Deterministic Constructions of Approximate Distance Oracles and Spanners

Liam Roditty[1], Mikkel Thorup[2], and Uri Zwick[1]

[1] School of Computer Science, Tel Aviv University, Israel
[2] AT&T Research Labs, USA

Abstract. Thorup and Zwick showed that for any integer $k \geq 1$, it is possible to preprocess any positively weighted undirected graph $G = (V, E)$, with $|E| = m$ and $|V| = n$, in $\tilde{O}(kmn^{1/k})$ *expected* time and construct a data structure (*a* $(2k-1)$*-approximate distance oracle*) of size $O(kn^{1+1/k})$ capable of returning in $O(k)$ time an approximation $\hat{\delta}(u, v)$ of the distance $\delta(u, v)$ from u to v in G that satisfies $\delta(u, v) \leq \hat{\delta}(u, v) \leq (2k-1) \cdot \delta(u, v)$, for any two vertices $u, v \in V$. They also presented a much slower $\tilde{O}(kmn)$ time *deterministic* algorithm for constructing approximate distance oracle with the slightly larger size of $O(kn^{1+1/k} \log n)$. We present here a *deterministic* $\tilde{O}(kmn^{1/k})$ time algorithm for constructing oracles of size $O(kn^{1+1/k})$. Our deterministic algorithm is slower than the randomized one by only a logarithmic factor.

Using our derandomization technique we also obtain the first deterministic *linear* time algorithm for constructing optimal spanners of weighted graphs. We do that by derandomizing the $O(km)$ expected time algorithm of Baswana and Sen (ICALP'03) for constructing $(2k-1)$-spanners of size $O(kn^{1+1/k})$ of weighted undirected graphs without incurring *any* asymptotic loss in the running time or in the size of the spanners produced.

1 Introduction

Thorup and Zwick [16] showed that for any integer $k \geq 1$, any graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, can be preprocessed in $\tilde{O}(kmn^{1/k})$ *expected* time, producing an approximate distance oracle of size $O(kn^{1+1/k})$ capable of returning, in $O(k)$ time, a stretch $2k-1$ approximation of $\delta(u, v)$, for any $u, v \in V$. As discussed in [16], the stretch-size tradeoff presented by this construction is believed to be optimal. The approximate distance oracles of [16] improve previous results of [4] and [7]. For other results dealing with approximate distances, see, [1],[6],[8],[10],[11] [12].

We present here two independent extensions of the result of [16]. The first extension deals with situations in which we are only interested in approximate distances from a specified set $S \subseteq V$ of sources. We show that both the construction time and the space requirements of the appropriate data structure can be reduced in this case. More specifically, we show that if $|S| = s$, then the expected preprocessing time can be reduced from $\tilde{O}(mn^{1/k})$ to $\tilde{O}(ms^{1/k})$ and the space required can be reduced from $O(kn^{1+1/k})$ to $O(kns^{1/k})$. This is significant when $s \ll n$. We call the obtained data structures *source-restricted approximate distance oracles*.

We then move on to solve a major open problem raised in [16], namely the development of *deterministic* algorithms for constructing approximate distance oracles that are almost as efficient as the randomized ones. The deterministic construction in [16] first computes exact APSP in $\tilde{O}(mn)$ time, and then uses the complete distance matrix to derandomize the randomized construction algorithm. In addition to being much slower, the space used by the constructed stretch $2k-1$ oracles is increased to $O(kn^{1+1/k}\log n)$. Our new derandomization loses only a logarithmic factor in running time and suffers no asymptotic loss in space. Thus we get a deterministic $\tilde{O}(mn^{1/k})$ time algorithm for constructing stretch $2k-1$ approximate distance oracles of size $O(kn^{1+1/k})$, solving the problem from [16]. For the source-restricted distance oracles with s sources, the deterministic construction time and space is $\tilde{O}(ms^{1/k})$ and $O(kns^{1/k})$, respectively.

The techniques we use to obtain the new deterministic algorithm can also be used to derandomize the expected linear time algorithm of Baswana and Sen [5] for constructing $(2k-1)$-spanners of size $O(kn^{1+1/k})$, retaining the linear running time and the $O(kn^{1+1/k})$ size of the spanners. Similarly, they can be used to improve the deterministic algorithm of Dor et al. [10] for the construction of 2-emulators (surplus 2 additive spanners) of unweighted graphs. The size of the emulators produced is reduced by a factor of $O(\sqrt{\log n})$ to the optimal $O(n^{3/2})$, with a similar improvement is obtained in the running time. Furthermore, our techniques can be used to improve the the algorithm of Baswana and Sen [6] for the construction of approximate distance oracles for unweighted graphs and make it run, deterministically, in $O(n^2)$ time, which is optimal in terms on n. (Due to lack of space we will not elaborate on this result here.)

The new deterministic algorithm uses two new ingredients that are of interest in their own right and may find additional applications. They are both simple and implementable. The first ingredient is an $\tilde{O}(qm)$ time algorithm that given a weighted directed graph $G=(V,E)$, a subset $U\subseteq V$ of sources, and an integer q, finds for every vertex $v\in V$ the set of the q vertices of U that are closest to v.

The second ingredient is a linear time deterministic algorithm for constructing *close dominating sets*. For a definition of this concept, see Section 4.

The rest of this extended abstract is organized as follows. In the next Section we present the construction of source-restricted approximate distance oracles. In Section 3 we present the algorithm for finding the nearest neighbors. In Section 4 we describe the linear time algorithm for constructing close dominating sets. In Section 5 we then present the main result of this paper, an efficient deterministic algorithm for constructing approximate distance oracles. Due to lack of space we cannot describe here the deterministic version of the linear time spanner construction algorithm of Baswana and Sen [5]. This algorithm will appear in the full version of the paper.

2 Source-Restricted Approximate Distance Oracles

We present here an extension of the approximate distance oracle construction of [16].

Theorem 1. *Let $G=(V,E)$ be an undirected graph with positive weights attached to its edges. Let $k\geq 1$ be an integer, and let $S\subseteq V$ be a specified set of sources. Then, it is*

```
algorithm $\textbf{prepro}_k(G,S)$

$A_0 \leftarrow S$ ; $A_k \leftarrow \phi$

for $i \leftarrow 1$ to $k-1$
    $A_i \leftarrow \textit{sample}(A_{i-1}, |S|^{-1/k})$

for every $v \in V$
    for $i \leftarrow 0$ to $k-1$
        let $\delta(A_i, v) \leftarrow \min\{\, \delta(w,v) \mid w \in A_i\}$
        let $p_i(v) \in A_i$ be such that $\delta(p_i(v), v) = \delta(A_i, v)$

    $\delta(A_k, v) \leftarrow \infty$

    let $B(v) \leftarrow \cup_{i=0}^{k-1}\{\, w \in A_i - A_{i+1} \mid \delta(w,v) < \delta(A_{i+1}, v)\}$
    let $H(v) \leftarrow \textit{hash}(B(v))$
```

Fig. 1. The randomized preprocessing algorithm

possible to preprocess G in $\tilde{O}(km|S|^{1/k})$ expected time, and produce a data structure of size $O(kn|S|^{1/k})$, such that for any $u \in S$ and $v \in V$ it is possible to produce, in $O(k)$ time, an estimate $\hat{\delta}(u,v)$ of the distance $\delta(u,v)$ from u to v in G that satisfies $\delta(u,v) \le \hat{\delta}(u,v) \le (2k-1)\cdot\delta(u,v)$.

Thorup and Zwick [16] prove Theorem 1 for the case $S = V$. The proof of Theorem 1 is obtained by slightly modifying the construction of [16]. For the sake of completeness, we give the full details. This also allows us to review the randomized construction of [16] before presenting a deterministic version of it later in this paper.

Proof. A high level description of the preprocessing algorithm is given in Figure 1. The algorithm starts by defining a hierarchy $A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_k$ of subsets of S in the following way: We begin with $A_0 = S$. For every $1 \le i < k$, we let A_i be random subset of A_{i-1} obtained by selecting each element of A_{i-1}, independently, with probability $|S|^{-1/k}$. Finally, we let $A_k = \phi$. The elements of A_i are referred to as *i-centers*. A similar hierarchy is used in [16]. There, however, we have $A_0 = V$, and A_i, for $1 \le i < k$, is obtained by selecting each element of A_{i-1} with probability $n^{-1/k}$. Interestingly, this is the *only* change needed with respect to the construction of [16].

Next, the algorithm finds, for each vertex $v \in V$, and each $1 \le i < k$, the distance $\delta(A_i, v) = \min\{\delta(w,v) \mid w \in A_i\}$ and an i-center $p_i(v) \in A_i$ that is closest to v. (We assume $A_{k-1} \ne \phi$.) For every vertex $v \in V$ it then defines the *bunch* $B(v)$ as follows:

$$B(v) \;=\; \cup_{i=0}^{k-1} B_i(v) \quad , \quad B_i(v) \;=\; \{w \in A_i \mid \delta(w,v) < \delta(A_{i+1}, v)\,\}\ .$$

Note that $B_i(v) \subseteq A_i - A_{i+1}$ as if $w \in A_{i+1}$ then $\delta(A_{i+1}, v) \le \delta(w,v)$. We show later that the centers $p_i(v)$ can be found in $\tilde{O}(km)$ time and that the bunches $B(v)$, and the distances $\delta(w,v)$, for every $w \in B(v)$, can be found in $\tilde{O}(km|S|^{1/k})$ expected time.

Finally, for every vertex $v \in V$ the preprocessing algorithm constructs a hash table $H(v)$ of size $O(|B(v)|)$ that stores for each $w \in B(v)$ the distance $\delta(w,v)$. The hash table is constructed in $O(|B(v)|)$ expected time using the algorithm of Fredman *et al.*

[13]. (For a deterministic version, see Alon and Naor [2].) For every $w \in V$ we can then check, in $O(1)$ time, whether $w \in B(v)$ and if so obtain $\delta(w, v)$.

The total size of the data structure produced is $O(kn + \sum_{v \in V} |B(v)|)$. We next show that for every $v \in V$ we have $E[\,|B(v)|\,] \leq k|S|^{1/k}$, and thus the expected size of the whole data structure is $O(kn|S|^{1/k})$.

Lemma 1. *For every vertex $v \in V$ we have $E[\,|B(v)|\,] \leq k|S|^{1/k}$.*

Proof. We show that $E[\,|B_i(v)|\,] \leq |S|^{1/k}$, for $0 \leq i < k$. For $i = k - 1$ the claim is obvious as $B_{k-1}(v) \subseteq A_{k-1}$, and $E[\,|A_{k-1}|\,] = |S|^{1/k}$. Suppose, therefore, that $0 \leq i < k - 1$, and suppose that A_i was already chosen, while A_{i+1} is now about to be chosen. Let $w_1, w_2, \ldots, w_\ell$ be the vertices of A_i arranged in non-decreasing order of distance from v. If $w_j \in A_{i+1}$, then $B_i(v) \subseteq \{w_1, w_2, \ldots, w_{j-1}\}$. Thus $\Pr[w_j \in B_i(v)] \leq \Pr[w_1, w_2, \ldots, w_{j-1} \notin A_{i+1}]$. As each vertex of A_i is placed in A_{i+1}, independently, with probability $p = |S|^{-1/k}$, we get that $\Pr[w_j \in B_i(v)] \leq (1-p)^{j-1}$ and thus $E[\,|B_i(v)|\,] \leq \sum_{j \geq 1}(1-p)^{j-1} = p^{-1} = |S|^{1/k}$, as required. □

The algorithm used to answer approximate distance queries is given in Figure 2.

Lemma 2. *For every $u \in S$ and $v \in V$, algorithm* $\mathbf{dist}_k(u, v)$ *runs in $O(k)$ time and returns an approximate distance $\hat{\delta}(u, v)$ satisfying $\delta(u, v) \leq \hat{\delta}(u, v) \leq (2k-1)\delta(u, v)$.*

Proof. Let $\Delta = \delta(u, v)$. We begin by proving, by induction, that at the start of each iteration of the while loop we have $w \in A_i$ and $\delta(w, u) \leq i\Delta$. This clearly holds at the start of the first iteration, when $i = 0$, as $w = u \in S = A_0$ and $\delta(w, u) = 0$. (Here is were we use the assumption that $u \in S$.) Suppose, therefore that the claim holds at the start of some iteration, i.e., $w \in A_i$ and $\delta(w, u) \leq i\Delta$, and that the while condition, i.e., $w \notin B(v)$, is satisfied. Let $w' = p_{i+1}(v) \in A_{i+1}$. As $w \notin B(v)$, we get, by the definition of $B(v)$, that $\delta(w', v) = \delta(A_{i+1}, v) \leq \delta(w, v)$. We therefore have

$$\delta(w', v) \leq \delta(w, v) \leq \delta(w, u) + \delta(u, v) \leq i\Delta + \Delta = (i + 1)\Delta\ .$$

Thus, by incrementing i, swapping u and v and letting $w \leftarrow w'$ we reestablish the invariant condition. (The algorithm performs these operations in a slightly different order.)

In each iteration of the while loop the algorithm performs only a constant number of operations. (To check whether $w \in B(v)$ it uses the hash table $H(v)$.) As $B(v) \supseteq A_{k-1}$ and $w \in A_i$, the algorithm performs at most $k-1$ iterations and hence the running time is $O(k)$.

When the while loop terminates, we have $\delta(w, u) \leq i\Delta$, $w \in B(v)$ and $i \leq k - 1$. The algorithm then returns the estimate $\hat{\delta}(u, v) = \delta(w, u) + \delta(w, v)$ which satisfies

$$\begin{aligned} \delta(w, u) + \delta(w, v) &\leq \delta(w, u) + (\delta(w, u) + \delta(u, v)) \\ &= 2\delta(w, u) + \Delta \leq 2(k - 1)\Delta + \Delta \leq (2k - 1)\Delta\ , \end{aligned}$$

as required. □

All that remain, therefore, is to explain how the preprocessing algorithm can be implemented to run in $\tilde{O}(km|S|^{1/k})$ time. Finding for each vertex $v \in V$ and every $0 \le i < k$ the vertex $p_i(v) \in A_i$ closest to v is fairly easy. For every $0 \le i < k$ we add a new source vertex s_i to the graph and connect it with zero weight edges to all the vertices of A_i. By running Dijkstra's algorithm (see [9]) we compute the distances from s_i to all other vertices and construct a shortest paths tree rooted at s_i. The distances thus computed are exactly $\delta(A_i, v)$, for every $v \in V$. Using the shortest paths tree it is easy to identify for every $v \in V$ a vertex $p_i(v) \in A_i$ for which $\delta(p_i(v), v) = \delta(A_i, v)$. The whole process requires only $\tilde{O}(km)$ time.

We next describe an $\tilde{O}(km|S|^{1/k})$ algorithm for constructing the bunches $B(v)$, for every $v \in V$. Instead of computing the bunches directly, we compute their 'duals'. For every i-center $w \in A_i - A_{i+1}$ we define the *cluster* $C(w)$ as follows:

$$C(w) \;=\; \{v \in V \mid \delta(w,v) < \delta(A_{i+1},v)\,\} \quad , \quad \text{for } w \in A_i - A_{i+1}\ .$$

It is easy to see that $v \in C(w)$ if and only if $w \in B(v)$. We now claim:

Lemma 3. *If $v \in C(w)$, and u is on a shortest path from w to v in G, then $u \in C(w)$.*

Proof. Suppose that $w \in A_i - A_{i+1}$. If $u \notin C(w)$, then $\delta(A_{i+1}, u) \le \delta(w, u)$. But then $\delta(A_{i+1}, v) \le \delta(A_{i+1}, u) + \delta(u, v) \le \delta(w, u) + \delta(u, v) = \delta(w, v)$, contradicting the assumption that $v \in C(w)$. □

It follows that the cluster $C(w)$ can be constructed using the modified version of Dijkstra's algorithm given in Figure 3. For the straightforward correctness proof, the reader is referred to [16]. The running time of the algorithm, when Fibonacci heaps [14]

```
algorithm dist_k(u,v)
(Assumption: $u \in S$)
$w \leftarrow u$ ; $i \leftarrow 0$
while $w \notin B(v)$
    $i \leftarrow i+1$
    $(u,v) \leftarrow (v,u)$
    $w \leftarrow p_i(u)$
return $\delta(w,u) + \delta(w,v)$
```

Fig. 2. Answering a distance query

```
algorithm cluster(G, w, A)

$d[w] \leftarrow 0$ ; $C \leftarrow \phi$
$Q \leftarrow \phi$ ; insert$(Q, w, d[w])$
while $Q \ne \phi$
        $u \leftarrow$ extract-min$(Q)$
        $C \leftarrow C \cup \{u\}$
        for every $(u,v) \in E$
            $d \leftarrow d[u] + \ell(u,v)$
            if $d < \delta(A, v)$ then
                if $v \notin Q$ then
                    $d[v] \leftarrow d$ ; insert$(Q, v, d[v])$
                else if $d < d[v]$ then
                    $d[v] \leftarrow d$ ; decrease-key$(Q, v, d)$
return $C$
```

Fig. 3. Constructing a cluster

(see also [9]) are used to implement the priority queue Q, is $O(m_w + n_w \log n_w) = O(m_w + n_w \log n)$, where $n_w = |C(w)|$ and m_w is the total number of edges touching the vertices of $C(w)$. This is $O(\log n)$ time per vertex v in $C(w)$ and constant time per edge touching such a vertex v. However, $v \in C(w) \iff w \in B(v)$ and $E[\,|B(v)|\,] \leq ks^{1/k}$, so the total expected running time needed for constructing all clusters is $O((m + n \log n)ks^{1/k})$, as required. The running time can be reduced to $O(kms^{1/k})$ using the techniques of Thorup [15]. This completes the proof of Theorem 1. □

3 A Deterministic Algorithm for Finding the *q* Nearest Centers

Let $G = (V, E)$ be a weighted directed graph and let $U \subseteq V$ be an arbitrary set of sources, or centers. We start with a formal definition of the set $U_q(v)$ of the q nearest centers from U of a vertex $v \in V$. We assume that all edge weights are positive. We also assume, without loss of generality, that $V = \{1, 2 \ldots, n\}$.

Definition 1 (Nearest centers from U). *Let $G = (V, E)$ be a directed graph with positive edge weights assigned to its edges. Let $U \subseteq V$ be an arbitrary set of sources, and let $1 \leq q \leq |U|$ be an integer. For every $v \in V$, the set $U_q(v)$ is defined to be the set of the q vertices of U that are closest to v. Ties are broken in favor of vertices with smaller indices. More precisely, for every $v \in V$ we have $U_q(v) \subseteq U$, $|U_q(v)| = q$ and if $w_1 \in U_q(v)$ while $w_2 \notin U_q(v)$ then either $\delta(w_1, v) < \delta(w_2, v)$ or $\delta(w_1, v) = \delta(w_2, v)$ and $w_1 < w_2$.*

The following lemma, which is reminiscent of Lemma 1, is easily verfied.

Lemma 4. *If $u \in U_q(v)$ and w lies on a shortest path from u to v in G, then $u \in U_q(w)$.*

We now claim:

Theorem 2. *Let $G = (V, E)$ be a directed graph with positive weights assigned to its edges. Let $U \subseteq V$ be an arbitrary set of sources, and let $1 \leq q \leq |U|$ be an integer. Then, the sets $U_q(v)$, for every $v \in V$, can be computed by performing q single-source shortest paths computations on graphs with at most $O(n)$ vertices and $O(m)$ edges.*

Proof. We begin by finding for every vertex $v \in V$ its nearest neighbor in U. This is easily done by adding a new source vertex s to the graph, connecting it with 0 length edges to all the vertices of U, and computing a tree of shortest paths in the resulting graph. This gives us $U_1(v)$, for every $v \in V$.

Suppose now that we have already computed $U_{i-1}(v)$, for every $v \in V$. We show that $U_i(v)$, for every $v \in V$ can be obtained by finding a tree of shortest paths in an auxiliary graph with $O(n)$ vertices and $O(m + n)$ edges. This auxiliary graph is constructed as follows:

1. Add to G a new source vertex s and *copies* of all vertices of U. If $u \in U$, we let $\bar{u}$ denote the copy of u. Add a 0 length edges from s to $\bar{u}$, for every $u \in U$.
2. For every edge $(v, w) \in E$:
   (a) If $U_{i-1}(v) = U_{i-1}(w)$, keep the edge (v, w).

(b) Otherwise, if $U_{i-1}(v) \neq U_{i-1}(w)$ and u is the first vertex in $U_{i-1}(v) - U_{i-1}(w)$, replace the edge (v,w) by an edge $(\bar{u},w)$ of length $\delta(u,v)+\ell(v,w)$.

The auxiliary graph thus contains $n+|U|+1$ vertices and $m+|U|$ edges. It is not difficult to check that u is the i-th nearest neighbor from U of all the vertices in the subtree of $\bar{u}$ in the tree of shortest paths from s in this auxiliary graph. The proof is fairly straightforward and is omitted due to lack of space. □

4 A Deterministic Construction of Close Dominating Sets

Instead of dealing directly with the close dominating sets from the introduction, it is convenient first to consider a simpler case phrased in terms of a matrix. In that context, we will talk about *early hitting sets*: Let M be an $n \times k$ matrix whose elements are taken from a finite set S of size $|S| = s$. We assume that the elements in each row of M are distinct. A set A is said to be a *hitting set* of M if and only if every row of M contains an element of A. A standard calculation shows that if each element of S is placed in A, independently, with probability $(c \ln n)/k$, for some $c > 1$, then with a probability of at least $1 - n^{1-c}$ the resulting set A is a hitting set of M. The expected size of A is then $(c\, s \ln n)/k$. We are interested in hitting sets of small size that hit the rows of M close to their beginnings, at least on average.

Definition 2 (Hitting sums). *Let M be an $n \times k$ matrix, let A be a set, and let $P \geq 0$ be a* penalty. *Let $hit(M_i, A)$ be the index of the first element of M_i, the i-th row of M, that belongs to A, or $k + P$, if no element of M_i belongs to A. Let $hit(M,A) = \sum_{i=1}^{n} hit(M_i, A)$ be the* hitting sum *of A with respect to M.*

Note that a set A need not be a hitting set of M for the hitting sum $hit(M,A)$ to be defined. A penalty of P, plus the length of the row, is paid, however, for each row that is not hit. Typically, the goal is to hit all rows avoiding all penalties. A set A with a small hitting sum $hit(M,A)$ is informally referred to as an *early* hitting set. The following simple probabilistic lemma proves the existence of small early hitting sets.

Lemma 5. *Let M be an $n \times k$ matrix whose elements are taken from a finite set S of size $|S| = s$ and let $P \geq 0$ be a penalty. Then, for every $0 < p < 1$ there exists a set $A \subseteq S$ for which $\frac{n}{p^2 s}|A| + hit(M,A) \leq 2n/p + (1-p)^k Pn$. In particular, if $pP \geq 3n$ and $pP(1-p)^k \leq 1$ than all rows are hit with $|A| < 3ps$ and $hit(M,A) < 3n/p$.*

Proof. Let A be a random subset of S obtained by selecting each element of S, independently, with probability p. It is easy to see that

$$E[\,|A|\,] \;=\; p\,s\;,$$

$$E[\,hit(M_i,A)\,] \;=\; \sum_{j=1}^{k}(1-p)^{j-1} + (1-p)^k P \;<\; p^{-1} + (1-p)^k P\;,$$

and thus $E[\,\frac{n}{p^2 s}|A| + hit(M,A)\,] \;\leq\; \frac{2n}{p} + (1-p)^k Pn$. This proves the existence of the required set.

Concerning the last statement, the condition $pP(1-p)^{-k} < 1$ implies that the right hand side is at most $3n/p$. By the first condition, this corresponds to at most a single penalty, but since we have other costs, we conclude that we pay no penalties. The bounds on $|A|$ and $hit(M, A)$ follow because each term on the left hand size is non-zero and strictly smaller than the right hand side. □

The main result of this section is a deterministic *linear time* algorithm for constructing early hitting sets that almost match the bounds of Lemma 5. Quite naturally, the algorithm is based on the *method of conditional expectations* (see, e.g., Alon and Spencer [3]). The challenge is to get a running time linear in the size of the matrix M.

Theorem 3. *Let M be an $n \times k$ matrix whose elements are taken from a finite set S of size $|S| = s$ and let $P \geq 0$ be a penalty. Let $0 < p < 1$. Then, there is a* deterministic *$O(nk)$ time algorithm that finds a set $A \subseteq S$ for which $\frac{n}{p^2 s}|A| + hit(M, A) \leq 3n/p + (1-p)^k Pn$. In particular, if $pP \geq 4n$ and $pP(1-p)^k \leq 1$ then all rows are hit with $|A| < 3ps$ and $hit(M, A) < 3n/p$.*

Proof. Let $A_0, A_1 \subseteq S$ be two disjoint sets. Define

$$hit(M \mid A_0, A_1) = E[\,\frac{n}{p^2 s}|A| + hit(M, A) \mid A_1 \subseteq A \subseteq A_0^c\,]\,.$$

In other words, $hit(M|A_0, A_1)$ is the (conditional) expectation of the random variable $\frac{n}{p^2 s}|A| + hit(M, A)$ where the set A is chosen in the following way: Each element of A_1 is placed in A. Each element of A_0 is *not* placed in A. Each other element is placed in A, independently, with probability p.

Lemma 5 states that $hit(M, A) = hit(M \mid \phi, \phi) \leq \mu = 2n/p + (1-p)^k Pn$. Our goal it to deterministically find a set $A \subseteq S$ such that $hit(M \mid A^c, A) \leq \mu$. Suppose that we have already found two disjoint sets $A_0, A_1 \subseteq S$ such that $hit(M \mid A_0, A_1) \leq \mu$ and that $e \in S - (A_0 \cup A_1)$. We then have

$$hit(M \mid A_0, A_1) \;=\; p\cdot hit(M \mid A_0, A_1 \cup \{e\}) \;+\; (1-p)\cdot hit(M \mid A_0 \cup \{e\}, A_1)\,.$$

Thus, at least one of the two conditional expectations appearing above is at least μ. We choose it and then consider another element that was not yet placed in either A_0 and A_1. Continuing in this way, we get two disjoint sets $A_0, A_1 \subseteq S$ such that $A_0 \cup A_1 = S$ and $hit(M \mid A_0, A_1) \leq \mu$, as required. This is precisely the method of conditional expectations.

The remaining question is the following: Given $hit(M \mid A_0, A_1)$ and an element $e \in S - (A_0 \cup A_1)$, how fast can we compute $hit(M \mid A_0, A_1 \cup \{e\})$ and $hit(M \mid A_0 \cup \{e\}, A_1)$? Let us focus on the computation of the conditional expectations $hit(M_i \mid A_0, A_1 \cup \{e\})$ and $hit(M_i \mid A_0 \cup \{e\}, A_1)$ corresponding to the i-th row of M.

Let $n_i = n_i(A_1)$ be the index of the first element in M_i that belongs to A_1. If none of the elements of M_i belongs to A_1, we let $n_i = \infty$. Let $n_{i,j} = n_{i,j}(A_0)$ be the number of elements among the first j elements of M_i that do *not* belong to A_0. (We let $n_{i,0} = 0$.) It is easy to see that

$$hit(M_i \mid A_0, A_1) \;=\; \sum_{j=1}^{\min\{n_i,k\}} (1-p)^{n_{i,j-1}} + \begin{cases} (1-p)^{n_{i,k}} P & \text{if } n_i = \infty \\ 0 & \text{otherwise} \end{cases}\,.$$

Maintaining the penalty term $(1-p)^{n_{i,k}}P$ is easy. To simplify the presentation we therefore ignore this term. (In other words we assume that $P=0$. The changes needed when $P>0$ are minimal.) Let

$$x_{i,j} \;=\; \begin{cases} (1-p)^{n_{i,j-1}} & \text{if } j \le n_i \;, \\ 0 & \text{otherwise} \;. \end{cases}$$

With this notation, and with the assumption $P=0$, we clearly have $hit(M_i \mid A_0, A_1) \;=\; \sum_{j=1}^{k} x_{i,j}$. We now consider the changes that should be made to the $x_{i,j}$'s when an element e is added to A_0 or to A_1. If e does not appear in M_i, then no changes are required. Assume, therefore, that $M_{ir} = e$, i.e., the e is the r-th element in M_i. If $r > n_i$, then again no changes are required. Assume, therefore, that $r < n_i$.

If e is added to A_1, then the required operations are $n_i \leftarrow r$ and $x_{i,j} \leftarrow 0$, for $r < j \le k$. If e is added to A_0, then the required operations are $n_{i,j} \leftarrow n_{i,j} - 1$, for $r \le j \le k$, and therefore $x_{i,j} \leftarrow x_{i,j}/(1-p)$, again for $r \le j \le k$. In both cases, the new conditional expectation is the new sum $\sum_{j=1}^{k} x_{i,j}$.

These operations can be implemented fairly efficiently using a data structure that maintains an array $x = [x_1, x_2, \ldots, x_q]$ of q real numbers under the following update operations: $init(x)$ – initialize the array x; $scale(i,j,a)$ – multiply the elements in the sub-array $[x_i, \ldots, x_j]$ by the constant a; sum – return the sum $\sum_{i=1}^{k} x_i$; and $undo$ – undo the last update operation. (The $undo$ operation is required for tentatively placing new elements in A_0 and then in A_1.) Using standard techniques it is not difficult to implement such a data structure that can be initialized in $O(k)$ time and that can support each update operation in $O(\log k)$ time. However, the description of such a data structure is not short, and the resulting algorithm would have an over all non-linear running time of $O(nk\log k)$. Luckily, there is a simpler to implement, and a more efficient, solution. Let us define the following variant of hitting sums:

Definition 3 (Dyadic hitting sums). *Let M be an $n \times k$ matrix, let A be a set, and let $P \ge 0$ be a* penalty. *Let $\overline{hit}(M_i, A) = 2^{\lceil \log_2 hit(M_i,A) \rceil}$ be the smallest power of 2 greater or equal to the index of the first element of M_i that belongs to A, or $q+P$, if no element of M_i belongs to A. Let $\overline{hit}(M, A) = \sum_{i=1}^{n} \overline{hit}(M_i, A)$ be the* dyadic hitting sum *of A with respect to M.*

Clearly $hit(M,A) \le \overline{hit}(M,A) < 2{\cdot}hit(M,A)$. Thus, as in the proof of Lemma 5, we get that $E[\,\frac{n}{p^2 s}|A| + \overline{hit}(M,A)\,] \le 3n/p + (1-p)^k Pn$. The conditional expectation $\overline{hit}(M \mid A_0, A_1)$ is defined in the obvious analogous way. Now define

$$\bar{k} = \lceil \log_2 k \rceil, \quad \bar{n}_i \;=\; \lceil \log_2 n_i \rceil \;, \quad \bar{n}_{i,j} \;=\; n_{i,2^j} \;,$$

$$\bar{x}_{i,j} \;=\; \begin{cases} (1-p)^{\bar{n}_{i,j-1}} & \text{if } j \le \bar{n}_i \;, \\ 0 & \text{otherwise} \;. \end{cases} , \quad \bar{y}_{i,r} = 1 + \sum_{j=1}^{r} \bar{x}_{i,j} 2^{j-1}$$

With these definitions we have $\overline{hit}(M_i \mid A_0, A_1) \;=\; \bar{y}_{i,\bar{k}-1}$.

Each update now trivially takes $O(\bar{k}) = O(\log k)$ worst-case time, even if we implement the updates naively. Furthermore, we argue that the *amortized* cost of each update is only $O(1)$!

```
procedure $init(i)$

$\bar{x}_{i,1} \leftarrow 1-p$
$\bar{y}_{i,1} \leftarrow 1+\bar{x}_{i,1}$
for $j \leftarrow 2$ to $\bar{k}-1$
    $\bar{x}_{i,j} \leftarrow \bar{x}^2_{i,j-1}$
    $\bar{y}_{i,j} \leftarrow \bar{y}_{i,j-1} + \bar{x}_{i,j} \cdot 2^{j-1}$
```

```
procedure $update_0(i,r)$

$\bar{r} \leftarrow \lceil \log_2 r \rceil$
for $j \leftarrow \bar{r}$ to $\bar{k}-1$
    $\bar{x}_{i,j} \leftarrow \bar{x}_{i,j}/(1-p)$
    $\bar{y}_{i,j} \leftarrow \bar{y}_{i,j-1} + \bar{x}_{i,j} \cdot 2^{j-1}$
return $\bar{y}_{i,\bar{k}-1}$
```

```
procedure $update_1(i,r)$

$\bar{r} \leftarrow \lceil \log_2 r \rceil$
for $j \leftarrow \bar{r}+1$ to $\bar{k}-1$
    $\bar{x}_{i,j} \leftarrow 0$
    $\bar{y}_{i,j} \leftarrow y_{i,j-1}$
return $\bar{y}_{i,\bar{k}-1}$
```

Fig. 4. Updating the conditional expectations

A complete description of procedures used to initialize and update the conditional expectations is given in Figure 4. A call to *init*(i) initializes $\bar{x}_{i,j} = (1-p)^{2^{j-1}}$ and $\bar{y}_{i,r} = 1 + \sum_{j=1}^{r} \bar{x}_{i,j} \cdot 2^{j-1}$, for $1 \le r \le \bar{k}-1$. Calls to *update*$_0(i,j)$ and *update*$_1(i,j)$, respectively, perform the necessary updates to the i-rows of the arrays $x[i,j]$ and $y[i,j]$ as a result of adding the element $e = M_{ir}$ to A_0, or to A_1, and return the new value of $\overline{hit}(M_i|A_0,A_1)$. The difference between the old and the new value of $\overline{hit}(M_i|A_0,A_1)$ should also be applied to the global sum $\overline{hit}(M \mid A_0,A_1) = \sum_{i=1}^{n} \overline{hit}(M_i|A_0,A_1)$. It is easy to implement an $undo(i)$ procedure that undos the last update performed on the i-th row. We simply need to record the operations made and undo them in reverse order. To obtain $\overline{hit}(M,A_0,A_1)$, we simply sum $\overline{hit}(M_i|A_0,A_1)$ up, for $1 \le i \le n$. The correctness of the computation follows from the long discussion above.

All that remains is to analyze the complexity of the proposed algorithm. Each element $e \in S$ is considered once by the algorithm. For each appearance of $e = M_{ir}$ in M we need to call *update*$_0(i,r)$ and *update*$_1(i,r)$. The complexity of these calls is $O(\bar{k} - \bar{r} + 1) = O(\lceil \log_2 k \rceil - \lceil \log_2 r \rceil + 1)$. For every $2^{\bar{k}-j} \le r \le 2^{\bar{k}-j+1}$, where $1 \le j \le \bar{k}$, the cost is $O(j)$. Thus, the total cost of handling all the elements of the i-th row is $O(k \sum_{j \ge 1} j 2^{-j}) = O(k)$. The total cost is therefore $O(kn)$, as required. The last statement of the theorem is derived like the last statement of Lemma 5. □

Theorem 4. *A close dominating set of any given size can be found in linear time.*

Proof. We now consider the closest dominating set problem from the introduction, modifying our early hitting set algorithm to solve this problem. The first change is to let each row M_i to have an individual length $k_i \le k$. The total number of elements is then $m = \sum_{i=1}^{n} k_i$. We also make the change that there is only a penalty P for not hitting a full row M_i with $k_i = k$. It is straightforward to modify the previous early hitting set algorithm for these variable length rows. Essentially, we just replace k and $\bar{k}$ by k_i and $\bar{k}_i$, and drop the penalty for the partial rows. We then get a deterministic algorithm that in $O(m)$ time finds a hitting set A with the same properties as those stated in Theorem 3. In particular, if $pP \ge 4n$ and $pP(1-p)^k \le 1$ then all full rows are hit with $|A| < 3ps$ and $hit(M,A) < 3n/p$.

We now need to transform our bipartite graph $G = (U,V,E)$ to the matrix form. The set S of elements that are placed in the matrix is simply the set U. The matrix constructed has a row for each vertex $v \in V$. Ideally, the row M_v would contain the neighboring centers u ordered according to the edge weights $\ell(u, v)$. The list should be truncated to only contain the $k = (s/h)(2 + \ln n)$ nearest centers. The lists with k centers are the full rows with a penalty P for not being hit. We use $p = h/(3s)$ and $P = 12ns/h$. Then $pP = 4n$ and

$$pP(1-p)^k < 4n \exp(-(h/s)(s/h(2 + \ln n)) < 4/e^2 < 1.$$

Since the conditions are satisfied, we get a set A hitting all full rows with $|A| < 3ps = h$, and $hit(M, A) < 3n/p = 9ns/h$. This also means that A is a close dominating set.

Our only remaining problem is that we cannot sort neighboring centers according to distance. However, thanks to the dyadic solution, it suffices to apply a linear time selection algorithm (see, e.g., [9]). First, if a vertex v has more the k neighboring centers, we apply selection to find the k nearest centers. Next, for r decreasing from $\lfloor \log_2 k_i \rfloor$ down to 0, we identify the 2^r nearest centers. The total running time is linear, and this provides a sufficient sorting for the diadic hitting sum algorithm. □

5 A Deterministic Construction of Approximate Distance Oracles

In this section we present a *deterministic* algorithm for constructing (source-restricted) approximate distance oracles. The algorithm is slower than the randomized algorithm of Theorem 1 by only a logarithmic factor. Obtaining such a deterministic algorithm is one of the open problems mentioned in [16].

Theorem 5. *Let $G = (V, E)$ be an undirected graph with positive weights attached to its edges. Let $k \geq 1$ be an integer, and let $S \subseteq V$ be a specified set of sources. Then, it is possible to preprocess G, deterministically, in $\tilde{O}(km|S|^{1/k})$ time, and produce a data structure of size $O(kn|S|^{1/k})$, such that for any $u \in S$ and $v \in V$ it is possible to produce, in $O(k)$ time, an estimate $\hat{\delta}(u, v)$ of the distance $\delta(u, v)$ from u to v in G that satisfies $\delta(u, v) \leq \hat{\delta}(u, v) \leq (2k-1) \cdot \delta(u, v)$.*

```
algorithm **detpre**$_k(G, S)$
$A_0 \leftarrow S$ ; $A_k \leftarrow \emptyset$
$p \leftarrow \frac{1}{4}|S|^{-1/k}$ ; $\ell \leftarrow 3|S|^{1/k} \ln n$ ; $P \leftarrow n^2$
for $i \leftarrow 1$ to $k-1$
    $N_{i-1} \leftarrow$ **near**$(G, A_{i-1}, \ell)$
    Create bipartite graph $B$ from $A_{i-1}$ to $V$ with
    an edge $(u, v)$ of length $\delta_G(u, v)$ if $u \in N_{i-1}[v]$.
    $A_i \leftarrow$ **domset**$(B, s^{1-i/k})$
for every $v \in V$
    $B(v) \leftarrow A_{k-1}$
    for $i \leftarrow 0$ to $k-2$
        $B(v) \leftarrow B(v) \cup \{ w \in N_i[v] \mid \delta(w, v) < \delta(A_{i+1}, v) \}$
```

Fig. 5. The deterministic preprocessing algorithm

Proof. The deterministic preprocessing algorithm is given in Figure 5. It is composed of $k-1$ iteration. The i-th iteration constructs the set A_i. We let $s = |S|$ and $\ell = \lceil s^{1/k}(2+\ln n)\rceil$, the iteration begins by finding for each vertex $v \in V$ the set $N_i[v]$ of the ℓ vertices of A_{i-1} that are nearest v, using algorithm **near** of Section 3. The running time of the algorithm is $\tilde{O}(ms^{1/k})$. Next we create a bipartite graph B from A_{i-1} to V with an edge (u,v) of length $\delta_G(u,v)$ if $u \in N_{i-1}[v]$. Using the algorithm of Theorem 4, which we here call **domset**, we now find a close dominating subset A_i of size $h_i = s^{1-i/k} = |A_{i-1}|/s^{1/k}$. Since each vertex have at least $s^{1/k}(2+\ln n) = |A_{i-1}|/h_i(2+\ln n)$ neighboring centers, we know that A_i hits all these neighborhoods. The result is that in the original graph G, the sum of the number of centers in from A_{i-1} nearer than then nearest center in A_i is at most $9|A_{i-1}|/h_i = O(ns^{1/k})$. It follows that the total size of the bunches returned by the algorithm is in $O(kns^{1/k})$, as required. □

References

1. D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28:1167–1181, 1999.
2. N. Alon and M. Naor. Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16:434–449, 1996.
3. N. Alon and J.H. Spencer. *The probabilistic method.* Wiley-Interscience, 2nd edition, 2000.
4. B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28:263–277, 1999.
5. S. Baswana and S. Sen. A simple linear time algorithm for computing $(2k-1)$-spanner of $O(n^{1+1/k})$ size for weighted graphs. In *Proc. of 30th ICALP*, pages 384–296, 2003.
6. S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in $O(n^2 \log n)$ time. In *Proc. of 15th SODA*, pages 264–273, 2004.
7. E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28:210–236, 1999.
8. E. Cohen and U. Zwick. All-pairs small-stretch paths. *Journal of Algorithms*, 38:335–353, 2001.
9. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms.* The MIT Press, 2nd edition, 2001.
10. D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. *SIAM Journal on Computing*, 29:1740–1759, 2000.
11. M. Elkin. Computing almost shortest paths. In *Proc. of 20th PODC*, pages 53–62, 2001.
12. M.L. Elkin and D. Peleg. $(1+\epsilon,\beta)$-Spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.
13. M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31:538–544, 1984.
14. M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.
15. M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.
16. M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.