Approximate Distance Oracles for Unweighted Graphs in Expected $O(n^2)$ time *

Surender Baswana
Max-Planck Institut fuer Informatik,
Stuhlsatzenhausweg 85,
66123 Saarbruecken, Germany.
Email : sbaswana@mpi-sb.mpg.de

Sandeep Sen
Department of Comp. Sc. and Engg.,
Indian Institute of Technology Delhi,
Hauz Khas, New Delhi-110016, India.
E-mail : ssen@cse.iitd.ernet.in

Abstract

Let $G = (V, E)$ be an undirected graph on n vertices, and let $\delta(u, v)$ denote the distance in G between two vertices u and v. Thorup and Zwick showed that for any +ve integer t, the graph G can be preprocessed to build a data-structure that can efficiently report t-approximate distance between any pair of vertices. That is, for any $u, v \in V$, the distance reported $\hat{\delta}(u, v)$ satisfies

$$\delta(u, v) \leq \hat{\delta}(u, v) \leq t\delta(u, v)$$

The remarkable feature of this data-structure is that, for $t > 2$, it occupies sub-quadratic space, i.e., it does not store all-pairs distances information explicitly, and still it can answer any t-approximate distance query in constant time. They named the data-structure "oracle" because of this feature. Furthermore the tradeoff between stretch t and the size of the data-structure is essentially optimal.

In this paper we show that we can actually construct approximate distance oracles in expected $O(n^2)$ time if the graph is unweighted. One of the new ideas used in the improved algorithm also leads to the first expected linear time algorithm for computing an optimal size $(2, 1)$-spanner of an unweighted graph. A $(2, 1)$ spanner of a graph $G = (V, E)$ is a subgraph $G = (V, E_S), E_S \subset E$, such that for any two vertices u and v in the graph, their distance in the subgraph is at most $2\delta(u, v) + 1$.

1 Introduction

The all-pairs shortest paths problem is one of the most fundamental algorithmic graph problem. Every computer scientist is aware of this classical problem right from the days he did his first course on algorithms. This problem is commonly phrased as follows : *Given a graph G on n vertices and m edges, compute shortest-paths/distances between each pair of vertices.*

In many applications the aim is not to compute *all* distances, but to have a mechanism (data-structure) through which we can extract distance/shortest-path for any pair of vertices efficiently. Therefore, the following is a useful alternate formulation of the APSP problem.

Preprocess a given graph efficiently to build a data-structure that can answer a shortest-path query or a distance query for any pair of vertices.

*Preliminary version of this work appeared in 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 271-280, 2004.

Throughout this paper, we stick to the above formulation of APSP. The objective is to construct a data-structure for this problem such that it is efficient both in terms of the space and the preprocessing time. There is a lower bound of $\Omega(n^2)$ on the space requirement of any data-structure for APSP problem, and space requirement of all the existing algorithms for APSP match this bound. However, there is a huge gap in the preprocessing time. In its most generic version, that is, for directed graph with real edge-weights, the best known algorithm [13] for APSP 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. In fact the best known upper bound on the worst case time complexity of this problem is $O(n^3/\log n)$ due to Chan [6], which is marginally sub-cubic. Surprisingly, despite the fundamental importance of the problem, there does not exist any algorithm at present that can solve APSP in truly sub-cubic time, i.e., $O(n^{3-\epsilon})$ for some $\epsilon > 0$. The existing lower bound on the time complexity of APSP is the trivial lower bound of $\Omega(n^2)$. This has motivated the researchers to explore ways to achieve (truly) sub-cubic preprocessing time and/or sub-quadratic space data-structures that can report *approximate* instead of exact shortest-paths/distances.

A data-structure is said to compute t-approximate distances for all-pairs of vertices, if for any pair of vertices $u, v \in V$, the distance $\hat{\delta}(u, v)$, that it would report, satisfies

$$\delta(u, v) \leq \hat{\delta}(u, v) \leq t\delta(u, v)$$

In the last ten years, many novel algorithms have been designed for all-pairs approximate shortest paths (APASP) problem that achieve sub-cubic running time and/or sub-quadratic space. The *approximate distance oracle* designed by Thorup and Zwick [15] is a milestone in this area. They show that any given weighted undirected graph can be preprocessed in sub-cubic time to build a data-structure of sub-quadratic size for answering a distance query with stretch 3 or more. Note that 3 is also the least stretch for which we can achieve sub-quadratic space for APASP (see [8]). There are two very impressive features of their data-structure. First, the trade off between stretch and the size of data-structure is essentially optimal and second, inspite of its sub-quadratic size their data-structure can answer any distance query in constant time, hence the name "oracle". In precise words, Thorup and Zwick achieve the following result.

Theorem 1.1 *[15] For any integer $k \geq 1$, and a given undirected weighted graph on n vertices and m edges, it takes expected $O(kmn^{1/k})$ time to build a data-structure of size $O(kn^{1+1/k})$ that can answer any $(2k-1)$-approximate distance query in $O(k)$ time.*

As mentioned in [15] the oracle model for shortest paths has been considered in the past also, at least implicitly, by Awerbuch *et al.* [2], Cohen [7] and by Dor *et al.* [9]. However, the approximate distance oracle of Thorup and Zwick significantly improves all these previous results on oracles for shortest paths. Having achieved essentially optimal query time as well as the space requirement for approximate distance oracles, the only aspect that can be potentially improved is the preprocessing time. Currently, the expected preprocessing time of the $(2k-1)$-approximate distance oracle is $O(mn^{1/k})$ which is certainly sub-cubic. Thorup and Zwick posed the following question : *Can a $(2k-1)$-approximate distance oracle be computed in $O(n^2)$ time?*

In this paper, we answer their question in affirmative for unweighted graphs. The following is the main result of our paper.

Theorem 1.2 *For any integer $k \geq 1$, an undirected unweighted graph on n vertices and m edges can be preprocessed in expected $O(\min(n^2,\ kmn^{1/k}))$ time to construct its $(2k-1)$-approximate distance oracle of size $O(kn^{1+1/k})$.*

1.1 Summary of related work on APASP

The algorithms for all-pairs approximate shortest paths fall under two categories, depending upon whether the error in the distance is additive or multiplicative. The algorithm that computes all-pairs t-approximate shortest paths, as defined earlier, is actually an algorithm that achieves a multiplicative error t. An algorithm is said to report distances with additive error k, if for any pair of vertices $u, v \in V$, the distance, that it reports, is at least $\delta(u,v)$ and at most $\delta(u,v) + k$. The first algorithm for reporting distances with additive error was given by Aingworth *et al.* [1]. Their algorithm reports distances with additive error 2. Dor *et al.* [9] improved and extended this algorithm for arbitrary additive error. Their algorithm requires $O(kn^{2-\frac{1}{k}} m^{\frac{1}{k}} \operatorname{polylog} n)$ time to report distances with additive error $2(k-1)$ for any pair of vertices. They also showed that all-pairs 3-approximate shortest paths can be computed in $O(n^2 \operatorname{polylog} n)$ time. Cohen and Zwick [8] later extended this result to weighted graphs. There are also some algorithms that achieve multiplicative as well as additive errors simultaneously. Elkin [10] presented the first such algorithm. For unweighted graphs, given arbitrarily small $\zeta, \epsilon, \rho > 0$, Elkin's algorithm requires $O(mn^{\rho} + n^{2+\zeta})$ time, and for any pair of vertices $u, v \in V$, reports distance $\hat{\delta}(u,v)$ satisfying the inequality :$\delta(u,v) \leq \hat{\delta}(u,v) \leq (1+\epsilon)\delta(u,v) + \beta$, where β is a function of ζ, ϵ, ρ. If the two vertices $u, v \in V$ are separated by sufficiently long distances in the graph, the stretch $\frac{\hat{\delta}(u,v)}{\delta(u,v)}$ ensured by Elkin's algorithm is quite close to $(1+\epsilon)$. But the stretch factor may be quite huge for short paths. This is because β depends on ζ as $(1/\zeta)^{\log 1/\zeta}$, depends inverse exponentially on ρ and inverse polynomially on ϵ. The output of all the algorithms mentioned above is an $n \times n$ matrix that stores pairwise approximate distance for all the vertices explicitly to answer each distance query in constant time. Recently Baswana *et al.* [3] presented a simple algorithm to compute *nearly* 2-approximate distances for unweighted graphs in expected $O(n^2 \operatorname{polylog} n)$ time. They essentially extend the idea of the 3-approximate distance oracle of Thorup and Zwick [15] to achieve stretch 2 at the expense of introducing a very small additive error (less than 3). All the algorithms for reporting approximate distances that we mentioned above, including the new results of this paper, can also report the corresponding approximate shortest path also, and the time for doing so is proportional to the number of edges in the approximate shortest-path.

1.2 Organization of the paper

Our starting point will be the 3-approximate distance oracle of Thorup and Zwick [15]. The following section explains notations, definitions, and lemmas, most of them adapted from [15]. In section 3, we provide a sketch of the existing 3-approximate distance oracle, and identify the most time consuming Steps in its preprocessing algorithm. Subsequently we explore ways to execute them in expected $O(n^2)$ time. We succeed in our goal by providing a tighter analysis of one of the Steps and by performing small but crucial changes in the remaining ones. An important tool used by our preprocessing algorithm is is a special kind of a $(2,1)$-spanner. For an unweighted graph $G = (V, E)$, a subgraph $G = (V, E_S)$, $E_S \subset E$ is said to be an (α, β)-spanner if for each pair of vertices $u, v \in V$, the distance between the two in the subgraph is at most $\alpha\delta(u,v) + \beta$. In section 4, we present a *parameterized* $(2,1)$-spanner and explain how its special features are helpful in improving the preprocessing time of the oracle without increasing the stretch. Finally in section 5, we present and analyze our new 3-approximate distance oracle. In a straightforward manner, the techniques used in our 3-approximate distance oracle lead to the expected $O(\min(n^2,\ mn^{1/k}))$ time preprocessing algorithm for $(2k-1)$-approximate distance oracle in section 6.

A second contribution of the paper, which is important in its own right, is the first expected linear time algorithm for computing a $(2,1)$-spanner of (optimal) size $O(n^{3/2})$. The previous linear time algorithms [4, 12] compute spanners of stretch three or more.

2 Preliminaries

For a given undirected unweighted graph $G = (V, E)$, we define the following notations.

- $\delta(u, v)$: the distance between vertex u and vertex v in the graph.
- $\delta(v, Y)$: $\min_{y \in Y} \delta(v, y)$
- $p(v, Y)$: the vertex from the set Y which is nearest to v, that is, at distance $\delta(v, Y)$ from v. In case there are multiple vertices from Y at distance $\delta(v, Y)$, we break the tie arbitrarily to ensure a unique $p(v, Y)$. Moreover, if $v \in Y$, we define $p(v, Y) = v$.

In this paper we deal with undirected unweighted graphs only. It is straightforward to observe that a single source shortest path tree rooted at a vertex $v \in V$ in an unweighted graph is the same as the breadth-first-search (BFS) tree rooted at v, which can be computed in just $O(m)$ time.

Given a set $Y \subset V$, it is also quite easy to compute $p(v, Y)$ 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. Thus the following lemma holds.

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

2.1 Ball around a vertex

An important construct of the approximate distance oracles 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, we define $Ball(v, X, Y)$ as a set in the following way*

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

In simple words, $Ball(v, X, Y)$ consists of all those vertices of the set X whose distance from v is less than the distance from $p(v, Y)$ to v. As a simple observation, it can be noted that $Ball(v, X, \emptyset)$ is the set X itself, whereas $Ball(v, X, X) = \emptyset$.

The following lemma from [15] suggests that if the set Y is formed by sampling vertices from the set X with suitable probability, the expected size of $Ball(u, X, Y)$ will be sublinear in terms of $|X|$. This observation plays the key role in achieving sub-quadratic bound on the size of the approximate distance oracles of [15]. These oracles basically store distances to a hierarchy of Balls around each vertex.

Lemma 2.2 *[15] 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 q. The expected size of $Ball(v, X, Y)$ is at most $1/q$.*

Proof: Consider the sequence $\langle x_1, x_2, \cdots \rangle$ of vertices of set X arranged in non-decreasing order of their distances from v. 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. Therefore, $x_i \in Ball(v, X, Y)$ with probability at most $(1 - q)^i$. Hence using linearity of expectation, the expected number of vertices in $Ball(v, X, Y)$ is at most $\sum_i (1 - q)^i < 1/q$. □

In order to answer an approximate distance query in constant time, $Ball(v, X, Y)$ can be kept in a hash table [11] so that one can determine in worst case constant time whether or not $x \in Ball(v, X, Y)$, and if so, report the distance $\delta(v, x)$. Once we have the set $Ball(v, X, Y)$, the preprocessing time as well as the space requirement of the corresponding hash-table would be of the order of $|Ball(v, X, Y)|$.

2.2 Computing Balls efficiently

Thorup and Zwick presented a very efficient way to compute Balls wherein for each vertex $x \in X$, we compute all those vertices $v \in V$ (and their distances from x) whose *Ball*(v, X, Y) contains x. The key observation is the following lemma.

Lemma 2.3 *[15] If a vertex $x \in X$ belongs to $Ball(v, X, Y)$, then for every vertex u lying on a shortest path between x and v, the vertex x will also belong to $Ball(u, X, Y)$.*

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

It follows from Lemma 2.3 that if we look at a shortest-path tree rooted at a vertex $x \in X$, the set of all those vertices $v \in V$ satisfying the condition "$x \in Ball(v, X, Y)$" appears as truncated sub-tree rooted at x. This implies that, for a vertex $x \in X$, computing the set of vertices $v \in V$ satisfying "$x \in Ball(v, X, Y)$", amounts to performing a restricted BFS traversal from x, wherein we have to confine the BFS traversal within this sub-tree only. The algorithm is described below.

Restrict_BFS(x, Y)

```
Initially all vertices are unvisited.
Visited(x)←true;  Enqueue(Q, x)
while Q not_empty
  v ← Dequeue(Q)
  For each neighbor w of v
  If not Visited(w) and (δ(x, v) + 1 < δ(w, p(w, Y)))
     Visited(w)←true
     δ(x, w) ← δ(x, v) + 1
     Enqueue(Q, w)
```

We can now state the following lemma whose proof is immediate.

Lemma 2.4 *The algorithm Restrict_BFS(x, Y) begins with exploring the adjacency list of x, and afterwards, it explores the adjacency list of only those vertices v that satisfy "$x \in Ball(v, X, Y)$".*

In order to compute $Ball(v, X, Y), \forall v \in V$, it suffices to perform *Restrict_BFS*(x, Y) on all $x \in X$. The running time of the algorithm is proportional to the number of edges traversed. To analyze this running time concisely, the following scheme is quite useful. For each edge that is traversed during *Restrict_BFS*(x, Y), the cost of the traversal of the edge is assigned to the endpoint that explores it. Using this scheme, it follows from Lemma 2.4 that a vertex $v \in V$ will be charged $deg(v)$ during *Restrict_BFS*(x, Y) if and only if $x \in Ball(v, X, Y)$. Furthermore, a cost of $deg(x)$ will be charged to x during *Restrict_BFS*(x, Y). These observations can be summarized in the following lemma.

Lemma 2.5 *[15] Given a graph $G = (V, E)$, and sets X, Y of vertices, we can compute $Ball(v, X, Y), \forall v \in V$ by running Restrict_BFS(x, Y) on each $x \in X$. During this computation, each vertex $v \in V$ will be charged $deg(v)$ cost once by every vertex of $Ball(v, X, Y)$. There will be an additional computation cost of $\sum_{x \in X} deg(x)$, which is at most $O(m)$.*

3 The 3-approximate distance oracle of Thorup and Zwick

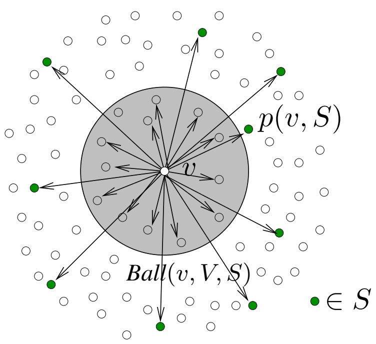


Figure 1: For v, we store distance to the vertices pointed by arrows.

Let $G = (V, E)$ be an undirected unweighted graph. The 3-approximate distance oracle of Thorup and Zwick can be constructed as follows :

Let $S \subset V$ be a set formed by selecting each vertex independently with probability $n^{-1/2}$. For each vertex $v \in V$, compute $p(v,S)$ and the distances to vertices of set $Ball(v,V,S)$. In addition, compute distances $\delta(v,x)$ for every $x \in S$ (see Figure 1).

The final data-structure would keep, for each vertex $v \in V$, the nearest vertex $p(v,S)$ and two hash tables : one hash table storing the distances to vertices of $Ball(u,V,S)$, and another hash table storing the distances to all vertices of the set S. Using these hash-tables, any distance query can be answered as follows.

Answering a distance query :

Let $u, v \in V$ be any two vertices whose approximate distance is to be computed. First it is determined whether or not $u \in Ball(v,V,S)$, and if so, the exact distance $\delta(u,v)$ is reported. Note that $u \notin Ball(v,V,S)$ would imply $\delta(v,p(v,S)) \leq \delta(v,u)$. In this case, report the distance $\delta(v,p(v,S)) + \delta(u,p(v,S))$, which is at least $\delta(u,v)$ (using the triangle inequality) and upper-bounded by $3\delta(u,v)$ as shown below.

$$\begin{aligned}
\delta(v,p(v,S)) + \delta(u,p(v,S)) &\leq \delta(v,p(v,S)) + (\delta(u,v) + \delta(v,p(v,S))) \quad \text{(using triangle inequality)} \\
&= 2\delta(v,p(v,S)) + \delta(u,v) \\
&\leq 2\delta(u,v) + \delta(u,v) = 3\delta(u,v) \qquad \{\text{since} \quad \delta(v,p(v,S)) \leq \delta(v,u)\}
\end{aligned}$$

It follows from Lemma 2.2 that the expected size of $Ball(v,V,S)$ would be $\sqrt{n}$, and hence the total expected size of the data-structure would be $O(n^{3/2})$. Thorup and Zwick showed that it takes $O(m\sqrt{n})$ preprocessing time to build 3-approximate distance oracle for an undirected graph.

3.1 Ideas for Improving the Preprocessing time for Computing the Oracle

Let us explore ways to improve the preprocessing time for 3-approximate distance oracle. There are two main computational tasks involved in building the 3-approximate distance oracle. The first task is the computation of *Balls*() and the second task is the computation of distances from sampled vertices to all the vertices in the graph. Note that there is an additional task of computing $p(v,S)$ for every $v \in V$. But that is quite easily computable in overall $O(m)$ time as mentioned in Lemma 2.1.

In order to compute *Balls*(), Thorup and Zwick build the *Restricted_BFS* trees from the vertices of the set S. They show, using Lemmas 2.2 and 2.5 in a fairly straightforward way, that the expected time required is $O(m\sqrt{n})$. For dense graphs, the expected time may be as large as $\theta(n^{2.5})$. We provide a tighter analysis of the same algorithm and show that the expected time is always bounded by $O(n^2)$ irrespective of how large m might be. The key idea here is to notice that we charge *deg*(v) units of computation cost to a vertex only if *Ball*(v) is non-empty. Since the vertices are sampled randomly, the heavy degree vertices are likely to have empty Balls around them, and so the computation cost for building their Balls would be nil. This crucial observation was missing in the analysis of Thorup and Zwick.

It is less obvious to perform the second task in $O(n^2)$ time. The second task that computes BFS trees from vertices of set S would require $O(m|S|)$ time, which is certainly not $O(n^2)$ when the graph is dense. To improve its preprocessing time to $O(n^2)$ when the given graph is dense, one approach would be to perform BFS traversal from vertices of the set S on a sparse subgraph which, inspite of its sparseness, preserves pairwise (approximate) distances too. Such a subgraph is called a *spanner*.

Definition 3.1 *For a graph $G = (V, E)$, and two real numbers $\alpha \geq 1, \beta \geq 0$, a subgraph $G = (V, E_S)$, $E_S \subset E$ is said to be an (α, β)-spanner if for all $u, v \in V$, the distance $\delta_S(u, v)$ in the spanner satisfies $\delta(u, v) \leq \delta_S(u, v) \leq \alpha\delta(u, v) + \beta$.*

Note that 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 second step, lest one should end up computing a $(\alpha + 2)$-approximate distance oracle [1] instead of a 3-approximate distance oracles. To explore the possibility of using a spanner in the second task of the algorithm, let us carefully revisit the distance reporting scheme of the 3-approximate distance oracle of Thorup and Zwick. For a pair of vertices $u, v \in V$, the distance reported is exact if $u \in Ball(v, V, S)$. So let us analyse carefully the case when u lies outside *Ball*(v, V, S).

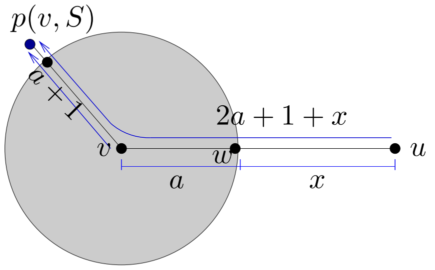


Figure 2: Carefully analysing the distance reported by the existing oracle of [15].

Let us partition a shortest path between v and u into two sub-paths : the sub-path of length a covered by $Ball(v, V, S)$, and the sub-path of length x lying outside the ball. With this partition, the distance $\delta(v, p(v, S))$ would be $a + 1$. Now considering the path between $p(v, S)$ and u that passes through v (see Figure 2), we can conclude that the distance $\delta(u, p(v, S))$ would be bounded by $2a + 1 + x$. Consequently, the distance "$\delta(v, p(v, S)) + \delta(u, p(v, S))$" as reported by 3-approximate distance oracle of Thorup and Zwick is at most $3a + x + 2$.

A comparison of this upper bound of $3a + x + 2$ on the distance reported with the actual distance $\delta(u, v) = a + x$ suggests that there is a possibility of stretching the sub-path (of length x) uncovered by *Ball*(v, V, S) by a factor of 3 and still keeping the distance reported to be 3-approximate. So we may employ a spanner for the second task of preprocessing algorithm if for each vertex $v \in V$, the shortest paths from v to all the vertices of *Ball*$(v, V, S) \cup \{p(v, S)\}$ are preserved in the spanner. The reader must note that this feature is required in addition to the $O(n^{3/2})$ size bound and suitable stretch (α, β) that the spanner must have. We introduce a special kind of spanner, called *parameterized spanner* that has these additional features.

The rest of the paper is organised as follows. In the following section we define a parameterized $(2, 1)$-spanner and present a linear time algorithm for this. The concept and the linear time algorithm for parameterized spanner is of independent interest. However, the reader who is interested in the parameterized spanner as a tool for efficient computation of approximate distance oracles, requires the features of parameterized spanner mentioned in Lemma 4.1 and Theorem 4.2 only. In section 5, we describe the expected $O(\min(n^2,\ m\sqrt{n}))$ time algorithm for 3-approximate distance oracles. We extend it to $(2k - 1)$-approximate distance oracles in section 6.

4 A parameterized $(2, 1)$-spanner

Definition 4.1 *Given a graph $G = (V, E)$, and a parameter $S \subset V$, a subgraph $G(V, E_S)$ is said to be a parameterized $(2, 1)$-spanner with respect to S if*

[1] by replacing $\delta(u, p(u, S))$ with $\alpha \cdot \delta(u, p(u, S)$ in the query bound of approximate distance oracle

(i) $G = (V, E_S)$ is a $(2,1)$-spanner.
(ii) If a vertex has no neighbor in set S, then all the edges adjacent to it are present in the parameterized spanner. [2]
(iii) All edges incident to S are present in the spanner.

Let $G = (V, E)$ be the given undirected unweighted graph and S be a subset of vertices from the graph. We shall now present an algorithm that computes a parameterized $(2,1)$-spanner $G = (V, E_S)$ with respect to S. The algorithm starts with empty spanner, i.e., $E_S = \emptyset$, and adds edges to E_S in the following three steps.

1. *Forming the clusters* :
   We form clusters (of vertices) around the vertices of the set S. Initially the clusters are $\{\{u\} | u \in S\}$. Each $u \in S$ will be referred to as the *center* of its cluster. We process each vertex $v \in V \backslash S$ as follows.

   (a) If v is not adjacent to any vertex in S, we add every edge incident on v to the set E_S.

   (b) If v is adjacent to some vertex $o \in S$, assign v to the cluster centered at $o \in S$ and add the edge $e(v, o)$ to the set E_S. In case there is more than one vertex in S adjacent to v, break ties arbitrarily [3].

   This is the end of the first step of the algorithm. At this stage let E' denote the set of all the edges of the given graph excluding the edges added to the spanner and all the intra-cluster edges. Let V' be the set of vertices corresponding to the end-points of the edges E'. It is evident that each vertex of set V' is either a vertex from S or adjacent to some vertex from S, and the first step has partitioned V' into disjoint clusters each centered around some vertex from S. The graph $G = (V', E')$ and the corresponding clustering is passed onto the second step.

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

3. *Adding edges incident on S* :
   Finally, we also add to the set E_S all the edges incident on vertices of S, that did not get added in the two steps given above to complete the construction of the spanner.

Note that the steps 1(a) and 3 ensure that the final spanner is a parameterized spanner with respect to S (refer to Definition 4.1). It is easy to verify that the 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 4.1 *If $e(u, v)$ is an edge not present in the parameterized spanner $G = (V, E_S)$ as constructed above, there is a one-edge or a two-edge path in the spanner 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 the same cluster or different clusters.
Case 1 : *(u and v belong to same cluster)*
Let the cluster centered at $o \in S$ contain both the vertices u and v. It follows from step 1(b) of the algorithm that the edges $e(u, o)$ and $e(v, o)$ are present in the spanner. So there is a one edge path from v to the center

[2] Note that it ensures that any edge $e(u, v)$ for which either u or v is not adjacent to S, is present in the spanner.
[3] We will define it more precisely in the context of the actual construction of distance oracles

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. It follows that u is adjacent to the cluster centered at y, therefore, an edge between u and some vertex, say w, of this cluster has been added to the spanner in the second step of the algorithm. It follows from step 1(b) of the algorithm that the edge $e(w, y)$ is also present in the spanner. So there is a two-edge path $u - w - y$ between u and y in the spanner. Using similar arguments, there is also a two-edge path between v and x in the spanner.

□

We can extend the above observation to $e(u, v) \in E_S$. Let $C(x)$ denote the vertex x if x does not belong to any cluster, otherwise let $C(x)$ denote the center of the cluster containing vertex x (a vertex that belongs to S is its own center). Note that a vertex x is at distance at most one from $C(x)$ implying the necessary result for $e(u, v) \in E_S$. Using the above observations, we shall now prove the following important Theorem.

Theorem 4.1 *For a graph $G = (V, E)$ and a subset $S \subset V$, the parameterized spanner $G(V, E_S)$ with respect to S computed by the algorithm above is a $(2, 1)$-spanner.*

Proof: In order to ensure that $\delta_S(u, w) \leq 2\delta(u, w) + 1$ in the subgraph $G(V, E_S)$, consider the sequence $s_{uw} : \langle u(= v_0), v_1, \cdots, v_{j-1}, w(= v_j)\rangle$ of vertices of a shortest path between u and v in the order they appear on it in the original graph. Now consider another sequence $c_{uw} : \langle v_0, C(v_1), v_2, C(v_3), \cdots\rangle$ formed by replacing each vertex v_{2i+1} of the sequence s_{uw} by $C(v_{2i+1})$, for $i < j/2$. It follows that 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 $\delta_S(u, w) \leq 2j$. 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 $G = (V, E_S)$. Thus $\delta_S(u, v) \leq 2j+1$. Combining the two cases (even and odd j), we can conclude that $\delta_S(u, v) \leq 2\delta(u, v) + 1$.

□

Now let us analyze the size of the parameterized $(2, 1)$-spanner $G = (V, E_S)$ when the set S is formed by selecting each vertex independently with probability q. In the first step of the algorithm, the expected number of edges contributed by a vertex v is at most $1 + deg(v)(1 - q)^{deg(v)}$, which is upper-bounded by $1/q$. Hence the expected number of edges added to the spanner during the first step is at most n/q. During the second step, the expected number of edges added to the spanner is at most n^2q since we add at most one edge for each vertex-cluster pair. The third step contributes expected mq edges to the spanner which is at most n^2q. Hence we can state the following lemma.

Lemma 4.2 *Let a set S be formed by selecting each vertex independently with probability q, then the expected number of edges in the parameterized $(2, 1)$-spanner $G = (V, E_S)$ will be at most $n/q + n^2q$.*

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 $2(n/q + n^2q)$. The expected number of iterations is $O(1)$. Thus a $(2, 1)$ spanner of size $O(n/q + n^2q)$ can be computed in $O(m)$ expected time. Now we state the following theorem that would highlight the crucial role played by the parameterized $(2, 1)$-spanner in constructing approximate distance oracles always in expected $O(n^2)$ time.

Theorem 4.2 *Let $G = (V, E)$ be a graph and let S be a set formed by selecting each vertex with probability p. There is an expected $O(m)$ time computable parameterized $(2, 1)$-spanner $G = (V, E_S)$ with the following properties.*
1. The spanner preserves a shortest path from v to every vertex $u \in Ball(v, V, S) \bigcup \{p(v, S)\}$
2. The number of edges in the spanner is $O(n/q + n^2q)$.

Proof: Let $v(=v_0), v_1, v_2, \cdots, v_{j-1}, u(=v_j)$ be a shortest path between v and any vertex $u \in Ball(v, V, S)$ in the graph $G = (V, E)$. Note that none of the vertices $v_i, i < j$ has any neighbor from the set S. Since the parameterized spanner would keep all those edges whose at least one end-point has no neighbor in the parameter vertex-set S, so all the edges incident on the vertices $v_i, i < j$ are present in the spanner. Hence a shortest path from v to every vertex in $Ball(u, V, S)$ is preserved in the spanner.

We shall now show that a shortest path from v to $p(v, S)$ is also preserved in the parameterized $(2, 1)$-spanner. If $p(v, S)$ is adjacent to v in the graph, then the shortest path between the two vertices is just the edge $e(v, p(v, S))$. Since $p(v, S) \in S$ and we add all those edges to the spanner which are incident on S, the edge $e(v, p(v, S))$ is surely present in the parameterized spanner. So let us consider the case when $p(v, S)$ is not adjacent to v. In this case, $Ball(v, V, S) \neq \emptyset$. So a shortest path from v to $p(v, S)$ is a shortest path from v to some vertex, say v', lying at the boundary of $Ball(v, V, S)$ concatenated with the edge $e(v', p(v, S))$. Again, using the same arguments as given above, the shortest path from v to v' as well as the edge $e(v', p(v, S))$ are present in the parameterized spanner, and we are done. $\square$

If we construct the set S by selecting each vertex independently with probability $q = n^{-1/2}$, we get a bound of $O(n^{3/2})$ on the size of $(2, 1)$-spanner. Thorup and Zwick [15] showed that $\theta(n^{3/2})$ bound on the size of (3,0)-spanner is indeed optimal. Since a $(2, 1)$-spanner is also a (3,0)-spanner, therefore, $\theta(n^{3/2})$ bound is optimal for $(2, 1)$-spanner too.

Theorem 4.3 *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.*

Note that although being of same size asymptotically, a $(2, 1)$-spanner is better than a 3-spanner since it achieves a better ratio of $\delta_S(u, v)/\delta(u, v)$, which is close to 2 for vertices separated by long distances.
Remark : The algorithm for a $(2, 1)$-spanner of expected size $O(n^{3/2})$ might appear similar to the algorithm for a 3-spanner in [4]. However, there is a subtle difference both in the construction and the analysis. The algorithm for a 3-spanner ensures that for each edge (u, v) not in the spanner, either there is a path from u to $C(v)$ of length at most two *or* a path from v to $C(u)$ of length at most two. But, as the reader may verify in Theorem 4.1, we require the existence of both these paths to prove that the spanner is a $(2, 1)$-spanner.

5 A 3-approximate distance oracle in expected $\mathbf{O}(\min(\mathbf{n}^2,\ \mathbf{m}\sqrt{\mathbf{n}}))$ time

We now describe the algorithm for computing a 3-approximate distance oracle that would require expected $O(\min(n^2,\ m\sqrt{n}))$ time. The preprocessing algorithm is essentially the same as that of [15], except that, for every $v \in V$ we store the distance to the vertices of sampled set in a parameterized $(2, 1)$-spanner rather than in the original graph.

Computing a 3-approximate distance oracle

Let $S \subset V$ contain each vertex of V independently with probability $= n^{-1/2}$.

1. Compute $p(v, S)$ and *Ball*(v, V, S) for each $v \in V$.
2. Compute a parameterized (2,1)-spanner (V, E_S) with respect to set S. (see Note below)
3. Compute distance $\delta_S(v, x)$ for each $v \in V, x \in S$ in the spanner (V, E_S).

Note : While computing a $(2,1)$-spanner in the second step of the preprocessing algorithm, we shall adopt the following strategy : in case a vertex v is a neighbor of more than one vertex from the set S, instead of assigning it to the cluster centered at any arbitrarily picked neighbor from S, we shall assign v to the cluster centered at $p(v,S)$. As a result, Lemma 4.1 for the $(2,1)$-spanner can be restated as follows.

Lemma 5.1 *If an edge $e(u,v)$ is not present in the parameterized $(2,1)$-spanner $G=(V,E_S)$, there is a one-edge or a two-edge path in the spanner between the vertex $p(u,S)$ and the vertex v, and vice versa.*

In the next subsection, we shall show that the new oracle is indeed a 3-approximate distance oracle. But prior to that, we shall prove that the expected time to compute the oracle is $O(\min(n^2,\ m\sqrt{n}))$ as follows.

There are three main steps in the construction of our oracle. The second task involves the computation of parameterized $(2,1)$-spanner with respect to S. It follows from Theorem 4.2 that this step would require expected $O(m)$ time and the size of the spanner will be $O(\min(m,\ n^{3/2}))$. The third step is accomplished by performing BFS traversal from vertices of S in the parameterized $(2,1)$-spanner. This step would require expected $O(\min(m,n^{3/2})\cdot|S|)=O(\min(m\sqrt{n},\ n^2))$ time since expected size of S is $\sqrt{n}$. Let us now analyze the computation time for the first Step. It follows from Lemma 2.1 that computing $p(v,S)$ for all $v\in V$ requires just $O(m)$ time in total. Using arguments which essentially combine Lemmas 2.2 and 2.5, Thorup and Zwick showed that it takes expected $O(m\sqrt{n})$ time to compute $Ball(v,V,S),\forall v\in V$. We shall now show that the expected time required to compute $Ball(v,V,S),\forall v\in V$ never exceeds $O(n^2)$ irrespective of how large m may be.

Lemma 5.2 *The expected time for computing $Ball(v,V,S),\forall v\in V$ never exceeds $O(n^2)$.*

Proof: We compute $Ball(v,V,S),\forall v\in V$ by executing the subroutine *Restrict_BFS*(x,S) on each $x\in V$. It follows from Lemma 2.5 that the expected cost charged to vertex v during all these subroutines will be

$$\sum_{u\in V} deg(v)\cdot \mathbf{Pr}[u\in Ball(v,V,S)]$$

Therefore, in order to establish an $O(n^2)$ bound on the expected time for computing $Ball(v,V,S),\forall v\in V$, we just need to show that this expected cost is $O(n)$. Let $\langle v_1,v_2,\cdots,v_{n-1}\rangle$ be the sequence of vertices $V\backslash\{v\}$ arranged in non-decreasing order of their distances from v. Now $v_j\in Ball(v,V,S)$ if and only if none of the vertices within distance $\delta(v,v_j)$ from v is present in S. Certainly there are at-least $\max(j,deg(v))$ such vertices. Furthermore, each vertex is selected in the set S independently with probability $q=n^{-1/2}$. So the expected cost charged to v can be bounded as follows.

$$\begin{aligned}
\sum_{u\in V} deg(v)\cdot \mathbf{Pr}[u\in Ball(v,V,S)] &= \sum_{j} deg(v)\cdot(1-q)^{\max(j,deg(v))}\\
&= \sum_{j\le deg(v)} deg(v)\cdot(1-q)^{deg(v)} + \sum_{j>deg(v)} deg(v)\cdot(1-q)^{j}\\
&\le (deg(v))^2\cdot(1-q)^{deg(v)} + \sum_{j>deg(v)} j\cdot(1-q)^{j}\\
&\le (deg(v))^2\cdot(1-q)^{deg(v)} + 1/q^2\\
&\le 1/q^2+1/q^2 \qquad \{\text{Since } x^2(1-\alpha)^x\le 1/\alpha^2, \forall\alpha,x>0\}\\
&= O(n) \quad \{\text{since } q=n^{-1/2}\ \}
\end{aligned}$$

□

Based on Lemma 5.2 and the preceding discussion, we can conclude that the expected preprocessing time for the new approximate distance oracle is $O(\min(n^2,\ m\sqrt{n}))$. The expected size of the oracle is $\mathbf{E}[\sum_u |Ball(u,S)| + |S| \cdot n]$ which, using Lemma 2.2, is at most $2n^{3/2}$. We shall repeat the algorithm if the size exceeds $4n^{3/2}$, and using Markov's Inequality, the probability for this event will be at most $1/2$. So the expected number of repetitions will be $O(1)$. Hence we can state the following theorem.

Theorem 5.1 *The new approximate distance oracle requires expected $O(\min(n^2,\ m\sqrt{n}))$ preprocessing time and occupies $O(n^{3/2})$ space.*

5.1 Reporting distance with stretch at most 3

We describe below the query answering procedure for the new approximate distance oracle. It differs from the algorithm of [15] only for the case when $u \notin Ball(v, V, S)$.

> $\mathcal{Q}(u, v)$ *: Reporting approximate distance between u and v*
> `If` $u \in Ball(v, V, S)$
>   `return` $\delta(u, v)$
> `Else report` **minimum** of
> $\delta(u, p(u, S)) + \delta_S(v, p(u, S))$ & $\delta(v, p(v, S)) + \delta_S(u, p(v, S))$

We shall now show that the new oracle ensures a stretch of 3 at most, that is, for any pair of vertices (u, v), the distance reported is at most 3 times $\delta(u, v)$. Clearly, from triangle inequality the reported distance is at least $\delta(u, v)$

Theorem 5.2 *The query answering algorithm $\mathcal{Q}(u, v)$ reports 3-approximate distance between u and v.*

Proof: If $u \in Ball(v, V, S)$, we report the exact distance since we, just like [15], store the exact distance from v to all the vertices of $Ball(v, V, S)$. So it is the case $u \notin Ball(v, V, S)$ that needs to be analyzed carefully. In fact there are the following two sub-cases.

Case 1: $\underline{Ball(u, V, S) = \emptyset}$

In this case we show that $\delta(u, p(u, S)) + \delta_S(v, p(u, S))$ is at most $3\delta(u, v)$ using Lemma 5.1 and the fact that $\delta_S(v, p(u, S))$ is the distance between v and $p(u, S)$ in a $(2, 1)$-spanner.

Let $v_0(= u), v_1, \cdots, v_l(= v)$ be a shortest path between u and v. Observe that $Ball(u, V, S) = \emptyset$ implies that u must be adjacent to $p(u, S)$, so $\delta(u, p(u, S)) = 1$. It follows from Lemma 5.1 that there is a path between $p(u, S)$ and v_1 in the spanner that consists of at most 2 edges. Furthermore, by the property of $(2, 1)$-spanner, the distance $\delta_S(v_1, v_l)$ between v_1 and v_l in the spanner is not greater than $3(l - 1)$. Hence $\delta_S(v, p(u, S)) \leq 3(l - 1) + 2$. So

$$\delta(u, p(u, S)) + \delta_S(v, p(u, S)) \leq 1 + 3(l - 1) + 2 = 3l = 3\delta(u, v)$$

Case 2: $\underline{Ball(u, V, S) \neq \emptyset}$

In this case we show that $\delta(v, p(v, S)) + \delta_S(u, p(v, S))$ is at most $3\delta(u, v)$ using the properties of the parameterized $(2, 1)$-spanner. The proof is along the lines as sketched in section 3.1.

A shortest path from v to u can be visualized as concatenation of two sub-paths (see Figure 3) : the sub-path $\mathcal{P}_{vw}$ of length a lying inside the Ball and the sub-path $\mathcal{P}_{wu}$ of length $x \geq 1$ outside the Ball. (In case, $a = 0$, the vertex w would be same as v.) Let the length of path $\mathcal{P}_{wu}$ be stretched to x' in the parameterized spanner. Since the spanner is a parameterized spanner with respect to S, it follows from Theorem 4.2 that

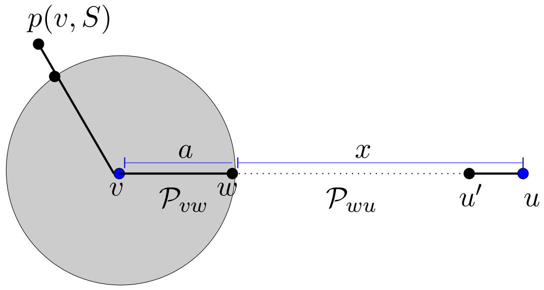


Figure 3: Analyzing the path from v to u when $Ball(u, V, S) \neq \emptyset$

$\delta_S(v, p(v, S)) = a + 1$ and $\delta_S(v, w) = a$. Now considering the path from u to $p(v, S)$ passing through v it follows that $\delta_S(u, p(v, S)) \leq 2a + 1 + x'$. Hence,

$$\delta(v, p(v, S)) + \delta_S(u, p(v, S)) \leq 3a + 2 + x'$$

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 $Ball(u, V, S) \neq \emptyset$ implies that $u' \in Ball(u, R)$. So it follows from Theorem 4.2 that the edge $e(u', u)$ must be present in the parameterized spanner. Moreover, the part of the sub-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 $(2, 1)$-spanner. Hence $x' \leq 3(x - 1) + 1 = 3x - 2$, and we are done. $\square$

Combining Theorems 5.1 and 5.2, we can state the following theorem.

Theorem 5.3 *An undirected unweighted graph on n vertices and m edges can be preprocessed in expected $O(\min(n^2,\ m\sqrt{n}))$ time to compute a 3-approximate distance oracle of size $O(n^{3/2})$.*

6 A $(2k - 1)$-approximate distance oracle in expected $O(n^2)$ time

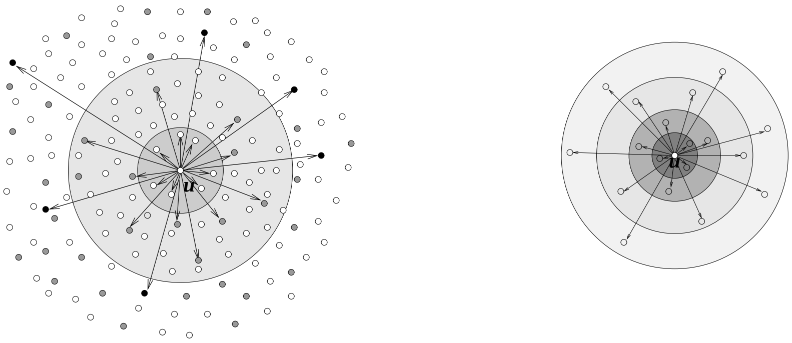
In this section we shall describe the construction of a $(2k - 1)$-approximate distance oracle which can be viewed as a generalization of the new 3-approximate distance oracle.

Computing (2k-1)-approximate distance oracle

$S_1 \longleftarrow V$
`For` $i \leftarrow 2$ `to` k
let S_i contain each element of S_{i-1}, independently, with probability $n^{-1/k}$

1. `For` $i \leftarrow 2$ `to` k
   Compute $p(v, S_i)$ and *Ball*(v, S_{i-1}, S_i) for all $v \in V$
2. Compute a parameterized $(2, 1)$-spanner (V, E_S) with $S = S_k$.
   (see Note below)
3. Compute $\delta_S(v, x)$ for each $v \in V, x \in S_k$ in the spanner (V, E_S) for $S = S_k$.

Note : While computing a $(2,1)$-spanner in the second step of the preprocessing algorithm, we shall adopt the following strategy : In case a vertex v is neighboring to more than one vertex from set S_k, instead of assigning it to the cluster centered at any arbitrarily picked neighbor from S_k, we shall assign v to the cluster centered at $p(v, S_k)$.



(i) (ii)

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

The final data-structure would keep k hash tables per vertex $v \in V$ as follows. For $1 < i \leq k$, the set of vertices $Ball(v, S_{i-1}, S_i) \cup \{p(v, S_i)\}$ and their distances from v are kept in a hash-table denoted by $Ball^{i-1}(v)$ henceforth. The distances from v to all the vertices of set S_k in the $(2,1)$-spanner is kept in a hash-table denoted as $Ball^k(v)$.

Since S_{i+1} is formed by selecting each vertex of the set S_i with probability $n^{-1/k}$, it follows from Lemma 2.2 that the expected size of $Ball(v, S_i, S_{i+1})$ is at most $n^{1/k}$. Hence, hash-table $Ball^i(v)$ would occupy expected $O(n^{1/k})$ space for each i. It follows that the final data-structure will require expected $O(kn^{1+1/k})$ space.

Let us first analyze the computation time required for the new distance oracle. There are three main tasks in the preprocessing algorithm. The second task involves the computation of a parameterized $(2,1)$-spanner with respect to S_k. It follows from Theorem 4.2 that it would require expected $O(m)$ time and the size of the spanner would be $O(\min(m,\ n^{2-1/k}))$. The third task requires computation of BFS trees in the spanner from each vertex of set S_k, and hence can be computed in expected $O(\min(mn^{1/k},\ n^2))$ time since the expected size of S_k is $O(n^{1/k})$. Now let us analyze the first task that requires computation of $p(v, S_i)$ and $Ball(v, S_{i-1}, S_i)$. Similar to the 3-approximate distance oracle, $Ball(v, S_{i-1}, S_i), \forall v \in V$ is computed by performing *Restrict_BFS*(x, S_i) on every vertex $x \in S_{i-1}$. Using arguments which essentially combine Lemmas 2.2 and 2.5, Thorup and Zwick showed that it takes expected $O(mn^{1/k})$ time to compute $Ball(v, S_{i-1}, S_i), \forall v \in V$. Using an improved analysis, which is basically a generalization of Lemma 5.2, we shall now provide a tighter bound on the expected time required to compute $Ball(v, S_{i-1}, S_i) \forall v \in V$.

Lemma 6.1 *The expected time required to compute $Ball(v, S_{i-1}, S_i), \forall v \in V$ never exceeds $O(n^{\frac{k+i}{k}})$.*

Proof: We compute $Ball(v, S_{i-1}, S_i), \forall v \in V$ by executing the subroutine *Restrict_BFS*(x, S_i) on each $x \in S_{i-1}$. It follows from Lemma 2.5 that the expected computation cost charged to a vertex v during all these subroutines is bounded by $\sum_u \deg(v) \mathbf{Pr}[u \in ball(v, S_{i-1}, S_i)]$. It would suffice if we show that this expected cost is $O(n^{i/k})$. To do so, let us first calculate $\mathbf{Pr}[u \in Ball(v, S_{i-1}, S_i)]$. Let $\mathcal{D}_v^u$ denote the set of vertices within distance $\delta(u, v)$ from v. It follows from Definition 2.1 that u will belong to $ball(v, S_{i-1}, S_i)$ iff $u \in S_{i-1}$ and none of the vertices of set $\mathcal{D}_v^u$ is present in S_i. Now whether or not a vertex belongs to set

S_i is independent of any other vertex. Using this independence,

$$\begin{aligned}\mathbf{Pr}[u \in ball(v, S_{i-1}, S_i)] &= \mathbf{Pr}[u \in S_{i-1} \wedge u \notin S_i] \cdot \prod_{w \in \mathcal{D}_v^u \setminus \{u\}} \mathbf{Pr}[w \notin S_i] \\ &= \mathbf{Pr}[u \in S_{i-1}] \cdot \mathbf{Pr}[u \notin S_i | u \in S_{i-1}] \cdot \prod_{w \in \mathcal{D}_v^u \setminus \{u\}} \mathbf{Pr}[w \notin S_i]\end{aligned}$$

Now note that $\mathbf{Pr}[u \in S_i | u \in S_{i-1}] > \mathbf{Pr}[u \in S_i]$ and, therefore, $\mathbf{Pr}[u \notin S_i | u \in S_{i-1}] < \mathbf{Pr}[u \notin S_i]$. So

$$\begin{aligned}\mathbf{Pr}[u \in ball(v, S_{i-1}, S_i)] &< \mathbf{Pr}[u \in S_{i-1}] \cdot \mathbf{Pr}[u \notin S_i] \cdot \prod_{w \in \mathcal{D}_v^u \setminus \{u\}} \mathbf{Pr}[w \notin S_i] \\ &= n^{\frac{-i+2}{k}} \cdot \prod_{w \in \mathcal{D}_v^u} \mathbf{Pr}[w \notin S_i] \;=\; n^{\frac{-i+2}{k}} \cdot \mathbf{Pr}[u \in ball(v, V, S_i)]\end{aligned}$$

Hence, the expected computation cost charged to v is $n^{\frac{-i+2}{k}} \sum_u deg(v) \cdot \mathbf{Pr}[u \in ball(v, V, S_i)]$. Using arguments similar to those used in Lemma 5.2, it follows that $\sum_u deg(v) \cdot \mathbf{Pr}[u \in ball(v, V, S_i)] = 1/q^2$, where $q = n^{\frac{-i+1}{k}}$. Hence, the expected cost charged to vertex v is $n^{\frac{i}{k}}$, and we are done. □

The preprocessing algorithm computes $Ball(v, S_{i-1}, S_i) \forall v \in V$ for each $i \leq k$. Hence using Lemma 6.1 and the preceding discussion, we can conclude that the expected preprocessing time for the new approximate distance oracle is $O(\min(n^2,\ mn^{1/k}))$. The expected size of the oracle is of the order of $kn^{1/k}$. We shall repeat the algorithm if the size exceeds twice this bound, and using Markov's Inequality, the probability of this event will be at most $1/2$. So the expected number of repetitions will be $O(1)$. Hence we can state the following theorem.

Theorem 6.1 *The new approximate distance oracle requires expected $O(\min(n^2,\ kmn^{1/k}))$ preprocessing time and occupies $O(kn^{1/k})$ space.*

6.1 Reporting approximate distance with stretch at most $(2\mathbf{k} - 1)$

Just like the new 3-approximate distance oracle, there is only a subtle difference between the query answering procedure of the new and the previously existing $(2k-1)$-approximate distance of [15]. First, we provide an overview of the underlying query answering procedure before we formally state it.

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

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

The entire query answering process is to search for a t-near vertex of u whose distance to both u and and v is known, and $t < k$. The search is performed iteratively as follows : The ith iteration begins with a vertex from set $w \in S_i$ which is $(i-1)$-near to u (and hence i-near to v) and its distance from u is known. It is determined whether or not its distance to v is known. Since $w \in S_i$, we do so by checking whether or not $w \in Ball^i(v)$. Now $w \notin Ball^i(v)$ only if, for the vertex v, the vertex $p(v, S_{i+1})$ is equidistant or nearer than the vertex $p(u, S_i)$. But this would imply that $p(v, S_{i+1})$ is also i-near to v, and note that its distance from v is known. Therefore, if the distance $\delta(v, w)$ is not known, we continue in the next iteration with vertex $w = p(v, S_{i+1})$, and swap the role of u and v. In this way we proceed gradually, searching over the sets $S_1, \cdots, S_k$. We are bound to find such a vertex within at most k iterations since the distance from S_k is known to all the vertices. We now formally state the query answering procedure which is essentially the same as that of [15] except the '`If`' loop at the end.

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

```
i ← 1
While (p(u, S_i) ∉ Ball^i(v)) do
  swap(u, v)
  i ← i + 1

If i < k
  return δ(u, p(u, S_i)) + δ(v, p(u, S_i))
Else
  return minimum of
  δ(u, p(u, S_k)) + δ_S(v, p(u, S_k))  &  δ(v, p(v, S_k)) + δ_S(u, p(v, S_k))
```

Based on the arguments above, the following lemma holds.

Lemma 6.2 *[15] In the beginning of the ith iteration of* `while` *loop,* $\delta(u, p(u, S_i)) \leq (i-1)\delta(u, v)$.

Theorem 6.2 *The query answering algorithm $\mathcal{Q}(u,v)$ reports $(2k-1)$-approximate distance between u and v.*

Proof: The query answering algorithm performs iterations of the `While` loop until the condition of the loop fails. As mentioned above, there will be at most $(k-1)$ successful iterations before the condition of the loop fails. Let the condition fails in the beginning of ith iteration. So the following inequality follows from Lemma 6.2 when we exit the `While` loop.

$$\delta(u, p(u, S_i)) \leq (i-1)\delta(u, v) \tag{1}$$

We shall analyze the distance reported by the oracle on the basis of the final value of i.
Case 1 : $i < k$
In this case, the reported distance is $\delta(u, p(u, S_i)) + \delta(v, p(u, S_i))$, which using the triangle inequality is at most $2\delta(u, p(u, S_i)) + \delta(u, v)$. Using Inequality 1, this distance is at most $(2i-1)\delta(u, v)$.
Case 2 : $i = k$.
There are the following two sub-cases depending upon whether or not v belongs to $Ball(u, V, S_k)$ as shown in Figure 5.

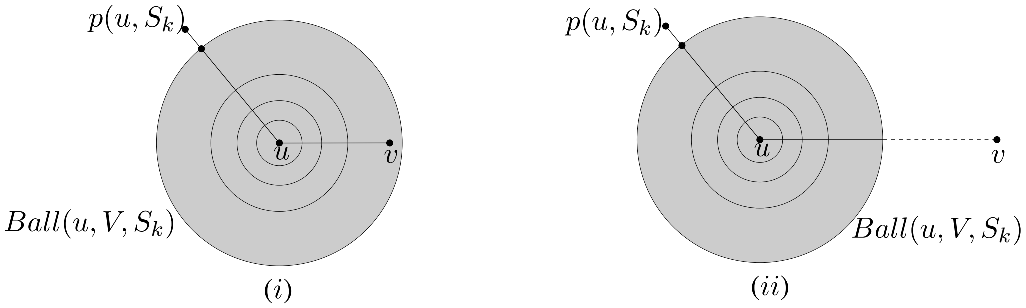


Figure 5: Two sub-cases depending upon whether or not $v \in Ball(u, V, S_k)$

In sub-case (i), the vertex v belongs to $Ball(u, V, S_k)$. Therefore, using Theorem 4.2, it follows that $\delta_S(v, u) = \delta(v, u)$ and $\delta_S(u, p(u, S_k)) = \delta(u, p(u, S_k))$. Combining these two equalities together and using the triangle inequality, it follows that $\delta_S(v, p(u, S_k)) \leq \delta(v, u) + \delta(u, p(u, S_k))$. Hence the distance reported is at most $(2k-1)\delta(u, v)$ as shown below.

$$\begin{aligned}\delta(u, p(u, S_k)) + \delta_S(v, p(u, S_k)) &\leq 2\delta(u, p(u, S_k)) + \delta(u, v) \\ &\leq 2(k-1)\delta(u, v) + \delta(u, v) \quad \{\text{using Inequality 1}\} \\ &= (2k-1)\delta(u, v)\end{aligned}$$

In sub-case (ii), the vertex v does not belong to $Ball(u, V, S_k)$. Therefore, $p(u, S_k)$ is in fact equidistant or closer to u than the vertex v, that is, $\delta(u, p(u, S_k)) \leq \delta(u, v)$. It is not difficult to observe that the sub-case (ii) is exactly the *same* as that addressed for 3-approximate distance oracle in Theorem 5.2 with S_k playing the role of S. So in this sub-case, the stretch of the distance reported by the oracle is in fact at most 3. □

Combining Theorems 6.1 and 6.2, we can state the following theorem.

Theorem 6.3 *An undirected unweighted graph on n vertices and m edges can be preprocessed in expected $O(\min(n^2,\ kmn^{1/k}))$ time to compute a $(2k-1)$-approximate distance oracle of size $O(kn^{1+1/k})$, for any integer $k > 1$.*

7 Conclusion and Open Problems

In this paper, we presented an expected $O(n^2)$ time algorithm for computing a $(2k-1)$-approximate distance oracle for unweighted graphs. Recently Roditty *et al.* [14] gave a deterministic algorithm that computes a $(2k-1)$-approximate distance oracle for weighted graphs in $O(mn^{1/k})$ time. They also claim that, for unweighted graphs there exists a deterministic algorithm for computing a $(2k-1)$-approximate distance oracle in $O(n^2)$ time. However, computing a $(2k-1)$-approximate distance oracles for *weighted graphs* in expected or deterministic $O(n^2 \operatorname{polylog} n)$ time is still an important open problem.

Another result of the paper is a simple and expected linear time algorithm to compute a $(2,1)$-spanner of $O(n^{3/2})$ size which is optimal. The algorithm is obtained by a slight modification in the algorithm [4] for computing a 3-spanner. It is a natural and interesting problem to extend this algorithm for arbitrary k, that is, designing a linear time algorithm for computing a $(k, k-1)$-spanner of size $O(n^{1+1/k})$. Subsequent to the submission of our paper, this result was achieved in [5].

Acknowledgement We wish to thank the anonymous referees who provided very detailed and insightful comments that has improved the overall presentation and organization. Their feedback also resulted in an $O(\log n)$ improvement in the running time from a previous version as well as rectification of a subtle drawback in Step 2 of the algorithm for computing $(2k-1)$-approximate distance oracle.

References

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

[2] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear time construction of sparse neighborhod covers. *SIAM Journal on Computing*, 28:263–277, 1998.

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

[4] Surender Baswana and Sandeep 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.

[5] Surender Baswana, Kavitha Telikepalli, Kurt Mehlhorn, and Seth Pettie. New construction of (α, β)-spanners and purely additive spanners. In *Proceedings of 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 672–681, 2005.

[6] Timothy Chan. All-pairs shortest paths with real edge weights in $O(n^3/\log n)$ time. In *Proceedings of Workshop on Algorithms and Data Structures*, volume 3608, pages 318–324, 2005.

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

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

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

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

[11] Michael L. Fredman, Janos Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case time. *Journal of Association of Computing Machinery*, 31:538–544, 1984.

[12] Shay Halperin and Uri Zwick. Linear time deterministic algorithm for computing spanners for unweighted graphs. *Unpublished Result*, 1996.

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

[14] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic construction of approximate distance oracles and spanners. In *Proceedings of 32nd International Colloquim on Automata, Languagaes and Programming (to appear)*, 2005.

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