Three problems about simple polygons*

Timothy M. Chan†

September 2, 2004

Abstract

We give three related algorithmic results concerning a simple polygon P:

1. Continuing previous efforts by Bespamyatnikh, Biedl, Bose, Czyzowicz, E. Demaine, M. Demaine, Kim, Kranakis, Lubiw, Maheshwari, Morin, Shin, Toussaint, Vigneron, and Yang, we show how to find a largest pair of disjoint congruent disks inside P in linear expected time.
2. As a subroutine for the above result, we show how to find the convex hull of any given subset of the vertices of P in linear worst-case time.
3. More generally, we show how to compute a triangulation of any given subset of the vertices or edges of P in almost linear time.

Keywords: Geometric optimization; Polygon triangulation; Convex hull

1 Introduction

This paper gives linear or almost-linear algorithms for three problems about simple polygons and addresses some basic questions that haven't been raised before (to the best of the author's knowledge) and are thus surprising (if one considers the wealth of existing results on simple-polygon computations).

The problem that started off this research is well-studied:

> PACK-2-DISKS: Given a simple polygon P with n vertices, find two disjoint congruent disks inside P, maximizing the radius. (See Figure 1 (left), shown with a suboptimal solution.)

This problem has attracted attention from a number of computational geometers, as it is not only an example of a polygon containment problem (how to pack multiple objects inside a container) but also an example of a facility location problem (a variant of the standard two-center problem) [2]. The first appearance of the problem [8], though, was inspired by quite a different theme (how to wrap a disk with a polygonal piece of paper).

Here is a quick summary of the previous results: For the special convex-polygon case, there is an optimal $O(n)$ algorithm by Bose *et al.* [9], improving a previous $O(n \log n)$ algorithm by Kim *et al.* [23]. For an arbitrary simple polygon, there is a randomized $O(n \log n)$ algorithm by Bose,

*This work was supported in part by an NSERC Research Grant.

†School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada, `tmchan@uwaterloo.ca`

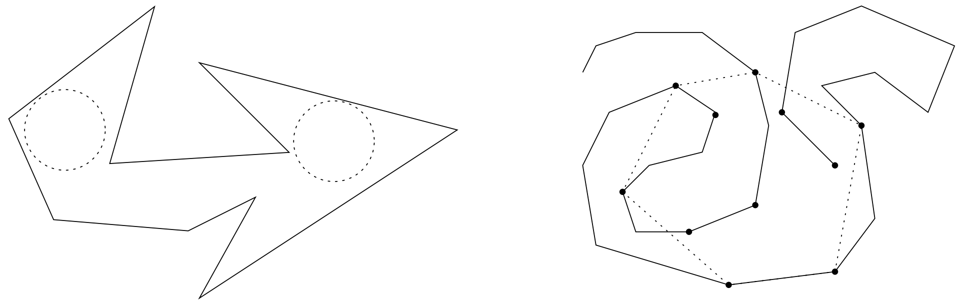

Figure 1: Examples for PACK-2-DISKS and CH-OF-SUBSET.

Morin, and Vigneron [10], improving previous deterministic $O(n \log^3 n)$ and $O(n \log^2 n)$ algorithms by Bespamyatnikh [5, 6], which in turn were improvements over the original $O(n^2)$ algorithm by Biedl *et al.* [8]. Bose, Morin, and Vigneron's $O(n \log n)$ algorithm works for general polygons with holes, for which there is an $\Omega(n \log n)$ lower bound [10]. The possibility of a linear-time algorithm for simple polygons was left open.

We settle this open question by presenting a randomized $O(n)$ algorithm for PACK-2-DISKS. The algorithm is obtained by solving the decision problem first and applying a general randomized optimization technique [11].

In designing the decision algorithm, we encounter a curious subproblem that seems new and fundamental:

> CH-OF-SUBSET: Given a simple polygon or chain P with n vertices and a subset Q of these vertices, compute the convex hull of Q. (See Figure 1 (right).)

The standard 2-d convex hull problem of course requires $\Theta(n \log n)$ worst-case time [34]. However, because the points in Q are already "sorted"—in the sense that they are ordered along a Jordan curve—we might expect a better result.

Indeed, in the 1980s, a number of simple algorithms have been proposed that can compute the convex hull of a simple polygon or chain in linear time, by variations of Graham's scan [7, 20, 27, 29, 31, 34]. Unfortunately, these algorithms cannot be applied to our convex hull problem, where Q is a subset of the polygon, not the whole polygon. A rough explanation is that these previous algorithms exploit not only the "sortedness" of the input, but also its "shape"—a sparse subset may lose the shape of the original polygon. For a more concrete explanation, the "Jordan sorting" problem can be reduced to CH-OF-SUBSET but seems to require an algorithm [18, 22] more powerful than Graham's scan. (The Jordan sorting problem is to sort the intersections of a polygon with a line; by bending the line slightly into a strictly convex curve, we can intersect each edge with the curve and sort the intersection points by computing their convex hull.)

We show that CH-OF-SUBSET can be solved optimally in $O(n)$ time—in other words, "sortedness" alone without "shape" still helps for the convex hull problem! Not surprisingly, the linear-time algorithm for polygon triangulation by Chazelle [12] (or Amato, Goodrich, and Ramos [3]) is used as preprocessing, but interestingly, an adaptation of Kirkpatrick's well-known method for point location [24] turns out to be the key. Alternatively, we also give a simpler randomized algorithm that runs in $O(n \log^* n)$ expected time, where $\log^*$ denotes the iterated logarithm function. The idea here is to modify directly the simpler, randomized $O(n \log^* n)$ triangulation algorithm by Clarkson, Tarjan, and Van Wyk [16] or Seidel [35].

Naturally one can ask whether sortedness helps for other problems, such as computing a triangulation of a point set or a triangulation of a set of disjoint line segments, which requires $\Theta(n \log n)$ time for unsorted data. This question leads to a problem stronger than CH-OF-SUBSET:

> TRIANGULATION-OF-SUBSET: Given a simple polygon P with n vertices and a subset Q of the vertices or edges of P, compute a triangulation of Q. (The triangulation should use only the vertices or endpoints of Q and cover the convex hull of Q.)

Chazelle's algorithm [12] can already triangulate the edge set of the polygon in linear time, if applied to the inside as well as the pockets of the polygon. However, when Q is an arbitrary subset of the vertex or edge set, the triangulation thus computed has extra vertices that need to be removed. It is not clear how a triangulation of a subset can be extracted from a triangulation of the whole set in linear time. Note that this is possible if "triangulation" is replaced by "Delaunay triangulation", as Chazelle *et al.* [13] have recently demontrated with a randomized algorithm (they refer to the problem as "trangulation splitting"). However, we are not given the Delaunay triangulation of the vertices of the polygon (which requires $\Omega(n \log n)$ time to construct in general). On another related note, Kirkpatrick, Klawe, and Tarjan [25] discussed the problem of removing Steiner points from a triangulation of a simple polygon, but not a triangulation of a point set.

Nevertheless, we show that TRIANGULATION-OF-SUBSET can be solved in $O(n \log^* n)$ time. The new algorithm repeatedly uses planar graph separators and invokes Bar-Yehuda and Chazelle's triangulation algorithm for polygons with holes [4] (which in turn calls Chazelle's triangulation algorithm for simple polygons). This result is noteworthy in that there are not too many examples of deterministic $O(n \log^* n)$ algorithms in this area (with one exception being [21]). Although $\log^* n$ is "practically" bounded by a constant, our result raises a tantalizing theoretical question: can TRIANGULATION-OF-SUBSET be solved in linear time?

2 Packing two disks in a simple polygon

In this section, we address the PACK-2-DISKS problem. The basic idea is to solve the decision problem first (testing whether the maximum radius is at least a given number r) and then invoke the author's optimization technique [11]. The decision problem is essentially a problem about an "offset" of the given polygon P (i.e., all points at a fixed distance from the boundary of P). The main difficulty in obtaining linear running time is that this offset "polygon" is not necessarily connected, so usual algorithms for simple polygons cannot be applied. Fortunately, it is possible to connect up the offset into one simple chain so that the result of the next section on CH-OF-SUBSET can be used. In addition, we have to overcome some technical challenges, to avoid working with curved segments of the offset and to ease the subsequent application of the optimization technique.

2.1 Preliminaries

We assume general position throughout, by known perturbation methods.

As preprocessing, a natural first step is to compute the *medial axis* of the given polygon P (i.e., the Voronoi diagram of its vertices and edges inside P). A point q inside P is on the medial axis if there are two points on the boundary of P that are both nearest to q. The medial axis forms a tree T_M consisting of straight-line and parabolic segments, with total size $|T_M| = O(n)$. It can be computed in $O(n)$ time by a randomized algorithm of Klein and Lingas [26] or a deterministic algorithm by

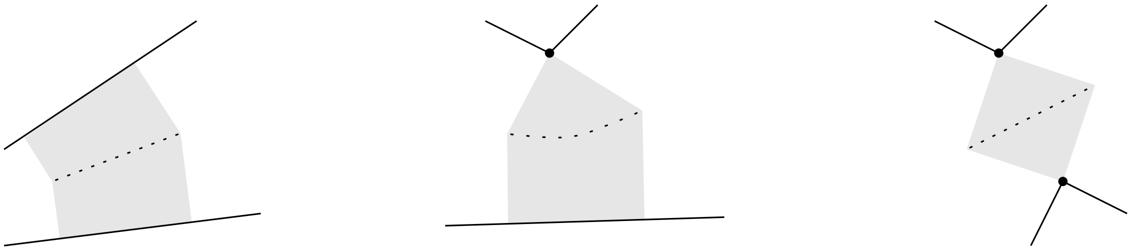

Figure 2: Possible pictures of the region $R(\gamma)$. (The medial-axis segment γ is shown in dotted lines.)

Chin, Snoeyink, and Wang [14]; both use a linear-time polygon triangulation algorithm [3, 12] as a subroutine. Alternatively, if we are happy with an almost-linear time bound, we can use a simpler randomized $O(n \log^* n)$ algorithm by Devillers [17], which is based on Seidel's polygon triangulation algorithm [35].

The medial axis naturally partitions P into regions (Voronoi regions). To avoid curved boundaries and ensure that regions have constant complexity, we work with a different partition: Every segment γ of the medial axis is defined by the bisector of two features f_1, f_2, where a feature can be either a vertex of P or the line through an edge of P. Let $R(\gamma)$ be the polygon inside P formed by taking each of the two endpoints p of γ and drawing straight line segments from p to its two nearest points on f_1 and f_2; as shown in Figure 2, this polygon has 4 to 6 sides. For any point $q \in R(\gamma)$, its nearest point on the boundary of P must then be defined by f_1 or f_2, depending on which side of γ the point is on. As can be easily checked, these regions $R(\gamma)$ are disjoint, cover P, and can be generated in $O(n)$ time.

Let Π_M be the cycle formed by traversing every segment of the medial axis T_M twice without crossing. We will solve a slight generalization of the problem:

> Given two subpaths Π_1 and Π_2 of Π_M, find the largest value r such that property $\mathcal{P}(\Pi_1, \Pi_2, r)$ is true, where $\mathcal{P}(\Pi_1, \Pi_2, r)$ refers to the existence of two points q_1 and q_2 (the "centers") satisfying:
>
> 1. $q_i \in R(\gamma_i)$ for some $\gamma_i \in \Pi_i$,
> 2. the distance of q_i to its nearest neighbor on the boundary of P exceeds r, and
> 3. the distance between q_1 and q_2 exceeds $2r$.

The original version of the problem is when $\Pi_1 = \Pi_2 = \Pi_M$.

2.2 Decision algorithm

The decision problem is this: given subpaths Π_1, Π_2 and a value r, determine whether $\mathcal{P}(\Pi_1, \Pi_2, r)$ is true.

We will rephrase the problem. Given a segment γ, let $R(\gamma, r)$ be the set of all points in $R(\gamma)$ whose nearest neighbor distance to the boundary of P exceeds r. We want to determine whether there exist a point in $\bigcup_{\gamma \in \Pi_1} R(\gamma, r)$ and a point in $\bigcup_{\gamma \in \Pi_2} R(\gamma, r)$ of distance at least $2r$ apart. It suffices to compute the farthest pair between $\bigcup_{\gamma \in \Pi_1} R(\gamma, r)$ and $\bigcup_{\gamma \in \Pi_2} R(\gamma, r)$.

Given the convex hulls of two sets in the plane, the farthest pair between the two sets can be determined in linear time by a simple method of "rotating calipers" [34]. So, it suffices to construct the convex hull of $\bigcup_{\gamma \in \Pi_i} R(\gamma, r)$ for each $i \in \{1, 2\}$.

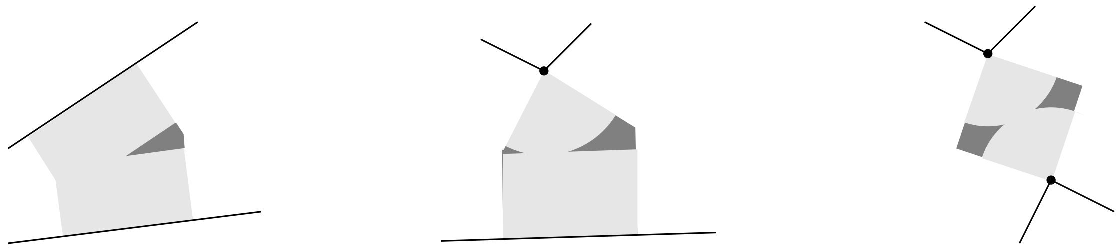

Figure 3: Possible pictures of the region $R(\gamma, r)$ (shown in dark shade).

What does $R(\gamma, r)$ look like? Say γ is defined by the bisector of features f_1 and f_2. Then $R(\gamma, r)$ is the intersection of the set $R(\gamma)$, the set of all points of distance at least r from f_1, and the set of all points of distance at least r from f_2. The first set is a polygon, but the second/third set can be a halfplane (if the feature is a line) or the complement of a disk (if the feature is a point). So, although $R(\gamma, r)$ has constant complexity, it may be disconnected and may involve circular arcs. (See Figure 3.)

A simple observation allows us to bypass curved boundaries: to compute the convex hull of $\bigcup_{\gamma \in \Pi_i} R(\gamma, r)$, it suffices to compute the convex hull of $S_i := \bigcup_{\gamma \in \Pi_i} \operatorname{ext} R(\gamma, r)$, where $\operatorname{ext} R(\gamma, r)$ denotes the set of extreme points of $R(\gamma, r)$ (vertices of its convex hull). The convex hull of each $R(\gamma, r)$ is a polygon, because the circular arcs that bound $R(\gamma, r)$ are concave. So, each $\operatorname{ext} R(\gamma, r)$ is a discrete set of points.

Easily, we can form the point set Q_i in $O(|\Pi_i|)$ time. Moreover, we can form a simple chain that passes through all points of Q_i and possibly extra points, in $O(|\Pi_i|)$ time, as follows. For each $\gamma \in \Pi_i$, we can generate one or two (depending on the multiplicity of γ in Π_i) non-intersecting polygonal subchains that goes from one endpoint to the other endpoint of γ, stays inside $R(\gamma)$, and visits all points in $\operatorname{ext} R(\gamma, r)$ (if any). Because $R(\gamma)$ and $\operatorname{ext} R(\gamma, r)$ have constant complexity, it is a simple matter to construct such subchains of constant size. Since Π_i is a path, we can join these subchains together into a single chain. The chain can self-intersect only at endpoints and can be made simple (by perturbing endpoints slightly if necessary). By the linear-time algorithm in the next section, we can construct the convex hulls of Q_i and therefore solve the decision problem in $O(|\Pi_1| + |\Pi_2|)$ time, excluding the initial preprocessing.

2.3 Optimization algorithm

Using this decision oracle, we could apply parametric search [30] to solve PACK-2-DISKS (as Bespamyatnikh did [5]), but this would result in polylogarithmic-factor slowdown (not to mention a tricky simulation of a parallel algorithm). Instead, we apply the author's randomized optimization technique [11].

The only requirement of the technique is a procedure to divide the problem into a constant number of subproblems, each of a fraction of the original size, such that the solution to the original problem is the minimum/maximum of the solutions to the subproblems. Our generalized formulation of the problem permits this without much effort: break Π_i into two subpaths Π_{i1}, Π_{i2}, each of size $|\Pi_i|/2$. Let $w(\Pi_1, \Pi_2)$ denote the largest r satisfying $\mathcal{P}(\Pi_1, \Pi_2, r)$. Clearly,

$$w(\Pi_1, \Pi_2) = \max\{w(\Pi_{11}, \Pi_{21}), w(\Pi_{11}, \Pi_{22}), w(\Pi_{12}, \Pi_{21}), w(\Pi_{12}, \Pi_{22})\}.$$

Thus, by the optimization technique [11], we can compute $w(\Pi_1, \Pi_2)$ with the same asymptotic expected time bound as the decision algorithm. So, the expected running time is $O(n)$, even including the preprocessing.

3 Finding the convex hull of a subset of a simple polygon

In this section, we solve the CH-OF-SUBSET problem (and in particular complete our solution of PACK-2-DISKS). We offer two solutions. The first is randomized and slightly slower, but avoids calling a linear-time polygon triangulation algorithm; it is a direct modification of the well-known $O(n \log^* n)$ triangulation algorithms by Clarkson, Tarjan, and Van Wyk [16] and Seidel [35]. The second is deterministic and harder to implement, but runs in optimal linear time; it is based on the well-known point location method by Kirkpatrick [24].

3.1 A randomized $O(n \log^* n)$ algorithm

We follow the sampling approach of Clarkson, Tarjan, and Van Wyk [16] and Seidel [35] (see also [17, 32]), with a slight change in parameters. Let S be the edge set of the given simple polygon. Pick a random permutation $s_1, \ldots, s_n$ of S and let $R_i = \{s_1, \ldots, s_{r_i}\}$, where $r_0, \ldots, r_\ell$ is a increasing sequence to be specified later, with $r_0 = 1$ and $r_\ell = n$.

The original approach was designed for the problem of constructing the *trapezoidal decomposition* $T(S)$, defined as the partition of the plane into cells (trapezoids or triangles) formed by S and the vertical rays shot upward and downward from each endpoint of S. The algorithm works by iterating over each $i = 0, \ldots, \ell - 1$ and refining $T(R_i)$ to build $T(R_{i+1})$. It is not necessary to repeat specific details of this algorithm and analysis (see [16, 32, 35]), but we mention two relevant facts: First, the expected running time of each iteration is $O(r_{i+1} \log(r_{i+1}/r_i))$. Second, within the same time bound, we can set up pointers between each cell $\Delta \in T(R_i)$ and each cell $\Delta' \in T(R_{i+1})$ that intersects Δ.

To solve CH-OF-SUBSET, the idea is to follow this "top-down" algorithm with a new "bottom-up" phase, which we now describe. For each cell Δ of the trapezoidal decompositions generated, we will maintain a convex polygon H_Δ satisfying two invariants: (i) all vertices of H_Δ is from $Q \cap \Delta$, and (ii) H_Δ contains all extreme points of Q that are inside Δ.

For $\Delta \in T(R_\ell)$ at the bottommost level, we just set H_Δ to be the convex hull of $Q \cap \Delta$, which has constant size. We then iterate over each $i = \ell - 1, \ldots, 0$, and for each $\Delta \in T(R_i)$, set H_Δ to be the convex hull of $\{\text{restrict}(H_{\Delta'}, \Delta) \mid \Delta' \in T(R_{i+1}),\ \Delta' \cap \Delta \neq \emptyset\}$, where $\text{restrict}(H_{\Delta'}, \Delta)$ denotes the convex polygon obtained by deleting vertices not in Δ from the vertex set of $H_{\Delta'}$. To compute H_Δ, we need to compute the convex hull of a collection of disjoint convex polygons; this is explained below. For $\Delta \in T(R_0)$ at the topmost level, we return H_Δ. The answer is correct by the invariants.

The following lemma provides the missing subroutine:

Lemma 3.1 *The convex hull of k disjoint convex n-gons can be constructed in $O(k \log k \log n)$ time.*

Proof: Nielsen and Yvinec [33] studied the 2-d convex hull problem for a collection of general convex objects and gave output-sensitive algorithms, but in our case, we can use instead the standard divide-and-conquer algorithm: Namely, divide the collection of disjoint convex polygons arbitrarily into two subcollections each with $k/2$ convex polygons, recursively compute the convex hull of each subcollection, and merge the two hulls.

The lower/upper hull of k disjoint convex polygons dualizes to the lower/upper envelope of k concave/convex polygonal "curves". Since a pair of disjoint convex polygons admits two common tangents, a pair of the dual curves intersect at most twice. By a simple result on Davenport-Schinzel sequences [1], the envelope thus consists of $O(k)$ curve pieces, or "arcs", forming an "arc-gon". We can compute the lower/upper envelope of two such arc-gons, by sweeping the arcs from left to right (in primal space, this corresponds to "rotating calipers"). The sweep requires $O(k)$ primitive operations, each involves intersecting two arcs (in primal space, this corresponds to finding a common tangent), doable in $O(\log n)$ time by binary search [34]. So, merging can be performed in $O(k \log n)$ time. The entire divide-and-conquer algorithm thus takes $O(k \log k \log n)$ time. $\square$

3.2 Analysis

To analyze the cost of the bottom-up phase, consider the i-th iteration. Let $n_\Delta = |Q \cap \Delta|$, $n_{\Delta'} = |Q \cap \Delta'|$, and k_Δ be the number of cells $\Delta' \in T(R_{i+1})$ that intersect Δ. We can collectively compute all restrict$(\cdot, \cdot)$ in time proportional to $\sum_{\Delta' \in T(R_{i+1})} n_{\Delta'}$ plus the total number of pointers $\sum_\Delta k_\Delta$. By Lemma 3.1, we can compute each convex hull H_Δ in time $O(k_\Delta \log k_\Delta \log n_\Delta)$. Note that this expression is upper-bounded by $O(n_\Delta + k_\Delta \log^2 k_\Delta)$ (just consider two cases $n_\Delta \leq k_\Delta^2$ and $n_\Delta \geq k_\Delta^2$).

Now, $\sum_{\Delta \in T(R_i)} n_\Delta = O(n)$ and $\sum_{\Delta' \in T(R_{i+1})} n_{\Delta'} = O(n)$. Since k_Δ is proportional to the number of segments in R_{i+1} that intersect Δ, and since R_i is a random subset of R_{i+1} of size r_i, we can apply the general sampling analysis of Clarkson and Shor [15] to bound the expectation of $\sum_{\Delta \in T(R_i)} k_\Delta \log^2 k_\Delta$ by $O(r_i \cdot (r_{i+1}/r_i) \log^2(r_{i+1}/r_i))$. The total expected cost for the i-th iteration is thus $O(n + r_{i+1} \log^2(r_{i+1}/r_i))$.

The expected cost of the entire algorithm is

$$O\left(n\ell + \sum_{i=0}^{\ell-1} r_{i+1} \log^2(r_{i+1}/r_i)\right).$$

Choosing the sequence $r_i = \left\lfloor n/(\log^{(i)} n)^2 \right\rfloor$ for $i = 1, \ldots, \ell$ with $\ell = \log^* n$, we can bound each term in the above sum by $O(n)$ (since $\log(r_{i+1}/r_i) = O(\log[(\log^{(i)} n)^2]) = O(\log^{(i+1)} n)$). So, the expected running time is $O(n \log^* n)$.

3.3 A deterministic $O(n)$ algorithm

We start with a triangulation T of the vertex set of the polygon, computed by Chazelle's linear-time algorithm [12]. We then build Kirkpatrick's hierarchical data structure, originally designed for the planar point location problem [24, 32, 34].

To remind the readers, a brief description of Kirkpatrick's approach is provided below. First, by adding three special vertices, assume that the outer face of T is a triangle. A sequence of triangulations $T_0, T_1, \ldots, T_\ell$ is built, where $T_0 = T$ and T_ℓ is a single triangle. We modify the triangulation T_i to form a smaller triangulation T_{i+1} as follows: Select an independent set I of size at least αn from T_i (excluding the three special vertices), where the degree of each vertex in I is bounded by c; this step can be carried out in linear time for an appropriate choice of the constants $\alpha > 0$ and c (see [24, 32, 34]). Remove I from T_i to obtain a planar graph T_i'. Note that each non-triangular face in T_i' must be a star-shaped polygon with at most c sides. Triangulate the faces of T_i' to obtain T_{i+1}. We can set up a pointer between each triangle $\Delta \in T_i$ and each triangle $\Delta' \in T_{i+1}$ that intersects Δ.

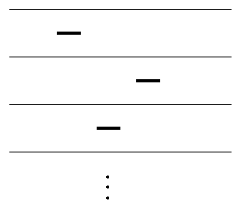

Figure 4: Knowing the trapezoidal decomposition of all the line segments does not help in computing the trapezoidal decomposition of the bold segments.

To solve CH-OF-SUBSET, we use an idea similar to the previous algorithm. For each triangle $\Delta \in T_i$ in the hierarchy, we will maintain a convex polygon H_Δ satisfying two invariants: (i) all vertices of H_Δ is from $Q \cap \Delta$, and (ii) H_Δ contains all extreme points of Q that are inside Δ.

For $\Delta \in T_0$, we just set H_Δ to be the convex hull of $Q \cap \Delta$, which has constant size. At the i-th iteration, for each $\Delta' \in T_{i+1}$, we set $H_{\Delta'}$ to be the convex hull of $\{\mathrm{restrict}(H_\Delta, \Delta') \mid \Delta \in T_i,\ \Delta \cap \Delta' \neq \emptyset\}$, where $\mathrm{restrict}(H_\Delta, \Delta')$ denotes the convex polygon obtained by deleting vertices not in Δ' from the vertex set of H_Δ. For $\Delta \in T(R_\ell)$, we return H_Δ. Correctness again follows from the invariants.

3.4 Analysis

The triangulation T_i has at most $n_i := (1-\alpha)^i n$ vertices. Each triangle in T_i is contained in one face of T_i' and thus intersects at most $c-2$ triangles in T_{i+1}. Similarly, each triangle in T_{i+1} intersects at most c triangles in T_i. Hence, for each $\Delta \in T_i$, the polygon H_Δ has at most $m_i := O(c^i)$ vertices.

Consider the i-th iteration. The triangulation T_{i+1} and the pointers between T_i and T_{i+1} can be computed in linear time, because c is a constant. This time, the computation of $\mathrm{restrict}(\cdot,\cdot)$ is trickier, as we cannot afford to scan through all vertices of the polygons H_Δ. Instead of a plain array, we need to store each H_Δ in a balanced search tree, so as to support splitting and joining in logarithmic time [36]. For each $\Delta \in T_i$, we generate $\mathrm{restrict}(H_\Delta, \Delta')$ for every triangle $\Delta' \in T_{i+1}$ that intersects Δ, by splitting the polygon H_Δ at the ≤ 6 intersections with $\partial\Delta'$. For each $\Delta' \in T_{i+1}$, we compute the convex hull H'_Δ, this time, of only a constant number c of disjoint convex polygons; Lemma 3.1 still applies, using a constant number of splits and joins, and yields a logarithmic running time. The total cost of the i-th iteration is thus $O(n_i \log m_i)$.

The cost of the entire algorithm is

$$O\left(\sum_{i=0}^{\ell} n_i \log m_i\right) \;=\; O\left(\sum_{i=0}^{\ell} (1-\alpha)^i n \log(c^i)\right) \;=\; O\left(n \sum_{i=0}^{\infty} i(1-\alpha)^i\right) \;=\; O(n).$$

4 Triangulating a subset of a simple polygon

To appreciate the delicacy of the algorithms from the previous section, notice that they cannot be strengthened to output the trapezoidal decomposition of a subset Q of edges. In fact, a simple reduction (see Figure 4) shows that if this were possible, we would be able to sort in $o(n \log n)$ time! In this section, we show that in contrast, a triangulation of a subset Q can be found in almost linear time by a different algorithm, thus solving the TRIANGULATION-OF-SUBSET problem.

4.1 A deterministic $O(n \log^* n)$ algorithm

As suggested in the introduction, our idea to solve TRIANGULATION-OF-SUBSET is to start with a triangulation T of the entire edge set of the given simple polygon, computed by Chazelle's linear-time algorithm [12], and then remove the extra vertices from T. Vertex removal is nontrivial, and we propose a new method that can be viewed as a variation of Kirkpatrick's point location method [24]—instead of an independent set of vertices, we find an "independent" collection of small clusters, remove them simultaneously, retriangulate, and repeat. A corollary of Lipton and Tarjan's planar separator theorem [28] will be used:

Lemma 4.1 *Given a planar graph G with n vertices and a parameter b, one can find a subset S of $O(n/\sqrt{b})$ vertices in $O(n)$ time, such that each connected component of $G \setminus S$ has $O(b)$ vertices.*

Proof: The result follows by applying the planar separator theorem recursively. See Goodrich's paper [19], which established the linear time bound for the recursive application. □

Our algorithm is iterative. We first mark vertices that need to be removed, i.e., vertices of T that are not vertices or endpoints of Q. By adding three special unmarked vertices, assume that the outer face of T is a fixed triangle. At the beginning of the i-th iteration, assume that there are at most r_i marked vertices, where $r_0, \ldots, r_\ell$ is a decreasing sequence to be specified later, with $r_0 = n$ and $r_\ell = 0$.

Let G_m be the subgraph of T induced by the $\le r_i$ marked vertices. We apply Lemma 4.1 to G_m to obtain a subset S of $\le r_{i+1}$ marked vertices such that each component of $G_m \setminus S$ has size $O((r_i/r_{i+1})^2)$. For each such component Γ, we build its *neighborhood* $N(\Gamma)$, defined as the union of all triangles in T that are incident to vertices in Γ. Since no edge joins two different components, no triangle is shared by two neighborhoods, i.e., the neighborhoods have disjoint interiors.

Each neighborhood $N(\Gamma)$ is a polygon possibly with holes. We can trace the boundaries of $N(\Gamma)$ by traversing along each face of Γ individually. Each such face yields a single closed polygonal chain that self-intersects only at vertices or edges and can be made simple (by slight perturbation). Thus, the number of boundary pieces or holes of $N(\Gamma)$ is bounded by the number of faces of Γ, which is $O(|\Gamma|)$.

We now remove the vertices in each component Γ from T and retriangulate its neighborhood $N(\Gamma)$ by using Bar-Yehuda and Chazelle's triangulation algorithm [4] for polygons with holes (perturbation can be undone by coalescing duplicate vertices and edges). As a result, we get a smaller triangulation T where the only marked vertices left are the $\le r_{i+1}$ vertices of the separator S. This completes the i-th iteration.

4.2 Analysis

Consider the i-th iteration. Bar-Yehuda and Chazelle's algorithm [4] triangulates a polygon with n vertices and p holes in $O(n + p \log^{1+\varepsilon} p)$ time for any fixed $\varepsilon > 0$. When applied to $N(\Gamma)$, the running time is $O(|N(\Gamma)| + |\Gamma| \log^{1+\varepsilon} |\Gamma|)$. Note that the total size of all neighborhoods is $O(n)$, and the total size of all components is $O(r_i)$, and the maximum component size is $O((r_i/r_{i+1})^2)$. The total cost for the i-th iteration is thus $O(n + r_i \log^{1+\varepsilon}[(r_i/r_{i+1})^2])$.

The cost of the entire algorithm is

$$O\left(n\ell + \sum_{i=0}^{\ell-1} r_i \log^{1+\varepsilon}(r_i/r_{i+1})\right).$$

Choosing the sequence $r_i = \left\lfloor n/(\log^{(\ell-i)} n)^{1+\varepsilon} \right\rfloor$ for $i = 0, \ldots, \ell$ with $\ell = \log^* n$, we can bound each term in the above sum by $O(n)$. So, the worst-case running time is $O(n \log^* n)$.

Acknowledgements

I thank Ralph Boland for helpful discussions on some of these problems.

References

[1] P. K. Agarwal and M. Sharir. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, New York, 1995.

[2] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30:412–458, 1998.

[3] N. M. Amato, M. T. Goodrich, and E. A. Ramos. A randomized algorithm for triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 26:245–265, 2001.

[4] R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *Int. J. Comput. Geom. Appl.*, 4:475–481, 1994.

[5] S. Bespamyatnikh. Efficient algorithm for finding two largest empty circles. In *15th European Workshop on Comput. Geom.*, pages 37–38, 1999.

[6] S. Bespamyatnikh. Packing two disks in a polygon. *Comput. Geom. Theory Appl.*, 23:31–42, 2002.

[7] B. K. Bhattacharya and H. ElGindy. A new linear convex hull algorithm for simple polygons. *IEEE Transactions on Information Theory*, IT-30:85–88, 1984.

[8] T. C. Biedl, E. D. Demaine, M. L. Demaine, A. Lubiw, and G. T. Toussaint. Hiding disks in folded polygons. In *Proc. 10th Canad. Conf. Comput. Geom.*, 1998.

[9] P. Bose, J. Czyzowicz, E. Kranakis, and A. Maheshwari. Algorithms for packing two circles in a convex polygon. In *Proc. Japan Conf. Discrete Comput. Geom. (JCDCG'98)*, Lect. Notes Comput. Sci., vol. 1763, Springer-Verlag, pages 93–103, 2000.

[10] P. Bose, P. Morin, and A. Vigneron. Packing two disks into a polygonal environment. *J. Discrete Algorithms*, 2:373–380, 2004.

[11] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete Comput. Geom.*, 22:547–567, 1999.

[12] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6:485–524, 1991.

[13] B. Chazelle, O. Devillers, F. Hurtado, M. Mora, V. Sacristán, and M. Teillaud. Splitting a Delaunay triangulation in linear time. *Algorithmica*, 34:39–46, 2002.

[14] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21:405–420, 1999.

[15] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[16] K. L. Clarkson, R. E. Tarjan, and C. J. Van Wyk. A fast Las Vegas algorithm for triangulating a simple polygon. *Discrete Comput. Geom.*, 4:423–432, 1989.

[17] O. Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *Int. J. Comput. Geom. Appl.*, 2:97–111, 1992.

[18] K. Y. Fung, T. M. Nicholl, R. E. Tarjan, and C. J. Van Wyk. Simplified linear-time Jordan sorting and polygon clipping. *Inform. Process. Lett.*, 35:85–92, 1990.

[19] M. T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Sys. Sci.*, 51:374–389, 1995.

[20] R. L. Graham and F. F. Yao. Finding the convex hull of a simple polygon. *J. Algorithms*, 4:324–331, 1983.

[21] J. Hershberger and J. Snoeyink. Cartographic line simplification and polygon CSG formula in $O(n \log^* n)$ time. *Comput. Geom. Theory Appl.*, 11:175–185, 1998.

[22] K. Hoffmann, K. Mehlhorn, P. Rosenstiehl, and R. E. Tarjan. Sorting Jordan sequences in linear time using level-linked search trees. *Inform. Control*, 68:170–184, 1986.

[23] S. K. Kim, C.-S. Shin, and T.-C. Yang. Placing two disks in a convex polygon. *Inform. Process. Lett.*, 73:33–39, 2000.

[24] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12:28–35, 1983.

[25] D. G. Kirkpatrick, M. Klawe, and R. E. Tarjan. Polygon triangulation in $O(n \log \log n)$ time with simple data structures. *Discrete Comput. Geom.*, 7:329–346, 1992.

[26] R. Klein and A. Lingas. Fast skeleton construction. In *Proc. 3rd European Sympos. Algorithms*, Lect. Notes Comput. Sci., vol. 979, Springer-Verlag, pages 582–595, 1995.

[27] D. T. Lee. On finding the convex hull of a simple polygon. *Int. J. Comput. Inform. Sciences*, 12:87–98, 1983.

[28] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Applied Math.*, 32:177–189, 1979.

[29] D. McCallum and D. Avis. A linear algorithm for finding the convex hull of a simple polygon. *Inform. Process. Lett.*, 9:201–206, 1979.

[30] N. Megiddo. Applying parallel computation algorithms to the design of serial algorithms. *J. ACM*, 30:852–865, 1983.

[31] A. Melkman. On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.*, 25:11–12, 1987.

[32] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice-Hall, Englewood Cliffs, NJ, 1994.

[33] F. Nielsen and M. Yvinec. An output-sensitive convex hull algorithm for planar convex objects. *Int. J. Comput. Geom. Appl.*, 8:39–66, 1998.

[34] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[35] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51–64, 1991.

[36] R. E. Tarjan. *Data Structures and Network Algorithms.* SIAM, Philadelphia, PA, 1983.