Compact Oracles for Reachability and Approximate Distances in Planar Digraphs

MIKKEL THORUP

AT&T Labs—Research, Florham Park, New Jersey

Abstract. It is shown that a planar digraph can be preprocessed in near-linear time, producing a near-linear space oracle that can answer reachability queries in constant time. The oracle can be distributed as an $O(\log n)$ space label for each vertex and then we can determine if one vertex can reach another considering their two labels only.

The approach generalizes to give a near-linear space approximate distances oracle for a weighted planar digraph. With weights drawn from $\{0, \ldots, N\}$, it approximates distances within a factor $(1+\varepsilon)$ in $O(\log\log(nN) + 1/\varepsilon)$ time. Our scheme can be extended to find and route along correspondingly short dipaths.

Categories and Subject Descriptors: E.1 [**Data Structures**]: *Graphs and networks*; F.2.2 [**Theory of Computation**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures, routing and layout*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms, graph labeling, path and circuit problems*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Planar graphs, reachability and shortest paths oracles

1. *Introduction*

We show that a planar digraph over n vertices can be preprocessed in $O(n \log n)$ time and space, producing an oracle that can answer reachability queries in constant time. No $o(n^2)$ bit oracle was known that could answer reachability queries in constant time.

Our reachability result is generalized to approximate distances. A *stretch-t distance oracle* provides distance estimates that are no less but up to t times bigger than the true distances. More precisely, if $\hat{\delta}(v, w)$ is the estimated distance from v to w and $\delta(v, w)$ is the true distance, then $\delta(v, w) \leq \hat{\delta}(v, w) \leq t\delta(v, w)$.

A preliminary version of this article was presented at the 42nd IEEE Symposium on Foundations of Computer Science and was published in *Proceedings of the* 42nd *IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, Calif., 2001, pp. 242–251.

Author's address: AT&T Labs—Research, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932, e-mail: mthorup@research.att.com.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2004 ACM 0004-5411/04/1100-0993 $5.00

If the edge weights are drawn from $\{0, 1, \ldots, N\}$, then for any fixed $\varepsilon > 0$, we provide a stretch-$(1 + \varepsilon)$ distances oracle using $O(n(\log n)(\log(nN))/\varepsilon)$ space to get a query time of $O(\log\log(nN) + 1/\varepsilon)$. The oracle is constructed in time $O(n(\log n)^3(\log(nN))/\varepsilon^2)$. Our results extend to provide routing along short dipaths.

Our computational model is word RAM, modeling what we can program in a standard programming language such as C [Kernighan and Ritchie 1988]. A word is a unit of space big enough to fit any vertex identifier or distance, and we have instructions like addition operating on a constant number of words in a single time unit. Thus, we measure *space* or *size* as the number of words used and *time* as the number of instructions performed.

Our results provide an efficient tool for navigating in planar networks, and should be of inherent interest to any lazy creature living on a sphere like the surface of planet Earth. For a list of potential applications of efficient (approximate) shortest path queries in planar digraphs, the reader is referred to Chen and Xu [2000]. Also, it should be mentioned that our approximate distance oracles for planar graphs can be used as input for sublinear approximation algorithms for general metrics [Indyk 1999]. In practice, we would have to extend our algorithms with heuristics to deal with exceptions such as bridges and no-left-turns, but this is beyond the scope of the current article.

1.1. RELATED WORK. Most of the previous oracles for reachability work for distances as well. Based on the separator theorems of Lipton and Tarjan [1979] and Frederickson [1987], Arikati et al. [1996] and Djidjev [1996] have presented distance oracles with a space-time product of $O(n^2)$. More precisely, if their distance oracle uses space s, they can answer distance/reachability queries in $O(n^2/s)$ time. Using the topology of planar graphs, Djidjev [1996] has shown that with space $s \in [n^{4/3}, n^{3/2}]$, the distance query time can be improved to $O((n/\sqrt{s})\log n)$. For space $s = O(n^{4/3})$, this gives a query time of $O(n^{1/3}\log n)$ and a space-time product of $O(n^{5/3}\log n)$. Very recently, Chen and Xu [2000] generalized Djidjev's bound so that for space $s \in [n^{4/3}, n^2]$, they get a distance query time of $O((n/\sqrt{s})\log(n/\sqrt{s}) + \alpha(n))$. We note that the extended range does not improve Djidjev's space-time product of $O(n^{5/3}\log n)$. For comparison, we note that our new space-time product for reachability is $O(n \log n)$.

There has also been related work on special classes of planar digraphs. In particular, for a planar s-t-graph, where all vertices are on dipaths between s and t, Tamassia and Tollis [1993] have shown that we can represent reachability in linear space, answering reachability queries in constant time. Also, Djidjev et al. [1991] have shown that if all vertices are on the boundary of a small set of f faces, there is an $O(n \log n + f^2)$-space distance oracle with query time $O(\log n)$. In the special case of outer-planar graphs ($f = 1$), they later [Djidjev et al. 1995] improved the oracle space to $O(n \log\log n)$ and the query time to $O(\log\log n)$. Finally, the above, mentioned result of Chen and Xu [2000] benefits from $f = o(n)$. More precisely, for any parameter $r \in [1, f]$, they combine a space of $O(n + f\sqrt{r} + f^2/r)$ with a query time of $O(\sqrt{r}\log r + \alpha(n))$.

1.2. UNDIRECTED GRAPHS. In the case of undirected graphs, we note that reachability is trivial in that each vertex just needs to remember what component it is in. For approximate undirected distances, we get slightly better results than we did in the directed case. More precisely, we provide an $O(n(\log n)/\varepsilon)$-space oracle

answering stretch-$(1+\varepsilon)$ distance queries in $O(1/\varepsilon)$ time. This undirected improvement was discovered independently by Klein [2002].

For exact distances in undirected planar graphs, nothing better than the directed results was known. However, Arkati et al. [1996] showed that using $O(n\sqrt{n})$ space, we can answer distance queries with stretch 2 in constant time. Also, Chen [1995] showed that using $O(n \log n)$ space, we can answer distance queries with stretch 3 in constant time. Our approximate distance oracle for undirected graph improves these bounds if, for example, we set $\varepsilon = 0.5$. Then, we get stretch-1.5 distances in constant time using $O(n \log n)$ space.

We note that for general sparse graphs, no non-trivial oracles are known for the directed case, even for reachability. However, for undirected weighted graphs, it has recently been shown by Thorup and Zwick [2001a] that one can construct a stretch-3 distance oracle in $O(n\sqrt{n})$ space with constant query time.

1.3. LABELING SCHEMES. A nice feature of our oracles is that they distribute perfectly into labeling schemes [Peleg 2000b]: in the case of reachability, each vertex v gets assigned an $O(\log n)$-space label $\ell(v)$, and given $\ell(v)$ and $\ell(w)$, but nothing else, we can compute if v reaches w in constant time. Similarly, for the approximate distances, we get labels of size $O((\log(nN))(\log n)/\varepsilon)$ so that given $\ell(v)$ and $\ell(w)$, the distance from v to w can be computed with stretch $(1+\varepsilon)$ in $O(\log\log(nN) + 1/\varepsilon)$ time. In the case of undirected graphs, the label size is reduced to $O((\log n)/\varepsilon)$ and the distance query time is reduced to $O(1/\varepsilon)$.

Labels for exact distances in planar graphs have been studied by Gavoille et al. [2001]. From the separator theorem of Lipton and Tarjan [1979], they get labels of size $O(\sqrt{n})$ supporting exact distance queries in $O(\sqrt{n})$ time. They provide a near-matching lower-bound showing that even for undirected graphs, we need $\Omega(\sqrt{n})$-bit labels to support exact distance queries, no matter the time available to compute the distances. In particular, this implies that our small approximate distance labels cannot be made exact even in the undirected case.

Of approximate distance labelings, Gupta et al. [2001] have shown that for undirected graphs, there are stretch-3 distance labels of size $O(\log n)$ with constant query time. We get much better bounds with our stretch-$(1+\varepsilon)$ distance labels of size $O((\log n)/\varepsilon)$ with query time $O(1/\varepsilon)$, for example, setting $\varepsilon = 0.1$.

1.4. NEW TECHNIQUE: DIPATH DECOMPOSITIONS. A technical novelty of this article is the concept of dipath decompositions. A *dipath decomposition* associates a small set $S(v)$ of dipaths with each vertex v in such a way that any dipath from a vertex u to a vertex w will intersect some dipath in $S(u) \cap S(w)$. Our dipath decompositions are compact in the sense that each set $S(v)$ is of logarithmic size.

To see the relation between dipath decompositions and reachability oracles, suppose for each $Q \in S(v)$, we store the number $\mathsf{to}_v[Q]$ of the first vertex in Q reachable from v. Similarly, $\mathsf{from}_v[Q]$ is the number of the last vertex in Q that can reach v. Then we can reach u from w if and only if there is a $Q \in S(u) \cap S(w)$ such that $\mathsf{to}_u[Q] \leq \mathsf{from}_w[Q]$.

1.5. NOTATION. If G is a digraph, $V(G)$ and $E(G)$ denote its vertex and edge set. Also, if S is a set of dipaths in G, $V(S)$ is the set of vertices appearing in these dipaths. We generally allow dipaths to be self-intersecting, unless we explicitly state that they are simple. Let u and v be vertices in G. Then, $u \rightsquigarrow_G v$ denotes that we can reach v from u in G, that is, that there is a dipath from u to v in G, and then

$\delta_G(u, v)$ is the distance from u to v, that is, the shortest length of such a dipath. The subscript $_G$ may be omitted if the graph G is clear from the context.

If W is a subset of the vertices in G, by $G \backslash W$, we denote G with the vertices in W and their incident edges deleted. By G/W, we denote G with all vertices in W contracted to a single vertex w, that is, w replaces all of W in the vertex set, and any edge end-point in W is replaced with w. Any self-loops created are removed.

If P is a dipath and U a subset of its vertices, by *P reduced to U*, we mean the dipath Q with vertex set U where the vertices appear in the same order as in P. If P is weighted, Q is weighted so as to preserve the distances from P between vertices in Q, that is, a segment of P between consecutive vertices from U is replaced by a single edge of the same length.

In this article, we will work with a directed graph G, but frequently, we get *disoriented,* forgetting the orientations in G, treating G as an undirected graph. For example, a disoriented path in G is a path in the underlying undirected graph.

If T is a rooted tree with a vertex v, by $T(v)$ we denote the path between the root and v, and we call $T(v)$ the *root path* of v.

2. *Reachability*

We will now describe our algorithm for constructing a reachability oracle for a planar digraph G. We are going to construct a series of "2-layered" digraphs $G_0, \ldots, G_k$ so that any reachability question in G can be addressed to a constant number of these digraphs, and such that each digraph G_i admits separators consisting of a constant number of dipaths. Here, a set of dipaths form a *separator* if removing all vertices in these dipaths splits the graph in components of at most half the size.

Below, we first give the definition and construction of the 2-layered G_i. Second, we describe the actual dipath separators. Third, we use them to form a dipath decomposition for reachability queries in logarithmic time. Fourth, we show how the query time can be tuned to be constant. Finally, we convert the oracle into a labeling scheme and show how to find and route along dipaths.

2.1. 2-LAYERED DIGRAPHS.

In this section, we will show how to reduce reachability in G to reachability in some digraphs with special properties. The reduction does not use planarity, but it does preserve planarity. The construction is illustrated in Figure 1.

Definition 2.1. A *t-layered* spanning tree T in a digraph H is a disoriented rooted spanning tree such that any path in T from the root is the concatenation of at most t dipaths in H. We say that H is t-layered if it has such a spanning tree.

LEMMA 2.2. *In linear time, given a digraph G, we can construct a series of digraphs $G_0, \ldots, G_{k-1}$ so that*

(i) *The total number of edges and vertices in all the G_i is linear in the number of edges and vertices in G.*

(ii) *Each vertex v has an index $\iota(v)$ such that a vertex w is reachable from v in G if and only if it is reachable from v in $G_{\iota(v)-1}$ or $G_{\iota(v)}$.*

(iii) *Each $G_i = (V_i, E_i)$ is a 2-layered digraph with a 2-layered spanning tree T_i with root r_i.*

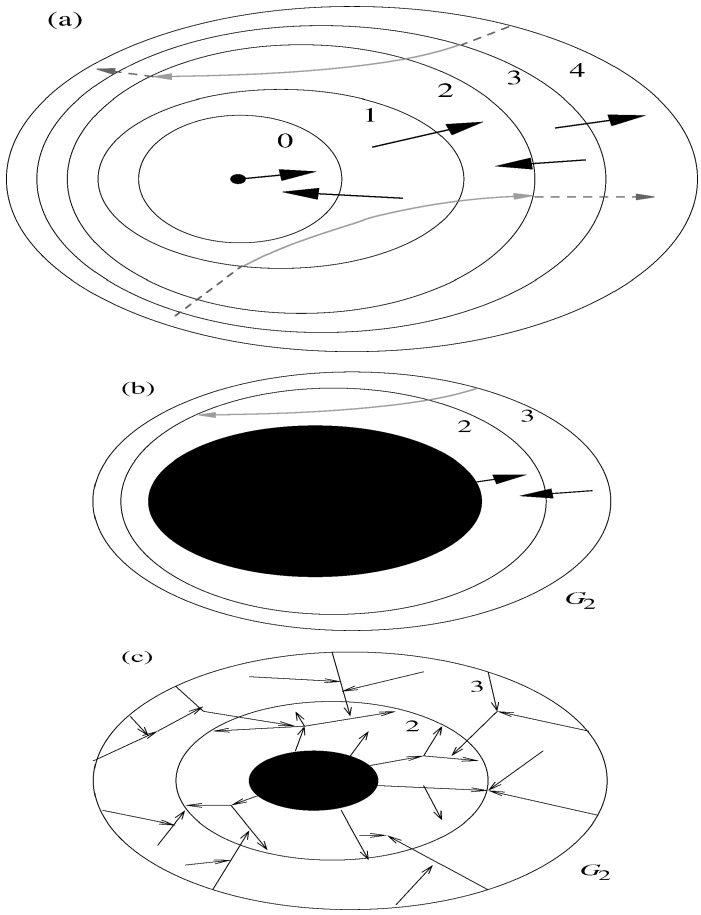

FIG. 1. (a) The partitioning of a graph into layers where each dipath is contained in at most two layers. (b) One of the two layered graphs, and (c) its two layered spanning tree.

(iv) *G_i is a* minor *of G, that is, G_i is obtained from G by deletion of edges and vertices and contraction of edges. In particular, if G is planar, so is G_i.*

PROOF. In our construction, we assume that G is connected in the undirected sense; otherwise, the construction is just applied independently to each connected component. We now first partition the vertices of G into layers $L_0, \ldots, L_{k-1}$ where L_0 is the set of vertices reachable from an arbitrary vertex v_0, and thereafter, alternating, a layer consists of all vertices reaching or reachable from the previous layers. More formally, for $i > 0$, we define:

$$L_i = \begin{cases} \{v \in V \backslash L_{<i} : v \rightsquigarrow L_{<i}\} & \text{if } i \text{ is odd} \\ \{v \in V \backslash L_{<i} : L_{<i} \rightsquigarrow v\} & \text{if } i \text{ is even.} \end{cases}$$

Here, $L_{<i}$ denotes $\bigcup_{j<i} L_j$. Similarly, $L_{\leq i}$ denotes $\bigcup_{j\leq i} L_j$, and we let k be the first index such that $L_{\leq k} = V$. We set $\iota(v) = i$ for all $v \in L_i$.

Each digraph G_i consists of two consecutive layers with all preceding layers contracted to a single vertex. More formally, the digraph G_i is obtained by taking

the subgraph of G induced by $L_{<i+1}$ and, for $i > 0$, contracting all vertices in $L_{<i}$ to the single root vertex r_i. For G_0, we set $r_0 = v_0$.

By construction, G_i satisfies (iv). Concerning (i), since the layering is a partitioning, each vertex occurs as a non-root at most twice. Thus, we have at most $2n + k = O(n)$ vertices in total. Also, we have a total of at most $2n$ edges incident to the roots, and besides that, each edge occurs at most twice, adding up to a total of $2n + 2m = O(m)$ edges, so (i) is satisfied.

To see that (ii) is satisfied, consider an arbitrary dipath P from a vertex v to a vertex w. We will show that it is contained in at most two layers. Let i be the lowest index of a layer intersected by P, and let x be a vertex in the intersection. By definition, if $j \geq i$ is even, $L_{\leq j}$ contains the part of P after x. Similarly, if $j \geq i$ is odd, $L_{\leq j}$ contains the part of P before x. Thus P is contained in $L_i \cup L_{i+1}$. It follows that P is contained in G_i. On the other hand, we know that $v \in P$ is only contained in $G_{\iota(v)-1}$ and $G_{\iota(v)}$, so we conclude that our dipath P from v to w is contained in one of these two digraphs. Thus, (ii) is satisfied.

Finally, to satisfy (iii), we need to construct the undirected spanning trees T_i. Suppose i is odd. By definition r_i reaches all of L_i, so a spanning tree U_i of $\{r_i\} \cup L_i$ can be constructed with all edges oriented away from r_i. Since $\{r_i\} \cup L_i$ is reached by all of L_{i+1}, U_i can be extended to a spanning tree of $\{r_i\} \cup L_i \cup L_{i+1} = V(G_i)$ with the new edges oriented towards $\{r_i\} \cup L_i$. Now any path in T_i from r_i has a first part oriented away from r_i and a second part oriented towards r_i, so (iii) is satisfied. The case where i is even is symmetric. This completes the proof of Lemma 2.2. □

Concerning the above proof, we note that division into layers L_i as well as the proof that each dipath is contained in only two layers stems from a previous article of the author [Thorup 1995]. However, having a quite different application in mind, that article proceeded with each layer independently instead of combining them two by two. Thus, it is a new idea to consider 2-layered digraphs containing all original dipaths.

Applying Lemma 2.2 to a planar digraph, for reachability, we can now assume that we are dealing with a 2-layered planar digraph, that is, we make a reachability oracle for each G_i and then we address reachability queries from v in G to $G_{\iota(v)-1}$ and $G_{\iota(v)}$.

2.2. UNDIRECTED PLANAR SPANNING TREE SEPARATION. In this section, we will take a planar digraph H with a 2-layered spanning tree T, but forget the orientation of the edges. We will just view H as an undirected graph H with a given rooted spanning tree T. Using just planarity, we will find three root paths in T whose removal separates the graph into components of at most half the size. The point is that these three paths correspond to at most six dipaths in the original digraph.

LEMMA 2.3. *In linear time, given an undirected planar graph H with a rooted spanning tree T and non-negative vertex weights, we can find three vertices u, v, and w such that each component of $H \setminus V(T(u) \cup T(v) \cup T(w))$ has at most half the weight of H.*

Essentially, the result is proved by Lipton and Tarjan [1979]. It is, however, hidden in other details because their target is to get separators with $O(\sqrt{n})$ vertices. Also, instead of three paths separating into components of $1/2$ the weight, they choose two root paths separating into components of $2/3$ the weight. We feel that

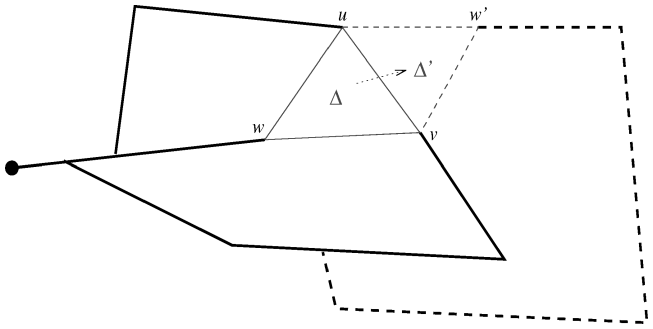

FIG. 2. The separator consists of paths to the root from the corners of some triangle. If that triangle doesn't work as a separator, there is a neighboring triangle doing a better job.

our three root paths are more natural, and they will give better constants in our recursions because $2\log_{3/2} n \approx 3.4 \log_2 n > 3 \log_2 n$.

Below, we present a self-contained proof of the existence of the three root paths. For the linear time construction, we refer to the techniques from Lipton and Tarjan [1979].

PROOF OF LEMMA 2.3. The proof idea is illustrated in Figure 2.

First, we embed the graph on a sphere and add auxiliary edges so as to triangulate H. Clearly, T is also a spanning tree for the resulting triangulated graph $H^{\triangleleft}$, and any separator for $H^{\triangleleft}$ is also a separator for H.

Consider any triangle Δ of $H^{\triangleleft}$. We now view Δ as the external face. The *fundamental cycle* of an edge in T is the cycle consisting of the edge and the unique path between its end-points in T. The *inside* of this cycle is the side that does not contain Δ. As a special case, we note that if $(v, w) \in T$, the inside is empty. Now, if u, v, and w are the corners of Δ, each component of $H^{\triangleleft} \setminus V(T(u) \cup T(v) \cup T(w))$ is contained inside the fundamental cycle of one of the boundary edges (u, v), (v, w), and (w, u).

Our goal is to identify a triangle Δ as an external face so that none of the fundamental cycles of the boundary edges of Δ have more than half the weight strictly inside. Here, by *strictly* inside, we mean that we do not count weights on vertices in the fundamental cycle itself. The triangle Δ is found as follows. We start with Δ an arbitrary triangle. If the fundamental cycle of one of the boundary edges (u, v) has more than half the weight strictly inside, we move to the triangle Δ' on the other side of (u, v).

To see that the above process terminates, we consider the number of triangles inside the fundamental cycle containing more than half the weight, if any. Before we moved our triangle from Δ to Δ', it was the fundamental cycle of (u, v) that contained more than half the weight. The move turned the cycle inside out in the sense that the new inside was the previous outside, and hence it now contains at most half the weight. However, if w' is the third corner of Δ', the new insides of the fundamental cycles of (u, w') and (v, w') are both contained in the previous inside of (u, v) minus the triangle Δ'. Thus, both have fewer triangles than the previous inside of (u, v).

The above construction is essentially contained in the proof of Lemma 2 in Lipton and Tarjan [1979], except that they stop moving the triangle as soon as no fundamental cycle of an edge of Δ contains more than $2/3$ of the weight.

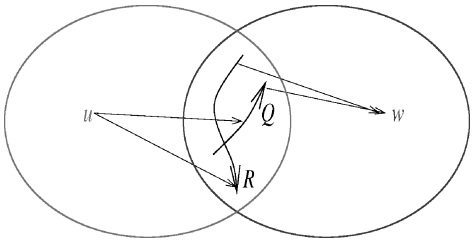

FIG. 3. The vertex u connects to Q earlier than the connection from Q to w, so u reaches w via Q. However, u connects too late in R for the connection to w, so u does not reach w via R.

Then, if (u, v) is the edge of Δ with most weight inside its fundamental cycle, the components of $H \setminus V(T(u) \cup T(v))$ are of at most $2/3$ the weight of H.

In Lipton and Tarjan [1979], it is shown how the above type of construction can be implemented in linear time. That is, first we embed and triangulate the graph. Next, we pick an arbitrary triangle Δ and find the weight inside the fundamental cycles of the boundary edges. Finally, we have a loop, moving Δ to the neighboring triangle Δ', each time finding the weight of the new inside of the fundamental cycle of each boundary edge. The details are described in Steps 1, 8, and 9 of the partitioning algorithm in Lipton and Tarjan [1979, Sect. 3]. □

2.3. REACHABILITY VIA A DIPATH. Our next step is to show that we can efficiently represent directed reachability via a separator dipath, as illustrated in Figure 3. Here, a vertex u reaches a vertex w *via* a dipath Q if there is a dipath from u to w which intersects Q. We are now forgetting that the digraph is planar. We note that the material in this section is very similar to the material in Subramanian [1993] on sparsifying intersection paths, though in Subramanian [1993], what is separated is vertices along the boundary of a single face.

Formally, we are given a directed graph H containing some directed path Q. We want to represent reachability via Q. For each vertex v, we wish to find the first vertex a in Q reached by v, and the last vertex b in Q that reaches v. We then say that v *connects* to a in Q, that v connects from b in Q, and that (v, a) and (b, v) are the connections between v and Q.

FACT 2.4. *There is a dipath from u to w intersecting Q if and only if u connects to a in Q and w connects from b in Q where a equals or precedes b in Q.*

We assume that the vertices in Q are numbered so that we can check precedence in constant time. Then, the above test takes constant time.

LEMMA 2.5. *For any digraph H and dipath Q, we can identify all connections between vertices in H and the dipath Q in linear time.*

PROOF. By symmetry, it suffices to identify the connections from Q. We use a simple recursive procedure. Let t be the last vertex in Q. Using, say, a breadth-first-search, we find all vertices v reached by t. By definition, all these vertices connect from t in Q. We now remove all these vertices including t from H and Q, and recurse on the remainder of H and Q. Each edge is only considered once, so the running time is linear.

To see that the recursion is correct, we need to argue (a) that Q remains a dipath, and (b) that we do not destroy any connections except for those from t. Here (a)

follows because if t reaches a vertex x in Q, t reaches all the successors of x in Q, and hence it is a suffix of Q that is removed. Also, concerning (b), if P was a dipath from (a vertex in) Q to a vertex v and a vertex from P is removed, then t reaches v. □

2.4. THE BASIC RECURSION. We are now going to combine the previous lemmas into a simple algorithm producing the reachability oracle with query time $O(\log n)$. In the next section, the query time will be reduced to a constant. First, we apply Lemma 2.2, reducing our problem to dealing with 2-layered digraphs. Consider any 2-layered digraph H with 2-layered rooted spanning tree T. We now apply Lemma 2.3, producing a separator S consisting of three root paths, corresponding to six separator dipaths in H. With unit vertex weights, each component C of $H \setminus V(S)$ has at most half the weight of H.

To each of the dipaths $Q \in S$, we apply Lemma 2.5 to produce all connections between H and Q. For each vertex v, we make two arrays to_v and from_v indexed by the separator dipaths such that $\mathsf{to}_v[Q]$ is the number in Q of the vertex that v connects to, and $\mathsf{from}_v[Q]$ is the number in Q of the vertex that v connects from. If v does not reach Q, $\mathsf{to}_v[Q] = \infty$, and if Q does not reach v, $\mathsf{from}_v[Q] = -1$. Then u reaches w via the separator S if and only if $\mathsf{to}_u[Q] \leq \mathsf{from}_w[Q]$ for some dipath Q in S. For given Q, this check takes constant time.

Since S consists of root paths, it is a connected part of the spanning tree T containing the root. To recurse, we contract S to a new root r^S. The resulting digraph H/S is a 2-layered digraph with spanning tree T/S rooted in r^S. The root r^S is *suppressed*, meaning that we are not interested in connections via r^S. We give r^S weight 0 whereas all other vertices get unit weight.

For each component C of $H \setminus V(S)$, we recurse on the subgraph H' consisting of C and the root r^S. Since the root has weight 0, the subgraph H' has only half the weight of H. We find a separator S' of H' using Lemma 2.3. However, since r^S is suppressed, we remove it from S' and H' when we find connections between the vertices in H' and the dipaths in S'. For the next level of recursion, the components C' of $H' \setminus V(S')$ are of at most half the weight of C, so the recursion depth is $O(\log n)$, and the total running time is $O(n \log n)$.

2.4.1. *Indexing the Separator Dipaths.* In the recursion tree, each vertex v participates in calls following a path from the root call down to the call where v is selected for the separator S, and we say that this is the *final* call of v. The vertex v enumerates, starting from 1, the dipaths in all the ancestor separators, starting with those in the root call and ending in the final call. These numbers are used as indices for to_v and from_v. The numbers are all $O(\log n)$ which is hence the size of the tables to_v and from_v. We let the dipaths in the separator of each call come in a fixed ordering respected by the enumeration done by each vertex participating in the call. Now, if Q is a separator dipath in a call involving both u and w, then u and w will use the same index for Q. To check reachability from u to w, we only need to consider calls involving both u and w, so if s is the number of separator dipaths in such calls, u reaches w in G_i if and only if for some $q \in \{1, \ldots, s\}$, $\mathsf{to}_u[q] \leq \mathsf{from}_w[q]$. Since $s = O(\log n)$, this check takes $O(\log n)$ time.

To find s, we store with each call C its *separation number*, defined as the total number of separator dipaths in all ancestor calls of C, including C. Then, s is the separation number of the nearest common ancestor call of the final calls of u and w. Since the depth of the recursion tree is logarithmic, finding the nearest common

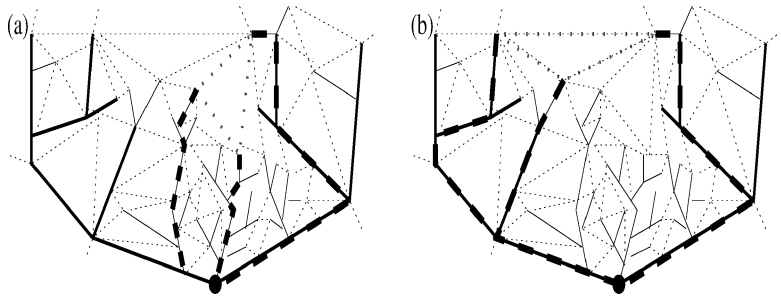

FIG. 4. In both figures, the solid lines are tree edges whereas dotted lines are non-tree edges. The thick solid lines show the current frame whereas the very thick dashed lines show the new separator. In (a), we have a subgraph reducing separator where each of the new subgraphs have few vertices. In (b), we have a frame reducing separator where each new subgraph gets a frame with few leaves.

ancestor is trivially done in $O(\log n)$ time, which is hence our total query time for reachability.

2.5. REDUCING THE QUERY TIME. We will now reduce the query time to constant. To achieve this, instead of contracting the separators as we recurse, we will pass down a set F of root paths separating a subgraph H from the rest of G_i, that is, H will be a component of $G_i \setminus V(F)$. We call F the *frame* of H. If S is a separator of H separating u from w, then $F \cup S$ separates u from w in G_i. For each dipath $Q \in F \cup S$, we will identify the connections between H and Q *over* G_i, meaning that $v \in V(H)$ connects to (from) the first (last) vertex in Q that v can reach in G_i. Keeping F and S of constant size, we will only have a constant number of separator and frame dipaths to query to check reachability from u to w.

We note that since F is a set of root paths, its union is a tree with the same root as our rooted spanning tree. Since we want F to have few root paths, we can assume that it consists of the root paths from the leaves of this tree. Additional root paths in F would be redundant in that they would not affect $V(F)$. With this view in mind, we call the starts of root paths in F the *leaves* in F.

2.5.1. *Few Frame Paths.* In order to keep the frames of constant size, we will alternate between two types of recursion, namely, *subgraph reducing* and *frame reducing* recursions, both illustrated in Figure 4.

For both recursions, we apply the technique from Section 2.2 to $H + F$, denoting H and F together with all edges from G_i between H and F. In the subgraph reducing recursion, we pick the separator $S = \{T(u), T(v), T(w)\}$ so that no component of $(H + F)\setminus V(S)$ contains more than half the vertices from H. Thus, when applying Lemma 2.3 to $H + F$, the vertices in H get weight 1 whereas vertices in F get weight 0. In the frame-reducing recursion, we pick S so that no component of $(H + F)\setminus V(S)$ contains more than half the leaves of F. Thus, when applying Lemma 2.3 to $H + F$, these leaves get weight 1 whereas all other vertices get weight 0. Since every other recursion halves the weight of the subgraph H, we still only have logarithmic recursion depth.

We will now consider the number of root paths in the frames passed down to the recursions on the components of $H\setminus V(S)$, starting with a subgraph reducing recursion. If the our original frame F has f root paths, then trivially, each component receives a frame with at most $f+3$ root paths; namely, those from F plus the three in S.

Consider now a frame reducing recursion. We will always pass S down as a frame, but thereby, we can avoid passing down some root paths from F. Consider a root path $R \in F$ with a leaf x. Note that as soon as the path R intersects S on its way to the root, the rest of R is contained in S. In particular, if $x \in S$, the path R is subsumed by S. Hence, it is only components of $H \setminus V(S)$ containing x that need R in their frame. If f' is the current number of paths F, our choice of S is such that no component of $H \setminus V(S)$ contains more than $f'/2$ leaves from F. Hence, including the at most three root paths from S, we pass at most $f'/2 + 3$ frame paths down to each of the child recursions.

In the root call on G_i, the frame is empty. Generally, starting from a subgraph reducing recursion, we go from f to at most $f' = f + 3$ to at most $(f+3)/2 + 3 = f/2 + 4.5$ frame paths for a next subgraph reducing recursion. Hence, the maximal number of frame paths for a subgraph reducing recursion is $f \leq 9$, and for a frame reducing recursion, it is $f' \leq f + 3 \leq 12$. Each of these frame paths corresponds to at most 2 dipaths in the frame.

2.5.2. *Reachability via Framed Separators.* In order to answer reachability queries in G_i, we need all connections over G_i between H and dipaths in the frame F and in the separator S. We assume that all connections in G_i between H and dipaths in the frame are passed down in the recursion, and we let $H \star F$ denote $H + F$ including these connections as edges. Then, for vertices $u, w \in V(H)$, u reaches w in G_i if and only if it does so in $H \star F$. Note that $H \star F$ is typically nonplanar. Also, note that the number of these extra edges is linear in the number of vertices in H since each vertex in H has only one connection from and one connection to each of the constant number of dipaths in F.

We want to reduce $H \star F$ so that its size is linear in the number of edges in G_i incident to vertices in H. A vertex in F is *topological* if it is an end-point of a dipath in F, or a branching point between dipaths in F. Note that there are only a constant number of topological vertices. A vertex in F is *selected by* H if it is neighbor to a vertex of H in $H \star F$. We now skip all vertices in F that are neither topological nor selected, that is, if we have a segment of a dipath, all of whose interior vertices are not topological or selected, we contract the segment to a single edge. With F thus reduced, we apply Lemma 2.5 to $H \star F$ to find all connections in G_i between vertices in H and dipaths in S.

The most efficient way to construct the reduced frame, skipping all vertices that are neither topological nor selected, is to construct it as we recurse. Thus, before we can recurse, for each of the components H' of $H \setminus V(S)$, we need to construct the reduced frame F'. We do this in two phases. In the first phase, for each component H', we take the dipaths and vertices from F and S needed for F' and record with these dipaths and vertices that they are relevant to H'. We now take one dipath Q from $F \cup S$ at a time, and register its starting point with all components H' for which it is relevant. We then traverse Q. Whenever we arrive a vertex $v \in Q$, we consider the components H' it is relevant for. Each such component H' has registered its last relevant vertex u from Q and hence we can add the arc (u, v) to its reduced representation Q' of Q. All in all, the processing time is linear in the size of $H \star F$, so the total running time over the whole recursion is still $O(n \log n)$.

2.5.3. *Indexing with Frames.* In order to benefit from the frames, when enumerating dipaths from a vertex v, in each ancestor call of the final call of v, we enumerate both the dipaths in the frame and the dipaths in the separator. This

typically means that the same dipath gets numbered several times, first as a separator, and later as a frame. The separation number of a call is again the last number used for enumerating dipaths for that call. Now, let C be the nearest common ancestor of the final calls of u and w, let s be the separation number of C, and let p be the separation number of the parent of C. Then $\{p+1, \ldots, s\}$ are the indices of the frame and separator dipaths of C, so u reaches w in G_i if and only if there exists a $q \in \{p+1, \ldots, s\}$ such that $\mathsf{to}_u[q] \leq \mathsf{from}_w[q]$.

Applying Harel and Tarjan's [1984] linear time preprocessing to the recursion tree, we can find the nearest common ancestor call C in constant time, and thereby we also get p and s. Thus, the reachability query takes constant time. We conclude:

THEOREM 2.6. *We can preprocess a planar digraph in $O(n \log n)$ time and space, producing an $O(n \log n)$-space reachability oracle that, given any two vertices u and w, can determine if u reaches w in constant time.*

2.6. A PURE LABEL-BASED IMPLEMENTATION. We now show that our reachability oracle can be distributed as a labeling scheme, associating with each vertex v a label $\ell(v)$ of size $O(\log n)$, so that given $\ell(u)$ and $\ell(w)$, and nothing else, we can determine if u reaches w.

As the first contribution to our label for v, we have the $O(\log n)$-space tables from_v and to_v. At present, the separation numbers are stored globally with the calls in the recursion tree, but instead, for each vertex v and depth d, we store the separation number $s_v[d]$ of the depth d ancestor of the final call of v in the recursion tree. All that remains is to find a labeling for depths of nearest common ancestors. Peleg [2000a, Sect. 3] has presented such a scheme, but with $O(\log n)$ query time. To get constant query time, we simply translate the technique of Harel and Tarjan [1984] into a labeling scheme. We observe that whenever they access global information, it is associated with an ancestor in a tree of $O(\log n)$ height. If we copy this ancestor information down to each descendant, we get the desired label of $O(\log n)$ space. Globally, this increases the space for nearest common ancestors from linear to $O(n \log n)$, but our global space was $O(n \log n)$ anyway.

THEOREM 2.7. *The oracle of Theorem* 2.6 *can be distributed as a labeling scheme with labels of size $O(\log n)$.*

2.7. FINDING THE DIPATHS. If we find that u reaches w, the next natural task is to produce a simple dipath from u to w in time linear in its length. Below, we will first show how to do this for the basic recursion, and second, show how this extends for the tuned query time.

2.7.1. *Path Finding in the Basic Recursion.* In the basic recursion from Section 2.4, if u reaches w, our query algorithm finds an index q of a separator dipath Q of a digraph H such that (a) the connection via H from u to Q precedes the connection via H from Q to w, and (b) all dipaths from u to w in H are also dipaths in the original digraph G.

We will now extend Lemma 2.5 to provide dipaths witnessing the connections between H and Q. What the algorithm in the proof of Lemma 2.5 really finds is a forest of disjoint from-trees oriented away from roots in Q such that a is the root of v if and only if (a, v) is the connection from Q to v. Symmetrically, we get a forest of to-trees oriented towards roots in Q, witnessing connections from H to Q.

Using q as an index, we save the parent pointers of the to- and from-trees. More precisely, we let $\vec{\mathsf{to}}_v[q]$ store the parent of v in the to-trees generated from Q. If v is a root, $\vec{\mathsf{to}}_v[q] = \mathsf{nil}$, and if v does not reach Q, $\vec{\mathsf{to}}_v[q]$ is undefined. Similarly, we let $\overleftarrow{\mathsf{from}}_v[q]$ store the parent of v in the from-trees generated from Q. Note that these parent pointers do not affect our asymptotic space bounds.

Given the index q with $\mathsf{to}_u[q] \leq \mathsf{from}_w[q]$, we can follow the parent pointers $\vec{\mathsf{to}}_v[q]$ tracing a dipath P_1 from u to $a \in Q$, and follow the parent pointers $\overleftarrow{\mathsf{from}}_v[q]$ tracing backwards a dipath P_2 from $b \in Q$ to w where b equals or succeeds a in Q. If Q_0 is the segment of Q from a to b, $P_1 Q_0 P_2$ is a dipath from u to w that we find in time linear in its length.

The above dipath may have loops, but we can generate it without loops, yet in time proportional to the length of the resulting simple dipath. First, assume that $P_1 Q_0$ and $Q_0 P_2$ are both simple. We can then use the standard technique of tracing forwards along $P_1 Q_0$ and backwards along $Q_0 P_2$. The tracings are done in parallel. We stop as soon as one trace hits the other, discarding the remainder of the other trace. Since the traces are developed at the same speed and we keep all of the hitting trace, the discarded trace segment is no bigger the kept trace. The combined kept trace forms is a simple dipath from u to w, constructed in time proportional to its length.

However, Q_0 may have multiple intersections with P_1 and P_2. To deal with this, we need to do something more complicated. We trace forwards along P_1 keeping track of the earliest point q_1 seen in Q. In parallel, we trace backwards along P_2 keeping track of the latest point q_2 seen in Q. We stop tracing as soon as $q_1 \leq q_2$. Clearly, this will happen, for if we went to the ends, we would have $q_1 = \mathsf{to}_u[q] \leq \mathsf{from}_w[q] = q_2$. We also stop if one trace intersect the other.

Suppose we stop because the P_1 trace hits an earlier point $q_1 \leq q_2$. Instead of continuing along P_1, we continue the trace along Q until we hit the trace of P_2, discarding the rest of the P_2 trace. Similarly, if it is the P_2 trace that hits a later point $q_2 \leq q_1$, we continue backwards along Q until we hit the P_1 trace, discarding the rest of the P_1 trace. In all cases, the combined kept trace forms a simple dipath constructed in time proportional to its length.

2.7.2. *Path Finding with Constant Query Time.* We will now show how to find witnessing dipaths in combination with the constant query time from Section 2.5. Our basic problem is that condition (b) from Section 2.7.1 is not satisfied. More precisely, if u reaches w, our query algorithm returns an index q of a separator dipath Q of a digraph $H \star F$ such that (a) the connection via $H \star F$ from u to Q precedes the connection via $H \star F$ from Q to w. As in Section 2.7.1, this means that we can find a dipath P in $H \star F$ from u to v via Q. However, $H \star F$ and P may contain edges that are not edges in the original digraph. We could, of course, recursively expand these connections into dipaths in the original digraph, but for the sake of later routing issues, we do something different.

Our basic idea is this: when recursing on H with frame F and separator S, besides making *global* connections over $H \star F$ between $Q \in F \cup S$ and H, we make *local* connections over H between dipaths $Q \in S$ and H. The local connections over H can be turned into dipaths using parent pointers as described under the basic recursion. Thus, if u reaches w in G_i, our goal is to compute $\mathsf{sep}(u, w)$ denoting (an index of) a separator dipath Q of an ascending digraph H such that there is a dipath from u to w in H which intersects Q. Here, by ascending, we mean that H is from a

call ascending from the final calls of u and w. The indexing of separator dipaths for the local connections is done exactly as in the basic recursion in Section 2.4.1. Then $\mathsf{sep}(u, w)$ can play the role of q in the above dipath finding for the basic recursion in Section 2.7.1. We are going to compute $\mathsf{sep}(u, w)$ based on information stored with the global connections found by our reachability query.

In our recursion, we are going to label all edges and global connections with their sep-values. When we recurse with $H \star F$, we assume that we have the labels for all edges (x, y) that are not in H. When we pick a separator S, we index the separator dipaths in some order, as described in Section 2.4. We then go through the dipaths $Q \in S$ in this order, and take all edges incident to Q, including those in Q, that have not been labeled, and label them with the index of Q. We now construct the global connections over $H \star F$ between Q and H. These global connections are witnessed by to- and from-trees in $H \star F$. A global connection (v, a), $a \in Q$, is now labeled with the smallest label on the path from v to its root a in the to-trees. Similarly, (b, v), $b \in Q$, is labeled with the smallest label on the path from v to its root b in the from-trees. Processing the to- and from-trees top-down, we label the global connections in time linear in the sizes of the trees, hence in $O(n \log n)$ total time.

To see that the above labeling of global connections with sep-values is correct, consider a dipath P in the to-tree in $H \star F$ witnessing a global connection from v to a. Since all edges incident to Q have labels, P contains at least one label, and the smallest label is transfered to (v, a). Let (x, y) be an edge in P with smallest label q_0. Then this label is the index of a separator dipath Q_0 of an ascending digraph H_0 with a dipath P_0 in H_0 which intersects Q_0. Then H_0 contains H, so all unlabeled edges in P are in H_0. Also, all labeled edges in P have labels at least as high as q_0. However, the higher the index of separator dipath, the younger a digraph it separates, so all other edges in P have their endpoints connected by dipaths in digraphs equal to or descending from H_0, hence contained in H_0. Concatenating all these edges and dipaths, including the above P_0 from x to y, we get a dipath from v to a in H_0 which intersects Q_0, as desired.

We now return to a dipath query. When asking if u reaches w, a positive answer provides the index of a separator dipath Q and global connections (u, a) and (b, w), $a, b \in Q$, with a equal to or preceding b in Q. We now claim that we can set

$$\mathsf{sep}(u, w) = q_0 = \min\{\mathsf{sep}(u, a), \mathsf{sep}(b, w)\}. \tag{1}$$

The only surprising thing in (1) is that we do not need to consider sep-values of the edges in the segment of Q between a and b, that is, clearly there will be an ascending digraph H_0 with separator dipath Q_0 indexed q_0 such that H_0 contains dipaths from u to a and from b to w, and such that one of these dipaths intersects Q_0. However, we need to argue that H_0 contains the segment of Q from a to b. Suppose for a contradiction that some strict ancestor of H_0 had a separator path R intersecting this segment of Q_0. Since any separator path is a root path, R would contain either a or b, contradicting that these are both in H_0. Thus, there is a dipath in H_0 from u to a to b to w which intersects Q_0, as desired.

Storing with each vertex the sep-values of the connections to and from each ascending separator dipath, we can compute (1) in constant time, and subsequently, we can return a simple dipath from u to w in time linear in its length using the method from the basic recursion in Section 2.4.

2.8. ROUTING. Our basic routing model is as follows: When a packet arrives a node that is not the destination, the node decides which outgoing link to forward the packet onto based on a local routing table and information stored in the header of the packet. The header is computed in an initial hand-shaking process based on the labels of the source and the destination. Typically, we imagine that many packets are going to be sent between the source and the destination, so we do not worry too much about the details of the hand-shaking. For example, it could be done via some bigger servers knowing the labels of all nodes.

As described in Section 2.7.2, we can assume we are dealing with the basic recursion. First we give a high level description of the routing, and subsequently discuss the details of what information goes in the labels, the header, and the local routing tables. Our target is to get a packet from u to w following the dipath described in Section 2.7.1.

Using the labels from Section 2.6 as a starting point, the hand-shaking can compute the index q of a separator dipath Q between u and w in constant time. The index q is placed in the header of packets from u to w, and all the information we describe below is implicitly indexed by q. That is, we now only talk about a single to-tree, but really, for each vertex v, the parent pointer is the entry $\overrightarrow{\mathsf{to}}_v[q]$ in the array from Section 2.7 of parent pointers indexed by the $O(\log n)$ separator dipaths relevant to v. Also, we have access to the pointers along Q.

With the above pointers, we can move towards the root of the to-tree and continue along Q. We want to stop as soon as we reach an ancestor x of w in the from-tree. Here we use the idea of interval routing [Santoro and Khatib 1985; van Leeuwen and Tan 1986]. We enumerate the from-tree in depth-first-search order, and store with each node the first and last depth-first-search number of a descendant. We can then, in constant time, determine if one vertex is a descendant of another in the from-tree.

Supposing that w is a descendant of v in the from-tree, we have to figure out which child of v to go to. This could be done by searching among the depth-first-search numbers of the children of v, but for a high-degree node this would not be efficient. Instead we use the scheme described under "Fine tuning" in Section 2.1 in Thorup and Zwick [2001b]. With each node, we store a tree routing table of constant size and a tree destination label of $O(\log n)$ size. The tree destination label of w should be transfered to the header, and then we can find the appropriate child of v in the from-tree in constant time.

We will now add up the space in the labels, headers, and routing tables. From 2.6, we have $O(\log n)$-space labels. In addition, the label of a vertex should contain, for each of the $O(\log n)$ separator dipaths, its depth-first-search number and its tree destination label, adding up to $O((\log n)^2)$ space. In the header of a packet, we put the index q and the depth-first-search number and destination label of w associated with q, so the header takes $O(\log n)$ space. Finally, for each of the $O(\log n)$ relevant separator dipaths, the routing table of v consists of the parent pointer of the to-tree, the first and the last depth-first-search number descending from v in the from-tree, and the $O(1)$ size tree routing table for v in the from-tree. Hence, the routing table is of $O(\log n)$ size. With the above information, the hand-shaking and forwarding takes constant time. In Gupta et al. [2003], it is shown how some of the routing ideas of this article can be implemented with a current router technology.

3. *Approximate Distances*

In this section, we will generalize the approach from the previous section to give approximate distances. As a very first step, we note that in order to identify zero-distances, we can just use a reachability oracle for the subgraph with all the zero-weight edges. Thus, we only need to worry about approximating positive distances. For these, we will use the general idea of scaling. A *scale-*(α, ε) *distance oracle* approximates distances below α with an absolute error below $\varepsilon\alpha$. Moreover, it does not underestimate any distance. Later, we will combine such oracles with exponentially increasing values of α in a stretch-$(1+\varepsilon)$ distance oracle for arbitrary distances.

3.1. $(3, \alpha)$-LAYERED DIGRAPHS.

Definition 3.1. A (t, α)-*layered* spanning tree T in a digraph H is a disoriented rooted spanning tree such that any path in T from the root is the concatenation of at most t shortest dipaths in H, each of length at most α. We say that H is (t, α)-layered if it has such a spanning tree.

LEMMA 3.2. *In linear time, given a digraph G and a scale α, we can construct a series of digraphs $G_1^\alpha, \ldots, G_k^\alpha$ satisfying* (*i*) *and* (*iv*) *from Lemma* 2.2 *and such that*

(*ii*′) *Each vertex v has an index $\iota(v)$ such that a vertex w is at distance $d \le \alpha$ from v in G if and only if d is the smallest distance from v to w in $G^\alpha_{\iota(v)-2}$, $G^\alpha_{\iota(v)-1}$, and $G^\alpha_{\iota(v)}$.*

(*iii*′) *Each $G_i^\alpha = (V_i, E_i)$ is a $(3, \alpha)$-layered digraph with a $(3, \alpha)$-layered spanning tree denoted T_i^α with a root denoted r_i.*

PROOF. The construction is very similar to that of Lemma 2.2. Again, we first partition the vertices of G into layers $L_0, \ldots, L_k$. This time, L_0 is the set of vertices reachable within distance α from v_0 and for $i > 0$, we define:

$$L_i = \begin{cases} \{v \in V \backslash L_{<i} : \delta(v, L_{<i}) \le \alpha\} & \text{if } i \text{ is odd} \\ \{v \in V \backslash L_{<i} : \delta(L_{<i}, v) \le \alpha\} & \text{if } i \text{ is even.} \end{cases}$$

Above, $\delta(v, L_{<i})$ ($\delta(L_{<i}, v)$) is the shortest distance from (to) v to (from) a vertex in $L_{<i}$. Again, we define k as the first index such that $L_{\le k} = V$, and $\iota(v) = i$ for $v \in L_i$.

This time, each digraph G_i^α consists of three consecutive layers with all preceding layers contracted to a single vertex. More formally, the digraph G_i^α is obtained by taking the subgraph of G induced by $L_{\le i+2}$ and, for $i > 0$, contracting all vertices in $L_{<i}$ to the single root vertex r_i. We define $r_0 = v_0$.

The proof of (i) and (iv) are as in Lemma 2.2. Also, the proof for (iii′) is very similar, except that in the construction of T_i^α, for $j = i, i+1, i+2$, we use shortest dipaths from (to) $L_{<j}$ to (from) the vertices in L_j when j is even (odd). For $i > 0$, the root is suppressed for connections, as in Section 2.4. The suppression can be implemented by orienting all edges incident on the root towards (away from) the root if i is odd (even).

To see that (ii′) is satisfied, consider an arbitrary dipath P from a vertex v to a vertex w of length at most α. Consider the innermost layer L_i containing a vertex x from P. By definition of the layers, if $j > i$ is even, $L_{\le j}$ contains the part of

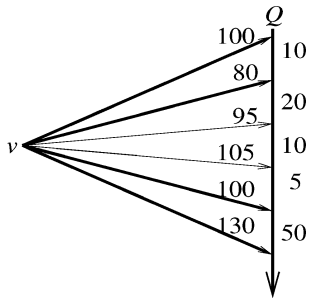

FIG. 5. The figure shows distances from v to vertices in the separator dipath Q. Also, it shows edge lengths in Q. The bold arrows from u to Q indicate connections that together with Q provide distances that are at most 5 bigger than the original distances.

P after x. Similarly, if $j > i$ is odd, $L_{\leq j}$ contains the part of P before x. This is as in the proof of Lemma 2.2, except that there we could consider $j = i$, so we needed one fewer layers. It follows that P is contained in $L_i \cup L_{i+1} \cup L_{i+2}$, hence that P is contained in G_i^α. On the other hand, we know that $v \in P$ is only contained in $G^\alpha_{l(v)-2}$, $G^\alpha_{l(v)-1}$, and $G^\alpha_{l(v)}$, so we conclude that our dipath P from v to w is contained in one of these three digraphs. Thus, (ii′) is satisfied. □

Given a $(3, \alpha)$-layered digraph with its $(3, \alpha)$-layered rooted spanning tree T_i^α, we can use T_i^α to find separators as in Section 2.2. This time, each root path is the concatenation of up to 3 shortest dipaths, each of length at most α.

3.2. REPRESENTING APPROXIMATE DISTANCES VIA A DIPATH. We are given a digraph G_i^α containing a shortest dipath Q from s to t whose length is at most α. We want to represent distances $\leq \alpha$ via Q, accepting an additive error of $O(\varepsilon\alpha)$ for a given $\varepsilon \in (0, 1]$.

We note that a similar undirected problem has been studied in Klein and Subramanian [1998], but for undirected approximate distances, one can just consider connections via $1/\varepsilon$ equally spaced points in Q. The directed case is much more complicated. Below, we first show how to represent approximate directed distances via Q. The difficult part is to construct such a representation efficiently, and we defer this to Section 3.4.

3.2.1. *Approximation Connections to Q.* When we have a vertex v and a dipath Q, we may need several connections from v to Q in order to get good approximations of distances from v to Q. This issue is illustrated in Figure 5. More precisely, a *connection* from v to Q is a new edge $(v, a) \in \{v\} \times V(Q)$. In this section, its length is always equal to the distance, that is, $\ell(v, a) = \delta(v, a)$. In our later efficient construction, some lengths will be longer.

If a equals or precedes b in Q and v is a vertex, we say a connection (v, a) *ε-covers* (v, b) if $\ell(v, a) + \delta(a, b) \leq \delta(v, b) + \varepsilon\alpha$. Since Q us a shortest path, $\delta(a, b)$ is the distance from a to b in Q. We say a set $C(v, Q)$ of connections from v to Q is *ε-covering* if it ε-covers all pairs in $\{v\} \times V(Q)$ of distance at most α. Paraphrasing this definition, we have

FACT 3.3. *Let x be a vertex in Q of distance at most α from v. If $C(v, Q)$ is* ε-covering, *there is a connection $(v, a) \in C(v, Q)$ such that $\ell(v, a) + \delta_Q(a, x) \leq \delta(v, x) + \varepsilon\alpha$.*

Next, we want to argue that small ε-covering sets exist.

LEMMA 3.4. *There is an ε-covering set $C(v, Q)$ size at most $\lceil 2/\varepsilon \rceil$.*

PROOF. We take the first vertex a_0 in Q that is within distance α from v, and make (v, a_0) the first connection in $C(v, Q)$. We now scan the remaining vertices b in Q. If (v, a) is the last connection added, we add (v, b) to $C(v, Q)$ if it is not ε-covered by (v, a), that is, if $\ell(v, a) + \delta(a, b) > \delta(v, b) + \varepsilon\alpha$.

We want to argue that the set $C(v, Q)$ is small. Let t be the last vertex in Q. We study the quantity $\ell(v, a) + \delta(a, t)$ where (v, a) is the last connection added. When we add the first connection (v, a_0), the quantity is $\ell(v, a_0) + \delta(a_0, t) \leq 2\alpha$. Also, when we add (v, b) to $C(v, Q)$, we decrease the quantity by $(\ell(v, a) + \delta(a, t)) - (\ell(v, b) + \delta(b, t)) = \ell(v, a) + \delta(a, b) - \delta(v, b) > \varepsilon\alpha$. Since the quantity cannot turn negative, we conclude that at most $\lceil 2/\varepsilon \rceil$ connections are added. □

3.2.2. *Approximate Distances via Q.* Connections from Q to v are defined symmetric to the connections from v to Q. In particular, if (b, v) is a connection and a precedes b in Q, then (b, v) ε-covers (a, v) if $\delta(a, b) + \ell(b, v) \leq \delta(a, v) + \varepsilon\alpha$. Formally, we can revert the orientation of all edges in our graph, apply the definition and construction of connections from v to Q, and then revert the orientation of the connections to get connections from Q to v. Generally, we will just talk about connections in one direction, letting the other direction be understood implicitly.

If (u, a) and (b, w) are connections from u to Q and from Q to w, we define

$$\mathsf{dist}((u, a), (b, w)) = \ell(u, a) + \delta_Q(a, b) + \ell(b, w).$$

In order to compute $\delta_Q(a, b)$ efficiently, we store with each vertex $c \in Q$, its distance $d(c, Q)$ from the start of Q as well as its number $i(c, Q)$ in Q. Then a equals or precedes b in Q if $i(a, Q) \leq i(b, Q)$, and then $\delta_Q(a, b) = d(b, Q) - d(a, Q)$. We note that we cannot use $d(a, Q) \leq d(b, Q)$ to determine if a equals or precedes b as there may be zero-weight edges in Q. If $i(a, Q) > i(b, Q)$, we have $\delta_Q(a, b) = \infty$.

If $C(u, Q)$ and $C(Q, w)$ are sets of connections from u to Q and from Q to w, we define

$$\mathsf{dist}(C(u, Q), C(Q, w)) = \min_{(u,a)\in C(v,Q),(b,w)\in C(Q,w)} \mathsf{dist}((u, a), (b, w)).$$

If there are no connections from u or to w, $\mathsf{dist}(C(u, Q), C(Q, w)) = \infty$. Thus, we always have $\delta(u, w) \leq \mathsf{dist}(C(u, Q), C(Q, w))$.

LEMMA 3.5. *Suppose the shortest dipath from u to w is of length at most α and intersects Q. If $C(u, Q)$ and $C(Q, w)$ are ε-covering,* $\mathsf{dist}(C(u, Q), C(Q, w)) \leq \delta(u, w) + 2\varepsilon\alpha$.

PROOF. Let x be a vertex in which a shortest dipath from u to w intersects Q. Then $\delta(u, x) \leq \alpha$, so by Fact 3.3, there is a connection $(u, a) \in C(u, Q)$ such that $\ell(v, a) + \delta_Q(a, x) \leq \delta(u, x) + \varepsilon\alpha$. Symmetrically, there is a connection $(b, w) \in C(Q, w)$ such that $\delta_Q(x, b) + \ell(b, w) + \leq \delta(x, w) + \varepsilon\alpha$. Now,

$$\begin{aligned} \mathsf{dist}(C(u, Q), C(Q, w)) &\leq \ell(v, a) + \delta_Q(a, x) + \delta_Q(x, b) + \ell(b, w) \\ &\leq \delta(u, x) + \delta(x, w) + 2\varepsilon\alpha \\ &= \delta(u, w) + 2\varepsilon\alpha. \end{aligned}$$

□

We say that $C(v, Q)$ is *clean* if all connections are significant for distances from v to Q, that is, if it contains no two distinct connections (v, a) and (v, b) such that $\ell(v, a) + \delta_Q(a, b) \leq \ell(v, b)$. Also, $C(v, Q)$ is *ordered* if the connections are ordered according to the ordering of the vertices in Q. It is easy to see that the set $C(v, Q)$ constructed by Lemma 3.4 is clean and ordered.

LEMMA 3.6. *If $C(u, Q)$ and $C(Q, w)$ are clean and ordered, we can determine* $\mathsf{dist}(C(u, Q), (Q, w))$ *in $O(|C(u, Q)| + |C(Q, w)|)$ time.*

PROOF. We just merge the two sets in linear time, resolving ties in favor of $C(u, Q)$, that is, if $(u, c) \in C(u, Q)$ and $(c, w) \in C(Q, w)$, we put (u, c) in front of (c, w). We seek the pairs $(u, a) \in C(u, Q)$ and $(b, w) \in C(Q, w)$ minimizing $\mathsf{dist}((u, a), (b, w))$. However, since $C(u, Q)$ and $C(Q, w)$ are clean, (u, a) and (b, w) must be consecutive in the merged list, and hence we can find them in a linear time traversal of the merged list. □

Combining Lemma 3.4, Lemma 3.5, and Lemma 3.6, we get.

LEMMA 3.7. *Given a shortest dipath Q of length at most α, we can find $O(1/\varepsilon)$ connections for each vertex such that if there is a shortest path from u to w of length at most α that intersects Q, then we can compute the distance from u to w in $O(1/\varepsilon)$ time with an additive error of $O(\varepsilon\alpha)$.* □

3.3. EXISTENCE OF APPROXIMATE DISTANCE ORACLES. First, we want to construct a scale-$(\alpha, O(\varepsilon))$ distance oracle approximating distances below α with an additive error of $O(\varepsilon\alpha)$. Essentially, we just apply Lemma 3.7 to each dipath in our reachability construction, replacing the previous reachability connections with our new approximation connections. More precisely, on each $(3, \alpha)$-layered graph G_i^α, we run the recursion with framed separators from Section 2.5. In each recursive call, we have a subgraph H of G_i^α, a frame F, and a separator S such that if u and w are in different components of $H \setminus S$, then any path from u to w in G_i^α intersects $F \cup S$. Now, $F \cup S$ consists of a constant number of root paths in the $(3, \alpha)$-layered spanning tree T_i^α, hence a constant number of dipaths of length at most α. We apply Lemma 3.7 to each of these dipath, hence using $O(1/\varepsilon)$ connections per vertex. Also, consulting each of these dipaths, we approximate the distance from u to w in $O(1/\varepsilon)$ time with an additive error of $O(\varepsilon\alpha)$. Thus, relative to our original reachability construction, our space and query time blows up with a factor $O(1/\varepsilon)$. Adjusting ε by a constant factor, we get.

LEMMA 3.8. *We can construct a $O(n(\log n)/\varepsilon)$ space scale-(α, ε) distance oracle approximating distances in $O(1/\varepsilon)$ time.*

In order to get a stretch-$(1 + \varepsilon)$ distance oracle, for $i = 1, \ldots, \lceil \log_2(nN) \rceil$, $\alpha = 2^i$, and $\varepsilon' \in \{1/2, \varepsilon/4\}$, we construct a scale-$(\alpha, \varepsilon')$ distance oracle as in Lemma 3.8. We use $\varepsilon' = 1/2$ with constant query time to quickly approximate distances within a constant factor. More precisely, to approximate the distance from u to w, we first make a binary search over the $\lceil \log_2(nN) \rceil$ values of i, looking for a value of $\alpha = 2^i$ such that $\alpha/4 < \hat{\delta}^{(\alpha, 1/2)}(u, w) < \alpha$. Here $\hat{\delta}^{(\alpha, \varepsilon')}$ denotes the distance estimates of the scale-(α, ε') distance oracle. We note that the condition is always satisfied for $i = \lfloor \log_2 \delta(u, w) \rfloor - 1$, and possibly also for $i = \lfloor \log_2 \delta(u, w) \rfloor$. With this value of α, $\hat{\delta}^{(\alpha, \varepsilon/4)}(u, w)$ gives us the desired approximation since $\hat{\delta}^{(\alpha, \varepsilon/4)}(u, w)/\delta(u, w) = 1 + (\varepsilon\alpha/4)/(\alpha/4) = 1 + \varepsilon$. The total query time is $O(\log\log(nN) + 1/\varepsilon)$. The

reduction is captured in the following lemma:

LEMMA 3.9. *We can construct a stretch-*$(1+\varepsilon)$ *distance oracle using* $O(\log(nN))$ *scale-*(α, ε') *distance oracles where* $\alpha = 2^i$, $i = 0, \ldots, \lceil \log_2(nN) \rceil$, $\varepsilon' \in \{1/2, \varepsilon/4\}$. *If a scale-*$(\alpha, \varepsilon')$ *distance oracle approximates distances in time* $t(\varepsilon')$ *independently of* α, *then the stretch-*$(1+\varepsilon)$ *distance oracle approximates distances in time* $O(t(1/2)/\varepsilon + t(\varepsilon/4)\log\log(nN))$. *Moreover, if the scale-*(α, ε') *distance oracles are distributed on labels, the stretch-*$(1+\varepsilon)$ *combines the labels for each vertex.*

Combining Lemma 3.8 and Lemma 3.9, we get

LEMMA 3.10. *We can construct a stretch-*$(1+\varepsilon)$ *using* $O(n(\log(nN))(\log n)/\varepsilon)$ *space and which approximates distances* $O(\log\log(nN) + 1/\varepsilon)$ *time. The oracle can be distributed as a labeling scheme with labels of size* $O((\log(nN))(\log n)/\varepsilon)$.

To get an efficient construction of the oracle in Lemma 3.10, we need an efficient construction of covering connections like those in Lemma 3.4.

3.4. CONSTRUCTING APPROXIMATION CONNECTIONS VIA A DIPATH EFFICIENTLY. We are given a digraph H containing a shortest dipath Q from s to t whose length is at most α. We want to efficiently construct ε-covering sets of connections between Q and each vertex in H. Here, all distances are understood to be in H. Technically, this efficient construction is the most complex part of this article.

We will generally focus on connections from Q to vertices v in H, the symmetric connections from v to Q being understood. The ε-covering sets will at first have size $O((\log n)/\varepsilon)$, but the size will be reduced with the following generic lemma.

LEMMA 3.11. *Given an ordered* ε_0*-covering set* $D(Q, v)$ *of connections from* Q *to* v, *we can construct a clean ordered* $(\varepsilon_0+\varepsilon_1)$*-covering set* $C(Q, v) \subseteq D(Q, v)$ *of size at most* $1 + (2+\varepsilon_0)/\varepsilon_1 = O(1/\varepsilon_1)$ *in* $O(|D(Q, v)|)$ *time.*

PROOF. This proof is a symmetric generalization of that for Lemma 3.4. First we delete any $(a, v) \in D(Q, v)$ with $\ell(a, v) > \alpha + \varepsilon_0\alpha$. Assuming this done, we visit the connections of $D(Q, v)$ in backwards order. The first connection (c, v) is always included in $C(Q, v)$. For a later connection (a, v), let (b, v) be the last connection added to $C(Q, v)$. We then add (a, v) to $C(Q, v)$ if $\delta(a, b)+\ell(b, v) > \ell(a, v)+\varepsilon_1\alpha$.

To see that $C(Q, v)$ is $(\varepsilon_0 + \varepsilon_1)$-covering, consider any $a \in Q$, and let b be a successor of a in Q such that $(b, v) \in D(Q, v)$ and $f(b) = \delta(a, b) + \ell(b, v)$ is minimized. Since $D(Q, v)$ is ε_0-covering, $f(b) \leq \delta(a, v) + \varepsilon_0\alpha$. However, (b, v) is only excluded from $C(Q, v)$ if there is a $(c, v) \in C(Q, v)$ such that $\delta(b, c) + \ell(c, v) \leq \ell(b, v) + \varepsilon_1\alpha$. Then, $\delta(a, c) + \ell(c, v) \leq \delta(a, b) + \ell(b, v) + \varepsilon_1\alpha = f(b) + \varepsilon_1\alpha \leq \delta(a, v) + (\varepsilon_0 + \varepsilon_1)\alpha$.

We now want to argue that the set $C(Q, v)$ is small. The simple point is that when we add (a, v) to $C(Q, v)$, we decrease the distance to v from the first point s in Q by $> \varepsilon_1\alpha$. More precisely, when (b, v) is added, the decrease is by $(\delta(s, b) + \ell(b, v)) - (\delta(s, a) + \ell(a, v)) = \delta(a, b) + \ell(b, v) - \ell(a, v) > \varepsilon_1\alpha$. When the first connection (c, v) is added, the distance was at most $\delta(s, c) + \ell(c, v) \leq 2\alpha + \varepsilon_0\alpha$, and hence we can add $< (2 + \varepsilon_0)/\varepsilon_1$ additional connections. □

We are now going to present an efficient way of constructing ε-covering sets of size $O((\log n)/\varepsilon)$. In order to state the result formally, let $\text{sssp}(Q, H)$ be the

smallest number such that if

—H_0 is a subgraph of H,

—Q_0 is the reduction (c.f. Section 1) of Q to vertices from H_0, and

—b is any vertex in Q_0,

then we can compute single source shortest dipaths from b in $H_0 \cup Q_0$ in $O(\text{sssp}(Q, H)|E(H_0)|)$ time. We note that with a classic heap [Williams 1964], $\text{sssp}(Q, H) = O(\log |V(H)|)$. Also, since we have integer weights, we can use the priority queue from Thorup [2000] with $\text{sssp}(Q, H) = O(\log \log |V(H)|)$. Finally, if $H \cup Q$ is planar, $\text{sssp}(Q, H) = O(1)$ [Henzinger et al. 1997]. In our applications, however, $H \cup Q$ will not always be planar.

LEMMA 3.12. *Given a graph H with a shortest dipath Q, for each $v \in V(H)$, we can construct an ordered ε-covering set $C(Q, v)$ of size $O((\log |V(Q)|)/\varepsilon)$ in $O(\text{sssp}(Q, H)|E(H)|(\log |V(Q)|)/\varepsilon)$ time.*

PROOF. The proof of Lemma 3.12 is rather technical, but the construction itself is quite simple and natural.

Construction. In the construction, we will say that (b, v) *semi-ε-covers* (a, v) if a equals or precedes b in Q and $\delta(a, b)+\ell(b, v) \leq \ell(a, v)+\varepsilon\alpha$. If $\ell(a, v) = \delta(a, v)$, this is the same as ε-covering, but $\ell(b, v)$ may be larger than $\delta(b, v)$. We introduce semi-ε-covering because it can be tested in constant time.

We will use a recursion taking a pair (Q_0, H_0) where Q_0 is a segment of Q and H_0 is an induced subgraph of H. Here, *induced subgraph* means that H_0 contains all edges from H between the vertices in H_0. The recursion will then make some connections from interior vertices in Q_0 to the vertices in H_0. The recursion assumes that we have connections from each end-point of Q_0 to all vertices in H_0. Some of these may be infinite, but are included for ease of presentation.

To get started, we make a single source shortest path computation for each of the end-points s and t of Q, and then, for each $v \in V(H)$, we connect s to v with $\ell(s, v) = \delta(s, v)$ and t to v with $\ell(t, v) = \delta(t, v)$. We can now recurse on (Q, H).

Given (Q_0, H_0), we recurse as follows: Let a be the first and c be the last point in Q_0. Let H_0^* be the digraph consisting of H_0 and Q_0 and all connections from the end-points a and c of Q_0 to vertices in H_0. Let b be a vertex in the *middle* of Q_0, that is, if Q_0 has q vertices, b is the $\lceil q/2 \rceil$th vertex in Q_0. After a single source shortest dipath computation from b in H_0^*, for each $v \in V(H_0)$, we connect b to v with $\ell(b, v) = \delta_{H_0^*}(b, v)$.

Next, let Q_1 be the part of Q_0 before b and let Q_2 be the part after b. Let U_1 to be the set of vertices v with $\ell(a, v) > 2\alpha$ or (b, v) semi-ε-covering (a, v). Similarly, let U_2 to be the set of vertices v with $\ell(b, v) > 2\alpha$ or (c, v) semi-ε-covering (b, v). Finally, set $H_1 = H_0 \backslash U_1$ and $H_2 = H_0 \backslash U_2$, and recurse on (Q_1, H_1) and (Q_2, H_2). This completes the description of our construction.

In order to prove that our construction satisfies the statement of Lemma 3.12, we need to argue that it correctly produces an ε-covering set of connections, that there are not too many connections, and that it is efficient.

3.4.1. *Correctness.* As our first step towards proving correctness, we argue

CLAIM 3.12.1. *For $v \in V(H_0)$ and $d \in \{a, b, c\}$, $\ell(d, v) = \delta_{H_0^*}(d, v)$.*

PROOF. From the construction of H_0^*, we immediately have $\ell(a, v) \geq \delta_{H_0^*}(a, v)$ and $\ell(c, v) \geq \delta_{H_0^*}(c, v)$. Also, the new connections from b satisfy Claim 3.12.1. It only remains to prove $\ell(a, v) \leq \delta_{H_0^*}(a, v)$ and $\ell(c, v) \leq \delta_{H_0^*}(c, v)$. Clearly, this is true when we make the first recursive call, and it follows inductively for the subproblems because H_1^* and H_2^* cannot provide shorter distances than H_0^*. □

We are now ready to prove the main property of our recursion:

CLAIM 3.12.2. *Let x be in the interior of Q_0 and v be any vertex in H with $\delta(x, v) \leq \alpha$. If some shortest dipath P from x to v leaves H_0, then (x, v) is ε-covered.*

PROOF. The proof is by induction. For the base case, the statement is vacuously true when we start with $(Q_0, H_0) = (Q, H)$, for then no dipath can leave H_0. Now, assume that the claim holds for (H_0, Q_0). To show that the recursion on (Q_1, H_1) satisfies the claim, let x be in the interior of Q_1 and consider a shortest dipath P from x to v which leaves H_1 and is of length at most α. We need to show that (x, v) is ε-covered. If P leaves H_0, we know that (x, v) is ε-covered by the claim on (Q_0, H_0), so we can assume that P remains in H_0. Let u be the first point at which P leaves H_1, that is, $u \in U_1$. Then

$$\begin{aligned}\delta(x, v) &= \delta_{H_0}(x, u) + \delta_{H_0}(u, v)\\ &\geq \delta_{H_0^*}(a, u) - \delta(a, x) + \delta_{H_0}(u, v)\\ &= \ell(a, u) - \delta(a, x) + \delta_{H_0}(u, v).\end{aligned}$$

Since the last expression is bounded by $\delta(x, v) \leq \alpha$ and $\delta(a, x)$ is bounded by the length of Q, which is at most α, $\ell(a, u) \leq 2\alpha$. Since $u \in U_1$, this implies that (a, u) is semi-ε-covered by (b, u), so $\ell(a, u) \geq \delta(a, b) + \ell(b, u) - \varepsilon\alpha$. Thus, we get

$$\begin{aligned}\delta(x, v) &\geq \delta(a, b) + \ell(b, u) - \varepsilon\alpha - \delta(a, x) + \delta_{H_0}(u, v)\\ &\geq \delta(x, b) + \delta_{H_0^*}(b, v) - \varepsilon\alpha\\ &= \delta(x, b) + \ell(b, v) - \varepsilon\alpha.\end{aligned}$$

Thus, (x, v) is ε-covered, so (H_1, Q_1) satisfies the claim. The proof that (H_2, Q_2) satisfies the claim is identical but with a and b replaced with b and c. This completes our inductive proof of Claim 3.12.2. □

From Claim 3.12.2, we will now argue that we have an ε-covering set of connections. Let $(b, v) \in V(Q) \times V(H)$, $\delta(b, v) \leq \alpha$. If $b \in \{s, t\}$, (b, v) is trivially covered, since we connected b to v with $\ell(b, v) = \delta(b, v)$ in our preliminary step before recursing. Otherwise, consider the recursive call (Q_0, H_0) with b the middle vertex in Q_0. From Claim 3.12.2, it follows that if (b, v) is not ε-covered by previous recursions, v is in H_0 and $\delta_{H_0}(b, v) = \delta(b, v)$, so we connect b to v with $\ell(b, v) = \delta(b, v)$. This completes the proof of correctness.

3.4.2. *Number of Connections.* We now want to show that the number of connections to each vertex in H is $O((\log |V(Q)|)/\varepsilon)$. Besides the initial two connections from s and t, we have one connection to v from b for each recursive call involving v. Here, a call *involves* v if v is in H_0. Since the recursion depth is at most $\log_2 |V(Q)|$, it suffices to argue that there are $O(1/\varepsilon)$ minimal calls involving v. Here, by *minimal* calls involving v, we mean calls involving v but with no

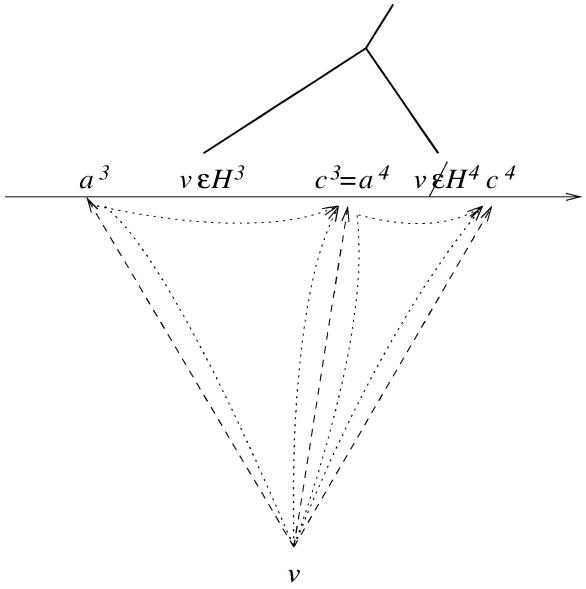

FIG. 6. The call (Q^3, H^3) involves v whereas its sibling (Q^4, H^4) does not involve v. Because (Q^3, H^3) involves v, when going to c^3, the direct connection (v, c^3) saves at least $\varepsilon\alpha$ over the detour via a^3.

descending call involving v. Below we assume that the root call is not the only call involving v; otherwise the result is trivially true since $\varepsilon \leq 1$.

By a maximal antichain of recursive calls, we mean a sequence of calls $(Q^0, H^0), \ldots, (Q^k, H^k)$ such that if a^i and c^i are the first and last vertices of Q^i, $a^0 = s, a^i = c^{i-1}$, and $c^k = t$. We consider the maximal anti-chain containing all minimal calls involving v plus nonascending children of ascending calls. A segment of this antichain is illustrated in Figure 6. As a result, we have connections from both a^i and c^i to v, and we are interested in the quantity $d^i = \delta(a^i, c^i) + \ell(c^i, v) - \ell(a^i, v)$. If $d^i \leq \varepsilon\alpha$, (c^i, v) semi-ε-covers (a^i, v). Hence, $d^i > \varepsilon\alpha$ if v is in H^i. We also need

CLAIM 3.12.3. *Each d^i is nonnegative.*

PROOF. From Claim 3.12.1, we know that the digraph H^{i*} is such that $\ell(a^i, v) = \delta_{H^{i*}}(a^i, v)$ and $\ell(c^i, v) = \delta_{H^{i*}}(c^i, v)$. However, H^{i*} contains Q^i, so $\delta_{H^{i*}}(a^i, v) \leq \delta(a^i, c^i) + \delta_{H^{i*}}(c^i, v)$, and hence $\ell(a^i, v) \leq \delta(a^i, c^i) + \ell(a^i, v)$ □

Let (Q^l, H^l) be the last of the calls involving v. Since (Q^l, H^l) has v in H^l, we know that $\ell(a^l, v) \leq 2\alpha$. However, since Q is of length at most α, we get

$$\sum_{i<l} d^i = \delta(a^0, a^l) + \ell(a^l, v) - \ell(a^0, v) \leq 3\alpha.$$

Now, since each d^i is nonnegative and $d^i > \varepsilon\alpha$ if (Q^i, H^i) involves v, we conclude that the total number of minimal calls involving v is at most $3/\varepsilon + 1 = O(1/\varepsilon)$, as desired. This completes our analysis of the number of connections.

3.4.3. *Efficiency.* Above, we argued that each vertex is only involved in a limited number of calls, so now we just have to argue that in each call, we spend limited time per involved vertex.

We assume that the reduction Q_0^r of Q_0 to vertices in H_0 is available. Trivially, this is the case for the initial call with $(Q_0, H_0) = (Q, H)$ since Q is a dipath in H, and hence reduced in itself. Also, given Q_0^r, we can trivially reduce it to vertices in H_1 and H_2, producing Q_1^r and Q_2^r, in $O(|V(Q_0^r)|) = O(|V(H_0)|)$ time.

Given Q_0^r, we compute all distances from b in the digraph $H_0 \cup Q_0^r$. By definition, this takes $O(\text{sssp}(Q, H)|E(H_0)|)$ time. If a and c are the first and last points in Q_0, we may be missing distances over a and c. Let v be any vertex in H_0. By Claim 3.12.1, we know that $\delta_{H_0^*}(a, v) = \ell(a, v)$ and $\delta_{H_0^*}(c, v) = \ell(c, v)$. The distance to v via c is now computed as $\delta(b, c) + \ell(c, v)$. It remains to deal with dipaths via a that do not first go over c. We note that such a dipath can only exist if a is in H_0, and then we compute its length as $\delta_{H_0 \cup Q_0^r}(b, a) + \ell(a, v)$. Summing up, having computed $\delta_{H_0 \cup Q_0^r}(b, v)$ for all v in H_0, for each v, we set

$$\delta_{H_0^*}(b, v) = \min\left\{\delta_{H_0 \cup Q_0^r}(b, v), \delta(b, c) + \ell(c, v), \delta_{H_0 \cup Q_0^r}(b, a) + \ell(a, v)\right\}.$$

Here $\delta_{H_0 \cup Q_0^r}(b, a) = \infty$ if a is not in H_0. Thus, the processing time for a call (Q_0, H_0) is $O(\text{sssp}(Q, H)|E(H_0)|)$.

For each edge, we pay $O(\text{sssp}(Q, H))$ time for each of the $O((\log |V(Q)|)/\varepsilon)$ calls involving one of its endpoints, so the total running time is $O(\text{sssp}(Q, H)|E(H)|(\log |V(Q)|)/\varepsilon)$, as desired. This completes our analysis of the efficiency.

Above, we have established that the algorithm constructs an ε-covering set of $O((\log |V(Q)|)/\varepsilon)$ connections to each vertex in $O(\text{sssp}(Q, H)|E(H)|(\log |V(Q)|)/\varepsilon)$ total time. This completes the proof of Lemma 3.12. □

We are now ready to make an efficient construction for Lemma 3.7.

LEMMA 3.13. *Given a digraph H with a shortest dipath Q, for each $v \in V(H)$, we can construct ε-covering sets $C(v, Q)$ and $C(Q, v)$ of size $O(1/\varepsilon)$ in $O(\text{sssp}(Q, H)|V(H)|(\log |V(Q)|)/\varepsilon)$ total time. Afterward, if a shortest dipath intersects Q and has length below α, we can approximate its length with an additive error of $O(\varepsilon\alpha)$ in $O(1/\varepsilon)$ time.*

PROOF. For the construction, first we apply Lemma 3.12 with $\varepsilon' = \varepsilon/2$, and second we apply Lemma 3.11 with $\varepsilon_0 = \varepsilon_1 = \varepsilon/2$ to each set of connections. We get the approximation accuracy and time from Lemma 3.5 and Lemma 3.6. □

3.5. FIRST EFFICIENT CONSTRUCTIONS OF APPROXIMATE DISTANCE ORACLES. In Section 3.3, disregarding the construction time, we took our reachability oracle and replaced reachability connections with approximation connections. Thereby, we lost a factor $1/\varepsilon$ in space and query time. More precisely, as stated in Lemma 3.8, we got a scale-(α, ε) distance oracle using $O(n(\log n)/\varepsilon)$ space which approximated distances in $O(1/\varepsilon)$ time.

We would now like to include an efficient construction of approximation connections. If we just use our basic recursion with separators from Section 2.4, we can apply Lemma 3.13 directly. More precisely, each recursive call has a subgraph H and a separator S with a constant number of dipaths Q. For that call, we are only interested in dipaths in H that intersect Q so Lemma 3.13 applies. Since H is planar, $\text{sssp}(Q, H) = O(1)$ [Henzinger et al. 1997], so we only pay $O(\log n)$ time per connection found, and we end up with $O((\log n)/\varepsilon)$ connections per vertex. Now, to approximate the distance from u to w, we have to check $O(\log n)$ dipaths Q. If the distance is below α, one of these dipaths is intersected, and we

approximate the distance with an error of $O(\varepsilon\alpha)$. Adjusting ε by a constant factor, we have now produced a scale-(α, ε) distance oracle in $O(n(\log n)^2/\varepsilon)$ time, using $O(n(\log n)^2/\varepsilon)$ space, and approximating distances in $O((\log n)/\varepsilon)$ time. Applying Lemma 3.9, we get

PROPOSITION 3.14. *In $O(n(\log n)^2(\log(nN))/\varepsilon)$ time, we can construct a stretch-$(1+\varepsilon)$ using $O(n(\log(nN))(\log n)/\varepsilon)$ space and which approximates distances $O((\log n)(\log\log(nN)+1/\varepsilon))$ time.*

However, what we really want a distance oracle like in Lemma 3.8 which approximates the distances in $O(1/\varepsilon)$ time. It gains a factor $(\log n)$ in query time using approximation connections via the frames from Section 2.5. However, for a frame dipaths Q, the connections have to be based on all dipaths in G and not just dipaths in the subgraph H. In the reachability recursion with frames, we constructed $H \star F$ based on connections from the previous recursion. Then, $H \star F$ represented reachability via G between H and F. We can construct $H \star F$ accordingly based on previous approximation connections, but the errors will add up as we recurse. To solve this problem, we apply Lemma 3.13 with $H' = H \star F$ and $\varepsilon' = \varepsilon/(\log n)$. With $\log n$ recursion levels, our errors add up to $O(\varepsilon'\alpha \log n) = O(\varepsilon\alpha)$, as desired. Now, from the previous recursion, each vertex in H got $O(1/\varepsilon')$ connections, so the possibly non-planar digraph H' has $O(|V(H)|/\varepsilon')$ edges. Applying Lemma 3.13 with $\text{sssp}(Q, H') = O(\log\log n)$ [Thorup 2000], we construct the connections to the dipaths in the frame and separator in $O((\log\log n)|V(H)|(\log n)/\varepsilon'^2)$ time, thus ending up with a total construction time of $O(n(\log n)^2(\log\log n)/\varepsilon'^2) = O(n(\log n)^4(\log\log n)/\varepsilon^2)$.

We note that the $O(1/\varepsilon') = O((\log n)/\varepsilon)$ connections between a vertex v in H and a separator or frame dipath Q are there to support later recursions. When done with all the recursions, with Lemma 3.11, we can reduce the number of connections between v and Q to $O(1/\varepsilon)$. Then, as in Lemma 3.8, we have a scale-(α/ε) distance oracle approximating distances in $O(1/\varepsilon)$ time.

3.6. REDUCING THE CONSTRUCTION TIME. We are now going to reduce the construction time above by a factor of $(\log n)(\log\log n)$ First, we run the alternating frame and subgraph reducing recursions as in Section 2.5. With each call, we get a digraph H together with a frame F and a separator S, both of constant size. However, we only make connections over H between S and H as in the basic recursion Section (2.4), that is, we do not construct the digraph $H \star F$ to make global connections over G_i^α.

We now want to construct connections between F and H representing distances of dipaths in G_i^α which intersect F and whose end-points are vertices in H. We view any such dipath as divided into two parts: a *first part*, P_0, having all interior vertices in H and the last vertex in F, and a *second part*, P_1, being the rest of the dipath, starting in the last vertex of P_0, and possibly intersecting F many times before eventually ending in H. Our connections from H to F will represent the distances of the first parts whereas our connections to H from F will represent the second parts.

The first parts P_0 are easily dealt with: for each frame dipath Q in F, let Q^H be Q reduced to vertices neighboring vertices in H, and let $Q^H + H$ be the digraph consisting of Q^H, H, and all edges between Q and H. These reductions are done as part of the general recursion in linear total time. We now construct connections

from H to Q^H using Lemma 3.13. Since we are dealing with a planar digraph, $\text{sssp}(Q, H) = O(1)$ [Henzinger et al. 1997], and hence the running time is bounded by $O(|E(Q^H + H)|(\log n))$.

For the second parts P_1, we are going to use the connections via the $O(\log n)$ separator dipaths from ascending recursive calls. We know that these represent the desired distances within an error of $O(\varepsilon\alpha)$. More precisely, for each frame dipath Q, we construct a digraph X_Q^H as follows. First it has the vertices from Q^H, then it has the $O(\log n)$ ascending separator dipaths $\mathcal{S}$, and finally it has the vertices in H. Each separator dipath has its private copy of each vertex in it so that they do not intersect with each other, or with Q^H. From $V(Q^H)$, we have all the connections to $\mathcal{S}$, that is, $O(1/\varepsilon)$ connections from each vertex in Q^H to each dipath in $\mathcal{S}$, hence $O(|V(Q^H)|(\log n)/\varepsilon)$ connections in total. From $\mathcal{S}$, we have all the $O(|V(H)|(\log n)/\varepsilon)$ connections to the vertices in H. Finally, the dipaths in $\mathcal{S}$ are reduced to vertices with connections from Q^H or to H. Thus, X_Q^H has $O(|V(Q^H) \cup V(H)|(\log n)/\varepsilon)$ vertices and edges, and it represents all distances from Q^H to H with an error of $O(\varepsilon\alpha)$. We can now apply Lemma 3.13 to get the desired connections from Q^H to H. It is easy to see that $\text{sssp}(Q^H, X_Q^H) = O(1)$: with starting point in any vertex a in Q^H, first we get distances to $\mathcal{S}$ using a single connection from a, then we progress the distances through each reduced dipath from $\mathcal{S}$, and finally, we get the distances to H using single connections from $\mathcal{S}$ to H. The total running time is therefore $O(|V(Q^H) \cup V(H)|(\log n)^2/\varepsilon)$.

The reductions of Q and the dipaths in $\mathcal{S}$ are easily done within the same time bound as we have an extra $O(\log n)$ factor for sorting the relevant vertices within each dipath, so our total time bound for constructing the first part connections from H to Q and the second part connections to H from Q is $O(|V(Q^H) \cup V(H)|(\log n)^2/\varepsilon^2)$.

Since there are only a constant number of dipaths Q in the frame F, the construction for the whole call takes $O(|N(H)|(\log n)^2/\varepsilon)$ time. Here $N(H)$ is the set of vertices that are either in H or a neighbors to vertices H. Since each vertex, and hence each endpoint of an edge, participates in $O(\log n)$ calls, and since there are $O(n)$ edges in a planar digraph, the total construction time is $O(n(\log n)^3/\varepsilon^2)$. Adjusting ε down by a constant factor in the above construction, we get.

LEMMA 3.15. *In $O(n(\log n)^3/\varepsilon^2)$ time, we can construct a $O(n(\log n)/\varepsilon)$ space scale-(α, ε) distance oracle approximating distances in $O(1/\varepsilon)$ time.*

Lemma 3.15 together with Lemma 3.9 gives us our main theorem.

THEOREM 3.16. *We can preprocess a planar digraph in $O(n(\log(nN))(\log n)^3/\varepsilon^2)$ time, producing an $O(n(\log(nN))(\log n)/\varepsilon)$-space distance oracle that can answer stretch-$(1+\varepsilon)$ distance queries in $O(\log\log(nN) + 1/\varepsilon)$ time. The oracle can be distributed as a labeling scheme with labels of size $O((\log(nN))(\log n)/\varepsilon)$.*

We note that Theorem 3.16 does not dominate Proposition 3.14. Theorem 3.16 has the faster query time but Proposition 3.14 has the faster preprocessing time. Here, we do consider the query time the most important, so Theorem 3.16 is viewed as the main result.

3.7. FINDING THE DIPATHS AND ROUTING. To actually find the approximate shortest dipaths, and to route along them, we use the techniques from Sections 2.7

and 2.8. What we did there was to use the to- and from-tree produced in Section 2.3 for each separator dipath. Using the method from Section 2.7.2, we can restrict our attention to the basic recursion.

Now, for approximate distances, as described in Section 3.4, we construct many to- and from-trees. More precisely, in Lemma 3.12, we produce a set of from-trees witnessing all the connections from Q, and each vertex v is involved in $O((\log n)/\varepsilon)$ of these trees. The idea now is that we preserve all these trees using $O((\log n)/\varepsilon)$ parent pointers per vertex. We do this even if the individual vertex is only interested in the $O(1/\varepsilon)$ from-trees witnessing the $O(1/\varepsilon)$ connections it has left after application of Lemma 3.11. As a result, the space blows up by a factor of $O(\log n)$.

A small caveat above is that given an appropriate identifier for a to- or from-tree, we want to access parent pointers in constant time. Our problem is that a given separator dipath can generate $O(n)$ trees out of which each vertex participates in $O((\log n)/\varepsilon)$. We provide the desired access via a dictionary, as described in Hagerup et al. [2001]. The space of the dictionary is linear in the number of elements, so it does not affect our space bounds. However, it costs a log-factor to construct it. Since each vertex has $O(\log n)$ ascending separator dipaths, the total space for accessible parent pointers is $O(n(\log n)^2/\varepsilon)$, and they are constructed in $O(n(\log n)^3/\varepsilon)$ time. The construction time is, however, dominated by the overall construction time from Theorem 3.16.

Now, our query algorithm for the approximate distance from u to w identifies a separator dipath Q as well as connections (u, a) and (b, w) with a before b in Q. It returns the distance $\ell(u, a) + \delta(a, b) + \ell(b, w)$. We can now get a dipath of at most this length by considering the to-tree from u witnessing (u, a) and from-tree to w witnessing (b, w). Using the method from Section 2.7, we identify the dipath in time proportional to its length.

For routing, we use the method from Section 2.8, spending $O(\log n)$ space for each tree a vertex is involved in. Thus, the space blows up by an additional factor $O(\log n)$, but now we can perform each routing decision in constant time.

3.8. UNDIRECTED SIMPLIFICATIONS. We note that the above construction for approximate distances can be simplified and improved in the case of undirected graphs. Instead of the $(3, \alpha)$-layered graphs and trees for different α, we simply take an arbitrary vertex s in our input graph G and construct one shortest path tree T from s. This shortest path tree is used to find separators as in Section 2.2. Thereby, we will avoid paying a multiplicative $\log(nN)$ factor for the exponentially increasing α in the construction time and space. Also, we avoid the additive $\log\log(nN)$ in the query time.

Our problem now is that the separator paths are of unbounded length. We want to use a construction similar to the one in Section 3.4, but first we need to make an appropriate definition of the ε-covering by connections from a vertex v to an undirected separator path Q. Let $\delta(Q, v)$ denote the distance between v and its nearest vertex in Q. If a shortest path from u to vertex v intersects Q, then $\delta(u, v) \geq \delta(Q, v)$, and hence we can tolerate an error of $\varepsilon\delta(Q, v)$ when estimating the distance to v. It is therefore natural to say that a connection (b, v) *ε-covers* (a, v), $a, b \in Q$, if $\delta(a, b) + \ell(b, v) \leq \delta(a, v) + \varepsilon\delta(Q, v)$.

LEMMA 3.17. *There is an ε-covering set of connections from Q to Q of size $O(1/\varepsilon)$.*

PROOF. Let c be the vertex in Q nearest v, that is, $\delta(c, v) = \delta(Q, v)$. Our first connection is (c, v) with $\ell(c, v) = \delta(c, v) = \delta(Q, v)$. We now set $y = c$ and iterate x through the vertices from c towards one of the end-points s of Q. Each time (y, v) does not cover (x, v), we make the connection (x, v) with $\ell(x, v) = \delta(x, v)$, and set $y = x$.

To see that, we add at most $2/\varepsilon$ connections as we iterate x, we note that when we add a connection (x, v) and move y to x, we reduce $\delta(s, x) + \delta(x, v)$ by more than $\varepsilon\delta(c, v)$. However, we start with $\delta(s, x) + \delta(x, v) = \delta(s, c) + \delta(c, v)$ and the triangle inequality and symmetry implies that $\delta(s, x) + \delta(x, v) \geq \delta(s, v) \geq \delta(s, c) - \delta(c, v)$. Thus, the total improvement can be at most $2\delta(c, v)$.

The same procedure is applied towards the other end-point of Q, so we end up with at most $1 + 4/\varepsilon$ connections. □

As a result we end up with an $O(n(\log n)/\varepsilon)$-space oracle answering stretch-$(1+\varepsilon)$ distance queries in $O(1/\varepsilon)$ time. An undirected approximate distance oracle with similar bounds for space and query time has been found independently by Klein [2002].

In order to get a fast construction of an ε-covering set of connections, we again modify the definition of ε-covering. We say that (b, v) *ε-covers* (a, v) if $\delta(a, v) \geq \delta(a, b) + (1 - \varepsilon)\ell(b, v)$. Note that if x is between a and b, then $\delta(x, v) \geq \delta(a, v) - \delta(a, x) \geq \delta(x, v) + (1 - \varepsilon)\ell(b, v)$, so (b, v) covers (x, v) as well. This would not necessarily have been the case if we had defined ε-covering by the more natural $(1 + \varepsilon)\delta(a, v) \geq \ell(v, a) + \delta(a, b)$. Using our definition $\delta(a, v) \geq \delta(a, b) + (1 - \varepsilon)\ell(b, v)$, we get the following analog to Lemma 3.12:

LEMMA 3.18. *For each $v \in V(H)$, we can construct an ε-covering set $C(Q, v)$ of size $O((\log |V(Q)|)/\varepsilon)$ in $O(\text{sssp}(Q, H)|V(H)|(\log |V(Q)|)/\varepsilon)$ time.*

We note here that since H is undirected, $\text{sssp}(Q, H) = O(1)$ [Thorup 1999].

PROOF. The proof is very similar to that of Lemma 3.12.

Construction. This time, we say that (v, b) *semi-ε-covers* (v, a) if $\ell(v, a) \geq (1 - \varepsilon)\ell(v, a) + \delta(a, b)$. Again the point is that we replace the unknown $\delta(b, v)$ from the definition of ε-covering by the known, but potentially larger, $\ell(v, b)$.

The recursion is very similar to the one from Lemma 3.12. It is defined in terms of a segment Q_0 of Q and an induced subgraph H_0 of H, with connections from the end-points of a and c of Q_0 to all vertices in H_0. To get started, we just add connections from the end-points s and t of Q, and set $(Q_0, H_0) = (Q, H)$.

Given (Q_0, H_0), we recurse as follows: Let a be the first and c be the last point in Q_0. Let H_0^* be the graph consisting of H_0 and Q_0 and all connections from the end-points a and c of Q_0 to vertices in H_0. Let b be a vertex in the middle of Q_0. After a single source shortest path computation from b in H_0^*, for each $v \in V(H_0)$, we connect b to v with $\ell(b, v) = \delta_{H_0^*}(b, v)$.

The interesting new point is the definition of U_1 and U_2. We define U_1 to be the set of vertices v where one of (v, a) and (v, b) semi-ε-covers the other, and U_2 to be the set of vertices v where one of (v, b) and (v, c) semi-ε-covers the other. Apart from the fact that we have a new definition of semi-ε-covering, we note that we do not pick vertices based on $\ell(a, v) > 2\alpha$. We now set $H_1 = H_0 \backslash U_1$ and $H_2 = H_0 \backslash U_2$, and recurse on (Q_1, H_1) and (Q_2, H_2). This completes our description of the Construction.

Correctness. The proof of Claim 3.12.1 is unchanged, so we still have $\ell(d, v) = \delta_{H_0^*}(a, v)$ for $v \in V(H_0)$ and $d \in \{a, b, c\}$.

We need to prove Claim 3.12.2: *Let x be in the interior of Q_0 and v be any vertex in H with $\delta(x, v) \leq \alpha$. If some shortest path P from x to v leaves H_0, then (x, v) is ε-covered.*

We assume that Claim 3.12.2 is satisfied for (Q_0, H_0), and we want to establish it for (Q_1, H_1). We consider some shortest path P which leaves H_1 and which goes from a point x in Q_1 to a vertex v in H. Inductively, we may assume that P stays in H_0. Let u be the first point at which P leaves H_1. Then, one of (a, v) and (b, v) semi-ε-covers the other. Assume it is (b, v) that semi-ε-covers (a, v). Then,

$$\begin{aligned}
\delta(x, v) &= \delta_{H_0}(x, u) + \delta_{H_0}(u, v) \\
&\geq \delta_{H_0^*}(a, u) - \delta(a, x) + \delta_{H_0}(u, v) \\
&\geq \delta(a, b) + (1 - \varepsilon)\delta_{H_0^*}(b, u) - \delta(a, x) + \delta_{H_0}(u, v) \\
&\geq \delta(x, b) + (1 - \varepsilon)(\delta_{H_0^*}(b, u) + \delta_{H_0}(u, v)) \\
&\geq \delta(x, b) + (1 - \varepsilon)\delta_{H_0^*}(b, v) \\
&= \delta(x, b) + (1 - \varepsilon)\ell(b, v).
\end{aligned}$$

The last inequality uses the fact that $\varepsilon \leq 1$. If instead (a, v) semi-ε-covers (b, v), we would just swap the roles of a and b in the above derivation. The proof of Claim 3.12.2 for (Q_2, H_2) is identical but with c replacing a. This completes the proof of correctness.

Limited Involvement. As in the directed case, we limit both the number of connections and the efficiency of the construction if we can show that the number of minimal calls involving a given vertex v is $O(1/\varepsilon)$. Assuming that the root call is not the only call involving v, it is not minimal.

For this purpose, as for the directed case, we consider the maximal anti-chain of calls $(Q^0, H^0) \cdots (Q^k, H^k)$ containing all minimal calls involving v and all siblings not involving v of calls involving v. With a^i and c^i the first and last vertex in Q^i, $a^0 = s$, $a^{i+1} = c^i$, and $c^k = t$.

Let m be such that $\ell(a^m, v)$ is minimized. We will show that the number of minimal calls on either side of a^m is bounded by $2/\varepsilon$. For $i < m$, we consider the quantity $d^i = \delta(a^i, c^i) + \ell(c^i, v) - \ell(a^i, v)$. As in Claim 3.4.2 in the undirected case, $d^i \geq 0$. Also, if $d^i \leq \varepsilon\ell(c^i, v)$, (c^i, v) semi-ε-covers (a^i, v), so $d^i \geq \varepsilon\ell(c^i, v)$ if (Q^i, H^i) is one of the minimal calls involving v. Moreover, since a^m minimizes $\ell(a^m, v)$, $\ell(c^i, v) \geq \ell(a^m, v)$.

CLAIM 3.18.1. $\sum_{i<m} d^i \leq 2\ell(a^m, v)$.

PROOF. $\sum_{i<m} d^i = \delta(s, a^m) + \ell(a^m, v) - \ell(s, v)$. However, since our graph and connections are undirected, $\delta(s, a^m) \leq \ell(s, v) + \ell(a^m, v)$, so

$$\sum_{i<m} d^i \leq \ell(s, v) + \ell(a^m, v) + \ell(a^m, v) - \ell(s, v) = 2\ell(a^m, v) \qquad \square$$

Thus, there can be at most $2/\varepsilon$ minimal calls involving v before a^m.

The argument for $i \geq m$ is symmetric. This time, we study $d^i = \delta(c^i, a^i) + \ell(a^i, v) - \ell(c^i, v)$, and again we get that there can be at most $2/\varepsilon$ minimal calls

involving v. Thus, the number of minimal calls involving v is at most $1+4/\varepsilon$. This completes our analysis of the involvement.

As in the proof of Lemma 3.12, the $O(1/\varepsilon)$ bound on the number of minimal calls involving v implies that the number of connections to v is $O((\log|V(Q)|)/\varepsilon)$ and that they are computed in $O(\text{sssp}(Q,H)|V(H)|(\log|V(Q)|)/\varepsilon)$ time. Together with the correctness of our construction, this settles Lemma 3.18. □

With the above changes in mind, the undirected construction is as the directed construction. We get.

THEOREM 3.19. *We can preprocess a planar undirected graph in $O(n(\log n)^3/\varepsilon^2)$ time, producing an $O(n(\log n)/\varepsilon)$-space distance oracle that can answer stretch-$(1+\varepsilon)$ distance queries in $O(1/\varepsilon)$ time. The oracle can be distributed as a labeling scheme with labels of size $O((\log n)/\varepsilon)$.*

4. *Bounded Genus and Excluded Minors*

Our construction has only addressed planar graphs. However, our approach only hinges on two properties of undirected planar graphs:

Minor closed. (c.f. Lemma 2.2 (iv)).

Tree path separable. Given an arbitrary spanning tree, we can find a constant number of tree paths whose removal separates the graph into components of at most half the size (c.f. Lemma 2.3).

The question now is which other classes of graphs share these properties? Obviously, the last property is not satisfied for all graphs. A simple counter example is a complete graph with a star spanning tree. However, in the future, based on the work in Gilbert et al. [1984] and Robertson and Seymour [1986, 2003], we hope to show that any minor closed class of graphs excluding some graph is tree path separable, hence that our approach generalizes to any such class. In particular, this would generalize our approach to graphs of bounded genus.

5. *Concluding Remarks*

We have shown that reachability and approximate distances can be represented very efficiently for planar digraphs. An interesting practical challenge would be to use the ideas in this article as the basis for heuristics dealing with real road maps, like the one of the US, with bridges and other obstructions to planarity. One could, of course, just view bridges as incrementing the genus of the graph, but that would lead to a very high genus. Somehow, we would like to view the bridges in Boston, MA, as independent of the bridges in Washington, DC.

ACKNOWLEDGMENTS. I would like to thank Uri Zwick, Howard Karloff, Phil Klein, and several anonymous referees for thorough and useful comments.

REFERENCES

ARIKATI, S., CHEN, D., CHEW, L., DAS, G., SMID, M., AND ZAROLIAGIS, C. 1996. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proceedings of the 4th European Symposium on Algorithms* (Barcelona, Spain). Lecture Notes in Computer Science, vol. 1136. Springer-Verlag, New York, 514–528.

CHEN, D. 1995. On the all-pairs Euclidian shortest path problem. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, Calif.). ACM, New York, 292–301.

CHEN, D., AND XU, J. 2000. Shortest path queries in planar graphs. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing* (Portland, Ore.). ACM, New York, 469–478.

DJIDJEV, H. 1996. Efficient algorithms for shortest path queries in planar digraphs. In *Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science* (Como, Italy). Lecture Notes in Computer Science, vol. 1197. Springer-Verlag, New York, 151–165.

DJIDJEV, H., PANZIOU, G., AND ZAROLIAGIS, C. 1991. Computing shortest paths and distances in planar graphs. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 1991. Springer-Verlag, New York, 327–339.

DJIDJEV, H., PANZIOU, G., AND ZAROLIAGIS, C. 1995. Fast algorithms for maintaining shortest paths in outerplanar and planar digraphs. In *Proceeding of 10th International Symposium on Fundamentals of Computation Theory*. Lecture Notes in Computer Science, vol. 965. Springer-Verlag, New York, 191–200.

FREDERICKSON, G. 1987. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput. 16*, 1004–1022.

GAVOILLE, C., PELEG, D., PÉRENNES, S., AND RAZ, R. 2001. Distance labeling in graphs. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms* (Washington, D.C.). ACM, New York, 210–219.

GILBERT, J. R., HUTCHINSON, J. P., AND TARJAN, R. 1984. A separator theorem for graphs of bounded genus. *J. Algorithms 5*, 391–407.

GUPTA, A., KUMAR, A., AND RASTOGI, R. 2001. Traveling with a pez dispenser (or, routing issues in MPLS). In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Alamitos, Calif., 148–157.

GUPTA, A., KUMAR, A., AND THORUP, M. 2003. Tree based MPLS routing. In *Proceedings of the 15th ACM Symposium on Parallel Algorithms*. ACM, New York, 193–199.

HAGERUP, T., MILTERSEN, P., AND PAGH, R. 2001. Deterministic dictionaries. *J. Algorithms 41*, 1, 69–85.

HAREL, D., AND TARJAN, R. 1984. Fast algorithms for finding nearest common ancestor. *SIAM J. Comput. 13*, 338–355.

HENZINGER, M., KLEIN, P., RAO, S., AND SUBRAMANIAN, S. 1997. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci. 55*, 3–23.

INDYK, P. 1999. Sublinear time algorithms for metric space problems. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing* (Atlanta, Ga.). ACM, New York, 428–434.

KERNIGHAN, B., AND RITCHIE, D. 1988. *The C Programming Language*, Second ed. Prentice-Hall.

KLEIN, P. 2002. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, Calif.). ACM, New York, 820–827.

KLEIN, P., AND SUBRAMANIAN, S. 1998. A fully dynamic approximation scheme for shortest path problems in planar graphs. *Algorithmica 23*, 235–249.

LIPTON, R., AND TARJAN, R. 1979. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36*, 177–189.

PELEG, D. 2000a. Informative labeling schemes for graphs. In *Proceedings of the 25th Mathematical Foundations of Computer Science* (Bratislava, Slovak Republik). Lecture Notes in Computer Science, vol. 1893. Springer-Verlag, New York, 579–588.

PELEG, D. 2000b. Proximity-preserving labeling schemes. *J. Graph Theory 33*, 167–176.

ROBERTSON, N., AND SEYMOUR, P. S. 1986. Graph minors V: Excluding a planar graph. *J. Combin. Theory, Ser. B 41*, 1, 92–114.

ROBERTSON, N., AND SEYMOUR, P. S. 2003. Graph minors XVI: Excluding a non-planar graph. *J. Combin. Theory, Ser. B 89*, 1, 43–76.

SANTORO, N., AND KHATIB, R. 1985. Labelling and implicit routing in networks. *Comput. J. 28*, 1, 5–8.

SUBRAMANIAN, S. 1993. A fully dynamic data structure for reachability in planar digraphs. In *Proceedings of the 1st European Symposium on Algorithms* (Bad Honnef, Germany). Lecture Notes in Computer Science, vol. 726. Springer-Verlag, New York, 372–383.

TAMASSIA, R., AND TOLLIS, I. 1993. Dynamic reachability in planar digraphs with one source and one sink. *Theoret. Comput. Sci. 119*, 331–343.

THORUP, M. 1995. Shortcutting planar digraphs. *Combin., Prob. Comput. 4*, 287–315.

THORUP, M. 1999. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM 46*, 362–394.
THORUP, M. 2000. On RAM priority queues. *SIAM J. Comput. 30*, 1, 86–109.
THORUP, M., AND ZWICK, U. 2001a. Approximate distance oracles. In *Proceedings of the 33th Annual ACM Symposium on Theory of Computing* (Crete, Greece). ACM, New York, 183–192.
THORUP, M., AND ZWICK, U. 2001b. Compact routing schemes. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures* (Crete, Greece). ACM, New York, 1–10.
VAN LEEUWEN, J., AND TAN, R. 1986. Computer networks with compact routing tables. In *The book of L*, G. Rozenberg and A. Salomaa, Eds. Springer-Verlag, New York, 259–273.
WILLIAMS, J. 1964. Heapsort. *Commun. ACM 7*, 5, 347–348.

RECEIVED NOVEMBER 2001; REVISED MARCH 2004; ACCEPTED JUNE 2004