Distance Oracles for Vertex-Labeled Graphs

Danny Hermelin[1], Avivit Levy[2], Oren Weimann[3], and Raphael Yuster[4]

[1] Max-Planck Institut fur informatik, `hermelin@mpi-inf.mpg.de`
[2] Shenkar College and CRI at University of Haifa, `avivitlevy@shenkar.ac.il`
[3] Weizmann Institute, `oren.weimann@weizmann.ac.il`
[4] University of Haifa, `raphy@math.haifa.ac.il`

Abstract. Given a graph $G = (V, E)$ with non-negative edge lengths whose vertices are assigned a *label* from $L = \{\lambda_1, \ldots, \lambda_\ell\}$, we construct a compact *distance oracle* that answers queries of the form: "What is $\delta(v, \lambda)$?", where $v \in V$ is a vertex in the graph, $\lambda \in L$ a vertex label, and $\delta(v, \lambda)$ is the distance (length of a shortest path) between v and the closest vertex labeled λ in G. We formalize this natural problem and provide a hierarchy of *approximate distance oracles* that require subquadratic space and return a distance of constant stretch. We also extend our solution to dynamic oracles that handle label changes in sublinear time.

1 Introduction

In this paper we consider an *all-pairs shortest paths* variant on vertex-labeled graphs: We are given an undirected graph $G = (V, E)$ with $m = |E|$ edges and $n = |V|$ vertices. Each edge is assigned a nonnegative length and each vertex is assigned a *label*, given as a function $\lambda : V \to L$, where $L = \{\lambda_1, \ldots, \lambda_\ell\}$ is a set of $\ell \leq n$ distinct labels. The goal is to preprocess G in order to answer *vertex-label distance queries*, *i.e.* queries of the form: "What is $\delta(v, \lambda)$?", where $v \in V$ is a vertex in the graph, $\lambda \in L$ a label, and $\delta(v, \lambda)$ is the distance (length of a shortest path) between v and the closest vertex labeled λ in G.

Vertex-labeled graphs are commonly used to analyze behaviors and structures of networks. Typically, labels specify a common functionality of a set of nodes in the network. Often in such networks we are not interested in shortest paths between two specific vertices, but rather from a specific vertex to any vertex that can provide a specific function. For instance, in a computer network labels might indicate different types of servers in the network, while in an automotive network they can indicate different services provided for drivers on the road. Common type of queries in these types of networks include queries such as "What is the closest FTP server?" or "Where is the closest McDonald's?"

A simple solution for the above problem is to construct a table indexed by vertex-label pairs, storing at entry (v, λ) the distance $\delta(v, \lambda)$, thus allowing queries to be answered in $O(1)$ time. Such a table can be constructed with ℓ calls to a standard single-source shortest-paths algorithm such as Dijkstra's Algorithm. This is done by constructing for each label $\lambda \in L$, an auxiliary graph G_λ, created by removing all vertices labeled λ from G and adding a new vertex

v_λ which is adjacent to all neighbors of the removed vertices (using appropriate minimum edge-lengths). Running Dijkstra's Algorithm in G_λ from v_λ gives all distances in G to vertices labeled λ. The total running time of this construction is $O(m\ell)$ and the space required for the table is $O(n\ell)$.

There are two problems with this solution. First, the $O(n\ell)$ space requirement is unacceptable in many cases, especially when the number of labels ℓ in the graph is quite large. It is therefore beneficial in these cases to replace this solution by a more compact data structure, perhaps at the cost of providing only approximate distances. Such data structures have been coined *approximate distance oracles* in the literature. The second problem with the data structure is its inability so support efficient changes of labels. A change of a label can reflect a change in its functionality or in availability of services in the network. Indeed, even a single label change may incur $\Omega(n)$ changes in the data structure.

Distance oracles for unlabeled graphs have been studied quite extensively. A seminal result of Thorup and Zwick [9] achieves for any given integer $k \geq 2$, an $O(kn^{1+1/k})$ space oracle which returns distances with at most $(2k-1)$ *stretch* in $O(k)$ time. That is, given an input pair of vertices (u, v), the oracle outputs a distance $\mathbf{dist}(u, v)$ with $\delta(u, v) \leq \mathbf{dist}(u, v) \leq (2k-1) \cdot \delta(u, v)$, where $\delta(u, v)$ is the actual distance between u and v. An earlier observation by Matoušek [4] shows that this is essentially optimal for any k, assuming a conjecture of Erdős [3] concerning the girth of undirected graphs. A derandomization of the oracle was presented in [6], and faster construction times where presented in [2] and [1] with a slight increase of the promised stretch. Recently, Pătraşcu and Roditty [5] showed how to construct, for unweighted graphs, an oracle of size $O(n^{5/3})$ that, when queried about a pair of vertices (u, v), returns in $O(1)$-time a distance $\mathbf{dist}(u, v)$ bounded by $\delta(u, v) \leq \mathbf{dist}(u, v) \leq 2\delta(u, v) + 1$. For further results see also [5, 9].

Applying the existing approximate distance oracles to labeled graphs faces several obstacles. First of all, one might be tempted to construct an oracle for the complete bipartite graph $B(G)$ formed on the vertex sets V and L, where the length of the edge $\{v, \lambda\}$ is set to $\delta(v, \lambda)$. However, the oracle constructed for this graph might output false distances, as paths in $B(G)$ using an intermediate vertex $\lambda \in L$ do not occur in G. Similarly, the solution that adds a new vertex for each label, connecting it with zero weight edges to all vertices with the same label, fails. Another option is to build ℓ different oracles, one per each label. This unfortunately yields a solution that requires more space than the naive $O(n\ell)$ solution described above, even when using the optimal construction of Thorup and Zwick. Thus, "black-box" solutions such as these are destined for failure.

The next obvious approach is to hack and modify the existing oracles, adapting them to support vertex-label queries. However, this has difficulties as well. All known oracles achieving close to optimal stretch are highly dependent on knowing both the source and the target vertex. For instance, Thorup and Zwick's $(2k-1)$-stretch oracles find the path between the pair of query vertices by advancing from both of them simultaneously. This obviously cannot be applied to vertex-label queries, as we only know the identity of the source vertex in our query. There is, however, another oracle scheme by Thorup and Zwick [8] designed for routing that does not switch between query vertices. Oracles produced by this scheme require $O(kn^{1+1/k})$ space, and answer queries with $4k-5$ stretch in $O(k)$ time. We show in Section 2 how to adapt this scheme to vertex-label oracles achieving the same performance.

Theorem 1 (Thorup-Zwick Vertex-Label Distance Oracles). *A vertex-label distance oracle of expected size $O(kn^{1+1/k})$ with stretch $(4k-5)$ and query time $O(k)$ can be constructed in $O(kmn^{1/k})$ time.*

There are two problems with the oracles given by Theorem 1. First, the expected space of the oracles depends only on the number of vertices in the graph, and not on the number of labels. This might be too much when ℓ is significantly smaller than n. For instance, when say $\ell = O(\sqrt{n})$, the trivial $O(n\ell)$ solution discussed above gives an $O(n^{3/2})$-space oracle which produces exact distances, as opposed to the stretch-3 distances returned by the oracle of Theorem 1. This problem becomes even more apparent when $\ell = \text{polylog}(n)$. The second problem is that the oracles given in Theorem 1 cannot be adapted to support dynamic labels efficiently. Indeed, even a single label change requires in the worst case reconstruction of almost the entire oracle from scratch.

In Section 3, we address the first issue raised above, and give a first step towards the second issue as well. We show how to construct an alternative scheme of vertex-label distance oracles which have expected space bounds depending on both ℓ and n, thereby achieving better bounds than the scheme of Theorem 1. While the stretch of this scheme grows exponentially in k, we are able to obtain the *same* stretch as in Theorem 1 for the cases of $k = 2$ and $k = 3$. Furthermore, for the case of $\ell = \text{polylog}(n)$, our scheme gives an $O(n \lg n)$-space oracle with constant stretch, while Theorem 1 cannot achieve even poly-logarithmic stretch within comparable space bounds.

Theorem 2 (Compact Vertex-Label Distance Oracles). *A vertex-label distance oracle of expected size $O(kn\ell^{1/k})$ with stretch (2^k-1) and query time $O(k)$ can be constructed in $O(kmn^{k/(2k-1)})$ time.*

The oracle schemes given by both theorems above still do not support label changes. Nevertheless, in Section 4 we show how the scheme of Theorem 2 can be modified in a way that allows oracles supporting such updates in sublinear time. This results in oracles with size asymptotically the same as the ones given by Theorem 1, and with stretch slightly bigger then the stretch of the oracles given in Theorem 2.

Theorem 3 (Dynamic Vertex-Label Distance Oracles). *A vertex-label distance oracle of expected size $O(kn^{1+1/k})$ with stretch $(2\cdot 3^{k-1}+1)$ can support label changes in $O(kn^{1/k}\lg n)$ time and queries in $O(k)$ time.*

We also remark that it is possible to combine our construction with the ideas of Pătraşcu and Roditty [5] to obtain a stretch-3 oracle of expected size $O(n^{5/3})$, supporting label changes in in $O(n^{2/3}\lg n)$ time (see Section 4).

2 Adaptation of Thorup-Zwick Oracles

In this section we outline a simple adaptation of Thorup and Zwick's [9] result that supports vertex-label distance queries, thus providing a proof for Theorem 1. Our adaptation is based on an alternative query algorithm given by Thorup and Zwick in [8]. We will use the following notation throughout the section, and the remainder of the paper. For a pair of vertices u and v, we let $\delta(u, v)$ denote the distance between u to v, where $\delta(u, v) = \infty$ if there is no path connecting them. For a non-empty vertex-subset $S \subseteq V$, we let $\delta(v, S) := \min_{u \in S} \delta(u, v)$, and we set $\delta(v, \emptyset) := \infty$. For a label $\lambda \in L$, we denote by $\delta(v, \lambda)$ the distance $\delta(v, V_\lambda)$, where $V_\lambda := \{v \in V : \lambda(v) = \lambda\}$.

The data structure. For a given positive integer k, the preprocessing algorithm of Thorup and Zwick constructs the sets $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k-1} \supseteq A_k = \emptyset$. These sets are referred to as the levels of G, and a vertex v is said to be in *level* i if $v \in A_i$. The i-th level A_i is constructed by sampling vertices of A_{i-1} independently at random, taking a vertex $v \in A_{i-1}$ to A_i with probability $n^{-1/k}$. For $i \in \{1, \ldots, k-1\}$, the *i-th pivot* of a vertex $v \in V$, denoted $p_i(v)$, is defined to be the vertex closest to v in the i-th level of G. That is, $\delta(v, p_i(v)) = \delta(v, A_i)$. We also set $p_0(v) := v$. The oracle stores for each vertex v, the identity of each of its k pivots, along with the k distances $\delta(v, p_i(v))$. In addition, v stores the distances to all vertices in its *bunch* $B(v)$, where

$$B(v) := \bigcup_{i=0}^{k-1} \big\{u \in A_i \setminus A_{i+1} : \delta(v, u) < \delta(v, A_{i+1})\big\}.$$

We next describe the additional information that our adaptation requires. For a label $\lambda \in L$, define the bunch $B(\lambda)$ by $B(\lambda) := \bigcup_{v \in V_\lambda} B(v)$. Now for every vertex $u \in B(\lambda)$, we define u_λ to be the λ-labeled vertex closest to u and that satisfies $u \in B(u_\lambda)$. For each $\lambda \in L$ and $u \in B(\lambda)$, we store the distance $\delta(u, u_\lambda)$. Note that each one of these distances is computed by the original Thorup-Zwick construction. Furthermore, the additional space required by our construction does not change the asymptotic size of the original oracles, since

$$\sum_{\lambda \in L} |B(\lambda)| = \sum_{\lambda \in L} \Big| \bigcup_{v \in V_\lambda} B(v) \Big| \le \sum_{\lambda \in L} \sum_{v \in V_\lambda} |B(v)| = \sum_{v \in V} |B(v)|.$$

Adapted query algorithm. Our adapted query algorithm examines each of the k vertices $p_i(v)$ for $0 \le i < k$, starting with $p_0(v) := v$. For each such vertex $w := p_i(v)$, we check if $w \in B(\lambda)$. If so, we declare i to be a *valid* index and we compute the distance $\delta(v, w) + \delta(w, w_\lambda)$ in $O(1)$ time (via hash tables). After $\Theta(k)$ time, our query algorithm returns the distance

$$\mathbf{dist}(v, \lambda) := \min\{\delta(v, w) + \delta(w, w_\lambda) : w = p_i(v) \text{ and } i \text{ is valid }\}.$$

The important difference between our query algorithm and that of [8] is that we need to check *all* valid indices while [8] can stop when it reaches the first valid index. In other words, the query algorithm of [8] upon query (v, u) returns $\mathbf{dist}(v, u) = \delta(v, w) + \delta(w, u)$ where $w = p_i(v)$ for the smallest i such that $w = p_i(v) \in B(u)$. They show (Lemma A.1 in [8]) that the stretch is then bounded by $\mathbf{dist}(v, u) \leq (4k - 3) \cdot \delta(v, u)$. In our settings, let u be the (unknown) λ-labeled vertex such that $\delta(v, u) = \delta(v, \lambda)$. When we discover the first $w = p_i(v) \in B(\lambda)$, we cannot promise that $\delta(w, w_\lambda) \leq \delta(w, u)$ since maybe $w \notin B(u)$. We can however guarantee, from the definition of $B(\lambda)$, that for *some* i we will have $w = p_i(v) \in B(u)$. We therefore have that $\mathbf{dist}(v, \lambda) \leq (4k - 3) \cdot \delta(v, \lambda)$. Finally, the same argument used in [8] (Lemma A.2) shows how the stretch bound can be reduced from $(4k - 3)$ to $(4k - 5)$, thus proving Theorem 1.

3 More Compact Oracles

In what follows we describe our vertex-label distance oracles for vertex-labeled graphs. In particular, we provide a complete proof of Theorem 2.

3.1 The Data Structure

The first step in constructing our $(2^k - 1)$-stretch oracle is similar to the Thorup-Zwick adaptation of Section 2. For a given positive integer k, we first construct the sets $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k-1} \supseteq A_k = \emptyset$ which will form the levels of G. However, unlike the construction of Section 2, we select the vertices into levels with probability depending on ℓ. That is, the i-th level A_i is constructed by sampling vertices of A_{i-1} independently at random, taking a vertex $v \in A_{i-1}$ to A_i with probability $\ell^{-1/k}$. Thus, for any $v \in V$, the probability that $v \in A_i$ for some $i \in \{0, \ldots, k-1\}$, is exactly $\ell^{-i/k}$. The following bound on the expected size of A_i follows immediately:

Lemma 1. *$\mathbb{E}[|A_i|] = n\ell^{-i/k}$ for each $i \in \{0, \ldots, k-1\}$.*

The idea of sampling vertices with probability independent of n was already suggested by Roditty, Thorup, and Zwick in [6]. The problem they considered was a distance oracle that answers $\delta(u, v)$ queries where u can be any vertex but v is known to belong to some subset $S \subset V$. For this problem, they showed that if we sample vertices with probability $|S|^{-1/k}$ then the original Thorup-Zwick oracle works and requires only $O(n|S|^{-1/k})$ space. For our problem however, the original Thorup-Zwick oracle can not be made to work if we sample with probability $\ell^{-1/k}$. This is because we are not dealing with a set of vertices but rather with a set of sets (the set of labels where each label is a set of vertices). We now show how to overcome this by presenting a different oracle that in particular uses *balls* instead of *bunches*.

For a vertex $v \in A_i \setminus A_{i+1}$, we define the *ball of* v to be the set of labels $B(v)$ of *all* vertices in G that are closer to v than A_{i+1}. That is, $B(v) = \{\lambda(u) : \delta(u, v) < \delta(v, A_{i+1})\}$. Notice the difference between our *balls* and Thorup-Zwick's *bunches*. Each vertex stores all the labels in its ball in an appropriate hash table which allows us to determine in $O(1)$ time whether $\lambda \in B(v)$, for any label $\lambda \in L$.

Furthermore, we store the distances $\delta(v, \lambda)$ to each label $\lambda \in B(v)$, allowing $O(1)$ answers to vertex-label queries for labels appearing inside the ball of the query vertex. Note that as $A_k = \emptyset$, we have $\delta(v, A_k) = \infty$ for all $v \in V$, and so $B(v) = L$ for any vertex $v \in A_{k-1}$. Nevertheless, the following lemma shows that the expected number of labels is not big at vertices appearing in lower levels.

Lemma 2. $\mathbb{E}[|B(v)|] \le \ell^{(i+1)/k}$ *for any vertex* $v \in A_i \setminus A_{i+1}$, $i \in \{0, \ldots, k-1\}$.

Proof. The lemma is clearly true for $i = k-1$. Let $i \in \{0, \ldots, k-2\}$, and let $v \in A_i \setminus A_{i+1}$. Consider the vertices of G sorted by non-decreasing distance from v, breaking ties arbitrarily. For each label $\lambda \in L$, keep in this list the first vertex labeled λ. An upper bound on $|B(v)|$ is the location of the first element from A_{i+1} in the list. Therefore, the size of $B(v)$ is bounded from above by a geometric random variable with rate $\ell^{-(i+1)/k}$. Hence, the expected size of $B(v)$ is at most $\ell^{(i+1)/k}$. □

To complete the description of our distance oracle, we assign routers to the vertices in our graph. For a vertex $v \in A_i \setminus A_{i+1}$, $i \in \{0, \ldots, k-2\}$, we let the *router of* v, denoted $r(v)$, be a vertex for which $\delta(v, r(v)) = \delta(v, A_{i+1})$. That is, $r(v)$ is the closest vertex to v at the next level of G. Since vertices at level $k-1$ have no vertices at the next level, we set $r(v) = v$ for all $v \in A_{k-1}$. Along with the ball of labels $B(v)$ stored at each vertex $v \in V$, our distance oracle also stores at v the identity of its router $r(v)$, together with the distance $\delta(v, r(v))$. Thus, the total space required by our data structure is (asymptotically) the total sizes of the balls $B(v)$, which can easily be bounded using Lemma 1 and Lemma 2.

Lemma 3. *The expected space of our data structure is* $O(kn\ell^{1/k})$.

Proof. According to the above, to prove the lemma it suffices to bound the total expected size of all balls $B(v)$, $v \in V$. Since the vertices of G are partitioned into the levels $A_i \setminus A_{i+1}$, we can write this expected total size as:

$$\begin{aligned}\mathbb{E}\Big[\sum_v |B(v)|\Big] &= \sum_i \sum_{v \in A_i \setminus A_{i+1}} \mathbb{E}[|B(v)|] \\ &\le \sum_i \sum_{v \in A_i \setminus A_{i+1}} \ell^{(i+1)/k} < \sum_i |A_i| \cdot \ell^{(i+1)/k} \\ &\le \sum_i n\ell^{-i/k} \cdot \ell^{(i+1)/k} = kn\ell^{1/k}.\end{aligned}$$

Here, the first inequality comes from Lemma 2 and the last inequality comes from Lemma 1. □

3.2 Vertex-Label Queries

We next proceed to describe how a vertex-label query is processed. Let $(v \in V, \lambda \in L)$ denote an input vertex-label pair. The query algorithm starts by determining whether λ is in the ball of v. If so, the exact distance $\delta(v, \lambda)$ is retrieved immediately. Otherwise, it hops to the router of v, and continues the search there. If $\lambda \notin B(r(v))$, the algorithm hops to the router of $r(v)$, and so forth.

To be more precise, let us introduce the following notation: For $i \in \{0, \ldots, k-1\}$, we let r_i denote the vertex $r^{(i)}(v)$, where $r^{(i)}$ is the function resulting in concatenating r with itself i times. That is, $r_i = r(r_{i-1})$ for $i \in \{1, \ldots, k-1\}$, and $r_0 = v$. (Again, notice the difference between these routers and the pivots of Section 2.) The query algorithm determines the smallest integer $i_0 \in \{0, \ldots, k-1\}$ for which $\lambda \in B(r_{i_0})$, and returns the distance

$$\mathbf{dist}(v, \lambda) := \sum_{0 \le i < i_0} \delta(r_i, r_{i+1}) + \delta(r_{i_0}, \lambda).$$

See Figure 1 for an example. Note that as $r_{k-1} \in A_{k-1}$, we have $\lambda \in B(r_{k-1})$, and so i_0 is well-defined. Furthermore, the path from v to r_{i_0}, and then from r_{i_0} to a vertex labeled λ, gives a path from v to a vertex labeled λ as required.

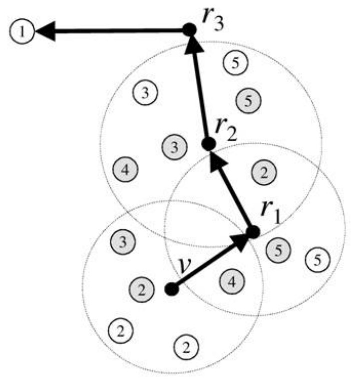

Fig. 1. An example of the query procedure for $k = 4$. The input to the query is $(v, 1)$, and the arrows depict the output path. The dashed circles around v, r_1, r_2, and r_3 represent the balls around them. The gray colored vertices are the vertices which are stored at each ball (the distances are assumed to be Euclidean).

Determining the smallest integer i_0 for which $\lambda \in B(r_{i_0})$ takes $O(k)$ time, by iteratively hopping through the r_i's. Furthermore, as the distances $\delta(r_i, r_{i+1})$, for $i \in \{0, \ldots, i_0 - 1\}$, have been stored by our data-structure, along with the distance $\delta(r_{i_0}, \lambda)$, we can report the resulting distance $\mathbf{dist}(v, \lambda)$ in $O(k)$ time as well. This gives us the promised $O(k)$ query time of Theorem 2. The next lemma shows that the stretch of our query is also as stated in Theorem 2:

Lemma 4. $\delta(v, \lambda) \leq \mathbf{dist}(v, \lambda) \leq (2^k - 1) \cdot \delta(v, \lambda)$ *for all* $(v, \lambda) \in V \times L$.

Proof. Let $(v, \lambda) \in V \times L$, and let $i_0 \in \{0, \ldots, k-1\}$ denote the smallest integer for which $\lambda \in B(r_{i_0})$. Then $\mathbf{dist}(v, \lambda) := \sum_{0 \leq i < i_0} \delta(r_i, r_{i+1}) + \delta(r_{i_0}, \lambda)$, where $r_0 := v$ and $r_i := r(r_{i-1})$ for all $i \in \{1, \ldots, k-1\}$. The lower bound $\delta(v, \lambda) \leq \mathbf{dist}(v, \lambda)$ in the lemma follows from the fact that $\mathbf{dist}(v, \lambda)$ is the length of an actual path from v to a vertex labeled λ. The proof of the upper bound relies on the following crucial inequality which follows from our definition of the balls $B(v)$, and from the fact that $\lambda \notin B(r_i)$ for all $i \in \{0, \ldots, i_0 - 1\}$:

$$\delta(r_i, r_{i+1}) \leq \delta(r_i, \lambda) \text{ for all } i \in \{0, \ldots, i_0 - 1\}. \tag{1}$$

As an intermediate step in proving the upper bound, we use (1) to prove inequality (2) below by induction on i:

$$\delta(r_i, \lambda) \leq 2^i \cdot \delta(v, \lambda) \text{ for all } i \in \{0, \ldots, i_0\}. \tag{2}$$

For $i = 0$, we have $\delta(r_0, \lambda) = \delta(v, \lambda)$ so (2) holds. Assume therefore that $i > 0$, and that (2) holds for all $j < i$. By the triangle-inequality, we get the following bound $\delta(r_i, \lambda) \leq \sum_{0 \leq j < i} \delta(r_j, r_{j+1}) + \delta(v, \lambda)$. Thus, by (1) and our inductive hypothesis we have:

$$\begin{aligned} \delta(r_i, \lambda) &\leq \textstyle\sum_{0 \leq j < i} \delta(r_j, r_{j+1}) + \delta(v, \lambda) \\ &\leq \textstyle\sum_{0 \leq j < i} \delta(r_j, \lambda) + \delta(v, \lambda) \\ &\leq \textstyle\sum_{0 \leq j < i} 2^j \cdot \delta(v, \lambda) + \delta(v, \lambda) \\ &= 2^i \cdot \delta(v, \lambda). \end{aligned}$$

Inequality (2) therefore holds. Now, using (1) and (2) the upper bound easily follows as

$$\begin{aligned} \mathbf{dist}(v, \lambda) &= \textstyle\sum_{0 \leq i < i_0} \delta(r_i, r_{i+1}) + \delta(r_{i_0}, \lambda) \\ &\leq \textstyle\sum_{0 \leq i < i_0} \delta(r_i, \lambda) + \delta(r_{i_0}, \lambda) \\ &\leq \textstyle\sum_{0 \leq i < i_0} 2^i \cdot \delta(v, \lambda) + 2^{i_0} \cdot \delta(v, \lambda) \\ &= (2^{i_0+1} - 1) \cdot \delta(v, \lambda). \end{aligned}$$

This proves the upper bound in the lemma, since $i_0 \leq k - 1$. □

3.3 Construction

The time to construct the data structure is composed of two parts. The first is the time to find for every vertex v the distance from v to all vertices $\widehat{B}(v) = \{u : \delta(u, v) < \delta(v, A_{i+1})\}$. Observe that $B(v)$ can be immediately obtained from $\widehat{B}(v)$. A simple modification of Dijkstra's Algorithm starting from source v computes $\widehat{B}(v)$ by inspecting only $|\widehat{B}(v)|$ vertices. The inspected edges are only edges (u, w) where $u \in \widehat{B}(v)$ *or* $w \in \widehat{B}(v)$. There are at most $n \cdot |\widehat{B}(v)|$ such edges so the total expected time complexity is

$$\sum_{i} \sum_{v \in A_i \setminus A_{i+1}} n \cdot \mathbb{E}[|\widehat{B}(v)|] = kn^2 \ell^{1/k} = O(kn^{2+1/k}).$$

The second (and dominating) term of the construction time is to find $\delta(v, \lambda)$ for every $v \in A_{k-1}$ and every $\lambda \in L$. This can either be done by running (the standard version of) Dijkstra's Algorithm from ℓ sources in total $O(m\ell)$ time, or from $|A_{k-1}|$ sources in total $O(m \cdot |A_{k-1}|)$ time. The value $O(m \cdot \min\{\ell, |A_{k-1}|\})$ is maximized when $\ell = |A_{k-1}| = n/\ell^{1-1/k}$ and amounts to $O(mn^{k/(2k-1)})$. This concludes the proof of Theorem 2.

4 Oracles Supporting Dynamic Labels

In this section we consider the situation where the labels may change dynamically. That is, we want a distance oracle that not only supports the usual vertex-label distance queries, but also supports updates of the form **change**(v, λ), for $v \in V$ and $\lambda \in L$, which sets the label of v to be λ and leaves all labels of vertices in $V \setminus \{v\}$ unchanged. We will show how to modify the oracle scheme of Section 3 so that it supports such queries, at an increase to the stretch and space of the constructed oracles. The main difficulty in achieving this comes from the fact that a vertex may be present in $\Omega(n)$ balls, and thus a change in its label may require updating $\Omega(n)$ hash-tables. The following describes how to overcome this.

4.1 The Data Structure

The construction of our dynamic distance oracle scheme is similar to the construction of Section 3. Below we focus on the main changes. As in Section 3, we first select the sets of vertices $V = A_0 \supseteq A_1 \supseteq \ldots \supseteq A_{k-1} \supseteq A_k = \emptyset$. However, here we select the vertices in the levels with probability depending on n, as in Section 2. That is, we select A_i by sampling vertices of A_{i-1} independently at random with probability $n^{-1/k}$. Thus, the probability that an arbitrary vertex $v \in V$ is in A_i is exactly $n^{-i/k}$, and the expected-size of A_i is $n^{1-i/k}$. Again, the sets A_i are referred to as the *levels* of G, and we designate for each vertex $v \in V$ a router $r(v) \in V$, defined identically as in Section 3.

The main difference between our static and dynamic schemes lies in the definition of balls. Here our balls will be sets of vertices instead of sets of labels. Moreover, we store half-balls rather than balls. For a vertex $v \in A_i \setminus A_{i+1}$, we define the *half-ball* of v as the set of vertices $B^{\frac{1}{2}}(v) := \{u : \delta(u, v) < \frac{1}{2} \cdot \delta(v, r(v))\}$. That is, a vertex u is in $B^{\frac{1}{2}}(v)$ if it is closer to v than half the distance from v to its router. We also define the cluster $C(v)$ of a vertex $v \in V$ to be the set of all vertices for which v is in their half-ball. That is, $C(v) := \{u : v \in B^{\frac{1}{2}}(u)\}$.

Let us next describe the exact information stored for each vertex $v \in V$. First, we store the cluster $C(v)$ of v, along with all distances $\delta(v,u)$ for $u \in C(v)$. Second, we store all distances to vertices u in the half-ball $B^{\frac{1}{2}}(v)$ of v. The distances to vertices in the half-ball are organized into heaps, one per each label appearing in $B^{\frac{1}{2}}(v)$, that support the following three generic operations:

– **insert**(δ): insert a new distance δ into the heap.
– **remove**(δ): remove an existing distance δ from the heap.
– **minimum**(): return the minimum distance in the heap.

A standard construction of a heap supports the first two operations above in $O(\lg n)$ time (even $O(\lg \lg n)$ time if the edge lengths are integral [10]), while **minimum**() requires constant time. Apart from the cluster $C(v)$, the heaps, the router $r(v)$ and the distance $\delta(v, r(v))$, we also store a hash table at v which allows us to determine in $O(1)$ time whether there exists a vertex labeled λ in $B^{\frac{1}{2}}(v)$, given any label $\lambda \in L$. In Section 4.2 below we show that the expected size of $C(v)$ is $O(kn^{1/k})$ for every vertex $v \in V$. All other information stored at v is asymptotically at most $|B^{\frac{1}{2}}(v)|$, which by similar arguments as those used in Section 3, can also be bounded by $O(kn^{1/k})$ in expectation. Thus, the total size of our data structure can be bounded as in the lemma below.

Lemma 5. *The expected size of our data structure is* $O(kn^{1+1/k})$.

4.2 Label Changes

We next turn to describe how our oracle supports updates of the form **change**(v, λ). The key idea is to use the information stored at the cluster $C(v)$ of v. We begin by bounding the size of $C(v)$ in expectation.

Lemma 6. $\mathbb{E}[|C(v)|] \leq kn^{1/k}$ *for any vertex* $v \in V$.

Proof. For $i \in \{0, \ldots, k-1\}$, let $C_i(v)$ denote the set of vertices $u \in A_i \setminus A_{i+1}$ for which $v \in B^{\frac{1}{2}}(u)$. To prove the lemma, we show that the expected size of $C_i(v)$ is bounded by $n^{1/k}$. This indeed proves the lemma, since $C(v) = \bigcup_i C_i(v)$, and so by linearity of expectation we get

$$\mathbb{E}[|C(v)|] = \sum_i \mathbb{E}[|C_i(v)|] \leq \sum_i n^{1/k} = kn^{1/k}.$$

For this, let $B_{i+1}(v)$ denote the set of all vertices $u \in A_i \setminus A_{i+1}$ closer to v than A_{i+1}, *i.e.* $B_{i+1}(v) := \{u \in A_i \setminus A_{i+1} : \delta(v,u) < \delta(v, A_{i+1})\}$. We first argue that the expected size of $B_{i+1}(v)$ is at most $n^{1/k}$ using a similar argument as the one used in Lemma 2. Indeed, the size of $B_{i+1}(v)$ can be bounded by the first location of a vertex from A_{i+1} in the list of all vertices in A_i sorted in increasing distance from u. Thus, $\mathbb{E}[|B_{i+1}(v)|]$ is bounded from above by a geometric random variable with rate $n^{-1/k}$, and so $\mathbb{E}[|B_{i+1}(v)|] \leq n^{1/k}$.

To complete the proof of the lemma, we argue that $C_i(v) \subseteq B_i(v)$. Consider an arbitrary vertex $u \in A_i \setminus A_{i+1}$, and suppose that $u \notin B_{i+1}(v)$. Let $w \in V$ be a vertex satisfying $\delta(v,w) = \delta(v, A_{i+1})$. Then $\delta(u, r(u)) \leq \delta(u,w)$, since $w \in A_{i+1}$ and $\delta(u, r(u)) = \delta(u, A_{i+1})$ by definition. Furthermore, as $u \notin B_{i+1}(v)$, we have $\delta(v,w) \leq \delta(v,u)$. Thus,

$$\delta(u, r(u)) \leq \delta(u,w) \leq \delta(u,v) + \delta(v,w) \leq 2\delta(u,v),$$

and so $v \notin B^{\frac{1}{2}}(u)$ by definition. It follows that $u \notin C_i(v)$, and so $C_i(v) \subseteq B_i(v)$. □

The idea behind our label changing procedure is simple: Given an update request of the form **change**(v, λ), we check which half-balls v belongs to using $C(v)$, and update the corresponding heaps at each half-ball. More precisely, for each vertex u for which $v \in B^{\frac{1}{2}}(u)$, we perform a **remove**(δ) operation on the $\lambda(v)$-heap of u with $\delta := \delta(u,v)$, and perform an **insert**(δ) operation on the λ-heap with the same δ. This requires $O(\lg n)$ time for each vertex u with $v \in B^{\frac{1}{2}}(u)$, *i.e.* $u \in C(v)$, since the distance $\delta(u,v)$ is stored in $C(v)$. Thus, according to Lemma 6, we get the following processing time for each label update.

Lemma 7. *The time required for computing* **change**(v, λ) *is* $O(kn^{1/k} \lg n)$.

4.3 Vertex-Label Queries

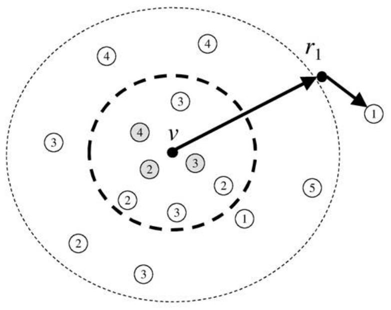

Fig. 2. A graphical depiction of a half ball.

The query procedure of our oracle is similar to the algorithm in Section 3.2. Again the routers $r_0, \ldots, r_{k-1}$ are defined by $r_0 := v$ and $r_i := r(r_{i-1})$ for $i \in$

$\{1,\ldots,k-1\}$. The algorithm determines the smallest integer $i_0 \in \{0,\ldots,k-1\}$ for which $\lambda \in B^{\frac{1}{2}}(r_{i_0})$, and returns the distance

$$\mathbf{dist}(v,\lambda) := \sum_{0\le i<i_0} \delta(r_i, r_{i+1}) + \delta(r_{i_0}, \lambda).$$

The distance $\delta(r_{i_0}, \lambda)$ is retrieved via a single **minimum**() query to the λ-heap stored at $B^{\frac{1}{2}}(r_{i_0})$. The total running-time of our query algorithm is thus $O(k)$. The next lemma bounds the stretch of the output $\mathbf{dist}(v,\lambda)$, completing the proof of Theorem 3.

Lemma 8. $\delta(v,\lambda) \le \mathbf{dist}(v,\lambda) \le (2\cdot 3^{k-1}-1)\cdot\delta(v,\lambda)$ *for any* $(v,\lambda) \in V \times L$.

Proof. Let $(v,\lambda) \in V \times L$, and let $i_0 \in \{0,\ldots,k-1\}$ denote the smallest integer for which $\lambda \in B^{\frac{1}{2}}(r_{i_0})$. We use the following inequality which follows from our definition of the half balls $B^{\frac{1}{2}}(v)$, and from the fact that $\lambda \notin B^{\frac{1}{2}}(r_i)$ for all $i \in \{0,\ldots,i_0-1\}$:

$$\delta(r_i, r_{i+1}) \;\le\; 2\cdot\delta(r_i,\lambda) \text{ for all } i \in \{0,\ldots,i_0-1\}. \tag{3}$$

Using induction on i and (3), one can show the following inequality holds in a similar manner as was done in the proof of Lemma 4:

$$\delta(r_i,\lambda) \le\; 3^i\cdot\delta(v,\lambda) \text{ for all } i \in \{0,\ldots,i_0\}. \tag{4}$$

Thus, we get

$$\begin{aligned}\mathbf{dist}(v,\lambda) &= \textstyle\sum_{0\le i<i_0} \delta(r_i,r_{i+1}) + \delta(r_{i_0},\lambda)\\ &\le \textstyle\sum_{0\le i<i_0} 2\cdot\delta(r_i,\lambda) + \delta(r_{i_0},\lambda)\\ &\le \textstyle\sum_{0\le i<i_0} 2\cdot 3^i\cdot\delta(v,\lambda) + 3^{i_0}\cdot\delta(v,\lambda)\\ &= (2\cdot 3^{i_0}-1)\cdot\delta(v,\lambda),\end{aligned}$$

which proves the lemma since $i_0 \le k-1$. □

This completes the proof of the first part of Theorem 3.

4.4 Small Stretch vs. Efficient Update/Space

We conclude this section by showing that by combining our construction with some of the ideas in [5], it is possible to achieve a stretch-3 oracle with space $O(n^{5/3})$ and label-update time $O(n^{2/3}\lg n)$. Thus, this oracle gives a better stretch than the stretch-5 given by the oracle of Theorem 3, but requires substantially more space and label-update time.

Consider our construction for $k=2$. Then G has three levels $V := A_0 \supseteq A_1 \supseteq A_2 := \emptyset$, with only A_1 non-trivial. In contrast to our original construction, we consider balls, rather than half-balls, around vertices of G. That is, the set $B(v) := \{u \in V : \delta(v,u) < \delta(v,r(v))\}$ for $v \in V$. Pătraşcu and Roditty [5] show how to randomly select the set A_1 such that the following three properties hold:

– $\mathbb{E}[|A_1|] \leq n^{2/3}$.
– $\mathbb{E}[|B(v)|] \leq n^{1/3}$ for any $v \in A_0 \setminus A_1$.
– $\mathbb{E}[|C(v)|] \leq 2n^{2/3}$ for any $v \in V$.

The first two properties imply that our oracle requires $O(n^{5/3})$ space since we store all $O(n^{4/3})$ distances $\delta(u,v)$ for which $u \in V$ and $v \in A_0 \setminus A_1$, and all $O(n^{5/3})$ distances $\delta(u,v)$ for which $u \in V$ and $v \in A_1$. The last property implies that, as in Section 4.2, we can support label updates for any $v \in V$ in $O(n^{2/3} \lg n)$ time. Moreover, since we store distances in balls, rather than half-balls, the stretch analysis of Lemma 4 applies for this construction. Plugging $k := 2$ in Lemma 4, we get an $O(n^{5/3})$-space oracle with stretch 3 and $O(n^{2/3} \lg n)$ label update time.

5 Discussion

In this paper we defined the natural generalization of distance oracles to vertex-labeled graphs and provided small space and stretch approximate distance oracles. Observe that the known lower bounds [5, 7, 9] for unlabeled graphs apply for the generalization when all vertices are uniquely labeled. These lower bounds imply that for any fixed $k \geq 1$, any oracle with stretch $2k-1$ requires $\Omega(n^{1+1/k})$ space. Thus, our $O(n\ell^{1/2})$-space oracle with stretch 3 is optimal up to a logarithmic factor. However, we do not know of any better lower bounds, even more so for the dynamic case. Considering this, we list below the important open questions that remain from our work:

1. Is there a vertex-label $(2k-1)$-stretch oracle scheme with $O(kn^{1+1/k})$ space and $O(k)$ query time?
2. Is it possible to get a general scheme of space $O(kn\ell^{1/k})$, with poly(k) stretch and $O(k)$ query time?
3. Can one get a scheme as in (2) that also supports efficient label updates?
4. Are there $O(kn\ell^{1/k})$-space oracles supporting efficient label updates?
5. Are there $O(n^{3/2})$-space oracles with stretch-3 constant-time queries and $O(n^{1/2})$ label updates?

References

1. S. Baswana, A. Gaur, S. Sen, and J. Upadhyay, *Distance oracles for unwieghted graphs: Breaking the quadratic barrier with constant additive error*, Proc. of the 35th International Colloquium on Automata, Languages and Programming (ICALP), 2008, pp. 609–621.
2. S. Baswana and T. Kavitha, *Faster algorithms for approximate distance oracles and all-pairs small stretch paths*, Proc. of the 47th IEEE Symposium on Foundations of Computer Science (FOCS), 2006, pp. 591–602.
3. P. Erdős, *Extremal problems in graph theory*, Theory of graphs and its applications (1964), 29–36.
4. J. Matoušek, *On the distortion required for embedding finite metric spaces into normed spaces*, Israel Journal of Mathematics (1996), no. 93, 333–344.

5. M. Pătraşcu and L. Roditty, *Distance oracles beyond the thorup-zwick bound*, Proc. of the 51st annual symposium on Foundations Of Computer Science (FOCS), 2010, pp. 815–823.
6. L. Roditty, M Thorup, and U. Zwick, *Deterministic constructions of approximate distance oracles and spanners*, Proc. of the 32nd International Colloquium on Automata, Languages and Programming (ICALP), 2005, pp. 261–272.
7. C. Sommer, E. Verbin, and W. Yu, *Distance oracles for sparse graphs*, Proc. of the 50th IEEE Symposium on Foundation of Computer Science (FOCS), 2009, pp. 703–712.
8. M. Thorup and U. Zwick, *Compact routing schemes*, Proc. of the 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA), 2001, pp. 1–10.
9. ———, *Approximate distance oracles*, Journal of the ACM **52** (2005), no. 1, 1–24.
10. P. van Emde Boas, R. Kaas, and E. Ziljstra, *Design and implementation of an effcient priority queue*, Mathematical Systems Theory **10** (1977), 99–127.