)$.

LEMMA 5.4. *The set $C(v, S_k^{i+1})$ can be computed in $O(\deg(v) + \sum_{u \in C(v, \mathcal{R}_k^{i+1})} \deg(u))$ time.*

Hence the total time T_i required for computing $\{C(v, S_k^{i+1}) | v \in S_k^i\}$ for $i < k$ is

$$\begin{aligned}
T_i &= \sum_{v \in S_k^i} \left(\deg(v) + \sum_{u \in C(v, \mathcal{R}_k^{i+1})} \deg(u) \right) \\
&= \sum_{v \in S_k^i} \deg(v) + \sum_{v \in S_k^i} \sum_{u \in C(v, \mathcal{R}_k^{i+1})} \deg(u) \\
&\leq m + \sum_{u \in V} \sum_{v \in Ball^i(u)} \deg(u) \{\text{using Equation 5.2 }\} \\
&= m + \sum_{u \in V} \left(|Ball^i(u)| \cdot \deg(u) \right)
\end{aligned}$$

Now using Theorem 3.1, it follows that $Ball^i(u) = \{u\}$ if $\deg(u)$ is more than $n^{i/k} \ln n$. So contribution of all such vertices in the above expression is no more than m. Hence the inequality can be written as

$$\begin{aligned}
T_i &\leq 2m + \sum_{u \in V} \left(|Ball^i(u)| \cdot \min(n^{i/k} \ln n, \deg(u)) \right) \\
&\leq 2m + \sum_{u \in V} \left(n^{1/k} \cdot \min(n^{i/k} \ln n, \deg(u)) \right) \\
&\qquad \{\text{using Lemma 3.1 }\} \\
&\leq 2m + n^{1/k} \min \left(\sum_{u \in V} n^{i/k} \ln n \ , \ \sum_{u \in V} \deg(u) \right) \\
&= 2m + \min \left(n^{\frac{k+i+1}{k}} \ln n, n^{1/k} m \right) \\
&= O(\min(n^{\frac{k+i+1}{k}} \ln n, mn^{1/k})) \ \ \{\text{for all } 1 \leq i < k \ \}
\end{aligned}$$

Combining the upper bound of T_i mentioned above and Lemma 5.2, we can state the following theorem.

THEOREM 5.1. *Given an unweighted graph $G(V,E)$, and the family of sets $\{S_k^j | j \leq k\}$, it takes $O(\min(n^{\frac{k+i+1}{k}} \ln n, mn^{1/k}))$ time to compute $\{Ball^i(u) | u \in V\}$ for $i < k$.*

The total time required to compute the family of sets $\{Ball^i(u) | u \in V, i < k\}$ is no more than $O(\min(n^2 \ln n, kmn^{1/k})$ as follows from Theorem 5.1. Computing distances from a vertex $x \in S_k^k$ to all the vertices in the $(2,1)$-spanner $G_{S_k^k}$ requires merely a BFS traversal of the spanner with x as root, which can be performed in time of the order of the number of edges in the spanner. Since the spanner $G_{S_k^k}$ has $O(\min(m, n^{2-1/k} \ln n))$ edges and there are $O(n^{1/k})$ vertices in the set S_k^k, the total time to compute $\{\delta^*(x,v) | x \in S_k^k, v \in V\}$ in $(2,1)$-spanner is

$O(\min(mn^{1/k}, n^2 \ln n)$. Hence the total time required to compute $(2k-1)$-approximate distance oracle is $O(\min(n^2 \ln n, kmn^{1/k}))$. We repeat the algorithm if the size of the approximate distance oracle exceeds $O(kn^{1+1/k})$. The expected number of repetitions is $O(1)$. Combining these facts with Theorem 4.1, we can state the following theorem.

THEOREM 5.2. *An unweighted graph $G(V,E)$ can be processed in $O(\min(n^2 \ln n, kmn^{1/k})$ time to compute a data-structure of size $O(kn^{1+1/k})$ that can report $(2k-1)$-approximate distance between any two vertices in $O(k)$ time for any integer $k > 1$.*

6 An optimal size $(2,1)$-spanner

Consider the $(2,1)$-spanner $G(V, E'_R \cup E^*_R)$ from Lemma 2.4. Let the sample set R be formed by picking each vertex independently with some fixed probability p. The expected size of the set R will be np. Therefore it follows from Lemma 2.1 that the expected size of the set E^*_R will be n^2p. The following lemma bounds the expected size of the set E'_R.

LEMMA 6.1. *Given a graph $G(V,E)$, and a sample set R formed by picking each vertex independently with probability p, the expected number of edges E'_R is $O(n/p)$.*

Proof. Note that the edges adjacent to a vertex $u \in V - R$ appear in the set E'_R if neither u nor any of its neighbor is adjacent to any vertex of the sample R. Since each vertex is selected in the sample R independently with probability p, so the expected number of edges contributed by vertex v to E'_R is $\deg(v) \cdot (1-p)^{\deg(v)+1}$. Using elementary calculus and the fact that $p \ll 1$, it can be verified that vertex v contributes maximum number of edges to E'_R if $\deg(v)$ is close to $\theta(1/p)$, and the maximum number is $\theta(1/p)$. Hence, using linearity of expectation, the expected number of edges in the set E'_R is bounded by $\theta(1/p)$.

Combining Lemmas 6.1 and 2.4, we can state the following theorem.

THEOREM 6.1. *For an unweighted graph $G(V,E)$, and a sample $R \subset V$ formed by selecting each vertex independently with probability p, the subgraph $G(V, E'_R \cup E^*_R)$ is a $(2,1)$-spanner with expected size $O(n/p + n^2p)$*

As mentioned earlier, the running time of our algorithm is $O(m)$. We would repeat the algorithm if the size of the spanner is more than $O(n/p + n^2p)$. The expected number of iterations is $O(1)$. Thus a $(2,1)$ spanner of size $O(n/p + n^2p)$ can be computed in $O(m)$ expected time. In order to minimize the size of the spanner, we balance the two terms n/p and n^2p. Choosing $p = n^{-1/2}$, we get a bound of $O(n^{3/2})$ on the size of $(2,1)$-spanner. We can thus state the following Theorem.

THEOREM 6.2. *Given an unweighted undirected graph $G(V,E)$, a $(2,1)$-spanner of size $O(n^{3/2})$ can be computed in expected $O(m)$ time.*

Thorup and Zwick [9] show that $\theta(n^{3/2})$ bound on the size of 3-spanner is indeed optimal. Since a $(2,1)$-spanner is also a 3-spanner, therefore, $\theta(n^{3/2})$ bound is optimal for $(2,1)$-spanner too. There does not exist any previous algorithm for computing $(2,1)$-spanner. However there exist linear time algorithms [1, 8] for computing $(2k-1)$-spanner for $k \geq 3$. Elkin and Peleg [6] presented algorithm for computing $(1+\epsilon, \beta)$ spanner, but with $O(n^{2+\mu})$ running time.

References

[1] S. Baswana and S. Sen. A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size in weighted graphs. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 384–396, 2003.

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

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

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

[5] M. Elkin. Computing almost shortest paths. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 53–62, 2001.

[6] M. Elkin and D. Peleg. $(1+\epsilon, \beta)$-spanner construction for general graphs. In *Proceedings of 33rd ACM Symposium on Theory of Computing (STOC)*, pages 173–182, 2001.

[7] 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.

[8] S. Halperin and U. Zwick. Unpublished result. 1996.

[9] M. Thorup and U. Zwick. Approximate distance oracle. In *Proceedings of 33rd ACM Symposium on Theory of Computing (STOC)*, pages 183–192, 2001.

[10] V. V. Vazirani. In *Approximation Algorithms*. Springer Verlag, 2001.

[11] U. Zwick. All-pairs shortest paths in weighted directed graphs - exact and almost exact algorithms. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 310–319, 1998.