CCOSKEG Discs in Simple Polygons*

Prosenjit Bose† Anthony D'Angelo‡ Stephane Durocher§

Abstract

We consider the problem of finding a geodesic disc D of smallest radius containing at least k points among n inside a simple polygon P. The centre of D must lie on a chord in P. The polygon P has m vertices, r of which are reflex. We present an exact algorithm using parametric search that runs in $O(n \log^2 n + m)$ time with high probability and $O(n \log r + m)$ space.

1 Introduction

The *smallest / minimum enclosing disc* problem takes as input a set S of n points in the plane and returns the smallest Euclidean disc that contains S. This can be solved in $O(n)$ expected time [58] and $O(n)$ worst-case time [43]. The *smallest k-enclosing disc* problem is a generalization that asks for a smallest disc that contains at least $k \leq |S|$ points[1] of S, for any given k, and has been well studied [3, 24, 26, 27, 32, 39, 40]. It is conjectured that an exact algorithm that computes the smallest k-enclosing disc in the plane requires $\Omega(nk)$ time [31, §1.5].

Matoušek [39] presented an algorithm that first computes a constant-factor approximation[2] in $O(n \log n)$ time and $O(n)$ space (recently improved to $O(n)$ expected time for a 2-approximation that uses $O(n)$ expected space [32]), and then uses that approximation to seed an algorithm for solving the problem exactly in $O(n \log n + nk)$ expected time using $O(nk)$ space or $O(n \log n + nk \log k)$ expected time using $O(n)$ space (recently improved to $O(nk)$ expected time using $O(n+k^2)$ expected space [24, 32]). Matoušek [40] also presented an algorithm for computing the smallest disc that contains all but at most q of n points in $O(n \log n + q^3 n^\epsilon)$ time, where ϵ is "a positive constant that can be made arbitrarily small by adjusting the parameters of the algorithms; multiplicative constants in the $O()$ notation may depend on ϵ" [40].

In this paper we generalize the smallest k-enclosing disc problem to simple polygons using the *geodesic metric*, meaning that the distance $d_g(a, b)$ between two points a and b is the length of the shortest path $\Pi(a, b)$ between a and b that lies completely inside the simple polygon P. A *geodesic disc* $D(c, \rho)$ of radius ρ centred at $c \in P$ is the set of all points in P whose geodesic distance to c is at most ρ. Our article focuses on the *Chord-Constrained Smallest k-Enclosing Geodesic* (CCOSKEG) disc problem.

CCOSKEG Disc Problem

> Consider a simple polygon P_{in} defined by a sequence of m vertices in $\mathbb{R}^2$, $r > 0$ of which are reflex vertices, a set S of n points of $\mathbb{R}^2$ contained in P_{in},[3] an integer $k \leq n$, and an input chord $\ell \subset P_{in}$.[4] Find a *CCOSKEG disc*, i.e., a geodesic disc of minimum radius ρ^* in P_{in} *centred on* ℓ that contains at least k points of S.

Without loss of generality, we consider ℓ to be the x-axis. We make the general position assumptions that no two points of S are equidistant to a vertex of P_{in}, and no four points of S are geodesically co-circular. Under these assumptions, a smallest k-enclosing geodesic (SKEG) or CCOSKEG disc contains exactly k points. Let $D(c^*, \rho^*)$ be a CCOSKEG disc for the points of S in P_{in} constrained to the input chord ℓ. For convenience, at times we will refer to this as simply D^*. A k-enclosing geodesic disc (*KEG* disc) is a geodesic disc in P_{in} that contains exactly k points of S. The main result of our article is the following theorem.

Theorem 4 *Given a chord $\ell \subset P_{in}$ we compute a CCOSKEG disc $D(c^*, \rho^*)$ in $O(n \log^2 n + m)$ time with high probability[5] using $O(n \log r + m)$ space.*

1.1 Related Work

Other than our work on SKEG discs [15], we are not aware of other work tackling the subject of this paper. In our previous work [15], we presented an algorithm to compute a 2-approximation SKEG disc that

*This work is funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

†Carleton University, Ottawa, Canada, jit@scs.carleton.ca

‡anthony.dangelo@carleton.ca

§University of Manitoba, Winnipeg, Canada, stephane.durocher@umanitoba.ca

[1] In this paper, we use the notation $|Z|$ to denote the number of points in Z if Z is a point set, or the number of vertices of Z if Z is a face or a polygon.

[2] An α-approximation means that the disc returned has a radius at most α times the radius of an optimal solution.

[3] When we refer to a point p being in a polygon P, we mean that p is in the interior of P or on the boundary, ∂P.

[4] We use the terms *chord* and *diagonal* interchangeably.

[5] We say an event happens with high probability if the probability is at least $1 - n^{-\lambda}$ for some constant λ.

runs in expected time $O(n \log^2 n \log r + m)$ and expected space $O(n + m)$ if $k \in O(n/\log n)$; if $k \in \omega(n/\log n)$, it computes such a disc with high probability in $O(n \log^2 n \log r + m)$ deterministic time with $O(n+m)$ space. We compared it to the approach we presented in the same paper that uses higher-order geodesic Voronoi diagrams to find the exact solution. Assuming general position, a SKEG disc has either two or three points of S on its boundary, allowing techniques involving Voronoi diagrams to be applied. Ignoring polylogarithmic factors, the worst-case runtime for the Voronoi diagram approach for $k = n$ is $O(n + m)$; for $k = n - 1$ and $r/\log^2 r \in \Omega(k \log k)$ is $O(nr+m)$; for $k = n-1$ and $r/\log^2 r \in o(k \log k)$ is $O(n^2+nr+r^2+m)$; for $k < n-1$ and for $n \log n \in o(r/\log r)$ is $O(k^2 n + \min(rk, r(n - k)) + m)$; and $O(k^2 n + k^2 r + \min(kr, r(n - k)) + m)$ otherwise. Higher-order Voronoi diagrams have been considered to solve the smallest k-enclosing disc problem in the plane [3, 26].

There has been other work done with geodesic discs in polygons. A region Q is *geodesically convex* relative to a polygon P if for all points $u, v \in Q$, the geodesic shortest path from u to v in P is in Q. The *geodesic convex hull* CH_g of a set of points S in a polygon P is the intersection of all geodesically convex regions in P that contain S. The geodesic convex hull of n points in a simple m-gon can be computed in $O(n \log n+m)$ time using $O(n + m)$ space [29, 53].

The *geodesic centre* problem asks for a smallest geodesic disc that lies in the polygon and encloses all vertices of the polygon (stated another way, a point that minimizes the geodesic distance to the farthest point). This problem is well studied [4, 11, 16, 48, 53] and can be solved in $O(m)$ time and space [4]. The geodesic centre problem has been generalized to finding the geodesic centre of a set of points S inside a simple polygon in $O(n \log n + m)$ time [10]. Generalized versions of the geodesic centre for polygons [12, 46, 47, 55, 56]; packing and covering [49, 55]; and clustering [14] have all been studied.

Dynamic k-nearest neighbour queries were studied by de Berg and Staals [25]. They presented a static data structure for geodesic k-nearest neighbour queries for n sites in a simple m-gon that is built in $O(n(\log n \log^2 m + \log^3 m))$ expected time using $O(n \log n \log m+m)$ expected space and answers queries in $O(\log(n + m) \log m + k \log m)$ expected time.

If P_{in} has no reflex vertices, it is a convex polygon and the SKEG disc problem is solved by the algorithm for planar instances which uses a grid-refinement strategy. This works in the plane because $\mathbb{R}^2$ with the Euclidean metric is a doubling metric space, meaning that for any disc of radius $\rho > 0$ in $\mathbb{R}^2$ it can be covered by $O(1)$ discs of radius $\rho/2$ [33]. Geodesic discs do not have this property; it may take $\Theta(r)$ smaller discs to cover the larger one (refer to Fig. 1 in Appendix B). Another difficulty of the geodesic metric is that for two points u and v of S on opposite sides of a given chord, their geodesic bisector (formed by concatenating their bisector and hyperbolic arcs) can cross the chord $\Theta(r)$ times. See Figs. 2 to 4 in Appendix C.

Section 2 describes the preprocessing procedures and data structures used by our algorithms. Section 3 discusses how we use a technique known as *parametric search* to solve the CCOSKEG disc problem. Section 4 summarizes our result. Appendices A and B contain details omitted from the paper due to space constraints. Appendix C contains figures illustrating some concepts from the paper.

2 Preprocessing, Data Structures, and Definitions

We perform the following preprocessing in $O(m)$ time and space.

Polygon Simplification Convert P_{in} into a simplified polygon P consisting of $O(r)$ vertices using the $O(m)$ time and space algorithm of Aichholzer et al. [5] that computes a polygon P such that: $P \supseteq P_{in}$; $|P|$ is $O(r)$; the reflex vertices in P_{in} also appear in P; P preserves the visibility of points in P_{in}; and the shortest path between two points in P_{in} remains unchanged in P. As with P_{in}, we assume the points of S are in general position with the vertices of P, and no four points of S are geodesically co-circular in P.

Shortest-Path Data Structure We use the $O(r)$ time and space algorithm of Guibas and Hershberger [29, 34] on P to build a data structure that gives the length of the shortest path between any two query points in P in $O(\log r)$ time and space. Querying the data structure with two points in P returns a tree of $O(\log r)$ height whose in-order traversal is the shortest path in P between the two query points. The query also provides the length of the path from the source to each node along the path (which is stored at the respective node in the tree). This data structure can provide the first or last edge along the path between the two points in $O(\log r)$ time by traversing the tree to a leaf. We can also perform a search through this tree to find the midpoint of the shortest path in $O(\log r)$ time [57, Lemma 3]. The returned tree has $O(r)$ nodes and edges, but the query adds $O(\log r)$ nodes and edges linking to pre-computed structures to produce the result.

Funnel [29, 37] The vertices of the geodesic shortest path $\Pi(a, b)$ are the vertices a, b, and a subset of the vertices of the polygon P forming a polygonal chain [20, 38]. Consider a diagonal ℓ of P, its

two endpoints ℓ_1 and ℓ_2, and a point p in P. The union of the three paths $\Pi(p, \ell_1)$, $\Pi(p, \ell_2)$, and ℓ form what is called a funnel. This funnel represents the shortest paths from p to the points on ℓ in that their union is the funnel. Starting at p, the paths $\Pi(p, \ell_1)$ and $\Pi(p, \ell_2)$ may overlap during a subpath, but there is a unique vertex p_a called the *apex* (which is the farthest vertex on their common subpath from p) where the two paths diverge. After they diverge, the two paths never meet again. The path from p_a to an endpoint of ℓ forms an *inward-convex* polygonal chain (i.e., a convex path through vertices of P with the bend protruding into the interior of P). In our paper we often make use of the portion of the funnel between the apex and ℓ, which we shall refer to as a *truncated funnel*. Fig. 5 in Appendix C illustrates the notion of a funnel.

Definition 1 (Aronov 1989 **[9, Definition** 3.1**])** *For any two points u and v of P, the last vertex (or u if there is none) before v on $\Pi(u, v)$ is referred to as the* ***anchor*** *of v (with respect to u).*

Guibas and Hershberger [29] and Oh and Ahn [45] point out that given the trees representing the shortest paths between a fixed source and two distinct destination points on the same chord, the apex of their funnel can be computed in $O(\log r)$ time.

Observation 1 *The apex of a funnel from a source point in P to the diagonal ℓ can be computed in $O(\log r)$ time and $O(r)$ space. The distance from the source point to the apex can also be determined in $O(\log r)$ time and $O(r)$ space.*

Distance Function of a Point $u \in S$: Let us review the graph we get by plotting the distance from a point u to a line ℓ where the position along ℓ is parameterized by x. Abusing notation, we call the x-monotone curve representing this graph the *distance function*, which we denote by $\text{dist}_u(\cdot)$. The domain of this function is ℓ and it returns the geodesic distance from u to $x \in \ell$ where x is the input of the function. Without loss of generality, we can assume that the x-axis is the line in question. For a point u, u_x is the x-coordinate of u and u_y is its y-coordinate. This distance function is actually a branch of a *right hyperbola*[6] whose eccentricity is $\sqrt{2}$ and whose focus is therefore at $\sqrt{2} \cdot u_y$. In our polygon P, $\text{dist}_u(\cdot)$ is a continuous piecewise hyperbolic function. If the funnel from u to the endpoints ℓ_1 and ℓ_2 of ℓ is trivial (i.e., a Euclidean triangle), then $\text{dist}_u(\cdot)$ has one piece expressed as $\text{dist}_u(x) = \sqrt{(x - u_x)^2 + u_y^2}$. If there are reflex vertices of P in u's funnel, $\text{dist}_u(\cdot)$ has multiple pieces. The formula for each piece is $\text{dist}_u(x) = \sqrt{(x - w_x)^2 + w_y^2} + d_g(u, w)$, where w is the anchor, and the **domain** of this hyperbolic piece is the set of values of x for which w is the anchor. Refer to Fig. 6 in Appendix C for an example of a multi-piece distance function.

[6] Also called a *rectangular* or *equilateral* hyperbola.

Definition 2 (Aronov 1989 **[9, Definition** 3.7**])** *The* shortest-path tree *of P from a point s of P, $T(P, s)$, is the union of the geodesic shortest paths from s to vertices of P.*

Definition 3 (Aronov 1989 **[9, between** 3.8 **and** 3.9**])** *Let e be an edge of $T(P, s)$ and let its endpoint furthest from s be v. Let $\overrightarrow{h}$ be the open half-line collinear with e and extending from v in the direction of increasing distance from s. If some initial section of $\overrightarrow{h}$ is contained in the interior of P, we will refer to the maximal such initial section as the* **extension segment** *of e.*

Definition 4 (Aronov 1989 **[9, Definition** 3.9**])** *Let the collection of extension segments of edges of $T(P, s)$ be denoted by $E(P, s)$ (also simplified to E when the polygon and point are clear from the context).*

Consider the subset $E \subseteq E(P, u)$ whose elements define the domains of the pieces of $\text{dist}_u(\cdot)$ along ℓ. We refer to the intersection of an element of this set with ℓ as a *marker*. Sometimes we will need to identify domains that have specific properties so that an appropriate hyperbolic piece of $\text{dist}_u(\cdot)$ can be analyzed. Similar to other papers that find intervals of interest along shortest paths and chords [1, 2, 8, 45], we can use the funnel between u and ℓ to perform a binary search among the domain markers to find a domain of interest. Since domain markers are points along ℓ, one way they can be used is to provide distances away from u to compare against. We have the following observation.

Observation 2 *For an extension segment $e \in E$, if it takes $O(1)$ time and space to determine which side of $\ell \cap e$ contains a domain of interest along ℓ, then we can find a domain of interest along ℓ and its corresponding hyperbolic piece of $\text{dist}_u(\cdot)$ in $O(\log r)$ time and $O(r)$ space.*

3 CCOSKEG Disc: Parametric Search

Refer to Appendix A for missing details. Let $\partial D(u, \rho)$ denote the boundary of the geodesic disc centred at u with radius ρ. We use parametric search to find a SKEG disc centred on the chord ℓ.

Parametric search is a technique introduced by Megiddo [41, 42] for optimizing a numeric parameter through deduction using two algorithms in tandem. The

first is a sequential *decision* algorithm. Given a candidate for the optimal value, the decision algorithm determines how this candidate relates to the optimal value (i.e., it determines whether the candidate is less than, equal to, or greater than the optimal value). Testing a candidate using the decision algorithm is usually costly, which is why the problem and the candidates need to have the following *monotonicity* property: if the test reveals that the optimum is greater (less) than the candidate tested, then it is also greater (less) than everything less (greater) than the tested candidate.

The second algorithm used is a parallel *generic* algorithm. This parallel algorithm (which is usually converted back into a sequential algorithm) typically solves a problem using comparisons whose outcomes depend on the parameter being optimized, or, in other words, comparisons of objects that *would* result if the optimal value were given. In a way, we work backwards by examining which properties/objects would exist if we had the optimum as well as how these objects would relate to each other. For example, our algorithm to solve the CCOSKEG disc problem sorts, along ℓ, $\partial D(u, \rho^*) \cap \ell$ for all $u \in S$. Using sorting algorithms as the generic algorithm has been done before [21, 28, 42, 52, 54]. See Figs. 7 and 8 in Appendix C.

The comparisons in the generic sorting algorithm are typically expressed as a polynomial equation featuring the parameter to be optimized as a variable in the equation. Refer to Fig. 9 in Appendix C. The comparison is resolved by computing the sign of the equation (i.e., positive, negative, or zero) given a value for the parameter. Each of these polynomial equations has roots that together form the sortable set of candidates for the optimal value. Parametric search uses the decision algorithm to test the candidates. As more of the relations of the candidates to the optimum are determined, more comparisons in the generic algorithm can be resolved. In this way we are able to eventually deduce the optimal value.

Either ρ^* will coincide with the closest distance of ℓ to some point in S; or at least two of the intersection points from distinct discs will coincide and hence ρ^* will be a root for some pair of equations. See Fig. 10 in Appendix C. When comparing two of these intersections/equations, to get our candidate radii through which we search for ρ^* we set the equations equal to each other and solve for the roots (which is where they have coinciding intersection points along ℓ). For the pair of intersection points that created a given set of roots, the roots create intervals in the parameter space (see Fig. 11 in Appendix C). Given the equation for a pair from which we extracted the roots, plugging in any value for the radius that lies in one of these root-defined intervals results in the equation having the same sign (either positive or negative), and thus the intersection points having the same order along ℓ.

The sorting algorithm proceeds until it cannot continue without resolving any comparisons (i.e., until it gets stuck). Being a parallel algorithm (or a sequentialized version of a parallel algorithm), the comparisons in one parallel step are all independent and present us with a set of candidate radii. The decision algorithm is run on a judiciously-chosen candidate radius followed by a cull of the remaining candidates that we infer are too large or small. This is repeated until some comparison can be resolved, at which point the algorithm proceeds until it again becomes stuck. Eventually, the relation of ρ^* to all of the roots in the candidate set is known.

The (at most) two intersection points of a disc with ℓ tell us where the intersection of a disc with ℓ begins and ends. We are interested in overlapping intervals of at least k discs.

3.1 Preliminaries

3.1.1 Testing Closest Points

We precompute, for each point $u \in S$, the closest point of ℓ to u, also known as the projection of u onto ℓ (see Fig. 12 in Appendix C for an example of projections). Let u_c be the closest point of ℓ to u. Equivalently, u_c is the point along ℓ that minimizes $\text{dist}_u(\cdot)$. We showed the following in our previous work [15].

Lemma 1 (Bose et al. 2023 [15]) *We compute the set of projections of the points of S onto ℓ in $O(n \log r)$ time and $O(n + r)$ space.*

We are looking for ρ^*, the smallest radius of a KEG disc centred on ℓ, and the centre of such a disc. If a CCOSKEG disc has only one point $u \in S$ on its boundary, then a CCOSKEG disc is centred at the projection u_c and $\rho^* = d_g(u, u_c)$. Each of the n closest distances defined by projections is a candidate radius. To effectively perform a binary search among these candidates, we repeatedly perform an $O(n)$ time and space median selection algorithm on these radii [6, 13, 22]. In each iteration, we find the median, test it with the decision algorithm, then cull the remaining candidates now known to not be the optimum. Since we halve the number of elements to consider in each round, we perform $O(\log n)$ rounds and make $O(\log n)$ calls to the decision algorithm. The overall time spent over the $O(\log n)$ rounds performing median selections and culling the list is $O(n)$. After the search has finished, either we have found ρ^*, or we know that there will be at least two points of S on the boundary of a CCOSKEG disc.

Corollary 2 *With $O(\log n)$ calls to the decision algorithm and additional $O(n \log r)$ time and $O(n+r)$ space, we either compute ρ^* and a point c^* along ℓ such that $D(c^*, \rho^*)$ is a CCOSKEG disc of the points of S along*

ℓ, or we conclude that there are at least two points of S on the boundary of a CCOSKEG disc.

3.1.2 How to Compare Elements in the Sort

We will sort $\partial D(u, \rho^*) \cap \ell$ for all $u \in S$. We need to express these intersection points in terms of the variable radius ρ of the discs centred at the points of S. Assume that $\partial D(u, \rho)$ intersects ℓ twice (the cases of one and zero intersections are omitted). Assume that we know that the point $w = (w_x, w_y)$ is the last reflex vertex on the path from $u \in S$ to at least one of the intersection points. Let $\Delta = \rho - d_g(u, w)$. The equation for the circular arc defining $\partial D(u, \rho)$ where it intersects ℓ is given by the equation of a circle of radius Δ centred at w. Using the equation of a circle, we have the following.

$$x = \left(\pm\sqrt{\Delta^2 - w_y^2}\right) + w_x \tag{1}$$

If x is defined, it is only valid in the domain of w. If x is undefined, then after passing w, $D(u, \rho)$ does not intersect ℓ. We will assume we know the last reflex vertex before every intersection point and thus the $O(n)$ equations to use for the intersection points of the discs with ℓ. We show in Section 3.4 that we can use a parametric-search-like approach to find these $O(n)$ reflex vertices using an idea similar to one of the steps of the Goodrich and Pszona parametric search paper [28].

Now that we have our items to be sorted, we need to know how to compare them. Consider two of these intersection points, one for each of the points u and v, $\{u, v\} \in S$. Let the reflex vertex of the intersection point of u (resp. v) being considered be w (resp. z) and let the intersection points considered be the ones computed by taking the positive square roots in their equations (from Eq. (1)). Let $\delta = d_g(u, w)$ and $\psi = d_g(v, z)$. When we sort the intersection points, we are asking if one x-value is less than, greater than, or equal to another along ℓ. Therefore, we want to know the following at a variable radius ρ.

$$\sqrt{(\rho - \psi)^2 - z_y^2} + z_x \lessgtr \sqrt{(\rho - \delta)^2 - w_y^2} + w_x \tag{2}$$

We can expand and simplify Eq. (2) to get a cubic function replacing constant expressions by constant C_i.

$$0 \lessgtr C_1\rho^3 + C_2\rho^2 + C_3\rho + C_4 \tag{3}$$

The sign of the answer of Eq. (3) reveals which intersection point is to the left. Eq. (3) gives us a polynomial in ρ which determines the comparisons of the parallel sorting algorithm and whose roots are the candidates with which to run the decision algorithm. The roots are the values for which the two intersection points coincide. Once it is known to which side of each root the optimal ρ^* lies for this instance of Eq. (3), we know the result of the comparison for ρ^* for this instance.

3.2 Decision Problem

Lemma 3 *Given a polygon P with $O(r)$ vertices, a chord ℓ, a set S of n points, a radius ρ, and a constant $k \leq n$, having performed the preprocessing of Lemma 1, we can decide if there is a KEG disc of radius ρ centred on ℓ and return such a disc in $O(n(\log r + \log n))$ time and $O(n + r)$ space, and report whether $\rho < \rho^*$, $\rho > \rho^*$, or $\rho = \rho^*$.*

Proof. [Sketch] In $O(\log r)$ time and $O(r)$ space we can build the two funnels of u between u_c and the endpoints of ℓ and then perform a binary search in each to locate the domain in which a point at distance ρ lies. This tells us which reflex vertex to use in Eq. (1). Thus, in $O(n \log r)$ time and $O(n + r)$ space, we create $O(n)$ labelled intervals: $\{D(u, \rho) \cap \ell : u \in S\}$. We then sort the interval endpoints in $O(n \log n)$ time and $O(n)$ space, and then walk along ℓ and count the maximum number of discs we are concurrently in at any given point. If the maximum is smaller than k, then ρ is too small. If the maximum is larger than k, then ρ is too large. If the maximum is k and there is a subinterval that is larger than a single point in which there are k overlapping intervals, then ρ is too large. Otherwise, $\rho = \rho^*$ and the single point of k overlaps is the centre for a CCOSKEG disc. □

3.3 Using Boxsort

Goodrich and Pszona [28] show that boxsort [50] can be used as our sorting algorithm. It can be described as quicksort with multiple pivots which produces a number of recursive calls proportional to the number of pivots. See Fig. 13 in Appendix C for an illustrated example of a recursive call. This allows them to take advantage of the optimization technique of Cole [21] to reduce the running time.

Theorem 4 *Given a chord $\ell \subset P_{in}$ we compute a CCOSKEG disc $D(c^*, \rho^*)$ in $O(n \log^2 n + m)$ time with high probability using $O(n \log r + m)$ space.*

Proof. [Sketch] Preprocessing from Section 2 takes $O(m)$ time and space. It will be shown in Section 3.4 that with $O(\log n + \log r)$ calls to the decision algorithm and additional $O(n \log r)$ time and $O(n \log r + r)$ space, we compute the last reflex vertices on the paths from each point $u \in S$ to $\partial D(u, \rho^*) \cap \ell$, effectively giving us $O(n)$ items to sort. Given this result and Corollary 2, the preprocessing from Section 3.1 makes $O(\log n + \log r)$ calls to the decision algorithm

of Lemma 3, uses $O(n \log r + r)$ space, and takes time $O(n \log n \log r + n \log^2 n + n \log^2 r)$.

As seen in Goodrich and Pszona [28], Motwani and Raghavan [44], and Reischuk [50], with high probability (i.e., at least $1 - e^{-\log^b n}$ for some constant $b > 0$) boxsort chooses a "good" sequence of pivots so that it only requires $O(\log n)$ calls to the decision algorithm of Lemma 3; and with the same probability, taking into account the number of recursive calls and the time we spend in a recursive call to create boxes and then sort the remaining comparisons into their boxes, using boxsort for parametric search takes $O(n \log n + \log n \cdot n(\log r + \log n)))$ time and $O(n + r)$ space.

Considering the $O(m)$ time spent in preprocessing, we can simplify the runtime to $O(n \log^2 n + m)$ with high probability by assuming some terms are dominant and arriving at a contradiction. The overall space used is $O(n \log r + m)$. □

3.4 Decreasing to a Linear Number of Items to Sort

In this section, our goal is to discover which $O(n)$ reflex vertices to use for Eq. (1) for the points of S. The procedure (and its analysis) is like one of the steps used in the boxsort parametric search of Goodrich and Pszona [28]. Similar to routing unsorted elements through the binary tree of sorted pivots to find their "box" for the next recursive call, we independently route through $2n$ binary search trees of $O(\log r)$ height, where the outcomes of the comparisons depend on the solution to the parametric search. This allows us to find the reflex vertices for each $u \in S$ that anchor $\partial D(u, \rho^*) \cap \ell$. However, instead of inferring the optimum by sorting intersection points described as equations from which candidates for the optimum are extracted, here our comparisons are directly in the parameter space: we are directly comparing distances against the optimum. We illustrate an example in Fig. 14 in Appendix C.

Since domain markers for the funnel of a point in S with ℓ are points along ℓ, in addition to defining domains for reflex vertices they also provide distances to use as candidate radii. We have the following monotonicity property: for any point $u \in S$, the distance to ℓ increases monotonically as we move from its closest point $u_c \in \ell$ to the endpoints of ℓ [48]. Thus, if we can decide how the radius produced by a given marker compares to the optimal radius, we can perform two binary searches (recall Observation 2) among these markers between u_c and the endpoints of ℓ to find the domains which contain $\partial D(u, \rho^*) \cap \ell$, and hence discover which reflex vertices to use in Eq. (1) for u in the main parametric search.

We sequentialize the running of $2n$ parallel searches through binary trees of $O(\log r)$ height (one per funnel). Routing an element through these search trees is similar to following a directed path of $O(\log r)$ height. For each point $u \in S$ we search through the domain markers implicitly contained in its two funnels looking for the domains that contain $\partial D(u, \rho^*) \cap \ell$, which are the domains that contain a point that is distance ρ^* away from u.

For each tree through which we are routing, each step produces a comparison to resolve. Since a call to the decision algorithm is considered costly, we do not want to call the decision algorithm to resolve each comparison individually. Using the fact that the candidate radii have the monotonicity property we need for parametric search (i.e., given the relation between ρ^* and a candidate radius, we know the relation between ρ^* and either everything bigger than or less than the candidate) and the fact that the domain markers used in the comparisons of the searches also provide candidate radii, we can route through the trees with a logarithmic number of calls to the decision algorithm. Following Goodrich and Pszona [28], we assign weights to the comparisons in the searches. The routing can be considered as iterations involving three steps: in the first step, we use a linear-time weighted-median-finding algorithm [51] to choose the weighted median candidate radius; in the next step, we input that radius into the decision algorithm; when the decision algorithm returns, the last step is to repeatedly resolve all active comparisons that can be resolved until no more routing can be performed without knowing the result of another call to the decision algorithm. At this point, the next iteration begins.

Lemma 5 (Cole 1987 **[21], Goodrich and Pszona** 2013 **[28])** *For $j \geq 5(i + (1/2)\log(4n))$, during the $(j+1)^{st}$ iteration there are no active comparisons at depth i.*

Plugging our $2n$ routing trees of height $O(\log r)$ into the analyses of Goodrich and Pszona [28] and Cole [21] yields that there are $O(\log n + \log r)$ calls to the decision algorithm.

Corollary 6 *With $O(\log n + \log r)$ calls to the decision algorithm, with additional $O(n \log r)$ time and $O(r + n \log r)$ space, we compute for each $u \in S$ the anchors of $\partial D(u, \rho^*) \cap \ell$.*

4 Conclusion

By using the result of Goodrich and Pszona [28], we were able to use boxsort [50] to implement parametric search to solve the CCOSKEG disc problem. Though a 2-approximation to a SKEG disc contains $\Theta(\min(n, kr))$ points of S in general, we can use Theorem 4 together with an exact smallest k-enclosing algorithm for planar instances [32] to find a radius ρ at most twice the optimal radius of a SKEG disc such that any disc with radius ρ contains at most $4k$ points of S (see Appendix B).

References

[1] Pankaj K. Agarwal, Lars Arge, and Frank Staals. Improved dynamic geodesic nearest neighbor searching in a simple polygon. In *Symposium on Computational Geometry*, volume 99 of *LIPIcs*, pages 4:1–4:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[2] Pankaj K. Agarwal, Lars Arge, and Frank Staals. Improved dynamic geodesic nearest neighbor searching in a simple polygon. *CoRR*, abs/1803.05765, 2018.

[3] Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12(1):38–56, 1991.

[4] Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 56(4):836–859, 2016.

[5] Oswin Aichholzer, Thomas Hackl, Matias Korman, Alexander Pilz, and Birgit Vogtenhuber. Geodesic-preserving polygon simplification. *International Journal of Computational Geometry & Applications*, 24(4):307–324, 2014.

[6] Andrei Alexandrescu. Fast deterministic selection. In *SEA*, volume 75 of *LIPIcs*, pages 24:1–24:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[7] Nancy M. Amato, Michael T. Goodrich, and Edgar A. Ramos. Linear-time triangulation of a simple polygon made easier via randomization. In *Symposium on Computational Geometry*, pages 201–212. ACM, 2000.

[8] Lars Arge and Frank Staals. Dynamic geodesic nearest neighbor searching in a simple polygon. *CoRR*, abs/1707.02961, 2017.

[9] Boris Aronov. On the geodesic voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4(1):109–140, 1989.

[10] Boris Aronov, Steven Fortune, and Gordon T. Wilfong. The furthest-site geodesic voronoi diagram. *Discrete & Computational Geometry*, 9:217–255, 1993.

[11] Tetsuo Asano and Godfried Toussaint. Computing the geodesic center of a simple polygon. In *Discrete Algorithms and Complexity*, pages 65–79. Elsevier, 1987.

[12] Sang Won Bae, Matias Korman, and Yoshio Okamoto. Computing the geodesic centers of a polygonal domain. *Comput. Geom.*, 77:3–9, 2019.

[13] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

[14] Magdalene G. Borgelt, Marc J. van Kreveld, and Jun Luo. Geodesic disks and clustering in a simple polygon. *Int. J. Comput. Geometry Appl.*, 21(6):595–608, 2011.

[15] Prosenjit Bose, Anthony D'Angelo, and Stephane Durocher. Approximating the smallest k-enclosing geodesic disc in a simple polygon. In *WADS*, page (to appear). LNCS, 2023.

[16] Prosenjit Bose and Godfried T. Toussaint. Computing the constrained euclidean geodesic and link center of a simple polygon with application. In *Computer Graphics International*, pages 102–110. IEEE Computer Society, 1996.

[17] Timothy M. Chan. Geometric applications of a randomized optimization technique. *Discret. Comput. Geom.*, 22(4):547–567, 1999.

[18] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991.

[19] Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas J. Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[20] Orin Chein and Leon Steinberg. Routing past unions of disjoint linear barriers. *Networks*, 13(3):389–398, 1983.

[21] Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[23] Anthony D'Angelo. *Constrained Geometric Optimization Problems*. PhD thesis, Carleton University, 2023. doi:10.22215/etd/2023-15445.

[24] Amitava Datta, Hans-Peter Lenhof, Christian Schwarz, and Michiel H. M. Smid. Static and dynamic algorithms for k-point clustering problems. *J. Algorithms*, 19(3):474–503, 1995.

[25] Sarita de Berg and Frank Staals. Dynamic data structures for k-nearest neighbor queries. *Computational Geometry*, 111:101976, 2023.

[26] Alon Efrat, Micha Sharir, and Alon Ziv. Computing the smallest k-enclosing circle and related problems. *Comput. Geom.*, 4:119–136, 1994.

[27] David Eppstein and Jeff Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11:321–350, 1994.

[28] Michael T. Goodrich and Pawel Pszona. Cole's parametric search technique made practical. In *CCCG*. Carleton University, Ottawa, Canada, 2013.

[29] Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

[30] Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Endre Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[31] Sariel Har-Peled. *Geometric approximation algorithms*, volume 173. American Mathematical Soc., 2011.

[32] Sariel Har-Peled and Soham Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005.

[33] Juha Heinonen. *Lectures on analysis on metric spaces*. Springer, New York, 2001.

[34] John Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, 38(5):231–235, 1991.

[35] John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18(3):403–431, 1995.

[36] David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[37] Der-Tsai Lee and Franco P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[38] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

[39] Jiří Matoušek. On enclosing k points by a circle. *Inf. Process. Lett.*, 53(4):217–221, 1995.

[40] Jiří Matoušek. On geometric optimization with few violated constraints. *Discrete & Computational Geometry*, 14(4):365–384, 1995.

[41] Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979.

[42] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[43] Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12(4):759–776, 1983.

[44] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[45] Eunjin Oh and Hee-Kap Ahn. Voronoi diagrams for a moderate-sized point-set in a simple polygon. *Discrete & Computational Geometry*, 63(2):418–454, 2020.

[46] Eunjin Oh, Sang Won Bae, and Hee-Kap Ahn. Computing a geodesic two-center of points in a simple polygon. *Comput. Geom.*, 82:45–59, 2019.

[47] Eunjin Oh, Jean-Lou De Carufel, and Hee-Kap Ahn. The geodesic 2-center problem in a simple polygon. *Comput. Geom.*, 74:21–37, 2018.

[48] Richard Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4:611–626, 1989.

[49] George Rabanca and Ivo Vigan. Covering the boundary of a simple polygon with geodesic unit disks. *CoRR*, abs/1407.0614, 2014.

[50] Rüdiger Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM J. Comput.*, 14(2):396–409, 1985.

[51] Angelika Reiser. A linear selection algorithm for sets of elements with weights. *Inf. Process. Lett.*, 7(3):159–162, 1978.

[52] Sivan Toledo. *Extremal polygon containment problems and other issues in parametric searching*. PhD thesis, Citeseer, 1991.

[53] G Toussaint. Computing geodesic properties inside a simple polygon. *Revue D'Intelligence Artificielle*, 3(2):9–42, 1989.

[54] René van Oostrum and Remco C. Veltkamp. Parametric search made practical. *Comput. Geom.*, 28(2-3):75–88, 2004.

[55] Ivo Vigan. Packing and covering a polygon with geodesic disks. *CoRR*, abs/1311.6033, 2013.

[56] Haitao Wang. On the geodesic centers of polygonal domains. *JoCG*, 9(1):131–190, 2018.

[57] Haitao Wang. An optimal deterministic algorithm for geodesic farthest-point voronoi diagrams in simple polygons. In *SoCG*, volume 189 of *LIPIcs*, pages 59:1–59:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[58] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 1991.

A Parametric Search

Let $\partial D(u, \rho)$ denote the boundary of the geodesic disc centred at u with radius ρ. We assume preprocessing has already been performed. Note that the initial input chord ℓ of P_{in} may no longer be a chord in our simplified polygon P. We continue to use the initial chord since (a) shortest paths between points in P_{in} don't change when P_{in} is simplified to P; and (b) the endpoints of our given chord would define an interval of solution validity anyway if we chose to extend it into a chord for P (which could be done in $O(\log r)$ time and $O(1)$ space using ray-shooting queries).

Ray-Shooting Queries In $O(r)$ time and space we preprocess P to allow us to perform $O(\log r)$-time, $O(1)$-space ray-shooting queries that take as input a point in P and a direction and returns the point on ∂P (i.e., the boundary of P) where the ray first intersects ∂P [19, 35].

Remark 1 *It is not clear whether it is possible to apply the simpler recursive random sampling technique of Chan's that rivals parametric search to solve the CCOSKEG disc problem [17]. That approach requires one to partition the points of S into a constant number of fractional-sized subsets such that the overall solution is the best of the solutions of each of the subsets. It is not clear to us how to partition the points of S to take advantage of this approach.*

A.1 Testing Closest Points

Lemma 7 (Bose et al. 2023 [15]) *We compute the set of projections of the points of S onto ℓ in $O(n \log r)$ time and $O(n + r)$ space.*

Proof. Let ℓ be horizontal. For ease of presentation, we consider ℓ as having subdivided P into two polygons. We consider one of these polygons, let it keep the name P, and assume the points of S are in P. The other subpolygon can then be analyzed identically. Let the downward direction be toward the side of ℓ containing the exterior of the polygon P. Let the left endpoint of ℓ be ℓ_1 and the right endpoint be ℓ_2. Consider a point $p \in \ell$ and the last edge e of $\Pi(u, p)$ (i.e., the edge to which p is incident). Let the *angle of* e be the smaller of the two angles formed by e and ℓ at p. The range of this angle is $[0, \pi/2]$. We know from Pollack et al. [48, Corollary 2] that $\text{dist}_u(\cdot)$ is minimized when e is perpendicular to ℓ. We also know from Pollack et al. [48, Corollary 2] that given $p' \in \ell$ and an edge e' analogous to e, if the angle of e' is closer to $\pi/2$ than that of e, then $\text{dist}_u(p') < \text{dist}_u(p)$. Lastly, we know from Pollack et al. [48, Lemma 1] that $\text{dist}_u(\cdot)$ is a convex function which means it has a global minimum.

Using Observations 1 and 2 we can retrieve the truncated funnel of u and ℓ in $O(\log r)$ time and $O(r)$ space and use the convex chains to perform a binary search along ℓ using the markers defined by the elements of E to find the domain in which u_c lies. See Fig. 5 in Appendix C. This domain has the property that the angle of the last edge on the shortest path from u to the points in this domain is closest to $\pi/2$.

In the binary search, at each marker (as determined by the node currently being visited in the tree representing the convex chain), in $O(1)$ time and space we compute the angle of ℓ with the extension segment defining the marker. Since $\text{dist}_u(\cdot)$ is a convex function, we know that as we slide a point $p \in \ell$ from ℓ_1 to ℓ_2, the angle of the edge incident to p on $\Pi(u, p)$ will monotonically increase until it reaches $\pi/2$, then monotonically decrease. Thus, after computing the angle of the extension segment with ℓ, we know to which side of its marker to continue our search: the side that contains the smaller angle (because moving in this direction will increase the smaller angle). Thus by Observation 2 the search takes $O(\log r)$ time and $O(r)$ space.

At the end of our search we will have the reflex vertex whose domain contains the edge that achieves the angle closest to $\pi/2$. Then in $O(1)$ time and space we can build the corresponding piece of $\text{dist}_u(\cdot)$ and find the value along ℓ that minimizes it.

The space bounds follow from the n projections that are computed and the $O(r)$ space used by the shortest-path data structure queries. □

A.2 How to Compare Elements in the Sort

The trick when using a sorting algorithm as the generic algorithm in the parametric search technique is deciding what to sort. Once that has been determined, we use parametric search to run the sorting algorithm as if the things we are sorting were produced knowing ρ^*.

We will sort $\partial D(u, \rho^*) \cap \ell$ for all $u \in S$ (i.e., the intersection points of ℓ with the boundaries of the geodesic discs of radius ρ^* centred at the points of S). We need to express these intersection points in terms of the variable radius ρ of the discs centred at the points of S. Notice that the boundary of a geodesic disc of radius ρ is constructed piecewise. Part of the disc's boundary is formed by the boundary of the polygon at distance less than ρ away from the centre of the disc, and the rest is circular arcs from the circle centred at the disc's centre or from circles centred on reflex vertices contained in the disc's interior.

Let us assume for the moment that $\partial D(u, \rho)$ intersects ℓ twice (the cases of one and zero intersections are simple to figure out afterwards and are omitted). Assume that we know that the point $w = (w_x, w_y)$ is the last reflex vertex on the path from $u \in S$ to at least one of the intersection points. This intersection point is

where ℓ is intersected by a circular arc centred on w. Let $\Delta = \rho - d_g(u, w)$. If Δ were negative, we would have a contradiction ($D(u, \rho)$ would not even contain w). The equation for the circular arc defining $\partial D(u, \rho)$ where it intersects ℓ is given by the equation of a circle of radius Δ centred at w. Once again, assume ℓ is the x-axis. Using the equation of a circle, we have the following.

$$\begin{aligned}(x - w_x)^2 + (y - w_y)^2 &= \Delta^2 \\ (x - w_x)^2 + (0 - w_y)^2 &= \Delta^2 \\ (x - w_x)^2 + w_y^2 &= \Delta^2 \\ (x - w_x)^2 &= \Delta^2 - w_y^2 \\ x - w_x &= \pm\sqrt{\Delta^2 - w_y^2} \\ x &= \left(\pm\sqrt{\Delta^2 - w_y^2}\right) + w_x \qquad (4)\end{aligned}$$

If x is defined, it is only valid in the domain of w (i.e., the interval along ℓ in which w is the last reflex vertex on the path from u). If x is undefined, then after passing w, $D(u, \rho)$ does not intersect ℓ. If both values computed by Eq. (4) fall outside of w's domain, then we contradict that w is the last vertex on the path from u to the considered intersection point for the given radius ρ (which means that w would not be used in computing the boundary of $D(u, \rho)$). Otherwise, if an x-value from Eq. (4) lies within the domain of w, then this x-value would be one of at most two intersection points of $\partial D(u, \rho)$ and ℓ. If only one x-value computed by Eq. (4) falls in the domain of w, then the process must be repeated with some other reflex vertex (which is the case if the last reflex vertex from u is not the same for both intersection points).

Though we want to sort intersection points of ℓ with the boundaries of geodesic discs, our intersection points are equations until the variable ρ has been provided. Nonetheless, it is these intersection points we would like to sort. Ideally, we would have only $O(n)$ candidate intersection points along ℓ to consider (up to two per $u \in S$). As we saw above though, the intersection points of a geodesic disc centred on $u \in S$ depend on the last reflex vertex on the path from u to ℓ, which in turn depends on the optimal radius, which we do not know ahead of time. Initially, it seems that for each $u \in S$ we have to consider the $O(r)$ intersection points computed by using each reflex vertex of its truncated funnel. However, we do not want to spend $\Omega(nr)$ time. Luckily for us, as we show in Appendix A.5, we can use a parametric-search-like approach to whittle these $O(nr)$ candidates back down to $O(n)$ using an idea similar to one of the steps of the Goodrich and Pszona parametric search paper [28]. We will assume we know the last reflex vertex before every intersection point and thus the $O(n)$ equations to use for the intersection points of the discs with ℓ.

Now that we have our items to be sorted, we need to know how to compare them. Consider two of these intersection points, one for each of the points u and v, $\{u, v\} \in S$. Let the reflex vertex of the intersection point of u (resp. v) being considered be w (resp. z) and let the intersection points considered be the ones computed by taking the positive square roots in their equations (from Eq. (4)). Let $\delta = d_g(u, w)$ and $\psi = d_g(v, z)$. When we sort the intersection points, we are asking if one x-value is less than, greater than, or equal to another along ℓ. Therefore, we want to know the following at a variable radius ρ. Let C_i be constant i.

$$\begin{aligned}&\sqrt{(\rho - \psi)^2 - z_y^2} + z_x \lessgtr \sqrt{(\rho - \delta)^2 - w_y^2} + w_x \\ \Rightarrow \quad &0 \lessgtr (\rho - \delta)^4 + (\rho - \psi)^4 \\ &\quad - 2(\rho - \delta)^2(\rho - \psi)^2 \\ &\quad + C_1(\rho - \delta)^2 + C_2(\rho - \psi)^2 \\ &\quad + C_3 \qquad (5)\end{aligned}$$

We can expand and simplify Eq. (5) to get a cubic function, once again replacing constant expressions by constant C_i.

$$0 \lessgtr C_4\rho^3 + C_5\rho^2 + C_6\rho + C_7 \qquad (6)$$

We end up with the cubic Eq. (6). After testing the projections of S onto ℓ in Section 3.1.1/Appendix A.1, we know and discard the points of S too far from ℓ to intersect ℓ with a disc of radius ρ^*. Thus, Eq. (4) will be defined at radius ρ^* for each point being considered, and the abscissa will be in the domain of the associated reflex vertex. Thus, when the comparison of Eq. (6) is resolved, a value for the radius is used that: (a) produces the same sign as ρ^*; and (b) adheres to the restriction that the results of using that radius with the two instances of Eq. (4) that created the comparison lie in the respective domains (along ℓ) of the reflex vertices associated with the instances of Eq. (4). The sign of the answer reveals which intersection point is to the left. When the comparison is resolved in the parametric search, both intersection points are defined and valid.

Eq. (6) gives us the next piece of the parametric search puzzle: a low-degree polynomial in ρ which determines the comparisons of the parallel sorting algorithm and whose roots are the candidates with which to run the decision algorithm. The roots are the values for which the two intersection points coincide. As mentioned above, if ρ^* is not defined by the closest point of ℓ to some point in S, then at ρ^* there will be at least one pair of intersection points that coincide since the overlapping interval of the $\geq k$ discs along ℓ will collapse to a single point. The constant number of roots for an instance of Eq. (6), which can be computed in $O(1)$ time

and space since it is a cubic function, split the possible values of ρ for that instance into a constant number of intervals in the parameter space. Each interval has the property that evaluating the instance of Eq. (6) using any value of ρ in the interval produces the same sign. Therefore, once it is known to which side of each root the optimal ρ^* lies for this instance of Eq. (6), we know the result of the comparison for ρ^* for this instance.

A.3 Decision Problem

To use parametric search, we need a sequential decision algorithm that, given a radius as a candidate for ρ^*, can tell us if this candidate is less than, greater than, or equal to ρ^*.

Lemma 8 *Given a polygon P with $O(r)$ vertices, a chord ℓ, a set S of n points, a radius ρ, and a constant $k \leq n$, having performed the preprocessing of Lemma 1, we can decide if there is a KEG disc of radius ρ centred on ℓ and return such a disc in $O(n(\log r + \log n))$ time and $O(n + r)$ space, and report whether $\rho < \rho^*$, $\rho > \rho^*$, or $\rho = \rho^*$.*

Proof. Consider the geodesic disc of radius ρ, $D(u, \rho)$. Since the disc is geodesically convex, if the chord intersects the disc in only one point, it will be at the projection u_c. If it does not intersect the disc, then at u_c the distance from u to ℓ will be larger than ρ. Otherwise, if the chord intersects the disc in two points, u_c splits ℓ up into two intervals, each with one intersection point (i.e., each one contains a point of $\partial D(u, \rho) \cap \ell$). If u_c is an endpoint of ℓ, assuming ℓ has positive length, one of these intervals may degenerate into a point, making u_c coincide with one of the intersection points.

These two intervals to either side of u_c have the property that on one side of the intersection point contained within, the distance from u to ℓ is larger than ρ, and on the other side, the distance is less than ρ. Therefore, if $\partial D(u, \rho)$ does intersect ℓ in two points we can proceed as in the proof of Lemma 1: in $O(\log r)$ time and $O(r)$ space we can build the two funnels of u between u_c and the endpoints of ℓ (truncated at the apices) and then perform a binary search in each to locate the domain in which a point at distance ρ lies. We find the subinterval delimited by the domain markers of the reflex vertices wherein the distance from u to ℓ changes from being more (less) than ρ to being less (more) than ρ. The final subinterval for a given intersection point tells us which reflex vertex to use in Eq. (1). Once we find this domain, we can compute $\partial D(u, \rho) \cap \ell$ in $O(1)$ time and space. Thus, in $O(n \log r)$ time and $O(n + r)$ space, we create $O(n)$ labelled intervals along ℓ, one for each geodesic disc of radius ρ centred on each $u \in S$. In other words, the set of these intervals is $\{D(u, \rho) \cap \ell : u \in S\}$. We then sort these interval endpoints in $O(n \log n)$ time and $O(n)$ space, associating each endpoint with the interval it opens or closes.

When we walk along ℓ and enter the interval $D(u, \rho) \cap \ell$ for some $u \in S$, we say we are in the disc of u. Our next step, done in $O(n)$ time and space, is to walk along ℓ and count the maximum number of discs we are concurrently in at any given point. In other words, we are counting the maximum number of overlapping intervals. If the maximum is fewer than k, then ρ is too small. If the maximum is larger than k, then since we assume no four points are co-circular (and thus the CCOSKEG disc contains exactly k points), ρ is too large. If the maximum is k, if there is a subinterval that is larger than a single point in which there are k overlapping intervals, then ρ is too large. Otherwise, $\rho = \rho^*$ and the single point of k overlaps is the centre for the CCOSKEG disc. □

A.4 Using Boxsort

Goodrich and Pszona [28] use boxsort [50] as their sorting algorithm. It can be described as quicksort with multiple pivots which produces a number of recursive calls proportional to the number of pivots. This allows them to take advantage of the optimization technique of Cole [21] to reduce the running time. Although the pivots first need to be sorted and then the remaining elements need to be sorted into the correct boxes (i.e., placed between the correct pivots) before recurring, we have multiple boxes once the recursive calls start. Each box has an independent set of comparisons (i.e., the comparisons in a box are independent of other boxes). This allows the recursion in the different boxes to be at different levels. Rather than running a median-finding algorithm on the *value* of the candidate radii, however, a weighting scheme for the candidate radii is applied based on the depth of their defining comparisons in the recursion. The next radius to test with the decision algorithm is based on a linear-time weighted-median-finding algorithm [51]. The sum of the weights of the current candidates is called the *active* weight. The weighted-median-finding algorithm considers the couples of radius and weight and returns the set of elements whose sum of weights is at most half the active weight. Furthermore, all radii in this set are less than the radius in the computed weighted median, and adding the weight of the computed weighted median produces a weight larger than half the active weight. The algorithm can easily be modified to return the weighted median as well. As such, rather than each call removing half of the *number* of candidate radii and comparisons, each call removes at least a quarter[7] of the

[7] Although the weighted median resolves the comparisons of a weighted *half* of the candidates, the weighting scheme applied by Goodrich and Pszona [28] equally assigns half of the weight of a comparison to its children. Thus, half of the active weight is

active weight.

The candidate radii are not separated by recursive subproblem; the weighted-median that gets resolved is chosen from the complete set of untested radii that have not already been culled. Within a recursive subproblem of the Goodrich and Pszona approach [28], however, there are "synchronization points" in the algorithm represented as "virtual comparisons" that are not activated until the current batch of comparisons has been resolved. These virtual comparisons do not represent real work, but they assist in the analysis of the runtime. The analysis is done by creating a dependency graph between the comparisons in the algorithm where the height of the graph of one recursive subproblem (i.e., the longest path between the start of the recursive call and the point when the next recursive calls start) is $O(\log n)$, and then noting that this implies the height for the entire simulation is also $O(\log n)$ with high probability.

Recall that the items we are sorting are the $O(n)$ intersection points of the boundaries of the candidate geodesic discs with ℓ. Call these points *crossings*. We repeat the algorithm of Goodrich and Pszona [28] that uses the following weighting rule for the comparisons (virtual or not). The following algorithm description (which does not mention the virtual comparisons as they do not represent real work) assumes each comparison produces one root. See Fig. 13 in Appendix C for an illustrated example of a recursive call.

Weight Rule When comparison $\mathbb{C}$ of weight σ gets resolved and causes q comparisons $\mathbb{C}_1, \ldots, \mathbb{C}_q$ to become active, each of these comparisons gets weight $\sigma/(2q)$.

1. Randomly mark $\sqrt{n}$ crossings.
2. Sort the marked crossings by comparing every pair in $O(n)$ comparisons, each of weight σ. Order them with insertion sort.
3. After all of the comparisons of the previous step have been resolved, *activate* comparisons for routing the remaining crossings through the *tree of marked items* (i.e., we do a binary search through the marked items), where each comparison at the root of this tree has weight $\sigma/(2n^2)$. In other words, create comparisons and assign the appropriate weight to them to prepare the $n - \sqrt{n}$ unmarked crossings for a binary search through the marked crossings to place the unmarked crossings between the marked pivots.
4. Route the unmarked crossings through the tree (i.e., do a binary search for each of them with the marked items) by repeatedly finding and testing the weighted median and then resolving comparisons (following the weighting rule when comparisons get resolved).
5. Once we know in which box each unmarked element lies (i.e., between which marked items it lies), insert it into its appropriate box.
6. Assign weight $\sigma/(4n^{4.5})$ to the initial comparisons in the new subproblems.
7. Recur into subproblems simultaneously.

The Goodrich and Pszona analysis [28] omits a discussion about comparisons with multiple roots and how the weights change in such cases. Below we alter their analysis to use three roots.

After the marked crossings are sorted, we use the sorted crossings to perform a binary search to position each unsorted element between a pair of sorted marked crossings. In the Goodrich and Pszona analysis [28], this is presented as a binary search through a perfectly balanced binary search tree for each unmarked element independently. To keep the analysis simple, rather than routing $n - \sqrt{n}$ items, we route n items. Our comparisons have three roots, so the weight of the comparison at the root of this binary search tree (which is the same for each element being routed) must change accordingly.

We begin the analysis. To sort the marked crossings by brute force, each comparison between a pair of crossings actually produces three comparisons of roots against the optimal radius. Each of these three comparisons started with weight σ. Thus, after these crossings are sorted, following the Goodrich and Pszona analysis [28] using an upside-down virtual binary tree of $\log(3n)$ height in the dependence graph, the weight at the root of the virtual tree is $\sigma/(3n)$. This virtual root then activates (and equally shares half of its weight to) the comparison nodes that start routing the unmarked crossings through the binary search tree of marked crossings. Each comparison at the root of these binary search trees that route the unmarked crossings through the search tree of marked crossings, however, also creates three root comparisons. Thus, each of these root comparisons gets weight $\sigma/(2 \cdot (3n)(3n)) = \sigma/(2(3n)^2)$ (i.e., weight $\sigma/(3n)$ divided among $3n$ comparisons).

As the routing progresses through the binary search trees, the trees get whittled down to paths determining where an element lies in relation to the sorted crossings. When each comparison in the tree produces one root comparison, the weight at the bottom of the tree is the weight at the top divided by $2^{\log \sqrt{n}} = n^{0.5}$ because each comparison along the way passes half its weight to its one child, i.e., the next comparison on the path through the tree. However in our scenario, although resolving a routing comparison in the tree activates at

removed, but if each comparison involved had children, then we add back a quarter of the weight (i.e., half of half).

most one new routing comparison, it has three children, one for each root comparison of the next tree node. To aid in the analysis, we replace each routing comparison with the three root comparisons, all of which are the parents of a virtual comparison representing the routing comparison they resolve. Each of the three root comparisons of a routing node depend on (i.e., are children of) the virtual comparison of the node above it. In this way, rather than dividing the weight by half each step down the routing tree, we divide it by $2 \cdot (2 \cdot 3)$: each of the three root comparisons passes half of its weight to their (virtual) child (meaning it gets half the weight of any one of them), and this virtual node passes half of its weight equally shared amongst its three children, meaning each child gets half of a third of its weight. Thus, the weight at the bottom of our tree is the weight at the top divided by $(2 \cdot 2 \cdot 3)^{\log \sqrt{n}} = n \cdot 3^{\log \sqrt{n}}$. Although after the last routing comparison we do not create three new root comparisons, we create three virtual comparisons to make the analysis cleaner. Therefore, the weight at the bottom of the tree is $\sigma/(18n^2 \cdot n \cdot 3^{\log \sqrt{n}}) = \sigma/(18n^3 \cdot 3^{\log \sqrt{n}})$.

The next part of the dependence graph is another upside-down virtual binary tree like the one used after the sorting of the marked crossings. At the root of this tree, the weight becomes $\sigma/(18n^3 \cdot 3^{\log \sqrt{n}} \cdot 3n) = \sigma/(18n^4 \cdot 3^{\log \sqrt{n}+1})$. All initial comparisons in the subsequent recursive calls depend on the root of this tree and its weight. Thus the weight of the initial root comparisons in subsequent recursive calls is $\sigma/(2 \cdot (3n) \cdot (18n^4 \cdot 3^{\log \sqrt{n}+1})) = \sigma/(36n^5 \cdot 3^{\log \sqrt{n}+2})$ (i.e., weight $\sigma/(18n^4 \cdot 3^{\log \sqrt{n}+1})$ divided among $3n$ comparisons).

We can follow the approach of Goodrich and Pszona [28] and use Cole's analysis [21], which we repeat here modified for this specific case of at most three roots per comparison, to show that there are $O(\log n)$ calls to the decision algorithm.

Lemma 9 (Cole 1987 **[21])** *At the start of the $(j+1)^{st}$ iteration, the active weight is bounded above by $(3/4)^j \cdot (3n)$ for $j \geq 0$.*

Proof. We prove the result by induction on j. At the start of the first iteration there are $3n$ active comparisons at depth 0, and all other comparisons are inactive. So for $j = 0$, the result holds. To prove the inductive step, it is sufficient to show that in each iteration the active weight is reduced by at least one quarter. We now show this.

Consider an active comparison $\mathbb{C}$ of weight σ that has just been resolved. Then $\mathbb{C}$ ceases to be active, and up to three new comparisons may become active, each an equal share of half the weight of σ (e.g., if three new comparisons are activated, they each have weight $\sigma/(2 \cdot 3) = \sigma/6$). So the resolution of $\mathbb{C}$ reduces the active weight by at least $\sigma/2$. Let the active weight be $\mathbb{W}$. In one iteration, we are guaranteed that the comparisons resolved have combined weight at least $\mathbb{W}/2$. Thus, in one iteration, the active weight is reduced from $\mathbb{W}$ to at most $3\mathbb{W}/4$. □

Lemma 10 (Goodrich and Pszona 2013 **[28])** *Each comparison at depth i has weight $\geq (1/4)^i$.*

Proof. We prove this by induction on the depth of the boxsort recursion. Assume that the current recursive call operates on a subproblem of size $3n$, and that comparisons at the beginning of the recursive call have depth i and weight σ. By the inductive assumption, $\sigma \geq (1/4)^i$.

Consider comparisons in the current recursive call. Comparisons at depth j in the first tree of *virtual* comparisons (global depth $i + j$) have weight $\sigma/2^j \geq (1/4)^i \cdot (1/2)^j \geq (1/4)^{i+j}$. The last of them has (local) depth $\log(3n)$ and weight $\sigma/(3n)$. It then spreads half of its weight to $3n$ comparisons at depth $\log(3n) + 1$ (global depth $i + \log(3n) + 1$), setting their weight to

$$\sigma/(2(3n)^2) \geq \sigma/(4(3n)^2) = \sigma/(4^{\log(3n)+1}) \geq (1/4)^{i+\log(3n)+1}$$

The same reasoning follows for the case of the second *virtual* tree and recursive split.

Routing through the tree of sorted *marked* items has two levels of comparison nodes per node in the tree. For each step down this routing tree, we have three comparisons followed by a virtual comparison which then splits its weight among the three comparisons at the next level down in the routing tree. Let $\sigma' = \sigma/(2(3n)^2)$ be the weight of each comparison node at the root of the routing tree. The next node in the analysis tree (at global depth $i + \log(3n) + 2$) is the virtual node whose weight is

$$\sigma'/2 = \sigma/(4(3n)^2) = \sigma/(4^{\log(3n)+1}) \geq (1/4)^{i+\log(3n)+2}$$

The children of this node in the tree (at global depth $i + \log(3n) + 3$) each have weight

$$\begin{aligned}
\sigma'/(2 \cdot 2 \cdot 3) &= \sigma/(6 \cdot 4(3n)^2) \\
&= \sigma/(6 \cdot 4^{\log(3n)+1}) \\
&\geq \sigma/(4^2 \cdot 4^{\log(3n)+1}) \\
&= \sigma/(4^{\log(3n)+3}) \\
&\geq (1/4)^{i+\log(3n)+3}
\end{aligned}$$

We can map our analysis tree back on to the routing tree if we divide the weight by $2 \cdot 2 \cdot 3$ each level down the routing tree. This means the analysis tree has one more level beyond the last virtual comparison. This last level has three virtual comparisons per tree. This means that the number of levels in this routing tree is $2\log(\sqrt{n})+1$ and the weight at the bottom is the weight

at the top divided by $(2\cdot 2\cdot 3)^{\log(\sqrt{n})}$ (here we need to use the number of levels in the routing tree). Thus we have the weight of each node at the bottom of the routing tree (at global depth $i + \log(3n) + 2\log(\sqrt{n}) + 1$) is

$$\begin{aligned}
\sigma/(2(3n)^2 \cdot (2\cdot 2\cdot 3)^{\log(\sqrt{n})}) &\geq \sigma/(4(3n)^2 \cdot (4\cdot 3)^{\log(\sqrt{n})}) \\
&\geq \sigma/(4^{\log(3n)+1} \cdot (4\cdot 3)^{\log(\sqrt{n})}) \\
&\geq \sigma/(4^{\log(3n)+1} \cdot (4\cdot 4)^{\log(\sqrt{n})}) \\
&\geq \sigma/(4^{\log(3n)+1} \cdot (4^2)^{\log(\sqrt{n})}) \\
&\geq \sigma/(4^{\log(3n)+1} \cdot (4)^{2\cdot\log(\sqrt{n})}) \\
&= \sigma/(4^{\log(3n)+2\log(\sqrt{n})+1}) \\
&\geq (1/4)^{i+\log(3n)+2\log(\sqrt{n})+1}
\end{aligned}$$

To finish the proof, note that the base case is realized in the very first call to the algorithm, since a comparison at depth 0 has weight $1 = (1/4)^0$. □

Lemma 11 (Cole 1987 **[21])** *For $j \geq 5(i + (1/2)\log(6n))$, during the $(j+1)^{st}$ iteration there are no active comparisons at depth i.*

Proof. At the start of the $(j+1)^{\text{st}}$ iteration the total active weight $\mathbb{W}$ is bounded by $(3/4)^{5(i+(1/2)\log(6n))} \cdot (3n)$ (by Lemma 9).

We note $(3/4)^5 < (1/4)$. So

$$\begin{aligned}
\mathbb{W} &< (1/4)^{i+(1/2)\log(6n)} \cdot (3n) \\
&= (1/4)^i \cdot (1/4)^{(1/2)\log(6n)} \cdot (3n) \\
&= (1/4)^i \cdot (1/(6n)) \cdot (3n) \\
&= (1/4)^i \cdot (1/2)
\end{aligned}$$

But an active comparison at depth i has weight at least $(1/4)^i$. So there is no such comparison. □

Goodrich and Pszona [28] point out that the dependency graph for one recursive call has $O(\log n)$ height, and one recursive call of boxsort performs $O(\log n)$ parallel steps. They also cite Motwani and Raghavan [44] stating that with high probability boxsort terminates in $O(\log n)$ parallel steps, and thus the height of the whole dependency graph of the parametric search boxsort also has height $O(\log n)$ with high probability. Plugging this height into Lemma 11 as the value for i, we get that we require $O(\log n)$ calls to the decision algorithm.

Theorem 4 *Given a chord $\ell \subset P_{in}$ we compute a CCOSKEG disc $D(c^*, \rho^*)$ in $O(n\log^2 n + m)$ time with high probability using $O(n\log r + m)$ space.*

Proof. Preprocessing from Section 2 takes $O(m)$ time and space. It will be shown in Appendix A.5 that with $O(\log n + \log r)$ calls to the decision algorithm and additional $O(n\log r)$ time and $O(n\log r + r)$ space, we compute the last reflex vertices on the paths from each point $u \in S$ to $\partial D(u, \rho^*) \cap \ell$, effectively giving us $O(n)$ items to sort. Given this result and Corollary 2, the preprocessing from Section 3.1 makes $O(\log n + \log r)$ calls to the decision algorithm of Lemma 3, uses $O(n\log r + r)$ space, and takes time

$$\begin{aligned}
&O(n\log r + \log n \cdot n(\log r + \log n) + \log r \cdot n(\log r + \log n)) \\
=&O(n\log n\log r + n\log^2 n + n\log^2 r)
\end{aligned}$$

As seen in Goodrich and Pszona [28], Motwani and Raghavan [44], and Reischuk [50], with high probability (i.e., at least $1 - e^{-\log^b n}$ for some constant $b > 0$) boxsort chooses a "good" sequence of pivots so that it only requires $O(\log n)$ calls to the decision algorithm of Lemma 3; and with the same probability, taking into account the number of recursive calls and the time we spend in a recursive call to create boxes and then sort the remaining comparisons into their boxes, using boxsort for parametric search takes $O(n\log n + \log n \cdot n(\log r + \log n)))$ time and $O(n + r)$ space.

Together with the preprocessing, the time to run our parametric search using boxsort is $O(n\log n\log r + n\log^2 n + n\log^2 r + m)$ with high probability and it uses $O(n\log r + m)$ space. It produces the CCOSKEG disc by the fact that the parametric search technique finds a SKEG disc for S centred on ℓ (i.e., a disc with minimum radius centred on ℓ containing at least k points of S).

Considering the $O(m)$ time spent preprocessing the polygon, we can simplify the runtime. Consider the largest-order terms in the running time:

$$\underbrace{n\log n\log r}_{A} + \underbrace{n\log^2 n}_{B} + \underbrace{n\log^2 r}_{C} + \underbrace{m}_{D} \tag{7}$$

Either $\log r < \log n$ or $\log n < \log r$, so A is always dominated by B or C. Consequently, Expression (7) can be simplified to

$$\underbrace{n\log^2 n}_{B} + \underbrace{n\log^2 r}_{C} + \underbrace{m}_{D} \tag{8}$$

Assume C dominates B and D. This implies:

$$n \log^2 r \in \omega(m) \tag{9}$$
$$n \log^2 r \in \omega(n \log^2 n) \tag{10}$$
$$\log r \in \omega(\log n) \quad \text{by (10)} \tag{11}$$
$$\Rightarrow \quad r > n^3$$
$$\Rightarrow \quad m > n^3$$
$$\Rightarrow \quad m^{1/2} > n^{3/2}$$
$$\Rightarrow \quad m^{1/2} \in \Omega(n^{3/2})$$
$$\Rightarrow \quad m^{1/2} \in \omega(n) \tag{12}$$
$$m^{1/2} \in \omega(\log^2 m)$$
$$\Rightarrow \quad m^{1/2} \in \omega(\log^2 r) \tag{13}$$
$$m \in \omega(n \log^2 r) \quad \text{by (12) and (13)} \tag{14}$$

Consequently, Expression (8) can be simplified to $n \log^2 n + m$, meaning that the time to run our parametric search using boxsort is $O(n \log^2 n + m)$ with high probability. □

A.5 Decreasing to a Linear Number of Items to Sort

In this section, our goal is to discover, for each point of S, which reflex vertices to use for Eq. (1) when we have the optimal radius, giving us $O(n)$ equations / intersection points to use in the parametric search. We do so by using another parametric search. The procedure is straightforward; it is like one of the steps used in the boxsort parametric search of Goodrich and Pszona [28] presented in Appendix A.4, except here each comparison has one root. Similar to routing unmarked elements through the binary tree of sorted crossings, we independently route through $2n$ binary search trees of $O(\log r)$ height where the outcomes of the comparisons depend on the solution to the parametric search. This allows us to find the reflex vertices for each $u \in S$ that anchor $\partial D(u, \rho^*) \cap \ell$. However, instead of inferring the optimum by sorting intersection points described as equations from which candidates for the optimum are extracted, here our comparisons are directly in the parameter space: we are directly comparing distances against the optimum.

Since domain markers for the funnel of a point in S with ℓ are points along ℓ, in addition to defining domains for reflex vertices they also provide distances to use as candidate radii. We have the following monotonicity property: for any point $u \in S$, the distance to ℓ increases monotonically as we move from its closest point $u_c \in \ell$ to the endpoints of ℓ [48]. Thus, if we can decide how the radius produced by a given marker compares to the optimal radius, we can perform two binary searches among these markers between u_c and the endpoints of ℓ to find the domains which contain $\partial D(u, \rho^*) \cap \ell$, and hence discover which (at most two) reflex vertices to use in Eq. (1) for u in the main parametric search. Recall Observation 2 which uses the shortest-path data structure to extract the truncated funnel of u and ℓ and to perform a binary search along ℓ using the edges of this funnel.

We use this binary search to mimic the step of boxsort that routes the unmarked elements through the binary search tree of sorted crossings. The binary search can be represented as routing an element through a binary tree, discerning a particular path. Routing an element through these search trees is similar to following a directed path of $O(\log r)$ height. Each node on the path corresponds to a comparison that must be resolved before the routing is able to continue on to the next node in the path. As we route along these paths, the current node in a path is the *active comparison* in the path. When we have enough information (i.e., how ρ^* relates to the candidate at this comparison), we resolve the comparison and it is no longer active. A dependence relation arises wherein a node cannot be active until all its ancestors have been resolved, at which point it may be immediately resolved and inactivated if it is known how ρ^* relates to the candidate radius of the current comparison node; otherwise, the comparison remains active until some call to the decision algorithm reveals the relation of ρ^* to the candidate.

Since we know u_c and we have the monotonicity property, once a comparison is resolved then we know for an extension segment e in constant time and space to which side of $\ell \cap e$ a point along ℓ of distance ρ^* to u lies. Ignoring the calls to the decision algorithm, the time spent over all of the points of S to discover which reflex vertices to use for Eq. (1) in the main parametric search (i.e., the time to perform the binary searches along ℓ, or, equivalently, route through the binary trees) is $O(n \log r)$. The space is $O(r + n \log r)$ due to the fact that we will be holding on to all $2n$ shortest-path query structures at once to sequentially simulate a parallel algorithm, and since careful reading of Guibas and Hershberger [23, 29, 34] implies that a truncated funnel takes up $O(\log r)$ extra space. Each point of S has up to two independent binary trees of $O(\log r)$ height through which it is routing to find the domain of interest.

First, for each $u \in S$ and the endpoints ℓ_1 and ℓ_2 of ℓ, we compute the truncated funnel between u, ℓ_1 and u_c; and the truncated funnel between u, u_c, and ℓ_2. This is done in $O(n \log r)$ time and $O(r + n \log r)$ space via Observation 1 and the space analysis in the previous paragraph. Either of these funnels can replace the one from Observation 2 to yield the same time and space for the searches.

Then we sequentialize the running of $2n$ parallel searches through binary trees of $O(\log r)$ height (one per funnel). For each point $u \in S$ we search through the domain markers implicitly contained in its two fun-

nels looking for the domains that contain $\partial D(u, \rho^*) \cap \ell$, which are the domains that contain a point that is distance ρ^* away from u.

For each tree through which we are routing, each step produces a comparison to resolve. Since a call to the decision algorithm is considered costly, we do not want to call the decision algorithm to resolve each comparison individually. Using the fact that the candidate radii have the monotonicity property we need for parametric search (i.e., given the relation between ρ^* and a candidate radius, we know the relation between ρ^* and either everything bigger than or less than the candidate) and the fact that the domain markers used in the comparisons of the searches are also candidate radii, we can route through the trees with a logarithmic number of calls to the decision algorithm. Following Goodrich and Pszona [28], we assign weights to the comparisons in the searches. Initially, each gets a weight of 1; when a comparison is resolved its child receives half of its weight. The sum of the weights of the currently active candidates is called the *active weight*. The routing can be considered as iterations involving three steps: in the first step, we use a linear-time weighted-median-finding algorithm [51] to choose the weighted median candidate radius (as described in Appendix A.4); in the next step, we input that radius into the decision algorithm; when the decision algorithm returns, the last step is to repeatedly resolve all active comparisons that can be resolved until no more routing can be performed without knowing the result of another call to the decision algorithm. At this point, the next iteration begins.

We can follow the approach of Goodrich and Pszona [28] and use Cole's analysis [21], which we repeat here modified for this specific case, to show that there are $O(\log n + \log r)$ calls to the decision algorithm.

Lemma 12 (Cole 1987 **[21])** *At the start of the* $(j+1)^{st}$ *iteration, the active weight is bounded above by* $(3/4)^j \cdot (2n)$ *for* $j \geq 0$.

The proof of Lemma 12 is similar to that of Lemma 9 and is omitted.

Lemma 13 (Goodrich and Pszona 2013 **[28])** *Each comparison at depth* i *has weight* $\geq (1/4)^i$.

Proof. The first comparison node in each of these $2n$ paths representing the search, i.e., depth 0, has weight $1 \geq (1/4)^0$. Each step down the path halves the weight of its parent, so at depth i the weight is $(1/2)^i \geq (1/4)^i$. □

Lemma 14 (Cole 1987 **[21], Goodrich and Pszona** 2013 **[28])** *For* $j \geq 5(i + (1/2)\log(4n))$, *during the* $(j+1)^{st}$ *iteration there are no active comparisons at depth* i.

Proof. At the start of the $(j+1)^{\text{st}}$ iteration the total active weight $\mathbb{W}$ is bounded by $(3/4)^{5(i+(1/2)\log(4n))} \cdot (2n)$ (by Lemma 12).

We note $(3/4)^5 < (1/4)$. So $\mathbb{W} < (1/4)^{i+(1/2)\log(4n)} \cdot (2n) = (1/4)^i \cdot (1/4)^{(1/2)\log(4n)} \cdot (2n) = (1/4)^i \cdot (1/(4n)) \cdot (2n) = (1/4)^i \cdot (1/2)$. But an active comparison at depth i has weight at least $(1/4)^i$. So there is no such comparison. □

Plugging the height of the binary search trees into Lemma 5 as i, we get the following.

Corollary 15 *With* $O(\log n + \log r)$ *calls to the decision algorithm, with additional* $O(n \log r)$ *time and* $O(r + n \log r)$ *space, we compute for each* $u \in S$ *the anchors of* $\partial D(u, \rho^*) \cap \ell$.

B Depth Bounds

Consider the following. Imagine we preprocess P_{in} by simplifying it to P and then triangulating[8] P. If it is known that the points of S in a SKEG disc lie completely in one of those triangles, then we can solve the SKEG problem as follows. First we build Kirkpatrick's [18, 36] $O(\log r)$ query-time point-location data structure on these triangles in $O(r)$ time with $O(r)$ space. For each of the $O(r)$ triangles we build a list of the points of S contained within. So far we have used $O(n \log r + m)$ time and $O(n + m)$ space.

Now we iterate over the triangles and in each triangle use an exact algorithm for the smallest k-enclosing disc for planar instances [32]. If triangle i has n_i points of S in it, then we run the exact algorithm in $O(n_i k)$ expected time and $O(n_i + k^2)$ expected space. We know over all of the triangles, the n_i sum to n, so we have the following.

Theorem 16 *Simplify P_{in} to P and triangulate P. If the points of S in a SKEG disc are contained in a triangle of P, we solve the SKEG disc problem in $O(n \log r + nk + m)$ expected time and $O(n + k^2 + m)$ expected space.*

Below we show some bounds on the *depth* of a geodesic disc whose radius is at most four times the optimal radius of a SKEG disc, ρ^*. The depth of a disc is the number of points of S contained within. We assume no four points are geodesically co-circular, thus there are at most k points in any disc of the optimal radius ρ^*. In this paper, the $depth(\rho)$ is the maximum depth over all points in the polygon P for a geodesic disc of radius ρ. By definition, $depth(\rho^*) = k$. In this subsection, we assume that $n > kr$ (otherwise the bounds should be expressed as $\min(n, kr)$).

Lemma 17 *We have $depth(2\rho^*) \in \Omega(kr)$ under our general position assumption.*

Proof. Consider the optimal disc. It could jut into $\Omega(r)$ spikes of the polygon (i.e., contain $\Omega(r)$ reflex vertices, each of which obstructs visibility between points in the disc; e.g., Fig. 1). In each spike, if we put a disc of radius ρ^* on the boundary of the optimal disc, it could contain $\Omega(k)$ points. Then a disc of radius $2\rho^*$ centred on the optimal disc's centre has $\Omega(kr)$ points in it. □

Lemma 18 *For a constant $c > 1$, we have $depth(c\rho^*) \in \Theta(kr)$ under our general position assumption.*

[8]Note that building the shortest-path data structure of Guibas and Hershberger [29, 34] triangulates P, as does Kirkpatrick's [18, 36] point-location data structure. They both run in linear time since we have linear-time polygon triangulation algorithms [7, 18].

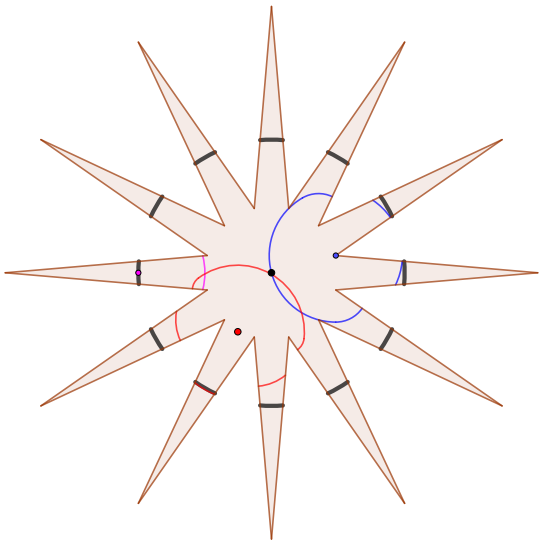


Figure 1: A star-shaped simple polygon with a geodesic disc of radius 1 and some attempts to cover multiple spikes with geodesic discs of radius 0.5.

Proof. The lower bound from Lemma 17 can be directly extended to a radius of $c\rho^*$.

We now show the upper bound. Consider the geodesic disc of radius $c\rho^*$ centred on $u \in P$, $D(u, c\rho^*)$. Let $T(P, u)$ be the shortest path tree of u and let $E(P, u)$ be the set of extension segments of $T(P, u)$. Consider the tree $T(P, u) \cup E(P, u)$. The tree $T(P, u) \cup E(P, u)$ subdivides P into $O(r)$ Euclidean triangles such that every point q in the triangle has the same anchor on $\Pi(u, q)$ [9, Note 3.10][30]. Thus, $D(u, c\rho^*)$ may intersect $O(r)$ triangles. Within any given triangle Δ, any portion of a geodesic disc appears Euclidean. As such, $D(u, c\rho^*) \cap \Delta$ (which looks locally in Δ like a Euclidean disc of radius at most $c\rho^*$) can be covered by a constant number of discs of radius ρ^* (due to the bounded doubling dimension of the Euclidean metric), each with at most k points of S in it. The bound follows. □

These bounds hold in general if nothing more is known about the manner in which a 2-approximation is produced. However, if we combine the CCOSKEG disc algorithm from Section 3 with something similar to Theorem 16 from the beginning of the section to produce a 2-approximation, then we can get a better upper bound on the number of points of S in a disc with that radius.

We assume the preprocessing of Section 2 and the preprocessing for Theorem 16 (including building a list of the points of S in each triangle) has been performed in $O(n \log r + m)$ time and $O(n + m)$ space. Either the points of S of a SKEG disc lie completely in a triangle of P, or the disc contains a point of S from each side of some diagonal. If the points of the disc are contained

in a triangle, then running the expected linear-time 2-approximation algorithm for planar instances [32] in each triangle will give us a 2-approximation. Making the appropriate change in Theorem 16, we have spent $O(n \log r + m)$ expected time and $O(n + m)$ expected space. If a SKEG disc contains at least one point of S from each side of some diagonal, then running the algorithm of Theorem 4 from Section 3.3 on each diagonal will give us a 2-approximation. Either a SKEG disc is centred on a diagonal, in which case we find it; or a SKEG disc intersects a diagonal, in which case when processing that diagonal we either compute a KEG disc centred on a point inside that SKEG disc, or a KEG disc centred on a point outside of the SKEG disc that gives us a KEG disc with a smaller radius than any KEG disc centred on a point of the diagonal inside the SKEG disc, either of which gives us a 2-approximation. Running the CCOSKEG disc algorithm of Theorem 4 on each diagonal, we spend $O(nr \log^2 n + nr \log^2 r + m)$ expected time[9] and use $O(n \log r + m)$ space.

Thus, in $O(nr \log^2 n + nr \log^2 r + m)$ expected time we have produced a 2-approximation using $O(n \log r + m)$ expected space. Let the 2-approximation radius we have found be ρ_2. We now prove $depth(\rho_2) \leq 10k$. Either a disc of radius ρ_2 is centred on a diagonal, or in a triangle of P. By definition, any disc of radius ρ_2 centred on a diagonal has at most k points in it. Now consider a disc D_2 of radius ρ_2 centred in a triangle of P. The boundary of this disc could intersect the three diagonals of the triangle. Consider the closest point of one of the diagonals to the centre of D_2, and create a disc D_ℓ of radius ρ_2 centred there. The portion of D_2 on the other side of the diagonal is contained in D_ℓ, and thus contains at most k points on the other side of the diagonal since ρ_2 is at most the radius of the CCOSKEG disc on this diagonal. Thus, at most $3k$ points of S in D_2 come from outside the triangle. Inside the triangle, locally it looks like the Euclidean plane. Thus, by the bounded doubling dimension of the Euclidean plane, D_2 contains at most $7k$ points of S contained in the triangle. If instead of *modifying* Theorem 16 we use the exact algorithm it specifies in each triangle, then the portion of D_2 inside the triangle captures at most k points of S, in which case we get $depth(\rho_2) \leq 4k$.

Theorem 19 *In $O(nr \log^2 n + nr \log^2 r + m)$ expected time we compute a radius ρ_2 using $O(n \log r + m)$ expected space that is a 2-approximation to ρ^* such that $depth(\rho_2) \leq 10k$. If we use $O(nr \log^2 n + nr \log^2 r + nk + m)$ expected time and $O(n \log r + k^2 + m)$ expected space, we can improve ρ_2 such that $depth(\rho_2) \leq 4k$.*

[9]The runtime stated in Theorem 4 hides a term dominated by the preprocessing time. Since we do not run the preprocessing for each diagonal, that simplification does not apply to this expression.

C Figures

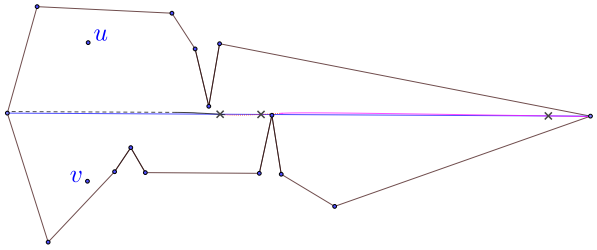


Figure 2: The bisector of points u and v on opposite sides of the blue chord of the polygon can intersect the chord $\Theta(r)$ times. The intersections are labelled with "$\times$". The different arcs that form the bisector are drawn with different ink styles (e.g., dashed vs dotted vs normal ink).

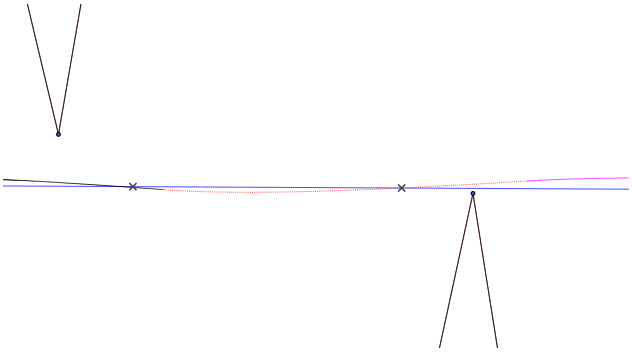


Figure 3: Zoomed-in view of the first two crossings of Fig. 2.

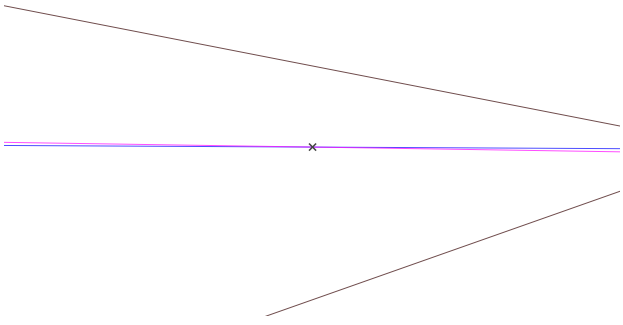


Figure 4: Zoomed-in view of the third crossing of Fig. 2. As one zooms in infinitesimally and the right side of the polygon moves toward infinity, more reflex vertices can be added to force the bisector to cross $\Theta(r)$ times.

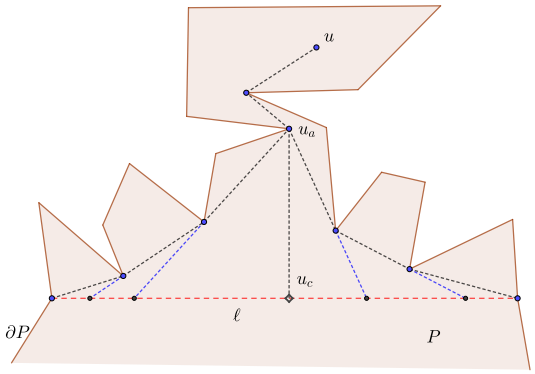


Figure 5: The funnel from u to the endpoints of ℓ, including the apex u_a and the projection u_c of u onto ℓ. Also seen are the extensions of funnel edges (in blue) and their intersection points with ℓ. These intersection points can be used to perform a binary search along ℓ.

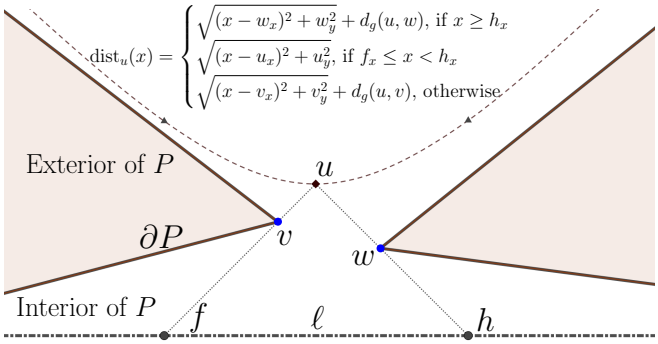


Figure 6: Considering the chord ℓ of P to be the x-axis, given a point $u \in S$ we refer to the dashed graph of the function $\text{dist}_u(\cdot)$ as the distance function of u to ℓ. The points f and h on ℓ mark where different pieces of $\text{dist}_u(\cdot)$ begin.

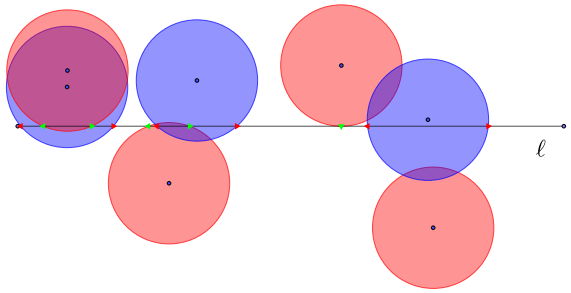


Figure 7: The geodesic discs (arbitrarily red and blue) of radius ρ^* centred on points of S (blue points) intersect ℓ. The intersection points of red (blue) disc boundaries with ℓ are marked by green (red) triangles.

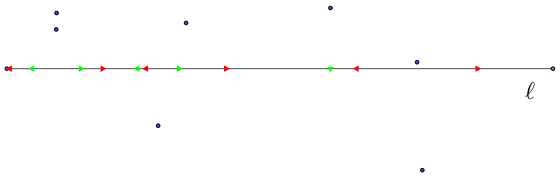


Figure 8: We use parametric search to sort the intersection points of disc boundaries (i.e., the red and green triangles) from Fig. 7 along ℓ, without knowing ρ^*, and are able to deduce ρ^* in the process.

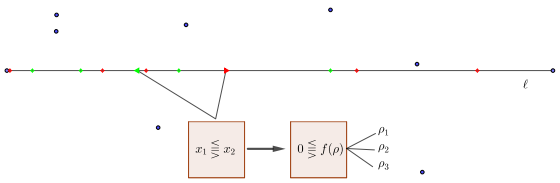


Figure 9: Parametric search lets us discover the relative order of two endpoints (e.g., the green and red triangles at positions x_1 and x_2 respectively) along ℓ. A resolved comparison in the generic algorithm reveals which of the two intersection points is to the left of the other when using ρ^* as the disc radii. Before a comparison can be resolved, the algorithm must solve for the roots of the two intersection equations (at most three roots in our case).

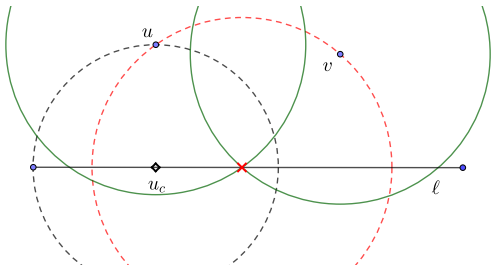


Figure 10: The CCOSKEG disc may be defined by one point on its boundary (such as the black dashed disc centred on the black hollow diamond representing u_c), in which case ρ^* is the distance from some point $u \in S$ to its projection u_c; or it is defined by at least two points, e.g., $u, v \in S$, on its boundary (such as the red dashed disc centred on the red "×"), in which case ρ^* is the radius such that the intersection of the boundaries of the green discs centred at u and v, i.e., $\partial D(u, \rho^*) \cap \partial D(v, \rho^*)$, is the red "×".

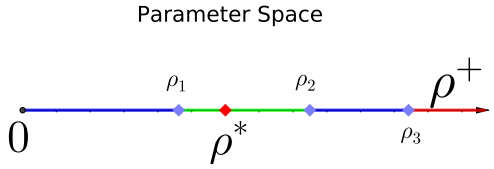


Figure 11: A comparison of intersection points along ℓ produces at most three roots and defines at most four intervals in the parameter space. Due to the monotonicity property of the parameter we are trying to optimize (i.e., radius of a CCOSKEG disc), once we determine which interval contains ρ^* we can resolve the comparison that produced the roots.

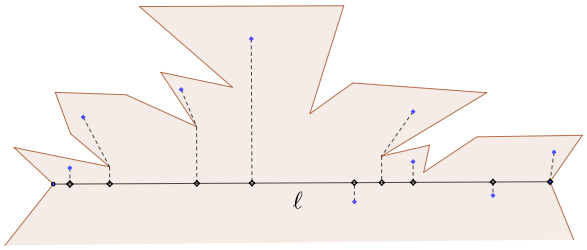
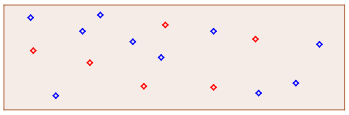
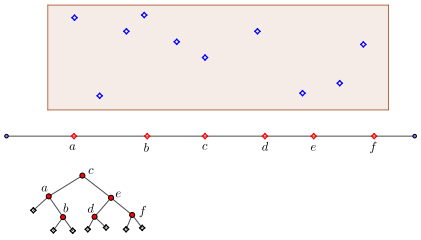


Figure 12: An example of points of S (blue diamonds) and their projections onto ℓ (hollow black diamonds).

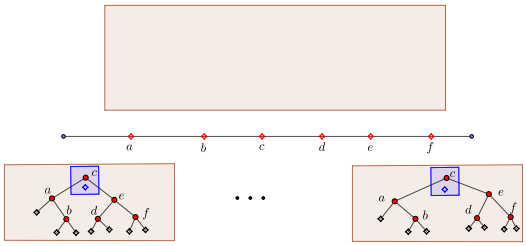


(a) We begin with a collection of equations (depicted as diamonds in a box) representing the intersection points of the disc boundaries with ℓ. Going forward we no longer distinguish between the equations and the points they represent. Of the n points, we randomly select $\sqrt{n}$ points to act as pivots. The selected points are red, and the rest are blue.

Figure 13: An illustration of a recursive call for boxsort when used as the sorting algorithm in parametric search.

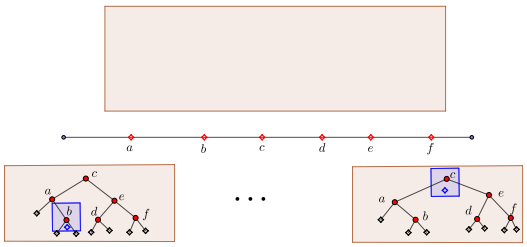


(b) We then decide the relative order of the red pivots along ℓ (i.e., decide their relative ordering) by creating $O(n)$ comparisons between them and using a logarithmic number of calls to the decision algorithm to resolve them. In this figure, the sorted red points are labelled a through f and carve ℓ up into seven relatively sorted intervals / boxes into which we must place the remaining blue points. This can be done by performing a binary search on the red pivots. The red pivots can be considered as creating a binary search tree (with c as the root in this example).

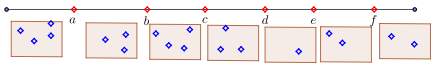


(c) The routing of each blue point through the tree of red pivots can be done independently of the other blue points, allowing us to perform their routing in parallel. For a blue point we are routing through the tree, each step in the tree creates a comparison between the blue point and the red pivot represented by the tree node. To route the elements through the tree, we repeatedly select the weighted median of the roots produced by the available comparisons and use this median in a call to the decision algorithm. After the call to the decision algorithm, we resolve whichever comparisons it is possible to resolve (which moves a blue point down the tree), and repeat until it is known into which intervals along ℓ each blue point belongs.

Figure 13: An illustration of a recursive call for boxsort when used as the sorting algorithm in parametric search.

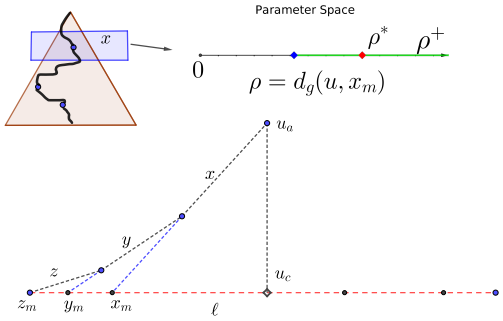


(d) Since the decision algorithm is called on the weighted median of the candidates and each instance of routing through the trees is independent, at any point in this routing process blue points may be at different levels in the binary search trees of red pivots.



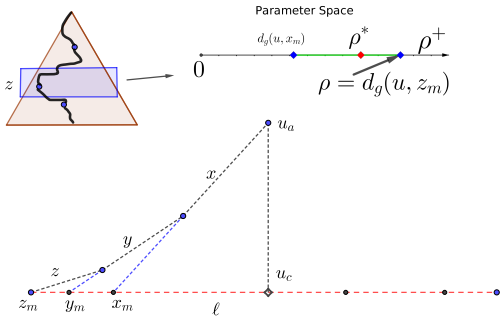
(e) Once it is known into which interval each blue point belongs, we collect them together in each interval to begin the next recursive calls.

Figure 13: An illustration of a recursive call for boxsort when used as the sorting algorithm in parametric search.

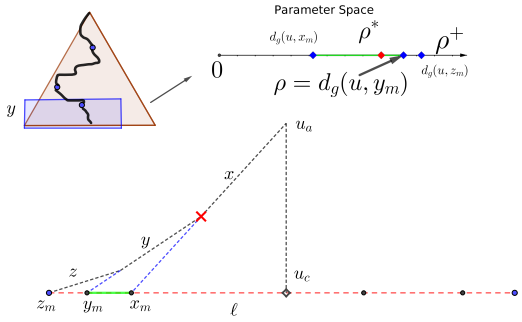


(a) Shown is the funnel between u_a for a point $u \in S$, its projection u_c, and an endpoint of ℓ. The convex chain representing the funnel edges is stored in a binary search tree (which stores the edges). In this example, the funnel edges from u_a to the left endpoint of ℓ are x, y, and z, and their extension segments (the blue dashed edges) intersect ℓ at the markers x_m, y_m, and z_m respectively. We can use the distances between u and the markers as candidate radii in the decision algorithm. Since the distance from u to the points on ℓ increases monotonically as we move from u_c to the left endpoint of ℓ, we can use these distances to find an interval in the parameter space in which ρ^* lies, and at the same time the interval along ℓ between two markers where the disc of radius ρ^* centred at u intersects ℓ. In this example, the decision algorithm has determined that the optimal radius is larger than $d_g(u, x_m)$, so we continue down the binary search tree on the side that brings us closer to the left endpoint of ℓ.

Figure 14: An illustration of the method presented in Section 3.4 for determining the reflex vertices to use in Eq. (1).

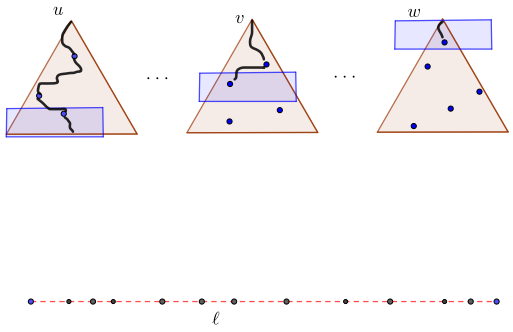


(b) After the edge x we visited the edge z. The decision algorithm has determined that the optimal radius is less than $d_g(u, z_m)$, so we continue down the binary search tree on the side that brings us closer to x_m.



(c) After the edge z we visited the edge y. The decision algorithm has determined that the optimal radius is less than $d_g(u, y_m)$, so we conclude that $D(u, \rho^*)$ intersects ℓ in the green interval between y_m and x_m, and the reflex vertex to use in Eq. (1) is the one marked by the red $\times$.

Figure 14: An illustration of the method presented in Section 3.4 for determining the reflex vertices to use in Eq. (1).



(d) Similar to boxsort (see Fig. 13), routing through the funnels to find the required reflex vertices is done in parallel, repeatedly testing the radius produced by the median weighted comparison with the decision algorithm and then resolving any number of comparisons. The routing process for different funnels may be at different depths in the trees at any given moment. In this example, the routing for $u \in S$ has finished, the routing for $v \in S$ is halfway through its binary search, and the routing for $w \in S$ is still at the beginning.

Figure 14: An illustration of the method presented in Section 3.4 for determining the reflex vertices to use in Eq. (1).