Compact Oracles for Reachability and Approximate Distances in Planar Digraphs

Mikkel Thorup
AT&T Labs - Research, Shannon Laboratory
180 Park Avenue, Florham Park, NJ 07932, USA
mthorup@research.att.com

Abstract

It is shown that a planar digraph can be preprocessed in near-linear time, producing a near-linear space distance 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 approximate distances in weighted planar digraphs where we can then get a $(1+\varepsilon)$ approximation distance in $O(\log\log\Delta + 1/\varepsilon)$ time where Δ is the longest finite distance in the graph and weights are assumed to be non-negative integers. Our scheme can be extended to find and route along the short dipaths.

Our technique is based on a novel dipath decomposition of planar digraphs *that instead of using the standard separator with $O(\sqrt{n})$ vertices, in effect finds a separator using a constant number of dipaths.*

1 Introduction

We show that a weighted planar digraph over n vertices can be preprocessed in near-linear time and space, producing an oracle that can answer reachability queries in constant time and distance queries within a factor $(1+\varepsilon)$ in $O(\log\log\Delta + 1/\varepsilon)$ time, where $\varepsilon > 0$ and Δ is the largest finite distance in the graph. Here, by distances in a graph refer to shortest dipath distances. Also, we generally assume that all weights are non-negative integers. The integrality is not essential, but the definition of Δ assumes that the smallest non-zero weight is at least 1.

An application of our ideas could be an efficient version of a software product like MapQuest that helps people finding driving distances in a near-planar road map. Another application could be as a preprocessing for so-called "sublinear time algorithms for metric space problems" [10], that given access to a distance oracle, solve certain optimization problems defined over the metric in $o(n^2)$ time.

So far, for planar digraphs, there has been no essential difference between the bounds for reachability and exact distances. No $o(n^2)$ bit oracle was known that could answer reachability queries in constant time, but in this paper, we get down to $O(n\log^2 n)$ bits that we identify in $O(n\log n)$ time!

In the rest of this paper, for simplicity we will generally just measure *space* as a number of words where each word is assumed big enough for a vertex identifier or distance. A more careful discussion of bits is deferred to the journal version.

Based on the separator theorems of Lipton and Tarjan [13] and Frederickson [6], Arkati et al. [1] and Djidjev [3], have presented a general trade-off between oracle space and query time for distance queries with a space-time product of $O(n^2)$, that is, using space s, we can answer distance/reachability queries in $O(n^2/s)$ time. Using the topology of planar graphs, Djidjev [3] has shown that with space $s \in [n^{4/3}, n^{3/2}]$, the 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 [2] generalized Djidjev's bound so that for space $s \in [n^{4/3}, n^2]$, they get a query time of $O(n/s\log(n/s) + \alpha(n))$. We note that the extended range does not give any space-time product better than Djidjev's $O(n^{5/3}\log n)$.

Our space-time product is near-linear, but we note that it does not work for exact distances. More precisely, for $(1+\varepsilon)$-approximate distances, we use $O(n\log n\log\Delta/\varepsilon)$ space to get a query time of $O(\log\log\Delta + 1/\varepsilon)$. The construction time is $O(n\log^3 n\log\Delta/\varepsilon^2)$.

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 [17] have shown that we can represent reachability in linear space, answering reachability queries in constant time. Also, Djidjev et al. [4] 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 [5] 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 [2] 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 time $O(\sqrt{r} \log r + \alpha(n))$.

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 exact distances in undirected planar graphs, nothing better than the above results was known. However, Arkati et al. [1] have shown that using $O(n\sqrt{n})$ space, we can answer distance queries with stretch[1] 2, which is much worse than our result for directed graphs. For the undirected case, we actually have a simpler $O(n \log n/\varepsilon)$ space oracle answering stretch $1 + \varepsilon$ distance queries in $O(1/\varepsilon)$ time.

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 [19] that one can construct a stretch 3 distance oracle in $O(n\sqrt{n})$ space with constant query time.

A nice feature of our oracles is that they distribute perfectly into labeling schemes [15]: 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 $O(\log \Delta \log n/\varepsilon)$, so that given $\ell(v)$ and $\ell(w)$, the distance from v to w with stretch $(1 + \varepsilon)$ in $O(\log \log \Delta + 1/\varepsilon)$ time. We note that labeling schemes are useful for both distributed computing and external memory.

In the case of labeling schemes, it is provably impossible to generalize our results to exact distances, even in the undirected case. Gavoille et al. [7] have recently shown that for undirected unweighted graphs, exact distances require a label size of $\Omega(\sqrt[3]{n})$ bits, and for undirected weighted graphs, the labels require $\Omega(\sqrt{n})$ bits.

In the journal version of this extended abstract it will be shown how to find and route along the short dipaths. Also, the results will there be generalized to graphs of bounded genus g, replacing a multiplicative factor $(\log n)$ with a factor $(\log n + g)$ in the space and construction time, and multiplying the query time by g.

1.1 Techniques

Most previous work on shortest dipaths and directed reachability in planar graphs uses somewhere the fact that they have separators of size $O(\sqrt{n})$. Applying this recursively, we end up with each vertex "seeing" $O(\sqrt{n})$ separator vertices, and given a any path P from v to w, there is a vertex $v \in P$ seen by both v and w. However, here we use separators that may be very large in terms of vertices, but which consist of a constant number of dipaths. More precisely, we end up with a *dipath decomposition* which is a family of dipaths $\mathcal{S}$ such that each vertex v "sees" $O(\log n \log \Delta)$ dipaths in $\mathcal{S}$, and given a shortest dipath P from u to w, there is a dipath $Q \in \mathcal{S}$ seen by both v and w which intersects P and which is of approximately the same length as P. Like the standard separators, our new dipath decompositions generalizes nicely to graphs of bounded genus g. Each vertex will then see $O((g + \log n) \log \Delta)$ dipaths in $\mathcal{S}$.

[1] Stretch t means no less, but up to a factor t more. Our $(1 + \varepsilon)$ approximation factor is really a stretch factor.

1.2 Notation

Given sets $X_0, X_1, ...,$ for $\circ\ = <, >, \leq, \geq$, we let $X_{\circ j}$ denote $\bigcup_{i \circ j} X_i$.

If G is a digraph, $V(G)$ and $E(G)$ denote the vertex and edge set. Let u, v, w be vertices in G. Then $u \rightsquigarrow_G v$ denotes that we can reach v from u in G, i.e., that there is a dipath from u to v in G, and then $\delta_G(v, w)$ is the distance from u to v, i.e., the shortest length of such a dipath. If the graph G is clear from the context, the subscript $_G$ will be omitted.

In this paper, 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* with leaf v.

2 Reachability

We will now describe our algorithm for constructing a reachability oracle for a planar graph G. As mentioned, our basic technique is to construct separators that are the the union of a constant number of dipaths. This is typically impossible, but we are going to construct a series of "2-layered" graphs $G_0, ..G_k$ so that any reachability question in G can be addressed to a constant number of these graphs, and such that each graph G_i admits separators consisting of a constant number of dipaths.

Below we first describe the construction of the 2-layered G_i, second we describe the actual dipath separators, third we describe how reachability queries can be answered via these separators, fourth we show how the query time can be tuned to be constant, and finally, we convert the oracle into a labeling scheme.

2.1 2-layered graphs

In this section, we will show how reduce reachability in G to reachability in some graphs with special properties. The reduction does not use planarity, but it does preserve planarity.

Definition 2.1 *A t-layered spanning tree T of 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 *Given a digraph G, in linear time, we can construct a series of digraphs $G_1, ..., G_k$ 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 $\imath(v)$ such that a vertex w is reachable from v in G if and only if it is reachable from v in $G_{\imath(v)-1}$ or $G_{\imath(v)}$.*

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

(iv) *Each G_i is a minor of G, that is, G_i is obtained from G by deletion 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 $L_0, ..., L_k$ where L_0 is the set of vertices reachable from an arbitrary vertex v_0, and thereafter, alternating, a layer consists of all vertices reachable from or reaching the previous layers. More formally, for $i > 0$, we define:

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

Then k is the first index such that $L_{\le k} = V$, and $\imath(v) = i$ for $v \in L_i$. Recall from §1.2 that $L_{\le k} = \bigcup_{i \le k} L$.

Each graph G_i consists of two consecutive layers with all preceding layers contracted to a single vertex. More formally, the graph G_i is obtained by taking the subgraph of G induced by $L_{\le i+1}$ and, for $i > 0$, contracting all vertices in $L_{<i}$ to the single root vertex r_i.

Trivially, the construction of the G_i preserves planarity, so (iv) is satisfied. 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 otherwise, 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 layer intersected by P, and let x be a vertex in the intersection. By definition, if $j \ge i$ is even, $L_{\le j}$ contains the part of P after x. Similarly, if $j \ge i$ is odd, $L_{\le j}$ 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 $G_{\imath(v)-1}$ and $G_{\imath(v)}$, so we conclude that our dipath P from v to w is contained in one of these two graphs. 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 augmented to a spanning tree of $\{r_i\} \cup L_i \cup L_{i+1} = V(G_i)$ with all edges oriented towards $\{r_i\} \cup L_i$. Now any path in T_i to r_i has a first part oriented towards r_i and a second part oriented away from 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 paper of the author [18]. However, having a quite different application in mind, that paper proceeded with each layer independently instead of combining them two by two.

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

2.2 Undirected planar spanning tree separation

In this section, we will take a 2-layered planar digraph H as from the last section, 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 the planar separator technique of Lipton and Tarjan [13], we will find two root paths in T whose removal separates the graph into components of at most $2/3$ of the size.

Lemma 2.3 *Given an undirected planar graph H with a rooted spanning tree T, in linear time, we can find two vertices v and w such that if we remove the vertices in the paths in T from v and w to the root, the components of $H \setminus V(T(v) \cup T(w))$ are at most $2/3$ the size of H. Here size generally just refers to number of vertices, but it may be based on any weighting of the vertices and edges.*

Proof: The proof is contained in [13], but somewhat hidden in other details because they need to ensure that the paths are of $O(\sqrt{n})$ length. The existence of v and w is what is actually proved in the proof of Lemma 2 in [13]. They find (v, w) as an edge in an arbitrary triangulation of H. No side of the fundamental cycle of (v, w) in T contains more than $2/3$ of H. The vertices v and w are found in linear time in steps 1, 8, and 9 of the partitioning algorithm in §3 in [13]. ■

2.3 Reachability via a dipath

Our next step is to show that we can efficiently represent directed reachability via a "separator" dipath. We are now forgetting that the digraph is planar. We note that the material in this subsection is very similar to the material in [16] on sparsifying intersection paths, though in [16], 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 are assuming 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 is suffices to identify the connections to Q. We use a simple recursive procedure. Let s be the first vertex in Q. Using, say, a breadth-first-search along edges in reverse direction, we find all vertices v reaching s. By definition, all these vertices connect to s in Q. We now remove all these vertices including s 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 to s. Here (a) follows because if a vertex x in Q reaches s, then so does all its predecessors in Q, and hence it is a prefix of Q that is removed. Also, concerning (b), if P was a dipath from a vertex v to Q and a vertex from P is removed, then v reached s. ■

2.4 The basic recursion

We are now going to combine the previous lemmas in a simple algorithm producing the reachability oracle with query time $O(\log n)$. In the next subsection, the query time will be reduced to a constant.

First, we apply Lemma 2.2, reducing our problem to dealing with 2-layered graphs. Consider any 2-layered graph H with 2-layered rooted spanning tree T. We now apply Lemma 2.3, producing a separator S consisting of a constant number of root paths, corresponding to a constant number of separator dipaths in H. In our recursion, the root r of T will typically be *suppressed*, meaning that we are not interested in connections via the root. In that case, before computing reachability via S, we remove the root from H and S. Thus, the root is really an auxiliary vertex needed for finding the separator with Lemma 2.3, but discarded for connectivity. We refer to the components of $H \setminus r$ as the *bodies* 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 $\hat{S}$. The resulting graph H/S is a 2-layer graph with spanning tree T/S rooted in $\hat{S}$. The root $\hat{S}$ is suppressed. Then v reaches w in H/S if and only if v reached w in $H \setminus V(S)$. We can now recurse on each component of $H \setminus V(S)$ as a separate body of H/S. Since the body sizes are reduced by a factor 3/2, the recursion depth is $O(\log n)$, so the total running time is $O(n \log n)$.

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. Then, 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 $q \in \{1, ..., s\}$, $\mathsf{to}_u[q] \leq \mathsf{from}_w[q]$. Since $s = O(\log n)$, this check takes $O(\log n)$ time.

To find s, we store with each call C, its *separation number* being 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 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 H from the rest of G_i. We call F the *frame* of H. If S is a separator of H, separating u from w, $F \cup S$ separates u from w in G_i. Keeping F and S of constant size, we will only have a constant number of separator dipaths to query to check reachability from u to w.

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* recursion. For both recursions, we apply the technique from §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 v and w so that no component of $H \setminus V(T(v) \cup T(w))$ contains more than $2/3$ of the vertices from H. In the frame reducing recursion, we pick v and w so that no component of $H \setminus V(T(v) \cup T(w))$ contains more than $2/3$ of the leaves of paths in F. Since every other recursion reduces the size 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 its frame F has f root paths, it can trivially pass down at most $f + 2$ root paths; namely those from F plus the two new root paths from v and w.

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 $Q \in F$ with leaf x. Note that as soon as the path Q intersects S on its way to the root, the rest of Q is contained in S. In particular, if $x \in S$, the path Q is subsumed by S. Hence, it is only components of $H \setminus V(S)$ containing x that need Q 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 $2f'/3$ leaves from F. Hence, including the at most two root paths from S, we pass at most $2f'/3+2$ frame paths down to the subsequent recursions.

In the root call on G_i, the frame is empty. Generally, starting from subgraph reducing recursion, we go from f to at most $f + 2$ to at most $2(f + 2)/3 + 2 = 2f/3 + 10/3$ frame paths for a next subgraph reducing recursion. Hence, the maximal number of frame paths for a subgraph reducing recursion is $f \leq 10$, and for a frame reducing recursion, it is $f' \leq 12$.

Reachability via framed separators In order to answer reachability queries in the body of G_i, we need all connections in 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 non-planar. 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 paths 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 in 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.

We can now apply Lemma 2.5 to $H \star F$ to find all connections in G_i between vertices in H and dipaths in S. We assume that we already know all connections between H and the dipaths in the frame F.

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 have already computed all the connections between H and F and S, so finding the "relevant" vertices to be included in F' is easy. In order to skip the other non-relevant vertices efficiently, we do it simultaneously for all components H'. More precisely, we take one dipath Q from $F \cup S$ at the 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 P 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 $O(n \log n)$.

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 $\{p + 1, ..., s\}$ are the indices of the frame and separator dipaths of C, so u reaches w in G_i if and only if $\exists q \in \{p + 1, ..., s\} : \mathsf{to}_u[q] \leq \mathsf{from}_w[q]$.

Applying Harel and Tarjan's [8] 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, we can determine if u reaches w in constant time.*

2.6 A pure label based implementation

As a last elaboration of our reachability oracle, we show how to distribute it 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 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 [14, §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 [8] to 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 label size $O(\log n)$.*

3 Approximate distances

In this section, we will generalize the approach from the previous section to give approximate distances. We will show that if we want to represent distances via a dipath Q of length x, and we are willing to accept an additive error of y, then it suffices to store $O(x/y + 1)$ connections between each vertex and Q. At present, there is no limit to the length of our separator dipaths, but we will show that for any scaling parameter α, we can find separator dipaths of length $\leq \alpha$ to separate vertices at distance $\leq \alpha$. Accepting an additive error of $\varepsilon\alpha$, we end up with $O(1/\varepsilon)$ connections for each vertex. The scheme is then repeated for exponentially increasing values of α.

3.1 $(3, \alpha)$-layered graphs

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

Lemma 3.2 *Given a digraph G, in linear time, we can construct a series of digraphs $G_1, ..., G_k$ satisfying (i) and (iv) from Lemma 2.2 and such that*

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

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

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

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

Again, k is the first index such that $L_{\leq k} = V$, and $\imath(v) = i$ for $v \in L_i$.

This time each graph G_i consists of three consecutive layers with all preceding layers contracted to a single vertex. More formally, the graph G_i is obtained by taking the subgraph of G induced by $L_{\leq i+2}$ and contracting all vertices in $L_{<i}$ to the single root vertex r_i.

The proof of (i) and (iv) are as in Lemma 2.2. Also, the proof for (iii)$^\alpha$ is very similar, except that we in the construction of T_i, for $j = i, i+1, i+2$, use shortest paths 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 §2.4.

To see that (ii)$^\alpha$ 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, $L_{\leq j}$ contains the part of P preceding x. Similarly, if $j > i$ is odd, $L_{\leq j}$ contains the part of P succeeding x. This is like in the proof of Lemma 2.2, except that there we could consider $j = i$, so we needed one less layer. 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. On the other hand, we know that $v \in P$ is only contained in $G_{\imath(v)-2}$, $G_{\imath(v)-1}$, and $G_{\imath(v)}$, so we conclude that our dipath P from v to w is contained in one of these three graphs. Thus (ii)$^\alpha$ is satisfied. ■

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

3.2 Approximate distances via a dipath

We are given a digraph H 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 some $\varepsilon > 0$.

We note that a similar undirected problem has been studied in [12], but in the undirected case, one can just consider connections via $1/\varepsilon$ equally spaced points in Q. The directed case is much more complicated.

A *connection* from v to Q is a new edge $(v, a) \in \{v\} \times V(Q)$ with length $\ell(v, a) \geq \delta(v, a)$, and similarly, a connection from Q to v is a pair $(a, v) \in V(Q) \times \{v\}$ with a length $\ell(a, v) \geq \delta(a, v)$. The connections and their lengths will be chosen by an algorithm below, with connection lengths being close to the distance between the end-points. If (u, a) and (b, w) are connections from u to Q and from Q to w, $\mathsf{dist}((u, a), (b, w)) = \ell(u, a) + \delta(a, b) + \ell(b, w)$ if a equals or precedes b in Q; otherwise, the value is ∞.

In order to compute $\mathsf{dist}((u, a), (b, w))$ efficiently, we can store with each vertex $c \in Q$, its distance $d(c, Q)$ from the start of Q. Then a equals or precedes b in Q if $a \leq b$, and then, since Q is a shortest path, $\delta(a, b) = d(b, Q) - d(a, Q)$.

Since connections to and from Q are symmetric, many definitions and results will be stated only for one direction, letting the other direction be understood implicitly. If v is a vertex in H and a and b are vertices in Q, we say that connection (v, a) *ε-covers* (v, b) if $a \leq b$ and $\ell(v, a) + \delta(a, b) \leq \delta(v, b) + \varepsilon\alpha$. 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.

If $C(u, Q)$ and $C(Q, w)$ are connections from u to Q and from Q to w, we define $\mathsf{dist}(C(u, Q), C(Q, w)) = \min_{(u,a)\in C(v,Q),(b,w)\in C(Q,w)} \mathsf{dist}((u, a), (b, w))$.

Fact 3.3 *Let $u, w \in V(H)$ with $\delta(u, w) \leq \alpha$ and the shortest path from v to w intersecting Q. If $C(u, Q)$ and $C(Q, w)$ are ε-covering, $\mathsf{dist}(C(u, Q), C(Q, w)) \leq \delta(v, w) + 2\varepsilon\alpha$.*

Proof: Let x be a point in which the shortest path from u to w intersects Q. Then $\delta(u, x) \leq \alpha$ so (u, x) is ε-covered by some $(u, a) \in C(u, Q)$. Similarly, (x, w) is ε-covered by some $(b, w) \in C(Q, w)$, and then $\mathsf{dist}((u, a), (b, w)) \leq \delta(u, w) + 2\varepsilon\alpha$. ■

We say that $C(v, Q)$ is *clean* if all connections are significant for distances from v to Q, i.e., if it contains no two distinct connections (v, a) and (v, b) such that $\ell(v, a) + \delta(a, b) \leq \ell(v, b)$.

Lemma 3.4 *If $C(u, Q)$ and $C(Q, w)$ are clean, we can determine $\mathsf{dist}(C(u, Q), (Q, w))$ in $O(|C(u, Q)| + |C(Q, w)|)$ time.*

Proof: We assume that the pairs in each of $C(v, Q)$ and $C(Q, w)$ are ordered according to the ordering of the vertices in Q, and then 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 (w, c). 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. ■

The next lemma implies that for any v, there is a clean ε-covering set $C(v, Q)$ of size at most $2/\varepsilon$ (let $D(v, Q) = \{v\} \times V(Q)$ with $\ell(v, a) = \delta(v, a)$ for $a \in V(Q)$. Then $D(v, Q)$ is 0-covering, so we use $\varepsilon_0 = 0$, $\varepsilon_1 = \varepsilon$).

Lemma 3.5 *Given an ε_0-covering set $D(v, Q)$ of connections from v to Q, we can construct a clean $(\varepsilon_0 + \varepsilon_1)$-covering set $C(v, Q) \subseteq D(v, Q)$ of size at most $(2 + \varepsilon_0)/\varepsilon_1$ in $O(|D(v, Q)|)$ time.*

Proof: First we delete any $(v, a) \in D(v, Q)$ with $\ell(v, a) > \alpha + \varepsilon_0\alpha$. Assuming this done, we visit the connections of $D(v, Q)$ following the order of the vertices in Q. Consider a connection (v, b) and let (v, a) be the last connection kept for $C(v, Q)$. We then add (v, b) to $C(v, Q)$ if $\ell(v, a) + \delta(a, b) > \ell(v, b) + \varepsilon_1\alpha$.

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

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

Lemma 3.6 *For each $v \in V(H)$, we can construct a ε-covering set $C(Q, v)$ of size $O(\log |V(Q)|/\varepsilon)$ in $O(\mathrm{SSSP}(H) \log |V(Q)|/\varepsilon)$ total time, where SSSP is the complexity of single source shortest paths computations in subgraphs of H.*

We note that Henzinger et al. [9] have shown that SSSP is linear when H is planar.

Proof: The proof of Lemma 3.6 is a bit technical, but the construction itself is quite simple and natural.

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

Define $\delta^{\leq\tau}(x,y) = \delta(x,y)$ if $\delta(x,y) \leq \tau$; ∞ otherwise. 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,v) with $\ell(s,v) = \delta^{\leq 2\alpha}(s,v)$ and (t,v) with $\ell(t,v) = \delta^{\leq 2\alpha}(t,v)$. We can now recurse on (Q,H).

Given (Q', H'), we recurse as follows. Let a be the first and c the last point in Q'. We now pick a vertex b in the unweighted middle of Q'. After a single source shortest path computation from b in H', for each $v \in V(H')$, we connect (a,v) with $\ell(a,v) = \delta^{\leq 2\alpha}_{H'}(a,v)$.

Next, let Q_1 be the part of Q' before b and let Q_2 be the part of Q' after b. Let U_1 to be the set of vertices v with (b,v) covering (a,v), that is, where $\delta(a,b)+\ell(b,v) \leq \ell(a,v)+\varepsilon\alpha$. Note that the (b,v) covers (a,v) with equality if $\ell(a,v) = \ell(b,v) = \infty$. Similarly, let U_2 to be the set of vertices v with (c,v) covering (b,v). Finally, set $H_1 = H' \setminus U_1$ and $H_2 = H' \setminus U_2$, and recurse on (Q_1, H_1) and (Q_2, H_2). •

In order to prove that our construction satisfies the statement of Lemma 3.6 we need to argue both that it produces ε-covering connections and that it is efficient.

Correctness Before descending into a rather messy proof, to see the basic idea behind the definition of U_1, assume $\ell(a,v) = \delta(a,v)$, and that x is between a and b in Q. Then $\delta(x,v)+\varepsilon\alpha \geq \delta(a,v) - \delta(a,x) + \varepsilon\alpha = \ell(a,v) + \varepsilon\alpha - \delta(a,x) \geq \delta(a,b) + \ell(b,v) - \delta(a,x) = \delta(x,b) + \ell(b,v)$, so (x,v) is covered by (b,v). Also, if $\delta(a,v) > 2\alpha$ and $\ell(a,v) = \infty$, we have $\delta(x,v) \geq \delta(a,v) - \delta(a,x) > \alpha$, and then (x,v) does not need covering. Unfortunately, our lengths are not necessarily that faithful.

In order to prove correctness, we will prove various properties of (Q', H') in beginning of each call. First, from the construction, we immediately have:

(1) If a is the first point in Q' and $v \in V(H')$, $\ell(a,v) \leq \delta_{H'}(a,v)$.

(2) If c is the last point in Q' and $v, w \in V(H')$, $\ell(a,w) \leq \ell(a,v) + \delta_{H'}(v,w)$.

In both inequalities above, we have assumed that the right hand side is at most 2α. We will now show:

(3) H' contains a prefix of Q'.

Clearly (3) is satisfied for the first call with (Q,H) since Q is contained in H. We will now show that (3) is preserved by the recursive step. Suppose a, x, y, b appears in this order in Q_1, possibly with $a = x$ or $y = b$. To reach a contradiction, suppose that x but not y is in U_1. Then y is in H', so by assumption on (Q', H'), x is in H'. Since $x \in U_1$, so $\delta(a,b) + \ell(b,x) \leq \ell(a,x) + \varepsilon\alpha$. However, from (1) and (3) on (Q', H') follows that $\ell(a,x) = \delta(a,x)$ and $\ell(a,y) = \delta(a,y)$. Hence, if we add $\delta(x,y)$ on either side of the inequality, we get $\delta(a,b) + \ell(b,y) \leq \delta(a,b) + \ell(b,x) + \delta(x,y) + \varepsilon\alpha \leq \ell(a,y) + \varepsilon\alpha$, contradicting that $y \notin U_1$. Hence H_1 contains a prefix of Q_1. The proof that H_2 contains a prefix of Q_2 is almost identical, this time just with x and y between b and c. Thus we conclude that (3) is satisfied for all (Q', H').

Next we show:

(4) Let x be in the interior of Q' and v be any vertex in H with $\delta(x,v) \leq \alpha$. If some shortest path P from a vertex x v in H leaves H', then (x,v) is ε-covered.

To show that the recursion on (Q_1, H_1) satisfies (2), let x be in the interior of Q_1 and let the shortest path P from x to v leave H_1. We need to show that (x,v) is ε-covered. If P leaves H', we know that (x,v) is ε-covered by (4) on (Q', H'), so we can assume that P remains in H'. Let u be the first point at which P leaves H_1. Then, using subscripts to indicate the reason for the (in)equalities, $\delta(x,v) = \delta_{H'}(x,u) + \delta_{H'}(u,v) \geq_{(3)} \delta_{H'}(a,u) - \delta(a,x) + \delta_{H'}(u,v) \geq_{(1)} \ell(a,u) - \delta(a,x) + \delta_{H'}(u,v) \geq_{\text{def}.U_1} \delta(a,b) + \ell(b,u) - \varepsilon\alpha - \delta(a,x) + \delta_{H'}(u,v) \geq_{\ell(b,\cdot)=\delta^{\leq 2\alpha}_{H'}(b,\cdot)} \delta(a,b) + \ell(b,v) - \varepsilon\alpha$. Hence (x,v) is ε-covered by (b,v).

The proof that (Q_2, H_1) satisfies (2) is very similar. This time, we have x in the interior of Q_2 and P leaving H_2, and again we can assume that P stays in H'. Let u be the first point at which P leaves H_2. Then $\delta(x,v) = \delta_{H'}(x,u) + \delta_{H'}(u,v) \geq \delta_{H'}(b,u) - \delta(b,x) + \delta_{H'}(u,v) \geq_{\ell(b,\cdot)=\delta^{\leq 2\alpha}_{H'}(b,\cdot)} \ell(b,u) - \delta(b,x) + \delta_{H'}(u,v) \geq_{\text{def}.U_2} \delta(b,c) + \ell(c,u) - \varepsilon\alpha - \delta(b,x) + \delta_{H'}(u,v) \geq_{(2)} \delta(x,c) + \ell(c,v) - \varepsilon\alpha$. Hence (x,v) is ε-covered by (c,v). This completes the proof of (4).

From (4) we can generally conclude that we get an ε-covering set of connections, for let $(b,v) \in V(Q) \times V(H)$, $\delta(b,v) \leq \alpha$. If b is the first or last point in Q, we connect (b,v) with $\ell(b,v) = \delta(b,v)$ before the recursion starts. Otherwise, consider the recursion $(Q'.H')$ with b the middle vertex in Q'. From (4) it follows that if (b,v) is not ε-covered by previous recursions, v is in H' and $\delta_{H'}(b,v) = \delta(b,v)$, so we connect (b,v) with $\ell(b,v) = \delta(b,v)$. •

Efficiency In each call (Q', H'), we spend $O(|V(H')|)$ time, assuming that the single source shortest path computation from b can be done in linear time. The result therefore follows if we can show that each vertex v in H appears in H' in at most $\log_2 |Q|/\varepsilon$ calls. Consider a call (Q', H') involving v. We are interested in the value $d' =$

$\delta(a,c) + \ell(c,v) - \ell(a,v)$. The corresponding values for (Q_1, H_1) and (Q_2, H_2) are $d_1 = \delta(a,b) + \ell(b,v) - \ell(a,v)$ and $d_2 = \delta(b,c) + \ell(c,v) - \ell(b,v)$. Then $d' = d_1 + d_2$. The definition of U_i, $i = 1, 2$, says that $d_i > \varepsilon\alpha$ if $v \notin U_1$. Thus, if v is passed down to both recursions, both d_i are bigger than $\varepsilon\alpha$. Also, if $d_1 = \infty$, $\ell(b,v) = \infty$, so $v \in U_2$, and the largest finite value is 2α, so the doubling can happen at most $2/\varepsilon$ times. On the other hand, each recursion halves the size of the interior of Q', so the recursion depth is at most $2\log_2 |V(Q)|$. It follows that v can be involved at most $2\log_2 |V(Q)|/\varepsilon$ times. •

Having established both correctness and efficiency of our construction, we conclude that Lemma 3.6 is satisfied. ■

Corollary 3.7 *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 $O(|V(H)| \log |V(Q)|/\varepsilon)$ total time, assuming that we are dealing with digraphs permitting single source shortest paths computations in linear time.*

Proof: First we apply Lemma 3.6 with $\varepsilon = \varepsilon/2$, and second we apply Lemma 3.5 with $\varepsilon_0 = \varepsilon_1 = \varepsilon/2$ to each set of connections. ■

3.3 The basic recursion

We can use the same basic recursion as in §2.4, but the connections between the vertices in H and the separator dipaths Q are those from §3.2 instead of those from §2.3. As a result, the preprocessing time is increased by a factor $O(\log n/\varepsilon)$ to $O(n \log^2 n/\varepsilon)$ and the oracle space is increased by a factor $O(1/\varepsilon)$ to $O(n \log n/\varepsilon)$ (c.f. Corollary 3.7 versus Lemma 2.5). Also, the query time is increased by a factor $O(1/\varepsilon)$ to $O(\log n/\varepsilon)$. Recall here that these bounds are only for finding distances below α with an additive error of $O(\varepsilon\alpha)$.

3.4 Reducing the query time

Using the separations with frames from §2.5, it is easy to see that we can reduce query time to $O(1/\varepsilon)$. More precisely, the recursion in §2.5 finds a system of dipaths $\mathcal{S}$ so that given two vertices, in constant time, we can identify a constant number of dipaths that are guaranteed to separate the vertices. This time, however, we spend $O(1/\varepsilon)$ time on querying a dipath. The space is again going to be $O(n \log n/\varepsilon)$. As for the reachability oracle, it can be distributed on vertex labels, each of size $O(\log n/\varepsilon)$.

Our problem in doing the recursion is that if we try to to construct connections based on connections from previous recursions, the errors add up. We resolve this by using an error of $1/\mu = \varepsilon/\log n$ for the construction, which with a recursion depth of $O(\log n)$ becomes an $O(\varepsilon)$ error at the end. Now, considering a recursive call, each vertex has $O(1/\mu)$ connections, so the graph $H \star F$ has size $O(|V(H)|/\mu)$. Applying Corollary 3.7 with a standard single source shortest paths algorithm, we construct the connections to the dipaths in the frame and separator in $O(|V(H)| \log^2 n/\mu^2)$ time, thus ending up with a total construction time of $O(n \log^3 n/\mu^2) = O(n \log^5 n/\varepsilon^2)$. Afterwards, each set $C(v,Q)$ has size $O(1/\mu)$ but can be reduce it to size $O(1/\varepsilon)$ with Lemma 3.5. Our oracle is now of size $O(n \log n/\varepsilon)$, as desired.

3.5 Finding α

In order to get stretch $1 + \varepsilon$ distances of arbitrary sizes, we repeat the above construction with $\alpha = 2^i$ for $i = 1, ..., \log_2 \Delta$. We will do it with maximal error of both $\alpha/2$ and $\varepsilon\alpha/4$. The idea of the constant relative error is to quickly approximate the distance from u to w within a constant factor, exploiting that the query time is constant for each α. More precisely, we make a binary search over the $\log_2 \Delta$ values of i,, we look for a value of $\alpha = 2^i$ such that $\alpha/4 \le \hat{\delta}^{(\alpha,\alpha/2)}(u,w) \le \alpha$. Here $\hat{\delta}^{(\alpha,\beta)}$ denotes the distances estimated by the construction with distance bound α and maximal additive error of β. We note that the condition is always satisfied for $i = \lfloor \log_2 \delta(u,w) \rfloor - 1$, and possibly also for $i = \lfloor \log_2 \delta(u,w) \rfloor - 1$. Now, $\hat{\delta}^{(\alpha,\varepsilon\alpha/4)}(u,w)$ gives us the desired approximation since $\hat{\delta}^{(\alpha,\varepsilon)}(u,w)/\delta(u,w) = 1 + (\varepsilon\alpha/4)/(\alpha/4) = 1 + \varepsilon$. The total query time is $O(\log\log \Delta + 1/\varepsilon)$.

Theorem 3.8 *We can preprocess a planar digraph in $O(n \log \Delta \log^5 n/\varepsilon^2)$ time, producing an $O(n \log \Delta \log n/\varepsilon)$ space distance oracle that can answer stretch $1 + \varepsilon$ distance queries in $O(\log\log \Delta + 1/\varepsilon)$ time. The oracle can be distributed as a labeling scheme with $O(\log \Delta \log n/\varepsilon)$ space labels.*

In the journal version, we will further improve the construction time to $O(n \log \Delta \log^3 n/\varepsilon^2)$.

3.6 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 §2.2. Thereby, in the construction time and space, we avoid paying a multiplicative $\log \Delta$ for the exponentially increasing α . Also, we avoid the additive $\log\log \Delta$ in the query time.

Our problem now is that the separator paths are of unbounded length. We want to construct connections similar to those in §3.2, but first we need to make an appropriate redefinition of the ε-covering by connections from a vertex v to an undirected separator path Q. Let $\delta(v, Q)$ denote the distance from v to the nearest vertex in Q. If a shortest path from v to another vertex w intersects Q, $\delta(v, w) \geq \delta(v, Q)$, and hence we can tolerate an error of $\varepsilon\delta(v, Q)$ when estimating the distance to w. For $a, b \in Q$, we therefore define that a connection (v, a) *ε-covers* (v, b) if $\ell(v, a) + \delta(a, b) \leq \ell(v, a) + \varepsilon\delta(v, Q)$.

Lemma 3.9 *There is an ε-covering set of connections from v to Q of size $O(1/\varepsilon)$.*

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

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

The same procedure is applied towards the other end-point of Q, so we end up with at most $1 + 2/\varepsilon$ 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 in [11].

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 (v, a) *ε-covers* (v, b) if $\delta(v, b) \leq (1 + \varepsilon)\ell(v, a) + \delta(a, b)$. It is for technical reasons that that the right hand side is not the more natural $(1 + \varepsilon)(\ell(v, a) + \delta(a, b))$. We can now use the construction from the proof of Lemma 3.6, but with U_1 being the set of vertices v where one of (v, a) and (v, b) cover the other, and with U_2 being the set of vertices v where one of (v, b) and (v, c) cover the other. With these modifications, Lemma 3.6 holds in the undirected case. Since we avoided the scaling with α, we end up with a construction time of $O(n \log^5 n/\varepsilon^2)$. As for the directed case, it can be improved to $O(n \log^3 n/\varepsilon^2)$ using some more involved methods to be revealed in the journal version.

Acknowledgment I would like to thank Uri Zwick and a FOCS'01 referee for thorough and and useful comments.

References

[1] S. Arikati, D. Chen, L. Chew, G. Das, M. Smid, and C. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proceedings of the 4th European Symposium on Algorithms, Barcelona, Spain*, pages 514–528, 1996.

[2] D. Chen and J. Xu. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, Oregon*, pages 469–478, 2000.

[3] H. Djidjev. 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*, pages 151–165, 1996.

[4] H. Djidjev, G. Panziou, and C. Zaroliagis. Computing shortest paths and distances in planar graphs. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming, LNCS 1991*, pages 327–339, 1991.

[5] H. Djidjev, G. Panziou, and C. Zaroliagis. Fast algorithms for maintaining shortest paths in outerplanar and planar digraphs. In *Proceeding of 10th International Symposium on Fundamentals of Computation Theory, LNCS 965*, pages 191–200, 1995.

[6] G. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.

[7] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, D.C.*, pages 210–219, 2001.

[8] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestor. *SIAM Journal on Computing*, 13:338–355, 1984.

[9] M. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55:3–23, 1997.

[10] P. Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing, Atlanta, Georgia*, pages 428–434, 1999.

[11] P. Klein. Preprocessing and undireted planar network to enable fast approximate distance queries. Manuscript, 2001.

[12] P. Klein and S. Subramanian. A fully dynamic approximation scheme for shortest path problems in planar graphs. *Algorithmica*, 23:235–249, 1998.

[13] R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[14] D. Peleg. Informative labeling schemes for graphs. In *Proc. 25th Mathematical Foundations of Computer Science, LNCS 1893*, pages 579–588, 2000.

[15] D. Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33:167–176, 2000.

[16] S. Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *Proc. 1st European Symp. Algorithms (ESA), LNCS 726*, pages 372–383, 1993.

[17] R. Tamassia and I. Tollis. Dynamic reachability in planar digraphs with one source and one sink. *Theoretical Computer Science*, 119:331–343, 1993.

[18] M. Thorup. Shortcutting planar digraphs. *Combinatorics, Probability & Computing*, 4:287–315, 1995.

[19] M. Thorup and U. Zwick. Approximate distance oracles. In *Proceedings of the 33th Annual ACM Symposium on Theory of Computing, Crete, Greece*, 2001.