Classifying a Point Set for a Convex Body with Few Queries

Sariel Har-Peled[*] Mitchell Jones[†] Saladi Rahul[‡]

December 7, 2018

Abstract

Consider a set $P \subseteq \mathbb{R}^d$ of n points, and a convex body C provided via a separation oracle. The task at hand is to decide for each point of P if it is in C. We show that one can solve this problem in two and three dimensions using $O(\hexagon_P \log n)$ queries, where $\hexagon_P$ is the largest subset of points of P in convex position. Furthermore, we show that in two dimensions one can solve this problems using $O(\circledcirc(P, C) \log^2 n)$ oracle queries, where $\circledcirc(P, C)$ is the minimal number of queries that any algorithm for this specific instance requires.

1. Introduction

Problem statement. We are given a set P of n points in $\mathbb{R}^2$ and access to a convex body C via a separation oracle. Specifically, given a query point $q \in \mathbb{R}^2$, the oracle either reports that q is inside C, or if not, it returns a separating line ℓ such that q lies on one side of ℓ, and C lies on the other, see Figure 1.1. Our goal is to decide for every point $p \in P$ whether or not p is contained in C, see Figure 1.2, using as few oracle queries as possible. This question can be naturally extended to higher dimensions.

Motivation & Previous work.

(A) *Separation oracles.* The use of separation oracles is a common tool in optimization (e.g., solving exponentially large linear programs) and operations research. It is natural to ask what other problems can be solved efficiently when given access to this specific type of oracle.

(B) *Other types of oracles.* Various models of computation utilizing oracles has been previously studied within the community. Examples of other models include nearest-neighbor oracles (i.e., black-box access to nearest neighbor queries over a point set P) [HKMR16], and proximity probes (which given a convex polygon C and a query q, returns the distance from q to C) [PAvdSG13]. It is reasonable to ask what classification-type problems can be solved with few oracle queries when using separation oracles.

[*]Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; sariel@illinois.edu; http://sarielhp.org/. Work on this paper was partially supported by a NSF AF awards CCF-1421231, and CCF-1217462.

[†]Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; mfjones2@illinois.edu; http://mfjones2.web.engr.illinois.edu/.

[‡]Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; saladi.rahul@gmail.com; https://www-users.cs.umn.edu/~sala0198/.

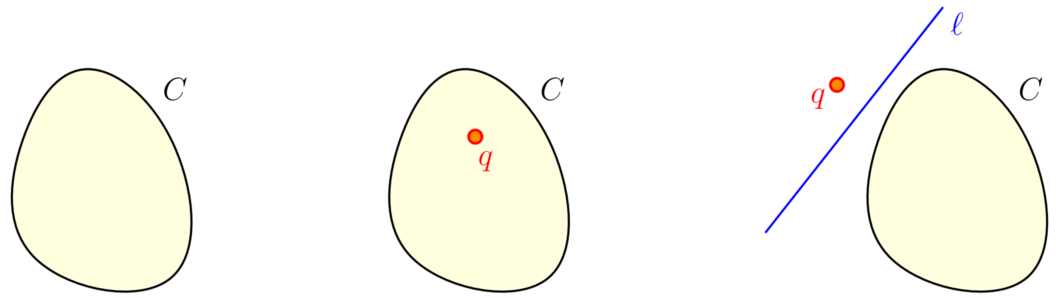


Figure 1.1: (I) A convex body C. (II) The query is inside C. (III) The oracle returns a line ℓ separating the query point and C.

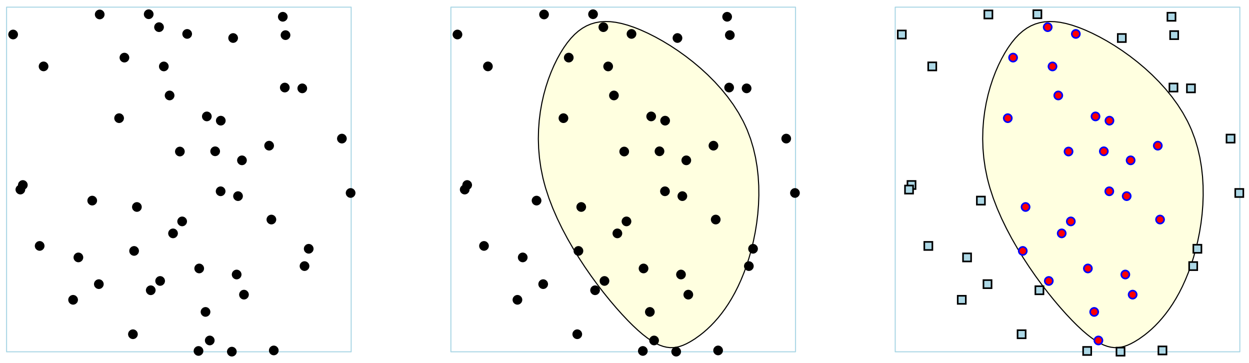


Figure 1.2

(C) *Active learning.* The problem at hand can be interpreted as active learning a convex body in relation to a set of points P that need to be classified (as either inside or outside the body), where the queries are via a separation oracle. We are unaware of any work directly on this problem in the theory community, while there is some work in the machine learning community that studies related problems (usually in more practical and realistic settings, and in higher dimensions, naturally).

Hard and easy instances. Note that in the worst case, an algorithm has to query the oracle of all input points — such a natural scenario happens when the input points are in convex position, and any possible subset can be the subset of points in the (appropriate) convex body. As such, the purpose here is to develop algorithms that are *instance sensitive* — if the given instance is easy, they work well. If the given instance is hard, they might deteriorate to the naive algorithm that queries all points.

Natural inputs where one can hope to do better, are when relatively few points are in convex position. Such inputs are grid points, or random point sets, among others. However, there are natural instances of the problem that are easy, despite the input having many points in convex position. For example, consider the convex body a triangle, with the input point set being $n/2$ points spread uniformly on a tiny circle around the origin, while the remaining $n/2$ points are outside the convex body, say spread uniformly on a circle of radius 10 centered at the origin, see Figure 1.3. Clearly, such a point set can be classified using a constant number of oracle queries. See Figure 3.1 for some related examples.

Our results.

(A) We develop a greedy algorithm, for points in the plane, which solves the problem using $O(\hexagon_P \log n)$ oracle queries, where $\hexagon_P$ is the largest subset of points of P in convex position. See Theorem 2.8. It is known that for a random set of n points in the unit square, $\hexagon_P = \Theta(n^{1/3})$, which readily

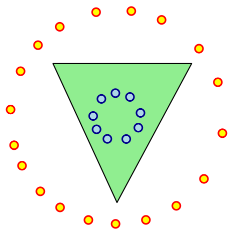


Figure 1.3

implies that classifying these points can be solved using $O(n^{1/3} \log n)$ oracle queries. A similar bound holds for the $\sqrt{n} \times \sqrt{n}$ grid. An animation of this algorithm can also be found on YouTube [HJR].

(B) The above algorithm naturally extends to three dimensions, also using $O(\hexagon_P \log n)$ oracle queries. While the proof idea is similar to that of the algorithm in 2D, we believe the analysis in three dimensions is also technically interesting. See Theorem 2.17.

(C) In Section 3 we present an improved algorithm for the 2D case, and show that the number of queries made is optimal up to a $O(\log^2 n)$ factor, see Theorem 3.5.

(D) We consider the extreme scenarios of the problem: Verifying that all points are either inside or outside of C. For each problem we present a $O(\log n)$ approximation algorithm to the optimal strategy. The results are presented in Appendix B, see Lemma B.2 and Lemma B.4.

2. The greedy classification algorithm in two and three dimensions

2.1. Preliminaries

For a set of points $P \subseteq \mathbb{R}^2$, let $\mathrm{ch}(P)$ denote the convex hull of P. Given a convex body $C \subseteq \mathbb{R}^d$, two points $p, x \in \mathbb{R}^d \setminus \mathrm{int}(C)$ are ***visible***, if the segment px does not intersect $\mathrm{int}(C)$, where $\mathrm{int}(C)$ is the interior of C. We also use the notation $P \cap C = \{p \in P \mid p \in C\}$.

For a point set $P \subseteq \mathbb{R}^d$, a ***centerpoint*** of P is a point $c \in \mathbb{R}^d$, such that for any closed halfspace h^+ containing c, we have $|h^+ \cap P| \geq |P| / (d+1)$. A centerpoint always exists, and it can be computed exactly in $O(n^{d-1} + n \log n)$ time [Cha04].

2.2. The greedy algorithm in 2D

2.2.1. Operations

Initially, the algorithm copies P into a set U of unclassified points. The algorithm is going to maintain an inner approximation $B \subseteq C$. There are two types of updates (Figure 2.1 illustrates the two operations):

(A) **expand**(p): Given a point $p \in C \setminus B$, the algorithm is going to:

  (i) Update the inner approximation: $B \leftarrow \mathrm{ch}(B \cup \{p\})$.
  (ii) Remove (and mark) newly covered points: $U \leftarrow U \setminus B$.

(B) **remove**(ℓ): Given a closed halfplane ℓ^+ such that $\mathrm{int}(C) \cap \ell^+ = \emptyset$, the algorithm marks all the points of $U_\ell = U \cap \mathrm{int}(\ell^+)$ as being outside C, and sets $U \leftarrow U \setminus U_\ell$.

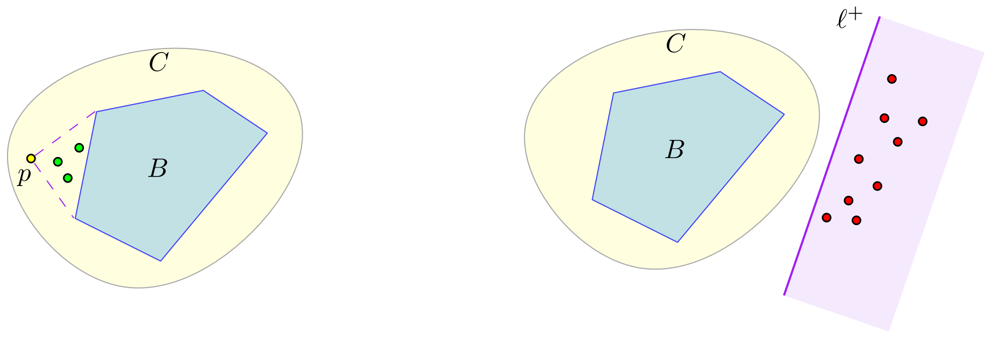


Figure 2.1: (I) Performing **expand**(p), and marking points inside C. (II) Performing **remove**$(\boldsymbol{\ell})$, and marking points outside C.

2.2.2. The algorithm

The algorithm repeatedly performs rounds, as described next, until the set of unclassified points is empty.

At every round, if the inner approximation B is empty, then the algorithm sets $U^+ = U$. Otherwise, the algorithm picks a line $\boldsymbol{\ell}$ that is tangent to B with the largest number of points of U on the other side of $\boldsymbol{\ell}$ than B. Let $\boldsymbol{\ell}^-$ and $\boldsymbol{\ell}^+$ be the two closed halfspace bounded by $\boldsymbol{\ell}$, where $B \subseteq \boldsymbol{\ell}^-$. The algorithm computes the point set $U^+ = U \cap \boldsymbol{\ell}^+$. We have two cases:

A. If $|U^+| = O(1)$, then the algorithm queries the oracle for the status of each of these points. For every point $p \in U^+$, such that $p \in C$, the algorithm performs **expand**(p). Otherwise, the oracle returned a separating line $\boldsymbol{\ell}$, and the algorithm calls **remove**$(\boldsymbol{\ell}^+)$.

B. Otherwise, the algorithm computes a centerpoint $\mathpzc{c} \in \mathbb{R}^2$ for U^+, and asks the oracle for the status of $\mathpzc{c}$. There are two possibilities:

   B.I. If $\mathpzc{c} \in C$, then the algorithm performs **expand**$(\mathpzc{c})$.

   B.II. If $\mathpzc{c} \notin C$, then the oracle returned a separating line $\mathpzc{h}$, and the algorithm performs **remove**$(\mathpzc{h})$.

2.2.3. Analysis

Let B_i be the inner approximation at the start of the ith iteration, and let z be the first index where B_z is not an empty set. Similarly, let U_i be the set of unclassified points at the start of the ith iteration, where initially $U_1 = U$.

Lemma 2.1. *The number of (initial) iterations in which the inner approximation is empty is* $z = O(\log n)$.

Proof: As soon as the oracle returns a point that is in C, the inner approximation is no longer empty. As such, we need to bound the initial number of iterations where the oracle returns that the query point is outside C. Let $n_i = |U_i|$, and note that $U_1 = P$ and $n_1 = |P| = n$. Let $\mathpzc{c}_i$ be the centerpoint of U_i, which is the query point in the ith iteration ($\mathpzc{c}_i$ is outside C). As such, the line separating $\mathpzc{c}_i$ from C, returned by the oracle, has at least $n_i/3$ points of U_i on the same side as $\mathpzc{c}_i$, by the centerpoint property. All of these points get labeled in this iteration, and it follows that $n_{i+1} \leq (2/3)n_i$, which readily implies the claim, since $n_z < 1$, for $z = \lceil \log_{3/2} n \rceil + 1$. ■

Definition 2.2 (Visibility graph). Consider the graph G_i over U_i, where two points $p, r \in U_i$ are connected $\iff$ the segment pr does not intersect the interior of B_i.

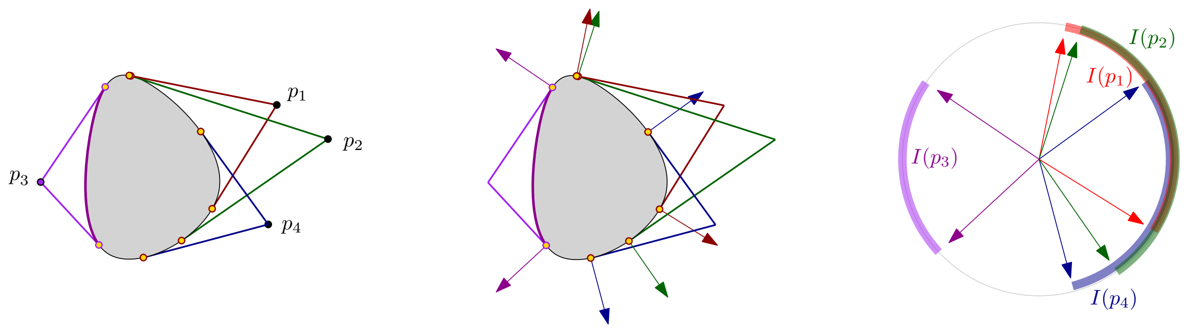


Figure 2.2

The visibility graph as an interval graph. For a point $p \in U_i$, let $I_i(p)$ be the set of all directions v (i.e., vectors of length 1) such that there is a line perpendicular to v that separates p from B_i. Formally, a line ℓ separates p from B_i, if the interior of B_i is on one side of ℓ and p is on the (closed) other side of ℓ (if $p \in \ell$, the line is still considered to separate the two). Clearly, $I_i(p)$ is a circular interval on the unit circle. See Figure 2.2. The resulting set of intervals is $\mathcal{V}_i = \{I_i(p) \mid p \in U_i\}$. It is easy to verify that the intersection graph of $\mathcal{V}_i$ is G_i. Throughout the execution of the algorithm, the inner approximation B_i grows monotonically, this in turn implies that the visibility intervals shrinks over time; that is, $I_i(p) \subseteq I_{i-1}(p)$, for all $p \in P$ and i. Intuitively, in each round, either many edges from G_i are removed (because intervals had shrunk and they no longer intersect), or many vertices are removed (i.e., the associated points are labeled).

Definition 2.3. Given a set $\mathcal{V}$ of objects (e.g., intervals) in a domain D (e.g., unit circle), the ***depth*** of a point $p \in D$, is the number of objects in $\mathcal{V}$ that contains p. Let $\operatorname{depth}(\mathcal{V})$ be the maximum depth of any point in D.

When it is clear, we use $\operatorname{depth}(G)$ to denote $\operatorname{depth}(\mathcal{V})$, where $G = (\mathcal{V}, E)$ is the intersection graph in Definition 2.2. We need the following well known lemma — a proof is provided for the sake of completeness.

Lemma 2.4. *Let $\mathcal{V}$ be a set of n intervals on the unit circle, and let $G = (\mathcal{V}, E)$ be the associated intersection graph. Then $|E| = \Theta(\alpha\omega^2)$, where $\omega = \operatorname{depth}(\mathcal{V})$ and $\alpha = \alpha(G)$ is the size of the largest independent set in G.*

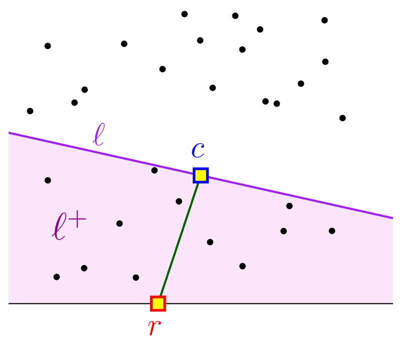
Proof: Let J be the largest independent set of intervals in G. The intervals of J divide the circle into $2|J|$ (atomic) circular arcs. Consider such an arc γ, and let $K(\gamma)$ be the set of all intervals of $\mathcal{V}$ that are fully contained in γ. All the intervals of $K(\gamma)$ are pairwise intersecting, as otherwise one could increase the size of the independent set. As such, all the intervals of $K(\gamma)$ must contain a common intersection point. It follows that $|K(\gamma)| \le \omega$.

Let $K'(\gamma)$ be the set of all intervals intersecting γ. This set might contain up to 2ω additional intervals (that are not contained in γ), as each such additional interval must contain at least one of the endpoints of γ. Namely, $|K'(\gamma)| \le 3\omega$. In particular, any two intervals intersecting inside γ both belong to $K'(\gamma)$. As such, the total number of edges contributed by $K'(\gamma)$ to G is at most $\binom{3\omega}{2} = O(\omega^2)$. Since there are $\leq 2\alpha$ arcs under consideration, the total number of edges in G is bounded by $O(\alpha\omega^2)$, which implies the claim.

The lower bound is easy to see by taking an independent set of intervals of size α, and replicating every interval ω times. ■

Lemma 2.5. *Let P be a set of n points in the plane lying above the x-axis, $\mathcal{c}$ be the centerpoint of P, and $S = \binom{P}{2}$ be set of all segments induced by P. Next, consider any point r on the x-axis. Then, the segment $\mathcal{c}r$ intersects at least $n^2/36$ segments of S.*

Proof: If the segment $\mathcal{c}r$ intersects the segment p_1p_2, for $p_1, p_2 \in P$, then we consider p_1 to no longer be visible from p_2 (and vice versa). It suffices to lower bound the number of pairs of points which lose visibility of each other.



Consider a line ℓ passing through the point $\mathcal{c}$. Let ℓ^+ be the closed halfspace bounded by ℓ containing r. Note that $|P \cap \ell^+| \geq n/3$, since $\mathcal{c}$ is a centerpoint of P, and $\mathcal{c} \in \ell$. Rotate ℓ around $\mathcal{c}$ until there are $\geq n/6$ points on each side of $r\mathcal{c}$ in the halfspace ℓ^+. To see why this rotation of ℓ exists, observe that the two halfspaces bounded by the line spanning $r\mathcal{c}$, have zero points on one side, and at least $n/3$ points on the other side — a continuous rotation of ℓ between these two extremes, implies the desired property.

Observe that points in ℓ^+ and on opposite sides of the segment $\mathcal{c}r$ cannot see each other, as the segment connecting them must intersect $\mathcal{c}r$. Consequently, the number of induced segments that $\mathcal{c}r$ intersects is at least $n^2/36$. ■

Lemma 2.6. *Let G_i be the intersection graph, in the beginning of the ith iteration, and let $m_i = |E(G_i)|$. After the ith iteration of the greedy algorithm, we have $m_{i+1} \leq m_i - \omega^2/36$, where $\omega = \mathrm{depth}(G_i)$.*

Proof: Recall that in the algorithm $U^+ = U_i \cap \ell^+$ is the current set of unclassified points and ℓ is the line tangent to B_i, where ℓ^+ is the closed halfspace that avoids the interior of B_i and contains the largest number of unlabeled points of U_i. We have that $\omega = |U^+|$.

If a **remove** operation was performed in the ith iteration, then the number of points of U^+ which are discarded is at least $\omega/3$. In this case, the oracle returned a separating line $\hbar$ between a centerpoint $\mathcal{c}$ of U^+ and the inner approximation. For the halfspace $\hbar^+$ containing $\mathcal{c}$, we have $t_i = |U^+ \cap \hbar^+| \geq |U^+|/3 \geq \omega/3$. Furthermore, all the points of U^+ are pairwise visible (in relation to the inner approximation B_i). Namely, $m_{i+1} = |E(G_i - (U^+ \cap \hbar^+))| \leq m_i - \binom{t_i}{2} \leq m_i - \omega^2/36$.

If an **expand** operation was performed, the centerpoint $\mathcal{c}$ of U^+ is added to the current inner approximation B_i. Let r be a point in $\ell \cap B_i$, and let $\mathcal{c}_i$ be the center point of U_i computed by the algorithm. By Lemma 2.5 applied to $r, \mathcal{c}$ and U^+, we have that at least $\omega^2/36$ pairs of points of U^+ are no longer visible to each other in relation to B_{i+1}. We conclude, that at least $\omega^2/36$ edges of G_i are no longer present in G_{i+1}. ■

Definition 2.7. A subset of points $X \subseteq P \subseteq \mathbb{R}^2$ are in ***convex position***, if all the points of X are vertices of $\mathrm{ch}(X)$ (note that a point in the middle of an edge is not considered to be a vertex). The ***index*** of P, denoted by $\hexagon_P$, is the cardinality of the largest subset of P of points which are in convex position.

Theorem 2.8. *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. The greedy classification algorithm performs $O\big((\hexagon_P + 1)\log n\big)$ oracle queries. The algorithm correctly identifies all points in $P \cap C$ and $P \setminus C$.*

Proof: By Lemma 2.1, the number of iterations (and also queries) in which the inner approximation is empty is $O(\log n)$, and let $z = O(\log n)$ be the first iteration such that the inner approximation is not empty. It suffices to bound the number of queries made by the algorithm after the inner approximation becomes non-empty.

For $i \geq z$, let $G_i = (U_i, E_i)$ denote the visibility graph of the remaining unclassified points U_i in the beginning of the ith iteration. Any independent set in G_i corresponds to a set of points $X \subseteq P$ that do not see each other due to the presence of the inner approximation B_i. That is, X is in convex position, and furthermore $|X| \leq \hexagon_P$.

For $0 \leq t \leq n$, let $s(t)$ be the first iteration i, such that $\mathrm{depth}(G_i) \leq t$. Since the depth of G_i is a monotone decreasing function, this quantity is well defined. An ***epoch*** is a range of iterations between $s(t)$ and $s(t/2)$, for any parameter t. We claim that an epoch lasts $O(\hexagon_P)$ iterations (and every iteration issues only one oracle query). Since there are only $O(\log n)$ (non-overlapping) epochs till the algorithm terminates, as the depth becomes zero, this implies the claim.

So consider such an epoch starting at $i = s(t)$. We have $m = m_i = |E(G_i)| = O(\hexagon_P t^2)$, by Lemma 2.4, since $\hexagon_P$ is an upper bound on the size of the largest independent set in G_i. By Lemma 2.6, as long as the depth of the intervals is at least $t/2$, the number of edges removed from the graph at each iteration, during this epoch, is at least $\Omega(t^2)$. As such, the algorithm performs at most $O(m_i/t^2) = O(\hexagon_P)$ iterations in this epoch, till the maximum depth drops to $t/2$. ■

2.3. The greedy algorithm in 3D

2.3.1. The algorithm

Consider the 3D variant of the 2D problem: Given a set of points P in $\mathbb{R}^3$ and a convex body C specified via a separation oracle, the task at hand is to classify, for all the points of P, whether or not they are in C, using the fewest oracle queries possible.

The greedy algorithm naturally extends, where at each iteration i a plane e_i is chosen that is tangent to the current inner approximation B_i, such that it's closed halfspace (which avoids the interior of B_i) contains the largest number of unclassified points from the set U_i. If the queried centerpoint is outside, the oracle returns a separating plane and as such points can be discarded by the **remove** operation. Similarly, if the centerpoint is reported inside, then the algorithm calls the **expand** and updates the 3D inner approximation B_i.

2.3.2. Analysis

Following the analysis of the greedy algorithm in 2D, we (conceptually) maintain the following set of objects: For a point $p \in U_i$, let $d_i(p)$ be the set of all unit length directions $v \in \mathbb{R}^3$ such that a plane perpendicular to v separates p from B_i. Let $\mathcal{P}_i = \{d_i(p) \mid p \in U_i\}$. A set of objects form a collection of

pseudo-disks if the boundary of every pair of them intersect at most twice. The following claim shows that $\mathcal{P}_i$ is a collection of pseudo-disks on $\mathbb{S}$, where $\mathbb{S}$ is the sphere of radius one centered at the origin.

Lemma 2.9. *The set $\mathcal{P}_i = \{ d_i(p) \subseteq \mathbb{S} \mid p \in U_i \}$ is a collection of pseudo-disks.*

Proof: Fix two points $p, r \in U_i$ such that the boundaries of $d_i(p)$ and $d_i(r)$ intersect on $\mathbb{S}$. Let ℓ be the line in $\mathbb{R}^3$ passing through p and r. Consider any plane e such that ℓ lies on e. Since ℓ is fixed, e has one degree of freedom. Conceptually rotate e until becomes tangent to B_i at point u'. The direction of the normal to this tangent plane, is a point in $X = \partial d_i(p) \cap \partial d_i(r)$. Note that this works also in the other direction — any point in X corresponds to a tangent plane passing through ℓ. The family of planes passing through ℓ has only two tangent planes to C. It follows that $|X| = 2$. As such, any two regions in $\mathcal{P}_i$ intersect as pseudo-disks. ■

We need the following two classical results that follows from the Clarkson-Shor [CS89] technique — for the sake of completeness the proofs are provided in the appendix.

Lemma 2.10 (Proof in Appendix A.1)**.** *Let $\mathcal{P}$ be a collection of n pseudo-disks, and let $V_{\leq k}(\mathcal{A})$ be the set of all vertices of depth at most k in the arrangement $\mathcal{A} = \mathcal{A}(\mathcal{P})$. Then $|V_{\leq k}(\mathcal{A})| = O(nk)$.*

Lemma 2.11 (Proof in Appendix A.2)**.** *Let $\mathcal{P}$ be a collection of n pseudo-disks. For two integers $0 < t \leq k$, a subset $X \subseteq \mathcal{P}$ is a* (t, k)***-tuple*** *if (i) $|X| \leq t$, (ii) $\exists p \in \cap_{d \in X} d$, and (iii) $\mathrm{depth}(p, \mathcal{P}) \leq k$. Let $L(t, k, n)$ be the set of all $(\leq t, k)$-tuples of $\mathcal{P}$. Then $|L(t, k, n)| = O(ntk^{t-1})$.*

Lemma 2.12. *Let $G_i = (\mathcal{P}_i, E_i)$ be the intersection graph of the pseudo-disks of $\mathcal{P}_i$ (in the ith iteration). If $\mathcal{A}(\mathcal{P}_i)$ has maximum depth k, then $|E_i| = O(nk)$. Furthermore, $\alpha(G_i) = \Omega(n/k)$, where $\alpha(G_i)$ denotes the size of the largest independent set in G_i.*

Proof: The first claim readily follows from Lemma 2.11 — indeed, $|E_i| = L(2, k, n) = O(nk)$ — since every intersecting pair of pseudo-disks induces a corresponding $(2, k)$-tuple.

For the second part, Turán's Theorem states that any graph has an independent set of size at least $n/(d_{\mathrm{avg}}(G_i) + 1)$, where $d_{\mathrm{avg}}(G_i) = 2|E_i|/n \leq ck$ is the average degree of G_i and c is some constant. It follows that $\alpha(G_i) \geq n/(ck + 1) = \Omega(n/k)$. ■

The challenge in analyzing the greedy algorithm in 3D is that visibility between pairs of points is not necessarily lost as the inner approximation grows. As an alternative, consider the *hypergraph* $H_i = (\mathcal{P}_i, \mathcal{E}_i)$, where a triple of pseudo-disks $d_1, d_2, d_3 \in \mathcal{P}_i$ form a hyperedge $\{d_1, d_2, d_3\} \in \mathcal{E}_i \iff d_1 \cap d_2 \cap d_3 \neq \varnothing$ (this is equivalent to the condition that the corresponding triple of points span a triangle which does not intersect B_i).

Lemma 2.13. *Let $H_i = (\mathcal{P}_i, \mathcal{E}_i)$ be the hypergraph in iteration i, and let G_i be the corresponding intersection graph of $\mathcal{P}_i$. If $\mathcal{A}(\mathcal{P}_i)$ has maximum depth k, then $|\mathcal{E}_i| = O(\alpha(G_i)k^3)$.*

Proof: Lemma 2.12 implies that G_i has an independent set of size $\Omega(n_i/k)$, where $n_i = |\mathcal{P}_i|$. Lemma 2.11 implies that $|\mathcal{E}_i| \leq |L(3, k, n_i)| = O(n_i k^2) = O(\alpha(G_i)k^3)$. ■

The following is a consequence of the Colorful Carathéodory Theorem [Bár82], see Theorem 9.1.1 in [Mat02].

Theorem 2.14. *Let P be a set of n points in $\mathbb{R}^d$ and c be the centerpoint of P. Let $S = \binom{P}{d+1}$ be the set of all $d + 1$ simplices induced by P. Then for sufficiently large n, the number of simplices in S that contain c in their interior is at least $c_d n^{d+1}$, where c_d is a constant depending only on d.*

Next, we argue that in each iteration of the greedy algorithm, a constant fraction of the edges in H_i are removed. The following is the higher dimensional version of Lemma 2.5.

Lemma 2.15. *Let P be a set of n points in $\mathbb{R}^3$ lying above the xy-plane, c be the centerpoint of P and $T = \binom{P}{3}$ be the set of all triangles induced by P. Next, consider any point r on the xy-plane. Then the segment cr intersects at least $\Omega(n^3)$ triangles of T.*

Proof: Let $S = \binom{P}{d+1}$ be the set of all simplices induced by P. Theorem 2.14 implies that the centerpoint c is contained in n^4/c_1 simplices of S for some constant $c_1 > 1$. Let K be a simplex that contains c and observe the segment cr must intersect at least one of the triangular faces τ of K. As $K \in S$, charge this simplex K to the triangular face τ. Applying this counting to all the simplices containing c, implies that at least n^4/c_1 charges are made. On the other hand, a triangle τ can be charged at most $n-3$ times (because a simplex can be formed from τ and one other additional point of P). It follows that cr intersects at least $(n^4/c_1)/(n-3) = \Omega(n^3)$ triangles of T. ■

Lemma 2.16. *In each iteration of the greedy algorithm, the number of edges in the hypergraph $H_i = (\mathcal{P}_i, \mathcal{E}_i)$ decreases by at least $\Omega(k^3)$, where k is the maximum depth of any point in $\mathcal{A}(\mathcal{P}_i)$.*

Proof: Recall that $U^+ = U_i \cap e^+$ is the current set of unclassified points and e is the plane tangent to B_i, where e^+ is the closed halfspace that avoids the interior of B_i and contains the largest number of unlabeled points. Note that $|U^+| \geq k$.

In a **remove** operation, arguing as in Lemma 2.6, implies that the number of points of U^+ which are discarded is at least $k/4$. Since all of the discarded points are in a halfspace avoiding B_i, it follows that all the triples they induce are in H_i. Namely, at least $\binom{k/4}{3} = \Omega(k^3)$ hyperedges get discarded.

In an **expand** operation, the centerpoint c of U^+ is added to the current inner approximation B_i. Since all of the points of U^+ lie above the plane e, applying Lemma 2.15 on U^+ with the centerpoint c and a point lying on the plane e inside the (updated) inner approximation, we deduce that at least $\Omega(k^3)$ hyperedges are removed. ■

Theorem 2.17. *Let $C \subseteq \mathbb{R}^3$ be a convex body provided via a separation oracle, and let P be a set of n points in $\mathbb{R}^3$. The greedy classification algorithm performs $O\big((\hexagon_P + 1)\log n\big)$ oracle queries. The algorithm correctly identifies all points in $P \cap C$ and $P \setminus C$.*

Proof: The proof is essentially the same as Theorem 2.8. Arguing as in Lemma 2.1 implies that there are at most $O(\log n)$ iterations (and thus also oracle queries) in which the inner approximation is empty.

Now consider the hypergraph $H_1 = (\mathcal{P}_1, \mathcal{E}_1)$ at the start of the algorithm execution. As the algorithm progresses, both vertices and hyperedges are removed from the hypergraph. Let $H_i = (\mathcal{P}_i, \mathcal{E}_i)$ denote the hypergraph in the ith iteration of the algorithm. Recall that $\mathcal{P}_i$ is a set of pseudo-disks associated with each of the points yet to be classified. Observe that any independent set of pseudo-disks in the corresponding intersection graph G_i corresponds to an independent set of points with respect to the inner approximation B_i, and as such is a subset of points in convex position. Therefore, the size of any such independent set is bounded by $\hexagon_P$.

Let k_i denote the maximum depth of any vertex in the arrangement $\mathcal{A}(\mathcal{P}_i)$. Lemma 2.13 implies that $|\mathcal{E}_i| = O(\hexagon_P k_i^3)$. Lemma 2.16 implies that the number of hyperedges in the ith iteration decreases by at least $\Omega(k_i^3)$. Namely, after $O(\hexagon_P)$ iterations, the maximum depth is halved. It follows that after $O(\hexagon_P \log n)$ iterations, the maximum depth is zero, which implies that all the points are classified. Since the algorithm performs one query per iteration, the claim follows. ■

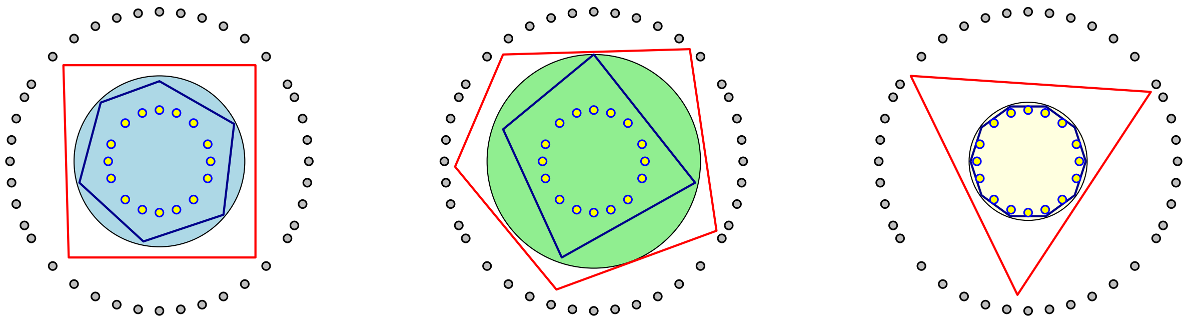


Figure 3.1: The separation price, for the same point set, is different depending on how "tight" the body is in relation to the inner and outer point set.

3. An improved instance-optimal approximation in two dimensions

Before presenting the improved algorithm, we present a lower bound on the number of oracle queries performed by any algorithm that classifies all the given points. We then present an algorithm which matches the lower bound up to a factor of $O(\log^2 n)$.

3.1. A lower bound

Given a set P of points in the plane, and a convex body C, the ***outer fence*** of P is a closed convex polygon F_{out} with minimum number of vertices, such that $C \subseteq F_{\mathrm{out}}$ and $C \cap P = F_{\mathrm{out}} \cap P$. Similarly, the ***inner fence*** is a closed convex polygon F_{in} with minimum number of vertices, such that $F_{\mathrm{in}} \subseteq C$ and $C \cap P = F_{\mathrm{in}} \cap P$. Intuitively, the outer fence separates $P \setminus C$ from ∂C, while the inner fence separates $P \cap C$ from ∂C. The ***separation price*** of P and C is

$$\circledcirc(P, C) = |F_{\mathrm{in}}| + |F_{\mathrm{out}}|,$$

where $|F|$ denotes the number of vertices of a polygon F. See Figure 3.1 for an example.

Lemma 3.1. *Given a point set P and a convex body C in the plane, any algorithm that classifies the points of P in relation to C, must perform at least $\circledcirc(P, C)$ separation oracle queries.*

Proof: Consider the set Q of queries performed by the optimal algorithm (for this input), and split it, into the points inside and outside C. The set of points inside Q_{in} has the property that $Q_{\mathrm{in}} \subseteq C$, and furthermore $\mathrm{ch}(Q_{\mathrm{in}}) \cap P = C \cap P$ — otherwise, there would be a point of $C \cap P$ that is not classified. Namely, the vertices of $\mathrm{ch}(Q_{\mathrm{in}})$ are vertices of a fence that separates the points of P inside C from the boundary of C. As such, we have that $|Q_{\mathrm{in}}| \geq |\mathrm{ch}(Q_{\mathrm{in}})| \geq |F_{\mathrm{in}}|$.

Similarly, each query in Q_{out} gives rise to a separating halfplane. The intersection of the corresponding halfplanes is a convex polygon H which contains C, and furthermore contains no point of $P \setminus C$. Namely, the boundary of H behaves like an outer fence. As such, we have $|Q_{\mathrm{out}}| \geq |H| \geq |F_{\mathrm{out}}|$.

Combining, we have that $|Q| = |Q_{\mathrm{in}}| + |Q_{\mathrm{out}}| \geq |F_{\mathrm{in}}| + |F_{\mathrm{out}}| = \circledcirc(P, C)$, as claimed. ■

3.2. Useful operations

We start by presenting some basic operations that the new algorithm will use.

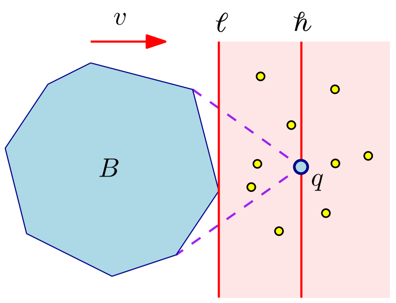


Figure 3.2: A directional climb. An iteration is done using the line ℓ. After updating B to include the query q, the algorithm chooses a new extreme line $\hbar$ tangent to B in the direction of v.

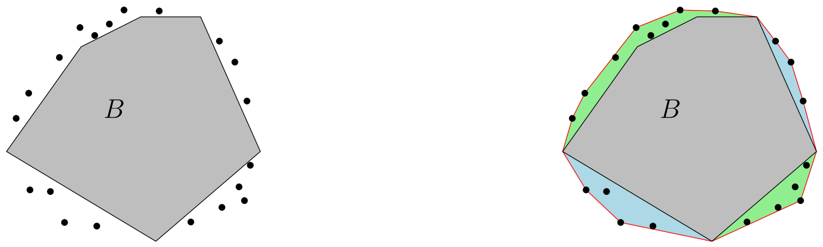


Figure 3.3: Unclassified points and their pockets.

3.2.1. A directional climb

Given a direction v, a ***directional climb*** is a sequence of iterations, where in each iteration, the algorithm finds the extreme line perpendicular to v, that is tangent to the inner approximation B. The algorithm then performs an iteration with this line, as described in Section 2.2.2. See Figure 3.2 for an illustration. The directional climb ends when the outer halfspace induced by this line contains no unclassified point.

Claim 3.2. *A directional claim this requires $O(\log n)$ oracle queries.*

Proof: Consider the tangent to B in the direction of v. At each iteration, we claim the number of points in this halfplane is reduced by a factor of $1/3$. Indeed, if the query (i.e., centerpoint) is outside C then at least a third of these points got classified as being outside. Alternatively, the tangent halfplanes moves in the direction of v, since the query point is inside C. But then the new halfspace contains at most $2/3$ fraction of the previous point set — again, by the centerpoint property. ■

3.2.2. Line cleaning

A ***pocket*** is a connected region of $\mathrm{ch}(U \cup B) \setminus B$, see Figure 3.3. For the set P of input points, consider the set of all lines

$$L(P) = \{\mathrm{line}(p, r) \mid p, r \in P\} \tag{3.1}$$

they span. Let ℓ be a line that splits a pocket Υ into two regions, and furthermore, it intersects B. Let $I = \ell \cap \Upsilon$, and consider all the intersection points of interest along I in this pocket. That is,

$$\Xi(\Upsilon, \ell, P) = I \cap L(P) = \left\{ (\Upsilon \cap \ell) \cap \hbar \;\middle|\; \hbar \in L(P) \right\}.$$

In words, we take all the pairs of points of P (each such pair induces a line) and we compute the intersection points of these lines with the interval I of interest. Ordering the points of this set along

ℓ, a prefix of them is in C, while the corresponding suffix are all outside C. One can easily compute this prefix/suffix by doing a binary search, using the separation oracle for C — see the lemma below for details. Each answer received from the oracle is used to update the point set, using **expand** or **remove** operations, as described in Section 2.2.1. We refer to this operation along ℓ as ***cleaning*** the line ℓ. See Figure 3.4.

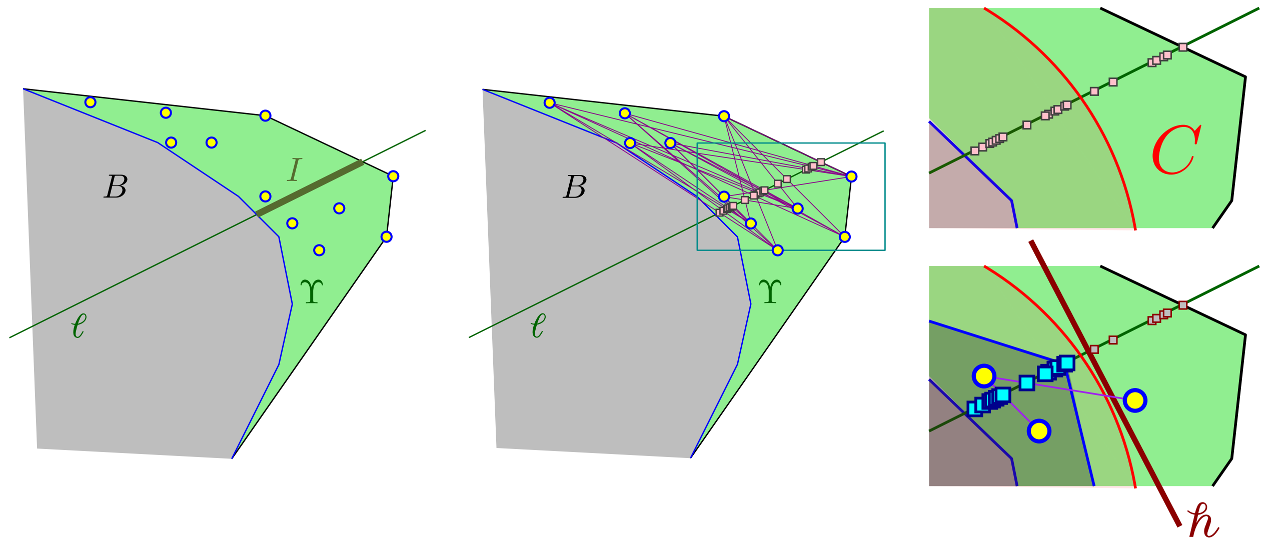


Figure 3.4: Line cleaning. All the intersection points of interest along ℓ are classified. The binary search results in the oracle returning a line $\hbar$ that separates the points outside from the points inside.

Lemma 3.3. *Given a pocket Υ, and a splitting line ℓ, one can clean the line ℓ — that is, classify all the points of $\Xi = \Xi(\Upsilon, \ell, P)$ using $O(\log n)$ oracle queries. By the end of this process, Υ is replaced by two pockets, Υ_1 and Υ_2 that do not intersect ℓ. The pockets Υ_1 or Υ_2 may be empty sets.*

Proof: First, we describe the line cleaning procedure in more detail. The algorithm maintains, in the beginning of the ith iteration, an interval J_i on the line ℓ containing all the points of Ξ that are not classified yet. Initially, $J_1 = \Upsilon \cap \ell$. One endpoint, say $p_i \in J_i$ is on ∂B_i, and the other, say p'_i, is outside C, where B_i is the inner approximation in the beginning of the ith iteration.

In the ith iteration, the algorithm computes the set $\Xi_i = J_i \cap \Xi$. If this set is empty, then the algorithm is done. Otherwise, it picks the median point u_i, in the order along ℓ in Ξ_i, and queries the oracle with u_i. There are two possibilities:

(A) If $u_i \in C$ then the algorithm sets $\Xi_{i+1} = \Xi_i \setminus [p_i, u_i)$, and $J_{i+1} = J_i \setminus [p_i, u_i)$.

(B) If $u_i \notin C$, then the oracle provided a closed halfspace h^+ that contains C. Let h^- be the complement open halfspace that contains u_i. The algorithm sets $\Xi_{i+1} = \Xi_i \setminus h^-$ and $J_{i+1} = J_i \cap h^+$.

This resolves the status of at least half the points in Ξ_i, and shrinks the active interval. The algorithm repeats this till Ξ_i becomes empty. Since $|\Xi| = O(n^2)$, this readily implies that the algorithm performs $O(\log n)$ iterations.

We now argue that the pocket is split — that is, Υ_1 and Υ_2 do not intersect ℓ. Assume that it is false, and let B' be the inner approximation after this procedure is done. Let L (resp. R) be all the points of $U_\Upsilon = U \cap \Upsilon$ that are unclassified yet on one side (resp. other side) of ℓ. If the pocket is not split, then there are two points $p \in L$ and $r \in R$, such that $pr \cap B' = \emptyset$, and $\partial \mathrm{ch}(B' \cup L \cup R)$ intersects ℓ at the point $u = pr \cap \ell$. However, by construction, the point $u \in \Xi$. As such, the point

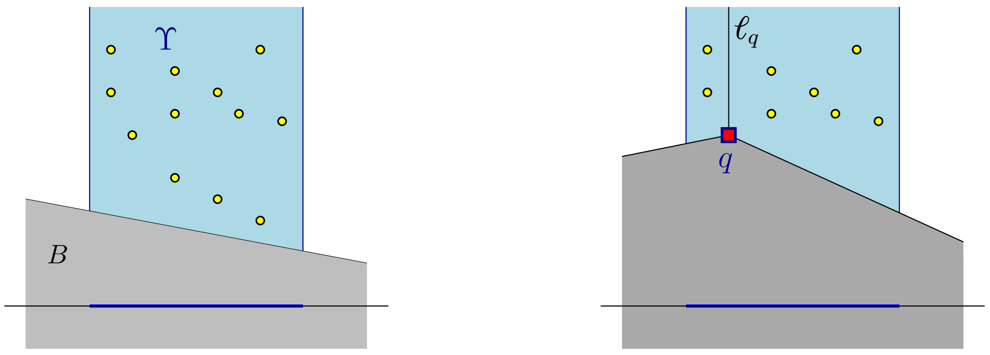


Figure 3.5: Vertical pocket splitting. The figure on the right is somewhat misleading — none of the unclassified points in the new pockets are visible to each other after the line cleaning operation was done on the separating line.

u is now classified as either being inside or outside C, as it is a point in Ξ. If u is outside, then the halfplane h^- that classified it as such, must had classified either p or r as being outside C, which is a contradiction. The other option, is that u is classified as being inside, but then, it is in B', which is again a contradiction, as it implies that B' intersects the segment pr. ■

3.2.3. Vertical pocket splitting

Consider a pocket Υ such that all of its points lie vertically above B, and the bottom of Υ is part of a segment of ∂B, see Figure 3.5. Such a pocket can be viewed as being defined by an interval on the x-axis corresponding to its two vertical walls. Let U_Υ be the set of unclassified points in this pocket. In each iteration, the algorithm computes the center point of U_Υ, and queries the separation oracle. As long as the query point is outside C, the algorithm performs a **remove** operation using the returned separating line.

When the oracle returns that the query point q is inside C, the algorithm computes the vertical line ℓ_q through q. The algorithm now performs line cleaning on this vertical line. This operation splits Υ into two sub-pockets. Crucially, since q was a centerpoint for U_Υ, the number of points in each of the two sub-pockets is at most $(2/3)\,|U_\Upsilon|$. See Figure 3.5.

3.3. The algorithm

The algorithm starts in the same way as the greedy algorithm, till it gets a non-empty inner approximation. The algorithm also maintains the convex hull of the unclassified points together with the inner approximation.

Next, the algorithm performs two directional climbs in the positive and negative directions of the x-axis. This uses $O(\log n)$ oracle queries. This results in a computed segment $vv' \subseteq C$, where v, v' are vertices of the inner approximation B at this stage, such that all unclassified points lie in the vertical strip induced by these two points.

The algorithm now handles all points of U lying above vv' (the points below the line would be handled in a similar fashion). Let B^+ be the set of vertices of B in the top chain. Note that B^+ consists of at most $O(\log n)$ vertices. For each vertex v of B^+, the algorithm performs line cleaning of the vertical line going through v. This results in $O(\log n)$ vertical pockets, where all vertical lines passing originally through B^+ are now clean.

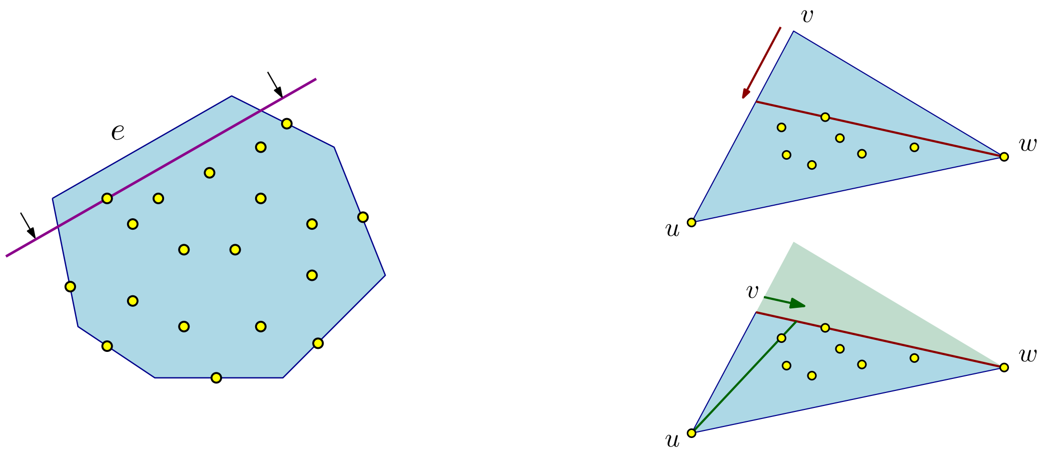


Figure 3.6: Constructing the polygon π from an inner fence σ.

The algorithm repeatedly picks a vertical pocket. If the pocket contains less than, say, three points the algorithm queries the oracle about these points, and continues to the next pocket. Otherwise, the algorithm performs a vertical pocket splitting on it, as described in Section 3.2.3. The algorithm stops when there are no longer any pockets (i.e., all the points above the segment vv' are classified). The algorithm then runs the symmetric procedure below this segment vv'.

3.4. Analysis

Lemma 3.4. *Given a point set P, and a convex polygon σ that is an inner fence for $P \cap C$; that is, $P \cap C \subseteq \sigma \subseteq C$. Then, there is a convex polygon π, such that*
(A) $P \cap C \subseteq \pi \subseteq \sigma$.
(B) $|\pi| \leq 2\,|\sigma|$.
(C) Every edge of π lies on a line of $L(P)$, see Eq. (3.1).

Proof: Any edge e of σ that does not contain any point of P on it, can be moved parallel to itself into the polygon, till it passes through a point of P. Next, split the edges that contain only a single point of P, by adding this point as a vertex.

Consider a vertex v that is not in P — and consider the two adjacent vertices u, w, which must be in P. If $\triangle uvw \setminus uw$ contains no point of P, then we delete v from the polygon and replace it by the edge uw. Otherwise, move v towards u, until the edge vw hits a point of P. Next, move v towards w, till the edge vu hits a point of P. See Figure 3.6.

Repeat this process till all edges contains two points of P. Clearly, the number of edges of the new polygon π is at most twice the number of edges of σ. ■

Theorem 3.5. *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. The improved classification algorithm performs $O\bigl(\bigl[1 + \circledcirc(P, C)\bigr] \log^2 n\bigr)$ oracle queries. The algorithm correctly identifies all points in $P \cap C$ and $P \setminus C$.*

Proof: The initial stage involves two directional climbs and $O(\log n)$ line cleaning operations, and thus requires $O(\log^2 n)$ queries.

For the later part, consider the inner and outer fences F_{in} and F_{out} of P in relation to C. Applying Lemma 3.4 to F_{in}, results in a convex polygon π that separates $P \cap C$ from ∂C, that has at most $2\,|F_{\text{in}}|$ vertices. Let V be the set of all vertices of the polygons $F_{\text{in}}, F_{\text{out}}$ and π.

A vertical pocket that contains a vertex of V is charged arbitrarily to any such vertex. Since the number of points in a pocket reduces by at least a factor of $1/3$, during a split operation, this means that a vertex would get charged at most $O(\log n)$ times. Each time a vertex gets charged, it has to pay for the $O(\log n)$ oracle queries that were issued in the process of creating this pocket, and later on the price of splitting it. Thus, we only have to account for queries performed in vertical pockets that do not contain a vertex of V.

Consider such a pocket Υ, and assume it lies above the inner approximation. The claim is that the vertical splitting operation, of Section 3.2.3, applied to Υ results in all its points being classified, and thus the algorithm is done with the points in this pocket.

To see that, let U be the set of unclassified points in the pocket at the beginning of the vertical splitting operation. The pocket Υ is bounded by two vertical lines that were cleaned, and as such, there are two points r_L, r_R on these vertical lines that are the intersection of the boundary of the inner approximation (at this stage) B with the two lines.

Similarly, let u_L, u_R be the points where the boundary of π intersect these two vertical lines. By definition, we have $u_L, u_R \in C$. Furthermore, these points lies on lines of $L(P)$. Since both the left and right vertical lines were cleaned before this operation commenced, it must be that $u_L u_R \subseteq B$. But this implies that all the points in Υ that are unclassified are above B, and are thus above $u_L u_R$, and thus they are above π. Namely, all the unclassified points are outside C.

In such a scenario, we claim that all the oracle queries return that the query point is outside C. Assume that this is false, and let $U' \subseteq U$ be the set of unclassified points such that q was the centerpoint for U' and the oracle returned that q is in C. Now q is inside a triangle induced by three points of U'. Namely, there are (at least) two points outside C in this pocket that are not visible to each other. But this implies that F_{out} must have a vertex somewhere inside the vertical pocket Υ, which is a contradiction.

Thus, all the queries return that they are outside the convex body. Each such query results in a constant reduction in the size of the unclassified points, since the query point is a center point of the unclassified points. It follows that after $O(\log n)$ queries, all the points in the pocket are classified.

The above implies that there are at most $O([1 + \circledcirc(P, C)] \log n)$ vertical pockets with no vertex of V throughout the algorithm execution. Since handling such a pocket requires $O(\log n)$ queries, the bound follows. ■

References

[Bár82] I. Bárány. A generalization of Carathéodory's theorem. *Discrete Math.*, 40(2-3):141–152, 1982. doi:10.1016/0012-365X(82)90115-7.

[Cha04] T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In J. Ian Munro, editor, *Proc. 15th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 430–436. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982853.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989. doi:10.1007/BF02187740.

[HJR] S. Har-Peled, M. Jones, and S. Rahul. An animation of the greedy classification algorithm in 2d. URL: https://www.youtube.com/watch?v=IZX0VQdIgNA.

[HKMR16] S. Har-Peled, N. Kumar, D. M. Mount, and B. Raichel. Space exploration via proximity search. *Discrete Comput. Geom.*, 56(2):357–376, 2016. URL: https://doi.org/10.1007/s00454-016-9801-7, doi:10.1007/s00454-016-9801-7.

[KLPS86] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1(1):59–71, 1986. doi:10.1007/BF02187683.

[Mat02] J. Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Grad. Text in Math.* Springer, 2002. URL: http://kam.mff.cuni.cz/~matousek/dg.html, doi:10.1007/978-1-4613-0039-7/.

[PAvdSG13] F. Panahi, A. Adler, A. F. van der Stappen, and K. Goldberg. An efficient proximity probing algorithm for metrology. In *Int. Conf. on Automation Science and Engineering, CASE 2013*, pages 342–349, 2013. URL: https://doi.org/10.1109/CoASE.2013.6653995, doi:10.1109/CoASE.2013.6653995.

A. Proofs

A.1. Proof of Lemma 2.10

The following is a standard consequence of the Clarkson-Shor [CS89] technique.

Restatement of Lemma 2.10. *Let $\mathcal{P}$ be a collection of n pseudo-disks, and let $V_{\leq k}(\mathcal{A})$ be the set of all vertices of depth at most k in the arrangement $\mathcal{A} = \mathcal{A}(\mathcal{P})$. Then $|V_{\leq k}(\mathcal{A})| = O(nk)$.*

Proof: Let $S \subseteq \mathcal{V}$ be a random sample where each pseudo-disk is independently placed into S with probability $1/k$. For each $p \in V_{\leq k}(\mathcal{A})$, let $\mathcal{E}_p$ be the event that p is a vertex in the union $\mathcal{U}(S)$ of this random subset of pseudo-disks. The probability that p is part of the union is at least the probability that both pseudo-disks defining p in $\mathcal{A}$ are sampled into S and the remaining $k-2$ objects containing p are not in S. Thus,

$$\mathbf{Pr}[\mathcal{E}_p] \geq \frac{1}{k^2}\left(1 - \frac{1}{k}\right)^k \geq \frac{1}{e^2 k^2},$$

since $1 - 1/x \geq e^{-2/x}$ for $x \geq 2$. If $|\mathcal{U}(S)|$ denotes the number of vertices on the boundary of the union, then linearity of expectations imply $\mathbf{E}[|\mathcal{U}(S)|] \geq |V_{\leq k}(\mathcal{A})| / (e^2 k^2)$. On the other hand, it is well known the union complexity of a collection of n pseudo-disks is $O(n)$ [KLPS86]. Therefore, $\mathbf{E}[|\mathcal{U}(S)|] \leq \mathbf{E}[c\,|S|] \leq cn/k$, for some appropriate constant c. Putting both bounds on $\mathbf{E}[|\mathcal{U}(S)|]$ together, it follows that $cn/k \geq |V_{\leq k}(\mathcal{A})| / (e^2 k^2) \iff |V_{\leq k}(\mathcal{A})| = O(nk)$. ■

A.2. Proof of Lemma 2.11

The following is a (slightly less) standard consequence of the Clarkson-Shor [CS89] technique.

Restatement of Lemma 2.11. *Let $\mathcal{P}$ be a collection of n pseudo-disks. For two integers $0 < t \leq k$, a subset $X \subseteq \mathcal{P}$ is a* **(t, k)-tuple** *if (i) $|X| \leq t$, (ii) $\exists p \in \cap_{d \in X} d$, and (iii) $\mathrm{depth}(\mathrm{p}, \mathcal{P}) \leq k$. Let $L(t, k, n)$ be the set of all $(\leq t, k)$-tuples of $\mathcal{P}$. Then $|L(t, k, n)| = O(ntk^{t-1})$.*

Proof: Let $R \subseteq \mathcal{P}$ be a random sample, where each pseudo-disk is independently placed into R with probability $1/k$. Consider a specific (t, k)-tuple X, with a witness point p of depth $\leq k$. Without loss of generality, by moving p, one can assume p is a vertex of $\mathcal{A}(\mathcal{P})$.

Let $\mathcal{E}_X$ be the event that p is of depth exactly t in $\mathcal{A}(R)$, and $X \subseteq R$. For $\mathcal{E}_X$ to occur, all the objects of X need to be sampled into R, and each of the at most $k - t$ pseudo-disks containing p in its interior are not in R. Therefore

$$\mathbf{Pr}[\mathcal{E}_X] \geq \frac{(1 - 1/k)^{\text{depth}(\text{p},\mathcal{P})-|X|}}{k^{|X|}} \geq \frac{(1 - 1/k)^k}{k^t} \geq \frac{1}{e^2 k^t}.$$

Note, that a vertex of depth $\leq k$ in $\mathcal{A}(R)$ corresponds to at most one such an event happening. We thus have, by linearity of expectations, that

$$\frac{|L(t,k,n)|}{e^2 k^t} \leq \mathbf{E}\big[|V_{\leq t}(\mathcal{A}(R))|\big] = O(tn/k),$$

by Lemma 2.10. ■

B. On emptiness variants in two dimensions

Here, we present two instance-optimal approximation algorithms for solving the following two variants:

(A) Emptiness: Find a point $p \in P \cap C$, or using as few queries as possible, verify that $P \cap C = \varnothing$.

(B) Reverse emptiness: Find a point $p \in P \setminus (P \cap C)$, or using as few queries as possible, verify that $P \cap C = P$.

For both variants we present $O(\log n)$ approximation (the algorithm for emptiness is randomized), improving over the general approximation algorithm of Section 3 which provides a $O(\log^2 n)$ approximation.

B.1. Emptiness: Are all the points outside?

Here we consider the problem of verifying that all the given points are outside the convex body.

Algorithm. The algorithm is a slight modification of the algorithm of Section 2.2.2. At each iteration the point set U^+ is the largest set of currently unclassified points in P contained in some halfspace tangent to the current inner approximation B. Let $\omega = |U^+|$. We make the following changes: If $\omega = O(1)$, test the membership of each point individually. Otherwise, choose a random point $q \in U^+$. If q is found to be inside C, we are done, as q is our witness. Otherwise q is outside, and a **remove** operation is performed. The algorithm then performs a regular iteration on U^+, as described in Section 2.2.2.

Analysis. Let G_i be the intersection graph in the beginning of the ith iteration only on the points outside C. We need the following technical lemma:

Lemma B.1. *Suppose $P \cap C = \varnothing$. Then at any iteration i, the largest independent set in the visibility graph G_i is at most $|F_{\text{out}}|$.*

Proof: For the body C and point set P, define the set $R \subseteq P$ to be the maximum set of points such that no two points in R are visible with respect to C. Observe that R corresponds to the maximum independent set in the visibility graph for P with respect to the body C. We claim $|R| \leq |F_{\text{out}}|$. Suppose that $|R| > |F_{\text{out}}|$. Given the polygon F_{out}, for each edge e of F_{out} consider the line ℓ_e through e and let $\hbar_e^+$ be the halfspace bounded by ℓ_e which does not contain C in its interior. Then $\{\hbar_e^+ \mid e \in F_{\text{out}}\}$ covers the space $\mathbb{R}^2 \setminus \text{int}(C)$. By the hypothesis, one halfspace $\hbar_e^+$ must contain at least two points of R. But then these two such points are visible with respect to C, contradicting the definition of R.

We know that the size of the largest independent set (with respect to the current inner approximation B_i) is monotone increasing over the iterations. Because all points of P are outside of C, each independent set can be of size at most $|R| \leq |F_{\text{out}}|$. ■

Lemma B.2. *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. The randomized greedy classification algorithm for emptiness performs $O\big((|F_{\text{out}}| + 1)\log n\big)$ oracle queries with high probability. The algorithm always correctly verifies that $P \cap C = \varnothing$ or finds a witness point of P inside C.*

Proof: Suppose $P \cap C = \varnothing$. Then Lemma B.1 along with the proof of Theorem 2.8 implies the result, by replacing the quantity $\hexagon_P$ with $|F_{\text{out}}|$. If $P \cap C \neq \varnothing$, let U^+ be a set of points in the current iteration, $U^+_{\text{in}} = U^+ \cap C$, and $U^+_{\text{out}} = U^+_{\text{out}} \setminus U^+_{\text{in}}$. Observe that U^+_{in} remains the same throughout the algorithm execution, while U^+_{out} shrinks. If $\left|U^+_{\text{out}}\right| > |U^+|/2$, then by Lemma 2.6 the number of edges removed from G_i is $\Omega\Big(\left|U^+_{\text{out}}\right|^2\Big)$ (though the hidden constants will be smaller). Thus, after at most $O\big((|F_{\text{out}}| + 1)\log n\big)$ iterations, we must encounter an iteration in which there is a set of points U^+ with $\left|U^+_{\text{out}}\right| < |U^+|/2$. Now the probability that our randomly sampled point lies in U^+_{in} is at least $1/2$. In particular, after an additional $O(\log n)$ iterations, the probability that we fail to find a witness point is at most $1/n^{O(1)}$, thus implying the bound on the number of queries. ■

B.2. Reverse emptiness: Are all the points inside?

Here we consider the problem of verifying that all the given points are inside the convex body.

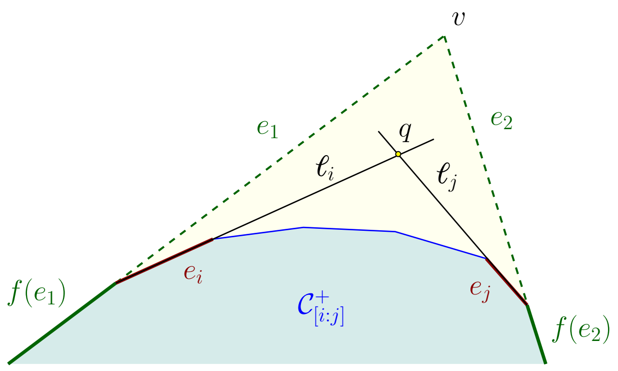
B.2.1. Algorithm

Initialization. Let $\mathcal{C} = \text{ch}(P)$. Let $v, v' \in P$ denote the extreme left and right vertices of $\mathcal{C}$. Let v_1 and v_2 be the vertices adjacent to v on $\mathcal{C}$. Similarly define v'_1 and v'_2. The algorithm asks the oracle for the status of v, v_1, v_2, v', v'_1, and v'_2. If any of them are outside, the algorithm halts and reports the witness found. Otherwise, all points must lie either above or below the horizontal segment vv'. We now describe how to handle the points above vv' (the below case is handled similarly).

Let $\mathcal{C}^+$ be the polygonal chain which is $\mathcal{C}$ clipped inside region bounded by the segment vv' and two vertical lines passing through v and v'. Label the edges along $\mathcal{C}^+$ by $e_1, \ldots, e_k$ clockwise from v to v'. For $1 \leq i < j \leq k$, let $\mathcal{C}^+_{[i:j]}$ be the polygonal chain consisting of the consecutive edges $e_i, \ldots, e_j$. The algorithm now invokes the following recursive procedure.

Recursive procedure. A recursive call is described by two indices (i, j), and the goal is to verify that all points of P lying below $\mathcal{C}^+_{[i:j]}$ are inside the convex body C.

For a given recursive instance (i, j), the algorithm proceeds as follows. Begin by computing the line ℓ_i and ℓ_j through the edges e_i and e_j respectively. Let $q = \ell_i \cap \ell_j$ be the point of intersection. The algorithm asks the oracle for the status of q. If q is inside, then all points below $\mathcal{C}^+_{[i:j]}$ must also be in C. The algorithm classifies the appropriate points and returns. Otherwise q is outside, and generates two recursive calls. Let $\ell = \lfloor (i + j)/2 \rfloor$ and $e_\ell = (x, y)$ be the middle edge in the chain $\mathcal{C}^+_{[i:j]}$. The algorithm queries the oracle with x and y. If either x or y is outside, the algorithm returns the appropriate witness found. Otherwise x and y are both inside. The algorithm recurses on the instances (i, ℓ) and (ℓ, j).



B.2.2. Analysis.

The analysis will use the polygon π as defined in Lemma 3.4. Specifically, it is an inner fence where $|\pi| = O(|F_{\text{in}}|)$ and every edge of π lies on a line of $L(P)$, see Eq. (3.1). In particular, since π is an inner fence, $\mathcal{C} \subseteq \pi$ and furthermore every edge of $\mathcal{C}$ lies on a line of $L(P)$. Each edge e of π can be matched with an edge $f := f(e)$ of $\mathcal{C}$ if there is some line in $L(P)$ which contains both edges. If e is matched, then it is ***active***.

A recursive call (i, j) is ***alive*** if the query $q = \ell_i \cap \ell_j$ generated (see above) is outside C.

Lemma B.3. *The number of alive recursive calls at the same recursive depth is at most $|\pi| = O(|F_{\text{in}}|)$.*

Proof: Fix an alive recursive call (i, j) with edges $e_i, \ldots, e_j$ of $\mathcal{C}$. Suppose that none of these edges are active. Because π is an inner fence for P and C, there must be a vertex v of π lying above the chain $\mathcal{C}^+_{[i:j]}$. Let e_1 and e_2 be the edges adjacent to v in π. For $\ell = 1, 2$, consider $f(e_\ell)$, the edge of $\mathcal{C}$ associated with e_ℓ. Since there are no active edges in $\mathcal{C}^+_{[i:j]}$, we have $f(e_\ell) \notin \{e_i, \ldots, e_j\}$ for $\ell = 1, 2$. But this readily implies that all vertices in the polygonal chain $\mathcal{C}^+_{[i:j]}$ are contained in the wedge formed by v and the two edges e_1 and e_2. In particular, the query q generated is inside π and thus C. Contradicting that the recursive call was alive.

It follows that each alive recursive call must contain at least one active edge. The number of active edges is bounded by $|\pi|$, implying the result. ■

Lemma B.4. *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. The classification algorithm for reverse emptiness performs $O\big(|F_{\text{in}}| \log n\big)$ oracle queries. The algorithm correctly verifies that $P \cap C = P$ or finds a witness point of P outside C.*

Proof: Suppose all points of P are inside C. By Lemma B.3, there are at most $O(|F_{\text{in}}|)$ alive recursive calls at each level of the recursion tree. Since the depth of the recursion tree is $O(\log n)$, the number of total alive recursive calls throughout the algorithm is $O(|F_{\text{in}}| \log n)$. At each alive recursive call of the above algorithm, $O(1)$ queries are made. This implies the result.

Otherwise not all points of P are inside C. At least one such point outside of C must be a vertex on the convex hull $\mathcal{C}$. Hence after at most $O(|F_{\text{in}}| \log n)$ oracle queries, this vertex will be queried and found to be outside C. ■