Chapter 11

Approximate distance queries

11.1 Preprocessing a planar graph to support arbitrary vertex-to-vertex approximate-distance queries

We will discuss ways to preprocess a planar graph to produce representations that support vertex-to-vertex distance queries.

There are techniques for supporting exact distance queries and approximate distance queries. We will (at least for now) focus on approximate distances.

For approximate distances, we have another input, a parameter $\epsilon > 0$. A query DISTANCE(u, v) will be answered with the length of a u-to-v path in G such that the length is at most $1 + \epsilon$ times the u-to-v distance.

For approximate distances, Mikkel Thorup has given a method for directed graphs and a technique for undirected graphs.

One key tool is the fundamental-cycle separator of edges (Lemma 5.3.4:

Lemma: *There is a linear-time algorithm that, given a triangulated plane graph G with a $\frac{1}{3}$-proper assignment of weights to edges, and a spanning tree T, returns a nontree edge $\hat{e}$ such that the fundamental cycle of $\hat{e}$ with respect to T is a $\frac{2}{3}$-balanced cycle separator for G.*

Let G be a planar embedded graph and let C be a simple cycle. The two *pieces* of G with respect to C are (1) the subgraph consisting of all edges enclosed by C and (2) the subgraph consisting of all edges not strictly enclosed by C. Note that each piece contains the edges of C itself.

We can use this lemma recursively: A *recursive decomposition* of a graph G_0 is a rooted binary tree R where

- each node x of R is labeled with a subgraph $g(x)$ of G,

- $g(\text{root}) = G_0$,
- each nonleaf node x of R is labeled with a separator $s(x)$ of $g(x)$,
- the two children of a nonleaf node x are the subgraphs resulting from "applying" the separator to $g(x)$.

The decomposition is *complete* if the leaf subgraphs are small enough according to some measure $m(\cdot)$ of graph size, i.e. for every leaf x, $m(g(x)) \leq c$ for a constant c.

For the present application, we start with a planar embedded graph G_0 with nonnegative edge-lengths and a spanning tree T. For distances, we can assume no self-loops and no parallel edges. The algorithm first adds infinite-length edges as needed to ensure that each face is a triangle. Next the algorithm finds a recursive decomposition using fundamental-cycle separators. The subgraphs resulting from applying a fundamental-cycle separator C are the pieces with respect to C.

Each separator in the decomposition is a fundamental cycle C. Note that C consists of (1) a nontree edge uv, together with (2) the $\text{lca}_T(u, v)$-to-u path in T, and (3) the $\text{lca}_T(u, v)$-to-v path in T. This implies that there are two leafward paths such that every vertex on the separator lies on at least one of the paths (the least common ancestor lies on both).

We say a path P *crosses* a simple cycle C if P contains both an edge strictly enclosed by C and an edge not enclosed by C.

Lemma 11.1.1. *Consider a recursive decomposition R of G_0 using fundamental-cycle separators. Let P be a path in G_0. If P crosses $s(x)$ and $s(y)$ then P crosses $lca_R(x, y)$.*

It follows that, for any path P, there is a unique rootmost node z in R such that P crosses $s(z)$.

Strategy for shortest-path approximation: Let u and v be two given vertices. Assume u and v do not belong to the same leaf subgraph of R. Then there is a leafmost node x of R such that $u, v \in g(x)$. Let P be the shortest u-to-v path in G_0. Then the rootmost node z such that P crosses $s(z)$ is an ancestor of x. For each ancestor y of x, therefore, the algorithm will estimate the minimum length of a path that (i) crosses $s(y)$ but (ii) does not cross $s(y')$ for any proper ancestor of y in R.

Therefore we turn to the problem of estimating paths that cross a separator $s(y)$. We will use the fact that the vertices of $s(y)$ belong to two leafward paths. In fact, we will choose the spanning tree T to make this easier.

For now, we will consider only undirected graphs. We can assume the graph is connected. Choose an arbitrary root r, and let T be a shortest-path tree rooted at r. It will follow that each of the two leafward paths making up a separator is itself a shortest path.

Let G be a planar embedded graph, let s be a fundamental-cycle separator, and let $\mathcal{P}$ be the set consisting of the two paths comprising s. For each path $P \in \mathcal{P}$, for each vertex v of G, we will select a set of vertices $r \in V(P)$ for

which we will record the r-to-v distance in G. We will call the pair (r, v) a *connection* for v, and we will call r a *connectee* for v. Our construction will have the following two properties.

(1) For each vertex v, the number of connections for v is at most $8/(\epsilon - \epsilon^2)$.

(2) For any two vertices u, v, the shortest u-to-v path in G that crosses s is ϵ-approximated by the shortest u-to-v path in G that goes from u to a connectee for u, then along a path $P \in \mathcal{P}$ containing r_u to a connectee for v, then to v.

Now we give an algorithm that, for a given vertex v and a path P, selects some vertices of P to be connectees of v.

Let r_0 be the vertex of P that is closest to v among all nodes of P. For every node r of P, define $h(r) = dist(r, r_0)$. The algorithm designates r_0 as a connectee for v. It then uses two phases, a forward phase and a backward phase, to select more connectees. The forward phase selects connectees $r_1, r_2, \ldots$ and the backward phase selects connectees $r_{-1}, r_{-2}, \ldots$.

We describe the forward phase. It considers vertices r of P one by one, in leafward order.

```
for i = 0, 1, 2, ...
    let r be the first vertex of P after r_i such that
            (1 + ε)dist(v, r) < dist(v, r_i) + h(r) − h(r_i)
    comment: h(r) − h(r_i) = length of r_i-to-r subpath of P = dist(r_i, r).
    r_{i+1} := r
until there is no such vertex r
```

In view of the comment, the expression $dist(v, r_i) + h(r) - h(r_i)$ is the length of an indirect path from v to r which goes via a shortest path to r_i, thence along P to r. Thus the condition in the procedure holds if the direct shortest v-to-r path is much shorter than the indirect path.

Let $r_1, \ldots, r_k$ be the connectees chosen by the forward phase.

Lemma 11.1.2. *For any vertex r of P that is leafward of r_0, there is a connectee r_i that is rootward of r such that*

$$dist(v, r_i) + dist(r_i, r) \leq (1 + \epsilon) dist(v, r) \tag{11.1}$$

Proof. Let r_i be the last vertex of P designated a connectee before a vertex after r was considered. If $r_i = r$ then (11.1) holds trivially. If not, the inequality in the procedure did not hold at the time r was considered, so we have

$$(1 + \epsilon) dist(v, r) \geq dist(v, r_i) + h(r) - h(r_i)$$

which is equivalent to (11.1). □

Lemma 11.1.3. *The number k of connectees chosen by the forward phase is less than $2/(\epsilon - \epsilon^2)$.*

Proof. By Taylor series expansion, $(1+\epsilon)^{-1} < 1-(\epsilon-\epsilon^2)$. The choice of r_{i+1} guarantees

$$\begin{aligned} dist(v, r_{i+1}) &< (1+\epsilon)^{-1}\left(dist(v, r_i) + h(r_{i+1}) - h(r_i)\right) \\ &\leq (1+\epsilon)^{-1} dist(v, r_i) + h(r_{i+1}) - h(r_i) \\ &\leq dist(v, r_i) - (\epsilon-\epsilon^2) dist(v, r_i) + h(r_{i+1}) - h(r_i) \\ &\leq dist(v, r_i) - (\epsilon-\epsilon^2) dist(v, r_0) + h(r_{i+1}) - h(r_i) \quad (11.2) \end{aligned}$$

Therefore we obtain the recurrence relation

$$dist(v, r_{i+1}) - h(r_{i+1}) < dist(v, r_i) - h(r_i) - (\epsilon-\epsilon^2) dist(u, r_0)$$

which yields

$$dist(v, r_i) - h(r_i) < dist(v, r_0) - h(r_0) - i(\epsilon-\epsilon^2) dist(v, r_0)$$

Recall that $h(r_i) = dist(r_0, r_i)$. Using the fact that $h(r_i) = 0$, we have

$$dist(v, r_i) - h(r_i) < dist(v, r_0) - i(\epsilon-\epsilon^2) dist(v, r_0) \quad (11.3)$$

By the triangle inequality, the r_0-to-r_i distance is at most the r_0-to-v distance plus the v-to-r_i distance, so

$$h(r_i) \leq dist(v, r_0) + dist(v, r_i)$$

which is equivalent to

$$dist(v, r_i) - h(r_i) \geq -dist(v, r_0)$$

which, combined with (11.3), yields

$$-dist(v, r_0) < dist(v, r_0) - k(\epsilon-\epsilon^2) dist(v, r_0)$$

where k is the number of connectees chosen by the forward phase. We therefore obtain

$$k < 2/(\epsilon-\epsilon^2)$$

□

Remark: In the proof of Lemma 11.1.3, we used (11.2), which is weaker than the inequality actually used in the procedure. We could replace the inequality used in the procedure with (11.2), and Lemma 11.1.3 would still hold. Lemma 11.1.2 would also still hold.

11.2 Efficient construction

We give an algorithm to find such a set S of connections covering all vertices of G such that each vertex v has $4/(\epsilon-\epsilon^2)$ connections.

The algorithm applies the MSSP algorithm to G and the path P to find, for each $0 \leq i < k$, the sequence A_i of pivots that transform the r_i-rooted shortest-path tree into the r_{i+1}-rooted shortest-path tree (ordered so that each intermediate result is still a tree). Each pivot is represented by a triple (uw, α, vw) where uw is the arc to be removed, vw is the arc to be inserted, and α is the decrease in the distance to the w-rooted subtree.

The algorithm constructs an auxiliary graph from G by adding an artificial root $\hat{r}$ and zero-length arcs $\hat{r}r_i$ to the vertices r_i of P. Next, the algorithm finds a shortest-path tree $\hat{T}$ of the auxiliary graph rooted at $\hat{r}$. For each i, let $\hat{T}_i$ be the subtree of $\hat{T}$ rooted at r_i. For each vertex $v \neq \hat{r}$, let $\hat{i}(v)$ denote that integer i such that v belongs to $\hat{T}_i$. That is, $dist(r_{\hat{i}(v)}, v) = \min_i dist(r_i, v)$.

The algorithm next performs two phases, a forward phase and a backward phase. In each phase, the algorithm designates pairs $(r, v) \in V(P) \times V(G)$ as connections. The output of the algorithm is the set of all pairs designated as connections. We describe the forward phase; the backward phase is similar.

At any point in the running of the phase, for a vertex v, let $r(v)$ denote the vertex r such that (r, v) was the last connection designated for v, or $r(v) = \perp$ if no connection has yet been designated during the phase.

The algorithm maintains a dynamic-tree representation of a tree T of G. The dynamic tree supports costs assigned to vertices, with descendant bulk updates and descendant searches. The cost of v is denoted $\sigma(v)$.

The dynamic tree also maintains for each vertex v a label $\boldsymbol{d}[v]$ satisfying

$$\boldsymbol{d}[v] = \text{root-to-}v \text{ distance in } T \tag{11.4}$$

The algorithm maintains the following invariant. Let r be the root of T. For each vertex v, if $r(v) \neq \perp$ then

$$\sigma(v) = (1 + \epsilon)\boldsymbol{d}[v] - (dist(r, r(v)) + dist(r(v), v)) \tag{11.5}$$

Therefore, if $\sigma(v)$ is nonpositive, the need to cover v suggests that (r, v) be designated a connection.

```
initialize T to be the r_0-rooted shortest-path tree
for each vertex v, initialize d[v] to be the length of the root-to-v path
initialize σ(v) := ∞ for every vertex v
for i := 0, 1, 2, ..., k,
1   for each vertex v in T̂_i,
2       designate (r_i, v) a connection
3       assign σ(v) := ε d[v]
4   while there exists a vertex v with σ(v) < 0,
5       designate (r_i, v) a connection
6       assign σ(v) := ε d[v]
7   if i < k,
8       comment: reroot the tree by r_{i+1}
```

```
9    remove the arc of T entering r_{i+1} and add the arc r_{i+1}r_i
11   for every v, increase d[v] by ℓ(r_{i+1}r_i)
12   set d[r_{i+1}] := 0
13   for every v, increase σ(v) by ϵℓ(r_{i+1}r_i)
14   comment: Carry out pivots.
15   for each (uw, α, vw) ∈ A_i,
16   remove uw from T and insert vw
17   subtract α from d[w'] for every vertex w' in the w-rooted tree
18   subtract (1 + ϵ)α from σ(w') for every vertex w' in the w-rooted tree
```

11.2.1 Correctness

The algorithm ensures that, for every vertex v, equations (11.4) and (11.5) hold. These hold immediately after the initializations. In Line 2 and in Line 5, a new connection is designated for v. The assignment to $\sigma(v)$ in Lines 3 and 6 preserve the invariant (11.5). After the change of root in Line 9, Line 11 and 13 restore (11.4 and (11.5) for every vertex v except r_{i+1}. Line 12 restores (11.4) for $v = r_{i+1}$. Since r_{i+1} belongs to $\hat{T}_{i+1}$, $r(r_{i+1}) = \perp$ so the invariant does not require 11.5 to hold for $v = r_{i+1}$.

The pivots in Lines 15-16 change the tree T, but the updates in Lines 17 and 18 restore (11.4) and (11.5).

Claim: The forward phase ensures that, after iteration i, for each vertex v, if $i \geq \hat{i}(v)$ then there is a connection (r_j, v) such that $(1 + \epsilon)dist(r_i, v) \geq dist(r_i, r_j) + dist(r_j, v)$.

Proof. If at the beginning of iteration i we have

$$(1 + \epsilon)dist(r_i, v) < dist(r_i, r(v)) + dist(r(v), v)$$

then v is selected in some iteration of the while-loop of Line 4, and (r_i, v) is designated a connection. □

11.2.2 Running time

Each iteration of the while-loop in Line 4 takes amortized $O(\log n)$ time. The number of iterations is the total number of connections established, which is $O(n\epsilon^{-1})$. Lines 9-13 takes $O(\log n)$, and these are executed $k \leq n$ times, for a total of $O(n \log n)$ time. Each execution of Lines 16-18 takes $O(\log n)$ time. Over the course of the whole phase, the number of iterations of the loop in Line 15 is $O(m)$, which is $O(n)$, so the total time for Lines 15-18 over the course of the algorithm is $O(n \log n)$. Thus the total time is $O(n\epsilon^{-1} \log n)$.