Lossless Derandomization for Undirected Single-Source Shortest Paths and Approximate Distance Oracles

Shuyi Yan *

Abstract

A common step in algorithms related to shortest paths in undirected graphs is that, we select a subset of vertices as centers, then grow a ball around each vertex until a center is reached. We want the balls to be as small as possible. A randomized algorithm can uniformly sample r centers to achieve the optimal (expected) ball size of $\Theta(n/r)$. A folklore derandomization is to use the $O(\log n)$ approximation for the set cover problem in the hitting set version where we want to hit all the balls with the centers.

However, the extra $O(\log n)$ factor is sometimes too expensive. For example, the recent $O(m\sqrt{\log n \log\log n})$ undirected single-source shortest path algorithm [DMSY23] beats Dijkstra's algorithm in sparse graphs, but the folklore derandomization would make it dominated by Dijkstra's.

In this paper, we exploit the fact that the sizes of these balls can be adaptively chosen by the algorithm instead of fixed by the input. We propose a simple deterministic algorithm achieving the optimal ball size of $\Theta(n/r)$ on average. Furthermore, given any polynomially large cost function of the ball size, we can still achieve the optimal cost on average. It allows us to derandomize [DMSY23], resulting in a deterministic $O(m\sqrt{\log n \log\log n})$ algorithm for undirected single-source shortest path.

In addition, we show that the same technique can also be used to derandomize the seminal Thorup-Zwick approximate distance oracle [TZ05], also without any loss in the time/space complexity.

*BARC, University of Copenhagen. Supported by VILLUM Foundation Grant 54451. Email: shya@di.ku.dk.

1 Introduction

The single-source shortest path is a fundamental problem in graph theory. Given a graph with n vertices and m edges with non-negative real weights, we want to find the distances from a given source s to all vertices. The textbook Dijkstra's algorithm [Dij59] with a Fibonacci heap [FT87] can solve this problem in $O(m+n\log n)$ time. Recently, [DMSY23] gave the first algorithm breaking this bound for undirected sparse graphs. Their algorithm is randomized and runs in $O(m\sqrt{\log n \log\log n})$ time.

Their algorithm uses randomization to solve the following subproblem, which widely appears in algorithms regarding shortest paths, distance oracles and spanners. We want to select a subset of vertices R as centers. Then for each vertex v, we grow a ball $B(v)$ until we reach a center, i.e. $B(v)$ contains all vertices that are closer than (all vertices in) R to v. The performance of the algorithm then depends on the size of R and the sizes of the balls. As an example, for [DMSY23], the total time complexity will be $O(|R|\log n + \sum_v |B(v)|\log|B(v)|)$[1].

For a randomized algorithm, it's easy to achieve the optimal tradeoff between these two sizes. By sampling r centers uniformly at random, the expected size of each ball will be $\Theta(n/r)$[2]. There is a folklore derandomization, which first fixes all the balls and then repeatedly selects a new center which hits the largest number of unhit balls. It introduces an additional $O(\log n)$ factor on the number of centers, and some algorithms cannot afford this. For example, applying this derandomization to [DMSY23] will increase the time complexity to $O(m\log n\sqrt{\log\log n})$.

However, it is unnecessary to fix the balls in the beginning. In this paper, we exploit the fact that these balls are growable by the algorithm. We show a very simple derandomization which preserves the optimal randomized bound in an amortized way. The result can be generalized to any polynomially large function of the ball size. Below, we state our result in the context of derandomizing [DMSY23]. The formal and more general results are included in Section 2.

Theorem 1 (Informal)**.** *For any $1 \le r \le n$, we can deterministically select the center set R with size $|R| = r$, such that $\sum_v |B(v)|\log|B(v)| = O(n\cdot(n/r)\log(n/r)) = O((n^2/r)\log(n/r))$, in $O(r + (n^2/r)\log(n/r))$ time.*

Combining Theorem 1 with the remaining deterministic part of [DMSY23], we get a deterministic algorithm with the same time complexity. The details are included in Section 4.

Theorem 2. *In an undirected graph with non-negative edge weights, there is a deterministic comparison-addition algorithm that solves the single-source shortest path problem in $O(m\sqrt{\log n\log\log n})$ time.*

Our derandomization can also be applied to some other randomized algorithms that involve similar center-ball techniques. An example is the seminal Thorup-Zwick approximate distance oracle [TZ05]. For any integer $k \ge 1$, [TZ05] can preprocess an undirected non-negatively weighted graph in $O(kn^{1/k}(m+n\log n))$ expected time, constructing a data structure of size $O(kn^{1+1/k})$, which is able to answer any pairwise distance query, of stretch $2k-1$[3], in $O(k)$ time.

The original derandomization in [TZ05] takes $\tilde{O}(mn)$ time and increases the space complexity to $O(kn^{1+1/k}\log n)$. Later, [RTZ05] proposed a better derandomization, which only loses one $O(\log n)$ factor in the time complexity and has no (asymptotic) loss in the space complexity. However, using our method, this last $O(\log n)$ factor can also be removed. The details are included in Section 5.

[1] This is for their main case that $n = \Omega(m)$.

[2] This is optimal, for example, when the graph is a path.

[3] It means that, if the real distance between u and v is $d(u,v)$, then the estimated distance $\hat{d}(u,v)$ satisfies $d(u,v) \le \hat{d}(u,v) \le (2k-1)d(u,v)$.

Theorem 3. *The Thorup-Zwick approximate distance oracle of size $O(kn^{1+1/k})$ can be deterministically constructed in $O(kn^{1/k}(m + n \log n))$ time.*

We remark that Theorem 3 can be combined with other improved query algorithms for the Thorup-Zwick oracle. For example, combining it with [Wul13] will give us $O(\log k)$ query time.

1.1 Other Related Works

Single-Source Shortest Path. Very recently, [DMM+25] proposed an $O(m \log^{2/3} n)$-time deterministic algorithm for directed graphs, using different techniques from [DMSY23].

In the word RAM model with integer weights, there are a sequence of works [FW93, FW94, Ram96, Ram97, Tho00a, Tho00b, Hag00] leading to a linear-time algorithm for undirected graphs [Tho99] and an $O(m + n \log \log \min\{n, C\})$-time algorithm for directed graphs [Tho04] where C is the maximum edge weight.

Approximate Distance Oracle. The size-stretch tradeoff of the Thorup-Zwick oracle [TZ05] is optimal up to a factor of k, assuming a widely believed girth conjecture [Erd64]. Besides, it is the first approximate distance "oracle" that achieves a constant query time for any fixed stretch. Before that, the query time was $\Omega(n^{1/k})$ [ADD+93, ABCP98, Coh98, Mat96].

The query time of the Thorup-Zwick oracle was improved to $O(\log k)$ by [Wul13]. A line of research achieved a universal constant query time and $O(n^{1+1/k})$ size [MN06, MS09, Wul13, Che14, Che15] by designing other approximate distance oracles. However, they have worse preprocessing times and/or worse stretches.

There are also many further improvements in various special settings [BS06, BK06, BGSU08, BK10, Wul12, PR14, PRT12, AG13, Aga14, RT23, CZ22, KKR24], e.g. for sparse/dense graphs, special stretches, etc.

2 Hitting Growable Balls

We define the problem of hitting growable balls as follows. There are n vertices (elements), m balls (sets) which are initially empty, a parameter $r \in [1, n]$ and a cost function $f(\cdot)$. The algorithm can select a ball (which does not contain all vertices) and ask the adversary to include one new vertex (determined by the adversary) in it. The algorithm can grow the balls as many times as it wants. In the end, it needs to select r vertices as centers, such that every ball is hit by the centers, i.e. each ball contains at least one center. The objective is to minimize the total cost $\sum_{i=1}^{m} f(|B_i|)$, where B_i denotes the i-th ball.

The cost function is typically positive and convex. For simplicity of presentation, in the remaining part of this section, we assume that the cost function is $f(x) = x^p$ for some constant $p \geq 1$. Our results could be extended to other polynomially large cost functions simply by applying Jensen's inequality.

A randomized algorithm could simply select the centers at random, and then grow each ball until it contains a center. Against an oblivious adversary, the expected cost will be $O_p(m \cdot f(n/r))$[4], which is (asymptotically) optimal since the adversary can easily force the average ball size to be $\Omega(n/r)$[5].

Surprisingly, we have a simple deterministic algorithm achieving the same bound. The idea is as follows. When each unhit ball has a size of n/r, we can use $O(r)$ centers to hit a constant

[4] $O_p(\cdot)$ indicates that the constant hidden in the O-notation may depend on p.

[5] An example is that, when B_i grows in the j-th time, the adversary let it include the vertex $v_{(i+j) \bmod n}$.

fraction, say $1-\varepsilon$, of unhit balls, by repeatedly selecting a new center that hits the largest number of unhit balls. Repeating it $O(\log m)$ times, we can hit all the balls with $O(r \log m)$ centers. To remove the $O(\log m)$ factor, we exponentially reduce the number of new centers in each round. In the i-th round, we only use $O(r/2^i)$ centers. This means that each unhit ball should have size $2^i n/r$. However, only ε^i fraction of the balls remain unhit in the i-th round. By setting $\varepsilon < 1/2$, the average ball size would still be $O(n/r)$.

The pseudocode of our algorithm is given below. Recall that the objective is to minimize the total cost $\sum_{i=1}^{m} f(|B_i|)$ where the cost function is $f(x) = x^p$.

Algorithm 1

Input: number of vertices n, number of balls m, target number of centers r, cost parameter p.
Output: at most r centers that hit all the balls.

Initially, all balls are empty and no centers are selected.
$b \leftarrow \lceil 2^{p+2} n/r \rceil$.
Grow each ball to the size $\min\{b, n\}$.
while not all the balls are hit **do**
    $m' \leftarrow$ the number of unhit balls.
    Repeatedly select a new center that hits the largest number of unhit balls, until the number of unhit balls is at most $m'/2^{p+1}$.
    $b \leftarrow 2b$.
    Grow each unhit ball to the size $\min\{b, n\}$.
return all selected centers.

Theorem 4. *For any $p \geq 1$ and $1 \leq r \leq n$, Algorithm 1 selects at most r centers that hit all the balls at a total cost $O_p(m(n/r)^p) = O_p(m \cdot f(n/r))$, and it can be implemented in $O_p(r + mn/r)$ time.*

Proof. As long as $b \geq n$, the algorithm will terminate in one step. Before that, there are at most $\log r$ rounds. In the i-th round, each unhit ball has a size of at least $2^{p+1+i} n/r$. When we select a new center, there are at least $m'/2^{p+1}$ unhit balls by definition. So, a new center selected in the i-th round will hit at least

$$\frac{m'}{2^{p+1}} \cdot \frac{2^{p+1+i} n}{r} \cdot \frac{1}{n} = \frac{2^i m'}{r}$$

unhit balls. On the other hand, there are only m' unhit balls. So, at most $r/2^i$ new centers are selected in the i-th round, which means the total number of selected centers is at most r.

In each round, the number of unhit balls is divided by at least 2^{p+1}. So, at the beginning of the i-th round, at most $m/2^{(i-1)(p+1)}$ balls are unhit, which have grown to the size $2^{p+1+i} n/r$. Therefore, the total cost is at most

$$\begin{aligned}
\sum_{i=1}^{\log r} \frac{m}{2^{(i-1)(p+1)}} \cdot f\left(\frac{2^{p+1+i} n}{r}\right) &= \sum_{i=1}^{\log r} \frac{m}{2^{(i-1)(p+1)}} \cdot \left(\frac{2^{p+1+i} n}{r}\right)^p \\
&= m(n/r)^p \sum_{i=1}^{\log r} 2^{(p+1)^2 - i} \\
&\leq 2^{(p+1)^2} m(n/r)^p \\
&= O_p(m(n/r)^p).
\end{aligned}$$

Let a_j denote how many unhit balls can be hit by vertex j. When we grow an unhit ball, either it becomes hit by some current center, or some a_j is increased by 1. When we hit a ball B_i, we decrease $|B_i|$ different a_j's by 1 since B_i is no longer unhit. Therefore, the total number of increases/decreases is $\sum_{i=1}^{m} |B_i| = O_p(mn/r)$. Since a_j is only changed by 1 each time, each increase, decrease and find-max operation can be done in constant time[6]. Therefore, the time complexity is $O_p(r + mn/r)$. □

In the applications, we will set the cost function according to the time complexity of the remaining part of the algorithm. In this paper, it is either $f(x) = x$ or $f(x) = x \log x$. By Jensen's inequality, for any $p > 1$, $\frac{1}{m}\sum_{i=1}^{m} |B_i|^p \le (n/r)^p$ would imply $\frac{1}{m}\sum_{i=1}^{m} |B_i| \log |B_i| \le (n/r)\log(n/r)$. This gives us the following corollary.

Corollary 5. *Let the cost function be $f(x) = x \log x$. For any $1 \le r \le n$, Algorithm 1 (with e.g. $p = 2$) selects at most r centers that hit all the balls at a total cost $O(m(n/r)\log(n/r)) = O(m \cdot f(n/r))$ in $O(r + mn/r)$ time.*

Proof. Let $g(x) = \sqrt{x} \log \sqrt{x}$ and $h(x) = x^2$. Note that g is increasing and concave (when $x \ge 1$), and $f(x) = g(h(x))$. Running Algorithm 1 with $p = 2$, we have

$$\frac{1}{m}\sum_{i=1}^{m} h(|B_i|) \le h(n/r).$$

So,

$$\frac{1}{m}\sum_{i=1}^{m} f(|B_i|) = \frac{1}{m}\sum_{i=1}^{m} g(h(|B_i|)) \le g\left(\frac{1}{m}\sum_{i=1}^{m} h(|B_i|)\right) \le g(h(n/r)) = f(n/r).$$

□

We remark that the result can be similarly extended to other polynomially large cost functions.

3 Preliminaries for the Applications

In the remaining part of the paper, we consider a weighted undirected graph G with n vertices, m edges and non-negative edge weights. For each edge (u, v), let $l(u, v)$ denote the weight (length) of the edge. For any vertices u and v, let $d(u, v)$ denote the distance between u and v. For any vertex u and any set of vertices S, let $d(u, S) = \min_{v \in S}\{d(u, v)\}$.

Bounded-Degree Graph. We assume that the degree of each vertex is $O(m/n)$. This can be accomplished by splitting the vertices of too large degrees so that in the new graph, the number of vertices is still $O(n)$ and the number of edges is still $O(m)$.

Comparison-Addition Model. We consider the comparison-addition model, where the weights are subject to only comparison and addition operations. Each comparison and addition takes unit time.

Heap. We will use heaps that support each initialization, insertion and decrease-key operation in $O(1)$ amortized time and each extract-min operation in $O(\log |H|)$ time where $|H|$ is the size of the heap, e.g. the Fibonacci heap [FT87].

[6]For example, we can maintain: (1) a_j for each vertex j; (2) a doubly linked list L_k of vertices with $a_j = k$ for each k; (3) the largest k such that L_k is non-empty.

Dijkstra's Algorithm. The Dijkstra's algorithm [Dij59] starts at a source vertex s and repeatedly finds the closest unvisited vertex from a heap. In the i-th step, the heap only contains the neighbors of the i closest vertices, hence its size is at most im/n. Therefore, it spends $O(m/n+\log(im/n))$ time in the i-th step.

4 Derandomizing Undirected Single-Source Shortest Paths

4.1 Bundle Dijkstra

In the single-source shortest path problem, given a source vertex s, we need to compute $d(s, v)$ for each vertex v. The randomized algorithm in [DMSY23] solves the problem in $O(m\sqrt{\log n \log\log n})$ time. The algorithm consists of two stages: bundle construction and bundle Dijkstra.

Constant-Degree Graph. For simplicity, [DMSY23] further assumes that each vertex has a constant degree. To achieve this, the number of vertices is increased to $O(m)$. We will follow the same assumption. Note that this assumption is not necessary for their algorithm (See [DMSY23] Section 5). Without it, the time complexity can be slightly improved to $O(\sqrt{mn\log n} + n\sqrt{\log n \log\log n})$ when $m = \omega(n)$ and $m = o(n \log n)$, and our derandomization also holds.

Bundle Construction. In this stage, we need to select a subset of vertices R as centers. Then for each vertex v, we run a partial Dijkstra's algorithm from v until we reach a center. Let $B(v)$ be the set of vertices that are reached before the center during the partial Dijkstra's algorithm. The time complexity is $O(\sum_v |B(v)| \log |B(v)|)$ on a constant-degree graph. [DMSY23] selects the centers at random, and this is the only part we need to derandomize.

Bundle Dijkstra. In this stage, given R and all $B(v)$, the bundle Dijkstra algorithm ([DMSY23] Section 3.2) can solve the single-source shortest path problem in $O(|R| \log n + \sum_v |B(v)|)$ time deterministically.

In total, the time complexity is $O(|R| \log n + \sum_v |B(v)| \log |B(v)|)$. When the centers are selected uniformly at random, we have $\mathbf{E}[|B(v)| \log |B(v)|] = (m/|R|) \log(m/|R|)$. Setting $|R| = m\sqrt{\log\log n}/\sqrt{\log n}$ gives us the time complexity $O(m\sqrt{\log n \log\log n})$.

4.2 Derandomization

The bundle construction can be modeled by our hitting-growable-balls problem in Section 2. Each $B(v)$ corresponds to one ball, which is initially empty. When Algorithm 1 asks to grow $B(v)$, we take one more step in the partial Dijkstra from v, which will include one more vertex in $B(v)$. In the end, Algorithm 1 selects r centers that hit all the balls $B(v)$, which means each partial Dijkstra has reached a center.

The total time complexity for growing the balls is $O(\sum_v |B(v)| \log |B(v)|)$, the same as the original bundle construction. So we let the cost function be $f(x) = x \log x$. By Corollary 5, we have $\sum_v |B(v)| \log |B(v)| = O(m(n/r) \log(n/r))$. Replacing the randomized bundle construction in [DMSY23] by this deterministic bundle construction and setting $r = m\sqrt{\log\log n}/\sqrt{\log n}$, we get a deterministic algorithm with the same time complexity $O(m\sqrt{\log n \log\log n})$, proving Theorem 2.

5 Derandomizing Approximate Distance Oracles

5.1 Thorup-Zwick Oracle

An approximate distance oracle is a data structure which is able to approximately answer any pairwise distance query. If the estimated distance $\hat{d}(u,v)$ returned by the oracle always satisfies $d(u,v) \le \hat{d}(u,v) \le k \cdot d(u,v)$, we say that the oracle is of stretch k.

The Thorup-Zwick approximate distance oracle [TZ05] works as follows. Let $A_0, A_1, \ldots, A_k$ be sets of vertices such that $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$. Consider each level $i \in [0, k-1]$. For each vertex v, let $B_i(v)$ be the set of vertices in A_i which are closer to v than any vertex in A_{i+1}, i.e. $B_i(v) = \{u \in A_i \mid d(v,u) < d(v, A_{i+1})\}$[7]. The Thorup-Zwick oracle stores all $B_i(v)$ and has size $O(\sum_{i=0}^{k-1} \sum_v |B_i(v)|)$. [TZ05] show that it can answer each query of stretch $2k-1$ in $O(k)$ time.

Since the query part is already deterministic, we focus on the preprocessing time and the size. [TZ05] let A_{i+1} contain each element in A_i with probability $n^{-1/k}$, so that each $B_i(v)$ has an expected size of $O(n^{1/k})$. Therefore the size of the data structure is $O(kn^{1+1/k})$. In each level, to compute all $B_i(v)$, they run a partial Dijkstra's algorithm from each vertex in A_{i+1}, so that only the vertices in each $B_i(v)$ are visited. The preprocessing time is therefore $O(kn^{1+1/k}(m/n + \log n)) = O(kn^{1/k}(m + n \log n))$. However, this method requires us to know the entire A_{i+1} before computing the balls, so it does not work well with our derandomization method.

5.2 Derandomization

Consider each level $i \in [0, k-2]$. Suppose that $A_0, \ldots, A_i$ have been determined and we are going to select $n^{-1/k}|A_i|$ vertices from A_i as A_{i+1}. This can be modeled by our hitting-growable-balls problem in Section 2. Each $B_i(v)$ corresponds to one ball, which is initally empty. However, now the set of vertices (in this hitting-growable-balls problem) is A_i instead of the vertex set V of the graph, which means it is non-trivial to grow a ball quickly.

In fact, the previous derandomization [RTZ05] faced the same problem. They proposed an algorithm, which, from our point of view, can grow all the balls (including those that have been hit) together in the desired time.

Let $S_{i,j}(v)$ denote the (ordered) set of j vertices in A_i that are closest to v. Ties are broken by some identical total order of vertices, e.g. by the indices. $S_{i,1}(v)$ for all v can be easily computed in $O(m + n \log n)$ time. Then the following algorithm can compute all $S_{i,j+1}(v)$ from all $S_{i,j}(v)$.

[7] In particular, we define $d(v, A_k) = d(v, \emptyset) = \infty$.

Algorithm 2 (restate from the proof of [RTZ05] Theorem 2)

Input: undirected graph $G = (V, E)$, step j, sets $S_{i,j}(v)$ for all $v \in V$.
Output: sets $S_{i,j+1}(v)$ for all $v \in V$.

1: Add a new source vertex s to G.
2: **for** each edge $(u, v) \in E$ in each direction **do**
3: **if** $S_{i,j}(u) = S_{i,j}(v)$ **then**
4: Keep the edge (u, v).
5: **else**
6: Let w denote the first vertex in $S_{i,j}(u) \setminus S_{i,j}(v)$.
7: Replace the edge (u, v) by an edge (s, v) of weight $d(w, u) + l(u, v)$ and label w.
8: Run Dijkstra's algorithm from s.
9: **for** each vertex v **do**
10: $S_{i,j+1}(v) \leftarrow S_{i,j} \cup \{$the label of the first edge of the shortest path to $v\}$.
11: **return** all $S_{i,j+1}(v)$.

Lemma 6 ([RTZ05] Theorem 2)**.** *For any $j \geq 1$, Algorithm 2 computes all $S_{i,j+1}(v)$ from all $S_{i,j}(v)$ in $O(m + n \log n)$ time by performing a single-source shortest paths computation on a graph with $O(n)$ vertices and $O(m)$ edges.*

If all the balls have the same size and we want to grow all of them by 1, we can call Algorithm 2. By Lemma 6, the amortized time for each single grow operation will be $O(m/n + \log n)$, which matches with the randomized algorithm. However, in Algorithm 1, we only want to grow all unhit balls. Nevertheless, by observing some key properties, we can "partially" run Algorithm 2 to achieve it.

Lemma 7. *When running Algorithm 1 for our current problem of selecting A_{i+1} from A_i, each grow operation can be done in $O(m/n + \log n)$ amortized time.*

Proof. Note that during Algorithm 1, all the unhit balls have the same size, and we always grow them together. Suppose that there are t unhit balls now, and they all have size j, and we want to grow all of them by 1. Then we know $B_i(v) = S_{i,j}(v)$ for all unhit $B_i(v)$. Let's try to run Algorithm 2. Let G' denote the graph constructed by Algorithm 2 (which we won't actually construct).

Notice that, if $B_i(u)$ is hit and $B_i(v)$ is unhit, then $S_{i,j}(u) \neq S_{i,j}(v)$ since $S_{i,j}(u)$ contains a center but $S_{i,j}(v)$ does not. So in G', there are no edges between the vertices whose balls are hit and the vertices whose balls are unhit. Therefore, we can safely discard all vertices whose balls are hit. We only need to construct the partial graph for the t vertices by examining edges incident to at least one of them.

Consider each such edge (u, v). If both $B_i(u)$ and $B_i(v)$ are unhit, then we just follow Algorithm 2. If $B_i(u)$ is hit and $B_i(v)$ is unhit, then we don't actually know $S_{i,j}(u)$. We only know $B_i(u) = S_{i,j'}(u)$ for some $j' \leq j$. However, since $S_{i,j'}(u)$ will contain (at least) one center, which must not be in $S_{i,j}(v)$, we could still get the correct w in line 6.

Since each vertex has degree $O(m/n)$, we only need to run Dijkstra's algorithm on a graph with $t + 1$ vertices and $O(tm/n)$ edges, which can be done in $O(t(m/n + \log n))$ time as desired. □

Now, let's put everything together. We run Algorithm 1 with $r = n^{-1/k}|A_i|$ and $p = 1$. By Theorem 4, we have $|A_{i+1}| \leq n^{-1/k}|A_i|$ and $\sum_v |B_i(v)| = O(n^{1+1/k})$. By Lemma 7, the time we spend in this level is $O(n^{1+1/k}(m/n + \log n)) = O(n^{1/k}(m + n \log n))$. Note that in the last level

$k-1$, since $A_k = \emptyset$ is fixed, we have $B_{k-1}(v) = A_{k-1}$. Since $|A_{k-1}| \leq (n^{-1/k})^{k-1}|A_0| = n^{1/k}$, the above properties also hold for $i = k-1$. Combining k levels together, we construct the Thorup-Zwick oracle of size $O(kn^{1+1/k})$ in $O(kn^{1/k}(m + n \log n))$ deterministic time as desired, proving Theorem 3.

Acknowledgments

We thank Mikkel Thorup and the anonymous reviewers for helpful comments.

References

[ABCP98] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM J. Comput.*, 28(1):263–277, 1998.

[ADD+93] Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discret. Comput. Geom.*, 9:81–100, 1993.

[AG13] Rachit Agarwal and Philip Brighten Godfrey. Distance oracles for stretch less than 2. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 526–538. SIAM, 2013.

[Aga14] Rachit Agarwal. The space-stretch-time tradeoff in distance oracles. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 49–60. Springer, 2014.

[BGSU08] Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 609–621. Springer, 2008.

[BK06] Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006, Berkeley, California, USA, October 21-24, 2006, Proceedings*, pages 591–602. IEEE Computer Society, 2006.

[BK10] Surender Baswana and Telikepalli Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM J. Comput.*, 39(7):2865–2896, 2010.

[BS06] Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Trans. Algorithms*, 2(4):557–577, 2006.

[Che14] Shiri Chechik. Approximate distance oracles with constant query time. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 654–663. ACM, 2014.

[Che15] Shiri Chechik. Approximate distance oracles with improved bounds. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 1–10. ACM, 2015.

[Coh98] Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28(1):210–236, 1998.

[CZ22] Shiri Chechik and Tianyi Zhang. Nearly 2-approximate distance oracles in subquadratic time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 551–580. SIAM, 2022.

[Dij59] EW Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[DMM+25] Ran Duan, Jiayi Mao, Xiao Mao, Xinkai Shu, and Longhui Yin. Breaking the sorting barrier for directed single-source shortest paths. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 36–44. ACM, 2025.

[DMSY23] Ran Duan, Jiayi Mao, Xinkai Shu, and Longhui Yin. A randomized algorithm for single-source shortest path on undirected real-weighted graphs. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 484–492. IEEE, 2023.

[Erd64] Paul Erdös. Extremal problems in graph theory. *Publ. House Cszechoslovak Acad. Sci., Prague*, pages 29–36, 1964.

[FT87] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[FW93] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

[FW94] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.

[Hag00] Torben Hagerup. Improved shortest paths on the word RAM. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, volume 1853 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2000.

[KKR24] Tsvi Kopelowitz, Ariel Korin, and Liam Roditty. On the space usage of approximate distance oracles with sub-2 stretch. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 101:1–101:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[Mat96] Jiří Matoušek. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics*, 93(1):333–344, 1996.

[MN06] Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. In *47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006, Berkeley, California, USA, October 21-24, 2006, Proceedings*, pages 109–118. IEEE Computer Society, 2006.

[MS09] Manor Mendel and Chaya Schwob. Fast C-K-R partitions of sparse graphs. *Chic. J. Theor. Comput. Sci.*, 2009, 2009.

[PR14] Mihai Pătraşcu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014.

[PRT12] Mihai Pătraşcu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 738–747. IEEE Computer Society, 2012.

[Ram96] Rajeev Raman. Priority queues: Small, monotone and trans-dichotomous. In Josep Díaz and Maria J. Serna, editors, *Algorithms - ESA '96, Fourth Annual European Symposium, Barcelona, Spain, September 25-27, 1996, Proceedings*, volume 1136 of *Lecture Notes in Computer Science*, pages 121–137. Springer, 1996.

[Ram97] Rajeev Raman. Recent results on the single-source shortest paths problem. *SIGACT News*, 28(2):81–87, 1997.

[RT23] Liam Roditty and Roei Tov. Approximate distance oracles with improved stretch for sparse graphs. *Theoretical Computer Science*, 943:89–101, 2023.

[RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *International Colloquium on Automata, Languages, and Programming*, pages 261–272. Springer, 2005.

[Tho99] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999.

[Tho00a] Mikkel Thorup. Floats, integers, and single source shortest paths. *J. Algorithms*, 35(2):189–201, 2000.

[Tho00b] Mikkel Thorup. On RAM priority queues. *SIAM J. Comput.*, 30(1):86–109, 2000.

[Tho04] Mikkel Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. *J. Comput. Syst. Sci.*, 69(3):330–353, 2004.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.

[Wul12] Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 202–208. SIAM, 2012.

[Wul13] Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 539–549. SIAM, 2013.