Compact Oracles for Approximate Distances Around Obstacles in the Plane

Mikkel Thorup

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

Abstract. Consider the Euclidean plane with arbitrary polygonal obstacles with a total of n corners. For arbitrary $\varepsilon > 0$, in $O(n(\log n)^3/\varepsilon^2)$ time, we construct an $O(n(\log n)/\varepsilon)$ space oracle that given any two points reports a $(1+\varepsilon)$ approximation of the obstacle avoiding distance in $O(1/\varepsilon^3 + (\log n)/(\varepsilon \log\log n))$ time. Increasing the oracle space to $O(n(\log n)^2/\varepsilon)$, we can further report a corresponding path in constant time per hop.

1 Introduction

As described by Hershberger and Suri [1], "the Euclidean shortest path problem is one of the oldest and best-known problems in computational geometry. Given a planar set of arbitrary polygonal obstacles with disjoint interiors, the problem is to compute a shortest path between two points avoiding all the obstacles. Due to its simple formulation and obvious applications in routing and robotics, the problem has drawn the attention of many researchers in computational geometry..." Here an obstacle avoiding path is allowed to touch obstacles, but not to cross them. This implies that a shortest path only bends at obstacle corners. With n denoting the total number of corners, Hershberger and Suri proceed to show that this off-line problem can be solved for any two points a and b in $O(n \log n)$ time, which is optimal.

The problem we consider here is to preprocess the plane with the obstacles so that we can quickly estimate the distance between any pair of points, and report a corresponding path. This kind of preprocessing for two-point queries problem has a rich history [2].

We consider the approximate version, where the distances and paths reported have stretch $1+\varepsilon$ for an arbitrarily small $\varepsilon > 0$. Formally, a distance estimator d has stretch s relative to the real distance function δ if for each pair of points a and b, we have $\delta(a,b) \leq d(a,b) \leq s\,\delta(a,b)$. Our main result is as follows.

Theorem 1. *Consider the Euclidean plane with polygonal obstacles with a total of n corners. Let $\varepsilon > 0$. In $O(n(\log n)^3/\varepsilon^2)$ time we can produce an $O(n(\log n)/\varepsilon)$ space distance oracle that estimates arbitrary obstacle avoiding distances with stretch $1+\varepsilon$ in $O(1/\varepsilon^3 + (\log n)/(\varepsilon \log\log n))$ time. Increasing the space to $O(n(\log n)^2/\varepsilon)$, we can report a corresponding path in constant time per hop.*

L. Arge, M. Hoffmann, and E. Welzl (Eds.): ESA 2007, LNCS 4698, pp. 383–394, 2007.

In the above bounds, we have assumed that our algorithm is implemented in a standard imperative programming language such as C [3] and that point locations and distances are represented as integers or floating point numbers. Such representation is certainly reasonable when aiming for approximate distances. In particular, we can use distance values to compute addresses as done in bucket and radix sort, and this allows us to compute undirected single source shortest paths in linear time [4,5]. The addressing also allows us to compute nearest common ancestors in a rooted tree in constant time [6]. On a comparison-addition based pointer machine, some of the bounds would grow by a factor $O(\log n)$. Our construction works not only for the Euclidean plane but for any ℓ_p norm.

Closely related work. Based on Clarkson's [7] approximate solution to the off-line problem, Chen [8] shows that a data structure can be built in near-quadratic time and space, estimating distances with stretch $1+\varepsilon$ in logarithmic time. For now we think of ε as a small positive constant. Chen [8] also constructs a near-linear space data structure in $\tilde{O}(n^{3/2})$ time which estimates distance with stretch $6+\varepsilon$ in logarithmic time. As a later improvement, Arikati et al. [9] show that using near-linear space, we can answer rectilinear distance queries with stretch $2+\varepsilon$ in logarithmic time. For the Euclidean metric, this implies stretch $\sqrt{2}(2+\varepsilon)$. Chiang and Mitchell [10] show that we can get exact answers in logarithmic time, but their data structure uses $O(n^{11})$ space. Gudmundson et al. [11,12] show that if there is a constant bounding the ratio of direct Euclidean distances over obstacle avoiding distances, then there is a near-linear space data structure answering distance queries with stretch $1+\varepsilon$ in constant time. However, the bounded ratio rules out thin obstacles like rivers.

Varadarajan [13] pointed out a construction implicit in previous work leading to near-linear space oracle like the one claimed in Theorem 1, providing stretch $1+\varepsilon$ distances in sub-logarithmic time. However, the previous oracle construction is more complicated and slower by a factor $1/\varepsilon^4$, which matters in theory when ε is $o(1)$, and in practice if, say, $\varepsilon < 1/5$. The construction uses a $1+\varepsilon$ spanner graph over the obstacle corners claimed by Arikati et al. [9] in a final remark. They provide no details, but Zeh [14, Lemma 14.18] calculates the size of this planar spanner to be $O(n/\varepsilon^4)$. A distance oracle for this planar spanner is constructed using Klein [15] or Thorup [16]. Finally Chen [8] has shown how to use a corner distance oracle to get obstacle avoiding distances between arbitrary pairs of points.

The technical contribution of this paper is to bypass the spanner construction above and instead modify the planar graph distance oracle construction from [16] to work directly with obstacles in the Euclidean plane. Instead of working with the planar spanner with $O(n/\varepsilon^4)$ vertices and edges, we work with Clarkson's [7] cone graph whose vertices are the n obstacle corners connected by $O(n/\varepsilon)$ edges. The extra edges do not affect the running time because the algorithm internally works with even more auxiliary edges. The fewer vertices gain us a factor $1/\varepsilon^4$ in construction time. The cone graph data structure is used internally in the previous construction, so it does not add new complexity. The cone graph is highly non-planar, but in a way that can be cut efficiently by the planar path

separators from [16]. For contrast, the traditional planar vertex separators of Lipton and Tarjan [17] cannot cut the cone graph.

2 Borrowed Framework and Techniques

Instead of the parameter ε, we will use an inverse integer parameter k, and get a stretch of $1 + O(1/k)$. We will use a technique of Clarkson [7] the off-line approximate shortest path problem among polygonal obstacles.

For any point a, the view is divided into k cones. Cone $i \in [k]$ covers directions between $360°(i-1/2)/k$ and $360°(i+1/2)/k$. Here, $0°$ represents some fixed base direction. For any cone i, we want to know the closest visible obstacle corner v. Here closeness is measured as projected on the central cone direction $360°i/k$. Unless otherwise stated, we will always have this understanding of closeness to a within a given cone i from a. Clarkson [7] presented a data structure for this problem called a *cone Voronoi diagram*. The cone Voronoi diagram is constructed in $O(nk \log n)$ time and $O(nk)$ space, and it supports queries in $O(\log n)$ time.

Clarkson [7] constructs a cone graph G whose vertices are the obstacle corners. For each corner u, he uses the cone Voronoi diagram to find the nearest corner $v_i \neq u$ in each of the k cones, and add the *cone edge* (u, v_i). This cone edge embeds into the straight line from u to v_i in the plane. The line does not cross any obstacle, so a path in G embeds into an obstacle avoiding path. Clarkson proves that a shortest path in G has stretch $1 + O(1/k)$ compared with shortest obstacle avoiding path in the plane. The cone graph G may be very far from planar in that the embedding of cone edges may cross arbitrarily.

Note that a cone edge (u, v_i) is asymmetric in the sense that u may not be the corner nearest v in the reverse cone from v. We will use the cone graph as an undirected graph, but remember the vertex u a cone edge is generated from.

To solve the shortest path problem approximately, Clarkson adds a and b as single point obstacle corners, and construct the above cone graph. The graph is constructed in $O(kn \log n)$ time. It has $n + 2$ nodes and at most $k(n + 2)$ edges. The shortest graph path from a to b is computed in $O(kn + n \log n)$ time. We shall need the following from Clarkson's work:

Lemma 1 (Clarkson[1]). *Given the plane with polygonal obstacles with a total of n corners, we can construct a cone Voronoi diagram in $O(nk \log n)$ time and $O(nk)$ space. Subsequently, for any cone i from a point a, we find a closest visible corner, if any, in $O(\log n / \log \log n)$ time. In particular, we can build the cone graph in $O(nk \log n)$ time. The vertices of the cone graph are the n obstacle corners connected by $O(nk)$ edges, and it represents obstacle avoiding distances between the corners with stretch $1 + O(1/k)$.*

Chen [8] has shown how to modify Clarkson's approach for the off-line shortest path problem into a preprocessing for two point queries. The preprocessing does not know the query points, but computes all-pairs shortest paths in Clarkson's

[1] We base our time bounds on the recent point location of Chan and Patrascu [18,19] which takes $O(\log n / \log \log n)$ time as opposed to the classic $O(\log n)$ time.

cone graph over all the original obstacle corners. This takes $\tilde{O}(kn^2)$ time. The resulting distance matrix is used as a stretch $1 + O(1/k)$ oracle for obstacle avoiding distances between corners.

Chen uses the conical Voronoi diagrams to construct cone edges from query points a and b to the obstacle corners. This takes $O(k \log n)$ time. For each cone edge (a, u) from a and (v, b) to b, he uses the corner oracle to estimate the distances from u to v, and add this to the Euclidean distances from a to u and from v to b. The smallest such value is a stretch $1 + O(1/k)$ estimate of the shortest path from a to b that touches some obstacle corner on the way. Computing all these combinations takes $O(k^2)$ time.

We have now simulated Clarkson's off-line algorithm except for the case with a direct cone edge between a to b, in which case the answer to the distance query should be the Euclidean distance from a to b. Chen augments the cone Voronoi diagram from the corners with extra information so as to check for this special case in $O(k \log n)$ time. Thanks to this fix of Chen, we can forget about this special case. However, we shall need some details from Chen's solution in our own construction, so we state here the relevant lemmas. First we have

Lemma 2 (Chen). *Consider a set of polygonal obstacles plus two arbitrary points a and b. Consider the cone i from a that contains b, that is, the direction to b is between $360°(i - 1/2)/k$ and $360°(i + 1/2)/k$. Suppose b is as close to a as any visible obstacle corner in cone i. If b is not visible from a then the first obstacle hit by the line from a to b is an obstacle segment (u, v) that is also the first obstacle hit by one of the cone boundary lines from a in directions $360°(i - 1/2)/k$ and $360°(i + 1/2)/k$.*

We already have Clarkson's cone Voronoi diagram from Lemma 1 which tells us the closest visible corner u in cone i from a, and if u is closer in the central cone direction than b, then we know that a cone edge from a to b is not needed. Hence we can assume that b is as close as any visible corner in cone i. Now to check if a cone edge from a to b is appropriate, using Lemma 2 we consider each directions $360°(i - 1/2)/k$ and $360°(i + 1/2)/k$, and find the first obstacle segment (u, v) hit, if any, and check if it blocks b from the view of a. If no such blocking is found, we know that a sees b, and then we return $\|a - b\|_2$ as the exact distance from a to b. To find the first obstacle hit in a cone boundary direction $360°(i + 1/2)/k$, Chen uses the preprocessing of the following lemma.

Lemma 3 (Chen). *For a given direction, we can preprocess the plane with polygonal obstacles with a total of n corners in $O(n \log n)$ time and $O(n)$ space so that for any query point a, we can identify the first obstacle point hit in the given direction in $O(\log n / \log \log n)$ time. Applying this preprocessing for each of the k cone boundary directions $360°(i + 1/2)/k$, the preprocessing takes $O(nk \log n)$ time and $O(kn)$ space while the query time along any one of these directions remain $O(\log n / \log \log n)$.*

Using Lemma 2 and Lemma 3, we can check if a cone edge from a to b is appropriate in $O(\log n / \log \log n)$ time. We note that Chen's construction can be used in conjunction with any distance oracle over the obstacle corners.

Lemma 4 (Chen). *Suppose we have an oracle that estimates the obstacle avoiding distance between obstacle corners with stretch s in time t. With an additional preprocessing $O(nk \log n)$ time and $O(kn)$ space, we can estimate the distance between any pair of points in $O(k^2 t + k \log n / \log\log n)$ time and stretch $\max\{s, 1 + 1/k\}$.*

The corner oracle based on an all pairs shortest path computation over Clarkson's cone graph has $s = 1 + O(1/k)$ and $t = O(1)$, which is very good, but the preprocessing time and space is near-quadratic. Chen presents a more economic oracle for corner distances. He constructs it in $\tilde{O}(n^{3/2})$ and it uses only near-linear space. However, the corner distance stretch is 6, which then becomes the stretch of his distance estimates for arbitrary two point queries. This paper is also about a more efficient distance oracle for corners, referring to Lemma 4 for distances between arbitrary points.

3 An Approximate Distance Oracle for Obstacle Corners

We will now show how to construct an oracle providing stretch $1 + O(1/k)$ distances between the corners of the obstacles. This is a modification of the technique of Klein and Thorup [15,16] to construct approximate distance oracles for the vertices of an undirected planar graph. In this section we show the existence of such an oracle, and pay only little attention to the construction time. This part is very similar to the planar graph construction by Thorup [16], yet our presentation is focused on the geometric case. The efficient construction in the next section is more technically interesting.

Distance notation. As a general notation, we shall use $\delta(a, X_1, ..X_j, b)$ to denote the distance from a to b passing $X_1, ..., X_j$. Here each X_i represents a portion of the plane, e.g., X_i could be a single point, a set of discrete points, or a curve. We want the shortest length of a path from a to b that passes points $p_1, ..., p_j$ in this order with $p_i \in X_i$. For example, if V is the set of obstacle corners, then $\delta(a, V, b)$ denotes the shortest distance via some corner.

Distances in a domain X are denoted δ_X. If the domain is understood, the subscript X may be omitted. Currently, δ denotes distances in the plane without obstacles. Now take Clarkson's cone edge graph G from Lemma 1. The fact that it represents distances between corner obstacles u and v with stretch $1 + O(1/k)$ can be stated as $\delta(u, v) \le \delta_{G_k}(u, v) \le (1 + O(1/k))\, \delta(u, v)$.

Separator paths. As a first step we pick an arbitrary corner from which we construct an exact shortest path tree T spanning all the obstacle corners. Hershberger and Suri [1] have shown that this can be done in $O(n \log n)$ time and space. Next, starting from the polygonal lines of the obstacles and the tree T, we triangulate the plane, both inside and outside the obstacles. We assume that our plane has a polygonal frame so the triangulation can be done with straight line segments inside the frame. The exterior region outside the frame may not be used by any shortest path and is triangulated with curved edges.

The triangulated plane is now a planar graph with a spanning tree T. As described in [16], in $O(n)$ time, we can find a triangle Δ so that if we look at the region of the fundamental cycle induced by any side (u, v) of Δ, then this cycle contains less than half the triangles. Here, the fundamental cycle consists of (u, v) together with the unique path in T between u and v. As a degenerate case, we note that if $(v, w) \in T$, the inside is empty.

We now have a separator S of the plane which is the union of at most 6 shortest paths; namely the at most three paths in T from the triangle corners plus any triangle side which is not inside an obstacle or in the exterior region. We refer to these paths as *separator paths*. Working with planar graphs, Thorup [16] did not need to include the triangle sides in his separators.

Connections via a shortest path. We will now show how to represent approximate distances via a given shortest path Q. We are going to do this for each of the at most 6 shortest paths in the above separator. For corners u and v, we want to approximate $\delta(u, Q, v)$. So far, the representation is only existential.

For each vertex u, we want a set of *connections* $C(u, Q)$ which are auxiliary weighted edges from u to points q_i in Q. The weight of (u, q_i) is $\delta(u, q_i)$. These connections have stretch s if for every point q in Q, the distance from u in $C(u, Q) \cup Q$ is at most s times longer than the original distance. Recall that Q is a shortest path. Hence, if we have a connection $(u, q_i) \in C(u, Q)$ and q is any point in Q, then $\delta_{Q \cup C(u,Q)}(u, q) \leq \delta(u, q_i, q)$.

Lemma 5. *There is a set $C(u, Q)$ of $4k$ connections from u to Q with stretch $1 + 1/k$.*

Proof. Let q_0 be the vertex in Q nearest u, that is, $\delta(u, q_0) = \delta(u, Q)$. We will now move from q_0 towards one end t of Q, identifying connection points q_i. Having identified q_i, $i > 0$, we pick q_i as the first subsequent point in Q such that $(1 + 1/k)\delta(u, q_i) < \delta(u, q_{i-1}, q_i)$.

We claim that this process results in at most $2k$ connections. The point is that $\delta(u, q_i) < \delta(u, q_{i-1}, q_i) - \delta(u, q_i)/k \leq \delta(u, q_{i-1}, q_i) - \delta(u, q_0)/k$. Hence $\delta(u, q_i) < \delta(u, q_0, q_i) - i\,\delta(u, q_0)/k$, so $i\,\delta(u, q_0)/k < \delta(u, q_0) + (\delta(q_0, q_i) - \delta(u, q_i)) \leq 2\,\delta(u, q_0)$. Thus $i < 2k$. The same procedure is applied to the other end-point of Q, so we end up with at most $1 + 2(2k - 1) \leq 4k - 1$ connections. □

Representing connections for efficient queries. We will now show how to represent connections to Q so that we can quickly estimate distances via these connections. We chose one end of Q as the starting point, and let $Q[x]$ denote the point at distance x from this starting point. We call x the *offset* of $Q[x]$ in Q. For each connection $(u, q_i) \in C(u, Q)$, let x_i be the offset of q_i. We represent this connection by the pair (x_i, y_i) where $y_i = \delta(u, q_i)$. Now, for any point $q = Q[x]$ in Q, we have $\delta(u, q_i, q) = y_i + |x - x_i|$. The connections in $C(u, Q)$ are always sorted according to the offset.

Now, consider two corners u and v. Suppose we have a connection $(x, y) \in C(u, Q)$ from u to Q and a connection $(x', y') \in C(v, Q)$ from v to Q. Then $\delta(u, Q[x], Q[x'], v) = y + |x - x'| + y'$. This way we use connections from u and v to Q to get distances from u to v via Q.

To estimate $\delta(u, Q, v)$ from $C(u, Q)$ and $C(v, Q)$ we first merge $C(u, Q)$ and $C(v, Q)$. In the resulting sorted list we look for neighbors $(x, y) \in C(u, Q)$ and $(x', y') \in C(v, Q)$. This neighboring pair corresponds to the distance $y + |x - x'| + y' = \delta(u, Q[x], Q[x'], v)$, and we return the minimal such distance via neighboring connections from u and v. We denote this estimate $d(u, Q, v)$.

Lemma 6. *If $C(u, Q)$ and $C(v, Q)$ provide connections from u and v to Q with stretch s, then $d(u, Q, v)$ estimates $\delta(u, Q, v)$ with stretch s in $O(|C(u, Q)| + |C(v, Q)|)$ time.*

Proof. The query time is trivially linear. Now consider the shortest path P from u to v intersecting Q in some point $Q[z]$. Since $C(u, Q)$ has stretch s, it has a connection (x, y) such that $\delta(u, Q[x], Q[z]) = y + |x - z| \leq s\,\delta(v, Q[z])$. Similarly, we have a connection $(x', y') \in C(v, Q)$ such that $\delta(v, Q[x'], Q[z']) = y' + |x' - z| \leq s\,\delta(v, Q[z])$. Hence $y + |x - x'| + y' \leq y + |x - z| + |x' - z| + y' \leq s\,\delta(u, Q[z]) + s\,\delta(Q[z], v) = s\,\delta(u, Q, v)$. Now, it may that the above (x, y) and (x', y') are not neighbors. However, let (x, y) and (x', y') be picked as close as possible yet producing the minimal distance. Then they must be neighbors; for otherwise there is a connection (x^*, y^*) between them. By symmetry, we can assume $(x^*, y^*) \in C(v, Q)$. Then $\delta(u, Q[x], Q[x'], v) = \delta(u, Q[x], Q[x^*], Q[x'], v) \geq \delta(u, Q[x], Q[x^*], v)$. Hence we get a no worse distance using (x^*, y^*) instead of (x', y'), contradicting the choice of (x', y'). Thus we conclude that (x, y) and (x', y') are neighbors, providing the distance with the desired stretch. □

A recursive distance oracle. We are now ready to describe our distance oracle for the corner recursively. We have a separator S consisting of at most 6 shortest paths. Using Lemma 5, for each vertex v and separator path $Q \in S$, we get a set of $O(k)$ connections with stretch $1 + 1/k$. The total space is now $O(kn)$.

Having represented the distances via the separator, we cut the plane along the 6 separator paths, and recurse on each region as an independent subproblem. This cutting may split corners and edges into multiple copies. Including the parts of T along the boundary of a piece, each piece inherits a shortest path tree and a triangulation. The size of a subproblem is measured by the number of triangles. Each time we divide into subproblems, each subproblem has at most half as many triangles, so the recursion depth is $O(\log n)$. We continue recursing until the subproblems are individual triangles, and we store the recursion tree $\mathcal{R}$ as part of our representation.

A corner u is only a participant in a subproblem Φ if it belongs to the region of the subproblem and it is not part of a separator on a higher levels. Then u can only participate in one subproblem on each level. It is only from participating corners that we store connections to the at most 6 separator paths Q of Φ. Hence the total space for these connections is $O(nk \log n)$. For each corner u we store the lowest subproblem Ψ_u that u participates in. Then u is part of the separator of this subproblem.

Now, how do we answer distance queries between vertices u and v? From Ψ_u and Ψ_v, we go up the recursion tree finding the $O(\log n)$ common ancestor problems Φ that both u and v participate in. For each common subproblem Φ,

we compute the estimate $d_\Phi(u, Q, v)$ via each separator path Q in $O(k)$ time. The smallest estimate over all common subproblems is returned. The total query time is then $O(k \log n)$.

To see that the stretch is $1+1/k$, consider a shortest path P from u to v. As we perform the recursion, there will be a first subproblem Φ with a separator path Q intersecting P. Since P was not intersected by a separator path on a higher level, we know that u and v both participate in this subproblem and that P is in Φ. Hence we have $\delta(u, v) = |P| = \delta_\Phi(u, v) = \delta_\Phi(u, Q, v)$. and then our connections in Φ from u and v to Q gives us an estimate $d_\Phi(u, Q, v) \leq (1+1/k)\, \delta_\Phi(u, Q, v) = (1 + 1/k)\delta(u, v)$.

4 Efficient Construction

We will now describe an efficient construction of the connections to separator paths. Technically, this is the most novel part of the paper. First we construct the recursive separator hierarchy. So far we have not made any connections. The construction time for the recursive hierarchy $\mathcal{R}$ is $O(n \log n)$.

We shall use Clarkson's cone edge graph G from Lemma 1 which represent distances between obstacle corners with stretch $1 + O(1/k)$. To construct the graph, for each corner u and each of k cones C_u from u, we added a straight line cone edge from u to the nearest corner v in C_u, if any. The result is not a planar graph as the straight line edges may cross. However, the straight lines do not cross any obstacle or the external boundary. Hence each path P in G is embedded into an obstacle avoiding path in the plane.

Our overall goal is to provide corner distance estimates $d(u, v)$ so that

$$\delta(u, v) \leq d(u, v) \leq (1 + 1/k)\delta_G(u, v). \tag{1}$$

Note that $d(u, v)$ is only lower bounded by the real distance in the plane with obstacle. Since $\delta_G(u, v) \leq (1 + O(1/k))\delta(u, v)$, we have that (1) implies that d has stretch $1 + O(1/k)$ compared with the real distances.

We are going to construct connections to a separator path $Q \in S$ which is a shortest obstacle avoiding path in the plane. The precise goal is that if there is a u-v path P in G whose embedding touch Q, the connections via Q should provide a distance estimate bounded by $(1 + 1/k)|P|$.

Our first step is to construct a new graph G^Q from G so as to encode the relationship between the graph G and the plane separator path Q. First we add all segments of Q as graph edges. Next we consider all intersections between cone edges (u, v) and segments (f, g) of the separator path Q. If they intersect in a point p, we make p a new auxiliary vertex, and replace (u, v) with (u, p) and (p, v) and (f, g) by (f, p) and (p, g).

Lemma 7. *For any corners u and v we have $\delta(u, v) \leq \delta_{G^Q}(u, v) \leq \delta_G(u, v)$. Moreover, if the embedding of a path P in G^Q intersects Q, then P intersects Q in G^Q.*

Proof. Introducing the new intersections and edges can only reduce distances in the graph, and since graph paths in G^Q embed into obstacle avoiding paths, they can never be too short. Finally, the graph G^Q has no edges whose embeddings cross Q. □

We are now left with a pure graph problem; namely to construct connections from each vertex in G^Q to Q with stretch $(1+1/k)$ compared with the distances in G^Q. For the efficiency of the construction, we need to show that G^Q is not too large.

Lemma 8. *A cone edge e can only intersect a separator path Q in a single point unless the cone edge is itself an edge in Q.*

Proof. First, as a matter of definition, if e coincide with a segment of Q, then e is an edge in Q because all corners touched by Q are considered vertices of Q.

Since Q is a shortest path and e a straight line, both avoiding obstacles, we cannot have Q leaving e and coming back to e. Hence, if Q and e were to intersect in more than one point, it would be in a straight line segment $\overline{pq}$. However, if Q arrives e not in the end of e, and turn to follow e, then p is a corner because the shortest path Q only bends at corners. However, then this point is closer to the start of the cone, contradicting the definition of cone edges. □

Lemma 9. *The graph G^Q has at most $(k+1)n$ vertices and at most $3kn+n-1 = O(kn)$ edges. At most n vertices of G^Q are not in Q.*

Proof. The starting point is at most n vertices, at most $n-1$ edges in Q, and at most kn cone edges. These numbers can grow when one of the kn cone edges intersect Q in a single point, as stated in Lemma 8. The subdivisions may lead to 2 extra edges and one extra vertex in Q. □

To get construct the connections, we use the lemma below which is implicit in the proof of [16, Lemma 3.18].

Lemma 10 (Thorup). *Given an arbitrary weighted undirected graph G^Q with a shortest path Q, for each vertex u, we can construct a set $C(u,Q)$ of connections to Q with stretch $1+1/k$ and size $O(1/k)$. The total construction is done via single source shortest path computations in subgraphs $H^{Q'}$ of G^Q except that the shortest path Q' may bypass some vertices in Q. The source is located in Q'. Each vertex appears in $O(k \log n)$ of these subgraphs $H^{Q'}$.*

Using linear time undirected single source shortest paths [4,5], we get

Lemma 11. *For the particular G^Q in our construction, for each vertex u, we can construct a set $C(u,Q)$ of connections to Q with stretch $1+1/k$ and size $O(1/k)$ in $O(nk^2 \log n)$ total time.*

Constructing G^Q. We will now show how to construct G^Q. First we apply the preprocessing of Lemma 3 to the separator path Q alone so that we from any point a in any obstacle boundary direction $360°(i+1/2)/k$ can find the first segment or vertex of Q hit in this direction in $O(\log n)$ time. This preprocessing takes $O(k|Q| \log n)$ time and $O(k|Q|)$ space. For each cone edge (u,v) in G we will now check if and where it intersects Q.

Lemma 12. *If the cone edge (u, v) in cone i is crossed by Q, then it is by a segment (f, g) which is the first part of Q hit from u in one of the two cone boundary directions $360°(i - 1/2)/k$ and $360°(i + 1/2)/k$.*

Proof. Assume that (u, v) is crossed by Q in cone i from u. We want to prove that the intersection is in one of the segments $(f, g) \in Q$ considered. We will apply Lemma 2 with (u, v) as (a, b). By definition of (u, v), we know that in the original plane with all its obstacles O, the vertex v is as close to u as any visible obstacle corner in cone i from u. If we add Q as an obstacle path, this can only decrease visibility, so v is still as close to u as any visible obstacle corner from $O \cup Q$ in cone i. This uses that all corners of Q were original obstacle corners.

From Lemma 2 we now know that if v is blocked from u by $O \cup Q$, then this is by a segment (f', g') that is also the first segment from $O \cup Q$ hit by one of the cone boundary lines from u in directions $360°(i - 1/2)/k$ and $360°(i + 1/2)/k$. However, if such an (f', g') blocks v from u, then (f', g') must be from Q since O did not block v from u. Then (f', g') must be the first segment from Q hit by one of the cone boundary lines from u in directions $360°(i - 1/2)/k$ and $360°(i + 1/2)/k$. Thus (f', g') is one of the segments of Q considered. □

The test in Lemma 12 takes $O(\log n)$ time given the preprocessing from Lemma 2, and from Lemma 8, we know that we only have to find a single intersection point in (u, v). We therefore conclude:

Lemma 13. *We can construct G^Q from G and Q in time $O(kn \log n)$.*

Dealing with all separator paths and recursing. We are now done with all connections to Q, and hence we can delete all edges incident to Q from G^Q including Q itself. More formally, from (1) and Lemma 7 we know that it suffices to estimate distances in G^Q as they are no worse than those in G. Moreover, for any vertices in u and v, $\delta_{G^Q}(u, v) \leq \delta_{G^Q}(u, Q, v) + \delta_{G^Q \setminus V(Q)}(u, v)$. The connections to Q give us an estimate $d(u, Q, v)$ of $\delta_{G^Q}(u, Q, v)$ with stretch $1+1/k$, so now it only remains to estimate distances in $G^Q \setminus V(Q)$. Let G' denote $G^Q \setminus V(Q)$. It is important to note that G' is a subgraph of G. Recursively, this will mean that the edges in G' are the original cone edges whose embedding do not touch any previous separator path.

Next we make connections to the next separator Q', remove it, and continue this way with the up to 6 separator paths in S. When done, we have a graph G^* which is the subgraph of G where we have removed all parts of G whose embedding intersects S. The separator S splits the plane into regions, each with at most half the triangles, and each component of G^* is embedded into one of these regions. We can find out which region in $O(\log n)$ time using the recursion tree $\mathcal{R}$. As in Section 3, we recurse on the regions of the plane obtained by cutting along the separator paths. It might have seemed more natural to recurse separately on each component of G^*, but we measure complexity in terms of the triangles in a region, and if we recursed separately on different components in the same region, the we would have multiple subproblems on the same level sharing the same triangles.

The recursion tree and triangulation was all done in $O(n \log n)$ time. We have $O(n)$ triangles to start with and each triangle occurs in only one subproblem on each level, so the total construction time for a level is $O(kn \log n)$ for constructing the graphs G^Q with Lemma 13 and $O(nk^2 \log n)$ for constructing the connections to separators Q using Lemma 11. Summing over all levels, we get

Lemma 14. *The above construction is done in $O(nk^2(\log n)^2)$ time and $O(kn \log n)$ space.*

Given two corners u and v, the query algorithm is the same as in Section 3, considering all the $O(\log n)$ separator paths connected to u and v, returning the best estimate $d(u, Q, v)$ from Lemma 11.

Lemma 15. *The above query algorithm provides stretch $1 + O(1/k)$ estimates of the original obstacle avoiding distances between corners in $O(k \log n)$ time.*

Proof. The total query time is $O(\log n)$ multiplied with the query time from Lemma 11. For the stretch, consider a shortest u-v path P in the original cone graph G. Then $|P| \leq (1+O(1/k))\delta(u, v)$, The path P remain intact in subgraphs H until the first time we process a separator path Q intersecting the embedding of P. Then the graph H^Q has a path P' with the same embedding as P and which intersects Q, and then the connections from u and v to Q provide an stretch $1 + 1/k$ estimate $d(u, Q, v)$ of $\delta_{H^Q}(u, Q, v) \leq |P|$. Hence $d(u, v) \leq d(u, Q, v) \leq (1 + 1/k)(1 + O(1/k))\delta(u, v) = (1 + O(1/k))\delta(u, v)$. Since any estimate we make corresponds to the length of an obstacle avoiding path in the plane, we know that $d(u, v) \geq \delta(u, v)$. □

Thus we have proved

Proposition 1. *Given a plane with polygonal obstacles, in $O(nk^2(\log n)^2)$ time we can construct an $O(kn(\log n))$ space distance oracle that can provide stretch $(1 + O(1/k))$ distances between obstacle corners in $O(k \log n)$ time.*

We can improve the query time of the above corner oracle to $O(k)$ using the same modifications as in [16] for planar graphs. In [16], the construction time is increased by a factor $k \log n$, but in our case, it only increases by a factor k. The reason is that the modified construction internally use $O(nk \log n)$ connections as auxiliary edges. This is a factor $k \log n$ more edges than in the planar graphs considered in [16], but only a factor $\log n$ more than the $O(kn)$ edges in the cone graphs considered here.

Proposition 2. *Given a plane with polygonal obstacles, in $O(nk^2(\log n)^3)$ time we can construct an $O(kn(\log n))$ space distance oracle that can provide stretch $(1 + O(1/k))$ distances between obstacle corners in $O(k)$ time.*

Plugging this into Chen's construction from Lemma 4, we get answers to arbitrary two point queries in $O(k^3 + k(\log n)/(\log \log n))$ time and with stretch $1+O(1/k)$. To get a desired stretch of $1+\varepsilon$, we choose a large enough $k = O(1/\varepsilon)$. As in [16], paying an extra factor $\log n$ in space,we can also produce a short path in constant time per hop, and even perform routing. This settles Theorem 1.

References

1. Hershberger, J., Suri, S.: An optimal algorithm for Euclidean shortest paths in the plane. SIAM Journal on Computing 28(6), 2215–2256 (1999)
2. Mitchell, J.: Geometric shortest paths and network optimization. In: Handbook of Computational Geometry, pp. 633–701. North Holland, Amsterdam (2000)
3. Kernighan, B.W., Ritchie, D.M.: The C Programming Language, 2nd edn. Prentice-Hall, Englewood Cliffs (1988)
4. Thorup, M.: Undirected single-source shortest paths with positive integer weights in linear time. J. ACM 46, 362–394 (1999)
5. Thorup, M.: Floats, integers, and single source shortest paths. J. Algorithms 35, 189–201 (2000)
6. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comp. 13(2), 338–355 (1984)
7. Clarkson, K.: Approximation algorithms for shortest path motion planning. In: Proc. 19th STOC, pp. 56–65 (1987)
8. Chen, D.Z.: On the all-pairs Euclidian shortest path problem. In: Proc. 6th SODA, pp. 292–301 (1995)
9. Arikati, S., Chen, D.Z., Chew, L.P., Das, G., Smid, M., Zaroliagis, C.D.: Planar spanners and approximate shortest path queries among obstacles in the plane. In: Díaz, J. (ed.) ESA 1996. LNCS, vol. 1136, pp. 514–528. Springer, Heidelberg (1996)
10. Chiang, Y.J., Mitchell, J.S.B.: Two-point euclidean shortest path queries in the plane. In: Proc. 10th SODA, pp. 215–224 (1999)
11. Gudmundson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles for geometric graphs. In: Proc. 13th SODA, pp. 828–837 (2002)
12. Gudmundson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles revisited. In: Bose, P., Morin, P. (eds.) ISAAC 2002. LNCS, vol. 2518, pp. 357–368. Springer, Heidelberg (2002)
13. Varadajan, K.R.: Personal Communication (2006)
14. Zeh, N.: I/O-Efficient Algorithms for Shortest Path Related Problems. PhD thesis, Carleton University (2002)
15. Klein, P.: Preprocessing an undirected planar network to enable fast approximate distance queries. In: Proc. 13th SODA, pp. 820–827 (2002)
16. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM 51(6), 993–1024 (2004) (Announced at FOCS'01)
17. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. 36, 177–189 (1979)
18. Chan, T.M.: Point location in $o(\log n)$ time, Voronoi diagrams in $o(n \log n)$ time, and other transdichotomous results in computational geometry. In: Proc. 47th FOCS, pp. 325–332 (2006)
19. Patrascu, M.: Planar point location in sublogarithmic time. In: Proc. 47th FOCS, pp. 325–332 (2006)