Extreme Point Detection and the Polar Dual

by

Adam J. Hartfiel

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the Department of Mathematics and Statistics
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

Approved by:

Dr. Richard J. Caron, Supervisor
Department of Mathematics and Statistics

Dr. Tim Traynor, Internal Reader
Department of Mathematics and Statistics

Asish Mukhopadhyay, External Reader
School of Computer Science

Dr. Myron Hlynka, Chair of Defense
Department of Mathematics and Statistics

Abstract

This thesis is concerned with the identification of the extreme points of a given finite set of points denoted by S. In Jibrin, Boneh and Caron[18] the authors explored the advantage of using probabilistic methods applied to the polar dual of S to quickly detect some extreme points. This thesis builds on those results with the introduction of a polynomial time procedure that uses the detected extreme points to eliminate redundant points.

Acknowledgments

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview and Outline of Thesis

This thesis builds on the results in Jibrin, Boneh and Caron [18] for the identification of the extreme points of the convex hull of a finite subset $S = \{p_1 \dots p_n\}$ of $\mathbb{R}^d$. The extreme point identification problem is to determine the smallest subset $\hat{S}$ of S such that the convex hulls of S and $\hat{S}$ are the same. The results in [18] are based on the fact that a point $p_k \in S$ is extreme if and only if the inequality $p_k^T y \leq 1$ is necessary with respect to the representation $\{\, y \in \mathbb{R}^d \,|\, p_i^T y \leq 1,\ \forall p_i \in S\}$ of the polar dual S^* of S. Thus, $\hat{S}$ can be found by classifying the inequalities in the representation as necessary or redundant; and since this can be done with linear programming, the extreme point identification problem is polynomial.

In [18] the authors show the advantage of using a probabilistic *hit-and-run* algorithm applied to the polar dual for the quick identification of extreme points. They also show how redundancy, i.e., points in $S \setminus \hat{S}$ can be detected. In Chapter 2 we present the work in [18] in detail and we explore the strengths

and weaknesses and offer improvement. In Chapter 3 we show how some of the assumptions in [18] can be relaxed. in Chapter 4 we propose a new preprocessor for removing points in $S \setminus \hat{S}$ that requires polynomial time in the worst case and scales to spaces of arbitrary dimension. Experimental results are given in Chapter 5.

1.2 Notation and Definitions

We adopt the notation and definitions of [15]. Let A be a subset of $\mathbb{R}^d$, K be a convex subset of $\mathbb{R}^d$ and S be a finite subset of $\mathbb{R}^d$. We denote the cardinality of a finite set S by $|S|$.

Definition 1.2.1 (Interior)**.** *We call $a \in A$ an interior point if we can find $\epsilon \geq 0$ such that if $|a - b| < \epsilon$ then $b \in A$.*

We call the collection of all interior points of a set A the interior of A and denote it $\text{int}(A)$.

Definition 1.2.2 (Convex Set)**.** *We say that $K \subseteq \mathbb{R}^d$ is a convex set if for all distinct x and y in K the closed line segment with endpoints x and y is contained in K. Equivalently K for all distinct x and y in K, $\lambda x + (1 - \lambda)y$ is in K for λ in $[0, 1]$.*

Definition 1.2.3 (Convex Sum)**.** *Given a finite set S we say a summation of these vectors is a convex sum if all coefficients are greater than or equal to 0 and the sum of all coefficients is 1.*

Definition 1.2.4 (Convex Hull)**.** *The convex hull of $A \subset \mathbb{R}^d$ is the intersection*

of all convex subsets of $\mathbb{R}^d$ *that contain* A*. We denote the convex hull of* A *as* $conv(A)$*.*

Lemma 1.2.1. *The convex hull of* $A \subseteq \mathbb{R}^d$ *convex.*

Proof. Let x, $y \in conv(A)$ and $x \neq y$. Any convex set K containing A also contains x and y and hence contains the closed line segment with endpoints x and y. Therefore $conv(A)$ contains the closed line segment with endpoints x and y and is hence convex. □

Lemma 1.2.2. *The convex hull of a finite set* S *is bounded.*

Proof. Let $S = \{p_1 \dots p_k\} \subset \mathbb{R}^d$. Let $min_i = min_j\{(p_j)_i\}$ and $max_i = max_j\{(p_j)_i\}$ for each $i \in \{1 \dots d\}$. The set $[min_1, max_1] \times [min_2, max_2] \times \dots \times [min_d, max_d]$ is a bounded convex set containing S and hence it contains $conv(S)$. □

Definition 1.2.5 (Full Dimensional Set)**.** *We say a subset* A *of* $\mathbb{R}^d$ *has full dimension if the interior of* $conv(A)$ *is not empty.*

Definition 1.2.6 (Extreme Point)**.** *Let* K *be a convex subset of* $\mathbb{R}^d$ *we call* $x \in K$ *an extreme point of* K *if* x *does not lie on the interior of any line segment contained in* K*. That is if* $x = \lambda y + (1 - \lambda)z$ *with* y *and* z *in* K *and* $0 < \lambda < 1$ *then* $x = y = z$*.*

We use an overset caret to denote the set of extreme points of the convex hull of a set. For example $\hat{A}$ is the set of extreme points of conv(A). We denote the cardinality of $\hat{A}$ by h.

Definition 1.2.7 (Chord of a Convex Hull)**.** *Let x and y be distinct extreme points of conv(S), we call the line segment xy a chord of the convex hull of S.*

1.2.1 Polytopes

In this section, we introduce the concept of a polytope along with related definitions and results. We conclude the section by showing that any polytope with non empty interior can be translated to contain the origin.

Definition 1.2.8 (Polyhedral Set)**.** *The set $A \subseteq \mathbb{R}^d$ is a polyhedral set if A is the intersection of a finite family of closed halfspaces of $\mathbb{R}^d$.*

Definition 1.2.9 (Polytope)**.** *A polytope is the convex hull of a finite set of points in $\mathbb{R}^d$.*

Equivalently a polytope is a bounded polyhedral set[15].

Definition 1.2.10 (Halfspaces Defined by a (d-1)-Hyperplane)**.** *Let $H = \{x \in \mathbb{R}^d \mid a^\top x = b\}$ be a $(d-1)$-hyperplane, the half spaces defined by H are $H_- = \{x \in \mathbb{R}^d \mid a^\top x \leq b\}$ and $H_+ = \{x \in \mathbb{R}^d \mid a^\top x \geq b\}$.*

Definition 1.2.11 (Distance Between H and A)**.** *Let H be a hyperplane and A be a subset of $\mathbb{R}^d$, the distance between H and A is $\inf\{|h - a| \mid h \in H, a \in A\}$.*

Definition 1.2.12 (Supporting Hyperplane)**.** *We call H a Supporting Hyperplane of $A \subseteq \mathbb{R}^d$ if the distance from H to A is zero and A is contained entirely in one of the closed half spaces defined by H.*

Definition 1.2.13 (Face)**.** *Let K be a convex subset of $\mathbb{R}^d$ and $F \subseteq K$. We call F a face of K if $F = \emptyset$ or $F = K$ or there exists a supporting hyperplane H*

of K such that $F = H \cap K$.

We adopt the notation *k-face* for a face of dimension k.

Definition 1.2.14 (Edge)**.** *Let K be a polytope and F be a 1-face of K, we call F an edge of K.*

Definition 1.2.15 (Facet)**.** *Let K be a polytope of dimension d and F be a maximal proper face (a (d-1)-face) of K, we call F a facet of K.*

Lemma 1.2.3. *The polytope defined by a set S is the same as the polytope defined by $\hat{S}$, the extreme points of S. That is conv(S) = conv($\hat{S}$).*

Proof. Without loss of generality we assume $|S| > 0$. By definition 1.2.6 $\text{conv}(S) \supseteq \hat{S}$ thus by definition 1.2.4 $\text{conv}(S) \supseteq \text{conv}(\hat{S})$. We now show that $\text{conv}(\hat{S}) \supseteq \text{conv}(S)$. The set S is finite, therefore $\hat{S}$ is also finite, subsequently the convex hull must be bounded. We proceed via induction on the dimension of $\text{conv}(S)$, we assume without loss of generality that the span of S is $/mathbbR^d$.

If $d = 0$ the result is trivial, let $d = 1$ then $conv(S) = [a, b]$ where a and b are the maximum and minimum elements in S respectively. Then $\hat{S} = \{a, b\}$ and $conv(\hat{S}) = [a, b] \supseteq conv(S)$. We assume this holds for $d \leq \delta - 1$ and consider $x \in \text{conv}(S) \subseteq \mathbb{R}^\delta$. Without loss of generality we assume x is an interior point, otherwise x is contained in a face of dimension less than d and the inductive hypothesis applies. We now select another distinct point $\bar{x} \in conv(S)$, these two points define a line passing through the convex hull that must intersect the boundary at 2 distinct faces of $\text{conv}(S)$ say $F_1 = conv(S) \cap H_1$ and $F_2 = \text{conv}(S) \cap H_2$ at points x_1 and x_2 respectively. We observe $x =$

$\lambda x_1 + (1-\lambda)x_2$ for some $\lambda \in (0,1)$. The extreme points of these 2 facets are also extreme points of conv(S), thus by the inductive hypothesis x_1 and x_2 are elements of conv$(\hat{S})$, thus $x \in$ conv$(\hat{S})$ and conv$(\hat{S}) \supseteq$ conv(S). Therefore conv$(\hat{S})$ =conv(S). □

Definition 1.2.16 (Barycentre)**.** *Let $S = \{p_1, p_2 \dots p_n\} \subseteq \mathbb{R}^d$, the barycentre of S is $B_S = \sum_{i=1}^{n} \frac{1}{n} p_i$.*

Theorem 1.2.1. *If S is full dimensional its barycentre is in the interior of* conv(S).

Proof. Definition 1.2.15 shows that the barycentre B_s is in conv(S). Without loss of generality we assume $B_S = 0$ as we can translate S by $-B_S$ otherwise. We now show that it is in the interior. Consider otherwise, that is 0 lies on a facet F of conv(S) with supporting hyperplane H then 0 can be expressed as a convex sum of only the extreme points $X = \hat{S} \cap H$. As H is a supporting hyperplane conv(S) lies entirely in one of the halfspaces it defines, say H_+, let v be a vector normal to H and pointing into H_+. We select an element a of $S \setminus X$ the projection of such an a onto v must be positive. We observe that any element of $S \cap H$ must have a 0 projection onto v. This leads to a contradiction as the barycentre of S must have a nonzero projection onto v a non zero vector, however the barycentre is 0 by construction. Therefore B_S does not lie on a facet and is hence in the interior of *conv*(S). □

1.2.2 The Polar Dual

The extreme point identification method of Jibrin, Boneh and Caron is based upon the concept of the polar dual of a convex set.

Definition 1.2.17 (The Polar Dual)**.** *The polar dual* A^* *of a set* $A \subseteq \mathbb{R}^d$ *is* $\{x \in \mathbb{R}^d \mid p^\top x \leq 1, \forall p \in A\}$.

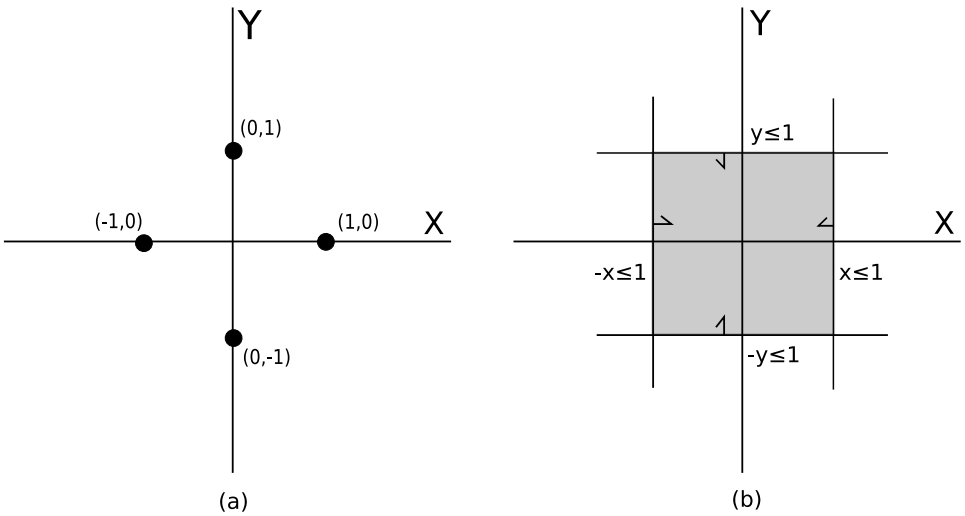

Figure 1.1: An Example Polar Dual

Example 1.2.1 (A Polar Dual)**.** *In Figure 1.1 (a) the members of the set* $S = \{(1,0)^\top, (0,1)^\top, (-1,0)^\top, (0,-1)^\top\}$ *in* $\mathbb{R}^2$ *are plotted. In Figure 1.1 (b) we have* S^* *with the interior shaded and the constraint corresponding to* p_i *labeled with the inequality* $(x,y)^\top p_i \leq 1$ *for each* p_i *in* S.

We now introduce some key properties of the Polar Dual.

Lemma 1.2.4. *Let* $A \subseteq \mathbb{R}^d$ *then* A^* *is bounded if and only if* $0 \in \text{int}(\text{conv}((A))$.

Proof. Let $0 \in \text{int}(\text{conv}(A))$. By definition 1.2.1 there exists ϵ such that $|b| < 2\epsilon \Rightarrow b \in \text{conv}(A)$. Let $E = \{\epsilon a V_i \mid a \in \{1,-1\}, (V_i)_i = 1, (V_i)_j = 0 \text{ if } i \neq j\}$.

By the selection of ϵ we have $E \subseteq A$ and hence $A^* \subseteq E^*$ because the set of constraints defining E^* is a subset of the constraints defining A^*. However by the construction of E we have that E^* is bounded and hence A^* is bounded. Indeed, for any $x \in E^*$ and $i \in 1 \dots d$ we have $\epsilon x_i \leq 1$ and $-\epsilon x_i \leq 1$ and hence $x_i \in [\frac{1}{-\epsilon}, \frac{1}{\epsilon}]$. Thus $x \in [\frac{1}{-\epsilon}, \frac{1}{\epsilon}]^d$ and A^* is bounded.

Suppose $0 \notin \text{int}(\text{conv}(A))$. If 0 is on the boundary of $\text{conv}(A)$ set $\vec{v}$ to be a vector normal to a supporting hyperplane of $\text{conv}(A)$ passing through 0. If 0 is an exterior point consider the supporting hyperplane H of $\text{conv}(A)$ nearest 0 and set $\vec{v}$ to the normal of H. The selection of $\vec{v}$ guarantees $\vec{v} \neq 0$ and $\text{conv}(A)$ is a subset of either $\vec{v}^\top x \leq 0$ or $-\vec{v}^\top x \leq 0$, in the second case we negate $\vec{v}$. We have $\vec{v}^\top x \leq 0 \ \forall x \in \text{conv}(A)$, so $0 + \alpha\vec{v} \in A^* \ \forall \alpha > 0$ hence A^* is unbounded. Therefore A^* is bounded implies $0 \in \text{int}(\text{conv}(K))$.

Therefore $K \subseteq \mathbb{R}^d$ then A^* is bounded if and only if $0 \in \text{int}(\text{conv}(A))$. □

This result allows us to find for any point set in $\mathbb{R}^d$ with full dimensional convex hull a translation with a bounded polar dual. We now show that such a translation must exist.

Lemma 1.2.5. *Let $S \subseteq \mathbb{R}^d$ and $\text{int}(\text{conv}((S)) \neq \emptyset$ then there exists a translation of S, S' with $0 \in \text{int}(\text{conv}(S'))$.*

Proof. We consider the Barycenter B_S of S. We have that $\text{int}(\text{conv}(S))$ is non empty so it is full dimensional and by theorem 1.2.1 it contains B_S . Let $S' = \{a \mid a + B_S \in S\}$ then $0 \in \text{int}(\text{conv}((S'))$. □

Lemma 1.2.6. *There are no duplicate constraints in S^*.*

Proof. Suppose we can find a pair of duplicate constraints in S^*, say $a_1^\top x \leq 1$ and $a_2^\top x \leq 1$ with $\{x \mid a_1^\top x \leq 1\} = \{x \mid a_2^\top x \leq 1\}$ with $a_1 \neq a_2$. Then by the seperation theorem we can find x such that $a_1^\top x \leq 1$ and $a_2^\top x > 1$, a contradiction. Therefore there are no duplicate constraints in S^*. □

1.3 Convex Hull Algorithms

In this section we introduce some of the standard algorithms for finding $\text{conv}(S)$ when the dimension $d = 2$. For this special case all that is necessary to determine the convex hull is either an ordered list of the extreme points or an ordered list of the facets of the convex hull. Given a set $\bar{S}$ of known extreme points of S determining the convex hull of $\bar{S}$ can provide a quick method of determining what points in $S \setminus \bar{S}$ are in $\text{conv}(\bar{S})$ and hence can not be extreme points of S. There are limitations to this method however, as algorithms for finding the convex hull of sets in $\mathbb{R}^d$ have time complexity that is exponential in d as we will see.

1.3.1 Graham Scan

The steps of the Graham Scan[13] algorithm are as follows.

0.0. Set p to be the element of S with minimum y coordinate, in the event of a tie take the point with minimum x coordinate.

0.1. Append p to the initially empty list T.

0.2. Sort the remaining points in increasing angular order around p, where the angle is taken between the vector from p to the point in question and the

positive x axis. Denote the sorted list as L.

0.3. Append the first point in L to T.

0.4. Set $i = 2$

1.0. Calculate the cross product o of the vector defined by the last two elements of T and the vector defined bythe last element of T with element i in L.

1.1. If $o > 0$ append $L[i]$ to T.

1.2. If $o < 0$ remove the last element of T and return to step 1.

1.3. If $o = 0$ remove the last element of T and append $L[i]$ to T.

2.0. Increment i, if i is less than $|L|$ return to step 1.0.

3.0. T now contains an ordered list of the extreme points of the convex hull.

We examine the time complexity of Graham Scan. Determining p in 0.0. requires $O(n)$ time. The sort in step 0.2. requires $O(n \log n)$ time. The calculation of the convex hull then requires $O(n)$ time as every element of L is only added to T once, and removed at most once. This results in $O(n \log n)$ total running time using $O(n)$ space.

While Graham Scan is an efficient and elegant way to find a two dimensional convex hull, the reliance on angular sorting prevents generalization to higher dimensions.

Example 1.3.1. *In Figure 1.2 we are given a set of points in* $\mathbb{R}^2$ *with the convex hull being calculated using Graham's scan. The point labeled T[0] is the point p added during step (0.1), T[1] is the point added at step (0.3) and T[2] through*

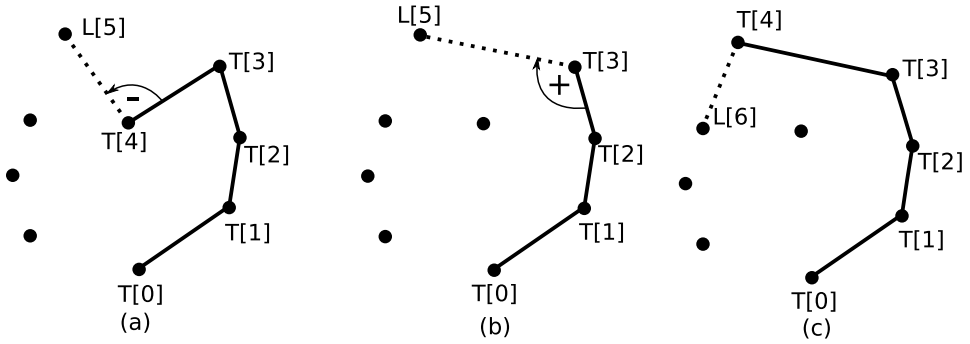

Figure 1.2: Example Graham Scan Iteration.

T[4] are points added during 3 subsequent iterations. Figure 1.2(a) shows the beginning of the 4th iteration, with $i = 5$ and L[5], the current point of interest labeled. We see the cross product calculated in step (1.0) is negative so we remove T[4] from T leading to the situation in Figure 1.2(b). Performing the cross product calculation of step (1.0) yields a positive value, so we add L[5] to T, making it the new T[4] and increment i as in Figure 1.2(c). Intuitively negative cross products correspond to counter clockwise angles and positive values correspond to clockwise angles.

1.3.2 Jarvis' March

We now examine Jarvis' March[17]. The Graham Scan algorithm determined the ordered set of extreme points. Jarvis' March instead determines the ordered set of facets. Given a set of points S in the $\mathbb{R}^2$ Jarvis' March works by selecting a point guaranteed to be on the convex hull and in essence wrapping a line around the set of points until reaching the start point again. The steps are as follows.

0.0. Set p to be the element of S with minimum y coordinate, in the event of

a tie take the point with minimum x coordinate.

0.1. Append p to the empty list T.

0.3. Find a in $S \setminus T$ that maximizes the angle between ap and the negative x axis.

0.4. Append a to T.

1.0. Find a in S that maximizes the cross product of the vector defined by the last two elements of T and the vector defined by a and the last element of T.

2.0. If $a \neq T[0]$, append a to T and go to (1.0).

3.0. T now contains the extreme points of the convex hull in sequential order.

We first observe that the algorithm visits each extreme point of the convex hull exactly once. At an extreme point the algorithm checks the resulting angle with every other point in the set, or O(n) points. Let $|\hat{S}| = h$, then the algorithm runs in O(nh) time. This yields worstcase O(n^2) time complexity when $S = \hat{S}$.

Example 1.3.2. *In Figure 1.3(a) we are given a set of points in* $\mathbb{R}^2$ *with the initial point p determined in step 0.0. labeled. In Figure 1.3(b) we see the initial convex hull point p added to the list T in step (0.1), the angle corresponding to the maximal cross product is drawn along with the second point added to T labeled T[1]. By completing another iteration yields the addition of T[2] to T as seen in (c). After five more iterations we detect T[0] again and terminate with the extreme points stored in T as in (d).*

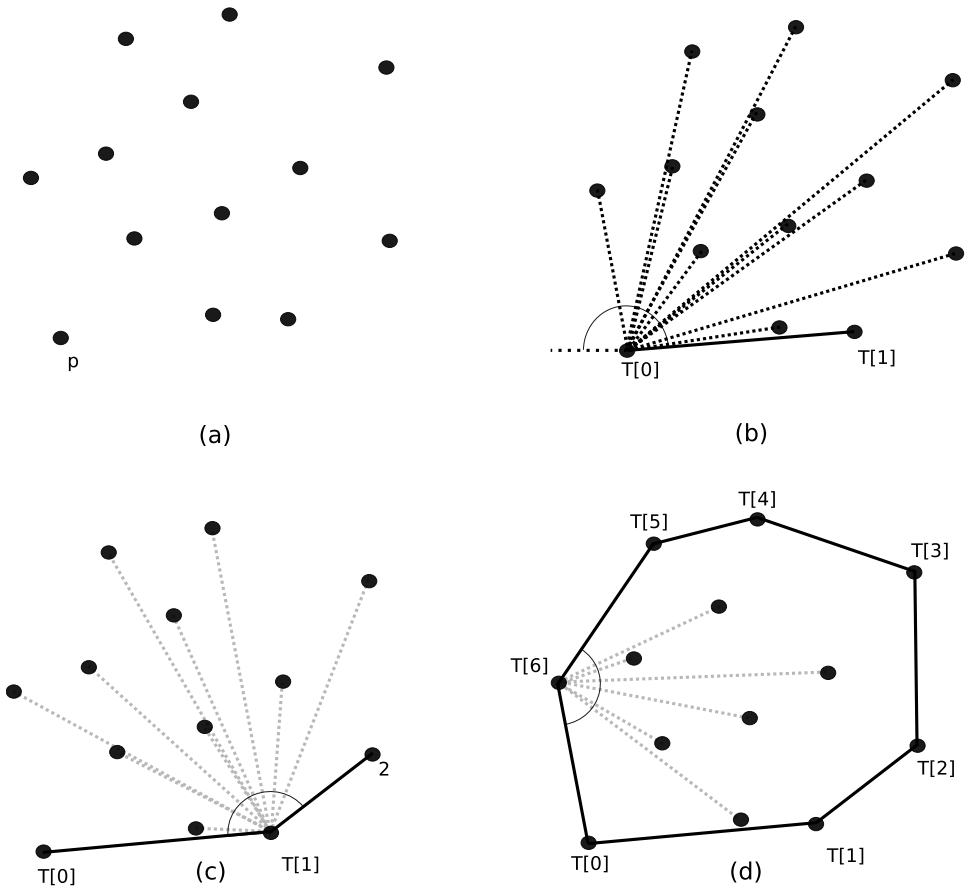

Figure 1.3: Example Jarvis' March Problem.

Jarvis' March can be viewed as a specialization of the Gift-wrapping [11] algorithm of Chand and Kapur. Gift-wrapping begins with a convex hull facet and detects neighbouring facets moving to them and repeating the process. While Jarvis's March can be interpretted as wrapping a string around a set in $\mathbb{R}^2$, Gift-wrapping can be interpretted as wrapping a hyperplane around a point set in $\mathbb{R}^d$. The worst case complexity of gift-wrapping is $\mathrm{O}(n^{\lfloor d/2 \rfloor + 1} + n^{\lfloor d/2 \rfloor} \log n)$. A full analysis can be found in [21] and is beyond the scope of this thesis.

1.3.3 Chan's Algorithm

The algorithm presented by Timothy Chan[10] in 1996 is a simple optimal output-sensitive convex hull algorithm for sets in $\mathbb{R}^2$. The algorithm works by arbitrarily partitioning the input set into subsets no larger than a parameter m that is assumed to be less than h. The convex hull of each partition is found using a $O(n \log n)$ algorithm like Graham Scan, resulting in a family of of sub hulls. The algorithm then attempts to merge the resulting sub hulls by performing at most $m + 1$ iterations of Jarvis' March on the extreme points of the sub hulls. If the iteration limit is reached the process starts again with a larger value for m. The value of m is squared after each failed attempt but not allowed to exceed n, thus the value of m at iteration $t(\geq 0)$ is $\min\left(n, 2^{2^t}\right)$. The resulting worst case complexity is $O(n \log h)$.

Chan's algorithm can be extended to the three dimensional case by using an $O(n \log n)$ three dimensional convex hull algorithm like Divide and Conquer[21] in place of Graham Scan along with a three dimensional implementation of Gift-wrapping.

Example 1.3.3. *In Figure 1.4 (a) we are given a set of points in $\mathbb{R}^2$, we now use Chan's Algorithm to find the convex hull. In (b) we see the result of attempting to find the hull with the parameter m set to 2. In (c) We see the result of attempting to find the hull with the parameter m set to 4. Finally in (d) we find the hull by setting m to 8.*

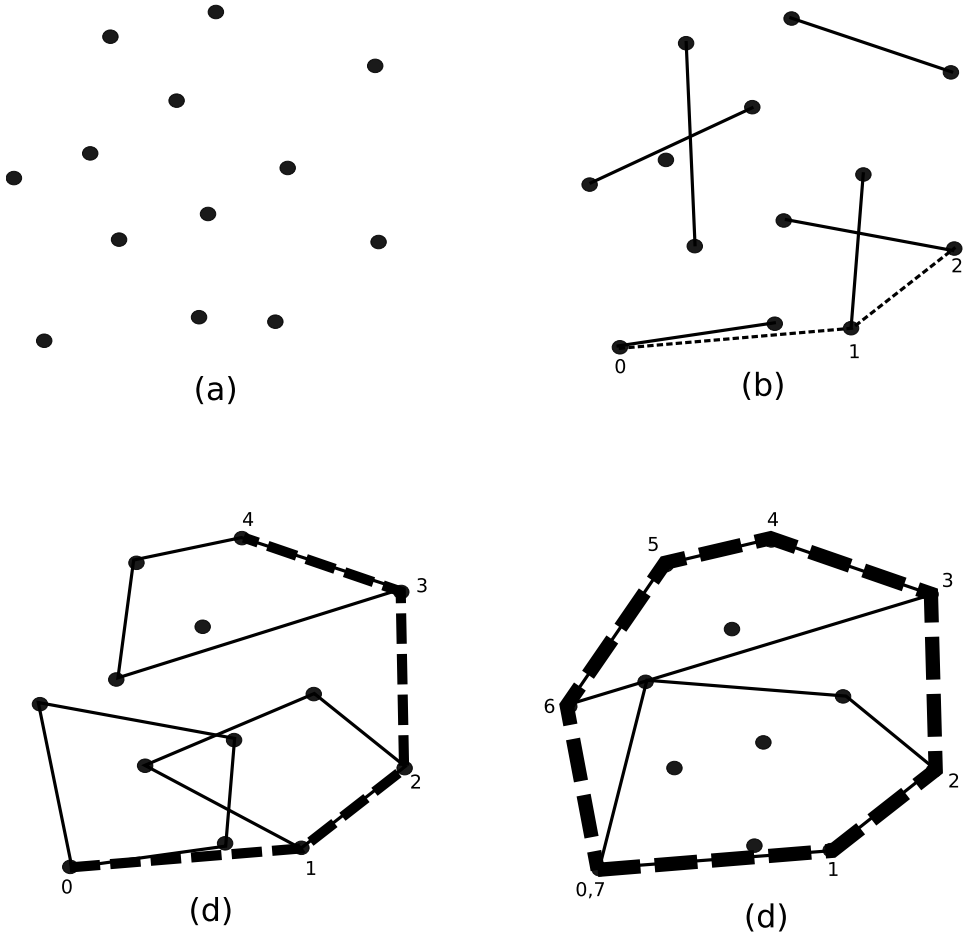

Figure 1.4: Chan's Algorithm Example.

1.3.4 Quickhull

We now look at a family of algorithms independantly derived by mulitiple sources[12, 6, 14] in the mid 1970s[21]. The algorithm functions in a fashion similar to the Quicksort[16], and ultimately has some of the same weaknesses.

The Quicksort algorithm is a comparison sort that functions by partitioning the input set into those less than or equal to and those greater than a value selected via a partitioning rule. Partitioning requires $O(n)$ time. The algorithm then recurses on the resulting subsets with cardinality greater than one. The expected performance with a good paritioning rule is $O(n \log n)$[16]. With a

naive partitioning rule performance can be as bad as $O(n^2)$. For example if we take the first element as the paritioning value, then input of a list that is already sorted in increasing order will result in $\sum_{i=1}^{n} i = \frac{n(n-1)}{2} = O(n^2)$ comparisons.

The Quickhull[21] algorithm in $\mathbb{R}^2$ begins by selecting a chord of the convex hull with end points l and r. An easy way of finding such a chord is selecting the points with maximum and minimum values in a coordinate. The chord partitions the set into two sets, the points on either side of the chord, say S_1 and S_2. We now find the point $p \in S_i$ that maximizes the area of the triangle prl (Δprl), with ties broken by finding the point which maximizes $\angle plr$.

Theorem 1.3.1. *Given a chord lr of $conv(S)$ and $p \in S_i$ that maximizes Δprl with ties broken by selecting p to maximize $\angle plr$, then p is an extreme point.*

Proof. Suppose that p is not extreme. Then p is either an interior point of $conv(S)$ or it is a boundary point of $conv(S)$.

If p is an interior point we select select a vector $\vec{o} \perp lr$ that points into the half plane containing p. We can find $\alpha > 0$ such that $p + \alpha\vec{o} = \hat{p}$ is a boundary point. if $\hat{p}$ is an extreme point then $\hat{p} \in S$ with $\Delta\hat{p}rl > \Delta prl$ a contradiction. Otherwise $\hat{p}$ is on an edge of $conv(S)$, say x_1x_2. We set x to be a x_i with maximum distance from the chord lr. Any such x will be such that $\Delta xrl > \Delta\hat{p}rl > \Delta prl$, contradicting maximality of Δprl among the possible $p \in S_i$.

If p is a boundary point it must be interior to an edge x_1x_2 of $\text{conv}(S)$, so $p = \lambda x_1 + (1-\lambda)x_2$. If x_1x_2 is parallel to lr then for any $p' \in x_1x_2$ we have $\Delta p'rl = \Delta prl$. However one of the following must be true, $\angle plr < \angle x_1lr$ or

$\angle plr < \angle x_2lr$, a contradiction. If x_1x_2 is not parallel to lr we set x to be a x_i with maximum distance from the chord lr. Any such x will be such that $\Delta xrl > \Delta prl$ a contradiction.

Therefore p must have been an extreme point of $conv(S)$

□

We proceed by eliminating the points interior to Δprl in S_i from consideration, as they are guaranteed to be interior to $conv(S)$. The remaining points are partitioned into the points opposite pr and the points opposite lp, by selection of p these sets are disjoint, otherwise if there was a point p_0 in the intersection then $\Delta p_0rl > \Delta prl$. We repeat this process on the two subsets using pr and lp in place of the original chord respectively. This recursion is continued until all the extreme points are identified.

Example 1.3.4. *In Figure 1.5 (a) we are given a set of points in $\mathbb{R}^2$, with the chord lr labeled and the point p in S_1 that maximizes Δprl also labeled. In (b) we see the points interior to Δprl have been removed and the remainder of S_1 has been partitioned into S_3 and S_4. The next step of the recursion is also shown using S_3, with a new p selected to maximize triangle area. Removal of points interior to the new triangle leaves only a single point that we label S_5 and recurse on in (c). In (d) we see the resulting partial hull after S_3 has been processed entirely and edges have been merged.*

The analogy to Quicksort is the partitioning and recursion at each step, also if partitions are no larger than some constant fraction of the input set size at

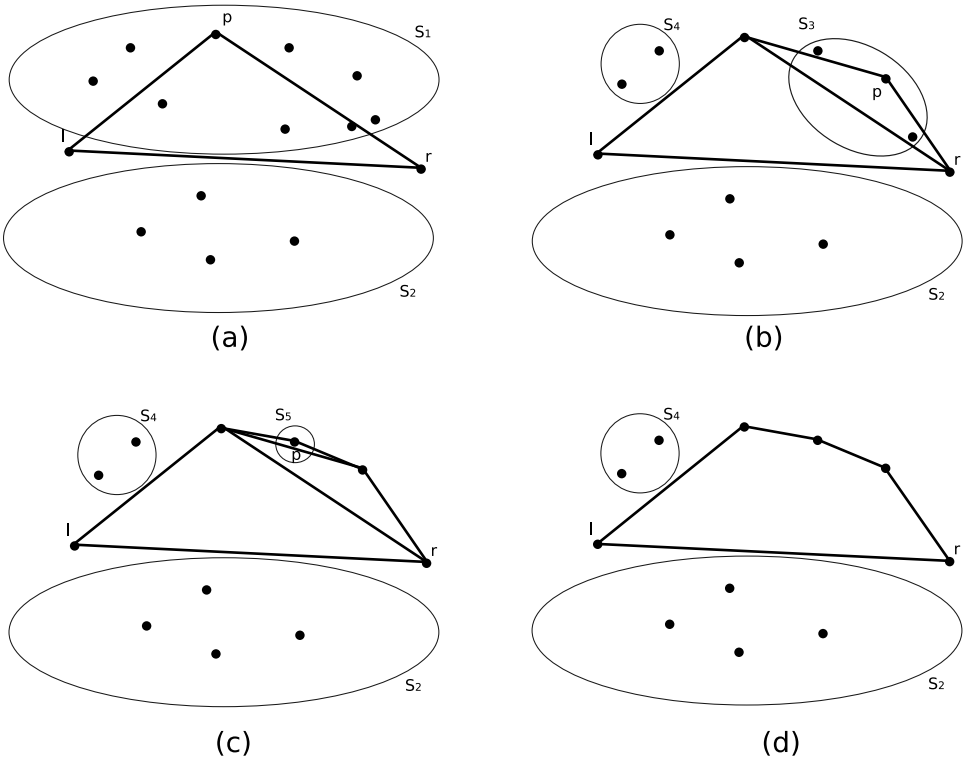

Figure 1.5: Quickhull Example.

each step we achieve $O(n \log n)$[21] time complexity. If this doesn't hold time complexity can be as bad as $O(n^2)$[21] like Quicksort.

1.4 Linear Constraint Classification

In this section we introduce and explore the concept of redundancy in sets of linear inequalities. We adopt definitions and terminology similar to that which is found in [7]. This combined with the polar dual will provide the basis for our probabalistic extreme point detection.

Let I be a set of indices with cardinality $|I|$, we define the corresponding constraint set as $C(I) = \left\{ a_i^\top x \leqslant b_i \mid i \in I \right\}$ where a_i and x are d-vectors. This set of constraints defines a possibly empty region in $\mathbb{R}^d$, $X(I) = \{x \in \mathbb{R}^d | a_i^\top x \leqslant b_i, \forall i \in I\}$. We call "$a_k^\top x \leqslant b_k$" the k-th constraint.

Definition 1.4.1 (Redundant Constraint)**.** *The k-th constraint is redundant in* $C(I)$ *if* $X(I) = X(I \setminus \{k\})$.

Definition 1.4.2 (Necessary Constraint)**.** *The k-th constraint is necessary in* $C(I)$ *if* $X(I) \neq X(I \setminus \{k\})$.

A Redundant Constraint is one that can be removed without changing the feasible region defined by the constraint set. A constraint that is not redundant is necessary and its removal will change the feasible region.

Definition 1.4.3 (Degenerate Extreme Point)**.** *We call* p *a degenerate extreme point of* $X(I)$ *if* p *is an extreme point and* $\left|\{i \in I \mid a_i^\top p = b_i\}\right| > d$.

We can classify a constraint as redundant or necessary by solving a linear optimization problem. Consider the linear optimization problem

$$\begin{aligned} \min \ & -a_k^\top x \\ \ni \ & a_j^T x \leq b_j \quad \forall i \in I,\ j \neq k \end{aligned} \tag{1.1}$$

.

If (1.1) is unbounded or has optimal solution x^* with $a_k^\top x^* > b_k$ then constraint k is necessary in $C(I)$. In fact if we can find a feasible $x \in X(I \setminus \{k\})$ with $a_k^\top x > b_k$ we can conclude constraint k is necessary. The constraint classification problem can be solved by solving the the family of linear optimization problems

$$LO_k : (k \in I) \tag{1.2}$$

$$\min \; -a_k^\top k$$

$$\ni \quad a_j^T x \leq b_j \quad \forall j \in I,\; j \neq k$$

.

In our context, given a point set $S = \{p_1, p_2, \ldots, p_n\}$ then $I = \{1, 2, \ldots, n\}$, $C(I) = \left\{p_i^\top x \leqslant 1 \mid i \in I\right\}$ and $X(I) = S^*$. We will use necessary in S^* as shorthand for necessary in $\left\{p_i^\top x \leqslant 1 \mid i \in I\right\}$, with similar notation for redundant and so on.

1.5 Hit-And-Run Algorithms

We now introduce the class of Hit-and-Run Algorithms and examine a couple of different variants. The first hit-and-run algorithm was introduced in 1979 by Boneh and Golan[4].

The Hit-and-Run Algorithms are a class markov processes. A typical iteration begins with an interior point in a set $S = \{p_1, p_2, \ldots p_n\}$ then, a random direction is selected using a probability distribution. This random direction and interior point define a feasible line in S which intersects the boundary of S in two places. A new interior point is chosen at random on the generated feasible line segment and becomes the interior point for the next iteration.

Example 1.5.1 (Hit-and-run example iteration)**.** *We consider the region in Figure 1.6 (a) and illustrate a single iteration. We first need an interior point*

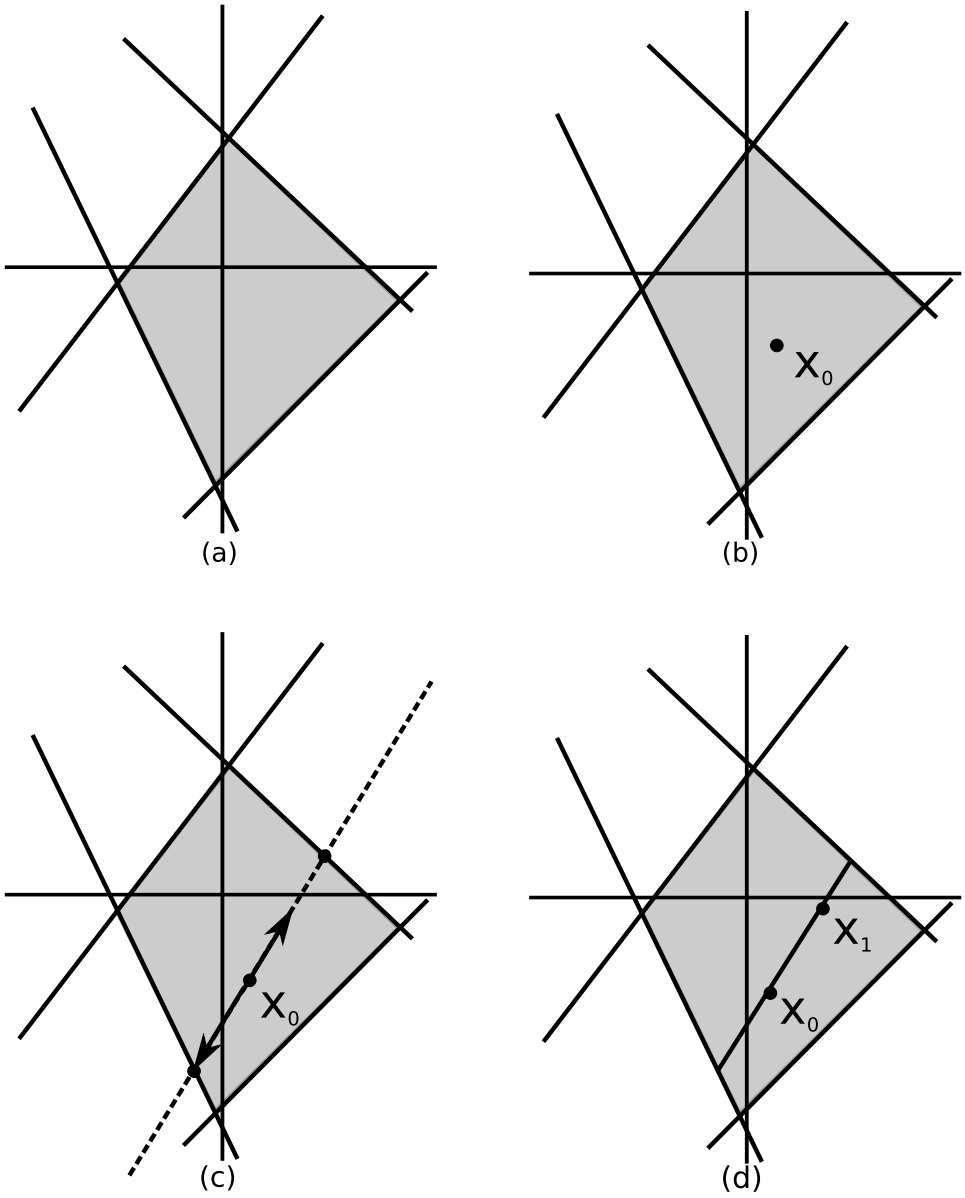

Figure 1.6: Example Hit-and-Run Iteration.

for this region in $\mathbb{R}^2$, we use the point labeled x_0 in Figure 1.6 (b). We next generate a direction and its negative, as shown by the arrows at x_0 in Figure 1.6 (c), the resulting line is drawn in as a dashed line. We see in Figure 1.6 (d) the resulting line segment and the new interior point x_1. The two constraints that determine the end points of the line segment are necessary.

1.5.1 Coordinate Direction Variant

The Coordinate Direction (CD) variant uses a random coordinate direction for the search direction and updates the current interior point with each iteration, using the origin as an initial point.

1.5.2 Computational Complexity of CD Iterations

CD Algorithm

0. Set $a_0 = 0$, $\mathcal{J} = \emptyset$, set $v_i = 0,\ \forall i \in I$.

1. Choose a random coordinate direction e_j.

1.1 Calculate possible step sizes, $\sigma_i = (b_i - v_i)/(p_i)_j\,(\forall i \in I\,(p_i)_j \neq 0)$.

1.2 Determine maximum feasible step size in the positive and negative e_j directions. That is find ℓ_1 and ℓ_2 so that $\sigma_{\ell_1} = \min_{i\in I}\{\sigma_i \mid (p_i)_j > 0\}$ and $\sigma_{\ell_2} = \max_{i\in I}\{\sigma_i \mid (p_i)_j < 0\}$.

2. Update detected constraint set by replacing $\mathcal{J}$ with $\mathcal{J} \cup \{\ell_1, \ell_2\}$.

2.1 Set $\sigma = u\sigma_{\ell_1} + (1-u)\sigma_{\ell_2}$, $u \sim$Uniform$(0,1)$, and update $v_i \leftarrow v_i + \sigma(p_i)_j\ \forall i \in I$. Return to Step 1 and repeat.

The set $\mathcal{J}$ contains indices corresponding to necessary constraints.

Theorem 1.5.1. *Each CD Iteration requires O(n) time.*

Proof. The selection of a random e_j requires $\mathrm{O}(\log d)$ time as we must set $\log_2 d$ bits to determine it. Calculating the σ_is in step 1.1 requires $O(n)$ time, as does calculating ℓ_1 and ℓ_2 in step 1.2. This results in a total cost of $O(n)$ as for Step

1 because $\log d < n$. Step 2 requires O(1) time. Therefore the cost of both Step 1 and Step 2 and thus of a CD iteration is O(n) time. □

1.5.3 Stand-and-Hit Variant

We now consider the Stand-and-Hit Variant (SH). This variant uses a fixed interior point (the origin) and random unit vectors for the search direction. Each iteration detects one necessary constraint, and unlike the CD variant when finding the intersections, the cost of calculating the inner products is linear in d instead of constant.

1.5.4 Computational Complexity of SH Iterations

SH Algorithm

The set $\mathcal{J}$ contains indices corresponding to necessary constraints.

0. Set $\mathcal{J} = \emptyset,\ f_i = 0,\ \forall i \in I$

1. Choose a random unit vector s as a search direction.

1.1 Find ℓ such that $\sigma = (p_\ell^\top s)^{-1} = \min_{i \in I}\{\, (p_i^\top s)^{-1} \mid p_i^\top s > 0\}$.

2. Replace $\mathcal{J}$ with $\mathcal{J} \cup \{\ell\}$ and f_ℓ with $f_\ell + 1$. Return to Step 1 and repeat.

Theorem 1.5.2. *Each SH Iteration requires O(nd) time.*

Proof. We observe that selecting a random vector v is linear in the dimension of the vector, as is calculating its length. Hence converting it to a unit vector thus calculating a random unit vector s requires $O(d)$ time. We observe calculating $(p_i^T s)^{-1}$ requires $O(d)$ time and it must be calculated for each $i \in I$ or n times.

Therefore calculating ℓ and σ requires $O(nd)$ time. Step 1 requires $O(d) + O(nd) = O(nd)$ time. Step 2 requires constant time for set insertion and a single addition, therefore an entire SH iteration requires $O(nd)$ time. $\square$

We now show how hit-and-run can be used to detect necessary constraints.

Theorem 1.5.3. *$p_k \in S$ is an extreme point if and only if $p_k^T x \leqslant 1$ is necessary in S^*.*

Proof. Suppose p_k is not an extreme point in S then $p_k = \lambda\alpha + (1-\lambda)\beta$ for some $\lambda \in [0,1]$ and $\alpha, \beta \in conv(K)$. However as $\alpha, \beta \in S$ we have $\alpha^T y \leqslant 1$ and $\beta^T y \leqslant 1$ for all $y \in S^*$ so that

$$p_k^\top y = \lambda(\alpha^T y) + (1-\lambda)(\beta^T y) \leqslant \lambda + (1-\lambda) = 1 \quad (\forall y \in K^*)$$

and $p_k^\top y \leqslant 1$ is redundant. This is the contrapositive of the desired result.

If $p_k^\top x \leqslant 1$ is redundant then there is an optimal solution $\hat{x}$ to the following Linear Optimization problem

$$\min\left\{-p_k^\top x \mid p_i^\top x \leqslant 1 \quad (\forall i \neq k)\right\}.$$

Let x^* be an optimal solution, the optimality conditions imply the existance of $u^* \geq 0$ with $\sum_{i \neq k} u_i^* p_i = p_k$. Therefore p_k is not an extreme point of K and we have the contrapositive of the desired result. $\square$

We can now use this result to identify extreme points in S using S^*.

Theorem 1.5.4. *If $\exists \hat{x} \in \mathbb{R}^d$ with $p_k^\top \hat{x} = 1$ and $p_i^\top \hat{x} < 1 (i \in I \setminus \{k\}$ then $p_k^\top x \leq 1$ is necessary in S^*.*

Proof. We observe that only the constraint corresponding to p_k is active at $\hat{x}$, if we were to multiply $\hat{x}$ by a constant $c > 1$ then $p_k^T(c\hat{x}) = c > 1$ so $c\hat{x} \notin S^*$. We claim that removing the constraint $p_k^T x < 1$ changes the region, let $c_0 = \min\left\{\left|\frac{1}{2p_i^T\hat{x}}\right| \mid i \in I\backslash\{k\}, p_i^T\hat{x} \neq 0\right\} \cup \left\{\frac{1}{2}\right\}$. by construction of c_0, $p_i^T(c_0\hat{x}) < 1 (i \in I\backslash\{k\}$ and $c_0 > 1)$. Therefore, $c_0\hat{x}$ while not in S^* would be in the region with the constraint $p_k^T x \leqslant 1$ removed, hence $p_k^T x \leqslant 1$ is a necessary constraint and p_k is extreme in S as required. □

This Theorem allows for the application of hit-and-run to the constraint classification problem. We need only find a point where one constraint is active to detect a necessary constraint, the probability of hit-and-run hitting any single point, and hence any extreme point of the region is 0.

Chapter 2

The Algorithm of Jibrin, Boneh and Caron

2.1 Introduction

In this section we introduce and analyze the probabalistic extreme point identification algorithm of Jibrin, Boneh and Caron[18]. Their algorithm takes advantageof the key relationship between the extreme points in S and the necessary constraints in S^*. We also explore their preprocessing methodology.

2.2 The Algorithm of Jibrin, Boneh and Caron

We now examine the algorithm of Jibrin, Boneh and Caron. As input the algorithm takes a point set $S \subset \mathbb{R}^d$ of cardinality $n \geq d+1$ such that any $d+1$ elements of S do not lie on a $(d-1)$-hyperplane.

1.0 Calculate B_S, the barycentre of S.

2.0 Find $\hat{S}$, the set S translated by $-B_S$

3.0 Set p to the origin.

3.1 Perform a hit-and-run iteration in $\hat{S}^*$ with p as the known interior point.

3.2 Label the point in S corresponding to the detected constraint(s) as necessary.

3.3 Update p as required.

3.4 If the stopping rule is not met go to 3.1.

4.0 The set of labeled elements in S form a subset of the extreme points of S.

As this is a probabalistic algorithm it is possible that the list of extreme points generated will be incomplete.

2.3 Preprocessing Application

In some problems, for example determining the convex hull of a set S, only the extreme points have bearing on the solution. Hit-and-Run provides a fast method for detecting a subset $\bar{S}$ of the extreme points of S. We also know that $\text{conv}(S) \supseteq \text{conv}(\bar{S})$ so points interior to $\text{conv}(\bar{S})$ must also be interior to $\text{conv}(S)$. Determining the points interior to $\text{conv}(\bar{S})$ allows the use of Hit-and-Run as a preprocessor for problems interested only with the extreme points.

In [18] a subset of $\bar{S}$ with cardinality $d+5$ is selected, the convex hull of this subset is then found using quickhull. The points in $S \setminus \bar{S}$ are then tested for membership in this convex hull, with points in the hull discarded.

2.4 Conclusions

The algorithm of Jibrin, Boneh and Caron has one immediate shortcoming and that is generality. The requirement that any set of $d+1$ points are non-coplanar guarantees every input set has full dimension, but it also precludes sets with three collinear points from being processed. In the next chapter we will remove this requirement and generalize the algorithm to arbitrary finite point sets in $\mathbb{R}^d$.

The preprocessor application can also be improved. Scalability is questionable as the performance of the quickhull algorithm degrades for $d > 9$[22], the work around for this in [18] results in possible wasted effort when more than $d+5$ extreme points are detected via Hit-and-Run. In chapter 4 we propose an alternate approach to the preprocessing step.

Chapter 3

Improvements to Jibrin, Boneh and Caron

3.1 Introduction

In this chapter we seek to augment the Jibrin, Boneh and Caron algorithm to allow the processing of general finite point sets in $\mathbb{R}^d$. We remove the requirement that any subset $\tilde{S}$ of the input set S with $|\tilde{S}| = d + 1$ do not lie on a $(d-1)$-hyperplane (are non-coplanar). We will refer to this requirement as *The Non-Coplanarity Condition*.

3.2 Coplanarity, Degeneracy and Dimensionality

In this section we will explore the implications of the non-coplanarity condition, namely that any S satisfying the non-coplanarity condition has full dimension and the extreme points of S^* are non degenerate.

3.2.1 Dimensionality

The most obvious implication of the non-coplanarity condition is that the convex hull of any $K \subset S$ with $|K| = d+1$ has full dimension as it must be a d-simplex.

Lemma 3.2.1. *Given a set $S \subset \mathbb{R}^d$ that satisfies the Non-Coplanarity Condition and $K \subset S$ with $|K| = d+1$ then* conv(K) *is a d-simplex.*

Proof. Let $\tilde{S} \subset S$ with $|\tilde{S}| = d+1$. If $\tilde{S}$ has full dimension it is a d-simplex and we are done. Suppose $\tilde{S}$ does not have full dimension, then we can find a $(d-1)-$*hyperplane* containing $\tilde{S}$, contradicting the Non-Coplanarity condition. Therefore $\tilde{S}$ must be a full dimensional set, and hence a d-simplex. □

As a d-simplex has non-empty interior and S has a subset with cardinality $d+1$ the convex hull of S must be non empty. Thus the convex hull of any set of cardinality $d+1$ or greater that meets the non-coplanarity condition is non empty.

Theorem 3.2.1. *If S satisfies the non-coplanarity condition then the extreme points of* conv(S^*) *are non degenerate.*

Proof. Suppose there is an extreme point $\hat{p}$ of S^* that is degenerate, that is there are $d+1$ or more active constraints at $\hat{p}$. However the set of active constraints at a point in the S^* determines a set of points in a $(d-1)$-hyperplane in S[15]. The existance of such an $\hat{p}$ thus leads to an immediate contradiction as S satisfies the non coplanarity condition. □

Example 3.2.1. *In Figure 3.1 (a) we are given a simplicial facet F of the convex hull of a point set in $\mathbb{R}^3$ with its three extreme points labeled. The dual*

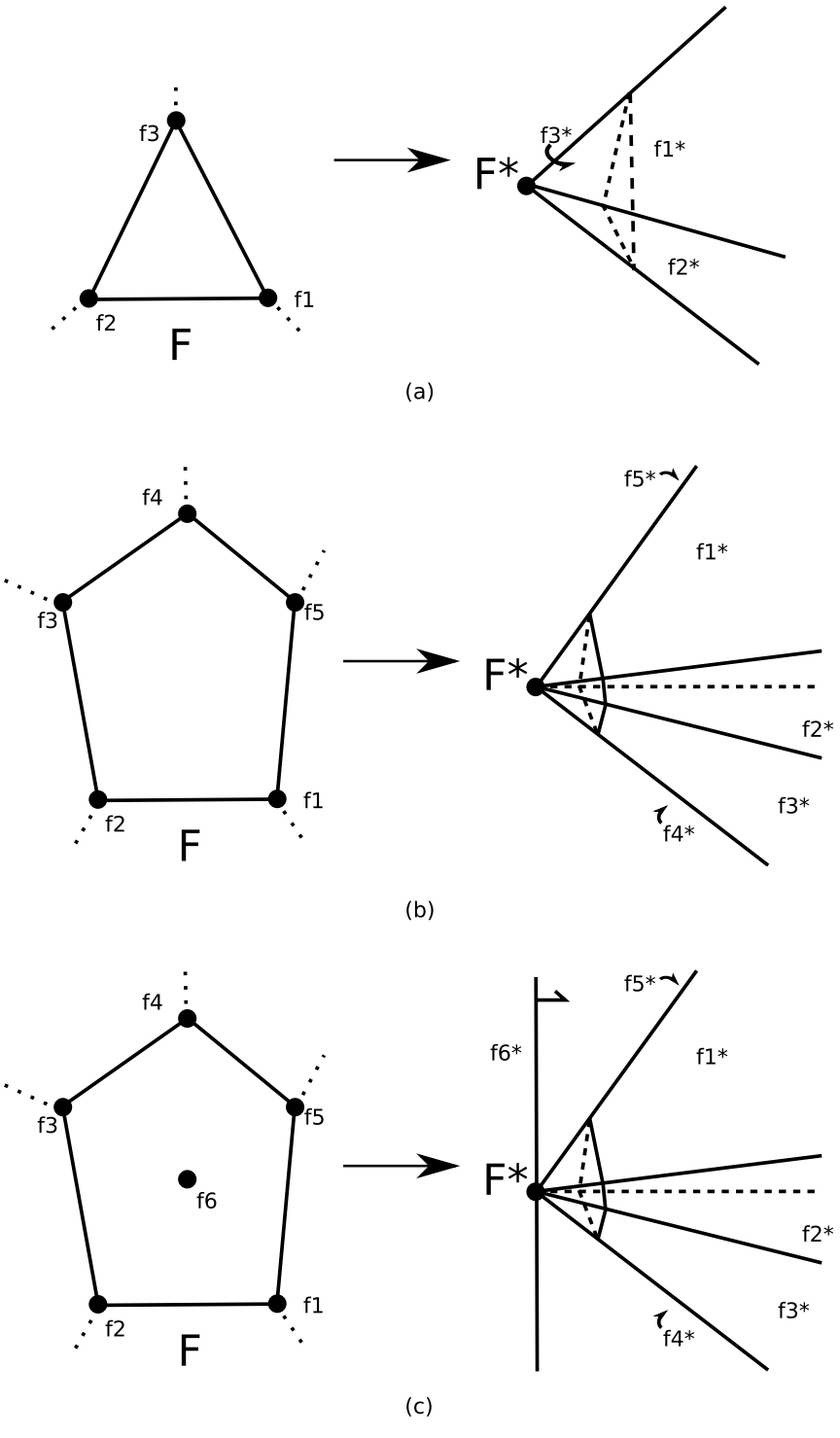

Figure 3.1: Facets and their duals.

of this is the extreme point F^* with exactly three active constraints, namely those corresponding to the extreme points of F. In the case of (a) we observe there

are no redundant constraints and F^* is non degenerate. In (b) We have a non simplicial facet, however all members of the point set incident on the facet are extreme points. The dual of this facet is a degenerate extreme point, but there are still no redundant constraints as shown. In (c) we have a non simplicial facet with an incident non extreme point labeled f6*. The dual of this facet is a degenerate extreme point, moreover the constraint* f6* *is redundant.*

The non-coplanarity condition guarantees S has full dimension and S^* has no degenrate extreme points. If we require input to have full dimension and handle degeneracy in S^* we can remove the non-coplanarity condition. Degeneracy in S^* is an issue when multiple constraints are detected active at a single point, in such a case we no longer have the guarantee that all active constraints are necessary in S^*. One solution to this problem is to ignore iterations where this occurs, this is the solution we adopt. We have thus reduced the Non-Coplanarity Condition to the requirement of full dimensional input.

We have removed the non-coplanarity condition, but we have replaced it with a weaker condition requiring full dimensionality. We eliminate this condition.

Suppose we are given an input set $S \subset \mathbb{R}^d$ that may not be full dimensional. Let P be the matrix with columns from S, if $\text{rank}(P) = d$ we have a full dimensional set. If $\text{rank}(P) < d$ we consider the smallest $k < d$ such that S can be embedded in a k-hyperplane.

We begin by performing a translation that places the barycentre of S at the origin, we label this translated set $\hat{S}$. The k-hyperplane containing $\hat{S}$ will be a subspace of $\mathbb{R}^d$. We now compute a basis β for this subspace and re-express the

elements of $\hat{S}$ as linear combinations of the elements of β. The coefficients of these expressions are elements of $\mathbb{R}^k$, we denote the the set of these coefficients as S_R. Letting B be the matrix with columns from β we show that detecting the extreme points of S_R is equivalent to detecting the extreme points of $\hat{S}$ and hence S.

Theorem 3.2.2. *For every $a \in \text{conv}(\hat{S})$ there exists $a_R \in \text{conv}(S_R)$ with $Ba_R = a$.*

Proof. Let $a \in conv(\hat{S})$ then $a = \sum_{i=1}^{n} \lambda_i p_i = \sum_{i=1}^{n} \lambda_i B x_i = B(\sum_{i=1}^{n} \lambda_i x_i) = Ba_R$ □

Theorem 3.2.3. *For every $a_R \in \text{conv}(S_R)$ then there exists $a \in \text{conv}(\hat{S})$ with $Ba_R = a$.*

Proof. Let $a \in \text{conv}(\hat{S})$ then $a_R = \sum_{i=1}^{n} \lambda_i x_i$ so $Ba_R = \sum_{i=1}^{n} \lambda_i B x_i = \sum_{i=1}^{n} \lambda_i p_i = a$. □

Theorem 3.2.4. *The point $(p_i)_R \in S_R$ is extreme in S_R if and only if p_i is extreme in $\hat{S}$.*

Proof. Let $(p_i)_R$ be an extreme point in S_R and suppose p_i is not an extreme point of $\hat{S}$. Then we can find a and b in $\text{conv}(\hat{S})$ with $\lambda a + (1-\lambda)b = p_i$ for some $\lambda \in (0,1)$. By the preceding theorem $\exists a_R, b_R \in \text{conv}(S_R)$ such that $Ba_R = a$ and $Bb_R = b$. This results in $B(\lambda a_R + (1-\lambda)b_R) = \lambda B a_R + (1-\lambda) B b_R = p_i = B(p_i)_R$ so $\lambda a_R + (1-\lambda)b_R = (p_i)_R$, contradicting that $(p_i)_R$ was extreme. Therefore p_i is extreme in $\hat{S}$.

We now let p_i be an extreme point in $\hat{S}$ and suppose $(p_i)_R$ is not an extreme point of S_R. Then we can find a_R and b_R in $\text{conv}(X_R)$ with $\lambda a_R + (1-\lambda)b_R = (p_i)_R$ for some $\lambda \in (0,1)$. By the preceding theorem $\exists a, b \in \text{conv}(\hat{S})$ such that $Ba_R = a$ and $Bb_R = b$. This results in $B(\lambda a_R + (1-\lambda)b_R) = \lambda Ba_R + (1-\lambda)Bb_R v = \lambda a + (1-\lambda)b = p_i$, contradicting that p_i was extreme. Therefore $(p_i)_R$ is extreme in S_R. □

By checking the rank of P determining a basis β for $\hat{S}$ if the rank is less than d and re-expressing S in terms of β we guarantee a full dimensional representation of the problem. The rank check and basis determination and re-expression can be computed easily using Gauss Jordan elimination at a cost of $\text{O}(n^3)$ time, or using more numerically stable methods like LU or QR decompositions.

3.3 Conclusion

We have generalized the algorithm of Jibrin, Boneh and Caron to arbitrary finite point sets in $\mathbb{R}^d$. If Gauss-Jordan elimination is used the cost of this step is $\text{O}(n^3)$ time. It is possible that this step could dominate total time required as running $n \log n$ CD iterations requires $\text{O}(n^2 \log n)$ time. In the samples tested so far this has not proven to be an issue.

Chapter 4

A New Preprocessor

4.1 Introduction

We now use the extreme points detected by hit-and-run $\bar{S}$ to expunge some of the interior points of conv(S). We perform this removal by reformulating the problem of testing $p_i \in \text{conv}(\bar{S})$ as a feasibility problem.

4.2 The Preprocessor

Now that we know a subset $\bar{S}$ of the extreme points of $conv(S)$ we wish to take advantage of this. For some problems the interior points of $conv(S)$ have no bearing on the final result and hence can be removed before solving. The interior of $conv(\bar{S})$ is a subset of the interior of $conv(S)$. In this section we explore a pair of methods for removing points in the interior of $conv(\bar{S})$.

In the Jibrin Boneh and Caron paper the method presented requires determining the convex hull of $\bar{S}$ using Quickhull then checking membership of the resulting hull. While this approach is straightforward and relatively efficient for $d < 9$ [22], performance degrades for higher dimensions. Indeed even the best

knwn algorithm can require time exponential in d [2]. This algorithm requires O(ndv) time where v is the number of facets in the convex hull. One possible method for bounding the performance is to fix the size of the considered set of extreme points relative to d, this size is set at $d+5$ in [18]. Besides the obvious wasted effort finding any more than $d+5$ this also relies on the full dimension aspect of the non-coplanarity condition.

We know that $a \in conv(\bar{S})$ if and only if there exist $\lambda_1 \dots \lambda_k$ such that there exist $\lambda_1 \dots \lambda_k$ such that

$$\begin{aligned} \sum_{i=1}^{k} \lambda_i \bar{p_i} &= a, \\ \sum_{i=1}^{k} \lambda_i &= 1, \\ 0 \leq \lambda_i \leq 1, &\forall i \in \{1 \dots k\}. \end{aligned} \tag{4.1}$$

This is a linear feasibility problem.

Let $P = [\bar{p_1} \dots \bar{p_k}]$, $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k)^\top$, e be a vector of ones and $\vec{0}$ be a vector of zeros. The system (4.1) can be rewritten as

$$\begin{aligned} P\lambda &= a \\ e^\top \lambda &= 1, \\ \vec{0} \leq \lambda &\leq e. \end{aligned} \tag{4.2}$$

Setting

$$A_0 = \begin{bmatrix} P \\ e^\top \end{bmatrix} \quad b_0 = \begin{pmatrix} a \\ 1 \end{pmatrix} \tag{4.3}$$

we can rewrite system (4.1) as

$$\begin{aligned} A_0\lambda &= b_0 \\ \vec{0} \leq \lambda &\leq e. \end{aligned} \tag{4.4}$$

Theorem 4.2.1. *There is a solution to $A_0\lambda = b_0$ and $\vec{0} \leq \lambda \leq e$ if and only if the optimal solution λ^* to*

$$\begin{aligned} \max\ & f^\top \lambda \\ \ni\ & A_0\lambda \leq b_0, \\ & \lambda \leq e, \\ & \lambda \geq \vec{0}. \end{aligned} \tag{4.5}$$

is such that $f^\top \lambda^* = e^\top b_0$ where $f = e^\top A_0$.

Proof. See [3]. □

We test membership of a in $\text{conv}(\bar{S})$ by solving (4.5). Letting $b = f$, $u = \lambda$

$$A^\top = \begin{bmatrix} A_0 \\ I \end{bmatrix} \quad c_a = \begin{pmatrix} b_0 \\ e \end{pmatrix}$$

we can test for membership by solving

$$\begin{aligned} \max\ & b^\top u \\ \ni\ & A^\top u \leq c_a, \\ & u \geq 0. \end{aligned} \tag{4.6}$$

We now consider the dual of (4.6), that is

$$\begin{aligned} \min\ & c_a^\top x \\ \ni\ & Ax \geq b, \\ & x \geq 0. \end{aligned} \tag{4.7}$$

This linear program reminds us of (1.1) which was used to identify necessary and redundant constraints. Specifically if the solution x^* is such that $c_a^\top x^* \leq b_0^\top e = a^\top e + 1$. We observe that for any x^* that is optimal in (4.7) we have $c_a^\top x \leq a^\top e + 1$ by weak duality, as we know the optimal objective function value is no larger than $a^\top e + 1$ by the construction of f in theorem 4.4.1. Let x^* be an optimal solution to (4.7) then either $c_a^\top x^* = a^\top e + 1$ and $a \in \text{conv}(\bar{S})$, or $c_a^\top x^* < a^\top e + 1$ and $a \notin \text{conv}(\bar{S})$. In the first case a is redundant and can be removed, the second case is indeterminate as a may be extreme.

When considering the family of objective functions corresponding to $S \backslash \bar{S}$ the LP in (4.7) is reminiscent of CLASFY[8, 9] except in the context of CLASFY the objective function also served as a constraint for the other LPs to be solved. In this context we must adapt CLASFY to simultaneously solve a family of constraints over a fixed feasible region. We solve the problem

$$\begin{aligned} \min\ & c_{p_i}^\top x \qquad \forall i \in I \\ \ni\ & Ax \geq b \\ & x \geq 0 \end{aligned} \tag{4.8}$$

where I is the set of indices corresponding to those points in S not classified as extreme by the hit-and-run step.

4.3 Computational Cost

We now analyze the cost of transforming the problem into a system of linear inequalities. To find the vector f we must add $|\bar{S}| \leq n$ d-vectors, this has worst case time complexity $\mathrm{O}(nd)$ and requires $\mathrm{O}(n)$ space. The matrix A is a $(k+1) \times (d+2k+2)$ matrix. As $d < n$ and $k \leq n$, A is no larger than $(n+1) \times (3n+2) < (2n) \times (4n)$ or $8n^2$ elements so generating A requires $\mathrm{O}(n^2)$ space and has $\mathrm{O}(n^2)$ time complexity. To find the constraint corresponding to a given a we must construct c_a and calculate $a^\top e + 1$. The vector c_a is a $(d+2k+2)$-vector so construction can be done in $\mathrm{O}(n)$ time using $\mathrm{O}(n)$ space. Computing $a^\top e + 1$ is an inner product of d-vectors and hence uses $\mathrm{O}(d)$=$\mathrm{O}(n)$ time and $\mathrm{O}(1)$ space. We have up to n such constraints to generate so the worst case cost for constraint generation is $\mathrm{O}(n^2)$. Total worst case time complexity for set up is thus $\mathrm{O}(nd)$ + $\mathrm{O}(n^2)$ + $\mathrm{O}(n)$ + $\mathrm{O}(n^2)$ = $\mathrm{O}(n^2)$, and worst case storage use is $\mathrm{O}(n^2)$.

Solving the $n-k =)(n)$ linear programs in (4.8) can be done in polynomial[1] time, In particular L is approximately $k + 1 * d + k + 1 = \mathrm{O}(n^2)$, so our upper bound is $\mathrm{O}(n^6/\ln n)$.

4.4 Further Improvements

We have achieved theoretical polynomial time complexity. We may be able to reduce practical running time further by using a strategy similar to CLASFY[8, 9], at the cost of the polynomial upper bound on time complexity[19]. We have an upper bound on the optimal value of the objective function for each objective

in (4.8)

0.0.0 Find $x_0 \geq 0$ satisfying $Ax \geq b$

0.1.0 For each $i \in I$ test for optimality of $\min c_{p_i}$ at x_0.

0.2.0 For each optimal $\min c_{p_i}$, check $c_{p_i}^\top x_0 = p_i^\top e + 1$.

0.2.1 If equal remove i from I and classify p_i as redundant.

0.2.2 If $c_{p_i}^\top x_0 < p_i^\top + 1$ remove i from I and classify p_i as indeterminate.

0.3.0 Set $j = 0$

1.0.0 Set O_s to c_{p_i} for some $i \in I$ and set $G_s = p_i^\top e + 1$. If I is empty, terminate.

2.0.0 Calculate a search direction s_j and step size σ_j using O_s as the objective to be minimized.

3.0.0 $x_{j+1} = \sigma_j s_j + x_j$

3.1.0 For each $i \in I$ test for optimality of $\min c_{p_i}$ at x_{j+1}.

3.2.0 For each optimal $\min c_{p_i}$, check $c_{p_i}^\top x_{j+1} = p_i^\top + 1$.

3.2.1 If equal remove i from I and classify p_i as redundant.

3.2.2 If $c_{p_i}^\top x_{j+1} < p_i^\top + 1$ remove i from I and classify p_i as indeterminate.

4.0.0 If O_s is optimal at x_{j+1} go to (1.0.0).

5.0.0 Go to (2.0.0).

We define $\breve{S} = \{p_i \in S \mid \ p_i \in \bar{S} \vee p_i$ was classified as indeterminate$\}$, equivalently this is $S \backslash \text{conv}(\bar{S})$.

4.5 Conclusion

In this chapter we have proposed an alternate preprocessor that uses all points detected by the Hit-and-Run step. Moreover this method avoids the exponential cost for higher dimension quickhull.

Chapter 5

Experimental Results

5.1 Introduction

In this section we discuss an implementation of the revised algorithm and analyze the performance and scaling of this implementation.

5.2 Technical Detail

The implementation of the revised Hit-and-Run sampler was done in C, using the Stand-and-Hit variant. For generating Pseudo-Random integer and Uniform [0,1] values an implementation of Mersenne Twister[20] seeded by system time was used. Normal(0,1) values were generated using the Box Muller[5] transformation. The implementation of the preprocessor was also done in C. The family of Linear Programs (4.8) was solved sequentially using lpsolve version 5.5.0.10, this solver is based on the simplex method and as such may violate our polynomial time bound. All tests were performed on a server with 2GB of RAM and dual Pentium 4 Xeon processors running at 2.2GHz.

5.3 Results

In this section we show the effectiveness of the new preprocessor and explore how well it scales as d, $|\bar{S}|$ and n increase. The distribution sampled when generating an example varies based on the example. Some examples are sampled uniformly from a d dimensional hypercube centered at the origin, some are based on d-tuples of Normal$(0, 1)$ samples, most however are a 1:2 mix of points chosen uniformly over $[-4, 4]^d$ and points with normal$(0, 1)$ coordinates. We will label these as Uniform, Normal and Mix respectively.

We begin by showing preprocessor performance on examples taken from all 3 distributions. We generate one example with $n = 5000$ for each of the three distributions and each dimension $d \in \{2, 3, 6, 9\}$.

Table 5.1: Preprocessor Performance by Distribution

Ex.	d	n	Dist.	$\lvert S\rvert$	|int|	|ind|	T_1	T_2	T_3
1	2	5000	Normal	14	4986	0	0.000	23.760	3.130
2	2	5000	Uniform	26	4974	0	0.000	23.840	4.390
3	2	5000	Mix	20	4980	0	0.000	23.810	3.600
4	3	5000	Normal	48	4952	0	0.000	26.390	7.460
5	3	5000	Uniform	89	4887	24	0.000	26.440	11.980
6	3	5000	Mix	69	4927	4	0.000	26.440	9.640
7	6	5000	Normal	357	4482	161	0.000	33.830	67.770
8	6	5000	Uniform	689	3687	624	0.000	33.820	132.420
9	6	5000	Mix	434	4322	244	0.010	33.720	82.250
10	9	5000	Normal	850	3263	887	0.010	41.350	229.090
11	9	5000	Uniform	1605	1653	1742	0.010	41.170	430.880
12	9	5000	Mix	866	3648	486	0.010	41.250	209.480

Table 5.1 shows the result of these 12 examples. The columns give the example number, the dimension d, the number of points n, the distribution of the points, the number of extreme points detected by $n \log n$ hit-and-run iterations $|\bar{S}|$, the number of points determined to be interior |int|, the number of points for which the classification is unknown |ind|, the time required for

dimensionality testing and reduction T_1, the time required for $n \log n$ stand and hit iterations T_2 and the time required to solve the family (4.8) of linear programs T_3.

We observe that for almost all sample sizes and dimensions the normally distributed examples have the fewest detected extreme points and the lowest times required for the solve step.

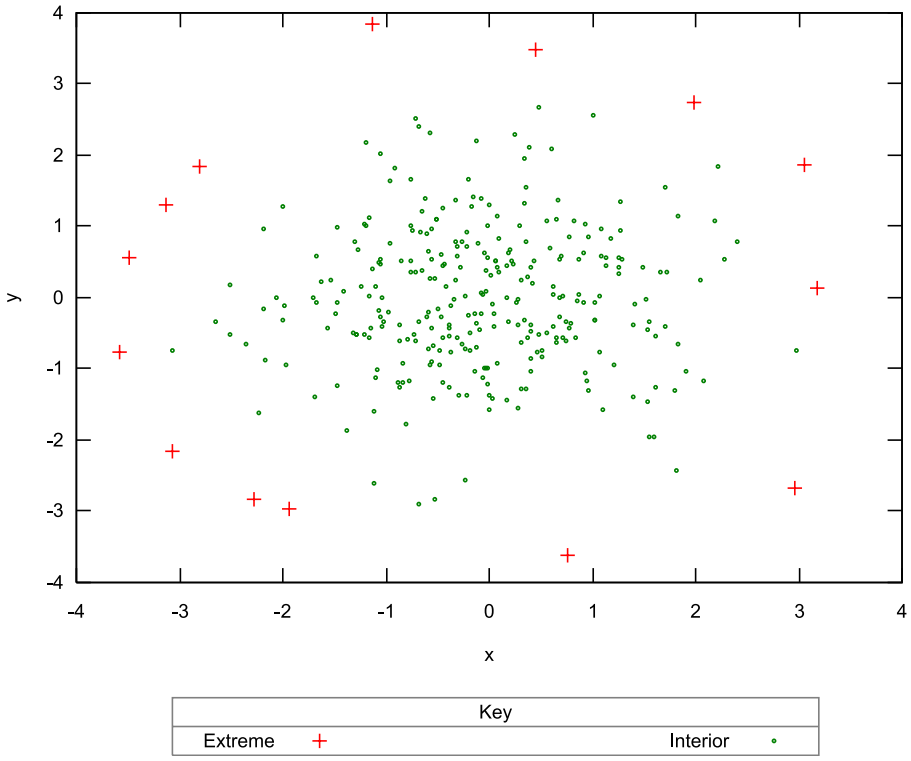

Figure 5.1: Normally Distributed Example Problem.

Example 5.3.1. *In Figure 5.1 we see the results of Example 1 from Table 5.1 with the interior points limited to 300 for the sake of readability. The vast majority of the points are clustered near the origin.*

On the other end of the spectrum the uniform examples have the most extreme points and longest time required for solving.

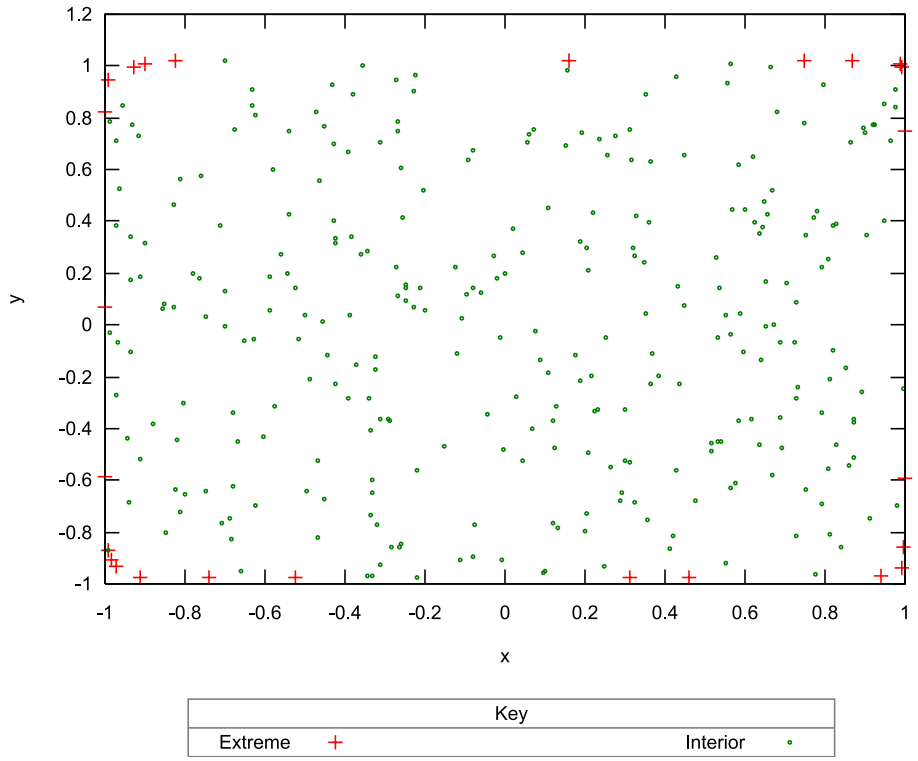

Figure 5.2: Uniformly Distributed Example Problem.

Example 5.3.2. *In Figure 5.2 we see the results of Example 2 from Table 5.1 with the interior points limited to 300 for the sake of readability. In this case the extreme points tend to cluster around the corners of* $[-1, 1]^2$.

For the remainder of the examples we will use the mixed distribution to maintain quick performance but still have relatively large sets of detected extreme points.

Example 5.3.3. *In Figure 5.3 we see the results of Example 3 from Table 5.1 with the interior points limited to 300 for the sake of readability. In this case the extreme points tend to cluster around the corners of* $[-4, 4]^2$. *The interior points cluster around the origin yielding fewer extreme points than a pure uniform sample, and more than a pure normal sample.*

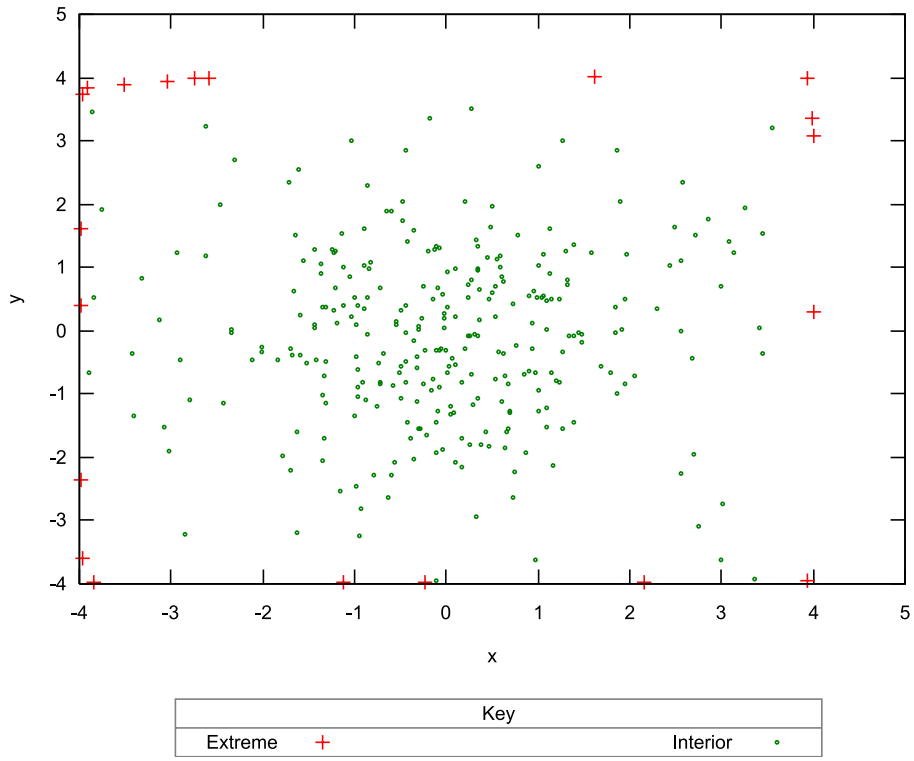

Figure 5.3: Mixed Distribution Example Problem.

We now examine how the preprocessor scales as d and n are increased. We use the preprocessor on twelve examples with varying dimensions and sample sizes. The results organized in table 5.2 use the same column labeling as table 5.1. We observe that the time for the hit-and-run step seems to grow with the size of n and the solve step is much more sensitive to increases in $|\bar{S}|$. The dimension of the example problems also seem to have little bearing on the time required for solving (4.8), other than the increase in extreme points due to the increase in d. One possible reason for this is that when d is increased by 1 the number of variables in (4.8) increases by 1, however when $|\bar{S}|$ is increased by one we increase the number of variables and constraints in (4.8) by 1.

Table 5.2: Preprocessor Scaling

Ex.	d	n	Dist.	$\|S\|$	$\|\text{int}\|$	$\|\text{ind}\|$	T_1	T_2	T_3
1	2	1000	Mix	13	987	0	0.000	0.770	0.580
2	2	5000	Mix	20	4980	0	0.000	23.840	3.700
3	2	10000	Mix	21	9979	0	0.000	102.910	7.330
4	3	1000	Mix	54	945	1	0.000	0.870	1.560
5	3	5000	Mix	61	4928	11	0.000	26.380	8.670
6	3	10000	Mix	93	9793	114	0.000	114.290	26.970
7	6	1000	Mix	167	761	72	0.000	1.100	5.450
8	6	5000	Mix	417	4337	246	0.010	33.720	75.050
9	9	1000	Mix	248	678	74	0.000	1.340	10.300
10	9	5000	Mix	877	3657	466	0.010	41.200	226.440
11	12	1000	Mix	282	642	76	0.000	1.580	15.070
12	12	5000	Mix	1150	3392	458	0.020	50.750	367.320

5.4 Comparison to Brute Force

We now compare the performance of the new preprocessor to the time required to determine interior points using the family of linear programs (1.1).

Table 5.3: New Preprocessor vs. Brute Force

Ex.	d	n	Dist.	r_{new}	c_{new}	p_{new}	r_{bf}	c_{bf}	p_{bf}	T_{new}	T_{bf}
1	2	1000	Mix	13	16	987	1000	2	1000	1.300	7.870
2	2	5000	Mix	20	23	4980	5000	2	5000	26.240	198.980
3	2	10000	Mix	21	24	9979	10000	2	10000	105.190	797.180
4	3	1000	Mix	54	58	946	1000	3	1000	2.390	12.380
5	4	1000	Mix	88	93	912	1000	4	1000	3.430	13.990
6	5	1000	Mix	125	131	875	1000	5	1000	4.770	16.600
7	6	1000	Mix	163	170	837	1000	6	1000	6.530	18.970
8	7	1000	Mix	200	208	800	1000	7	1000	8.580	22.310
9	9	1000	Mix	241	251	759	1000	9	1000	11.320	28.390
10	25	1000	Mix	329	355	671	1000	25	1000	51.910	88.410

Table 5.3 shows the result of 10 examples. The columns give the example number, the dimension d, the number of points n, the distribution of the points, the number constraints r_{new}, the number of variables c_new and the number of linear programs solved by the new preprocessor p_{new}. The number of rows, columns and problems solved by the brute force method are labeled r_{bf}, c_{bf} and p_{bf} respectively. The time for all steps required in the new preprocessor is

labeled T_{new} and the time required to solve the brute force problems is labeled T_{bf}.

We observe that in all the examples the new preprocessor outperforms the brute force method, in particular in cases with large set size and few extreme points. In example three the new preprocessor takes 87.5% of the time required for brute force, a savings of over 11 minutes. We observe that the only example where the time savings using the new preprocessor was less than 60% was example 10, where the number of extreme points detected was large relative to the size of n.

Chapter 6

Conclusions

In this thesis we have commented on and generalized the work of Jibrin, Boneh and Caron[18]. We have also introduced a new preprocessor.

We have relaxed the constraints on the input set significantly, from finite point sets satisfying the non-coplanarity condition to any finite set in $\mathbb{R}^d$. This generalization incurs a seemingly trivial additional computation cost.

We have designed a new preprocessor that scales well with dimension and set size. We have explored the performance of this preprocessor with varying problem sizes and dimensions, and contrasted it to the performance of the brute force method of redundancy checking.

Further work should be aimed at refining the preprocessor as the implementation discussed does not exploit all that we know about the reformulated problem. In particular a simultaneous solving strategy may further improve the performance of the preprocessor. The formulation of the new preprocessor also lends itself to easy parallelizability which may lead to further performance gains.

Bibliography

[1] Anstreicher, K. M. Linear programming in $o((n^3/\ln n)l)$ operations. *SIAM Journal on Optimization 9* (1999), 803–812.

[2] Avis, D., and Fukuda, K. A pivoting algorithm for convex hulls and vertex enumeration of arrangements of polyhedra. *Discrete and Computational Geometry 8* (1992), 295–313.

[3] Best, M. J., and Ritter, K. *Linear programming: active set analysis and computer programs*, first ed. Prentice-Hall, New Jersey, 1985.

[4] Boneh, A., and Golan, A. Constraints' redundancy and feasible region boundedness by random feasible point generator (rfpg). In *Presented at the Third European Con- gress on Operations Research (EURO III)* (1979)).

[5] Box, G. E. P., and Muller, M. E. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics 29* (1958), 610–611.

[6] Bykat, A. Convex hull of a finite set in two dimensions. *Information Processing Letters 7* (1978), 296–298.

[7] Caron, R. J., Boneh, A., and Boneh, S. Redundancy. *International Series in Operations Research and Management Science 6* (1997), 13.1 – 13.41.

[8] Caron, R. J., McDonald, J. F., and Ponic, C. M. Clasfy: A fortran iv subroutine to classify linear constraints as redundant or necessary. Tech. Rep. WMR 85-08, Department of Mathematics, University of Windsor, 1985.

[9] Caron, R. J., McDonald, J. F., and Ponic, C. M. A degenerate extreme point strategy for the classification of linear constraints as redundant or necessary. Tech. Rep. WMR 85-09, Department of Mathematics, University of Windsor, 1985. Revised 1988.

[10] Chan, T. M. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete and Computational Geometry 16* (1996), 361–368.

[11] Chand, D. R., and Kapur, S. S. An algorithm for convex polytopes. *Journal of the ACM 17* (1970), 78 – 86.

[12] Eddy, W. A new convex hull algorithm for planar sets. *ACM Transactions on Mathematical Software 3* (1977), 398–403.

[13] Graham, R. L. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters 1* (1972), 132–133.

[14] Green, P. J., and Silverman, B. W. Constructing the convex hull of a set of points in the plane. *The Computer Journal 22* (1979), 262–266.

[15] Grünbaum, B. *Convex Polytopes*, second ed. Springer, New York, 2003.

[16] Hoare, C. A. R. A new convex hull algorithm for planar sets. *The Computer Journal 5* (1962), 10–16.

[17] Jarvis, R. A. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters* (1973), 18–21.

[18] Jibrin, S., Boneh, A., and Caron, R. Probabalistic algorithms for extreme point identification. *Journal of Interdisciplinary Mathematics 10* (2007), 131–142.

[19] Klee, V., and Minty, G. J. How good is the simplex algorithm? *Inequalities 3* (1972), 159–175.

[20] Matsumoto, M., and Nishimura, T. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation 8* (1998), 3–30.

[21] Preparata, F. P., and Shamos, M. I. *Computational Geometry, an Introduction*, first ed. Springer-Verlag, New York, 1985.

[22] Skiena, S. S. *The Algorithm Design Manual*, first ed. Springer-Verlag, New York, 1997.

Vita Auctoris

Mr. Adam J. Hartfiel was born in 1982, in Montreal, Canada. Pursued a degree in Software Engineering at the University of Windsor. There, he earned his Honors Bachelor of Mathematics double majoring in Mathematics and Computer Science. He expects to graduate with a Master of Science degree in Mathematics in Summer 2007 and is to start a Ph.D. at the School of Computer Science at the University of Waterloo in Fall 2007.