OUTPUT SENSITIVE ALGORITHMS TO COMPUTE HIGHER-ORDER VORONOI DIAGRAMS IN EUCLIDEAN D-SPACE

by

Damon M. Kaller

B.Sc. University of British Columbia 1986

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
August 1992

APPROVAL

Name:	Damon M. Kaller
Degree:	Master of Science
Title of thesis:	Output Sensitive Algorithms to Compute Higher-Order Voronoi Diagrams in Euclidean *d*-Space

Examining Committee: Dr. Joseph Peters
Associate Professor, Computing Science, S.F.U.
Chair

Dr. Binay Bhattacharya
Associate Professor, Computing Science, S.F.U.
Senior Supervisor

Dr. Pavol Hell
Professor, Computing Science, S.F.U.
Supervisor

Dr. Ramesh Krishnamurti
Assistant Professor, Computing Science, S.F.U.
External Examiner

Date Approved: August 10, 1992

Title of Thesis/Project/Extended Essay

Output Sensitive Algorithms to Compute Higher-Order Voronoi Diagrams

in Euclidean d-Space.

Author: ___

(signature)

Damon Michael Kaller

(name)

August 12, 1992

(date)

Abstract

The order-k Voronoi diagram (denoted V_k^S) of a set S of n points in Euclidean d-space $\Re^d$ is a cell complex which partitions $\Re^d$. Each cell is a convex polytope which is associated with a k-subset $T \subset S$, and corresponds to the region of space for which every element of T is at least as close as any element of $S - T$. We present algorithms which compute V_k^S in a non-incremental manner: that is, V_{k-1}^S is not needed as a preliminary step in the computation of V_k^S.

The first algorithm enumerates all v vertices of V_k^S for a nondegenerate point set—along with the information on which polytopes each vertex lies. From this, the entire facial graph of the diagram may be derived. The approach is to move from vertex to vertex along edges, until all of the vertices have been visited. The algorithm has running time $\theta(d^2n + d^3 \log n)$ per vertex.

The second algorithm enumerates all polytopes along with their facets, and does not require that the input point set be nondegenerate. This is motivated by the problem of reference set thinning in pattern recognition. It can be shown that only the facet information of the order-k Voronoi diagram of the reference set is necessary for thinning under the k-nearest neighbor decision rule. An order-k Voronoi polytope may be expressed as the intersection of $k(n-k)$ constraints—the nonredundant ones determine the facets of the polytope. A two stage approach is used in the second algorithm to find all of the nonredundant constraints. In stage 1, a subset of "relevant" points of S is found: each such point lies on some hypersphere which separates T from $S - T$. This spherical separability problem in $\Re^d$ is equivalent to a linear separability problem in $\Re^{d+1}$, and also equivalent to an extreme point problem in $\Re^{d+1}$. In stage 2, the constraints generated by the relevant points are tested for nonredundancy. This, too, is equivalent to an extreme point problem. Linear programming techniques are used to solve the extreme point problems. The running time of the algorithm can be bounded by $O(3^{d^2}n + dk \log n)$ per facet.

The high dimension-dependent constant in the latter algorithm makes it unappealing from a practical point of view. The constant derives from Megiddo's (modified) linear-time linear programming technique. A more practical algorithm is obtained by techniques based on Dantzig's simplex method, which is well-known empirically to run in linear expected time, despite its exponential worst-case performance. This "practical" facet enumeration algorithm has been implemented, and some experimental results are presented.

Acknowledgement

I would like to express my heartfelt gratitude to my research supervisor, Prof. Binay Bhattacharya, for providing the motivation and counsel for this work, and for carefully reading through preliminary versions of the manuscript.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Voronoi diagram of a set of points—or "sites"—in Euclidean d-space partitions the space into disjoint cells. Each site corresponds to a single cell, consisting of that region of space for which the given site is the nearest, under the Euclidean distance metric. The Voronoi diagram has been studied under other distance metrics, and for sites which are geometric primitives other than points (see [Aur90] for an extensive bibliography). This thesis will be concerned only with the Voronoi diagram of point sets, under the Euclidean distance metric.

The Voronoi diagram is mentioned in the mathematical and scientific literature as early as 1840. It has applications in areas as diverse as crystallography, metallurgy, meteorology, biology, astrophysics, computer science and mathematics. As a result of this diversity, the Voronoi diagram has come to be known under different names in different disciplines: It has been called the *Voronoi diagram* or *Dirichlet tesselation*, after the mathematicians George Voronoi and Peter Lejeune-Dirichlet. The term *Wirkungsbereiche*—or, "domains of action"—has been used in crystallography. Metallurgists speak of *Wigner-Seitz zones*, in honor of the two scientists who first used this structure to describe the equilibrium of a molecular system. Geographers have used *Thiessan polygons* to map land surfaces for various applications. *Blum's (medial axis) transform*, used to model biological shapes, can be interpreted as a Voronoi diagram as well. For a more extensive survey and bibliography of such applications, the reader is referred to [Aur90] or [Bha82, chapter 4].

The Voronoi diagram has been generalized to the order-k Voronoi diagram. Given a set S of n sites in d-space, the order-k Voronoi diagram partitions the space into disjoint cells: each cell corresponds to the unique k-subset of S whose elements are the k nearest neighbors

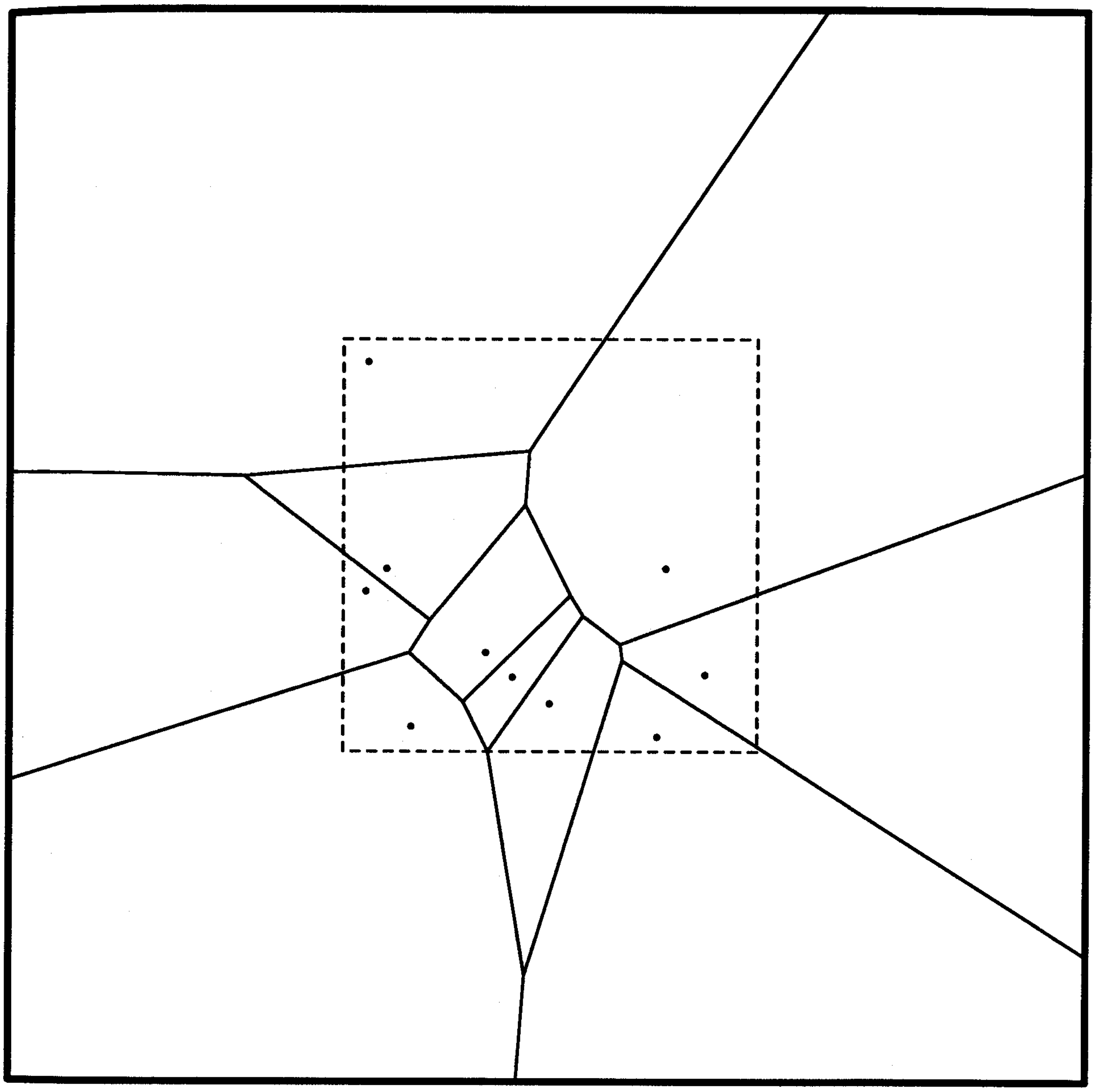

Figure 1.1: The Order-1 Voronoi Diagram of 10 Points in the Plane, Randomly Chosen from the Unit Square

(dashed lines indicate the unit square)

of any point in the cell. In this context the original "Voronoi diagram" is called the order-1 Voronoi diagram. Not all k-subsets of S will, in general, correspond to a nonempty cell in the order-k Voronoi diagram. Each cell is a polytope—that is, a convex region with linear boundaries. As an example, the order-1 Voronoi Diagram of 10 point sites in the plane is given in figure 1.1. Corresponding to each point is a convex polygon, which is the part of the plane which has the corresponding point as its nearest neighbor.

The order-k Voronoi Diagram has applications in pattern recognition [DH73] and density estimation [LD65]. For example, in pattern recognition, we are given a set of n patterns with which to build a classifier; each pattern is associated with a known class. We can measure some d real-valued parameters and associate with each pattern the resulting d-dimensional "feature vector". The set of n feature vectors is called the "reference set". A "test" pattern—with an unknown class—can be classified according to the k-nearest neighbor rule: that is, it is classified according to the dominant class among the k reference patterns whose feature vectors are nearest to it own feature vector. The order-k Voronoi Diagram of the reference set partitions d-space into regions having the same answer to the k-nearest neighbor query. In higher dimensional spaces, it is generally easier to solve the k-nearest neighbor problem directly by computing n distance functions, than by locating a point within the cell complex of the order-k Voronoi Diagram. However, by computing the order-k Voronoi Diagram, it is possible to thin the reference set [Bha82, chapter 7]: that is, to delete some subset of the feature vectors without affecting the k-nearest neighbor decision rule ("exact thinning"), or with only a small percentage of misclassifications ("inexact thinning").

In the following discussion, V_k^S will denote the order-k Voronoi Diagram of a point set $S \subset \Re^d$, where it is understood that $n = |S|$. A polytope (or cell) in V_k^S will be denoted by $V_k^S(T)$, where it is understood that $T \subset S$, and $|T| = k$.

The order-k Voronoi diagram was first introduced into the computer science literature by Shamos and Hoey [SH75]. They conjectured that the number of cells in any planar V_k^S is in $O(k(n-k))$. This was later proven by Lee [Lee82].

Lee [Lee82] presented the first algorithm to construct V_k^S in the plane. The algorithm requires $O(k^2 n \log n)$ time and $O(k^2(n-k))$ space. The approach is to incrementally construct V_i^S for $i = 1, 2 \ldots k$. So, the above complexity bounds hold for constructing all of the first k Voronoi diagrams. To construct V_{i+1}^S from V_i^S, the algorithm "partitions" each polytope $V_i^S(T)$, using the order-1 Voronoi Diagram of $S - T$.

Bhattacharya [Bha83] presented an algorithm to *directly* compute a planar V_k^S in $O(nk(n-$

k)) time and $O(k(n-k))$ space. The algorithm is direct, in the sense that V_i^S for $1 \le i < k$ are not needed as intermediate steps. The approach is to search from vertex to vertex along edges of the polygonal cells. This algorithm is reviewed in more detail in chapter 4.

Chazelle and Edelsbrunner [CE85] presented an algorithm to directly compute a planar V_k^S in $O(n^2 \log n + k(n-k)\log^2 n)$ time and $O(k(n-k))$ space. An alternate version, trading space for speed, requires $O(n^2 + k(n-k)\log^2 n)$ time and $O(n^2)$ space. The approach is to transform $S \subset \Re^2$ into an arrangement of planes in $\Re^3$. The k^{th} "level" of the arrangement can be projected to V_k^S in the original space. Constructing the k^{th} level can be reduced to a point set problem in $\Re^2$: this facilitates the direct construction of the k^{th} level. This approach generalizes to higher dimensions [EOS86] [ES86] (see below), although direct computation of the k^{th} level can no longer be performed efficiently.

Clarkson [Cla87] uses random sampling to compute a planar V_k^S in expected time $O(kn^{1+\varepsilon})$ (for any $\varepsilon > 0$) with a constant that is dependent upon ε. This algorithm, unlike the previous ones, requires that S be nondegenerate: hence any vertex in a planar V_k^S will be the circumcenter of exactly three sites. The algorithm uses divide-and-conquer, and randomly samples subsets of three sites in order to compute the vertices.

In d dimensions, the number of cells in all V_i^S, for $1 \le i \le k$, can be bounded by $O(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor})$ [CS89]. Edelsbrunner, O'Rourke and Seidel [EOS86] [ES86] have shown—by the equivalence of all V_k^S $(1 \le k < n)$ in $\Re^d$, to a particular arrangement of hyperplanes in $\Re^{d+1}$—that the total size of all $n-1$ Voronoi diagrams is $O(n^{d+1})$.

The approach of Edelsbrunner, O'Rourke and Seidel to compute all $n-1$ order-k Voronoi Diagrams is similar to the approach of [CE85] (reviewed above). The point set $S \subset \Re^d$ is transformed to an arrangement of hyperplanes in $\Re^{d+1}$, which encloses some origin. The k^{th} "level" of this arrangement is, roughly-speaking, the star-shaped region separated from the origin by $k-1$ hyperplanes; V_k^S can be constructed by projecting the k^{th} level back down into the original space $\Re^d$. The $(d+1)$-dimensional arrangement of n hyperplanes can be constructed in $O(n^{d+1})$ time [EOS86] by incrementally adding one hyperplane at a time, in any arbitrary order. Hence, all of the order-k Voronoi Diagrams can be computed in $O(n^{d+1})$ time.

Mulmuley [Mul89] modified the approach of Edelsbrunner, O'Rourke and Seidel, by randomizing the order of insertion of the hyperplanes. Additional storage space is required for the "conflict information": that is, the intersection of each not-yet-inserted hyperplane with the current arrangement. The resulting algorithm computes only the first k levels of a

$(d+1)$-dimensional arrangement, with expected running time $O(k^{\lceil\frac{d+1}{2}\rceil}n^{\lfloor\frac{d+1}{2}\rfloor})$.

Mulmuley [Mul90] also presented a deterministic algorithm to compute the first k Voronoi diagrams, in time $O(s\log n + k^d n^2)$, where s is the output size. He claims that this algorithm is output sensitive, under the conjecture that a lower bound on the size of the first k Voronoi diagrams is $\Omega(k^d n)$. The approach of the algorithm is to incrementally compute levels 1 through k of the corresponding arrangement of hyperplanes in $\Re^{d+1}$. This is done through linear programming calls; the $k^d n^2$ term of the complexity bound is derived using Megiddo's [Meg84] linear-time linear programming algorithm—which has $O(2^{2^d}n)$ complexity to solve a linear program with n constraints in $\Re^d$. The algorithm was modified by [Dye86] and [Cla86], resulting in an improved constant of 3^{d^2}. Hence, there is a high dimension-dependent constant hidden in the complexity bound.

Boissonnat, Devillers and Teillaud [BDT90] present a semi-dynamic algorithm to compute the first k Voronoi diagrams. Each site is added incrementally by updating the "k-Delaunay tree" which contains all of the information on the first k Voronoi diagrams. This allows on-line additions to the Voronoi diagrams. Using a randomized analysis, the expected complexity is $O(k^{\lceil\frac{d+1}{2}\rceil+1}n^{\lfloor\frac{d+1}{2}\rfloor})$ time, and $O(k^{\lceil\frac{d+1}{2}\rceil}n^{\lfloor\frac{d+1}{2}\rfloor})$ space.

None of these algorithms compute V_k^S directly in arbitrary dimension for arbitrary k. In this thesis, two algorithms are presented which do so. The first algorithm directly computes all vertices of V_k^S for a nondegenerate point set S, in time $O(d^2n + d^3\log n)$ per vertex, and space $O(d)$ per vertex. Given the output of this algorithm, all of the polytopes on which each vertex lies can be computed in a straight-forward manner. The second algorithm directly computes only the facets of V_k^S. It is expected that the number of facets is considerably smaller than the number of vertices, and there are applications for which only the facet information is needed. For example, the problem of reference set thinning under the k-nearest neighbor decision rule (reviewed earlier) requires only the facet information of the order-k Voronoi Diagram of the reference set.

The facet enumeration algorithm uses a two stage approach to determine all of the facets of each polytope $V_k^S(T)$. Every facet is determined by the perpendicular bisector of some $\mathbf{p} \in T$ and some $\mathbf{q} \in S - T$. The union of all such sets $\{\mathbf{p},\mathbf{q}\}$ which generate some facet of $V_k^S(T)$ constitutes the "relevant" points of S. In "stage 1", we determine the relevant points, by transforming the problem into one of determining the nonredundancies among a system of constraints (by duality, this is equivalent to an extreme point problem). The "stage 2" problem is to find, among the relevant points, those pairs which generate a facet of $V_k^S(T)$:

this, too, is a problem of determining nonredundancy among a system of constraints.

The algorithm has a running time of $O(n + kc + k \log n)$ per facet, where c is no greater than the maximum number of facets of any polytope in V_k^S. It is conjectured that the expected value of c is a constant, in fixed dimension: under this conjecture, the running time would be $O(n + k \log n)$ in fixed dimension. The space requirement is $O(k)$ per facet. The bound on the running time of the facet enumeration algorithm is obtained by using Megiddo's (modified) $O(3^{d^2} n)$ linear programming technique [Meg84] [Dye86] [Cla86]. In d dimensions, the algorithm has running time $O(3^{d^2} n + kd \log n)$ per facet. Hence, the algorithm is not practical.

A practical version of the algorithm has been developed, which uses techniques based on the simplex method [Dan63] of linear programming. Although the complexity of the simplex method cannot be bounded by a linear function of n, extensive empirical experience has demonstrated that its expected running time is linear in n. These techniques allow us to, in essence, solve many linear programs simultaneously. The practical algorithm to enumerate the facets of V_k^S has been implemented. Experimental evidence is presented supporting the claim that the expected running time is $O(n)$ per facet in fixed dimension, for any k.

1.1 Overview of Thesis

Chapter 2 of this thesis reviews the mathematical preliminaries needed in later chapters.

Chapter 3 presents definitions and properties of the order-k Voronoi Diagram (V_k^S). Section 3.1 defines V_k^S in general terms, covering the case of a degenerate point set S. Section 3.2 describes the simplified diagram which results from assuming that S is nondegenerate. Section 3.3 describes how degenerate input can be handled by an algorithm which make the assumption of nondegeneracy. Section 3.4 contains lemmas which describe the properties of the V_k^S of an unrestricted point set S. Section 3.5 contains lemmas which apply only when S is nondegenerate.

Chapter 4 presents the algorithm which directly enumerates all vertices of V_k^S, in time $O(d^2 n + d^3 \log n)$ per vertex. Section 4.1 reviews the algorithm of [Bha83], for the direct computation of V_k^S in a two-dimensional space. Section 4.2 generalizes the algorithm to d-dimensional spaces. The generalized algorithm, however, requires that the input set be nondegenerate.

Chapter 5 describes the two-stage algorithm for directly computing all facets of each polytope of V_k^S. Linear programming is reviewed in section 5.1: A Voronoi polytope is equivalent to the feasible region of a linear programming problem, and the facets are determined by the nonredundant constraints defining the feasible region. Section 5.2 reviews the standard simplex method of linear programming, and section 5.3 reviews the revised simplex method. Techniques for the determination of nonredundant constraints in a linear programming problem are reviewed in section 5.4. In addition, an algorithm—which uses Megiddo's [Meg84] linear programming technique—is presented to determine the f nonredundant constraints among a total of m constraints, in output-sensitive $O(fm)$ time. Section 5.5 mentions an approach for the determination of nonredundancy which was investigated, but was not fruitful. A new interpretation of a simplex pivot is presented in section 5.6, and in section 5.7 an algorithm is presented for determining nonredundancy, using this pivoting strategy. In section 5.8, it is shown how to transform the stage 1 problem into a problem of determining nonredundancy. A practical version of the facet enumeration algorithm is presented in section 5.9: this makes use of the pivoting algorithm of section 5.7. In section 5.10, the algorithm is analyzed to give time complexity of $O(n + k \log n)$ per facet, using the output-sensitive method of section 5.4 to determine nonredundancy.

Chapter 6 presents computational results, obtained from implementing the practical version of the facet enumeration algorithm. Section 6.2 presents results on the number of regions and number of facets for V_k^S of randomly generated point sets; in addition, the running time of the implementation is analyzed. Section 6.3 presents results on the efficiency of the pivoting algorithm (of section 5.7) in solving a single extreme point problem—which is equivalent, by duality, to the problem of determining nonredundancy.

Chapter 2

Mathematical Preliminaries

This chapter reviews the notation, terminology and mathematical preliminaries used in this thesis. It is assumed that the reader is familiar with the elementary concepts of linear algebra and of affine geometry. A familiarity with such fundamental topological properties as open and closed sets is also assumed. For a more detailed introduction, the reader is referred to any relevant introductory textbook.

2.1 Notation

Throughout this thesis, the following notational conventions are adopted:

- Integers are denoted by lower-case English letters $a, b, \ldots, z$.
- Real numbers are denoted by lower-case Greek letters: $\alpha, \beta, \ldots, \omega$; with the exception of the (real) coordinates of points $\mathbf{x} \in \Re^d$ (see below).
- Points in Euclidean d-space ($\Re^d$) are denoted by the boldface lower-case English letters $\mathbf{a}$, $\mathbf{b}$, $\ldots$, $\mathbf{z}$. The coordinates are denoted by subscripting (between 1 and d) the corresponding non-boldfaced letter. That is, it shall be understood that $\mathbf{x}$ denotes the point $(x_1, x_2, \ldots, x_d)$.
- Sets (understood to be sets of points in $\Re^d$, unless otherwise indicated) are denoted by capital letters: $A, B, \ldots, Z$.
- Matrices are denoted by capital letters: $A, B, \ldots, Z$.

Given two points $\mathbf{p}, \mathbf{q} \in \Re^d$ and scalar $\varrho > 0$, the following notation is used:

- $\mathbf{p} \cdot \mathbf{q} = \sum_{i=1}^{d}(p_i \times q_i)$ denotes the dot product of $\mathbf{p}$ and $\mathbf{q}$.
- $\|\mathbf{p}\|_2 = \sqrt{\sum_{i=1}^{d} p_i^2}$ denotes the Euclidean length (or 2-norm) of the vector $\mathbf{p}$.
- $d(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_2$ denotes the Euclidean distance between $\mathbf{p}$ and $\mathbf{q}$.
- $B(\mathbf{p}, \mathbf{q}) = \{\mathbf{x} \in \Re^d | d(\mathbf{x}, \mathbf{p}) = d(\mathbf{x}, \mathbf{q})\}$ denotes the perpendicular bisector of $\mathbf{p}$ and $\mathbf{q}$.
- $H(\mathbf{p}, \mathbf{q})$ denotes the open halfspace containing $\mathbf{p}$ and bounded by $B(\mathbf{p}, \mathbf{q})$.
- $\overline{H(\mathbf{p}, \mathbf{q})}$ denotes the closed halfspace containing $\mathbf{p}$ and bounded by $B(\mathbf{p}, \mathbf{q})$.
- $C(\mathbf{c}, \varrho) = \{\mathbf{x} \in \Re^d | d(\mathbf{c}, \mathbf{x}) = \varrho\}$ denotes the hypersphere centered at $\mathbf{c}$ with radius ϱ.

For a point set S:

- $CH(S)$ denotes the convex hull of S.
- $aff(S)$ denotes the affine hull of S.
- $V_k^S(T)$ denotes the Voronoi polytope corresponding to $T \subset S$, $|T| = k$, in the order-k Voronoi Diagram of S (see definition 3.1 on page 19).
- V_k^S denotes the order-k Voronoi Diagram of S (see definition 3.3 on page 20).

For a closed compact point set S:

- $int(S)$ denotes the interior of S: *i.e.* the maximal open point set contained in S.
- $bd(S)$ denotes the boundary of S: $bd(S) = S - int(S)$.

2.2 k-Flats

A point $\mathbf{p} \in \Re^d$ is said to be an *affine combination* of the point set $S = \{\mathbf{p}^i \in \Re^d\}_{i=1}^m$ if there exist real constants $\alpha_1, \alpha_2, \ldots, \alpha_m$ such that:

$$\mathbf{p} = \sum_{i=1}^{m} \alpha_i \mathbf{p}^i \qquad \text{and} \qquad \sum_{i=1}^{m} \alpha_i = 1$$

Furthermore, $S \subset \Re^d$ is said to be *affinely independent* whenever no $\mathbf{p} \in S$ is an affine combination of the remaining points, $S - \{\mathbf{p}\}$.

A k-flat $F \subset \Re^d$ (for $0 \leq k < d$) is the set of all affine combinations of some $k+1$ affinely independent points, $B = \{\mathbf{p^i} = (p_1^i, p_2^i, \ldots, p_d^i)\}_{i=1}^{k+1}$. These points (or any other set of $k+1$ affinely independent points in F) constitute a *basis* for F. Hence, any point $\mathbf{f} \in F$ may be expressed as follows:

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_d \end{bmatrix} = \begin{bmatrix} p_1^1 & p_1^2 & \cdots & p_1^{k+1} \\ p_2^1 & p_2^2 & \cdots & p_2^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ p_d^1 & p_d^2 & \cdots & p_d^{k+1} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_k \\ 1 - \alpha_1 - \alpha_2 - \ldots - \alpha_k \end{bmatrix}$$

Whenever $k < d$, this is an under-determined system and we can express $d-k$ coordinates of $\mathbf{f}$ in terms of the other k coordinates:

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{d-k} \end{bmatrix} = A \begin{bmatrix} f_{d-k+1} \\ f_{d-k+2} \\ \vdots \\ f_d \end{bmatrix} + \mathbf{b} \tag{2.1}$$

for an appropriate $(d-k) \times k$ matrix A, and $(d-k)$-dimensional vector $\mathbf{b}$. The terms *line* and *hyperplane* will be used interchangeably with 1-flat and $(d-1)$-flat, respectively.

By substituting $k = 1$ in equation (2.1), we can obtain the following equation of a line:

$$\mathbf{x} = \mathbf{p} + \tau \mathbf{v} \qquad \forall \tau \in \Re \tag{2.2}$$

for appropriate d-dimensional vectors $\mathbf{p}$, $\mathbf{v}$.

Similarly, the equation of a hyperplane can be found by making the substitution $k = d-1$:

$$\mathbf{a} \cdot \mathbf{x} = \beta \tag{2.3}$$

for an appropriate d-dimensional vector $\mathbf{a}$, and scalar β. Note that the perpendicular bisector $B(\mathbf{p}, \mathbf{q})$ of two distinct points $\mathbf{p}$ and $\mathbf{q}$ in $\Re^d$ is a hyperplane. $B(\mathbf{p}, \mathbf{q})$ contains all $\mathbf{x}$ satisfying:

$$\begin{array}{rrcl} & \sum_{i=1}^d (x_i - p_i)^2 & = & \sum_{i=1}^d (x_i - q_i)^2 \\ i.e. & \sum_{i=1}^d 2x_i(q_i - p_i) & = & \sum_{i=1}^d (q_i^2 - p_i^2) \\ i.e. & \sum_{i=1}^d 2x_i(q_i - p_i) & = & \sum_{i=1}^d (q_i - p_i)(q_i + p_i) \\ i.e. & (\mathbf{q} - \mathbf{p}) \cdot \mathbf{x} & = & \frac{(\mathbf{q}-\mathbf{p})\cdot(\mathbf{q}+\mathbf{p})}{2} \end{array}$$

A hyperplane $H = \{\mathbf{x} | \mathbf{a} \cdot \mathbf{x} = \beta\}$ determines two closed halfspaces which intersect in H:

- $\{\mathbf{x}|\mathbf{a}\cdot\mathbf{x} \leq \beta\}$
- $\{\mathbf{x}|\mathbf{a}\cdot\mathbf{x} \geq \beta\}$

The problem of *linear separability* of two sets $S_1, S_2 \subset \Re^d$ is to determine the existence of a hyperplane for which the points of S_1 lie in one of its closed halfspaces, and the points of S_2 lie in the other. This problem can be solved by linear programming (see section 5.1).

The *affine hull* of a point set $S \subseteq \Re^d$ (denoted $aff(S)$) is the set of all affine combinations of S. Hence $aff(S)$ is a k-flat for k the size of any maximal subset of affinely independent points of S.

In developing the properties of Voronoi diagrams, it will be useful to have the following lemma:

Lemma 2.1 *In $\Re^d$, the intersection of a k-flat, F, with a hyperplane is either:*

- *empty, or*
- *equal to F, or*
- *a $(k-1)$-flat*

Proof. omitted.

2.3 Hyperspheres

A hypersphere $C(\mathbf{c}, \varrho)$ in $\Re^d$ is the set of points which lie at a fixed distance (the *radius*) ϱ from a specified point $\mathbf{c}$, called the *center*.

Any point $\mathbf{x} \in C = C(\mathbf{c}, \varrho)$ will be said to be on the *surface* of C; any point $\mathbf{p}$ for which $d(\mathbf{p}, \mathbf{c}) < \varrho$ will be said to be in the *interior* of C; any point $\mathbf{p}$ for which $d(\mathbf{p}, \mathbf{c}) > \varrho$ will be said to be in the *exterior* of C. Furthermore, a hypersphere will be said to *contain* those points which lie on its surface, to *enclose* those points which lie in its interior, and to *exclude* those points which lie in its exterior.

In developing the properties of Voronoi diagrams, it will be useful to have the following operations:

Proposition 2.1 [To contract a hypersphere]
Given a hypersphere $C = C(\mathbf{c}, \varrho) \subset \Re^d$ containing some surface point $\mathbf{p}$ and enclosing a finite point set I: then for any $C_\varepsilon = C(\mathbf{c}', \varrho - \varepsilon)$ such that $d(\mathbf{c}, \mathbf{c}') = \varepsilon$ and $\mathbf{c}'$ lies on the open line segment between $\mathbf{c}$ and $\mathbf{p}$:

1. *C_ε contains $\mathbf{p}$*
2. *any point exterior to C is also exterior to C_ε*
3. *any surface point of C, other than $\mathbf{p}$, is exterior to C_ε*

Furthermore, ε may be chosen in such a way to ensure that all points of I are interior to C_ε. Alternatively, ε may be chosen such that some $\mathbf{e} \in I$ is on the surface of C_ε, and no point of I is in the exterior of C_ε.

Proof. Let $\mathbf{q}$ be an exterior or surface point of C. Then $d(\mathbf{q}, \mathbf{c}') = \varrho - \varepsilon$, only if $\mathbf{q} = \mathbf{p}$. Otherwise, $d(\mathbf{q}, \mathbf{c}') > \varrho - \varepsilon$. This establishes items (1) through (3).

For any $\mathbf{e} \in I$: $d(\mathbf{c}, \mathbf{e}) < d(\mathbf{c}, \mathbf{p})$. Consider the point $\mathbf{c}'$ as it moves along the open line segment from $\mathbf{c}$ to $\mathbf{p}$ (*i.e.* as ε increases). Since $d(\mathbf{c}', \mathbf{p})$ decreases towards 0, there must be a point at which $d(\mathbf{c}', \mathbf{e}) = d(\mathbf{c}', \mathbf{p})$. Let $\mathbf{c}'$ be fixed at the point where the first $\mathbf{e} \in I$ is equidistant to $\mathbf{p}$. So, for $\varepsilon = d(\mathbf{c}, \mathbf{c}')$, $\mathbf{e}$ will be a surface point of C_ε, and all other points of I are either interior or surface points of C_ε. If we wish, alternatively, to have all I interior points, we need merely to fix $\mathbf{c}'$ earlier than the point at which the first $\mathbf{e}$ is equidistant; that is, consider any hypersphere $C_{\varepsilon'}$, for $0 < \varepsilon' < \varepsilon$.

Q.E.D.

When C_ε is constructed from the hypersphere C of proposition 2.1, it will be said that C is "contracted towards $\mathbf{p}$". Similarly, C may be "expanded away from $\mathbf{p}$", according to the following:

Proposition 2.2 [To expand a hypersphere]
Given a hypersphere $C = C(\mathbf{c}, \varrho) \subset \Re^d$ containing some surface point $\mathbf{p}$ and excluding a finite point set E: then for any $C_\varepsilon = C(\mathbf{c}', \varrho + \varepsilon)$ such that $d(\mathbf{c}, \mathbf{c}') = \varepsilon$ and $\mathbf{c}$ lies on the open line segment between $\mathbf{c}'$ and $\mathbf{p}$:

1. C_ε contains $\mathbf{p}$

2. any point interior to C is also exterior to C_ε

3. any surface point of C, other than $\mathbf{p}$, *is interior to C_ε*

Furthermore, ε may be chosen in such a way to ensure that all points of I are exterior to C_ε. Alternatively, ε may be chosen such that some $\mathbf{e} \in I$ *is on the surface of C_ε, and no point of I is in the interior of C_ε.*

Proof. similar to the proof of proposition 2.1.

Proposition 2.3 *Given a hypersphere $C \subset \Re^d$ containing an affinely independent set G, enclosing a finite point set I, and excluding a finite point set E: for any partition of G into $G_S \cup G_I \cup G_E$, there exists a hypersphere C' such that:*

- *C' contains G_S on its surface.*
- *C' includes $I \cup G_I$.*
- *C' excludes $E \cup G_E$.*

Proof. Note that $|G| \leq d+1$ (since G is affinely independent). Assume that $G_S \neq G$ (if this were the case then $C = C'$). Let H be a hyperplane which contains G_S, such that G_I lies in one of its open halfspaces and G_E lies in the other. It is obvious that such a hyperplane exists when $|G_S| = d$. This hyperplane may be perturbed so that any $\mathbf{g} \in G_S$ lies in either of the open halfspaces and the other points, $G - \{\mathbf{g}\}$, are still contained by it. It follows inductively that H exists for any partition of G.

Let $\mathbf{c}$ be the center of the hypersphere C. Let $\mathbf{h}$ be the surface normal of H which is directed towards the halfspace containing G_I. Now, C' may be constructed with center $\mathbf{c}' = \mathbf{c} + \varepsilon\mathbf{h}$ and with G_S on its surface, for $\varepsilon > 0$ which is small enough to ensure that all points of I remain interior, and all points of E remain exterior.

Q.E.D.

2.4 Spherical Separability

The problem of spherical separability of $S_1 \subset \Re^d$ from $S_2 \subset \Re^d$ is to determine the existence of a hypersphere which encloses (or contains) each point of S_1, and excludes (or contains) each point of S_2. Note that spherical separability of S_1 from S_2 is not the same as spherical separability of S_2 from S_1.

The spherical separability problem in $\Re^d$ may be transformed into a problem of linear separability in $\Re^{d+1}$ by mapping each point $\mathbf{p} \in S_1 \cup S_2$ onto the paraboloid in $\Re^{d+1}$ which is the d-dimensional surface defined by: $p_{d+1} = \sum_{i=1}^{d} p_i^2$. Let us denote by $\mathbf{p}^* \in \Re^{d+1}$ the vertical projection of $\mathbf{p} \in \Re^d$ onto this paraboloid.

Definition 2.1 *The paraboloid transformation of a point set $S \subseteq \Re^d$ is the set:*

$$S^* = \{(p_1, p_2, \ldots, p_d, \sum_{i=1}^{d} p_i^2) | (p_1, p_2, \ldots, p_d) \in S\}$$

The distance $d(\mathbf{p}, \mathbf{c})$ of any $\mathbf{p} \in \Re^d$ from a fixed point $\mathbf{c}$ may be rewritten as:

$$\begin{aligned} d^2(\mathbf{p}, \mathbf{c}) &= \textstyle\sum_{i=1}^{d}(p_i - c_i)^2 \\ &= \textstyle\sum_{i=1}^{d} p_i^2 - 2\sum_{i=1}^{d} p_i c_i + \sum_{i=1}^{d} c_i^2 \\ &= (-2c_1, -2c_2, \ldots, -2c_d, 1) \cdot \mathbf{p}^* + \textstyle\sum_{i=1}^{d} c_i^2 \end{aligned}$$

A point $\mathbf{p}$ lies in the interior (respectively surface, exterior) of the hypersphere $C = C(\mathbf{c}, \varrho)$ whenever $d(\mathbf{c}, \mathbf{p})$ is less than (respectively equal to, greater than) ϱ.

$$\begin{array}{lclcll} d(\mathbf{p}, \mathbf{c}) < \varrho & \Longleftrightarrow & (2c_1, 2c_2, \ldots, 2c_d, -1) & \cdot & \mathbf{p}^* & > \sum_{i=1}^{d} c_i^2 - \varrho^2 \\ d(\mathbf{p}, \mathbf{c}) = \varrho & \Longleftrightarrow & (2c_1, 2c_2, \ldots, 2c_d, -1) & \cdot & \mathbf{p}^* & = \sum_{i=1}^{d} c_i^2 - \varrho^2 \\ d(\mathbf{p}, \mathbf{c}) > \varrho & \Longleftrightarrow & \underbrace{(2c_1, 2c_2, \ldots, 2c_d, -1)}_{\mathbf{c}'} & \cdot & \mathbf{p}^* & < \underbrace{\sum_{i=1}^{d} c_i^2 - \varrho^2}_{\beta} \end{array}$$

Hence, the points of S_1 lie in one of the closed halfspaces determined by the hyperplane $\{\mathbf{x} \in \Re^{d+1} | \mathbf{c}' \cdot \mathbf{x} = \beta\}$, and the points of S_2 lie in the other:

$$\begin{array}{lcl} \mathbf{p} \text{ is an interior point of } C(\mathbf{c}, \varrho) & \Longleftrightarrow & \mathbf{c}' \cdot \mathbf{p}^* > \beta \\ \mathbf{p} \text{ is a surface point of } C(\mathbf{c}, \varrho) & \Longleftrightarrow & \mathbf{c}' \cdot \mathbf{p}^* = \beta \\ \mathbf{p} \text{ is an exterior point of } C(\mathbf{c}, \varrho) & \Longleftrightarrow & \mathbf{c}' \cdot \mathbf{p}^* < \beta \end{array}$$

Note that $c'_{d+1} < 0$: this forces the hypersphere corresponding to H to enclose S_1, excluding S_2, and not the other way around.

Lemma 2.2 *There exists a solution* $\mathbf{a} \in \Re^{d+1}$ *to the following system:*

$$\begin{array}{rcl} \mathbf{a}\cdot\mathbf{p}^* & = & \beta;\ \forall \mathbf{p}^* \in S_0^* \\ \mathbf{a}\cdot\mathbf{p}^* & > & \beta;\ \forall \mathbf{p}^* \in S_1^* \\ \mathbf{a}\cdot\mathbf{p}^* & < & \beta;\ \forall \mathbf{p}^* \in S_2^* \\ a_{d+1} & \leq & 0 \end{array}$$

if and only if there exists a hypersphere enclosing S_1, *excluding* S_2, *and containing* S_0.

Proof. Assume there is a feasible solution $\mathbf{a}$. Since $p_{d+1} = \sum_{i=1}^d p_i^2$, and assuming that $a_{d+1} \neq 0$, this becomes:

$$\begin{array}{lrcl} (1) & \sum_{i=1}^d \frac{a_i p_i}{a_{d+1}} + \sum_{i=1}^d p_i^2 & = & \frac{\beta}{a_{d+1}};\ \forall \mathbf{p} \in S_0 \\ (2) & \sum_{i=1}^d \frac{a_i p_i}{a_{d+1}} + \sum_{i=1}^d p_i^2 & < & \frac{\beta}{a_{d+1}};\ \forall \mathbf{p} \in S_1 \\ (3) & \sum_{i=1}^d \frac{a_i p_i}{a_{d+1}} + \sum_{i=1}^d p_i^2 & > & \frac{\beta}{a_{d+1}};\ \forall \mathbf{p} \in S_2 \end{array}$$

Let $c_i = -\frac{a_i}{2a_{d+1}}$ for $1 \leq i \leq d$. Then:

$$\begin{array}{lrcl} (1) & \sum_{i=1}^d (p_i - c_i)^2 & = & \frac{\beta}{a_{d+1}} + \sum_{i=1}^d c_i^2;\ \forall \mathbf{p} \in S_0 \\ (2) & \sum_{i=1}^d (p_i - c_i)^2 & < & \frac{\beta}{a_{d+1}} + \sum_{i=1}^d c_i^2;\ \forall \mathbf{p} \in S_1 \\ (3) & \sum_{i=1}^d (p_i - c_i)^2 & > & \frac{\beta}{a_{d+1}} + \sum_{i=1}^d c_i^2;\ \forall \mathbf{p} \in S_2 \end{array}$$

Now, the equation (1) in the above system implies that $\sum_{i=1}^d c_i^2 + \frac{\beta}{a_{d+1}} \geq 0$. So the hyperplane $\mathbf{a}\cdot\mathbf{x} = \beta$ in $\Re^{d+1}$ corresponds to a hypersphere in $\Re^d$ with center $\mathbf{c} = (c_1, c_2, \ldots, c_d)$, with radius $\sqrt{\sum_{i=1}^d c_i^2 + \frac{\beta}{a_{d+1}}}$, which encloses S_1, excludes S_2 and contains S_0.

If $a_{d+1} = 0$, then the separating hypersphere degenerates into a separating hyperplane—which may be interpreted as a hypersphere centered at infinity—and the conclusion still holds.

The converse has been proven in the above derivation.

Q.E.D.

Hence, spherical separability in $\Re^d$ may be solved by linear programming in $\Re^{d+1}$; this is the approach taken in [OKM86], and will be used in chapter 5 of this thesis.

2.5 Polytopes

The most prevalent work in the field of polytopes is [Grü67]. This section mentions some of the definitions and results of this work.

Definition 2.2 *A polytope in $\Re^d$ is the intersection of a finite number of closed halfspaces.*

Any intersection of halfspaces in $\Re^d$—whether bounded or unbounded—is admitted as a polytope. In order for a polytope to be bounded, it must be the intersection of at least $d+1$ nonredundant halfspaces. A *simplex* is the name given to any bounded polytope which is the intersection of exactly $d+1$ nonredundant halfspaces. Note, also, that lower dimensional point sets (as well as the empty set) are admissible as polytopes.

A *supporting hyperplane* of a polytope P is a hyperplane H which intersects P, such that P lies completely within one of the closed halfspaces determined by H. The intersection $P \cap H$ is a *face* of P.

Definition 2.3 *For a polytope $P \subset \Re^d$: $F \subset P$ is called a face of P if one of the following holds:*

- $F = P$
- $F = \emptyset$
- $F = P \cap H$*, where H is a supporting hyperplane of P.*

Furthermore, a face is called a k-face whenever its affine hull is a k-flat; by convention, the empty set is called a (-1)-face.

The (-1)-face and the k-face of a k-dimensional polytope $P \subset \Re^d$ (where $k \leq d$) are called the *improper* faces of P. The remaining faces are the *proper* faces.

Definition 2.4 *A facet of a k-dimensional polytope $P \subset \Re^d$ is any maximal proper face of P.*

For an k-dimensional polytope in $\Re^d$ $(1 \leq k \leq d)$ the terms *vertex*, *edge* and *facet* are used interchangeably with 0-face, 1-face and $(k-1)$-face, respectively.

A halfspace is termed *strongly nonredundant* in a polytope $P \subset \Re^d$ whenever its bounding hyperplane H intersects P in a $(d-1)$-face. It is called *weakly redundant* (or *weakly nonredundant*) if $H \cap P$ is an m-face of P, for $1 \leq m < d-1$. Otherwise, $H \cap P = \emptyset$, and the halfspace is *strongly redundant.*

If a polytope P is a d-dimensional subset of $\Re^d$, then P may be expressed as the intersection of the unique set of strongly nonredundant halfspaces whose bounding hyperplanes are the affine hulls of the $(d-1)$-faces of P. If P is an k-dimensional subset of $\Re^d$, for $k < d$, then any intersecting halfspace is weakly nonredundant, so there is not a unique minimal representation of P as an intersection of halfspaces.

An m-face F $(-1 \leq m < k-1)$ of a k-dimensional polytope may be expressed as the intersection:

$$F = \bigcap_{i=1}^{r} F_i$$

where each F_i $(1 \leq i \leq r)$ is an m'-face, $m < m' \leq d-1$. Whenever such a relationship holds, F is said to be a *subface* of F_i, and F_i is said to be a *superface* of F. These relationships are equivalent to the elementary set relations: $F \subset F_i$; $F_i \supset F$.

Any m-face of a k dimensional polytope $(-1 \leq m \leq k)$ is an m-dimensional polytope.

2.6 Polar Transformation

The polar transformation of points to halfspaces, and vice-versa, is a trivial one. There is no computation involved in the transformation—it is merely a question of how we interpret the same d-vector.

Definition 2.5 *For a point* $\mathbf{p} \in \Re^d$, *the halfspace* $H = \{\mathbf{x} \in \Re^d | \mathbf{p} \cdot \mathbf{x} \leq 1\}$ *is called the polar dual of* $\mathbf{p}$. *Conversely,* $\mathbf{p}$ *is called the polar dual of* H.

We will refer to a set of n points and the corresponding set of n halfspaces as duals of one another. Thus, a polytope—the intersection of halfspaces—has a dual set of points. The following lemma provides an useful relationship between these sets.

Lemma 2.3 *Let S be a finite point set in $\Re^d$ with $\mathbf{0}$ in the interior of $CH(S)$, and let $D(\mathbf{p})$ be the polar dual of $\mathbf{p} \in S$. The intersection*

$$P = \bigcap_{\mathbf{p} \in S} D(\mathbf{p})$$

is a non-empty polytope such that:

- $\mathbf{0} \in P$
- $\mathbf{p} \in S$ *is an extreme point of S if and only if $D(\mathbf{p})$ is a nonredundant constraint of P*
- *for $T \subset S$: the elements of T lie on a common m-face of $CH(S)$ if and only if the bounding hyperplanes of $D(T)$ intersect in a $(d-m-1)$-face of P.*

Proof. [Grü67, section 3.4]

To find the polar dual of a set of points is always trivial. To find the dual of a set of hyperplanes may be tricky, since we must first express them in the form $\mathbf{p} \cdot \mathbf{x} \leq 1$. This implies that we must have an feasible point of P, the intersection of the halfspaces. It also implies that if P is empty then the set of hyperplanes does not have a polar dual. The determination of a feasible point of P is equivalent to solving a linear programming problem.

Chapter 3

Properties of Order-k Voronoi Diagrams

3.1 Definition of the Order-k Voronoi Diagram

Definition 3.1 *For a finite point set $S \subset \Re^d$, and $T \subset S$, $|T| = k$: the order-k Voronoi polytope corresponding to T (denoted $V_k^S(T)$) is the region of $\Re^d$ for which every element of T is at least as close as any element of $S - T$.*

The region of space for which $\mathbf{p}$ is at least as close as $\mathbf{q}$ is the closed halfspace, $\overline{H(\mathbf{p}, \mathbf{q})}$. So, the order-$k$ Voronoi polytope may be equivalently defined as the intersection of closed halfspaces.

$$V_k^S(T) = \bigcap_{\substack{\mathbf{p} \in T \\ \mathbf{q} \in S-T}} \overline{H(\mathbf{p}, \mathbf{q})} \tag{3.1}$$

Since S is a finite set, an order-k Voronoi polytope is the intersection of a finite number of closed halfspaces: $V_k^S(T) = H_1 \cap H_2 \cap \ldots \cap H_{k(n-k)}$, where $|S| = n$. Therefore, $V_k^S(T)$ is a polytope, by definition 2.2. We will refer to the m-faces (definition 2.3) of a Voronoi polytope, for $-1 \leq m \leq d-1$, as "Voronoi m-faces". The 0-faces and 1-faces will also be called "Voronoi vertices" and "Voronoi edges" (or simply "V-vertices" and "V-edges") respectively.

Definition 3.2 *$V_k^S(T)$ for $T \subset S \subset \Re^d$ is called:*

- *empty, whenever $V_k^S(T) = \emptyset$*
- *improper, whenever it is a k dimensional region for $0 \leq k < d$*
- *proper, whenever it is a d-dimensional region*

Definition 3.3 *The order-k Voronoi Diagram of S (denoted V_k^S) is the set of all nonempty order-k Voronoi polytopes.*

Henceforth, $V_k^S(T)$ will denote the order-k Voronoi polytope corresponding to T and it shall be understood that $T \subset S$, $|T| = k$ and that $V_k^S(T) \in V_k^S$. Furthermore a "face" of V_k^S will refer to a face of any $V_k^S(T) \in V_k^S$.

Given a set $S \subset \Re^d$: $T \subset S$ (with $|T| = k$) is called a "k-set" of S whenever T is linearly separable from $S - T$. An upper bound on the number of k-sets of S (for $|S| = n$) is $O(k^{\lceil \frac{d}{2} \rceil} n^{\lfloor \frac{d}{2} \rfloor})$ [CS89]. By the equivalence of linear separability in $\Re^d$ and spherical separability in $\Re^{d+1}$, it follows that the maximum number of Voronoi polytopes in the order-k Voronoi Diagram of S can be bounded by $O(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor})$.

For the purposes of this thesis, we will define a "Voronoi facet" (or "V-facet") slightly differently than a polytope facet (definition 2.4). This is to ensure that whenever $B(\mathbf{p}, \mathbf{q})$ determines a facet of $V_k^S(T)$, then $B(\mathbf{q}, \mathbf{p})$ will determine a facet of $V_k^S(T - \{\mathbf{p}\} \cup \{\mathbf{q}\})$. It turns out that these two definitions are equivalent whenever S is in general position (see section 3.2). Otherwise, a Voronoi polytope may have V-facets of differing dimension.

Definition 3.4 *A Voronoi facet of $V_k^S(T)$ is any nonempty intersection:*

$$V_k^S(T) \cap B(\mathbf{p}, \mathbf{q}); \qquad \mathbf{p} \in T;\ \mathbf{q} \in S - T$$

Using the terminology introduced in section 2.5, any defining halfspace H_i—among the $k(n-k)$ halfspaces in equation (3.1)—for which:

$$\bigcap_{\substack{j=1 \\ j \neq i}}^{k(n-k)} H_j = \bigcap_{j=1}^{k(n-k)} H_j$$

is called *redundant*; a redundant halfspace is called *weakly redundant* when it intersects $bd(V_k^S(T))$ in an m-face for $0 \leq m \leq d-2$ and *strongly redundant* otherwise.

Every strongly nonredundant halfspace $\overline{H(\mathbf{p},\mathbf{q})}$ intersects $bd(V_k^S(T))$ in a $(d-1)$-dimensional facet F. $B(\mathbf{p},\mathbf{q})$ will be said to "determine" F, and $\{\mathbf{p},\mathbf{q}\}$ will be called the *generating set* of F. If there are no weakly redundant halfspaces in equation (3.1), then all facets of $V_k^S(T)$ are $(d-1)$-dimensional and determined by strongly nonredundant halfspaces. It turns out that there are no weakly redundant halfspaces whenever S is in general position (see section 3.2).

All of the facets of a given Voronoi polytope $V_k^S(T)$ will be generated by a subset of the points of S: these are the *relevant* points of S.

Definition 3.5 $\mathbf{p} \in S$ *is called relevant with respect to* $V_k^S(T)$ *whenever either:*

- $\mathbf{p} \in T$, *and* $\exists \mathbf{q} \in (S-T)$ *such that* $B(\mathbf{p},\mathbf{q}) \cap V_k^S(T)$ *is nonempty, or*
- $\mathbf{p} \in (S-T)$, *and* $\exists \mathbf{q} \in T$ *such that* $B(\mathbf{q},\mathbf{p}) \cap V_k^S(T)$ *is nonempty.*

So, a point $\mathbf{p}$ is relevant whenever it belongs to the generating set of some constraint which is (strongly or weakly) nonredundant.

For example, figure 3.1 shows an order-k Voronoi polytope $V_3^S(T)$ in $\Re^2$; S is the set for which V_1^S is shown in figure 1.1. The relevant points with respect to $V_3^S(T)$ are R:

$$\begin{aligned} R &= \{\mathbf{q},\mathbf{t},\mathbf{s},\mathbf{v},\mathbf{w},\mathbf{y}\} \\ S &= \{\mathbf{p},\mathbf{q}, \ \dots \ ,\mathbf{y}\} \\ T &= \{\mathbf{t},\mathbf{u},\mathbf{v}\} \end{aligned}$$

Since $|S| = 10$ and $k = 3$, there are $3 \cdot (10-3) = 21$ constraints defining $V_3^S(T)$. Of these, 15 are (strongly) redundant; the 6 nonredundant constraints are shown in figure 3.1, and their bounding lines are labeled. The union of the generating sets of these 6 constraints constitutes the relevant points R.

The interior of an order-k Voronoi polytope ($int\ V_k^S(T)$) is the polytope without its bounding hyperplanes. It is the region of $\Re^d$ for which every element of T is strictly closer than any element of $S-T$. Thus, the interior can be defined as the intersection of open halfspaces:

$$int\ V_k^S(T) = \bigcap_{\substack{\mathbf{p}\in T \\ \mathbf{q}\in S-T}} H(\mathbf{p},\mathbf{q})$$

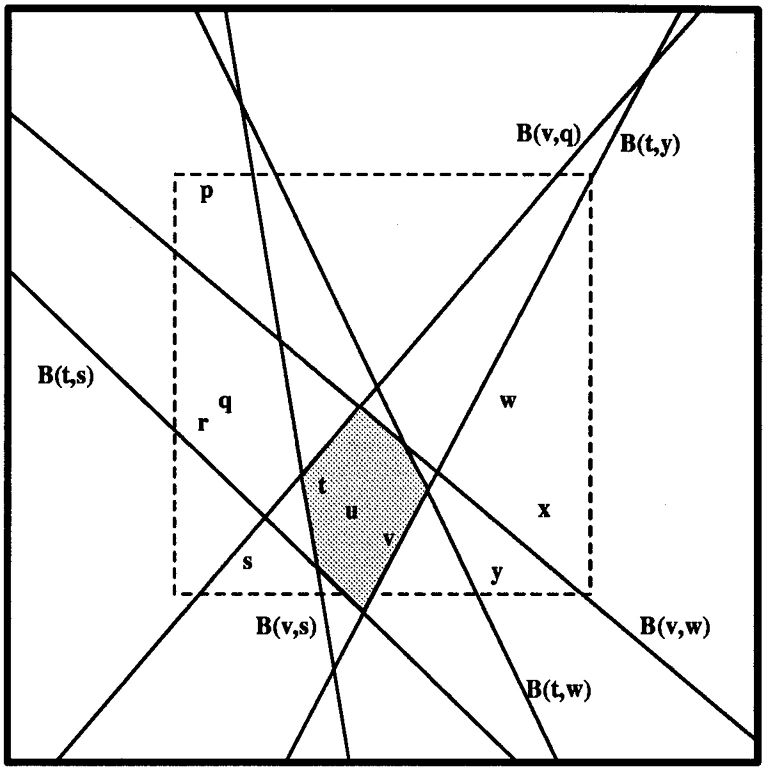

Figure 3.1: An Order-3 Voronoi Polytope $V_3^S(T) \subset \Re^2$

S is the same set whose order-1 Voronoi Diagram is shown in figure 1.1. $V_3^S(T)$ is shown (shaded region) for $T = \{\mathbf{t}, \mathbf{u}, \mathbf{v}\}$. The bounding lines of the 6 nonredundant halfplanes are labeled; the 15 redundant constraints are not shown.

Clearly, only the proper Voronoi polytopes will have a nonempty interior.

The boundary of $V_k^S(T)$ consists of hyperplanes (perpendicular bisectors), which intersect in lower dimensional faces. The generating set of any face is defined as follows:

Definition 3.6 *The generating set G of a face F of $V_k^S(T)$ is the union of all $\{\mathbf{p},\mathbf{q}\}$ for which $(\mathbf{p},\mathbf{q}) \in T \times (S-T)$ and $B(\mathbf{p},\mathbf{q})$ contains F:*

$$G = \bigcup_{\substack{\mathbf{p}\in T \\ \mathbf{q}\in S-T \\ F\subseteq B(\mathbf{p},\mathbf{q})}} \{\mathbf{p},\mathbf{q}\}$$

3.2 The Nondegenerate Situation

The order-k Voronoi Diagram of S becomes much more simple if we make the assumption that S is nondegenerate:

Definition 3.7 *A finite point set $S \subset \Re^d$ is said to be nondegenerate—or equivalently, S is said to be in general position—whenever the following two conditions are satisfied:*

- *Any subset of S of size $d+1$ (or less) is affinely independent.*
- *No $\mathbf{p} \in \Re^d$ is equidistant to more than $d+1$ points of S.*

When the conditions are not satisfied, S is said to be degenerate.

The vertex enumeration algorithm of chapter 4 will make the assumption that the points of S are in general position. This will be called the "nondegeneracy assumption". Section 3.3 discusses how degenerate sets could be handled. The facet enumeration algorithm of chapter 5 will not make this assumption. Later in this chapter, the properties of order-k Voronoi Diagrams are developed for the unrestricted situation (section 3.4) and for the nondegenerate situation (section 3.5). The following lemmas are immediate consequences of the nondegeneracy assumption, and they help to simplify the definition of the order-k Voronoi Diagram.

Lemma 3.1 *If S is nondegenerate then there is no improper order-k Voronoi polytope in V_k^S.*

Proof. Let $\mathbf{x} \in V_k^S(T)$ for any $V_k^S(T) \in V_k^S$, and let:

$$\delta = \max\{d(\mathbf{p}, \mathbf{x}) | \mathbf{p} \in T\}$$

It follows directly from definition 3.1 that the hypersphere $C = C(\mathbf{x}, \delta)$ separates T from $S - T$. Since S is nondegenerate, at most $d+1$ elements of S lie on the surface of C. Hence, we can construct a hypersphere C' which encloses T, excludes $S - T$, and has no points of S on its surface (proposition 2.3). The center of C' lies in $int(V_k^S(T))$; therefore $V_k^S(T)$ is a proper Voronoi polytope.

Q.E.D.

Lemma 3.2 *If S is nondegenerate then all V-facets of $V_k^S(T)$ are determined by strongly nonredundant halfspaces $\overline{H(\mathbf{p}, \mathbf{q})}$; where $\mathbf{p} \in T$, $\mathbf{q} \in (S - T)$.*

Proof. Let F be a facet of any $V_k^S(T)$ in the order-k Voronoi Diagram of S; let $B(\mathbf{p}, \mathbf{q})$ (where $\mathbf{p} \in T$ and $\mathbf{q} \in S - T$) be a perpendicular bisector containing F; let $\mathbf{x} \in F$ (note that $\mathbf{x}$ is not necessarily an interior point of F), and let $\delta = d(\mathbf{x}, \mathbf{p}) = d(\mathbf{x}, \mathbf{q})$. So, $\mathbf{p}$ and $\mathbf{q}$ lie on the surface of the hypersphere $C = C(\mathbf{x}, \delta)$.

It follows directly from definition 3.1 that C separates T from $S - T$. Since S is nondegenerate, at most $d + 1$ elements of S lie on the surface of C. Hence, we can construct a hypersphere C' which encloses $T - \{\mathbf{p}\}$, excludes $S - T - \{\mathbf{q}\}$, and has $\{\mathbf{p}, \mathbf{q}\}$ on its surface (proposition 2.3). The center of C' lies in $V_k^S(T)$, and in the *interior* of the facet F. Therefore, $\overline{H(\mathbf{p}, \mathbf{q})}$ is a strongly nonredundant halfspace.

Q.E.D.

As a consequence of lemma 3.2, all of the facets of any proper $V_k^S(T)$ are $(d - 1)$-dimensional, when S is nondegenerate. And since there are no improper Voronoi polytopes (by lemma 3.1), it follows that every facet in V_k^S for a nonredundant S is $(d-1)$-dimensional. Furthermore, definition 3.5 of a relevant point $\mathbf{p}$ with respect to $V_k^S(T)$ is equivalent to the following, in the nondegenerate case:

- $\mathbf{p} \in T$, and $\exists \mathbf{q} \in (S - T)$ such that $B(\mathbf{p}, \mathbf{q})$ is strongly nonredundant in $V_k^S(T)$, or
- $\mathbf{p} \in (S - T)$, and $\exists \mathbf{q} \in T$ such that $B(\mathbf{q}, \mathbf{p})$ is strongly nonredundant in $V_k^S(T)$

Lemma 3.3 *If $S \subset \Re^d$ is nondegenerate, then no m-face ($0 \leq m \leq d-1$) of the order-k Voronoi Diagram of S is equidistant to more that $d - m + 1$ points of S.*

Proof by contradiction. Assume that F, an m-face in the order-k Voronoi Diagram of S is equidistant to $G \subset S$ and $|G| = d - m + 2$.

- If F is a V-vertex ($m = 0$), then it is a point which is equidistant to $d+2$ points of S. Therefore S is degenerate (definition 3.7)
- Otherwise, the points of G lie on a common $(d-m)$-flat perpendicular to F. But any maximal set of affinely independent points in a $(d-m)$-flat has size $d - m + 1$. So $G \subset S$ is not affinely independent, and $|G| \leq d + 1$. Therefore S is degenerate (definition 3.7).

Q.E.D.

3.3 Degenerate Order-k Voronoi Diagrams

If the elements of a point set S are randomly chosen from any d-dimensional compact convex set, then S will be nondegenerate with probability 1. However, degeneracy may be introduced into a point set as a result of the round-off necessitated by finite-precision arithmetic, by deliberate construction, or may be present in the data of practical problems. Many geometric algorithms make the assumption that points sets are nondegenerate. Degenerate cases are then handled by an appeal to a *perturbation technique* [EM88] [Yap88] which essentially “fakes” nondegeneracy.

The techniques introduced in [EM88] [Yap88] add a small displacement $\mathbf{d_p}$ for each point $\mathbf{p} \in S$ so that the resulting set $\{\mathbf{p} + \mathbf{d_p} | \mathbf{p} \in S\}$ is free of degeneracy. The displacements are arbitrarily small and never actually computed but, rather, are used conceptually to "break ties" at a low level of the algorithm, so that a nondegenerate topology is simulated.

After perturbation, S is nondegenerate so lemmas 3.2 and 3.1 apply. This shows that perturbation has the effect of removing all improper Voronoi polytopes and all lower dimensional V-facets from the Voronoi diagram. Some of the improper polytopes are "promoted" to proper polytopes, while others are removed entirely. Similarly, some of the lower dimensional V-facets are "promoted" to $(d-1)$-dimensional facets, while others are removed entirely.

For example, figure 3.2 shows the order-1 Voronoi diagram of a set $S = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ of 4 degenerate points in the plane, as well as the corresponding perturbed set—obtained by moving **a** slightly to the left, leaving the other points where they were. Before perturbation (figure 3.2A) each of the four order-1 Voronoi polytopes has three facets: two are $d-1=1$ dimensional, and the other is 0 dimensional. For example, $V_1^S(\{\mathbf{a}\})$ has the facets:

$$\left.\begin{array}{r} V_1^S(\{\mathbf{a}\}) \cap B(\mathbf{a}, \mathbf{b}) \\ V_1^S(\{\mathbf{a}\}) \cap B(\mathbf{a}, \mathbf{d}) \end{array}\right\} \quad (d-1)\text{–dimensional}$$
$$V_1^S(\{\mathbf{a}\}) \cap B(\mathbf{a}, \mathbf{c}) = \{\mathbf{x}\} \quad (d-2)\text{–dimensional}$$

After the perturbation, the $(d-2)$-dimensional facets have been eliminated from $V_1^S(\{\mathbf{a}\})$ and from $V_1^S(\{\mathbf{c}\})$; on the other hand, the $(d-2)$-dimensional facets of $V_1^S(\{\mathbf{b}\})$ and $V_1^S(\{\mathbf{d}\})$ have been "promoted" to the $(d-1)$-dimensional facet which is labeled "E". The size of E has been exaggerated for the purpose of illustration: in fact, edge E will be arbitrarily small.

Figure 3.3 shows the order-2 Voronoi diagrams for the same point set, before and after perturbation. Initially (figure 3.3A) there are six order-2 Voronoi polytopes: four are proper polytopes, and two are improper. The proper polytopes are labeled in the figure. The improper ones are:

$$V_2^S(\{\mathbf{a}, \mathbf{c}\}) = V_2^S(\{\mathbf{b}, \mathbf{d}\}) = \{\mathbf{x}\}$$

The V-facets of these two polytopes are determined by all $2 \times 2 = 4$ possible perpendicular bisectors. After perturbation (figure 3.3B), $V_2^S(\{\mathbf{a}, \mathbf{c}\})$ disappears, while $V_2^S(\{\mathbf{b}, \mathbf{d}\})$ grows to an (arbitrarily) small polygon.

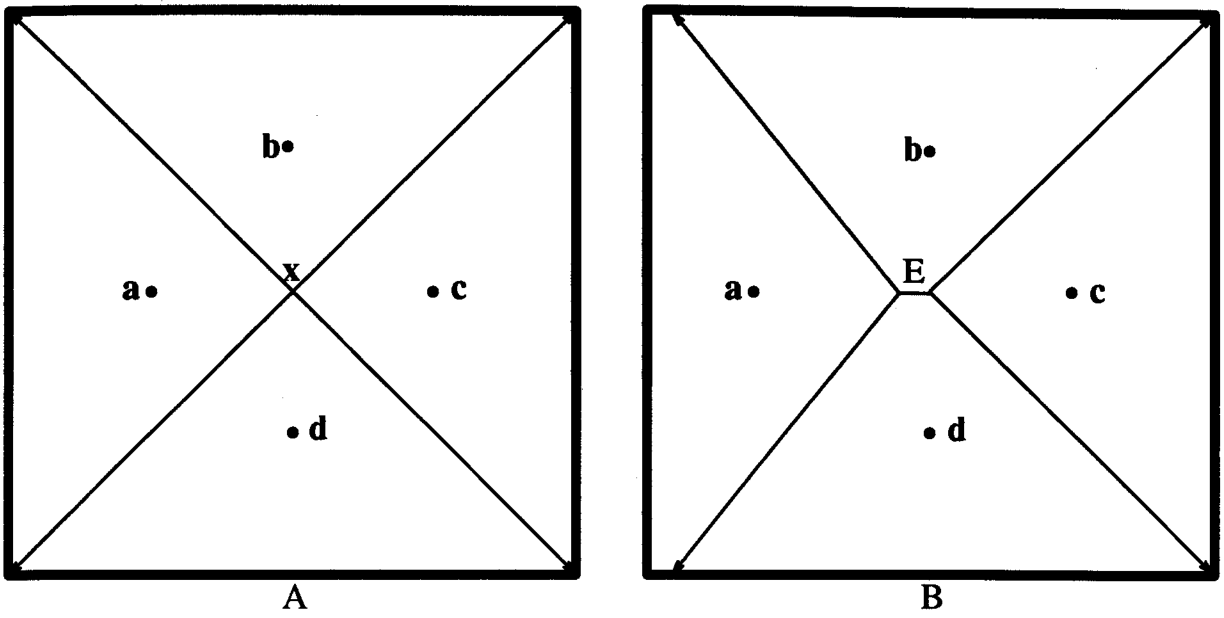

Figure 3.2: Order-1 Voronoi Diagrams of: (A) 4 Degenerate Points in the Plane, and (B) After Perturbation to Simulate Nondegeneracy

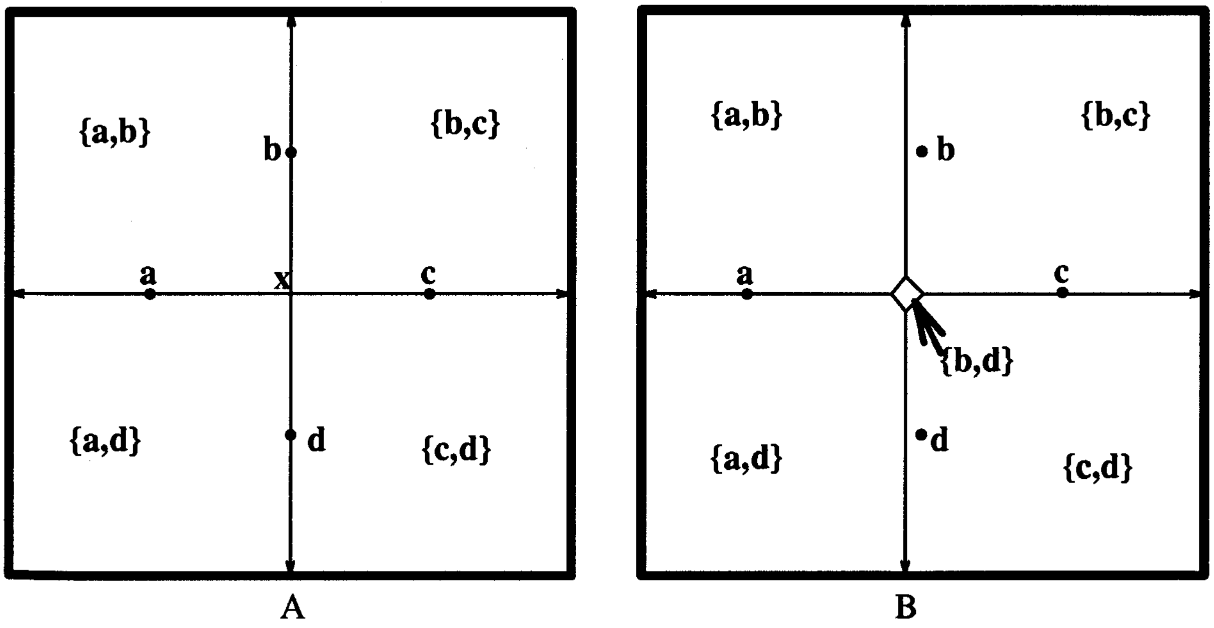

Figure 3.3: Order-2 Voronoi Diagrams of: (A) 4 Degenerate Points in the Plane, and (B) After Perturbation to Simulate Nondegeneracy

3.4 General Properties

The lemmas of this section apply to the order-k Voronoi Diagram of any point set S, whether degenerate or not.

Lemma 3.4 *The order-k Voronoi Diagram of S covers $\Re^d$.*

Proof. For any $\mathbf{x} \in \Re^d$, we can sort the elements $\mathbf{s_i} \in S$ in increasing order of $d(\mathbf{x}, \mathbf{s_i})$. Let this order be $(\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_n})$. It follows from definition 3.1 that $\mathbf{x} \in V_k^S(T)$ where $T = \{\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_k}\}$. So, $V_k^S(T)$ is non-empty and hence belongs to the order-k Voronoi Diagram of S. Since $\mathbf{x}$ was chosen arbitrarily, every $\mathbf{x} \in \Re^d$ belongs to some polytope in the order-k Voronoi Diagram.

Q.E.D.

Lemma 3.5 *If a point $\mathbf{r}$ of $V_k^S(T)$ lies on two distinct perpendicular bisectors $B(\mathbf{p_1}, \mathbf{q_1})$ and $B(\mathbf{p_2}, \mathbf{q_2})$ where $\mathbf{p_1}, \mathbf{p_2} \in T$ and $\mathbf{q_1}, \mathbf{q_2} \in (S - T)$, then $\mathbf{r}$ is equidistant to $\mathbf{p_1}, \mathbf{p_2}, \mathbf{q_1}$ and $\mathbf{q_2}$.*

Proof. Since $\mathbf{r} \in B(\mathbf{p_1}, \mathbf{q_1})$, then $d(\mathbf{r}, \mathbf{p_1}) = d(\mathbf{r}, \mathbf{q_1})$; since $\mathbf{r} \in B(\mathbf{p_2}, \mathbf{q_2})$, then $d(\mathbf{r}, \mathbf{p_2}) = d(\mathbf{r}, \mathbf{q_2})$. Since $\mathbf{r} \in V_k^S(T)$, it follows form definition 3.1 that $\mathbf{r}$ is at least as close to every point in T as to any point in $S-T$; in particular: $d(\mathbf{r}, \mathbf{p_1}) \leq d(\mathbf{r}, \mathbf{q_2})$ and $d(\mathbf{r}, \mathbf{p_2}) \leq d(\mathbf{r}, \mathbf{q_1})$.

$$d(\mathbf{r}, \mathbf{p_1}) \leq d(\mathbf{r}, \mathbf{q_2}) = d(\mathbf{r}, \mathbf{p_2}) \leq d(\mathbf{r}, \mathbf{q_1}) = d(\mathbf{r}, \mathbf{p_1})$$
$$\text{So,} \quad d(\mathbf{r}, \mathbf{p_1}) = d(\mathbf{r}, \mathbf{q_2}) = d(\mathbf{r}, \mathbf{p_2}) = d(\mathbf{r}, \mathbf{q_1})$$

Q.E.D.

Lemma 3.6 *Any point on a face F of $V_k^S(T)$ is equidistant to the points of the generating set of F.*

Proof. F is the intersection of some number, c, of perpendicular bisectors. For each pair $B(\mathbf{p_1}, \mathbf{q_1})$ and $B(\mathbf{p_2}, \mathbf{q_2})$, any $\mathbf{r} \in F$ is equidistant to $\{\mathbf{p_1}, \mathbf{q_1}\} \cup \{\mathbf{p_2}, \mathbf{q_2}\}$ (lemma 3.5). The union of all c such $\{\mathbf{p_i}, \mathbf{q_i}\}$ is, by definition, the generating set G of F. Inductively, any $\mathbf{r} \in F$ is equidistant to every element of G. Q.E.D.

Lemma 3.7 *For a Voronoi polytope $V_k^S(T)$ and a set $G \subset S$ such that:*

$$\begin{aligned} G \cap T &\neq \emptyset \\ G \cap (S-T) &\neq \emptyset \end{aligned}$$

The part of $V_k^S(T)$ which is equidistant to the elements of G is a face of $V_k^S(T)$. Furthermore, if $F \neq \emptyset$, then G is the generating set of F.

Proof. Let F denote the subset of $V_k^S(T)$ which is equidistant to G. Assume that F is nonempty; otherwise F is, trivially, the (-1)-face of $V_k^S(T)$.

$$F = \{\mathbf{x} \in V_k^S(T) | d(\mathbf{x}, \mathbf{g}) = \delta, \ \forall \mathbf{g} \in G\}$$

for some $\delta \in \Re$. It immediately follows that:

$$F = V_k^S(T) \cap \bigcap_{\substack{\mathbf{p} \in T \cap G \\ \mathbf{q} \in (S-T) \cap G}} B(\mathbf{p}, \mathbf{q})$$

That is, F is the intersection of $V_k^S(T)$ with some m-flat $(0 \leq m \leq d-1)$, whose generating set is G.

Suppose $\mathbf{x} \in F$ is in the interior of $V_k^S(T)$. Then,

$$d(\mathbf{x}, \mathbf{p}) < d(\mathbf{x}, \mathbf{q}); \qquad \forall \mathbf{p} \in T, \ \mathbf{q} \in S-T$$

But this contradicts the fact that $\mathbf{x}$ is equidistant to all points of G . Therefore, F lies on the boundary of $V_k^S(T)$.

Q.E.D.

Lemma 3.8 *The generating set of any m-face of $V_k^S(T)$, for $0 \leq m \leq d-1$, has size at least $d-m+1$.*

Proof. First note that a $(d-1)$-face lies within a single hyperplane, $B(\mathbf{p}, \mathbf{q})$ which is, by definition, equidistant to the 2 points $\mathbf{p}, \mathbf{q} \in S$.

Now, consider an arbitrary m-face, F, in $V_k^S(T)$, for $0 \leq m < d-1$, with generating set G. Let $c = |G|$, and let the elements of G be designated: $G = \{\mathbf{g_1}, \mathbf{g_2}, \ldots, \mathbf{g_c}\}$. So the affine hull of F is:

$$aff(F) = \bigcap_{i=1}^{c-1} B(\mathbf{g_i}, \mathbf{g_{i+1}})$$

Lemma 3.6 implies that the other $B(\mathbf{g_i}, \mathbf{g_j})$, for which $j \neq i+1$, are redundant in the specification of F.

The non-empty intersection of a j-flat with a hyperplane is either j or $(j-1)$ dimensional (lemma 2.1); so, inductively, the non-empty intersection of any c hyperplanes has dimension between $d-1$ and $d-c$. Since F is a non-empty m dimensional point set, the number of perpendicular bisectors which intersect in F is at least $d-m$. Therefore $c-1 \geq d-m$; *i.e.* $|G| = c \geq d-m+1$.

Q.E.D.

Lemma 3.9 *Given two faces F and F' of an order-k Voronoi polytope, whose generating sets are G and G', respectively: $F \subseteq F'$ if and only if $G \supseteq G'$.*

Proof. Let F, F' be faces of $V_k^S(T)$.

Assume that $F \subseteq F'$. Either $F = F'$—in which case $G = G'$—or $F \subset F'$—in which case F is the intersection of F' with some c additional perpendicular bisectors:

$$F = F' \cap B(\mathbf{p_1}, \mathbf{q_1}) \cap \cdots \cap B(\mathbf{p_c}, \mathbf{q_c})$$

It then follows from definition 3.6 that:

$$G = G' \cup \{\mathbf{p_1}, \ \ldots\ , \mathbf{p_c}, \mathbf{q_1}, \ \ldots\ , \mathbf{q_c}\}$$

Therefore, $G \supseteq G'$.

Conversely, assume that $G \supseteq G'$. For any $\mathbf{x} \in F$, $\mathbf{x}$ is equidistant to the elements of G (lemma 3.6), so *a fortiori* $\mathbf{x}$ is equidistant to the elements of $G' \subseteq G$. Since $\mathbf{x} \in V_k^S(T)$ and $\mathbf{x}$ is equidistant to the generating set of $F' \subset V_k^S(T)$, it immediately follows that $\mathbf{x} \in F'$. Therefore, $F \subseteq F'$.

Q.E.D.

Lemma 3.10 *The affine hull of any V-edge in V_k^S contains at most c V-vertices, where $c = \min\{|S| - d,\ 2k\}$.*

Proof. Let E be a V-edge with generating set $G \subset S$, $|G| = d$. According to lemma 3.9 and lemma 3.6, any V-vertex $\mathbf{v} \in E$ is equidistant to $G \cup \{\mathbf{p}\}$ for some $\mathbf{p} \in S - G$. Since there are only $|S| - d$ possible elements of $S - G$ to choose from, there can be no more than $(|S| - d)$ V-vertices in E.

Let the c V-vertices on E be ordered linearly as $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_c}$. For any $\mathbf{v_i}$ $(1 \le i \le c)$ there can be at most $k - 1$ points of S nearer to $\mathbf{v_i}$ than G is. Any $\mathbf{p} \in S - G$ is nearer to some $\mathbf{x} \in E$ whenever:

$$\begin{array}{rcll} d(\mathbf{p},\mathbf{x}) & < & d(\mathbf{g},\mathbf{x}); & \mathbf{g} \in G \\ i.e. \quad \sum_{i=1}^{d}(p_i - x_i)^2 & < & \sum_{i=1}^{d}(g_i - x_i)^2 & \\ i.e. \quad 2\sum_{i=1}^{d}(g_i - p_i)x_i & < & \sum_{i=1}^{d} g_i^2 - \sum_{i=1}^{d} p_i^2 & \end{array}$$

This is a linear function of $\mathbf{x}$. Hence, if $\mathbf{p} \in S$ is nearer than G to some $\mathbf{x}$ in the open line segment $(\mathbf{v_1}, \mathbf{v_c})$, then $\mathbf{p}$ must be nearer than G to either $\mathbf{v_1}$ or $\mathbf{v_c}$. A maximum of $2k - 2$ such points exist. A V-vertex lies between $\mathbf{v_1}$ or $\mathbf{v_c}$ only where one of these $2k - 2$ points becomes equidistant with G. Therefore, there can be no more than $2k$ V-vertices in E.

Q.E.D.

Lemma 3.11 *In the order-k Voronoi Diagram of S, for $T_i \neq T_j$: $V_k^S(T_i) \cap V_k^S(T_j)$ is either empty, or an m-face of $V_k^S(T_i)$ and of $V_k^S(T_j)$, for $0 \le m \le d - 1$.*

Proof. Assume that $F = V_k^S(T_i) \cap V_k^S(T_j)$ is non-empty, and let $\mathbf{x} \in \Re^d$ belong to F. Let $\mathbf{p}$, $\mathbf{q}$ be such that $\mathbf{p} \in T_i - T_j$, $\mathbf{q} \in T_j - T_i$. Since $\mathbf{x} \in V_k^S(T_i)$, then $d(\mathbf{x},\mathbf{p}) \le d(\mathbf{x},\mathbf{q})$; since $\mathbf{x} \in V_k^S(T_j)$, then $d(\mathbf{x},\mathbf{q}) \le d(\mathbf{x},\mathbf{p})$. So, let $\delta = d(\mathbf{x},\mathbf{q}) = d(\mathbf{x},\mathbf{p})$. Since $\mathbf{p}$ and $\mathbf{q}$ were chosen arbitrarily from $T_i - T_j$ and $T_j - T_i$, it follows that all points in $(T_i - T_j) \cup (T_j - T_i)$ are equidistant from F. Hence, by lemma 3.7, F is a face of both $V_k^S(T_i)$ and of $V_k^S(T_j)$, with generating set $(T_i - T_j) \cup (T_j - T_i)$.

Q.E.D.

The intersection of *any* two Voronoi polyhedra is a face in the Voronoi diagram. If $T_i = T_j$ then $V_k^S(T_i) \cap V_k^S(T_j) = V_k^S(T_i) = V_k^S(T_j)$, which is a d-face. An empty intersection is, by convention, a (-1)-face. The intersection of any other pair of Voronoi polyhedra is a face, by lemma 3.11.

Lemma 3.12 $\mathbf{x}$ *is an interior point of some facet* F *with generating set* $\{\mathbf{p},\mathbf{q}\}$ *in* V_k^S, *if and only if there exists a hypersphere centered at* $\mathbf{x}$, *with* $\mathbf{p}$ *and* $\mathbf{q}$ *on its surface, and with exactly* $k-1$ *points of* S *in its interior.*

Proof. Assume that $\mathbf{x}$ is an interior point of some facet F with generating set $B(\mathbf{p},\mathbf{q})$; let $T \subset S$ such that $F \subset V_k^S(T)$; and let $C = C(\mathbf{x},\delta)$ be the hypersphere centered at $\mathbf{x}$, with $\mathbf{p}$ and $\mathbf{q}$ on its surface. No point of $S - \{\mathbf{p},\mathbf{q}\}$ is on the surface of C, otherwise $\mathbf{x}$ would belong to an m-face (for $m \leq d-2$) and, hence, would not be an *interior* point of F.

It follows directly from definition 3.1 that C separates T and $S-T$. So, every point in T lies either on the surface or in the interior of C. We know that $\mathbf{p}$ is the only point of T which lies on the surface. Therefore, the $k-1$ points of $T-\{\mathbf{p}\}$ are in the interior of C. Furthermore, no point of $S-T$ is enclosed by C. Hence, exactly $k-1$ points of S are enclosed by C.

Conversely, assume that there is a hypersphere centered at $\mathbf{x}$, with $\mathbf{p}$ and $\mathbf{q}$ on its surface, and enclosing exactly $k-1$ points of S (let these points constitute the set R). It follows directly from definition 3.1 that $\mathbf{x}$ lies in both $V_k^S(R\cup\{\mathbf{p}\})$ and $V_k^S(R\cup\{\mathbf{q}\})$. Therefore, $\mathbf{x}$ lies on a face of V_k^S (lemma 3.11), and this face is $d-1$ dimensional (lemma 3.8) since its generating set has a size of 2. Furthermore, $\mathbf{x}$ is an *interior* point of this facet since it does not belong to any lower dimensional face.

Q.E.D.

The following lemma will be helpful in identifying the relevant (definition 3.5) points of S with respect to some $V_k^S(T)$, by solving a spherical separability problem.

Lemma 3.13 $\mathbf{p} \in S$ *is a relevant point in* $V_k^S(T)$ *if and only if there exists a hypersphere* C *such that* $\mathbf{p}$ *is on the surface of* C, *the elements of* T *are on the surface or interior of* C, *and the element of* $S-T$ *are on the surface or exterior of* C.

Proof. Assume that $\mathbf{p} \in S$ is relevant with respect to $V_k^S(T)$. By definition, $G = \{\mathbf{p},\mathbf{q}\}$ is the generating set of some facet F of $V_k^S(T)$, for some $\mathbf{q} \in S$. Let $\mathbf{x} \in F$, and let C be the hypersphere centered at $\mathbf{x}$ with $\mathbf{p}$ and $\mathbf{q}$ on its surface. So, we have on the surface of C one point from T and one from $S - T$. Since the center of C belongs to $V_k^S(T)$, it follows that the points of T cannot be exterior, and the points of $(S - T)$ cannot be interior.

Conversely, assume that there exists a hypersphere with $\mathbf{p} \in S$ on its surface, with the points of T either on the surface or interior, and with the points of $S - T$ either on the surface or exterior.

- If $\mathbf{p} \in T$, then we can expand C away from $\mathbf{p}$ until the first $\mathbf{q} \in S - T$ becomes a surface point (proposition 2.2). Of course, if there already exists some $\mathbf{q} \in (S - T)$ on the surface, we need not expand C.
- Otherwise we can contract C (proposition 2.1) towards $\mathbf{p}$ until some $\mathbf{q} \in T$ is a surface point.

The center of the hypersphere now lies on the perpendicular bisector B of $\mathbf{q}$ and $\mathbf{p}$, and lies in $V_k^S(T)$. Hence, B determines a facet of $V_k^S(T)$, Hence $\mathbf{p}$ (as well as $\mathbf{q}$) is relevant.

Q.E.D.

Lemma 3.14 *Exactly two Voronoi polytopes intersect in each $(d-1)$-dimensional facet of an order-k Voronoi Diagram.*

Proof. Let F be a $(d-1)$-dimensional facet in the order-k Voronoi Diagram of S; let the generating set of F be $\{\mathbf{p}, \mathbf{q}\}$; and let $\mathbf{f}$ be an interior point of F. According to lemma 3.12, there exists a hypersphere C centered at $\mathbf{f}$, with $\mathbf{p}$ and $\mathbf{q}$ on its surface, and with exactly $k-1$ points of S in its interior; let this set of $k-1$ points be denoted R.

Since R is the unique set of $k-1$ nearest neighbors of $\mathbf{f}$, any $V_k^S(T)$ containing $\mathbf{f}$ must satisfy $R \subset T$. Since exactly 2 elements of $S - R$ lie on the surface of C, we have exactly two choices for the k^{th} neighbor of $\mathbf{f}$.

Q.E.D.

Facets define a binary relationship among Voronoi polytopes, motivating definition 3.8, which will be useful in defining the "facet graph" in chapter 5. According to lemma 3.14, exactly two Voronoi polytopes are adjacent along any $(d-1)$-dimensional V-facet. Lower dimensional faces may lie in the intersection of more than two polytopes. Such faces are called V-facets only if they are the nonempty intersection of some $V_k^S(R \cup \{\mathbf{p}\})$ with $B(\mathbf{p},\mathbf{q})$, where R is a $(k-1)$-subset of S, and $\mathbf{p},\mathbf{q} \in S-R$ (definition 3.4). In this context, $V_k^S(R \cup \{\mathbf{p}\})$ will be said to be *adjacent* to $V_k^S(R \cup \{\mathbf{q}\})$.

A similar "adjacency" relationship exists among Voronoi vertices (definition 3.9), which will be useful in defining the "vertex graph" in chapter 4.

Definition 3.8 *$V_k^S(T)$ is said to be adjacent to $V_k^S(U)$ in the order-k Voronoi Diagram of S whenever:*

- $\{\mathbf{p}\} = T - U$
- $\{\mathbf{q}\} = U - T$
- $V_k^S(T) \cap B(\mathbf{p},\mathbf{q}) = V_k^S(U) \cap B(\mathbf{q},\mathbf{p})$ *is nonempty.*

Definition 3.9 *Two Voronoi vertices $\mathbf{v},\mathbf{u}$ in the order-k Voronoi Diagram of S are called adjacent whenever:*

- $\mathbf{v} \neq \mathbf{u}$, *and*
- $\mathbf{v},\mathbf{u} \in E$ *for some edge E of the Voronoi digram.*

3.5 Properties for Nondegenerate Point Sets

Lemma 3.15 *Under the nondegeneracy assumption: the generating set of any m-face of $V_k^S(T)$, for $0 \leq m \leq d-1$, has size exactly $d-m+1$.*

Proof. Let G be the generating set of an arbitrary m-face F of $V_k^S(T)$; so $|G| \geq d-m+1$ (lemma 3.8). The elements of $G \subset S$ are equidistant to F (lemma 3.6), and, since S is nondegenerate, F cannot be equidistant to more than $d-m+1$ points of S (lemma 3.3). So $|G| \leq d-m+1$. Q.E.D.

Table 3.1: Number of Facets Intersecting in m-Faces of Order-k Voronoi Polytopes in $\Re^d$ (under nondegeneracy assumption)

m	Type of Face F	Size of Generating Set	Number of Facets Intersecting In F		
			Minimum	Maximum (if $k \geq \lfloor \frac{d-m+1}{2} \rfloor$)	Maximum (otherwise)
0	vertex	$d+1$	d	$\lfloor \frac{d+1}{2} \rfloor \cdot \lceil \frac{d+1}{2} \rceil$	$k \cdot (d-k+1)$
1	edge	d	$d-1$	$\lfloor \frac{d}{2} \rfloor \cdot \lceil \frac{d}{2} \rceil$	$k \cdot (d-k)$
m		$d-m+1$	$d-m$	$\lfloor \frac{d-m+1}{2} \rfloor \cdot \lceil \frac{d-m+1}{2} \rceil$	$k \cdot (d-k-m+1)$
$d-2$	ridge	3	2	2	-
$d-1$	facet	2	1	1	-

The generating set G of an arbitrary m-face F ($0 \leq m \leq d-1$) is the union of the generating sets of the perpendicular bisectors containing F. Since the latter sets each contain one element from T and one from $(S-T)$, it follows that at least one of the points of G belongs to T, and at least one belongs to $(S-T)$. Under the nondegeneracy assumption, every distinct pair of points selected from G, such that one is from T and one the other from $(S-T)$, will determine a $(d-1)$-dimensional facet of $V_k^S(T)$ (this follows from the proof of lemma 3.2). Hence, $|T \cap G| \times |(S-T) \cap G|$ facets of an order-k Voronoi polytope intersect in an m-face with generating set G. Table 3.1 shows the minimum and maximum numbers of facets which may intersect in an m-face of $V_k^S(T)$.

Degeneracy exists in a polytope whenever more than d facets intersect in a vertex. Such a vertex—the intersection of the boundaries of more than d halfspaces—is degenerate in the sense that the polar dual (see section 2.6) of the polytope will contain a point for each of these halfspaces, all of which lie on a common hyperplane (lemma 2.3). Note that a nondegenerate point set does *not* imply nondegenerate order-k Voronoi polytopes, unless $k=1$ or $d=2$; this is confirmed by the information of table 3.1. Therefore, higher order (*i.e.* $k \geq 2$) Voronoi diagrams are inherently degenerate for $d \geq 3$.

Lemma 3.16 *Under the nondegeneracy assumption:* $\mathbf{q}$ *is an interior point of the edge with generating set* $\{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_d}\} \subset S$ *in* V_k^S *if and only if there is a hypersphere centered at* $\mathbf{q}$*, passing through the* d *points of* $\{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_d}\}$*, and enclosing* $k-t$ *points of* S*, for* $1 \leq t \leq d-1$ *(and, obviously* $t \leq k$*).*

Proof. Assume that $\mathbf{q}$ is an interior point of a Voronoi edge, E, with generating set $\{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_d}\}$. Let C be the hypersphere centered at $\mathbf{q}$ and passing through the d points of the generating set. Let F be one of the Voronoi facets which intersects E, and let $\mathbf{q}'$ be a point in the interior of F.

Since $E \subset F$, the generating set of F must be a subset of the generating set of E (lemma 3.9). Without loss of generality, assume that the generating set of F is $\{\mathbf{p_1}, \mathbf{p_2}\}$. Let C' be the hypersphere centered at $\mathbf{q}'$ and passing through $\mathbf{p_1}$ and $\mathbf{p_2}$. According to lemma 3.12, C' encloses $R \subset S$ (where $|R| = k-1$) and excludes $S - R - \{\mathbf{p_1}, \mathbf{p_2}\}$. Now, as $\mathbf{q}' \rightarrow \mathbf{q}$ so does $C' \rightarrow C$; as long as $\mathbf{q}$ and $\mathbf{q}'$ are distinct, the hypersphere encloses the subset R, and passes through only $\mathbf{p_1}$ and $\mathbf{p_2}$. Therefore, the points of R must be either enclosed by C or on C. We already know that $\mathbf{p_1}, \mathbf{p_2} \notin R$ are on C, and—under the nondegeneracy assumption—a total of d points are on C. Therefore, at most $d-2$ of the points of R may be on C; the rest remain enclosed by C—that is, at least $(k-1)-(d-2) = k-(d-1)$.

No point of $S - R - \{\mathbf{p_1}, \mathbf{p_2}\}$ may be enclosed by C, since the center of C belongs to $V_k^S(R \cup \{\mathbf{p_1}\})$ and $\mathbf{p_1}$ is a surface point. Therefore, at most $k-1$ points are enclosed by C.

Conversely, assume that there is a hypersphere, C, centered at $\mathbf{q}$ which passes through the d points of $G = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_d}\} \subset S$, and encloses a subset $I \subset S$ of $k-t$ points, for $1 \leq t \leq d-1$. We may choose any $G' \subset G$ of size t, and it follows that $\mathbf{q} \in V_k^S(G' \cup I)$. Since G' is not unique, $\mathbf{q}$ lies in the intersection of Voronoi polyhedra. Hence, by lemma 3.11, $\mathbf{q}$ belongs to an m-face, for $0 \leq m \leq d-1$. Since the size of the generating set of this face is d, it must be an edge (1-face), by lemma 3.15.

Q.E.D.

Lemma 3.17 *Under the nondegeneracy assumption:* $\mathbf{v}$ *is a Voronoi vertex with generating set* $\{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_{d+1}}\} \subset S$ *in* V_k^S *if and only if there is a hypersphere centered at* $\mathbf{v}$*, passing through the* $d+1$ *points* $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_{d+1}}$*, and enclosing* $k-t$ *points of* S*, for* $1 \leq t \leq d$ *(and, obviously* $t \leq k$*).*

Proof. Assume that $\mathbf{v}$ is a Voronoi vertex with generating set $\{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_d}\}$. Let C be the hypersphere centered at $\mathbf{v}$ and passing through the $d+1$ points of the generating set. Let F be one of the Voronoi facets which intersects E, and let $\mathbf{q}'$ be a point in the interior of F.

Since $\mathbf{v} \in F$, the generating set of F must be a subset of the generating set of $\mathbf{v}$ (lemma 3.9). Without loss of generality, assume that the generating set of F is $\{\mathbf{p_1}, \mathbf{p_2}\}$. Let C' be the hypersphere centered at $\mathbf{q}'$ and passing through $\mathbf{p_1}$ and $\mathbf{p_2}$. According to lemma 3.12, C' encloses $R \subset S$ (where $|R| = k-1$) and excludes $S - R - \{\mathbf{p_1}, \mathbf{p_2}\}$. Now, as $\mathbf{q}' \to \mathbf{q}$ so does $C' \to C$; as long as $\mathbf{v}$ and $\mathbf{q}'$ are distinct, the hypersphere encloses the subset R, and passes through only $\mathbf{p_1}$ and $\mathbf{p_2}$. Therefore, the points of R must be either enclosed by C or on C. We already know that $\mathbf{p_1}, \mathbf{p_2} \notin R$ are on C, and—under the nondegeneracy assumption—a total of $d+1$ points are on C. Therefore, at most $d-1$ of the points of R may be on C; the rest remain enclosed by C—that is, at least $(k-1)-(d-1) = k-d$.

No point of $S - R - \{\mathbf{p_1}, \mathbf{p_2}\}$ may be enclosed by C, since the center of C belongs to $V_k^S(R \cup \mathbf{p_1})$ and $\mathbf{p_1}$ is a surface point. Therefore, at most $k-1$ points are enclosed by C.

Conversely, assume that there is a hypersphere, C, centered at $\mathbf{v}$ which passes through the $d+1$ points of $G = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_{d+1}}\} \subset S$, and encloses a subset $I \subset S$ of $k-t$ points, for $1 \le t \le d$. We may choose any $G' \subset G$ of size t, and it follows that $\mathbf{v} \in V_k^S(G' \cup I)$. Since G' is not unique, $\mathbf{v}$ lies in the intersection of Voronoi polyhedra. Hence, by lemma 3.11, $\mathbf{q}$ belongs to an m-face, for $0 \le m \le d-1$. Since the size of the generating set of this face is $d+1$, it must be an vertex (0-face), by lemma 3.15.

Q.E.D.

The "symmetric difference" of two sets T and U is defined as $(T-U) \cup (U-T)$.

Lemma 3.18 *Under the nondegeneracy assumption: the symmetric difference between the generating sets of any two adjacent Voronoi vertices in V_k^S is equal to* 2.

Proof. Let $\mathbf{v_1}$ and $\mathbf{v_2}$ be two Voronoi vertices adjacent along an edge E, whose generating set is $G \subset S$, where $|G| = d$. The respective generating sets, G_1 and G_2, of $\mathbf{v_1}$ and $\mathbf{v_2}$ satisfy: $G_1 \supset G$ and $G_2 \supset G$ (lemma 3.9); $|G_1| = |G_2| = d+1$ (lemma 3.15).

Therefore, $\exists \mathbf{p_1} \in G_1$, $\mathbf{p_2} \in G_2$ such that:

$$G_1 = G \cup \{\mathbf{p_1}\} \qquad G_2 = G \cup \{\mathbf{p_2}\}$$

Furthermore $\mathbf{p_1} \neq \mathbf{p_2}$, otherwise $\mathbf{v_1}$ and $\mathbf{v_2}$ would not be distinct, violating definition 3.9. Therefore, $G_1 - G_2 = \{\mathbf{p_1}\}$ and $G_2 - G_1 = \{\mathbf{p_2}\}$.

Q.E.D.

Lemma 3.19 *Under the nondegeneracy assumption: for any Voronoi vertex* $\mathbf{v}$ *in* V_k^S*, let* G *be the generating set of* $\mathbf{v}$*, let* C *be the hypersphere centered at* $\mathbf{v}$ *with* G *on its surface, and let* I *be the subset of* S *enclosed by* C*. Then exactly one of the following will be true:*

- $|I| = k - 1$ *and* $\mathbf{v}$ *lies on exactly* $d + 1$ *Voronoi edges.*
- $|I| = k - d$ *and* $\mathbf{v}$ *lies on exactly* $d + 1$ *Voronoi edges.*
- $k - d < |I| \leq k - 2$ *and* $\mathbf{v}$ *lies on exactly* $2d + 2$ *Voronoi edges.*

Proof. Note that $k - d \leq |I| \leq k - 1$ (lemma 3.17); $|G| = d + 1$, and the size of the generating set of any edge containing $\mathbf{v}$ is d (lemma 3.15). Let $\{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_{d+1}}\}$ be the generating set G of $\mathbf{v}$.

Let $G_i = G - \{\mathbf{p_i}\}$, for $1 \leq i \leq d + 1$. The intersection of perpendicular bisectors of elements of any G_i will be a 1-flat (line) L_i that passes through $\mathbf{v}$. Any V-edge which contains $\mathbf{v}$, must be contained by one of these lines (lemma 3.9).

According to proposition 2.3, we may perturb C into a hypersphere C_i such that G_i is on its surface and $\mathbf{p_i}$ is either an interior or exterior point. The center $\mathbf{x_i}$ of C_i must lie on L_i; assume that $\mathbf{x_i}$ is close enough to $\mathbf{v}$ so that I remains enclosed, and $S - G - I$ remains excluded, by C_i.

Suppose that $\mathbf{p_i}$ is made an exterior point: then C_i has $|I|$ interior points. So, by lemma 3.16:

- if $|I| = k - d$ then $\mathbf{x_i}$ will not lie on a Voronoi edge.
- if $k - d < |I| \leq k - 1$ then $\mathbf{x_i}$ will lie on a Voronoi edge.

Now suppose that $\mathbf{p_i}$ is made an interior point: then C_i will enclose $|I| + 1$ points. So, by lemma 3.16:

- if $|I| = k - 1$ then $\mathbf{x_i}$ will not lie on a Voronoi edge.

- if $k - d \leq |I| \leq k - 2$ then $\mathbf{x_i}$ will lie on a Voronoi edge.

Therefore, if $|I| = k - 1$ or $|I| = k - d$, then $\mathbf{v}$ lies in exactly $d + 1$ V-edges. Otherwise, $k - d < |I| \leq k - 2$ and $\mathbf{v}$ lies in $2d + 2$ V-edges.

Q.E.D.

Lemma 3.19 motivates the following definition:

Definition 3.10 *In the order-k Voronoi Diagram of a nondegenerate point set: a V-vertex is called a terminal vertex whenever it is contained in $d + 1$ V-edges; it is called a cross vertex whenever it is contained in $2d + 2$ V-edges.*

Consider a point $\mathbf{x}$ as it moves in a straight line along a V-edge E, through a V-vertex and beyond, in the V_k^S for a nondegenerate set S. Let G be the generating set of E, and let C denote the hypersphere centered at $\mathbf{x}$ with G on its surface. Let I denote the maximal subset of S enclosed by C. When $\mathbf{x}$ encounters a V-vertex $\mathbf{v}$, some $\mathbf{p} \in S - G$ becomes the $(d + 1)^{st}$ surface point of C.

- If V is a cross vertex then, as $\mathbf{x}$ continues moving, it lies on some V-edge $E' \neq E$ whose generating set is G, and the corresponding hypersphere contains $I - \{\mathbf{p}\}$ (note that $\mathbf{p}$ may or may not be in I).

- If V is a terminal vertex then there are two cases to distinguish: either $|I| = k - d$ or $|I| = k - 1$: As $\mathbf{x}$ continues moving, it comes to lie in the interior of $V_k^S(I \cup G)$ in the former case, or the interior of $V_k^S(I \cup \{\mathbf{p}\})$ in the latter case.

Lemma 3.20 *Under the nondegeneracy assumption: for any two Voronoi vertices, $\mathbf{v_1}$ and $\mathbf{v_2}$ adjacent along some edge in the V_k^S: let C_1 and C_2 be the hyperspheres passing through their respective generating sets, and let I_1 and I_2 be the subsets of S enclosed by C_1 and C_2, respectively. Exactly one of the following will be true:*

1. $I_1 = I_2$

2. $|I_1| = |I_2|$, *and the symmetric difference between I_1 and I_2 is* 2.

3. $|I_1| = |I_2| + 1$ *and* $I_1 \cap I_2 = I_2$.

4. $|I_2| = |I_1| + 1$ *and* $I_1 \cap I_2 = I_1$.

Proof. Let E be the V-edge along which the V-vertices $\mathbf{v_1}$ and $\mathbf{v_2}$ are adjacent. Let G_1, G_2 and G be the respective generating sets of $\mathbf{v_1}$, $\mathbf{v_2}$ and E. Let I be the subset of S enclosed by any hypersphere C centered in the interior of E, and having G on its surface. According to lemma 3.9:

$$G = G_1 - \{\mathbf{p_1}\} = G_2 - \{\mathbf{p_2}\}$$

for some $\mathbf{p_1}, \mathbf{p_2} \in S$. According to proposition 2.3, C can be obtained from C_1 (or C_2) by making $\mathbf{p_1}$ (respectively, $\mathbf{p_2}$) either an interior or an exterior point, and leaving the relative positions of every other point the same. In either case:

$$\begin{aligned} I_1 &= I - \{\mathbf{p_1}\} \\ I_2 &= I - \{\mathbf{p_2}\} \end{aligned}$$

Now, the four cases in the lemma may be derived from the following four scenarios, which are mutually exclusive and exhaustive:

1. $(\mathbf{p_1} \notin I$ and $\mathbf{p_2} \notin I)$ or $\mathbf{p_1} = \mathbf{p_2}$.

2. $\mathbf{p_1}, \mathbf{p_2} \in I$ and $\mathbf{p_1} \neq \mathbf{p_2}$.

3. $\mathbf{p_1} \notin I$ and $\mathbf{p_2} \in I$.

4. $\mathbf{p_1} \in I$ and $\mathbf{p_2} \notin I$.

Q.E.D.

The four scenarios enumerated in lemma 3.20 are illustrated in figure 3.4, for two V-vertices adjacent along a V-edge in an order-3 Voronoi diagram. Note that scenarios (3) and (4) are equivalent, by symmetry, so there are only three distinct cases to illustrate.

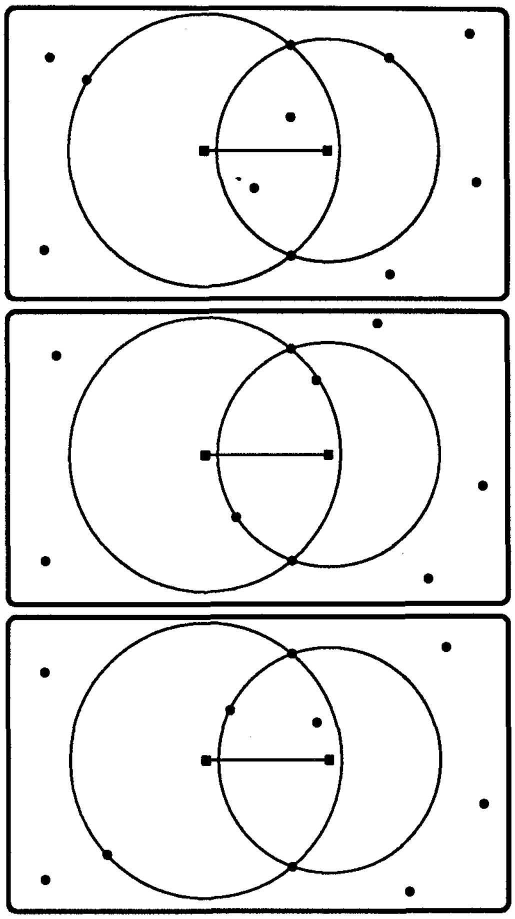

Figure 3.4: Illustration Of Adjacent Vertices

These three panels illustrate all possible scenarios enumerated in lemma 3.20. A V-edge of an order-3 Voronoi Diagram in $\Re^2$ is shown, with two adjacent V-vertices (denoted by square points). Data points are denoted by round points. The circles passing through the generating set of each V-vertex are shown. In any case, both circles must pass through two of the same data points (the generating set of the V-edge).

Top Frame Both circles enclose the same point set.

Middle Frame The symmetric difference of the enclosed point sets in 2.

Bottom Frame The symmetric difference of the enclosed point sets is 1.

Note, also, that any circle centered along the V-edge (which is also a V-facet in $\Re^2$) passes through two data points and encloses $k - 1 = 2$ data points.

Lemma 3.21 *Under the nondegeneracy assumption: for any V-vertex* $\mathbf{v}$*, with generating set* G*, in* V_k^S*:* $\mathbf{v}$ *belongs to* $V_k^S(T)$ *if and only if the hypersphere centered at* $\mathbf{v}$ *with* G *on its surface has* $I \subset S$ *in its interior and* $S - I - G$ *in its exterior, such that* $G \cup I \supset T \supset I$*.*

Proof. Let $\mathbf{v}$ be a V-vertex with generating set G. $|G| = d + 1$ by lemma 3.15.

Assume that $\mathbf{v} \in V_k^S(T)$. Then, by lemma 3.17, the hypersphere centered at $\mathbf{v}$ with G on its surface encloses $I \subset S$, where $|I| = k - t$, for $1 \leq t \leq d$ (and the hypersphere excludes the remaining points $S - G - I$). So the $k - t < k$ nearest neighbors of $\mathbf{v}$ are I; the $k - t + d + 1 > k$ neighbors of $\mathbf{v}$ are $G \cup I$. Furthermore, the k nearest neighbors of $\mathbf{v}$ are T, by virtue of $\mathbf{v} \in V_k^S(T)$. Therefore:

$$I \subset T \subset (G \cup I)$$

Conversely, assume that the hypersphere centered at $\mathbf{v}$ with G on its surface has $I \subset S$ in its interior and $S - I - G$ in its exterior. Let $t = k - |I|$; then $1 \leq t \leq d$ (lemma 3.17). Then we may select any t-subset G' of G and it follows that $\mathbf{v} \in V_k^S(G' \cup I)$. That is, $\mathbf{v} \in V_k^S(T)$ for any $(G \cup I) \supset T \supset I$.

Q.E.D.

Chapter 4

Vertex Enumeration Algorithm

In this chapter an algorithm is presented which enumerates all of the V-vertices of the order-k Voronoi Diagram (V_k^S) of a set S of n points in $\Re^d$. The algorithm computes V_k^S directly: that is, V_{k-1}^S is not needed as a preliminary step. This appears to be the first algorithm which directly computes V_k^S for $d > 2$. It is assumed that the point set is nonredundant. The running time of the algorithm is $O(d^2 n + d^3 \log n)$ per vertex, regardless of the value of k. The algorithm traverses the"vertex graph" of the Voronoi diagram.

Definition 4.1 *The vertex graph $G = (V, E)$ of a given Voronoi Diagram V_k^S has:*

- *$v \in V$ for each Voronoi vertex* $\mathbf{v}$ *of V_k^S*
- *$(v_i, v_j) \in E$ for all $v_i, v_j \in V$ such that v_i and v_j are adjacent along an edge in V_k^S*

Each V-vertex $\mathbf{v}$ lies on either $d+1$ or $2d+2$ V-edges (lemma 3.19), dependent only upon the number of points of S which are enclosed by the hypersphere corresponding [1] to $\mathbf{v}$. Some of these edges may extend to infinity; the others terminate in some V-vertex adjacent to $\mathbf{v}$. Hence, the degree of any node of the vertex graph is bounded by $2d+2$. Furthermore, the graph is connected, as shown in [AB83, theorem 2] (although this paper is only concerned with the order-1 Voronoi Diagram, the proof of connectedness makes no assumption about k).

[1]For ease of expression, the hypersphere "corresponding to" a V-vertex $\mathbf{v}$ will mean the hypersphere centered at $\mathbf{v}$ with the points of the generating set of $\mathbf{v}$ on its surface.

The vertex enumeration algorithm searches for all adjacent V-vertices of each V-vertex **v**—the circumcenter of some $(d+1)$-subset $G \subset S$. The proof of lemma 3.19 shows that the adjacent V-vertices lie on one of the $\binom{d+1}{d} = d+1$ lines, L_i, each of which is equidistant to some d-subset of $G_i \subset G$. Therefore, by considering each L_i in turn—and considering either one, or the other, or both directions, depending upon the number of points of S enclosed by the hypersphere corresponding to **v**—we can locate all of the V-vertices adjacent to **v**. That is, we find the point $\mathbf{v_i}$ of L_i which is equidistant to G_i and some $(d+1)^{st}$ point of $S - G$, such that $\mathbf{v_i}$ is as close as possible to **v**.

A planar version of this algorithm was presented in [Bha83]. This algorithm does not assume that the point set is nondegenerate, and the running time is bounded by $O(nk(n-k))$. The running time may also be bounded in terms of the output size: $O(vn + e\log v)$, where e is the number of V-edges.

If S is degenerate, we can make the following claims about any V-vertex **v** and V-edge E. These claims are analogous to lemmas 3.16, 3.17 and 3.19, and can be proved in the same way that the lemmas were, without making the nondegeneracy assumption. Let G be the generating set of **v**, and let I be the subset of S enclosed by the hypersphere corresponding to **v**. Let E be an edge with generating set G', such that **v** lies on E. Let I' be the subset of S enclosed by any hypersphere centered in the interior of the edge E with G' on its surface.

- $|G| = s \geq d+1$
- $|I| = k - t;\ 1 \leq t \leq s-1$
- $|G'| = s';\ d \leq s' \leq s-1$
- $|I'| = k - t + t' \begin{cases} t - s' + 1 \leq t' \leq t-1, & \text{and} \\ t' + s' \leq s \end{cases}$
- I' consists of the $k-t$ elements of I and t' additional elements of G

Furthermore, the points of G' must lie on a hyperplane, for which E is a surface normal directed towards the open halfspace which contains the t' points of $I' - I$.

Therefore, in order to identify all of the V-edges arising from **v**, we must identify all of the maximal subsets of G which lie on a hyperplane for which some t' points of G lie in one of the open halfspaces. This is greatly simplified by the nondegeneracy assumption, which implies that $|G| = d+1$, $|G'| = d$ and $t' \in \{0,1\}$. So, *any* d-subset of G will lie on a hyperplane H such that the $(d+1)^{st}$ point $\mathbf{g}'$ of G lies in one of the open halfspaces determined by H. Let $\mathbf{n}$ denote the surface normal of H directed towards the halfspace containing $\mathbf{g}'$. Then:

- $\{\mathbf{v}+\tau\mathbf{n}|\tau>0\}$ contains a V-edge $\Longleftrightarrow 1 \leq t-1$ (*i.e.* $t \geq 2$)
- $\{\mathbf{v}+\tau\mathbf{n}|\tau<0\}$ contains a V-edge $\Longleftrightarrow t-d+1 \leq 0$ (*i.e.* $t \leq d-1$)

This theory was developed more formally, under the nondegeneracy assumption, in chapter 3.

Lemma 3.21 tells us how to determine the Voronoi polytopes which contain any V-vertex $\mathbf{v}$ with generating set G, such that the corresponding hypersphere encloses $I \subset S$. Such a vertex will be contained by any $V_k^S(T)$ for which T properly contains I, and T is properly contained by $G \cup I$.

4.1 The Planar Case

The order-k Voronoi Diagram of $S \subset \Re^2$ has several simplifying features. Firstly, there can be no cross vertex (see definition 3.10) when S is nondegenerate. This follows directly from lemma 3.19: any hypersphere centered at a cross vertex and passing through the points of its generating set must contain c points, for $k-d+1 \leq c \leq k-2$; this clearly cannot occur when $d=2$.

Secondly, the V-edges are also V-facets, so the generating set of any V-edge has size 2, whether S is degenerate or not. This also follows from the observation that all circles centered along a V-edge E must pass through the points of the generating set G. But the intersection of any two such circles will contain exactly two points. In higher dimensions, the (nonempty) intersection of two m-dimensional hyperspheres is an $(m-1)$-dimensional hypersphere, so there is no finite limit on the number of points which may be equidistant to a V-edge. This demonstrates the difficulty with generalizing the algorithm to deal with degenerate point sets in higher dimensional spaces.

The following are proven in [Bha83], and also follow from the comments made on the more general case, at the beginning of this chapter.

1. any hypersphere centered along a V-edge in a 2-dimensional order-k Voronoi Diagram and passing through the (two) points of its generating set will contain exactly $k-1$ points in its interior.

2. the hypersphere centered at a V-vertex **v** with the points of the generating set G on its surface contains $I \subset S$, such that: $\begin{cases} |G| = s \geq 3 \\ |I| = k - t \quad 1 \leq t \leq s - 1 \end{cases}$

It follows that any edge E arising from **v** is associated with a hypersphere centered along E, with 2 points of G on its surface, and enclosing $t-1$ points of G (in addition to enclosing the $k-t$ points of $S-G$). Therefore, there are exactly s edges arising from **v**: for each such edge E, the line L which is perpendicular to E and passes through the points of its generating set must contain $t-1$ points of I in one of its open halfplanes.

The algorithm of [Bha83] moves to adjacent V-vertices by considering all possible s edges arising from a given V-vertex whose generating set has size s.

4.2 The Nondegenerate d-Dimensional Case

Algorithm **Vertex_Enumeration**, to enumerate all V-vertices of the order-k Voronoi Diagram of a nondegenerate point set $S = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_d\}$ is shown in figure 4.1. Each V-vertex **v** is "expanded" in turn—that is, each edge arising from **v** is examined for an adjacent V-vertex. The complexity of the algorithm is $O(d^2n + d^3 \log n)$, and the space requirement in $O(d)$, per V-vertex.

Each V-vertex in the output will be represented either by its coordinates in $\Re^d$, or as a list of the indices of the $d+1$ points of S which make up its generating set (or both). After the V-vertices have been enumerated, we can determine the Voronoi polytopes on which each V-vertex **v** lies in $O(nd)$ time if the former representation is used, and $O(d^3 + nd)$ time in the latter case.

Small modifications, which do not add to the time complexity, can allow the output to include:

- a direct indication of the Voronoi polytopes on which each V-vertex lies.
- construction of the vertex graph.

The former modification, however, does add to the space complexity of the algorithm.

The algorithm makes use of two data structures (see table 4.2): a balanced search tree (denoted Ψ) [AHU83, section 5.4], and a stack (denoted Φ) [AHU83, section 2.3]. In a balanced search tree containing m records, any insertion, deletion or query operation requires $O(\log m)$ probes (in the worst case and average case). If this algorithm were implemented,

Table 4.1: Data Structures for Algorithm **Vertex_Enumeration**

(Algorithm is presented in figure 4.1)

Ψ	balanced search tree, containing a record for each "known" V-vertex **v** • search key: $\bar{G}$ = lexicographically sorted list of indices $(i_1, i_2, \ldots, i_{d+1})$ (where the generating set of **v** is $G = \{p_{i_j} \mid 1 \leq j \leq d+1\}$)
Φ	stack of records $(\bar{G}, t)$ for V-vertices **v** left to be processed • $\bar{G}$ = lexicographically sorted indices of the generating set of **v** • the hypersphere corresponding to **v** encloses exactly $k - t$ points of S

the "open hashing" technique [AHU83, section 4.7] would probably be preferable to the balanced search tree, since the expected number of probes required for any insertion, deletion or query is constant (although the worst case requires $O(m)$ probes).

The balanced search tree Ψ is used to hold all of the V-vertices which have been discovered by the algorithm—*i.e.* those which are either on the stack, or have already been popped from the stack. This will be used during the expansions of a given vertex, to "query" whether or not some adjacent vertex has already been discovered. Since any V-vertex is uniquely specified by the $d + 1$ elements of its generating set, the tree may be keyed by the lexicographically-sorted indices of the generating sets. Since v—the total number of V-vertices—is an upper bound on the size of the tree, it follows that any insertion, deletion, or query operation will require $O(\log v)$ probes. Each probe requires $O(d)$ comparisons of indices, and v is trivially bounded by $\binom{n}{d+1} \in O(n^d)$. So the complexity of any insertion, deletion or query is $O(d^2 \log n)$.

The stack Φ is used to store all of the vertices which have been discovered, but not yet expanded. Every vertex is pushed onto Φ as soon as it is discovered for the first time. For each such vertex we need to know its generating set, and the number $k - t$ of points of S which are interior to the hypersphere corresponding to **v**. The size of the generating set will be $d + 1$, and the number t will be between 1 and d (lemma 3.17). The generating set will be represented by the ordered list of indices, in the same way as it is represented in Ψ. Note that the set $\bar{G}$ of indices for a given V-vertex need only be stored once in memory, and the appropriate fields of both Ψ and Φ could be implemented as pointers.

The procedure **vertex**($\mathbf{v}, \bar{G}, t$) is called for every V-vertex $\mathbf{v}$ discovered adjacent to the V-vertex currently being expanded. This procedure will check if $\bar{G}$ is already in Ψ and, if not a new record for $\mathbf{v}$ is added to the stack, and to the search tree; the procedure will also generate output, indicating either the coordinates of $\mathbf{v}$ or the indices $\bar{G}$ of the generating set of $\mathbf{v}$. The complexity of this procedure is dominated by the (at most two) operations on the search tree. Hence the complexity is $O(d^2 \log n)$.

```
procedure vertex($\mathbf{v}, \bar{G}, t$)
    • if $\bar{G}$ is not in $\Psi$ then:
        • insert $\bar{G}$ into $\Psi$
        • push the record $(\bar{G}, t)$ onto $\Phi$
    • perform an output operation, reporting the new V-vertex $\mathbf{v}$
```

Finding the initial V-vertex (in step (1) of the algorithm) can be performed in $O(d^2n)$ time as follows:

- find a convex hull facet F of S, using the algorithm of Chand and Kapur [CK70]. ($O(d^2n)$ time)
- let G' be the d-subset of S which lies on F
- compute the line $L = \{\mathbf{x} = \mathbf{p} + \tau\mathbf{y} | \tau \in \Re\}$ which is equidistant to G', where $\mathbf{y}$ is the surface normal of F directed away from $CH(S)$. ($O(d^3)$ time)
- compute τ_j ($\forall j \in \bar{S} - \bar{G}'$) such that $\mathbf{p} + \tau_j\mathbf{y}$ is equidistant to $G' \cup \{\mathbf{p_j}\}$. ($O(nd)$ time)
- find the ith largest τ_j ($j \in \bar{S} - \bar{G}'$) for $k - d + 1 \leq i \leq k$. ($O(n)$ time [BFP+72])
- then, $\mathbf{v_0} = \mathbf{p} + \tau_j\mathbf{y}$ will be the circumcenter of the $d+1$ points $G_0 = G' \cup \{\mathbf{p_j}\}$, and the corresponding hypersphere will enclose $i - 1$ points, $I_0 \subset S$. Hence, $\mathbf{v_0}$ is a V-vertex (lemma 3.17).

```
Algorithm Vertex_Enumeration

Input:   $d, k, n$
         $S = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}\}$: a nondegenerate point set in $\Re^d$
         (let $\bar{T}$ denote the indices of any $T \subseteq S$)

Output:  A list of the V-vertices of $V_k^S$

1. find an initial V-vertex $\mathbf{v_0}$ with generating set $G_0 \subseteq S$; let $I_0 \subset S$ denote the points
   enclosed by the hypersphere corresponding to $\mathbf{v_0}$.
2. insert $\bar{G}_0$ into $\Psi$
3. push the record $(\bar{G}_0, |I_0|)$ onto $\Phi$
4. while $\Phi$ not empty:
   (a) pop top record $(\bar{G}, t)$ from $\Phi$; $\bar{G} = (i_1, i_2, \ldots, i_{d+1})$
   (b) compute $\mathbf{v}$, the circumcenter of $G$
   (c) $\delta \leftarrow d(\mathbf{v}, \mathbf{g})$, for any $\mathbf{g} \in G$
   (d) for $j = 1, 2, \ldots, d+1$, compute $\mathbf{v_j}$ such that:
       • $\mathbf{v} + \tau \mathbf{v_j}$ is the equation of the line which is the circumcenter of $G_j = G - \{\mathbf{g_{i_j}}\}$
       • a hypersphere centered at $\mathbf{v} + \tau \mathbf{v_j}$ with $G_j$ on its surface encloses $\mathbf{g_{i_j}}$, whenever $\tau > 0$.
   (e) for $j = 1, 2, \ldots, d+1$:
       i. for all $h \in \bar{S} - \bar{G}$:
          • let $\mathbf{a} \cdot \mathbf{x} = \beta$ be the equation of $B(\mathbf{p_h}, \mathbf{g})$, for any $\mathbf{g} \in G_j$.
          • set $\tau_h = \frac{\beta - \mathbf{a} \cdot \mathbf{v}}{\mathbf{a} \cdot \mathbf{v_j}}$.
       ii. if $1 < t \leq d$ and $\exists \tau_h > 0$ $(h \in \bar{S} - \bar{G})$ then:
          • let $r$ be the index for which $\tau_r > 0$ and $\tau_r < \tau_h, \forall \tau_h > 0$ $(h \in \bar{S} - \bar{G})$.
          • if $d(\mathbf{v}, \mathbf{p_r}) < \delta$ then: call procedure vertex$(\mathbf{v} + \tau_r \mathbf{v_j}, \bar{G}_j \cup \{r\}, t)$
          • otherwise: call procedure vertex$(\mathbf{v} + \tau_r \mathbf{v_j}, \bar{G}_j \cup \{r\}, t-1)$
       iii. if $1 \leq t < d$ and $\exists \tau_h < 0$ $(h \in \bar{S} - \bar{G})$ then:
          • let $r$ be the index for which $\tau_r < 0$ and $\tau_r > t_h, \forall t_h < 0$ $(h \in \bar{S} - \bar{G})$.
          • if $d(\mathbf{v}, \mathbf{p_r}) < \delta$ then: call procedure vertex$(\mathbf{v} + \tau_r \mathbf{v_j}, \bar{G}_j \cup \{r\}, t+1)$
          • otherwise: call procedure vertex$(\mathbf{v} + \tau_r \mathbf{v_j}, \bar{G}_j \cup \{r\}, t)$
```

Figure 4.1: Algorithm to Enumerate All V-Vertices in the Order-k Voronoi Diagram of $S \subset \Re^d$

(see also table 4.2)

Step (2) and step (3) are trivial $O(1)$ operations (since the search tree is, at this stage, empty).

Step (4) is performed once for each V-vertex: *i.e.* v times. The overall complexity of each iteration is $O(d^2n + d^3 \log n)$. Each sub-step has complexity as follows:

4a $O(1)$

4b $O(d^3)$, by solving the following system, where $G = \{\mathbf{q_1}, \mathbf{q_2}, \dots, \mathbf{q_{d+1}}\}$:

$$\underbrace{\begin{bmatrix} \leftarrow & \mathbf{a_1} & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{a_d} & \rightarrow \end{bmatrix}}_{B} \mathbf{v} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}; \quad B(\mathbf{q_i}, \mathbf{q_{d+1}}) = \{\mathbf{x} | \mathbf{a_i} \cdot \mathbf{x} = \beta_i\}$$

4c $O(d)$

4d $O(d^3)$: $\mathbf{v_1} \dots \mathbf{v_d}$ are the d column vectors of B^{-1}. In order to compute $\mathbf{v_{d+1}}$, we could construct a matrix similar to B—for example with row vectors corresponding to $B(\mathbf{q_i}, \mathbf{q_1})$, for $2 \le i \le d+1$—whose inverse has column vectors $\mathbf{v_2} \dots \mathbf{v_{d+1}}$.

4e $O(d)$ repetitions, each of complexity $O(dn + d^2 \log n)$:

(i) $O(n)$ computations, each of complexity $O(d)$

(ii)/(iii) $O(n)$ to determine the minimal $|t_h|$; $O(d)$ to compute the distance from $\mathbf{v}$ to the new generating set point; $O(d^2 \log n)$ for procedure **vertex** (dominated by operations on the balanced search tree) .

Theorem 4.1 *Algorithm* **Vertex_Enumeration** *enumerates all v V-vertices of the order-k Voronoi Diagram of a nondegenerate set of n points in $\Re^d$ in time $O(vd^2n + vd^3 \log n)$*

Proof. Proof of correctness follows from the facts that the vertex graph is connected [AB83], the degree of each vertex is at most $2d + 2$ (lemma 3.19), and from the comments made in the proof of lemma 3.19 which tell us how to compute any adjacent V-vertices.

The complexity was calculated above.

Q.E.D.

Several issues related to algorithm **Vertex_Enumeration** are discussed below:

4.2.1 Computing Each V-edge Only Once

The $O(d^2n)$ determination of adjacent V-vertices has a built-in inefficiency of a factor of 2. Suppose a V-vertex **v** has been discovered during the expansion of another V-vertex **u**: then during the expansion of **v**, the algorithm still spends $O(nd)$ time to "rediscover" **u**. To avoid this, we could associate $2d+2$ flags with the record in Ψ, corresponding to any V-vertex **v** with generating set indexed by $(i_1, i_2, \ldots, i_{d+1})$. Each flag corresponds to one possible edge arising from **v**; say the flags are denoted $f_1^+, \ldots, f_{d+1}^+$ and $f_1^-, \ldots, f_{d+1}^-$, such that:

- f_j^+ is on $\Longleftrightarrow$ **v** has already been discovered adjacent to some V-vertex **u** whose generating set includes $\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_{j-1}}, \mathbf{p}_{i_{j+1}}, \ldots, \mathbf{p}_{i_{d+1}}$, and $\mathbf{p}_{i_j}$ is interior the hypersphere corresponding to **u**
- f_j^- is on $\Longleftrightarrow$ **v** has already been discovered adjacent to some V-vertex **u** whose generating set includes $\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_{j-1}}, \mathbf{p}_{i_{j+1}}, \ldots, \mathbf{p}_{i_{d+1}}$, and $\mathbf{p}_{i_j}$ is exterior the hypersphere corresponding to **u**

These flags can be set after any successful query, in procedure **vertex**, for the existence of the record associated with **v**, during the expansion of **u**. Selecting the correct flag can be done by noting the position j at which there is an element in the index list of **v** which is not in the corresponding list for **u**, and by noting whether or not this element came from the interior or the exterior of the hypersphere corresponding to **u** (this has already been done in the distance test of steps 4e(ii) or 4e(iii)).

Immediately after step (4a) when a record has been popped from the stack, we can find the corresponding record in the search tree; this adds nothing to the overall complexity. If all $2d+2$ flags are on (in the case of a cross vertex) or if $d+1$ flags are on (in the case of a terminal vertex), then we can skip steps (4b) through (4e). Otherwise, we need only search along those edges whose flags are off.

4.2.2 Reducing The Size of the Search Tree

If records are never deleted from the search tree Ψ, it could become quite large when the input set S is large. However, after all of the adjacent V-vertices of a given V-vertex **v** have been expanded, it is no longer necessary to keep the record for **v** in Ψ. After expanding **v**, we know exactly how many adjacent V-vertices there are; after all of them have been expanded, it is possible to delete the record from Ψ. Suppose that there is a counter $c(\mathbf{v})$ associated with the record for each V-vertex **v**, which represents the number of adjacent vertices discovered while expanding **v**, less the number of times **v** has been discovered adjacent to some other vertex.

- in step (2): $c(\mathbf{v_0}) \leftarrow 0$
- in procedure **vertex**, if a new record is created for **v** then: $c(\mathbf{v}) \leftarrow -1$
- while vertex **v** is being expanded in step (4): increment $c(\mathbf{v})$ every time an adjacent vertex is found.
- in procedure **vertex**: after any successful query of Ψ for the existence of a record for V-vertex **v**:
  - $c(\mathbf{v}) \leftarrow c(\mathbf{v}) - 1$
  - if $c(\mathbf{v}) = 0$ then delete the record for **v** from Ψ

So, Φ will contain only those "known" V-vertices **v** whose neighbors–*i.e.* the V-vertices adjacent to **v**—have *not* yet been expanded. The use of counters requires only one extra storage unit per V-vertex. There will be at most one deletion from Ψ for each V-vertex (an $O(d^2 \log n)$ operation), and updating the counters requires $O(d)$ time per vertex. Hence, the use of counters will not increase the time or space complexity of the algorithm. By selecting a good order for expanding the V-vertices in Φ, it may be possible to significantly reduce the size of the search tree: the last-in-first-out order that a stack implementation uses seems a particularly bad choice from this perspective. It remains an open question as to whether there exists an ordering such that the number of "known" V-vertices whose neighbors have not yet been expanded is asymptotically less than $O(n^{O(d)})$. If such an ordering exists, then the complexity of any operation on the search tree will have complexity less than $O(d^2 \log n)$.

4.2.3 Determining The Polytopes on Which Each V-vertex Lies

A V-vertex **v** belongs to any $V_k^S(T)$, for which $G \cup I \supset T \supset I$ (lemma 3.21), where G is the generating set of **v** and I is the subset of S enclosed by the hypersphere corresponding to **v**; $|I| = k - t$, $1 \leq t \leq d$. After the algorithm has terminated, we can determine all such polytopes for any V-vertex $\mathbf{v}$ by computing $d(\mathbf{v}, \mathbf{p})$ for all $\mathbf{p} \in S$. Assuming that we have the generating set G, as output of the algorithm: $\mathbf{p} \in I$ whenever $d(\mathbf{v}, \mathbf{p}) < d(\mathbf{v}, \mathbf{g})$ for any $\mathbf{g} \in G$. Hence we require:

- $O(d^3)$ time to determine $\mathbf{v}$
- $O(nd)$ time to compute the distances

Alternatively, if we have $\mathbf{v}$ as the output of the algorithm, the $O(d^3)$ component is not required. We can determine $d(\mathbf{v}, \mathbf{g})$ in $O(n)$ time [BFP+72] as the k^{th} least distance.

This $O(nd + d^3)$ complexity per V-vertex could be reduced to $O(k)$ time per V-vertex, if the vertex graph is the output of the algorithm (see section 4.2.4). The set I_0 of points enclosed by the hypersphere corresponding to the initial vertex (found in step 1 of the algorithm) are known. The vertex graph is traversed using a depth-first search with back-tracking: at most 2 updates are required to the set I, of enclosed points, between two adjacent vertices (lemma 3.20). Hence, we can output the indices of the set I for each vertex in time proportional to the size of I, which is $O(k)$.

The algorithm could be modified, so that the set I of enclosed points is part of the output for any V-vertex. This requires adding an additional field in the records of stack Φ which contains the indices of the $k - t$ $(1 \leq t \leq d)$ enclosed points. Since k can be $O(n)$ this could create a substantial increase of $O(nv)$ in the space requirement. No matter how inefficiently we represent the indices of I, we can add or delete an index in $O(k)$ time. At most 2 updates to I occur when we locate an adjacent V-vertex (lemma 3.20). Hence, pushing a new record on the stack has time complexity in $O(n)$ (since $k < n$); this is done once for each V-vertex, so nothing is added to the overall complexity of the algorithm.

The modified algorithm would output the indices of I together with the indices of the generating set of each V-vertex $\mathbf{v}$. Lemma 3.21 then tells us exactly which Voronoi polytopes contain $\mathbf{v}$.

4.2.4 The Vertex Graph As Output

In order to obtain the vertex graph as output of the algorithm, we could associate $2d + 2$ pointers with every record in the search tree Ψ. These pointers are analogous to the flags of section 4.2.1: $f_1^+, \ldots, f_{d+1}^+$ and $f_1^-, \ldots, f_{d+1}^-$. They point to other records of Ψ, and are updated in much the same way as were the flags of section 4.2.1; however, instead of merely indicating whether or not a vertex is already known to exist along a given edge, we actually store a pointer to that vertex. In addition, we store pointers to the adjacent V-vertices found whenever a V-vertex is expanded.

Whenever a V-vertex is found to be a terminal vertex (*i.e.* when $|I| \in \{k-1, k-d\}$), then one of the "banks" of pointers $f_1^+, \ldots, f_{d+1}^+$ or $f_1^-, \ldots, f_{d+1}^-$ can be deleted.

The algorithm terminates with $\leq 2d+2$ non-null pointers from each node, representing the V-edges of the order-k Voronoi Diagram. A pointer will be null, when a given V-edge extends to infinity.

Any time a pointer (V-edge) is added to a record, it has immediately been preceded by a query with $O(d^2 \log n)$ time complexity. Hence, the cost of adding a V-edge is negligible. There is no additional space requirement for the vertex graph either: Each record of Ψ has size $O(d)$ for the indices of the generating set, so the addition of $O(d)$ pointers does not change the size complexity.

Chapter 5

Facet Enumeration Algorithm

In this chapter an algorithm is presented which enumerates all facets of the order-k Voronoi Diagram of a set S of n points in $\Re^d$. This is motivated by the problem of reference set thinning in pattern recognition—discussed in chapter 1.

Each Voronoi polytope $V_k^S(T)$ is considered in turn: the facets of $V_k^S(T)$ are determined by the nonredundant constraints of equation (3.1), reprinted below:

$$V_k^S(T) = \bigcap_{\substack{\mathbf{p} \in T \\ \mathbf{q} \in S-T}} \overline{H(\mathbf{p}, \mathbf{q})} \tag{5.1}$$

Since each facet belongs to exactly two Voronoi polytopes (lemma 3.14), it follows that each facet will be found exactly twice by the algorithm. It will be helpful to define the "facet graph" of a Voronoi diagram.

Definition 5.1 *The Facet Graph $G = (V, E)$ of V_k^S has:*

- *$P \in V$ for each Voronoi polytope $P \in V_k^S$*
- *$(P_i, P_j) \in E$ for all $P_i, P_j \in V$ such that P_i and P_j are adjacent along a V-facet in V_k^S*

The algorithm will start from some arbitrary node of the facet graph—which may be discovered by determining the k nearest neighbors of any point in the space. Each node will be visited, enumerating all edges (*i.e.* the V-facets) arising from that node, and thereby discovering additional nodes. Since the graph is connected, this approach will find all edges in the graph—which correspond to V-facets in the Voronoi Diagram.

The facets of a Voronoi polyhedron $V_k^S(T)$ are determined by the bounding hyperplanes of the nonredundant constraints of equation (5.1). Therefore, determining the facets of $V_k^S(T)$ can be achieved by testing each constraint—determined by one point from T and one from $(S - T)$—for redundancy. This "brute force" approach must consider $k(n - k)$ constraints; however, in general, many of the points of S will not contribute to any facet in $V_k^S(T)$. This suggests a "two stage" approach of first determining the relevant (definition 3.5) points S', and then determining the nonredundancy among those constraints generated by the relevant points.

Stage 1 Determine a subset $S' \subseteq S$, $|S'| = n'$ such that:

1. $T' = S' \cap T$; $|T'| = k'$
2. $\forall \mathbf{p} \in T' : \exists \mathbf{q} \in (S' - T')$ such that $\overline{H(\mathbf{p}, \mathbf{q})}$ is nonredundant
3. $\forall \mathbf{q} \in (S' - T') : \exists \mathbf{p} \in T'$ such that $\overline{H(\mathbf{p}, \mathbf{q})}$ is nonredundant

Stage 2 Determine the nonredundancies among the $k'(n' - k')$ constraints of:

$$\bigcap_{\substack{\mathbf{p} \in T' \\ \mathbf{q} \in S' - T'}} \overline{H(\mathbf{p}, \mathbf{q})}$$

It will be shown in section 5.8 that the determination of the relevant points in $\Re^d$ can be transformed into a problem of determining the nonredundant constraints defining a polytope in $\Re^{d+1}$. Thus, the stage 1 problem is equivalent to a stage 2 problem in a higher dimensional space. This problem of determining the nonredundant constraints is equivalent to an extreme point problem (see section 2.6).

A Voronoi polytope is equivalent to the "feasible polytope" of a linear programming problem. Techniques which are based on linear programming algorithms can be used to determine redundancy among the constraints defining the polytope. The f facets of a single Voronoi polytope may be found by making $O(n)$ linear programming calls, each with $< f$ constraints, plus an additional overhead having complexity $O(nd + kd \log n)$ per facet. Using Megiddo's (modified) linear-time linear programming technique [Meg84] [Dye86] [Cla86], each of the linear programming calls has complexity $O(3^{d^2} f)$: hence, in fixed dimension an output-sensitive running time of $O(fn + fk \log n)$ is obtained. Since each facet is contained by exactly two Voronoi polytopes in the order-k Voronoi Diagram, this complexity bound holds for the enumeration of all facet of the order-k Voronoi Diagram. The output-sensitive facet enumeration algorithm is presented in section 5.10.

The high dimension-dependent constant of the above approach makes it impractical. Better performance is expected from the simplex method [Dan51] [Dan63] of linear programming, which is well known empirically to have linear expected running time in d dimensions, despite its exponential worst case performance. This approach has the added benefit of allowing the simultaneous consideration of all constraints for redundancy, instead of solving an independent linear program for each one.

Section 5.1 will review the fundamentals of linear programming. This will be followed by a description of the simplex method. Section 5.4 reviews methods, which are based on simplex method "pivots", for the determination of the nonredundant constraints defining a polytope. In section 5.6, a new interpretation of the simplex pivot is presented; and section 5.7 presents an method for the determination of nonredundant constraints, based on this interpretation. This method is used in the development of the "practical" algorithm for facet enumeration (section 5.9).

5.1 Linear Programming

Linear programming is a widely studied and widely applicable mechanism for solving optimization problems. It is commonly used in economics as well in computing science. The attraction of linear programming lies in the ease and intuitive appeal of problem formulation. The linear programming problem (LPP) is to optimize—*i.e.* to maximize the value of—a linear function of some d variables, subject to a set of m constraints on the values that the variables may assume. The constraints are themselves linear functions of the d variables. In algebraic form, an LPP may be expressed in the following "standard form":

$$\begin{array}{ll} \text{maximize} & \mathbf{c}\cdot\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \end{array} \tag{5.2}$$

where A is an $m \times d$ matrix, $\mathbf{c}$ a d dimensional vector, and $\mathbf{b}$ an m-dimensional vector. Let the i^{th} row of A be denoted $\mathbf{a_i}$ $(1 \leq i \leq m)$, and the β_i denote the i^{th} component of $\mathbf{b}$. The problem is to find a vector $\mathbf{x}$ which maximizes the value of the function $\mathbf{c}\cdot\mathbf{x}$, called the *objective function*, such that $\mathbf{a_i}\cdot\mathbf{x} \leq \beta_i$ for $1 \leq i \leq m$.

From a geometric point of view, the linear programming problem is very intuitive. Each constraint, $\mathbf{a_i} \cdot \mathbf{x} \leq \beta_i$ defines a halfspace with bounding hyperplane $\mathbf{a_i} \cdot \mathbf{x} = \beta_i$. The intersection of all m halfspaces defines the feasible region of the problem—a polytope by definition 2.2. That is, any $\mathbf{x}$ lying in this polytope is a feasible solution to the LPP, in the sense that it satisfies all of the constraints. The LPP is thus to determine a feasible point $\mathbf{x}$ at which $\mathbf{c} \cdot \mathbf{x}$ attains its maximum value, or to conclude that no feasible point exists. In the latter case, the intersection of the constraint halfspaces is empty and the LPP is said to be *infeasible*. The feasible polytope may be unbounded: in this case, it is possible (but not necessary) that the value of the objective function may be increased without bound. The LPP is said to be *unbounded* if the objective function has no finite maximum. Note that an unbounded LPP implies an unbounded feasible polytope, but the converse is not true.

An alternative geometric interpretation of a feasible solution to an LPP is as a hyperplane which separates two point sets $S_1, S_2 \subset \Re^d$. Each row vector of A corresponds to one of these points: for all $\mathbf{p} \in S_1$, $\mathbf{p}$ is the i^{th} row vector of A (for some $1 \leq i \leq m$) and $\beta_i \leftarrow 1$; for all $\mathbf{p} \in S_2$, $-\mathbf{p}$ is the i^{th} row vector of A (for some $1 \leq i \leq m$) and $\beta_i \leftarrow -1$. Then, for any feasible solution $\mathbf{f}$ to the LPP, the hyperplane with equation $\mathbf{f} \cdot \mathbf{x} = 1$ separates S_1 and S_2.

For a feasible solution $\mathbf{f}$ to linear program (5.2), let $\omega = \mathbf{c} \cdot \mathbf{f}$ be the value of the objective function at $\mathbf{f}$. Since $\mathbf{c} \cdot \mathbf{x} = \omega$ is the equation of a hyperplane, the objective function for any point $\mathbf{x}$ on this hyperplane has the same value ω. Thus, optimizing the objective function may be visualized as moving the hyperplane $\mathbf{c} \cdot \mathbf{x} = \omega$ in the direction of the normal $\mathbf{c}$, until it no longer intersects the feasible polytope. It is easy to show that the optimal value of $\mathbf{c} \cdot \mathbf{x}$ will be attained by some $\mathbf{x}$ which is a vertex of the polytope.

A *co-basis* for an LPP is any d-subset of the constraints such that the corresponding d bounding hyperplanes intersect in a vertex of the feasible polytope. Such a vertex—also called a *basic feasible solution*—may be generated by more than one distinct co-basis if degeneracies are present.

For many LPPs, especially those in economics, negative valued variables are not meaningful. Often the standard LPP formulation is given as:

$$\begin{array}{ll} \text{maximize} & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & A'\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array} \tag{5.3}$$

Enforcing positive valued variables amounts to the introduction of d additional constraints.

System (5.2)—with A an $m \times d$ matrix—may be rewritten as system (5.3)—with A' an $(m-d) \times d$ matrix—by a straightforward affine transformation: that is, d linearly independent rows of A become the coordinate system in system (5.3).

Every LPP has an associated dual problem. Given the matrix A and vectors $\mathbf{b}$ and $\mathbf{c}$ of system (5.2), the following is the dual problem:

$$\begin{array}{ll} \text{minimize} & \mathbf{b} \cdot \mathbf{y} \\ \text{subject to} & A^T\mathbf{y} \leq \mathbf{c} \end{array} \tag{5.4}$$

where A^T denotes the transpose of A. The "duality theorem of linear programming" states that system (5.2) has an finite optimal solution $\mathbf{x}$ if and only if system (5.4) has a finite optimal solution $\mathbf{y}$, and that the objective functions of the two systems have the same value at the optima: *i.e.* $\min\{\mathbf{b} \cdot \mathbf{y}\} = \max\{\mathbf{c} \cdot \mathbf{x}\}$. Furthermore if either problem is unbounded, then the other has no feasible solution. Hence, an LPP algorithm may assume that $m \leq d$ (or, conversely, that $m \geq d$) simply by solving the dual problem whenever this condition is not satisfied.

The most common algorithm for solving LPPs is the simplex method, developed by Dantzig in 1947 for the solution of US Air Force planning problems [Dan51] [Dan63]. The simplex method starts at a basic feasible solution of system (5.3). and successively pivots to an adjacent basis at which the value of the objective function is at least as large. To find the initial vertex—or determine that no feasible point exists—is equivalent to solving a separate linear program in $(d+1)$-dimensions, for which there is a trivial initial feasible solution. This determination of an initial feasible vertex comprises "Phase I" of the simplex method. Optimization of the objective function starting from that vertex comprises "Phase II" of the simplex method. In practice, phase II is solved in an $(m+d)$-dimensional space (and phase I in an $(m+d+1)$-dimensional space) after introducing m "slack" variables—one for each constraint—as will be described in the next section.

The simplex method has been used extensively to solve LPPs and empirical evidence has shown that, in almost all examples, it converges to an optimal solution in about m or $3m/2$ pivots [Chv83]. However, theoretical result have shown that, in the worst case, the number of pivots can be exponential in m, and examples have actually been constructed for which the simplex method does perform an exponential number of pivots [KM72]. Before 1979, all attempts to develop linear programming algorithms which were sub-exponential in the worst case, were fruitless; it was often conjectured that linear programming was a member of the complexity class NPI. [1] [GJ79]

A breakthrough result came in 1979, when Khachiyan [Kha79] showed that the ellipsoid algorithm—which had been developed in connection with convex programming [IN77]—could be used to solve LPPs (where $d \geq m$) in time bounded by $O(d^6 l)$, where l is the number of digits in the coefficients in the input [AS80]. The ellipsoid algorithm is initialized with an ellipsoid E which contains a feasible point of system (5.2), if one exists. On each iteration, the center $\mathbf{e}$ of E is tested for inclusion in the feasible polytope. If the test is successful, the algorithm halts having found a feasible point. Otherwise, some constraint, say $\mathbf{a} \cdot \mathbf{x} \leq \beta$, is violated: E is replaced by a smaller ellipsoid which contains $\{\mathbf{x} \in E | \mathbf{a} \cdot \mathbf{x} \leq \mathbf{a} \cdot \mathbf{e}\}$, and the iteration is repeated. So, the ellipsoid method generates a *feasible* point in the polytope $A\mathbf{x} \leq \mathbf{b}$ rather than directly solving the LPP. However, there are several ways to rephrase the standard LPP formulation as a feasible point query [BGT81]; for example, a formulation which looks for a feasible point of the primal and dual problems simultaneously.

Karmarkar's algorithm [Kar84] is similar to the ellipsoid method, but uses a complicated sequence of transformations to result in a better complexity bound of $O(d^{3.5} l)$. This bound was later improved to $O(d^3 l)$ [Gon89].

Although Khachiyan's result placed linear programming within the complexity class P, it has been argued [Meg84] [Dye86] [Cla86] that the (low level) computational model under which the ellipsoid method achieves its polynomial upper bound is not satisfactory. Under the more commonly-used real-arithmetic RAM model of computation [AHU74], the complexity of the ellipsoid method cannot be bounded by a polynomial in the dimension d and number of constraints m and is, therefore, not "genuinely" polynomial—even if the dimension is fixed. On the other hand, since the number of bases of the feasible polytope has an upper bound of $m^{\lfloor \frac{d}{2} \rfloor}$ [McM70], and since there exist pivoting strategies which guarantee that the simplex method does not visit the same basis more than once [Bla77], it follows that the simplex method is "genuinely" polynomial when the dimension is fixed.

In fact, Megiddo [Meg84] has shown that the fixed-dimensional LPP may be solved in linear time using a "recursive multidimensional search technique", described as follows. Let H denote a hyperplane with normal $\mathbf{c}$—where $\mathbf{c} \cdot \mathbf{x}$ is the objective function—and let $\mathbf{v}$ denote the projection of the optimal vertex onto H. The constraints of the LPP are paired, and the intersection of each pair is projected onto H. This projection defines a dividing $(d-2)$-flat, H', of H. Now, by localizing $\mathbf{v}$ to one side or the other of H', we may deterministically

[1] Under the assumption that P $\neq$ NP: NPI(NP-Incomplete) = NP - (P $\cup$ NP-Complete)

remove one of the constraints of the pair from further consideration, since it will not be tight at the optimal vertex. Localizing **v** on the hyperplane H is an LPP in $d-1$ dimensions, so the search can be done recursively in decreasing dimension. It turns out to be necessary to explicitly test only a constant number—*i.e.* independent of m, but dependent on d—of the $\frac{m}{2}$ intersections of pairs of constraints, in order to localize **v** with respect to all intersections. The time complexity of this approach was first reported as $O(2^{2^d}m)$ [Meg84]. Modifications by [Cla86] and [Dye86] improved the constant, resulting in an $O(3^{d^2}m)$ complexity bound.

From a practical point of view, neither Khachiyan's ellipsoid method nor Megiddo's recursive multidimensional search technique seriously rival the simplex method in expected-time performance. Despite the fact that these newer methods have better theoretical complexity bounds, the constants are prohibitively high. Karmarkar [Kar84] has claimed that his algorithm does have practical significance, but this claim is somewhat contentious.

5.2 The Simplex Method

Consider the following LP in the form of system 5.3, where A_N is an $m \times d$ matrix, **b** is an m-vector and $\mathbf{c_N}$ is a d-vector. The subscript N denotes the set $\{1, 2, \ldots, d\}$ and indicates that the columns of A_N, elements of $\mathbf{c_N}$ and $\mathbf{x_N}$ are indexed from 1 through d.

$$\begin{array}{ll} \text{maximize} & \mathbf{c_N} \cdot \mathbf{x_N} \\ \text{subject to} & A_N\mathbf{x_N} \leq \mathbf{b} \\ & \mathbf{x_N} \geq 0 \end{array}$$

For each i^{th} row $\mathbf{a_i}$ of A_N, we may introduce a "slack variable" x_{d+i} and rewrite the constraint as an equality:

$$\mathbf{a_i}\mathbf{x_N} + x_{d+i} = \beta_i$$

Requiring that all slack variable be positive enforces the inequality constraints of system (5.3). Let A denote the matrix $[A_B|A_N]$, where $A_B = I$ is the $m \times m$ identity matrix; let $\mathbf{c} = [\mathbf{c_N}|0, 0 \ldots, 0]$ and $\mathbf{x}$ be $(d+m)$-vectors. Then, system (5.3) can be rewritten as:

$$\begin{array}{ll} \text{maximize} & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array} \tag{5.5}$$

The vector $\mathbf{x}$ may be partitioned as $[\mathbf{x_B}|\mathbf{x_N}]$, where $B = \{d+1, d+2, \ldots, d+m\}$ denotes the subscripts of the slack variables. The m columns of A_B comprise a basis for $\Re^m$ and, consequently, the corresponding variables $\mathbf{x_B}$ are called *basic variables*; the original variables (*i.e.* $\mathbf{x_N}$) are called *nonbasic variables.* The constraints now have the form:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,d} & 1 & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,d} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,d} & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{d+m} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} \tag{5.6}$$

Assuming that $\mathbf{b} \geq \mathbf{0}$, there is an obvious initial feasible solution of:

$$\begin{aligned} x_i &= 0 & \forall i \in N \\ x_i &= \beta_{i-d} & \forall i \in B \end{aligned}$$

This is an example of a *basic feasible solution*—*i.e.* one for which all nonbasic variables are equal to 0. If it is not the case that $\mathbf{b} \geq \mathbf{0}$, then we may find an basic feasible solution by "phase 1" of the simplex method, as described later.

At each iteration—called a "pivot"—of the simplex method, one nonbasic variable x_p is chosen to enter the basis and its value is increased while all other nonbasic variables are held constant: this causes the basic variables to change in value. Unless the problem is unbounded, as x_p is increased, some basic variable x_r will decrease to 0 becoming a nonbasic variable. If degeneracies are present—*i.e.* if the bounding hyperplanes of more than d constraints intersect in a common vertex—then there may be more than one candidate x_r; otherwise, x_r will be uniquely determined. A pivot results in the following updates to the set B of basic indices and the set N of nonbasic indices:

$$\begin{aligned} B &\leftarrow B \cup \{p\} - \{r\} \\ N &\leftarrow N - \{p\} \cup \{r\} \end{aligned}$$

We wish to select p such that c_p is positive, thus increasing the value of the objective function $\mathbf{c} \cdot \mathbf{x}$. If no c_p ($p \in N$) is positive, then $\mathbf{x}$ is optimal and the algorithm terminates. Consider what happens to the i^{th} constraint of system (5.6) as x_p is increased with all other nonbasic variables held at 0:

$$\begin{aligned} \textstyle\sum_{j=1}^{d}(a_{ij}x_j) + x_{d+i} &= \beta_i \\ i.e. \qquad x_{d+i} &= \beta_i - a_{i,p}x_p \end{aligned}$$

Since x_{d+i} must remain positive, it follows that x_p cannot be increased by more than $\frac{\beta_i}{a_{i,p}}$, whenever $a_{i,p}$ is positive (note that β_i is always positive in a basic feasible solution). Therefore, the variable x_r (where $r = d + i$) to leave the basis is the one for which this ratio is the least positive. The pivot is performed making the substitution:

$$x_p = \frac{\beta_r - x_r - \sum_{\substack{j=1\\ j\neq p}}^{d} a_{r,j}x_j}{a_{r,p}}$$

in system (5.6). The objective function is updated by making the same substitution, except ignoring the constant term $c_p \cdot \frac{\beta_r}{a_{r,p}}$, since it will not change the optimal solution (it will only change the value of the objective function at this solution). It is easy to verify that this is achieved by making the following "rank-1" updates to A and $\mathbf{b}$:

$$\begin{array}{lcll}
a_{r,j} & \leftarrow & \frac{a_{r,j}}{a_{r,p}} & \\
a_{i,j} & \leftarrow & a_{i,j} - \frac{a_{r,j}a_{i,p}}{a_{r,p}} & i \neq r \\
\beta_r & \leftarrow & \frac{\beta_r}{a_{r,p}} & \\
\beta_i & \leftarrow & \beta_i - \frac{\beta_i a_{i,p}}{a_{r,p}} & i \neq r \\
c_j & \leftarrow & c_j - \frac{a_{r,j}c_p}{a_{r,p}} &
\end{array} \qquad (5.7)$$

After each pivot, the objective function is expressed in terms of the (current) nonbasic variables, and the updated matrix A may be expressed as $[A_N|I]$. Hence, the updated system is similar to system (5.6) and the simplex method can continue iteratively.

There are $d+m$ constraints in system (5.3), including the explicit nonnegativity requirement on the original variables. Each constraint has an associated slack variable (or, in the case of the nonnegativity constraints, an original variable). After 0 or more pivots, each row corresponds to a basic variable, and each column corresponds to a nonbasic variable. It is not necessary to explicitly store the m columns of $A_B = I$: we need only flag each row and each column of A_N indicating to which variable it corresponds. Using this strategy, we may interpret the $m \times d$ matrix A as representing the inequality constraints of system (5.3). We can combine A, $\mathbf{c}$ and $\mathbf{b}$ into a *simplex tableau* as follows:

$$\begin{array}{c|c|c|}
 & j_1 \ \cdots \ j_d & \\ \hline
i_1 & & \\
\vdots & A & \mathbf{b} \\
i_m & & \\ \hline
 & \mathbf{c} & \\ \hline
\end{array} \qquad (5.8)$$

where $j_c \in N$ $(1 \leq c \leq d)$ and $i_r \in B$ $(1 \leq r \leq m)$ are the labels of the nonbasic and basic variables, respectively. Let $\nu_{i,j}$ denote the elements of the above tableau, with the

$(m+1)^{st}$ row of the tableau being the objective function, and the $(d+1)^{st}$ column being **b**. Updates (5.7) may be rewritten:

$$\begin{array}{lll} \bullet \text{ swap labels of row } r \text{ and column } p & & \\ \nu_{r,p} \leftarrow \frac{1}{\nu_{r,p}} & & \\ \nu_{r,j} \leftarrow \frac{\nu_{r,j}}{\nu_{r,p}} & j \neq p & \\ \nu_{i,p} \leftarrow -\frac{\nu_{i,p}}{\nu_{r,p}} & i \neq r & \\ \nu_{i,j} \leftarrow \nu_{i,j} - \frac{\nu_{r,j} \times \nu_{i,p}}{\nu_{r,p}} & i \neq r,\ j \neq p & \end{array} \tag{5.9}$$

From a geometric perspective, it is most intuitive to think of the feasible polytope as a d-dimensional structure, and each slack variable as the distance from the corresponding constraint hyperplane to the current basic solution. Assuming that all of the original d variables have been pivoted into the basis, there will be at least d slack variables whose value is 0 at the current vertex: hence, the corresponding d hyperplanes intersect at that vertex. The updating done to A and **b** at each iteration may be viewed as an affine transformation: the current vertex becomes the origin, and the d intersecting hyperplanes define the coordinate system.

Since there is a finite number of bases for vertices in the feasible polytope, the simplex method will converge to the optimal solution, unless cycling occurs. Fortunately, there are simple rules for selecting the pivot variables which guarantee that cycles will not occur. For example, **Bland's rule** [Bla77] can be simply stated as:

- From all possible candidate variables x_i to enter the basis, choose the one with the least index i.

$$p = \min\{j \in N | c_j > 0\}$$

- From all possible candidate variables x_i to leave the basis, choose the one with the least index i.

$$q = \min\left\{ i \in B \;\middle|\; \begin{array}{l} \frac{\beta_i}{a_{i,p}} > 0 \\ \frac{\beta_i}{a_{i,p}} \leq \frac{\beta_j}{a_{j,p}} \quad \forall \frac{\beta_j}{a_{j,p}} > 0 \end{array} \right\}$$

If it was not the case that $\mathbf{b} \geq \mathbf{0}$ in system (5.6), then phase 1 of the simplex method must be used to find an initial feasible solution. An additional basic variable x_{d+m+1} is introduced into the system as follows:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,d} & 1 & 0 & \cdots & 0 & -1 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,d} & 0 & 1 & \cdots & 0 & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,d} & 0 & 0 & \cdots & 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{d+m+1} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix}$$

The objective function is replaced by:

$$\text{maximize } (-x_{d+m+1})$$

System (5.6) has a feasible solution if and only if this maximum is 0. An initial basic feasible solution is easily obtained for this system by setting x_{d+m+1} high enough. Then the usual simplex pivots are performed. If the required maximum is attained, the final tableau of stage 1 is converted to the initial tableau of stage 2 by simply deleting the $(d+m+1)^{st}$ column of the matrix A. Then stage 2 may proceed as previously described.

5.3 The Revised Simplex Method

The revised simplex method [DOH54] was developed by Dantzig and co-workers shortly after the "standard" simplex method was discovered. The theory of both the standard and revised methods can be found in any linear programming textbook, such as [Chv83] [Lue84].

The standard simplex method must update an $m \times d$ tableau for each pivot performed. The updated tableau facilitates selection of pivot row and pivot column at the next iteration: the pivot column p is selected by a sign test on the elements of $\mathbf{c}$, and the pivot row is selected by computing the ratios $\frac{\beta_i}{a_{i,p}}$ for every row i. Hence, the overall cost of selecting the pivot elements is $O(d+m)$ The disadvantages of this approach include:

- It is often the case that d is very large and $m << d$; so the $O(md)$ cost of updating can be large.

- After many pivots, numerical errors in the coefficients of the simplex tableau become compounded. To recompute the tableau requires an affine transformation, which takes $O(d^3 + md)$ time, and assumes that we have the original coefficients stored (an additional $O(md)$ storage).

The revised simplex method responds to both of these concerns. The coefficients of A, $\mathbf{b}$ and $\mathbf{c}$ are never updated. Instead, at each pivot, the $m \times m$ matrix A_B^{-1} is updated. A_B^{-1} is the inverse of the current basis. The current solution is then given by:

$$\mathbf{x} = A_B^{-1}\mathbf{b}$$

The rank-1 update operations for A_B^{-1} are analogous to those of the standard simplex method, except that they are done on an $m \times m$ matrix instead of an $m \times d$ matrix, so the cost is $O(m^2)$. We can always assume that $m \leq d$ since, whenever this is not the case, we can solve the dual problem (see equation (5.4)). This can result in significant computational savings when $m << d$. It has the additional benefit of avoiding numerical errors in the coefficients after many pivots have taken place: although numerical errors may corrupt the matrix A_B^{-1}, it is usually the practice to recompute the inverse (an $O(m^3)$ operation) after some number of pivots in order to increase stability. Suppose this is done after every m pivots: then the amortized cost of $O(m^2)$ per pivot adds nothing to the overall complexity.

These benefits, of course, do not come without a price: the selection of pivot elements becomes a $O(m^2+md)$ procedure. However, on large and sparse problems, the time requirement is much less. The modified simplex method is not well suited for use in determination of redundancy among a system of m constraints in $\Re^d$: this is because, in general, $m >> d$, and the dual problem cannot be used to determine redundancy, except by considering each primal constraint separately. In section 5.6, we present a variation of the revised simplex method, which is better suited for this purpose.

5.4 Redundancy In Linear Programming

The problem of determining the redundant constraints among a system of m inequalities defining a polytope $P \subset \Re^d$ arises naturally as a preprocessing step in linear programming.

$$P = \{\mathbf{x} \in \Re^d | A\mathbf{x} \leq \mathbf{b}\} \tag{5.10}$$

where A is an $m \times d$ matrix, with row vectors denoted $\mathbf{a_i}$ $(1 \leq i \leq m)$, and $\mathbf{b} = (\beta_1, , \ldots , \beta_m)$ is an m-vector. The determination of redundancy is equivalent to the problem of identifying the extreme points in a point set (as is discussed in section 2.6). Other applications of this problem are discussed in [Zio80].

A constraint $\mathbf{a_i} \cdot \mathbf{x} \leq \beta_i$ is called *redundant* if its removal from formulation (5.10) does not change P. A redundant constraint is termed *strongly redundant* if it is never satisfied as

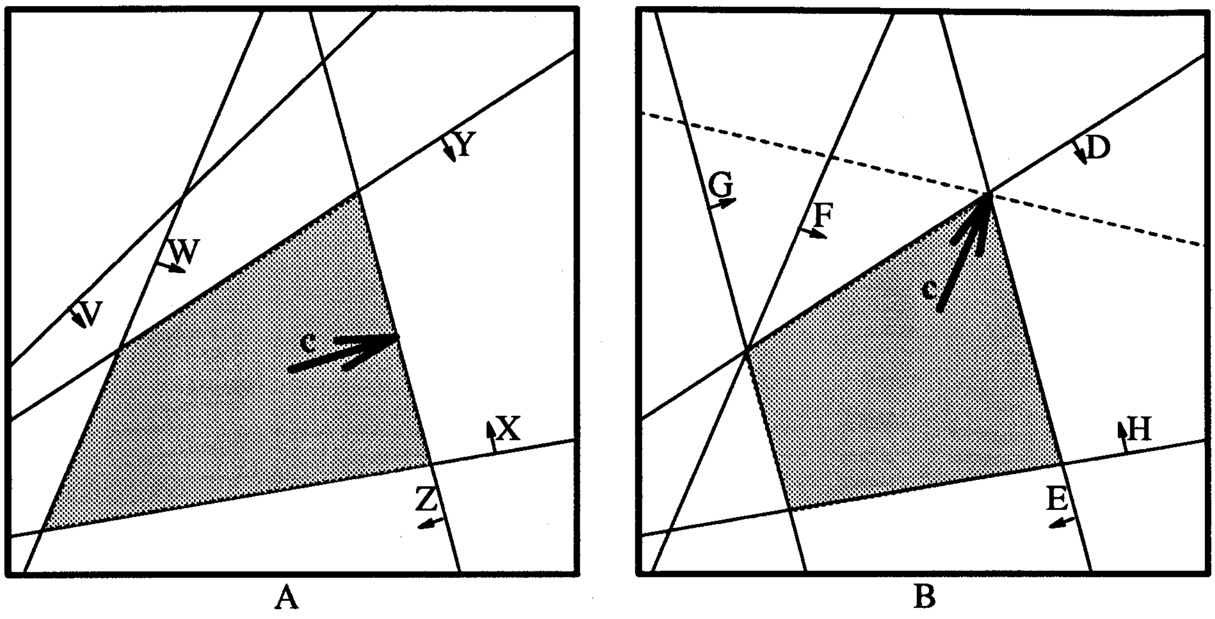

Figure 5.1: Illustration of Different Categories of Constraints

These diagrams each show a feasible polytope (shaded region) in $\Re^2$ as the intersection of halfplanes—each represented by a bounding line together with an arrow indicating the side on which the halfplane lies. The objective functions are represented by the heavy arrows, labeled **c**.

Box A The objective is parallel to the surface normal of constraint Z. Hence, any feasible point on the bounding line of Z will be optimal.

Box B The objective is maximal along the dashed line, which intersects the feasible region in a single vertex.

The constraints are classified as:

- V is strongly redundant and nonbinding
- F is weakly redundant (or weakly nonredundant) and nonbinding
- G, H and W are strongly nonredundant and nonbinding
- X and Y are strongly nonredundant and weakly binding
- D, E and Z are strongly nonredundant and strongly binding

an equality $\mathbf{a_i} \cdot \mathbf{x} = \beta_i$ for any feasible point $\mathbf{x}$, and *weakly redundant* if it is satisfied as an equality for some feasible $\mathbf{x}$. A constraint which is satisfied as an equality at some optimal solution is called *binding*. Such a constraint is termed *strongly binding* if it is satisfied as an equality at all optimal solutions, and *weakly binding* if it is satisfied as an equality at some, but not all, optimal solutions. These definitions are illustrated in figure 5.1 for 2-dimensional examples.

The earliest work on determining redundancy in LPP formulations was by Boot [Boo62]. For each i^{th} row of system (5.10), a feasible point exists in the following system:

$$\begin{aligned} &\mathbf{a_i} \cdot \mathbf{x} > \beta_i; \\ &\mathbf{a_j} \cdot \mathbf{x} \leq \beta_j; \quad j \neq i \end{aligned}$$

if and only if $\mathbf{a_i} \cdot \mathbf{x} \leq \beta_i$ is a strongly nonredundant constraint. This approach requires the solution of m linear programs, each with m constraints in $\Re^d$.

We present here an simple extension to this approach, for which each of the m linear programs has at most f constraints, where f is the number of nonredundant constraints defin P in system (5.10). The algorithm (given in figure 5.2) uses a weaker notion of nonredundancy, which classifies the weakly redundant constraints as nonredundant. These (either strongly or weakly) nonredundant constraints are responsible for determining the V-facets of a Voronoi diagram.

The algorithm assumes that we have an initial feasible point $\mathbf{y}$. If no such initial point is available, we can easily determine one by a single linear programming call of size m. The set F is initialized to $\emptyset$, and the algorithm successively places in F the index of each nonredundant constraint. The set W (initialized to $\{1, \ldots, m\}$) contains the indices of the currently "unknown" constraints. With every iteration of step (2), one constraint is removed from the set W.

In step (2c), we have located a point $\mathbf{z}$ which lies in the intersection of the f constraints which are (so far) known to be nonredundant, and yet does satisfy some $(f+1)^{st}$ constraint. Therefore, the ray directed from $\mathbf{y}$ to $\mathbf{z}$ must intersect the boundary of P at boundary of some new constraint. For each constraint $\mathbf{a_i} \cdot \mathbf{x} \leq \beta_i$, $i \in W$, we are interested in the point $\mathbf{x_i}$ for which;

$$\mathbf{a_i} \cdot \mathbf{x_i} = \beta_i; \qquad \text{and} \qquad \mathbf{x_i} = \mathbf{y} + \tau_i(\mathbf{z} - \mathbf{y})$$

The least nonnegative τ_i corresponds the the intersection, along the ray, which is nearest to $\mathbf{y}$. Step (2c[i]) of the algorithm determines the value of each τ_i. Then the minimal

```
Algorithm Non_Redundant($A, \mathbf{b}, \mathbf{y}, F$)

Input:   $A$                -an $m \times d$ matrix, with row vectors denoted $\mathbf{a_i}$, $1 \le i \le m$
         $\mathbf{b}$                -an $m$-vector: $\mathbf{b} = (\beta_i, \dots, \beta_m)$
         $\mathbf{y} \in \Re^d$
         such that:  $\left\{ \begin{array}{l} \mathbf{a_i} \cdot \mathbf{x} \le \beta_i \text{ defines a constraint of } P, \text{ for } 1 \le i \le m \\ \mathbf{y} \in P \end{array} \right.$

Output:  $F$                -set of indices $(1, \dots, m)$ of the nonredundant constraints

1. set $F \leftarrow \emptyset$; $W \leftarrow \{1, 2, \dots, m\}$
2. while $W \neq \emptyset$ do:
   (a) find a feasible point of:
           $\mathbf{a_i} \cdot \mathbf{x} \le \beta_i \quad \forall i \in F$
           $\mathbf{a_j} \cdot \mathbf{x} \ge \beta_j \quad \text{for some } j \in W$
   (b) if the LPP is infeasible then
       • $W \leftarrow W - \{j\}$
   (c) otherwise let $\mathbf{z}$ denote some feasible point.
       i. for all $i \in W$ do:
          • if $\mathbf{a_i} \cdot (\mathbf{z} - \mathbf{y}) = 0$ then:
            – if $\beta_i = \mathbf{a_i} \cdot \mathbf{y}$ then: $\tau_i \leftarrow 0$
            – otherwise: $\tau_i \leftarrow \infty$
          • otherwise: $\tau_i \leftarrow \frac{\beta_i - \mathbf{a_i} \cdot \mathbf{y}}{\mathbf{a_i} \cdot (\mathbf{z} - \mathbf{y})}$
       ii. set $r$ such that $\tau_r$ attains the minimum nonnegative value for $r \in W$
       iii. $F \leftarrow F \cup \{r\}$
       iv. $W \leftarrow W - \{r\}$
```

Figure 5.2: $O(fm)$ Algorithm to Find All f Nonredundant Constraints of the m Inequalities Defining a Polytope P

nonnegative one, τ_r, is chosen: hence, the r^{th} constraint is nonredundant.

Step (1) of the algorithm is trivial, and step (2) of the algorithm is executed $< m$ times. Overall, sub-step (2a) requires the solution of m LPPs, each with at most f constraints. Sub-step (2b), trivially, has complexity $O(m)$ for all m iterations. Sub-step (2c) is only executed when a new nonredundant facet is discovered (*i.e.* $< f$ times); it requires $< m$ computations of the intersection of a hyperplane with a line, which is an $O(d)$ operation; so, step (2c) has complexity $O(mdf)$.

Therefore, algorithm **Non_Redundant** requires the solution of m linear programs with $O(f)$ constraints, and an additional overhead of $O(mdf)$. In fixed dimension, the overall complexity is $O(mf)$ [Meg84].

Proof of correctness follows from the observations that no constraint is called redundant in P unless it is found to be redundant among a subset of the constraints of P; and, for any nonredundant constraint H, we have found a ray from $\mathbf{y} \in P$ to the boundary of H which lies completely within P.

Although the $O(mf)$ complexity is appealing from a theoretical standpoint, the high constant of 3^{d^2} [Dye86] [Cla86] makes it impractical. Indeed, better empirical performance would be expected by using the simplex method to solve the LPPs of step (3a), rather than using Megiddo's search technique. If the simplex method were used, a great deal of effort would be wasted, in general, by repeating similar pivots time and again while considering each constraint independently.

Several groups of researchers have developed algorithms which exploit features of the simplex method to simultaneously consider all constraints for redundancy [ZW83] [Gal83] [Tel83] [Rub83]. Essentially, these algorithms all use the same strategy: Each i^{th} row vector of A is viewed as the objective function:

$$\text{maximize} \quad \mathbf{a_i} \cdot \mathbf{x}$$

The constraint is redundant if and only if the optimal value is less than β_i. The algorithms are initialized with a known feasible solution (*i.e.* vertex), and the d constraints of this initial co-basis are flagged as nonredundant; the others are flagged as unknown. Standard simplex pivots are performed and, after each pivot, any unknown constraint for which $\mathbf{a_i} \cdot \mathbf{x} = \beta_i$ is flagged as nonredundant (note that this relationship holds for any constraint brought into the co-basis). Also, by simply examining the signs of the coefficients of each unknown constraint for which $\mathbf{a_i} \cdot \mathbf{x} < \beta_i$, we can determine if the i^{th} constraint attains its optimal value at the current vertex; if so, the constraint is flagged as "redundant". These methods are called the "sign-test methods", for this reason. This is identical to the manner in which the simplex method examines the signs of the objective function coefficients. The algorithm terminates when all constraints have been identified as either redundant or nonredundant.

As long as pivots are chosen which optimize the value of at least one of the unknown constraints, the algorithm will be finite; hence, any "unknown" constraint can serve as the objective function. An upper bound on the time complexity of determining redundancy in this manner will, of course, be exponential just as the simplex method is. In practice, however, much better performance is observed [KLTZ83], just as the observed performance of the simplex method itself is much better than the upper complexity bound.

A variation on the above approach was developed by Mattheiss [Mat73] [Mat83]. The same pivoting strategy is employed, but it is done on a polytope in $\Re^{d+1}$. The original polytope P, defined by the equation (5.10), is embedded in $\Re^{d+1}$ to form the polytope P':

$$P' = \{[\mathbf{x}|x_{d+1}] | A\mathbf{x} + \mathbf{t}x_{d+1} \leq \mathbf{b}\}$$

where the i^{th} component of $\mathbf{t}$ is $||\mathbf{a_i}||_2$, the 2-norm of the i^{th} row of A. P is a facet of the new polytope P' (*i.e.* P is the intersection of P' with the hyperplane having equation $x_{d+1} = 0$). Any facet of P' has a nonempty intersection with P [Mat73, theorem 1]. Hence, nonredundancy among the constraints defining P is equivalent to nonredundancy among the constraints defining P'. The claim is made that $P' - P$ has significantly fewer vertices than does P. Computational results are presented in [MS80] which support this claim, and show that the difference becomes much more pronounced as d is increased. This algorithm visits *all* vertices of $P' - P$, whereas the other sign-test algorithms, in general, do not visit all vertices of P.

5.5 Finding Nonredundant Constraints By Searching Vertices

The boundary of any nonredundant constraint defining a polytope must intersect some vertex of that polytope. Hence, we could determine the nonredundant constraints by visiting all of the vertices of a polytope. Since the intersection of m constraints in $\Re^d$ may have as many as $m^{\lfloor \frac{d}{2} \rfloor}$ vertices [McM70], it is tempting to speculate that we could visit only a

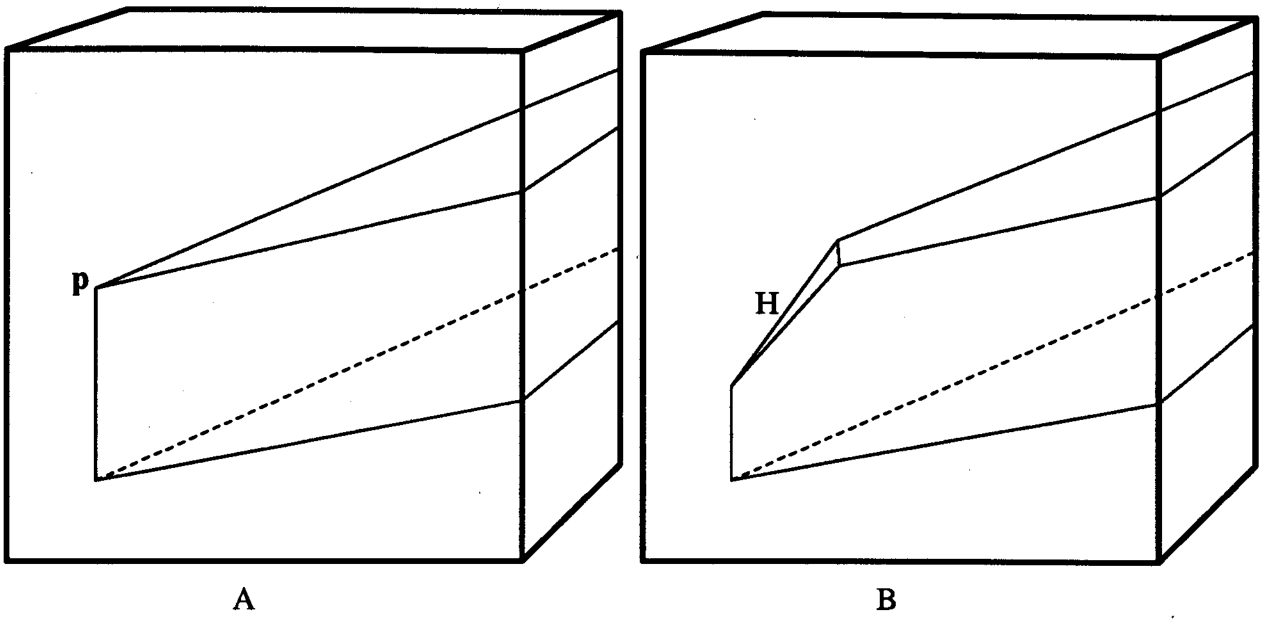

Figure 5.3: The Difficulty With Not Visiting All Vertices

(small) subset of the vertices—without performing explicit redundancy tests as in the sign-test methods of section 5.4—and thereby find all of the nonredundant constraints: those whose boundaries intersect in one of the vertices visited.

Recently, Avis and Fukuda [AF90] have developed an elegant algorithm which enumerates all v vertices of the intersection of m nondegenerate halfspaces in $\Re^d$, in $O(ndv)$ time and $O(nd)$ space. The algorithm performs simplex pivots to do a depth-first search of the vertices. By exploiting Bland's rule (see page 64), no additional storage is required for intermediate vertices.

Given some objective function, Bland's rule selects a unique simplex pivot from any nonoptimal vertex. Hence, if the optimal vertex is unique, then Bland's rule defines a tree on the vertices of the feasible polytope. If we have some feasible vertex $\mathbf{v}$ of the polytope—*i.e.* the intersection of the boundaries of d constraints—then we can easily select an objective function for which $\mathbf{v}$ is optimal. The algorithm of Avis and Fukuda reverses Bland's rule to traverse the tree defined on the feasible vertices.

Any attempt to use this strategy to determine the nonredundant constraints in an output-sensitive manner is doomed from the start. That is, we would like to find the f nonredundant constraints in time which not dependent upon v. However, if there is some vertex $\mathbf{p}$ which has not been visited (see figure 5.3A) and we have not performed any redundancy test on a constraint H, then the situation depicted in figure 5.3B is indistinguishable from that of figure 5.3A.

5.6 Modified Simplex Pivots

In this section, we present a pivoting strategy which resembles the revised simplex method. This strategy will be used for detection of redundancy among a system of m linear constraints in $\Re^d$. It differs from the revised simplex method in that it updates a $d \times d$ matrix which is the inverse of the current co-basis, whereas the revised simplex method updates an $m \times m$ matrix which is the inverse of the current basis. The new method is called the "modified simplex method", and it is superior to the revised method whenever $d < m$.

In matrix form, we have an $m \times d$ matrix A and m-vector $\mathbf{b}$ such that:

$$A\mathbf{x} \leq \mathbf{b} \tag{5.11}$$

Let P denote the polytope defined by the above system, and assume that we have a feasible vertex $\mathbf{v}$ of P. It follows that $\mathbf{v}$ lies on the bounding hyperplanes $\mathbf{a_i} \cdot \mathbf{x} = \beta_i$ of (at least) d of the constraints defining P. Without loss of generality, assume that $\mathbf{v}$ lies on the bounding hyperplanes of the first d constraints in system (5.11), and that their surface normals are linearly independent. Let $A_B\mathbf{x} \leq \mathbf{b_B}$ denote these first d constraints, and $A_N\mathbf{x} \leq \mathbf{b_N}$ denote the other $n - d$:

$$\underbrace{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,d} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d,1} & a_{d,2} & \cdots & a_{d,d} \end{bmatrix}}_{A_B} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{bmatrix} = \underbrace{\begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \end{bmatrix}}_{\mathbf{b_B}}$$

$$\underbrace{\begin{bmatrix} a_{d+1,1} & a_{d+1,2} & \cdots & a_{d+1,d} \\ a_{d+2,1} & a_{d+2,2} & \cdots & a_{d+2,d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,d} \end{bmatrix}}_{A_N} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{bmatrix} \leq \underbrace{\begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \end{bmatrix}}_{\mathbf{b_N}}$$

Since A_B is invertible, we can rewrite $\mathbf{x} \in \Re^d$ as follows:

$$\begin{array}{lcll} A_B\mathbf{x} & \leq & \mathbf{b_B} & \\ A_B\mathbf{x}+\mathbf{y} & = & \mathbf{b_B} & \text{for some } \mathbf{y} \in \Re^d \\ \mathbf{x} & = & A_B^{-1}\mathbf{b_B} - A_B^{-1}\mathbf{y} & \end{array}$$

Using the above equation, we may express the constraints $A_N\mathbf{x} \leq \mathbf{b_N}$ in terms of the variables $\mathbf{y}$: this constitutes an affine transformation.

$$\underbrace{-A_N A_B^{-1}}_{A'} \; \mathbf{y} \; \leq \; \underbrace{\mathbf{b_N} - A_N A_B^{-1}\mathbf{b_B}}_{\mathbf{b}'}$$

If we were to add a slack variable (in the style of the simplex method) to each of the $m-d$ rows of A', we would obtain a system in the form of equation (5.6). At the feasible solution $\mathbf{v}$, we have $\mathbf{y} = \mathbf{0}$. This implies that $\mathbf{b}' \geq \mathbf{0}$ and, in the terminology of the simplex method, $\mathbf{y}$ constitutes the nonbasic variables. The basic variables are the slack variables.

This shows that whenever we have a feasible vertex $\mathbf{v}$ of P and know d linearly independent constraints whose boundaries intersect in $\mathbf{v}$, then we have all the information needed to exploit the pivoting strategies of the simplex method. For consistency with the simplex method, matrix A_B will be called the co-basis. Let the objective function in the initial space be $\mathbf{c} \cdot \mathbf{x}$. In the transformed space, this corresponds to

$$\mathbf{c} \cdot \left(A_B^{-1}\mathbf{b_B} - A_B^{-1}\mathbf{y}\right)$$

Ignoring the constant term, we have the objective function:

$$\text{maximize } \mathbf{c}' \cdot \mathbf{y}; \qquad \mathbf{c}' = -\mathbf{c}^{\mathrm{T}} A_B^{-1}$$

Let us denote the column vectors of A_B^{-1} as $\mathbf{z^j}$ $(1 \leq j \leq d)$. Similarly, let $a'_{i,j}$ and β'_i denote the elements of the transformed A' and $\mathbf{b}'$. The simplex method then selects pivots as follows:

- the p^{th} column enters the basis (*i.e.* leaves the co-basis) based on the test:

$$c'_p > 0$$

Which is equivalent to:

$$-\mathbf{c} \cdot \mathbf{z^p} > 0$$

- the r^{th} row enters the co-basis, for which $a'_{r,p} > 0$ and the ratio:

$$\frac{\beta'_r}{a'_{r,p}}$$

is minimized. This is equivalent to:

$$\begin{array}{l} \mathbf{a_r} \cdot \mathbf{z^P} < 0; \text{ and} \\ -\frac{\beta_r - \mathbf{a_r} \cdot \mathbf{v}}{\mathbf{a_r} \cdot \mathbf{z^P}} \text{ is minimized} \end{array}$$

where $\mathbf{v} = A_B^{-1}\mathbf{b_B}$ is the current solution.

Bland's rule (see page 64) can be used to select among all possible candidates for pivot row and pivot column, to ensure cycling does not occur.

Note that the column vectors of $-A_B^{-1}$ are the rays directed along the edges of P, from $\mathbf{v}$ (see figure 5.4). Hence the selection of which column to leave the co-basis simply tests which of these rays ($\mathbf{z^P}$) has a positive projection onto the direction of optimization; such a ray lies on all but one of the facets intersecting in $\mathbf{v}$. The row (r) to enter the co-basis is selected as the first constraint which is intersected by the ray. Since any row selected satisfies:

$$\mathbf{a_r} \cdot \mathbf{z^P} \neq 0$$

it follows that the $\mathbf{a_r} \cdot \mathbf{x} = b_r$ does not contain the intersection of the $d-1$ rows B_i ($i \neq p$). Hence, the new basis is also linearly independent. This conclusion also follows directly from lemma 5.1, which says that the new basis is invertible.

After each pivot, the matrix B^{-1} could be recomputed from scratch—an $O(d^3)$ operation—or it can be computed by updating the previous B^{-1} in $O(d^2)$ time, using the following rank-1 updates:

$$\begin{array}{lcll} z_i^p & \leftarrow & \frac{z_i^p}{\mathbf{z^P} \cdot \mathbf{a_r}} & \\ z_i^j & \leftarrow & z_i^j - z_i^p \frac{\mathbf{z^j} \cdot \mathbf{a_r}}{\mathbf{z^P} \cdot \mathbf{a_r}} & j \neq p \end{array} \tag{5.12}$$

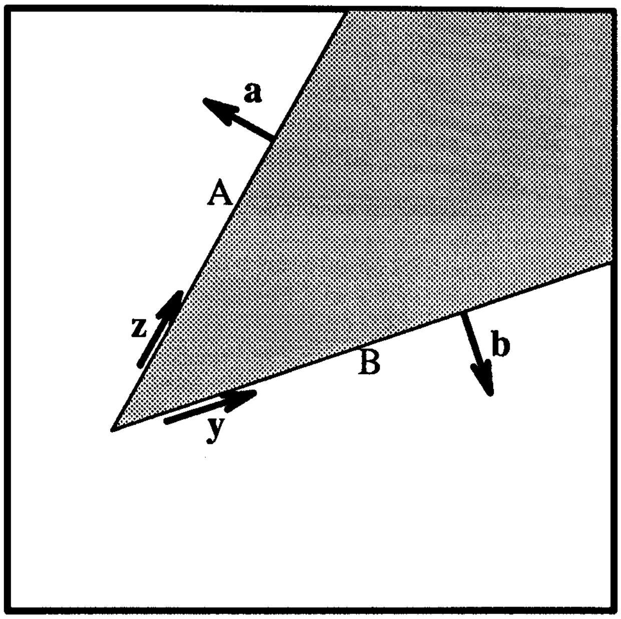

Figure 5.4: Illustration of Modified Simplex Pivot

For a feasible region $P \subset \Re^2$ (shaded region), a vertex $\mathbf{v}$ is determined by the intersection of the bounding lines A and B of 2 halfplanes, where:

$$A = \{\mathbf{x} | \mathbf{a} \cdot \mathbf{x} = \beta_1\}$$
$$B = \{\mathbf{x} | \mathbf{b} \cdot \mathbf{x} = \beta_2\}$$

and P lies in the intersection:

$$P \subseteq \{\mathbf{x} | \mathbf{a} \cdot \mathbf{x} \le \beta_1\} \cap \{\mathbf{x} | \mathbf{b} \cdot \mathbf{x} = \beta_2\}$$

Let M be the matrix whose row vectors are the normals of these halfplanes:

$$M = \begin{bmatrix} \leftarrow & \mathbf{a} & \rightarrow \\ \leftarrow & \mathbf{b} & \rightarrow \end{bmatrix}$$

Then the opposites of the column vectors of M^{-1} will be directed along the edges of P from $\mathbf{v}$.

$$M^{-1} = \begin{bmatrix} \uparrow & \uparrow \\ -\mathbf{y} & -\mathbf{z} \\ \downarrow & \downarrow \end{bmatrix}$$

Lemma 5.1 *Update system (5.12) applied to:*

$$B^{-1} = \begin{bmatrix} \leftarrow & \mathbf{b_1} & \rightarrow \\ \leftarrow & \mathbf{b_2} & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{b_d} & \rightarrow \end{bmatrix}^{-1} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{z^1} & \mathbf{z^2} & \cdots & \mathbf{z^d} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

will result in:

$$\begin{bmatrix} \leftarrow & \mathbf{b_1} & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{b_{p-1}} & \rightarrow \\ \leftarrow & \mathbf{a_r} & \rightarrow \\ \leftarrow & \mathbf{b_{p+1}} & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{b_d} & \rightarrow \end{bmatrix}^{-1} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{z^1} & \mathbf{z^2} & \cdots & \mathbf{z^d} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

Proof. The initial system implies that:

$$\begin{aligned} B_i \cdot \mathbf{z^i} &= 1; \quad 1 \leq i, j \leq d \\ B_i \cdot \mathbf{z^j} &= 0; \quad i \neq j \end{aligned}$$

The updates result in:

$$\begin{aligned} \mathbf{z^p} &\leftarrow \frac{\mathbf{z^p}}{\mathbf{z^p} \cdot \mathbf{a_r}} \\ \mathbf{z^j} &\leftarrow \mathbf{z^j} - \mathbf{z^p} \frac{\mathbf{z^j} \cdot \mathbf{a_r}}{\mathbf{z^p} \cdot \mathbf{a_r}} \quad j \neq p \end{aligned}$$

So it can easily be confirmed that

$$\begin{aligned} \mathbf{a_r} \cdot \mathbf{z^p} &= 1; \\ \mathbf{a_r} \cdot \mathbf{z^j} &= 0; \quad j \neq p \\ \mathbf{a_i} \cdot \mathbf{z^i} &= 1; \quad i \neq p \\ \mathbf{a_i} \cdot \mathbf{z^j} &= 0; \quad i \neq p; i \neq j \end{aligned}$$

Q.E.D.

5.7 Determination of Redundancy Using Modified Simplex Pivots

This section presents an algorithm to find the nonredundant constraints among the m constraints defining a polytope in $\Re^d$. It is similar to the sign-test methods [ZW83] [Gal83] [Tel83] [Rub83] reviewed in section 5.4. The main difference between these methods and the present one is that the present algorithm makes use of "modified simplex pivots" (presented in section 5.6), while the others use standard simplex pivots. Since the algorithm performs pivots from vertex to vertex of the polytope, it will not work in the degenerate situation of a polytope that does not contain any vertices.

Algorithm **Pivot** is shown in figure 5.5. Its input and output parameters are identical to those of algorithm **Non_Redundant** in figure 5.2, except that the initial feasible point $\mathbf{y}$ is required to be a vertex for the present algorithm, whereas $\mathbf{y}$ could be *any* feasible point for algorithm **Non_Redundant.** If no such point is available, we can find one by a linear programming call. Also, if only a non-vertex feasible point is available, it is easy "move" to a vertex by $< d$ iterations of intersecting a vector with the m constraints.

Algorithm **Pivot** determines the set $F \subseteq \{1, 2, \ldots, m\}$ of indices of nonredundant constraints. F is initialized (in step 1) to include any index i for which the boundary of the i^{th} constraint contains $\mathbf{y}$; the indices of the remaining constraints are put into the set W of "unknown" constraints (step 2). Each unknown constraint $\mathbf{a_i} \cdot \mathbf{x} \leq \beta_i$ is treated as the objective function $\mathbf{a_i} \cdot \mathbf{x}$: the constraint is nonredundant whenever the maximal value is β_i. In step (4), we store the value that the objective function corresponding to each unknown constraint attains at the initial vertex $\mathbf{y}$; we also store the indices of d linearly independent constraints which intersect at $\mathbf{y}$. This data is stored in the data structures μ and Γ (see table 5.1).

Modified simplex pivots are performed and, at each vertex $\mathbf{v}$ visited, we recompute the value $\mathbf{a_i} \cdot \mathbf{v}$ of each objective function. If the new value is greater than the current "maximum", then we update the corresponding fields of μ and Γ. So, whenever the i^{th} objective function is selected for optimization, we can "jump" [2] to the vertex—the intersection of the boundaries of the constraints indexed by $\Gamma(i)$—at which its value is known to be greatest. Hence, we will never pivot to the same vertex more than once.

[2]We will refer the operation of step (5b) as a "jump", and the operation of step (5c[i]) as a *bona-fide* pivot.

Algorithm Pivot(A, b, y, F)

```
1. $F \leftarrow$ the set of indices of the $\geq d$ constraints intersecting in $\mathbf{y}$
2. $W \leftarrow \{1, 2, \ldots, m\} - F$
3. let $B \subseteq F$ contain the indices of $d$ linearly independent row vectors of $A$: i.e. the
   constraints with indices in $B$ intersect in (and uniquely determine) $\mathbf{y}$
4. for all $i \in W$
   (a) $\mu(i) \leftarrow \mathbf{a_i} \cdot \mathbf{y}$
   (b) $\Gamma(i) \leftarrow B$
5. while $W \neq \emptyset$ do:
   (a) pick $\mathbf{a_j}$ as the objective function, for $j \in W$
   (b) $Z \leftarrow A_B^{-1}$;  $\mathbf{v} \leftarrow Z\mathbf{b_B}$; where $B$ denotes the set of $d$ indices $\Gamma(i)$
   (c) while $\mathbf{a_j}^T Z \not> \mathbf{0}$ and $\mathbf{a_j} \cdot \mathbf{v} < b_j$ do
       i. perform a modified simplex pivot, optimizing $\mathbf{a_j} \cdot \mathbf{x}$, such that constraint $p$
          leaves the co-basis, and constraint $r$ enters the co-basis; $p \in B$, $r \in W \cup F - B$
          • perform rank-1 update on $Z$
          • $\mathbf{v} \leftarrow Z\mathbf{b_B}$
       ii. if $r \in W$ then: $\left\{ \begin{array}{l} F \leftarrow F \cup \{r\} \\ W \leftarrow W - \{r\} \end{array} \right.$
       iii. $B \leftarrow B \cup \{r\} - \{p\}$
       iv. for all $i \in W$ do:
          • if $\mathbf{a_i} \cdot \mathbf{v} = b_i$ then: $\left\{ \begin{array}{l} F \leftarrow F \cup \{i\} \\ W \leftarrow W - \{i\} \end{array} \right.$
          • otherwise if $\mathbf{a_i} \cdot \mathbf{v} > \mu(i)$ then: $\left\{ \begin{array}{l} \mu(i) \leftarrow \mathbf{a_i} \cdot \mathbf{v} \\ \Gamma(i) \leftarrow B \end{array} \right.$
   (d) if $\mathbf{a_j}^T Z > \mathbf{0}$ and $\mathbf{a_j} \cdot \mathbf{v} < b_j$ then:
       • $W \leftarrow W - \{j\}$
```

Figure 5.5: Algorithm to Find All f Nonredundant Constraints of the m Inequalities Defining a Polytope P, Using Modified Simplex Pivots

(see also table 5.1)

Table 5.1: Parameters of Algorithm **Pivot**

(Algorithm is presented in figure 5.5)

Input	A $\mathbf{b}$ $\mathbf{y}$	an $m \times d$ matrix, with row vectors denoted $\mathbf{a_i}$, $1 \leq i \leq m$ an m-vector: $\mathbf{b} = (\beta_i, \ldots, \beta_m)$ $\mathbf{y} \in \Re^d$
	such that:	$\left\{ \begin{array}{l} \mathbf{a_i} \cdot \mathbf{x} \leq \beta_i \text{ defines a constraint of } P, \text{ for } 1 \leq i \leq m \\ \mathbf{y} \text{ is a vertex of } P \end{array} \right.$
Output	F	$F \subseteq \{1, \ldots, m\}$ is the set of indices of nonredundant constraints
Data Structures	$\mu(i)$ $\Gamma(i)$	maximum attained by $\mathbf{a_i} \cdot \mathbf{v}$, at any vertex $\mathbf{v}$ set of d indices of the constraints intersecting in that vertex $\mathbf{v}$

The main part of the algorithm is step (5), which is repeated as long as there is some unknown constraint. We select one such constraint $\mathbf{a_j} \cdot \mathbf{x} \leq \beta_j$, and jump to the vertex $\mathbf{v}$ at which $\mathbf{a_j} \cdot \mathbf{v}$ is known to be greatest, so far. Pivots are performed until either this constraint is found to intersect some vertex, or its objective value can no longer be increased. In the former case, the constraint is nonredundant; otherwise, it is redundant. At each vertex, we do not perform explicit redundancy tests for each constraint (since this would require $O(d^2)$ time per constraint). We do, however, test each i^{th} constraint for nonredundancy in step (5c[iv]): *i.e.* the test of whether the current objective value attains β_i. This test takes only $O(d)$ time per constraint and, once this computation has been done, updating of the data structures μ and Γ is, essentially, "free".

Table 5.2 compares algorithm **Pivot** to the sign-test methods, in terms of the time required for each component of pivoting and redundancy testing. In general, it will be the case that the number m of constraints is much greater than the dimension d. The comparison assumes that both methods are given the m constraints in $\Re^d$ in the form:

$$A\mathbf{x} \leq \mathbf{b}$$

together with some initial vertex $\mathbf{y}$, and the identities of d constraints whose boundaries

Table 5.2: Comparison of Standard Versus Modified Simplex Pivots In Redundancy Testing (Note: $m >> d$, in general)

		Standard Pivots (sign-test methods)	Modified Pivots (algorithm **Pivot**)
One Time Costs	Initialization	$O(md^2 + d^3)$	$O(d^3)$
	Redundancy Tests		$O(md^2)$ total
Cost Per Pivot	Redundancy Tests	$O(md)$ per pivot	
	Selection of Pivot Column	$O(d)$	$O(d^2)$
	Selection of Pivot Row	$O(m)$	$O(md)$
	Rank-1 Updates	$O(md)$	$O(d^2)$
Recomputation of Matrix (for "jumps" or numerical stability)		$O(md^2 + d^3)$	$O(d^3)$

intersect in $\mathbf{y}$. The sign-test methods must perform an affine transformation ($O(d^3 + md^2)$ time complexity), so that the constraints are in the the standard simplex form. Algorithm **Pivot** requires only $O(d^3)$ time to compute the inverse of the initial co-basis.

Performing each pivot requires $O(md)$ time for the sign-test methods; testing each unknown constraint for redundancy has the same complexity. So it is reasonable for these methods to perform the tests after each pivot.

The pivots performed by algorithm **Pivot** have the same complexity: $O(md)$ (assuming that $d \leq m$). Although the rank-1 updates require only $O(d^2)$ time, the selection of pivot row and pivot column has $O(md + d^2)$ complexity. We do not perform redundancy tests for all unknown constraints at each vertex: this would add $O(md^2)$ complexity to each pivot. Instead, we identify a single constraint as redundant if its objective value cannot be increased when it has been selected as the optimizing objective function (step (5a) of the algorithm): that is, the j^{th} constraint is redundant whenever $\mathbf{a_j}^T Z > \mathbf{0}$: *i.e.* when no pivot column can be selected. If such is the case, we choose another constraint to optimize, so the

$O(d^2)$ cost of unsuccessfully choosing a pivot column is the only redundancy test done for any constraint; whenever the test is successful, this $O(d^2)$ operation is a part of the pivot cost.

The above analysis hides the fact that a jump, with $O(d^3)$ complexity is done after an unsuccessful selection of a pivot column. However, this jump need not be performed if the next constraint selected has its maximum known objective value at the current vertex (and can be done by $O(d^2)$ rank-1 updating if its maximum is at an adjacent vertex). Otherwise, the jump brings us to a vertex at which the objective value is optimized, relative to the current vertex. The sign test methods would have required some number of pivots to achieve the same effect as the jump. Essentially, these jumps allow us to backtrack directly to the vertex at which any constraint has achieved its maximum known value, whereas the sign-test methods must backtrack one pivot at a time. In any case, at most m jumps are performed by the algorithm, adding a complexity of $O(md^3)$ overall. Provided at least d^2 pivots are performed by the algorithm, this cost is subsumed by the $O(md)$ complexity of each pivot.

Another advantage of the present method over the sign-test methods is that we can easily recompute the $d \times d$ matrix A_B^{-1} periodically to increase numerical stability: this requires $O(d^3)$ time, and no additional space. If the sign test methods were to recompute the simplex tableau, it would require $O(md^2 + d^3)$ time, and assumes that the original coefficients are stored, requiring an additional $O(md)$ space.

The disadvantage of the current approach is: it may be the case that the newly selected constraint is redundant and has its *optimal* objective value at the vertex which is currently known to be maximal. In this case, the sign-test methods would have already discarded it, but our approach must execute an $O(d^3)$ jump in order to perform the redundancy test. This event, however, is unlikely when the problem size is large: that is, a constraint is found to be redundant only at the vertex where its objective is optimal—this vertex is unique provided no two constraints have parallel boundaries. Section 6.3 presents computational results which show that the number of vertices visited by algorithm **Pivot** becomes very small, relative to the total number of vertices, as the dimension is increased. Hence, "stumbling across" the optimal vertex of some constraint, while optimizing the objective value of a different constraint, should not be a frequent event.

5.8 Transformation of Stage 1 Problem

The stage 1 problem is to identify the relevant points of $S \subset \Re^d$ with respect to some nonempty Voronoi polyhedron $V_k^S(T)$. Assume that we are given the subset $T \subset S$, together with some boundary point $\mathbf{f}$ of $V_k^S(T)$:

$$\mathbf{f} \in V_k^S(T) \cap B(\mathbf{a}, \mathbf{b}); \qquad \mathbf{a} \in T; \ \mathbf{b} \in S - T$$

It follows directly from the definitions that both $\mathbf{a}$ and $\mathbf{b}$ are relevant, and that the hypersphere centered at $\mathbf{f}$ with $\mathbf{a}$ and $\mathbf{b}$ on its surface separates T from $S - T$.

By lemma 3.13, for any relevant $\mathbf{q} \in S$, there exists a hypersphere with $\mathbf{q}$ on its surface, separating T from $S-T$. This is a problem of spherical separability. We will be transforming this into a problem of determining nonredundancy among a system of constraints in $\Re^{d+1}$ by the following sequence of operations on $\mathbf{p} = (p_1, p_2, \ldots, p_d) \in S$.

1. The paraboloid transformation (definition 2.1) of each $\mathbf{p} \in S$ into $\Re^{d+1}$:

   $$\mathbf{p} \rightarrow \mathbf{p}^* = (p_1, p_2, \ldots, p_{d+1}); \qquad p_{d+1} = \sum_{i=1}^{d} p_i^2$$

   Let $S^* \subset \Re^{d+1}$ denote the set obtained by paraboloid transformation of any $S \subset \Re^d$.

   This transforms the spherical separability problem into a linear separability problem (see figure 5.6): for any relevant $\mathbf{q}$, there will be a hyperplane $\mathbf{a} \cdot \mathbf{x} = \beta$ (with $a_{d+1} \leq 0$) through $\mathbf{q}$ which separates T^* from $S^* - T^*$.

2. Vertically projecting S^* onto the hyperplane $\{\mathbf{x} \in \Re^{d+2} | x_{d+2} = 1\}$, followed by rotating T^* about the origin $\mathbf{0}$.

   $$\mathbf{p}^* \rightarrow \mathbf{p}^{**} = \begin{cases} (-p_1, -p_2, \ldots, -p_{d+1}, -1) & \forall \mathbf{p} \in T \\ (p_1, p_2, \ldots, p_{d+1}, 1) & \forall \mathbf{p} \in S - T \end{cases}$$

   Let $S^{**} \subset \Re^{d+2}$ denote the set so obtained from $S^* \subset \Re^{d+1}$.

   This transforms the problem into one of finding those (relevant) points $\mathbf{q}$ for which there exists a supporting hyperplane of $S^{**} \cup \{\mathbf{v}\}$ (where $\mathbf{v} = (0, \ldots, 0, 1, 0)$, the $(d+1)^{st}$ standard unit vector) which passes through $\mathbf{q}$ and $\mathbf{0}$. The transformation is illustrated from the top frame to the middle frame of figure 5.7. Although the example shown does does not correspond to a meaningful stage 1 problem—since it is

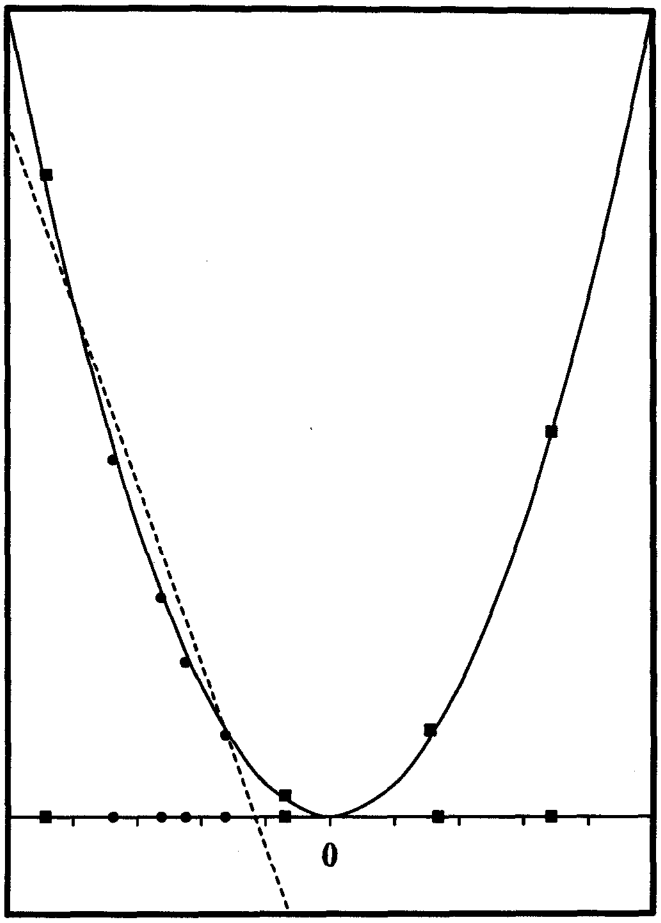

Figure 5.6: The Paraboloid Transformation

The points of T (denoted by circles) and the points of $S - T$ (denoted by squares) are spherically separable in $\Re^1$ (bottom line). After vertical projection onto the paraboloid, the points are linearly separable: the dashed line indicates one possible linear separator. Note that a point is relevant iff there exists such a linear separator which passes through it.

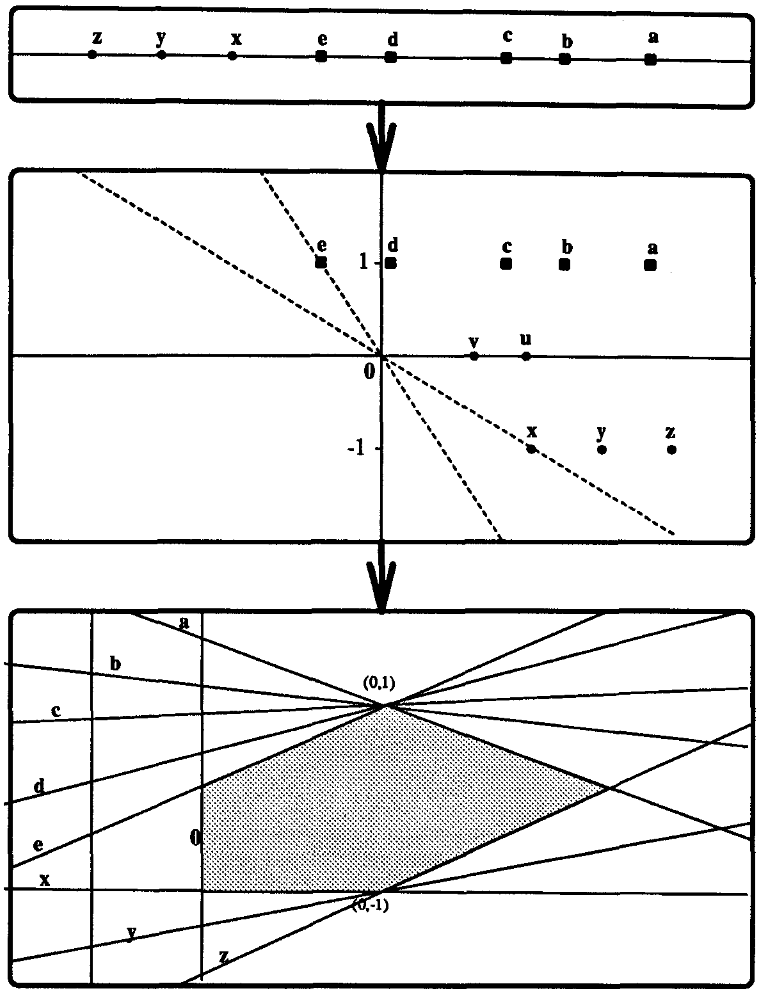

Figure 5.7: Transformation of Linear Separability Problem Into Problem of Determining Nonredundant Constraints

Top Frame The sets $S^* - T^* = \{\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{e}\}$ (denoted by squares) and $T^* = \{\mathbf{x},\mathbf{y},\mathbf{z}\}$ (denoted by circles) are linearly separable in $\Re^1$, by separators passing through the relevant points **e** or **x**.

Middle Frame There exist supporting lines (dashed lines) of S^{**} which pass through the relevant points. The additional point **v** is shown. An interior point **u** of $CH(S^{**} \cup \{\mathbf{0v}\})$ is shown.

Bottom Frame The boundary of each halfspace is labeled with the name of the point from which it derives (note that $D(-\mathbf{u})$ derives from the origin **0** of the middle frame). The unlabeled constraint corresponds to the point **v**; in this case, the constraint is strongly redundant.

The boundaries of the halfspaces which are dual to the relevant points intersect the boundary of $D(-\mathbf{u})$. The boundary of each constraint corresponding to $S^* - T^*$ passes through $(0,1)$, and those corresponding to T^* pass through $(0,-1)$, in the translated space.

an $\Re^1 \rightarrow \Re^2$ transformation, implying that the original problem was 0-dimensional—it does illustrate the technique used.

3. Translation of the coordinate system so that the new origin lies at some point $\mathbf{u}$ which is interior to $CH(S^{**} \cup \{\mathbf{0}, \mathbf{v}\})$: A suitable point $\mathbf{u}$ is indicated in the middle frame of figure 5.7.

$$\begin{array}{lcll} \mathbf{p}^{**} & \rightarrow & \mathbf{p}' & = \mathbf{p}^{**} - \mathbf{u} \\ \mathbf{v} & \rightarrow & \mathbf{v}' & = \mathbf{v} - \mathbf{u} \\ \mathbf{0} & \rightarrow & -\mathbf{u} & \end{array}$$

   Let S' denote the set of these $|S| + 2$ translated points.

4. Taking the polar dual $D(\mathbf{t})$ of each $\mathbf{t} \in S'$ (see definition 2.5). This transforms the problem into one of determining those (relevant) points $\mathbf{q}$ for which $D(\mathbf{q}')$ and $D(-\mathbf{u})$ intersect in a face of P':

$$P' = \bigcap_{\mathbf{t} \in S'} D(\mathbf{t}); \qquad D(\mathbf{t}) = \{\mathbf{x} \in \Re^{d+2} | \mathbf{t} \cdot \mathbf{x} \leq 1\} \tag{5.13}$$

   The transformation is shown from the middle frame to the bottom frame of figure 5.7.

5. Intersection of each $D(\mathbf{t})$ with $D(-\mathbf{u})$, for $\mathbf{t} \in S' - \{-\mathbf{u}\}$. Let $I(\mathbf{t})$ denote the intersection $D(\mathbf{t}) \cap D(-\mathbf{u})$ for $\mathbf{t} \in S' - \{-\mathbf{u}\}$. Now the relevant points $\mathbf{q} \in S$ are those points for which $I(\mathbf{q}')$ is a nonredundant constraint of the polytope:

$$P = \bigcap_{\mathbf{p} \in S \cup \{\mathbf{v}\}} I(\mathbf{p}') \tag{5.14}$$

   P is the facet of P' contained by the boundary of $D(-\mathbf{u})$. This can be seen in the bottom frame of figure 5.7.

The transformation is explained in algebraic terms below:

Our initial spherical separability problem (lemma 3.13) states that $\mathbf{q}$ is relevant with respect to $V_k^S(T)$ if and only if there is a hypersphere $C(\mathbf{c}, \varrho)$ with $\mathbf{q}$ on its surface, separating T from $S - T$. Hence, according to lemma 2.2, $\mathbf{q}$ is relevant if and only if the following system has a feasible solution $\mathbf{x} \in \Re^{d+1}$:

$$\begin{array}{lcll} \mathbf{q}^* \cdot \mathbf{x} & = & \beta; & \\ \mathbf{p}^* \cdot \mathbf{x} & \geq & \beta; & \forall \mathbf{p}^* \in T^* \\ \mathbf{p}^* \cdot \mathbf{x} & \leq & \beta; & \forall \mathbf{p}^* \in S^* - T^* \\ x_{d+1} & \leq & 0 & \end{array} \tag{5.15}$$

This is equivalent to the existence of a feasible solution $\mathbf{x} \in \Re^{d+2}$ of the following system:

$$\begin{array}{rcl} \mathbf{q}^{**} \cdot \mathbf{x} & = & 0; \\ \mathbf{p}^{**} \cdot \mathbf{x} & \leq & 0; \;\; \forall \mathbf{p}^{**} \in S^{**} \\ x_{d+1} & \leq & 0; \end{array} \tag{5.16}$$

where $\mathbf{p}^{**}$ is as earlier defined.

Let $\mathbf{v} = (0, 0, , \ldots , 0, 1, 0) \in \Re^{d+2}$ be the $(d+1)^{st}$ standard unit vector. Then we may simplify linear program (5.16) as:

$$\begin{array}{rcl} \mathbf{q}^{**} \cdot \mathbf{x} & = & 0; \\ \mathbf{p}^{**} \cdot \mathbf{x} & \leq & 0; \;\; \forall \mathbf{p}^{**} \in S^{**} \cup \{\mathbf{v}\} \end{array}$$

Let C denote the convex hull of $S^{**} \cup \{\mathbf{v}, \mathbf{0}\}$. It is clear that $\mathbf{0}$ is an extreme point of C. Equation 5.16 implies that $\mathbf{q}^{**}$ lies on a common face with $\mathbf{0}$ in C for every relevant $\mathbf{q}$.

Let us select some interior point $\mathbf{u}$ of C, and without loss of generality, assume that $u_{d+1} \neq 0$ and $u_{d+2} = 0$. By translating the coordinate system so that $\mathbf{u}$ is the origin, we obtain:

$$S' = \{\mathbf{p}^{**} - \mathbf{u} | \; \mathbf{p}^{**} \in S^{**}\} \cup \{-\mathbf{u}, \mathbf{v} - \mathbf{u}\}$$

Let $\mathbf{p}' \in S'$ denote the translation of any $\mathbf{p}^{**} \in S^{**}$, and let $\mathbf{v}'$ denote the translation of $\mathbf{v}$. The polar dual $D(\mathbf{t})$ of any $\mathbf{t} \in S'$ may be expressed in terms of the coordinates of $\mathbf{u}$ and of the points of S as follows:

$$\begin{array}{ccccccl} \sum_{i=1}^{d}(-p_i - u_i) \cdot x_i & + & \left(-u_{d+1} - \sum_{i=1}^{d} p_i^2\right) \cdot x_{d+1} & - & x_{d+2} & \leq 1 & \forall \mathbf{p} \in T \\ \sum_{i=1}^{d}(p_i - u_i) \cdot x_i & + & \left(-u_{d+1} + \sum_{i=1}^{d} p_i^2\right) \cdot x_{d+1} & + & x_{d+2} & \leq 1 & \forall \mathbf{p} \in S - T \\ -\sum_{i=1}^{d} u_i \cdot x_i & + & (1 - u_{d+1}) \cdot x_{d+1} & & & \leq 1 & \end{array} \tag{5.17}$$

By lemma 2.3: $\mathbf{q} \in S$ is relevant with respect to $V_k^S(T)$ if and only if $D(-\mathbf{u})$ and $D(\mathbf{q}')$ intersect in a face of the polytope P' (see equation 5.13) which is the intersection of the constraints of system (5.17) with the constraint $D(-\mathbf{u}) : \mathbf{u} \cdot \mathbf{x} \geq 1$. Since $D(-\mathbf{u})$ is strongly nonredundant in P', it follows that $bd(D(-\mathbf{u})) \cap P$ is a $(d+1)$-dimensional facet of P'; hence, it is a $(d-1)$-dimensional polytope whose faces are precisely those faces of P' which are lie in $bd(D(-\mathbf{u}))$. Hence the intersection of $bd(D(-\mathbf{u}))$ with $D(\mathbf{p}')$, for all $\mathbf{p}' \in S'$ is a polytope P (see equation 5.14) such that $\mathbf{q}$ is relevant if and only if $I(\mathbf{q})$ is nonredundant in P.

Since we have assumed that $u_{d+2} = 0$ and $u_{d+1} \neq 0$, the equation of $bd(D(-\mathbf{u}))$ may be written as:

$$\begin{array}{rcl} \mathbf{u} \cdot \mathbf{x} &=& -1 \\ i.e.\ \sum_{i=1}^{d+1} u_i x_i &=& -1 \\ i.e.\qquad x_{d+1} &=& -\frac{1+\sum_{i=1}^{d} u_i \cdot x_i}{u_{d+1}} \end{array}$$

So, we can derive the equation for each $I(\mathbf{p}')$, by making the above substitution in system (5.17), resulting in:

$$\begin{array}{rcrcl} \sum_{i=1}^{d} \left(-p_i + \frac{u_i \cdot p_{d+1}}{u_{d+1}}\right) \cdot x_i & - & x_{d+2} & \leq & -\frac{p_{d+1}}{u_{d+1}} \quad \forall \mathbf{p} \in T \\ \sum_{i=1}^{d} \left(p_i - \frac{u_i \cdot p_{d+1}}{u_{d+1}}\right) \cdot x_i & + & x_{d+2} & \leq & -\frac{p_{d+1}}{u_{d+1}} \quad \forall \mathbf{p} \in S - T \\ -\sum_{i=1}^{d} \frac{u_i \cdot x_i}{u_{d+1}} & & & \leq & \frac{1}{u_{d+1}} \end{array} \tag{5.18}$$

These are the $n+1$ halfspaces which intersect in P: each point of S corresponds to one of the halfspaces, and the $(n+1)^{st}$ is present to ensure that the corresponding hypersphere (in the original space) contains T, excluding $S-T$, and not the other way around. This $(n+1)^{st}$ constraint will be nonredundant only when T is spherically separable from $S-T$, and $S-T$ is spherically separable from T. In other words, it will be nonredundant only for those unbounded $V_k^S(T)$ for which T and $S-T$ are linearly separable (in the original space).

Note that P lies in a $(d-1)$-dimensional space whose coordinates are, as an artifact of our derivation: $1, 2, \ldots, d, d+2$. If we are given some point:

$$\mathbf{f} \in V_k^S(T) \cap B(\mathbf{a}, \mathbf{b}); \qquad \mathbf{a} \in T;\ \mathbf{b} \in S - T$$

for some nonredundant $B(\mathbf{a}, \mathbf{b})$, then we can easily calculate an initial boundary point $\mathbf{g}$ of P, lying in the intersection of $I(\mathbf{a})$ and $I(\mathbf{b})$:

$$\mathbf{g} = \left(\frac{2 \cdot f_1}{\beta}, \frac{2 \cdot f_2}{\beta}, \ldots, \frac{2 \cdot f_d}{\beta}, \frac{\varrho^2 - \sum_{i=1}^{d} f_i^2}{\beta}\right) \tag{5.19}$$

where:

$$\begin{array}{l} \varrho = d(\mathbf{f}, \mathbf{a}) \\ \beta = u_{d+1} - 2\sum_{i=1}^{d} f_i \cdot u_i \end{array}$$

Lemma 5.2 **g** *is a boundary point of P, lying in the intersection of the boundaries of $I(\mathbf{a}')$ and $I(\mathbf{b}')$.*

Proof. In the $(d+2)$-dimensional space, $\mathbf{g}$ corresponds to the point:

$$\mathbf{g}' = \left(\frac{2f_1}{\beta}, \frac{2f_2}{\beta}, \dots, \frac{2f_d}{\beta}, -\frac{1}{\beta}, \frac{\varrho^2 - \sum_{i=1}^{d} f_i^2}{\beta}\right)$$

Taking the polar dual, and translating the resulting halfspace we derive:

$$\begin{array}{llcl}
 & \left(\frac{2f_1}{\beta}, \frac{2f_2}{\beta}, \dots, \frac{2f_d}{\beta}, -\frac{1}{\beta}, \frac{\varrho^2 - \sum_{i=1}^{d} f_i^2}{\beta}\right) \cdot (\mathbf{x} - \mathbf{u}) & & \leq 1 \\
i.e. & \left(\frac{2f_1}{\beta}, \frac{2f_2}{\beta}, \dots, \frac{2f_d}{\beta}, -\frac{1}{\beta}, \frac{\varrho^2 - \sum_{i=1}^{d} f_i^2}{\beta}\right) \cdot \mathbf{x} & - \frac{-\beta}{\beta} & \leq 1 \\
i.e. & \left(2f_1, 2f_2, \dots, 2f_d, -1, \varrho^2 - \sum_{i=1}^{d} f_i^2\right) \cdot \mathbf{x} & & \leq 0
\end{array}$$

We have already seen that this corresponds directly to a supporting hyperplane of S^{**}, passing through $\mathbf{a}^{**}$, $\mathbf{b}^{**}$ and $\mathbf{0}$. By duality, therefore, $\mathbf{g}$ is a boundary point of P lying in the intersection of the boundaries of $I(\mathbf{a}')$ and $I(\mathbf{b}')$.

Q.E.D.

5.9 A Practical Simplex-Based Algorithm

Algorithm **Facet_Enumeration** for the enumeration of all f facets of the order-k Voronoi Diagram of a set S of n points in $\Re^d$ is shown in figure 5.8. The algorithm considers each Voronoi polytope, in turn, and enumerates all of its V-facets by detecting the nonredundant constraints among the $k(n-k)$ constraints which define it, by equation (3.1). Algorithm **Pivot** (presented in section 5.7) is used for the determination of nonredundancy. The transformation (presented in section 5.8) of the stage 1 problem into a problem of determining nonredundancy among a system of linear constraints is used. The running time of algorithm **Facet_Enumeration** cannot be bounded in an output sensitive manner, since the determination of redundancy is performed by techniques based on the simplex method.

The algorithm makes use of two data structures (see table 5.3): a balanced search tree (denoted Ψ) [AHU83, section 5.4], and a stack (denoted Φ) [AHU83, section 2.3]. These were discussed in conjunction with algorithm **Vertex_Enumeration** on page 47.

The balanced search tree Ψ is used to hold a record for each Voronoi polytope $V_k^S(T)$ discovered by the algorithm, and is keyed by the lexicographically sorted list of indices of

Table 5.3: Parameters of Algorithm **Facet_Enumeration**

(Algorithm is presented in figure 5.8)

Input	d k n $S \subset \Re^d$	dimension size of input point set, S $S = \{\mathbf{p_1}, \mathbf{p_2}, \cdots, \mathbf{p_n}\}$
Output	A list of the nonempty Voronoi polytopes in V_k^S, together with the facets of each.	
Data Structures	Ψ Φ	balanced search tree containing $T \subset S$, for each "known" $V_k^S(T)$ (keyed by the lexicographically sorted list of k indices i, for $p_i \in T$.) stack of records $(T, \mathbf{x})$ for $V_k^S(T)$ left to be processed $\mathbf{x}$ is a vertex of $V_k^S(T)$

```
Procedure new_facet($T, t, s, \mathbf{y}$)
    • output: facet generated by $B(\mathbf{p_t}, \mathbf{p_s})$
    • if $T - \{\mathbf{p_t}\} \cup \{\mathbf{p_s}\}$ not in $\Psi$ then:
        • insert $T - \{\mathbf{p_t}\} \cup \{\mathbf{p_s}\}$ into $\Psi$
        • push record $(T - \{\mathbf{p_t}\} \cup \{\mathbf{p_s}\}, \mathbf{y})$ onto $\Phi$.
```

the elements of T. Ψ is queried every time a facet is found, to see if the polytope which is adjacent along that facet has already been discovered by the algorithm.

The stack Φ holds a record for each Voronoi polytope, discovered during the redundancy testing of some adjacent polytope, but which has not yet had its facets enumerated. A record for each polytope is pushed onto the stack as soon as it is discovered for the first time. For each $V_k^S(T)$ on the stack, we need to store the indices of T, and some V-vertex $\mathbf{y}$. Note that any new polytope is discovered at some vertex $\mathbf{y}$ of an adjacent polytope, and $\mathbf{y}$ is also a vertex of the new polytope. If S is nondegenerate, the generating set of each V-vertex has size exactly $d + 1$ (lemma 3.15); in this case, it will also be helpful to store the indices of the generating set on the stack.

Algorithm Facet_Enumeration

```
1. find an initial V-vertex $\mathbf{y} \in \Re^d$, and a set $T$ such that $\mathbf{y} \in V_k^S(T)$
2. push record $(T, \mathbf{y})$ onto $\Phi$
3. insert $T$ into $\Psi$
4. while $\Phi$ not empty
    (a) pop top record (T, $\mathbf{y}$)
    (b) output: indices of $T$
    (c) call Algorithm Pivot($A, \mathbf{b}, \mathbf{g}, F$) where:
        • $A$ is the $(n+1) \times (d+1)$ matrix, and $\mathbf{b}$ the $(n+1)$-vector containing the
          coefficients of system 5.18
        • $\mathbf{g}$ is calculated by equation 5.19
    (d) $S' \leftarrow \left\{\mathbf{p_j} | j \in F\right\}$; $n' = |S'|$
    (e) $T' \leftarrow S' \cap T$; $|T'| = k'$
    (f) call Algorithm Pivot($A, \mathbf{b}, \mathbf{y}, f$) where:
        • $A$ is a $k'(n'-k') \times d$ matrix with row vectors denoted $\mathbf{a_h}$ $(1 \leq h \leq n'(n'-k'))$
        • $\mathbf{b} = (\beta_1, \beta_2, \ldots, \beta_{n'(n'-k')})$
        • such that: for each $h$ $(1 \leq h \leq n'(n'-k'))$, there is a unique pair, $\mathbf{p_i} \in T'$
          and $\mathbf{p_j} \in S' - T'$, for which:

              $\overline{H(\mathbf{p_i}, \mathbf{p_j})} = \{\mathbf{x} | \mathbf{a_h} \cdot \mathbf{x} \leq \beta_h\}$

        • for each nonredundant constraint $\overline{H(\mathbf{p_t}, \mathbf{p_s})}$ identified in step (1), procedure
          new_facet($T, t, s, \mathbf{y}$) is called.
        • whenever a nonredundant constraint $\overline{H(\mathbf{p_t}, \mathbf{p_s})}$ is found in step (5c[ii]) or
          (5c[iv]), at some vertex $\mathbf{v}$, then procedure new_facet($T, t, s, \mathbf{v}$) is called.
```

Figure 5.8: Algorithm to Enumerate All Facets in the Order-k Voronoi Diagram of $S \subset \Re^d$

(see also table 5.3)

Step (1) of the algorithm—the determination of an initial V-vertex—can be done as described on page 48. Steps (2) and (3) are trivial.

In step (4), we transform the stage 1 problem into a problem of determining nonredundancy (see section 5.8), which is solved using algorithm **Pivot** (see section 5.7). This provides the set S' ($|S'| = n'$) of relevant points: k' of which constitute the $T' \subseteq T$. Then, in stage 2, we determine the nonredundant constraints among the intersection:

$$V_k^S(T) = \bigcap_{\substack{\mathbf{p} \in T' \\ \mathbf{q} \in S'-T'}} \overline{H(\mathbf{p},\mathbf{q})}$$

Whenever some constraint $\overline{H(\mathbf{p},\mathbf{q})}$ is found to be nonredundant, procedure **new_facet** is called (see table 5.3) which, in addition to generating output, checks to see if the adjacent polytope $V_k^S(T - \{\mathbf{p}\} \cup \{\mathbf{q}\})$ is in the search tree. If not, a record for the new polytope is added to both the search tree and to the stack.

The first step of algorithm **Pivot** detects which of the constraints contain the initial vertex $\mathbf{y}$ in their boundaries. In the case of a nondegenerate point set S, this step can be avoided by including, with the record for $V_k^S(T)$ on the stack, the indices of the $d+1$ elements of the generating set G of $\mathbf{y}$.

- For the stage 1 call to algorithm **Pivot**—*i.e.* from **Facet_Enumeration** step (4c)—the initial vertex $\mathbf{g}$, calculated by equation 5.19, is contained by the boundaries of the constraints corresponding all $\mathbf{p} \in G$. This follows from lemma 5.2. Furthermore, $\mathbf{g}$ will not be contained by the constraint corresponding to any point of $S - G$; otherwise, by duality, this would imply that more than $d+1$ points are in the generating set of $\mathbf{y}$.

- For the stage 2 call to algorithm **Pivot**—*i.e.* from **Facet_Enumeration** step (4f)—the initial vertex $\mathbf{y}$ is contained by the boundary of $\overline{H(\mathbf{p},\mathbf{q})}$ iff $\mathbf{p},\mathbf{q} \in G$.

5.10 An $O(nf)$ Output-Sensitive Algorithm

By using algorithm **Non_Redundant** (figure 5.2) instead of algorithm **Pivot**, for the determination of redundancy, we can obtain an output sensitive bound of $O(n)$ per facet, in fixed dimension, on the running time of algorithm **Facet_Enumeration** (figure 5.8). For every new $V_k^S(T)$ pushed onto the stack—discovered in step (2c[iii]) of algorithm **Non_Redundant**—we know some $\mathbf{x} \in V_k^S(T)$, which lies on the boundary $B(\mathbf{p},\mathbf{q})$ of some nonredundant constraint. Hence we can include, with the record for $V_k^S(T)$ in the stack, the indices of $\mathbf{p}$ and $\mathbf{q}$, thereby avoiding one iteration of the nonredundancy algorithm.

Any call to procedure **new_facet** has complexity $O(kd\log n)$, dominated by the query (and insertion, in the case of an negative response to the query) into the search tree Ψ. As discussed on page 20, the maximum number of Voronoi polytopes can be bounded by $O(k^{\lceil\frac{d+1}{2}\rceil}n^{\lfloor\frac{d+1}{2}\rfloor})$. [CS89]. It follows that any addition or query to Ψ requires $O(d\log n)$ probes. Each probe requires $O(k)$ comparisons of indices: hence the complexity of any insertion or query is $O(kd\log n)$.

It has been shown in section 5.4, that the complexity of algorithm **Non_Redudnant** in determining the f nonredundant constraints among a set of m constraints is $O(3^{d^2}mf)$ using the technique of [Meg84], as modified by [Dye86] [Cla86]. In fixed dimension, this bound can be expressed as $O(mf)$.

For a given Voronoi polytope $V_k^S(T)$, let f' denote the number of its V-facets; let n' be the total number of relevant points, and let k' be the number of relevant points in the set T. The stage 1 call to algorithm **Non_Redundant** has $m = n+1$ constraints of which n' are nonredundant: so the complexity will be $O(3^{d^2}nn')$. Clearly, the coefficients of the constraints—given by system (5.18)—and of the initial vertex—given by equation (5.19)—can be calculated in $O(nd)$ time. Hence, the overall complexity of stage 1 is $O(3^{d^2}nn')$. Every relevant point is in the generating set of some facet, so there are at least $\frac{n'}{2}$ facets. Therefore, this complexity may be expressed as $O(3^{d^2}n)$ per facet.

The stage 2 call to algorithm **Non_Redundant** has $k'(n'-k')$ constraints of which f' are nonredundant, so the complexity will be $O(3^{d^2}k'n'f')$. Again, the coefficients of the constraints can easily be calculated in $O(nd)$ time. For every facet, a call is made to procedure **new_facet**: this adds an additional $O(kd\log n)$ to the complexity. Therefore, the overall complexity of stage 2 is $O(3^{d^2}k'(n'-k') + kd\log n)$ per facet. Let $c = n' - k'$ be the number of relevant points in $S - T$: then this complexity can be expressed as $O(3^{d^2}kc + kd\log n)$ per facet.

So, the overall complexity of enumerating the facets of a single Voronoi polytope is $O(3^{d^2}(n+kc) + kd\log n)$ per facet.

Each facet of the order-k Voronoi Diagram is found exactly twice by the algorithm, since it lies on precisely two Voronoi polytopes. So, if f is the total number of facets in

V_k^S, then the complexity of enumerating all f facets is $O(3^{d^2}(n + kc)f + kdf \log n)$, where c is the maximum number of relevant points from $S - T$ of any $V_k^S(T)$. The value of c is bounded from above by the maximum number of facets in any given Voronoi polytope. This number is widely conjectured to be a constant and, in section 6.2, computational results are presented which support this claim.

Chapter 6

Implementation and Experimental Results

6.1 Computational Experience

6.1.1 IMSL/NAG Linear Programming Routines

The first version of the facet enumeration algorithm to be implemented made use of linear programming routines from the NAG [Num91] and IMSL [IMS87] libraries. The implementation was virtually identical to the "output-sensitive" algorithm of section 5.10. Although the linear programming routines did not make use of Megiddo's [Meg84] [Dye86] [Cla86] linear time algorithm, they were in effect a "black box" for solving LPPs. That is, the strategy of algorithm **Non_Redundant** (figure 5.2) was used. The complexity bound of $O(n)$ per facet in fixed dimension does not hold, since the LPPs were solved by methods which, although efficient in practice, can not be bounded by a linear function in the number of constraints.

- the IMSL routine "DLPRS" [IMS87, pp. 888–891] uses the revised simplex method. It always terminated successfully—with an optimal point or a message that no feasible point existed—even in the case of degenerate input.

- the NAG routine "E04MBF" [Num91, chapter E04] does not employ the simplex method; it is possible to request that the subroutine returns after finding a feasible point (omiting the optimization phase). In the case of degenerate input, it sometimes performed excessive iterations and terminated unsuccessfully. However, E04MBF did return successfully in most cases, and its execution time was considerably faster than that of DLPRS.

In both cases, the double-precision versions of the routines were used. The fact that the NAG routine performed more quickly than the IMSL routine should not have been greatly influenced by the additional optimization phase (which could be omitted by NAG, but not by IMSL). This is because all coefficients of the objective function were set to 0: hence, any feasible solution would be optimal.

In the implementation, E04MBF was always called first, with a low limit set on the number of allowable iterations. Any time that it terminated unsuccessfully, DLPRS was called. In all of the executions that were performed, there was never a time in which DLPRS was able to find a feasible point after E04MBF had terminated unsuccessfully. Hence, an unsuccessful termination could have been interpreted as a "no feasible point" response, without affecting any of the results.

6.1.2 Simplex Pivots

The execution time of the NAG/IMSL implementation was quite slow, especially for large problem sizes. This fueled the search for ways to exploit the inner workings of the simplex method in order to simultaneously consider each constraint for redundancy. As a result, algorithm **Pivot** (figure 5.5) was developed.

At first, a "sign-test method" implementation was used, in the style of [ZW83] [Gal83] [Tel83] [Rub83] (see section 5.4), in order to determine redundancy. This proved to be numerically unstable: after a sequence of pivots, updating the coefficients of the simplex tableau, errors become magnified. We could have recomputed the coefficients—an $\theta(d^3 + md^2)$ affine transformation—at regular intervals in order to restore stability. If this were done after every constant number (*i.e.* independent of m and of d) of pivots, it would add considerably to the complexity of each pivot.

Algorithm **Pivot** (figure 5.5) makes use of modified simplex pivots (introduced in section 5.6). The recomputation needed to increase stability is simply an inversion of a $d \times d$ matrix B: *i.e.* an $O(d^3)$ operation. To avoid all numerical problems, the matrix B^{-1} was recomputed after every pivot. Section 5.6 presents an $O(d^2)$ method to update B^{-1} after a pivot to an adjacent vertex, but this was not implemented. In any case, the reinversion is necessary after any "jump" (pivot to a nonadjacent vertex) which occurs in step (5c) of algorithm **Pivot**.

6.2 Results on the Complexity of Voronoi Diagrams

The implementation of algorithm **Facet_Enumeration** has been used to compute the d-dimensional order-k Voronoi Diagram of randomly chosen point sets of varying sizes, and for varying values of k and d. For the case of $k = 1$, a two stage algorithm would be superfluous; so the V_1^S were computed by using "stage 2" on all possible $n - 1$ constraints defining each polytope. The V_k^S for $2 \leq k \leq 5$ were computed with the two stage approach.

Tables 6.1, 6.2, 6.3, 6.4 and 6.5 show the results for points sets of size 50, 100, 200, 500 and 1000, respectively, randomly chosen from a uniform distribution in the unit hypercube. The value of k was varied from 1 to 5, and the value of d was varied from 2 to 6. Tables 6.6, 6.7, 6.8, 6.9 and 6.10 show the results for points chosen from a uniform distribution in the interior of a unit hypersphere. Each box of these tables represents a single run of the program, for given values of n, k and d. In the case of the smaller problems, the runs were repeated with several different sets of data: the results were virtually identical. The results for both distributions—inside the hypersphere and inside the hypercube—also are virtually identical: this is not surprising, since only the peripheral polytopes in V_k^S would be expected to be different between the two distributions.

The tables show the number r of regions, the number f of facets and the cpu time required on a Silicon Graphics 4D/320S. Also shown are the average number $\frac{f}{r}$ of facets per region—note that each facet lies in two different regions, so the number reported is actually $\frac{2f}{r}$—and the cpu time $\frac{t}{r}$ per facet expressed in milliseconds (ms). The tables are arranged to allow comparison of data for increasing d (down the columns) and for increasing k (along the rows).

One additional run of the program was performed on a set of "real-world" data [1] from 2998 cervical cell images. Each image has a 4-dimensional feature vector representing the following parameters of a biological cell:

1. log of (cytoplasm diameter/nucleus diameter)

[1]These data were graciously provided by Dr. Binay Bhattacharya, and had been used in computations presented in [Bha82]. They were originally obtained from the Biomedical Image Processing Laboratory at McGill University

2. log of nucleus area
3. average cytoplasm density
4. average nucleus density

The cells had also been classified into normal versus abnormal classes, so pattern recognition techniques could be applied [Bha82]. The order-3 Voronoi diagram of these data was computed, with the following results:

			r	$=$	216,416
n	$=$	2998	f	$=$	2,251,712
d	$=$	4	$\frac{f}{r}$	$=$	20.8
k	$=$	3	cpu time	$\approx$	75hours
			$\frac{t}{f}$	$\approx$	120ms

The cpu time given is only approximate, since the computer "crashed" several times during the run and some overhead was required to restart the program.

The following observations can be made on the size of V_k^S:

- $\frac{f}{r}$ is approximately constant, in fixed dimension, regardless of n and k. Some variability of this ratio is noted when $k = 1$ (and, to a lesser degree, when $k = 2$) but as k increases, the ratio tends to stabilize. For example, when $d = 4$, the ratio is approximately 20—for the random data as well as the cervical cell data.

- $\frac{f}{r}$ increases approximately quadratically with d

- the average number $\frac{f}{n}$ of facets per data point grows slowly with n. This increase is more pronounced at lower values of n, suggesting an effect at the periphery of the diagram. For example, when $k = 3$ and $d = 4$, using the random data from the hypercube distribution as well as the cervical cell data, we obtain the following. Note that the last line of the following table represents a different distribution than is represented in the remaining lines.

n	$\frac{f}{n}$
50	326
100	464
200	564
500	668
1000	722
2998	751

Figure 6.1 shows the growth of the ratio $\frac{t}{f}$ with increasing n, for $d = 2, \ldots, 6$. The ratios $\frac{t}{f}$ for all values of $k > 1$ in tables 6.1 through 6.10 are plotted against n, for each value of d. These plots demonstrate the running time of the program to be $O(n)$ per facet. The ratios $\frac{t}{fn}$ can be observed from the plots as:

d	$\frac{t}{fn}$
2	27 μs
3	25 μs
4	35 μs
5	60 μs
6	90 μs

The ratio $\frac{t}{fn}$ for the cervical cell data ($d = 4$) is approximately 40 μs. This agrees well with the above results—especially in light of the fact that some additional cpu time was required after the computer had "crashed".

Note that the ratio $\frac{t}{fn}$ is approximately the same in 2 or 3 dimensions. For $d \geq 3$, this ratio grows approximately quadratically with d. Hence, the running time of the implementation appears to be $O(d^2n)$ per facet.

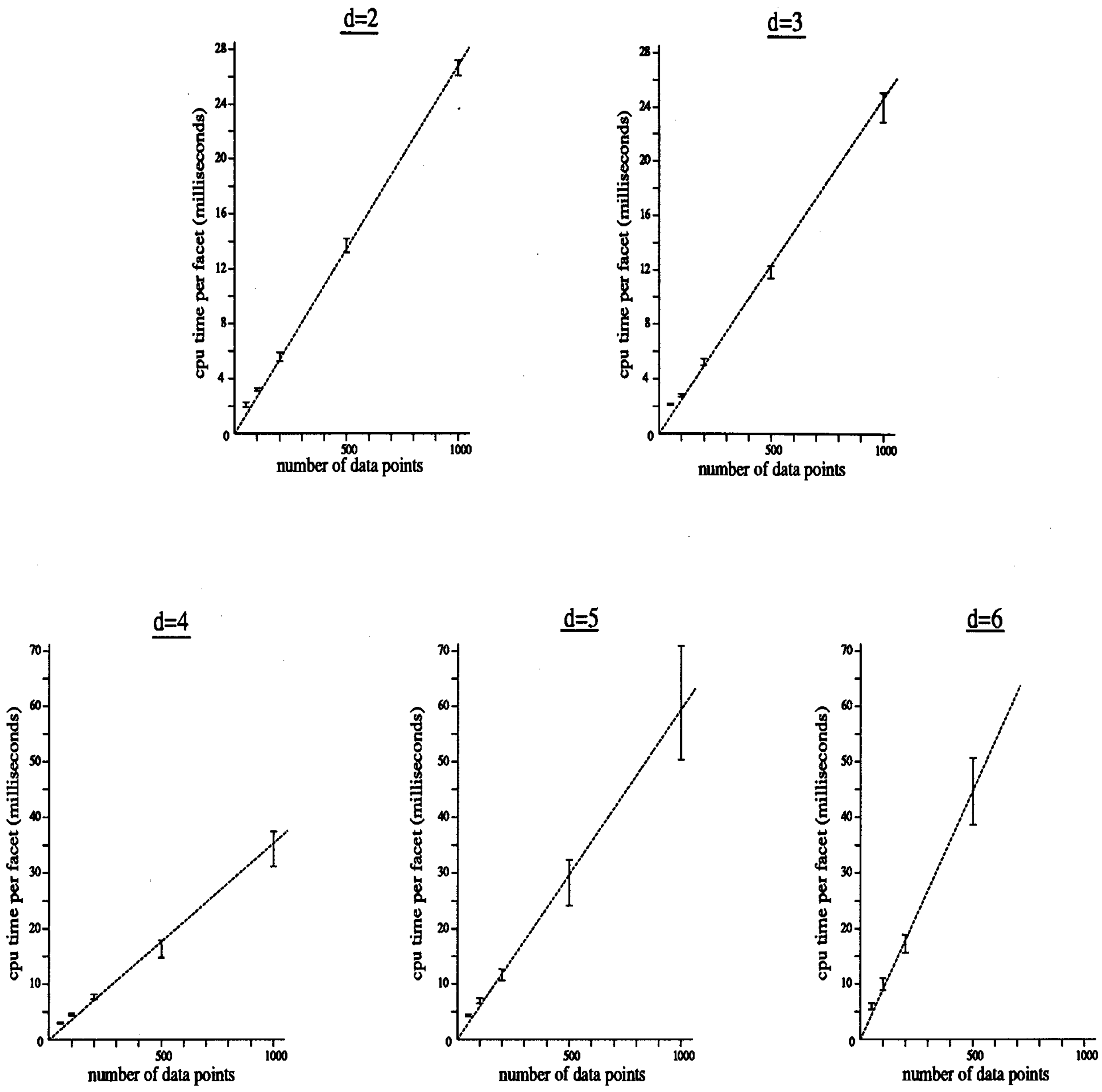

Figure 6.1: Plots of cpu Time Per Facet Versus $|S|$, for V_k^S with $2 \leq k \leq 5$

The ratios t/f for the computations presented in tables 6.1 through 6.10 are shown as ranges, for a given value of n. A separate plot is shown for each value of d. In each case, a linear function (dashed lines) approximates the growth of t/f with n.

Table 6.1: V_k^S of 50 Random Points From A Uniform Distribution In The Hypercube

$$\left[\begin{array}{ll} r = & \text{number of regions} \\ f = & \text{number of facets} \\ \multicolumn{2}{l}{\text{(cpu time in brackets)}} \\ t/f = & \text{cpu time per facet (in milliseconds)} \end{array}\right]$$

d		k				
		1	2	3	4	5
2	$r =$	50	139	217	284	348
	$f =$	139	397	622	822	1004
	$f/r =$	5.6	5.7	5.7	5.8	5.8
		(0.21 sec.)	(0.81 sec.)	(1.2 sec.)	(1.7 sec.)	(2.3 sec.)
	$t/f =$	1.5 ms	2.0 ms	1.9 ms	2.1 ms	2.3 ms
3	$r =$	50	293	715	1237	1881
	$f =$	293	1653	4032	7093	10744
	$f/r =$	11.7	11.3	11.3	11.5	11.4
		(0.44 sec.)	(3.6 sec.)	(8.4 sec.)	(15 sec.)	(23 sec.)
	$t/f =$	1.5 ms	2.2 ms	2.1 ms	2.1 ms	2.1 ms
4	$r =$	50	487	1739	4002	7346
	$f =$	487	4557	16283	37635	69506
	$f/r =$	19.5	18.7	18.7	18.8	18.9
		(0.98 sec.)	(14 sec.)	(48 sec.)	(1.9 min.)	(3.6 min.)
	$t/f =$	2.0 ms	3.1 ms	2.9 ms	3.0 ms	3.1 ms
5	$r =$	50	705	3617	10843	24313
	$f =$	705	10095	50902	152851	344054
	$f/r =$	28.2	28.6	28.1	28.2	28.3
		(2.2 sec.)	(43 sec.)	(3.5 min.)	(11 min.)	(26 min.)
	$t/f =$	3.1 ms	4.3 ms	4.1 ms	4.3 ms	4.5 ms
6	$r =$	50	902	6264	24006	64983
	$f =$	902	18126	123674	471828	1279164
	$f/r =$	36.1	40.2	39.5	39.3	39.4
		(4.1 sec.)	(1.7 min.)	(11 min.)	(45 min.)	(138 min.)
	$t/f =$	4.5 ms	5.6 ms	5.3 ms	5.7 ms	6.5 ms

Table 6.2: V_k^S of 100 Random Points From A Uniform Distribution In The Hypercube

$r =$	number of regions
$f =$	number of facets
(cpu time in brackets)	
$t/f =$	cpu time per facet (in milliseconds)

	k				
d	1	2	3	4	5
	$r =$ 100	$r =$ 285	$r =$ 465	$r =$ 632	$r =$ 788
2	$f =$ 285	$f =$ 837	$f =$ 1366	$f =$ 1858	$f =$ 2320
	$f/r =$ 5.7	$f/r =$ 5.9	$f/r =$ 5.9	$f/r =$ 5.9	$f/r =$ 5.9
	(0.64 sec.)	(2.7 sec.)	(4.4 sec.)	(6.0 sec.)	(7.7 sec.)
	$t/f =$ 2.2 ms	$t/f =$ 3.2 ms	$t/f =$ 3.2 ms	$t/f =$ 3.2 ms	$t/f =$ 3.3 ms
	$r =$ 100	$r =$ 646	$r =$ 1642	$r =$ 3074	$r =$ 4766
3	$f =$ 646	$f =$ 3776	$f =$ 9644	$f =$ 17931	$f =$ 27988
	$f/r =$ 12.9	$f/r =$ 11.7	$f/r =$ 11.7	$f/r =$ 11.7	$f/r =$ 11.7
	(1.6 sec.)	(12 sec.)	(30 sec.)	(56 sec.)	(90 sec.)
	$t/f =$ 2.5 ms	$t/f =$ 3.2 ms	$t/f =$ 3.1 ms	$t/f =$ 3.1 ms	$t/f =$ 3.2 ms
	$r =$ 100	$r =$ 1237	$r =$ 4731	$r =$ 11505	$r =$ 22094
4	$f =$ 1237	$f =$ 12437	$f =$ 46363	$f =$ 112058	$f =$ 215332
	$f/r =$ 24.7	$f/r =$ 20.1	$f/r =$ 19.6	$f/r =$ 19.5	$f/r =$ 19.5
	(4.5 sec.)	(60 sec.)	(3.4 min.)	(8.4 min.)	(17 min.)
	$t/f =$ 3.6 ms	$t/f =$ 4.8 ms	$t/f =$ 4.4 ms	$t/f =$ 4.5 ms	$t/f =$ 4.7 ms
	$r =$ 100	$r =$ 1913	$r =$ 10961	$r =$ 35501	$r =$ 85120
5	$f =$ 1913	$f =$ 30501	$f =$ 164103	$f =$ 523022	$f =$ 1248171
	$f/r =$ 38.3	$f/r =$ 31.9	$f/r =$ 29.9	$f/r =$ 29.5	$f/r =$ 29.3
	(12 sec.)	(3.7 min.)	(18 min.)	(57 min.)	(143 min.)
	$t/f =$ 6.3 ms	$t/f =$ 7.3 ms	$t/f =$ 6.6 ms	$t/f =$ 6.5 ms	$t/f =$ 6.9 ms
	$r =$ 100	$r =$ 2700	$r =$ 22143	$r =$ 93750	
6	$f =$ 2700	$f =$ 63659	$f =$ 476320	$f =$ 1956375	
	$f/r =$ 54.0	$f/r =$ 47.2	$f/r =$ 43.0	$f/r =$ 41.7	
	(24 sec.)	(10 min.)	(71 min.)	(308 min.)	
	$t/f =$ 8.9 ms	$t/f =$ 9.4 ms	$t/f =$ 8.9 ms	$t/f =$ 9.4 ms	

Table 6.3: V_k^S of 200 Random Points From A Uniform Distribution In The Hypercube

$r =$	number of regions
$f =$	number of facets
(cpu time in brackets)	
$t/f =$	cpu time per facet (in milliseconds)

d		k				
		1	2	3	4	5
2	$r =$	200	582	955	1312	1661
	$f =$	582	1721	2826	3891	4929
	$f/r =$	5.8	5.9	5.9	5.9	5.9
		(2.2 sec.)	(9.5 sec.)	(16 sec.)	(23 sec.)	(29 sec.)
	$t/f =$	3.8 ms	5.5 ms	5.7 ms	5.9 ms	5.9 ms
3	$r =$	200	1402	3594	6745	10812
	$f =$	1402	8330	21290	39926	63865
	$f/r =$	14.0	11.9	11.8	11.8	11.8
		(6.3 sec.)	(44 sec.)	(1.9 min.)	(3.5 min.)	(5.7 min.)
	$t/f =$	4.5 ms	5.3 ms	5.4 ms	5.3 ms	5.4 ms
4	$r =$	200	2785	11254	28406	56383
	$f =$	2785	29222	112765	281470	557253
	$f/r =$	27.9	21.0	20.0	19.8	19.8
		(19 sec.)	(3.9 min.)	(14 min.)	(35 min.)	(72 min.)
	$t/f =$	6.8 ms	8.0 ms	7.4 ms	7.5 ms	7.8 ms
5	$r =$	200	4865	29913	101141	250216
	$f =$	4865	82837	463646	1524971	3735625
	$f/r =$	48.6	34.1	31.0	30.2	29.9
		(60 sec.)	(17 min.)	(83 min.)	(287 min.)	(776 min.)
	$t/f =$	12.3 ms	12.3 ms	10.7 ms	11.3 ms	12.5 ms
6	$r =$	200	7252	65201		
	$f =$	7252	186689	1468604		
	$f/r =$	72.5	51.5	45.0		
		(2.4 min.)	(59 min.)	(382 min.)		
	$t/f =$	19.9 ms	19.0 ms	15.6 ms		

Table 6.4: V_k^S of 500 Random Points From A Uniform Distribution In The Hypercube

$r =$	number of regions
$f =$	number of facets
(cpu time in brackets)	
$t/f =$	cpu time per facet (in milliseconds)

d				k		
		1	2	3	4	5
2	$r =$	500	1480	2443	3395	4334
	$f =$	1480	4405	7285	10130	12934
	$f/r =$	5.9	6.0	6.0	6.0	6.0
		(14 sec.)	(59 sec.)	(96 sec.)	(2.3 min.)	(2.9 min.)
	$t/f =$	9.5 ms	13.4 ms	13.2 ms	13.6 ms	13.5 ms
3	$r =$	500	3611	9651	18403	29919
	$f =$	3611	21993	57792	110041	178265
	$f/r =$	11.9	12.2	12.0	12.0	11.9
		(39 sec.)	(4.2 min.)	(11 min.)	(21 min.)	(35 min.)
	$t/f =$	10.8 ms	11.5 ms	11.4 ms	11.5 ms	11.8 ms
4	$r =$	500	7708	32600	85058	174335
	$f =$	7708	83810	333811	856888	1743419
	$f/r =$	30.8	21.7	20.5	20.1	20.0
		(2.2 min.)	(25 min.)	(87 min.)	(211 min.)	(471 min.)
	$t/f =$	17.1 ms	17.9 ms	15.6 ms	14.8 ms	16.2 ms
5	$r =$	500	14509	94347		
	$f =$	14509	259111	1505103		
	$f/r =$	58.0	35.7	31.9		
		(8.6 min.)	(128 min.)	(604 min.)		
	$t/f =$	35.6 ms	29.6 ms	24.1 ms		
6	$r =$	500	24407			
	$f =$	24407	677467			
	$f/r =$	97.6	55.5			
		(26 min.)	(550 min.)			
	$t/f =$	63.9 ms	48.7 ms			

Table 6.5: V_k^S of 1000 Random Points From A Uniform Distribution In The Hypercube

$r =$	number of regions
$f =$	number of facets
(cpu time in brackets)	
$t/f =$	cpu time per facet (in milliseconds)

	k				
d	1	2	3	4	5
2	$r =$ 1000 $f =$ 2979 $f/r =$ 6.0 (59 sec.) $t/f =$ 19.8 ms	$r =$ 2979 $f =$ 8900 $f/r =$ 6.0 (3.9 min.) $t/f =$ 26.3 ms	$r =$ 4940 $f =$ 14771 $f/r =$ 6.0 (6.6 min.) $t/f =$ 26.8 ms	$r =$ 6887 $f =$ 20593 $f/r =$ 6.0 (9.3 min.) $t/f =$ 27.1 ms	$r =$ 8813 $f =$ 26357 $f/r =$ 6.0 (12 min.) $t/f =$ 27.3 ms
3	$r =$ 1000 $f =$ 7381 $f/r =$ 14.8 (3.0 min.) $t/f =$ 24.4 ms	$r =$ 7381 $f =$ 45301 $f/r =$ 12.3 (18 min.) $t/f =$ 23.8 ms	$r =$ 19919 $f =$ 120033 $f/r =$ 12.1 (46 min.) $t/f =$ 23.0 ms	$r =$ 38428 $f =$ 230415 $f/r =$ 12.0 (92 min.) $t/f =$ 24.0 ms	$r =$ 62642 $f =$ 375061 $f/r =$ 12.0 (148 min.) $t/f =$ 23.7 ms
4	$r =$ 1000 $f =$ 16286 $f/r =$ 32.6 (11 min.) $t/f =$ 40.5 ms	$r =$ 16286 $f =$ 179512 $f/r =$ 22.0 (106 min.) $t/f =$ 35.4 ms	$r =$ 69960 $f =$ 722280 $f/r =$ 20.6 (374 min.) $t/f =$ 31.1 ms		
5	$r =$ 1000 $f =$ 32187 $f/r =$ 64.4 (46 min.) $t/f =$ 85.7 ms	$r =$ 32187 $f =$ 588957 $f/r =$ 36.6 (636 min.) $t/f =$ 64.8 ms	$r =$ 214803 $f =$ 3469222 $f/r =$ 32.3 (2906 min.) $t/f =$ 50.3 ms		
6	$r =$ 1000 $f =$ 56682 $f/r =$ 113.4 (156 min.) $t/f =$ 165.1 ms				

Table 6.6: V_k^S of 50 Random Points From A Uniform Distribution In The Hypersphere

$$\left[\begin{array}{ll} r = & \text{number of regions} \\ f = & \text{number of facets} \\ \multicolumn{2}{l}{\text{(cpu time in brackets)}} \\ t/f = & \text{cpu time per facet (in milliseconds)} \end{array}\right]$$

	k				
d	1	2	3	4	5
	$r =$ 50	$r =$ 137	$r =$ 216	$r =$ 293	$r =$ 360
2	$f =$ 137	$f =$ 392	$f =$ 629	$f =$ 852	$f =$ 1051
	$f/r =$ 5.5	$f/r =$ 5.7	$f/r =$ 5.8	$f/r =$ 5.8	$f/r =$ 5.8
	(0.20 sec.)	(0.81 sec.)	(1.3 sec.)	(1.7 sec.)	(2.2 sec.)
	$t/f =$ 1.5 ms	$t/f =$ 2.1 ms	$t/f =$ 2.1 ms	$t/f =$ 2.0 ms	$t/f =$ 2.1 ms
	$r =$ 50	$r =$ 303	$r =$ 715	$r =$ 1263	$r =$ 1950
3	$f =$ 303	$f =$ 1701	$f =$ 4088	$f =$ 7302	$f =$ 11197
	$f/r =$ 12.1	$f/r =$ 11.2	$f/r =$ 11.4	$f/r =$ 11.6	$f/r =$ 11.5
	(0.47 sec.)	(3.6 sec.)	(8.5 sec.)	(15 sec.)	(24 sec.)
	$t/f =$ 1.6 ms	$t/f =$ 2.1 ms	$t/f =$ 2.1 ms	$t/f =$ 2.1 ms	$t/f =$ 2.1 ms
	$r =$ 50	$r =$ 502	$r =$ 1808	$r =$ 4220	$r =$ 7770
4	$f =$ 502	$f =$ 4758	$f =$ 17125	$f =$ 40072	$f =$ 73858
	$f/r =$ 20.1	$f/r =$ 19.0	$f/r =$ 18.9	$f/r =$ 19.0	$f/r =$ 19.0
	(1.1 sec.)	(14 sec.)	(50 sec.)	(2.0 min.)	(3.8 min.)
	$t/f =$ 2.2 ms	$t/f =$ 2.9 ms	$t/f =$ 2.9 ms	$t/f =$ 3.0 ms	$t/f =$ 3.1 ms
	$r =$ 50	$r =$ 706	$r =$ 3623	$r =$ 10967	$r =$ 24429
5	$f =$ 706	$f =$ 10081	$f =$ 51276	$f =$ 154590	$f =$ 345542
	$f/r =$ 28.2	$f/r =$ 28.6	$f/r =$ 28.3	$f/r =$ 28.2	$f/r =$ 28.3
	(2.4 sec.)	(43 sec.)	(3.5 min.)	(11 min.)	(25 min.)
	$t/f =$ 3.4 ms	$t/f =$ .43 ms	$t/f =$ 4.1 ms	$t/f =$ 4.3 ms	$t/f =$ 4.3 ms
	$r =$ 50	$r =$ 895	$r =$ 6258	$r =$ 24000	$r =$ 65147
6	$f =$ 895	$f =$ 18052	$f =$ 123742	$f =$ 473067	$f =$ 1286224
	$f/r =$ 35.8	$f/r =$ 40.3	$f/r =$ 39.5	$f/r =$ 39.4	$f/r =$ 39.5
	(3.9 sec.)	(1.7 min.)	(11 min.)	(46 min.)	(134 min.)
	$t/f =$ 4.4 ms	$t/f =$ 5.7 ms	$t/f =$ 5.3 ms	$t/f =$ 5.8 ms	$t/f =$ 6.3 ms

Table 6.7: V_k^S of 100 Random Points From A Uniform Distribution In The Hypersphere

$r =$	number of regions
$f =$	number of facets
(cpu time in brackets)	
$t/f =$	cpu time per facet (in milliseconds)

			k		
d	1	2	3	4	5
2	$r =$ 100	$r =$ 285	$r =$ 463	$r =$ 634	$r =$ 797
	$f =$ 285	$f =$ 835	$f =$ 1364	$f =$ 1871	$f =$ 2357
	$f/r =$ 5.7	$f/r =$ 5.9	$f/r =$ 5.9	$f/r =$ 5.9	$f/r =$ 5.9
	(0.62 sec.)	(2.6 sec.)	(4.2 sec.)	(6.0 sec.)	(7.6 sec.)
	$t/f =$ 2.2 ms	$t/f =$ 3.1 ms	$t/f =$ 3.1 ms	$t/f =$ 3.2 ms	$t/f =$ 3.2 ms
3	$r =$ 100	$r =$ 651	$r =$ 1674	$r =$ 3055	$r =$ 4820
	$f =$ 651	$f =$ 3834	$f =$ 9771	$f =$ 17958	$f =$ 28207
	$f/r =$ 13.0	$f/r =$ 11.8	$f/r =$ 11.7	$f/r =$ 11.8	$f/r =$ 11.7
	(1.7 sec.)	(12 sec.)	(30 sec.)	(59 sec.)	(90 sec.)
	$t/f =$ 2.6 ms	$t/f =$ 3.1 ms	$t/f =$ 3.1 ms	$t/f =$ 3.3 ms	$t/f =$ 3.2 ms
4	$r =$ 100	$r =$ 1233	$r =$ 4804	$r =$ 11700	$r =$ 22570
	$f =$ 1233	$f =$ 12532	$f =$ 47229	$f =$ 114323	$f =$ 220631
	$f/r =$ 24.7	$f/r =$ 20.3	$f/r =$ 19.7	$f/r =$ 19.5	$f/r =$ 19.6
	(4.5 sec.)	(60 sec.)	(3.6 min.)	(8.5 min.)	(17.4 min.)
	$t/f =$ 3.6 ms	$t/f =$ 4.8 ms	$t/f =$ 4.6 ms	$t/f =$ 4.5 ms	$t/f =$ 4.7 ms
5	$r =$ 100	$r =$ 1922	$r =$ 10943	$r =$ 35315	$r =$ 84420
	$f =$ 1922	$f =$ 30567	$f =$ 163942	$f =$ 520185	$f =$ 1237614
	$f/r =$ 38.4	$f/r =$ 31.8	$f/r =$ 30.0	$f/r =$ 29.5	$f/r =$ 29.3
	(12 sec.)	(3.6 min.)	(18.1 min.)	(56 min.)	(143 min.)
	$t/f =$ 6.2 ms	$t/f =$ 7.1 ms	$t/f =$ 6.6 ms	$t/f =$ 6.5 ms	$t/f =$ 6.9 ms
6	$r =$ 100	$r =$ 2635	$r =$ 21336	$r =$ 91079	$r =$ 268968
	$f =$ 2635	$f =$ 61386	$f =$ 460272	$f =$ 1902804	$f =$ 5553097
	$f/r =$ 52.7	$f/r =$ 46.6	$f/r =$ 43.1	$f/r =$ 41.8	$f/r =$ 41.3
	(24.3 sec.)	(10.5 min.)	(70.4 min.)	(305 min.)	(1019 min.)
	$t/f =$ 9.2 ms	$t/f =$ 10.3 ms	$t/f =$ 9.1 ms	$t/f =$ 9.6 ms	$t/f =$ 11.0 ms

Table 6.8: V_k^S of 200 Random Points From A Uniform Distribution In The Hypersphere

$r =$	number of regions
$f =$	number of facets
(cpu time in brackets)	
$t/f =$	cpu time per facet (in milliseconds)

				k		
d		1	2	3	4	5
2	$r =$	200	583	952	1318	1677
	$f =$	583	1720	2826	3919	4983
	$f/r =$	5.8	5.9	5.9	5.9	5.9
		(2.4 sec.)	(9.3 sec.)	(15 sec.)	(22 sec.)	(29 sec.)
	$t/f =$	4.1 ms	5.4 ms	5.3 ms	5.6 ms	5.8 ms
3	$r =$	200	1375	3585	6892	10912
	$f =$	1375	8219	21425	40723	64867
	$f/r =$	13.8	12.0	12.0	11.8	11.9
		(6.3 sec.)	(45 sec.)	(1.8 min.)	(3.4 min.)	(5.6 min.)
	$t/f =$	4.6 ms	5.5 ms	5.0 ms	5.0 ms	5.2 ms
4	$r =$	200	2746	11275	28632	57198
	$f =$	2746	28971	113368	284668	566682
	$f/r =$	27.5	21.1	20.1	19.9	19.8
		(19 sec.)	(3.9 min.)	(13.6 min.)	(34 min.)	(70 min.)
	$t/f =$	6.9 ms	8.1 ms	7.2 ms	7.2 ms	7.4 ms
5	$r =$	200	4766	29327	99699	248588
	$f =$	4766	81179	456324	1509028	3722081
	$f/r =$	47.7	34.1	31.1	30.3	29.9
		(60 sec.)	(17 min.)	(81 min.)	(267 min.)	(744 min.)
	$t/f =$	12.6 ms	12.6 ms	10.7 ms	10.6 ms	12.0 ms
6	$r =$	200	7148	64089	288106	
	$f =$	7148	183563	1445720	6209881	
	$f/r =$	71.5	51.4	45.1	43.1	
		(2.4 min.)	(57 min.)	(389 min.)	(1702 min.)	
	$t/f =$	20.1 ms	18.6 ms	16.1 ms	16.4 ms	

Table 6.9: V_k^S of 500 Random Points From A Uniform Distribution In The Hypersphere

$r =$	number of regions
$f =$	number of facets
(cpu time in brackets)	
$t/f =$	cpu time per facet (in milliseconds)

d		k				
		1	2	3	4	5
2	$r =$	500	1482	2454	3418	4371
	$f =$	1482	4420	7330	10213	13060
	$f/r =$	5.9	6.0	6.0	6.0	6.0
		(15 sec.)	(59 sec.)	(98 sec.)	(2.3 min.)	(3.1 min.)
	$t/f =$	10.1 ms	13.3 ms	13.4 ms	13.5 ms	14.2 ms
3	$r =$	500	3654	9746	18597	30211
	$f =$	3654	22284	58475	111343	180532
	$f/r =$	14.6	12.2	12.0	12.0	12.0
		(42 sec.)	(4.4 min.)	(12 min.)	(22 min.)	(36 min.)
	$t/f =$	11.5 ms	11.8 ms	12.3 ms	11.9 ms	12.0 ms
4	$r =$	500	7810	32890	85846	
	$f =$	7810	84940	337286	865789	
	$f/r =$	31.2	21.8	20.5	20.2	
		(2.3 min.)	(25 min.)	(92 min.)	(217 min.)	
	$t/f =$	17.7 ms	17.7 ms	16.4 ms	15.0 ms	
5	$r =$	500	14365	93249		
	$f =$	14365	255919	1489394		
	$f/r =$	57.5	35.6	31.9		
		(9.2 min.)	(138 min.)	(612 min.)		
	$t/f =$	38.4 ms	32.4 ms	24.7 ms		
6	$r =$	500	23752	229257		
	$f =$	23752	653831	5364437		
	$f/r =$	95.0	55.1	46.8		
		(26 min.)	(551 min.)	(3459 min.)		
	$t/f =$	65.7 ms	50.6 ms	38.7 ms		

Table 6.10: V_k^S of 1000 Random Points From A Uniform Distribution In The Hypersphere

$r =$	number of regions
$f =$	number of facets
(cpu time in brackets)	
$t/f =$	cpu time per facet (in milliseconds)

d		k				
		1	2	3	4	5
2	$r =$	1000	2982	4950	6907	8857
	$f =$	2982	8916	14811	20677	26521
	$f/r =$	6.0	6.0	6.0	6.0	6.0
		(59 sec.)	(3.9 min.)	(6.6 min.)	(9.4 min.)	(12 min.)
	$t/f =$	19.8 ms	26.2 ms	26.7 ms	27.3 ms	27.1 ms
3	$r =$	1000	7480	20100	38854	63331
	$f =$	7480	45926	121412	233208	379890
	$f/r =$	15.0	12.3	12.1	12.0	12.0
		(3.3 min.)	(19 min.)	(47 min.)	(98 min.)	(151 min.)
	$t/f =$	26.5 ms	24.8 ms	23.2 ms	25.2 ms	23.8 ms
4	$r =$	1000	16371	70712		
	$f =$	16371	181108	730914		
	$f/r =$	32.7	22.1	20.7		
		(11 min.)	(113 min.)	(408 min.)		
	$t/f =$	40.3 ms	37.4 ms	33.5 ms		
5	$r =$	1000	31932	213105		
	$f =$	31932	583723	3448392		
	$f/r =$	63.9	36.6	32.4		
		(52 min.)	(690 min.)	(3017 min.)		
	$t/f =$	97.7 ms	70.9 ms	52.5 ms		
6	$r =$	1000				
	$f =$	55493				
	$f/r =$	111.0				
		(164 min.)				
	$t/f =$	177.3 ms				

6.3 Analysis of the Efficiency of Algorithm Pivot

To analyze the efficiency of algorithm **Pivot**, a series of extreme point problems in $\Re^d$ were solved. By duality (see lemma 2.3) this is equivalent to the problem of determining nonredundancy among a system of constraints. The extreme points problems were solved on sets of n points randomly chosen from uniform distributions in the interior of the unit hypercube (tables 6.11 and 6.12), and in the interior of the unit hypersphere (tables 6.13 and 6.14). The value of n ranged from 100 to 1000, and the dimension d ranged from 3 to 10.

A counter was kept, during execution of algorithm **Pivot**, to count the number of pivots performed on the dual polytope (the intersection of n constraints). After the algorithm had terminated, the total number of polytope vertices was determined, using an implementation of Avis and Fukuda's technique [AF90], discussed in section 5.5.

The data in tables 6.11, 6.12, 6.13 and 6.13 indicates the number e of extreme points (or nonredundant constraints in the dual picture), the total number t of convex hull facets (or polytope vertices in the dual picture), and the number v of convex hull facets visited by the algorithm—also expressed as a percentage of the total. The data shown in each square of the tables (for some given values of n and d) represent the results of a single execution of the algorithm. The smaller problems were performed several times on different sets of data, and the results always agreed within about 10%.

The data for the hypersphere distribution are very similar to those for the hypercube distribution. However, the following observations can be made: [2]

- In lower dimensional spaces, the number e of extreme points, and the number t of convex hull facets, in C are greater than the corresponding numbers in S. However, as d increases, the growth of both t and e is more rapid in S than it is in C. So, in higher dimensional spaces, the numbers of extreme points and of convex hull facets are greater in S that in C.

- The ratio $\frac{v}{t}$ is greater for S in lower dimensions, but is greater for C in higher dimensions. This follows the general pattern of $\frac{v}{t}$ decreasing as t increases.

The expected values $E(t)$ and $E(e)$ for convex hull facets and extreme points of random distributions, have been reported in the literature, for fixed d. Both $E(t)$ and $E(e)$ are

[2]For ease of expression, "C" will refer to the uniform distribution in the interior of the hypercube, and "S" will refer to the uniform distribution in the hypersphere.

$\Theta(\log^{d-1} n)$ for a uniform distribution in any polytope (such as distribution C) [Dwy90]. For a uniform distribution in the interior of a hypersphere (*i.e.* distribution S), the expected number has been reported as $\Theta(n^{\frac{d-1}{d+1}})$ [Ray70] The ratio between the upper and lower bounds, however, is exponential in d.

Note that the number t of convex hull facets grows very quickly with d; a cursory examination of the data reveals that t grow by about 500% with each increase of 1 dimension. The number v of convex hull facets visited by the algorithm grows much more slowly with d. The key observation here is that the ratio $\frac{v}{t}$ of convex hull facets (dually, vertices) visited drops by an order of magnitude, with each increase of 1 dimension. In lower dimensional spaces, the ratio increases slowly with n. As the dimension of the problem grows, this increase with n becomes less significant; $\frac{v}{t}$ appears to stabilize for any value of n—in fact, a drop in the ratio is observed from the smallest problem ($n = 100$) to the second smallest problem ($n = 200$) for $d \geq 8$.

The number of facets in the convex hull of n points may be as high as $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ [McM70]. If algorithm **Pivot** were to visit a large percentage of these facets, its time complexity could become very large, as d increases. Fortunately, as shown by the results of this section, the percentage of facets visited decreases markedly with increasing d—for point sets chosen from uniform distributions in the interior of a hypersphere or a hypercube. As a result, the actual number of facets visited grows only moderately with d. This makes the algorithm practical for problems in higher dimensional spaces, unlike Megiddo's modified $O(3^{d^2} n)$ approach [Meg84] [Dye86] [Cla86].

Table 6.11: Performance of Algorithm **Pivot** in Determining Extreme Points For Uniform Distribution in Hypercube

e =number of extreme points
t =total number of convex hull facets
v =number of facets visited
(bracketted number indicates percentage of facets visitted)

d	Number of constraints (n)									
	100		200		300		400		500	
3	$e =$	29	$e =$	40	$e =$	42	$e =$	52	$e =$	55
	$t =$	54	$t =$	76	$t =$	80	$t =$	100	$t =$	106
	$v =$	47	$v =$	69	$v =$	70	$v =$	90	$v =$	98
		(87%)		(91%)		(87%)		(90%)		(92%)
4	$e =$	52	$e =$	79	$e =$	96	$e =$	104	$e =$	113
	$t =$	261	$t =$	403	$t =$	471	$t =$	520	$t =$	576
	$v =$	120	$v =$	200	$v =$	258	$v =$	320	$v =$	352
		(46%)		(50%)		(55%)		(62%)		(61%)
5	$e =$	73	$e =$	122	$e =$	159	$e =$	188	$e =$	212
	$t =$	1052	$t =$	2038	$t =$	2706	$t =$	3286	$t =$	3822
	$v =$	172	$v =$	331	$v =$	548	$v =$	738	$v =$	887
		(16%)		(16%)		(20%)		(22%)		(23%)
6	$e =$	88	$e =$	157	$e =$	216	$e =$	264	$e =$	305
	$t =$	4534	$t =$	10272	$t =$	15276	$t =$	19337	$t =$	22716
	$v =$	230	$v =$	463	$v =$	747	$v =$	1129	$v =$	1375
		(5.1%)		(4.5%)		(4.9%)		(5.8%)		(6.1%)
7	$e =$	92	$e =$	176	$e =$	256	$e =$	323	$e =$	379
	$t =$	16622	$t =$	45890	$t =$	73428	$t =$	99930	$t =$	124760
	$v =$	241	$v =$	575	$v =$	993	$v =$	1327	$v =$	1890
		(1.4%)		(1.3%)		(1.4%)		(1.3%)		(1.5%)
8	$e =$	97	$e =$	190	$e =$	279	$e =$	356	$e =$	431
	$t =$	62775	$t =$	197522	$t =$	354707	$t =$	521582	$t =$	681223
	$v =$	277	$v =$	594	$v =$	1147	$v =$	1578	$v =$	2033
		(0.44%)		(0.30%)		(0.32%)		(0.30%)		(0.30%)
9	$e =$	98	$e =$	196	$e =$	287	$e =$	375		
	$t =$	209160	$t =$	777410	$t =$	1540600	$t =$	2389160		
	$v =$	292	$v =$	728	$v =$	1184	$v =$	1710		
		(0.14%)		(0.09%)		(0.08%)		(0.07%)		
10	$e =$	100	$e =$	199	$e =$	297				
	$t =$	762621	$t =$	3350856	$t =$	7539609				
	$v =$	294	$v =$	709	$v =$	1122				
		(0.039%)		(0.021%)		(0.015%)				

Table 6.12: Performance of Algorithm **Pivot** in Determining Extreme Points For Uniform Distribution in Hypercube

e =number of extreme points
t =total number of convex hull facets
v =number of facets visited
(bracketted number indicates percentage of facets visitted)

d	Number of constraints (n)				
	600	700	800	900	1000
	$e =$ 60	$e =$ 66	$e =$ 67	$e =$ 73	$e =$ 75
3	$t =$ 116	$t =$ 128	$t =$ 130	$t =$ 142	$t =$ 146
	$v =$ 108	$v =$ 116	$v =$ 121	$v =$ 132	$v =$ 143
	(93%)	(91%)	(93%)	(93%)	(98%)
	$e =$ 121	$e =$ 140	$e =$ 145	$e =$ 149	$e =$ 160
4	$t =$ 623	$t =$ 738	$t =$ 774	$t =$ 805	$t =$ 858
	$v =$ 411	$v =$ 486	$v =$ 521	$v =$ 543	$v =$ 588
	(66%)	(66%)	(67%)	(67%)	(69%)
	$e =$ 237	$e =$ 269	$e =$ 273	$e =$ 291	$e =$ 314
5	$t =$ 4152	$t =$ 4950	$t =$ 5042	$t =$ 5426	$t =$ 5954
	$v =$ 1050	$v =$ 1251	$v =$ 1345	$v =$ 1490	$v =$ 1609
	(25%)	(25%)	(27%)	(27%)	(27%)
	$e =$ 344	$e =$ 389	$e =$ 423	$e =$ 449	$e =$ 489
6	$t =$ 25273	$t =$ 29682	$t =$ 31875	$t =$ 34412	$t =$ 38549
	$v =$ 1652	$v =$ 2025	$v =$ 2315	$v =$ 2564	$v =$ 2977
	(6.5%)	(6.8%)	(7.2%)	(7.4%)	(7.7%)
	$e =$ 424	$e =$ 486	$e =$ 541	$e =$ 593	$e =$ 647
7	$t =$ 143008	$t =$ 171622	$t =$ 194250	$t =$ 216326	$t =$ 240308
	$v =$ 2176	$v =$ 2749	$v =$ 3216	$v =$ 3541	$v =$ 3993
	(1.5%)	(1.6%)	(1.7%)	(1.6%)	(1.7%)
	$e =$ 514	$e =$ 593	$e =$ 658	$e =$ 726	$e =$ 783
8	$t =$ 826378	$t =$ 939616	$t =$ 1052651	$t =$ 1259329	$t =$ 1406114
	$v =$ 2671	$v =$ 3087	$v =$ 3699	$v =$ 4261	$v =$ 4939
	(0.32%)	(0.33%)	(0.35%)	(0.34%)	(0.35%)

Table 6.13: Performance of Algorithm **Pivot** in Determining Extreme Points For Uniform Distribution in Hypersphere

e =number of extreme points
t =total number of convex hull facets
v =number of facets visited
(bracketted number indicates percentage of facets visitted)

d		Number of constraints (n)				
		100	200	300	400	500
3	$e =$	22	35	38	39	43
	$t =$	40	66	72	74	82
	$v =$	36	60	67	71	79
		(90%)	(91%)	(93%)	(96%)	(96%)
4	$e =$	43	62	77	92	105
	$t =$	191	291	387	461	540
	$v =$	91	174	232	293	349
		(48%)	(60%)	(60%)	(64%)	(65%)
5	$e =$	64	104	146	163	189
	$t =$	924	1560	2332	2712	3298
	$v =$	161	357	525	642	879
		(17%)	(23%)	(23%)	(24%)	(27%)
6	$e =$	82	143	199	253	290
	$t =$	3970	8530	12669	16978	21118
	$v =$	200	535	820	1121	1380
		(5.0%)	(6.3%)	(6.5%)	(6.6%)	(6.5%)
7	$e =$	93	174	247	319	389
	$t =$	16456	43726	70090	97564	127484
	$v =$	231	561	936	1286	1710
		(1.4%)	(1.3%)	(1.3%)	(1.3%)	(1.3%)
8	$e =$	99	192	280	368	453
	$t =$	62942	211619	371841	546307	731627
	$v =$	211	579	951	1400	1828
		(0.34%)	(0.27%)	(0.26%)	(0.26%)	(0.25%)
9	$e =$	99	199	294	389	
	$t =$	229618	929170	1863012	2939822	
	$v =$	302	656	1063	1511	
		(0.13%)	(0.07%)	(0.06%)	(0.05%)	
10	$e =$	100	200	300		
	$t =$	855599	4216847	9613703		
	$v =$	289	687	1226		
		(0.034%)	(0.016%)	(0.013%)		

Table 6.14: Performance of Algorithm **Pivot** in Determining Extreme Points For Uniform Distribution in Hypersphere

e =number of extreme points
t =total number of convex hull facets
v =number of facets visited
(bracketted number indicates percentage of facets visitted)

d	Number of constraints (n)				
	600	700	800	900	1000
	$e=$ 47	$e=$ 50	$e=$ 47	$e=$ 49	$e=$ 51
3	$t=$ 90	$t=$ 96	$t=$ 90	$t=$ 94	$t=$ 98
	$v=$ 89	$v=$ 95	$v=$ 90	$v=$ 93	$v=$ 97
	(99%)	(99%)	(100%)	(99%)	(99%)
	$e=$ 116	$e=$ 120	$e=$ 122	$e=$ 130	$e=$ 139
4	$t=$ 599	$t=$ 615	$t=$ 619	$t=$ 663	$t=$ 713
	$v=$ 420	$v=$ 438	$v=$ 466	$v=$ 497	$v=$ 552
	(70%)	(71%)	(75%)	(75%)	(77%)
	$e=$ 208	$e=$ 226	$e=$ 237	$e=$ 255	$e=$ 272
5	$t=$ 3624	$t=$ 4058	$t=$ 4344	$t=$ 4674	$t=$ 4970
	$v=$ 985	$v=$ 1184	$v=$ 1354	$v=$ 1509	$v=$ 1607
	(27%)	(29%)	(31%)	(32%)	(32%)
	$e=$ 333	$e=$ 372	$e=$ 397	$e=$ 426	$e=$ 462
6	$t=$ 24108	$t=$ 28056	$t=$ 30038	$t=$ 33398	$t=$ 36553
	$v=$ 1618	$v=$ 1984	$v=$ 2323	$v=$ 2676	$v=$ 3025
	(6.7%)	(7.1%)	(7.7%)	(8.0%)	(8.3%)
	$e=$ 449	$e=$ 508	$e=$ 555	$e=$ 613	$e=$ 660
7	$t=$ 150782	$t=$ 176506	$t=$ 193826	$t=$ 222360	$t=$ 245540
	$v=$ 2142	$v=$ 2520	$v=$ 3006	$v=$ 3564	$v=$ 4027
	(1.4%)	(1.4%)	(1.6%)	(1.6%)	(1.6%)
	$e=$ 518	$e=$ 594	$e=$ 660	$e=$ 739	$e=$ 815
8	$t=$ 910542	$t=$ 1103088	$t=$ 1244239	$t=$ 1477395	$t=$ 1668428
	$v=$ 2308	$v=$ 2921	$v=$ 3279	$v=$ 4033	$v=$ 4437
	(0.25%)	(0.26%)	(0.26%)	(0.27%)	(0.27%)

Chapter 7

Conclusion

7.1 Results

The main results of this thesis are as follows:

1. The first known algorithm to *directly* compute all vertices of an order-k Voronoi Diagram in $\Re^d$ is presented. This algorithm has time complexity $O(d^2n + d^3 \log n)$ per vertex, and space complexity $O(d)$ per vertex.

2. A second algorithm is presented which *directly* computes only the facets of an order-k Voronoi Diagram in $\Re^d$. In fixed dimension, the time complexity of the algorithm can be bounded from above by $O(nd + kd \log n)$ per facet. However, the complexity bound has a high dimension-dependent constant and, hence, is not practical.

3. A new technique is developed for determining the nonredundant constraints among a system of constraints in $\Re^d$. This approach is based on the revised simplex method of linear programming.

4. A practical version of the facet enumeration algorithm—item (2) above—is developed using the technique of item (3) to determine nonredundancy among a system of constraints. A time complexity which is output-sensitive in the worst case cannot be derived for this algorithm. However, experimental results are shown which show that the time complexity per facet grows with d^2n, approximately.

5. An analysis of the complexity of order-k Voronoi Diagrams of randomly generated point sets—supported by computational evidence—is given for small values of k ($1 \leq$

$k \leq 5$), in 2 through 6 dimensional spaces.

7.2 Open Problems

Several open problems related to this work are:

1. To find a tighter bound on the number of Voronoi polytopes in V_k^S.

   The bound of $\Theta(k^{\lceil \frac{d}{2} \rceil} n^{\lfloor \frac{d}{2} \rfloor})$ is tight for the number of k-sets of $S \subset \Re^d$, as $k \to \infty$ and $\frac{n}{k} \to \infty$; the bound is achieved by the vertices of cyclic polytopes [CS89]. Since spherical separability of $T \subset \Re^d$ and $S \subset \Re^d$ is equivalent to linear separability of the paraboloid transformations of these point sets, the upper bound of $O(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor})$ certainly holds for the number of polytopes in V_k^S. However, this bound is probably not tight. Certainly for low values of k ($k < \left\lfloor \frac{d+1}{2} \right\rfloor$), the trivial upper bound of $\binom{n}{k}$ is better than the $O(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor})$ bound.

2. To find a tight bound on the number of V-vertices in V_k^S.

3. Is it possible to enumerate the V-vertices without using additional storage space?

   The algorithm of Avis and Fukuda [AF90] searches through all of the vertices in the intersection of m halfspaces in $\Re^d$, in $O(dm)$ time per vertex. No additional storage is needed for intermediate vertices in the search, since a fixed search order is followed from one vertex to the next. Can this approach be extended to provide a fixed search order among all of the V-vertices in V_k^S?

4. Can the V-vertices (or the V-facets) of V_k^S be enumerated at a logarithmic cost per vertex?

   The v facets of the convex hull of m points in $\Re^d$ (or, dually, the v vertices of the intersection of m halfspaces in $\Re^d$) can be enumerated by a shelling technique [Sei86] in worst-case $O(m^2 + v \log m)$ time. Using this technique, the V-vertices of a single Voronoi polytope $V_k^S(T)$ can be enumerated in $O(n^2k^2 + v \log n)$ time. However, each V-vertex can lie in many Voronoi polytopes. In the nondegenerate situation, a V-vertex can lie in as many as $\binom{d+1}{t}$ polytopes, where $t = \min\{\frac{d+1}{2}, k\}$.

Bibliography

[AB83] David Avis and Binay K. Bhattacharya. Algorithms for computing d-dimensional Voronoi diagrams and their duals. *Advances in Computing Research*, 1:159–180, 1983.

[AF90] David Avis and Komei Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. Technical report, McGill University, 1990.

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1974.

[AHU83] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures And Algorithms.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1983.

[AS80] Bengt Aspvall and Richard E. Stone. Khachiyan's linear programming algorithm. *Journal of Algorithms*, 1:1–13, 1980.

[Aur90] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. Technical report, Institut für Informatik, Fachbereich Mathematik, Freie Universität Berlin, 1990.

[BDT90] Jean-Daniel Boissonnat, Oliver Devillers, and Monique Teillaud. A semi-dynamic construction of higher order Voronoi diagrams and its randomized analysis. Technical Report 1207, INRIA, 1990.

[BFP$^+$72] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1972.

[BGT81] Robert G. Bland, Donald Goldfarb, and Michael J. Todd. The ellipsoid method: a survey. *Operations Research*, 29(6):1039–1091, 1981.

[Bha82] Binay K. Bhattacharya. *Application of Computational Geometry to Pattern Recognition Problems.* PhD thesis, McGill University, Montreal, Quebec, 1982.

[Bha83] Binay K. Bhattacharya. An algorithm for computing order k Voronoi diagrams in the plane. Technical Report TR 83-9, Computing Science Department, Simon Fraser University, 1983.

[Bla77] Robert G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107, 1977.

[Boo62] J.C.G. Boot. On trivial and binding constraints in programming problems. *Management Science*, 8(4):419–441, 1962.

[CE85] Bernard Chazelle and Herbert Edelsbrunner. An improved algorithm for constructing k^{th}-order Voronoi diagrams. In *Proceedings of the* 1^{st} *Annual Symposium on Computational Geometry*, pages 228–234, Baltimore, June 1985. ACM.

[Chv83] Vašek Chvátal. *Linear Programming.* W.H. Freeman & Company, New York, 1983.

[CK70] Donald R. Chand and Sham S. Kapur. An algorithm for convex polytopes. *Journal of the ACM*, 17:78–86, 1970.

[Cla86] Kenneth L. Clarkson. Linear programming in $O(n \times 3^{d^2})$ time. *Information Processing Letters*, 22:21–24, 1986.

[Cla87] Kenneth L. Clarkson. New applications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2:195–222, 1987.

[CS89] Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4:387–421, 1989.

[Dan51] George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Tjalling C. Koopmans, editor, *Activity Analysis of Production and Allocation*, number 13 in Cowles Commission for Research in Economics

Monograph, pages 339–347, University of Chicago, 1951. Princeton University Press.

[Dan63] George B. Dantzig. *Linear Programming And Extensions.* Princeton University Press, Princeton, New Jersey, 1963.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, Inc., New York, 1973.

[DOH54] George B. Dantzig and W. Orchard-Hayes. The product form of the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*, 8:64–67, 1954.

[Dwy90] Rex.A. Dwyer. Kinder, gentler average-case analysis for convex hulls and maximal vectors. *SIGACT News*, 21(2):64–71, 1990.

[Dye86] M.E. Dyer. On a multidimensional search technique and its application to the euclidean one-centre problem. *SIAM Journal on Computing*, 15(3):725–738, 1986.

[EM88] Herbert Edelsbrunner and Ernst Peter Mücke. A technique to cope with degenerate cases in geometric algorithms. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pages 118–133, Urbana-Champaign, Illinois, June 1988. ACM.

[EOS86] Herbert Edelsbrunner, J. O'Rourke, and Raimund Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15:341–363, 1986.

[ES86] Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1:25–44, 1986.

[Gal83] Tomas Gal. A method for determining redundant constraints. In M. Beckmann and W. Krelle, editors, *Redundancy in Mathematical Programming: A State-of-the-Art Survey*, volume 206 of *Lecture Notes in Economics and Mathematical Systems*, chapter 4. Springer-Verlag, Berlin, 1983.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[Gon89] C.C. Gonzaga. Conical projection algorithms for linear programming. *Mathematical Programming*, 43:151–173, 1989.

[Grü67] Branko Grünbaum. *Convex Polytopes*. Interscience Publishers: A division of John Wiley & Sons, Inc., London, 1967.

[IMS87] IMSL, Houston, Texas. *Math/Library User's Manual*, 1.0 edition, April 1987.

[IN77] D. B. Iudin and A. S. Nemiroviskii. Informational complexity and effective methods of solution for convex extremal problems. *Matekon: Translations of Russian and East European Mathematical Economics*, 13(3):25–45, 1977.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akad. Nauk. SSSR*, 224(5):1093–1096, 1979.

[KLTZ83] Mark H. Karwan, Vahid Lotfi, Jan Telgen, and Stanley Zionts. *Redundancy in Mathematical Programming: A State-of-the-Art Survey*, volume 206 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, 1983.

[KM72] V. Klee and G.J. Minty. How good is the simplex algorithm? In O.Shisha, editor, *Inequalities-III*, pages 159–175. Academic Press, New York, 1972.

[LD65] D.O. Loftsgaarden and D.P.Quesenbery. A nonparametric density function. *Annals of Mathematical Statistics*, 36:1049–1051, 1965.

[Lee82] Der-Tsai Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Transactions on Computers*, C-31(6):478–487, 1982.

[Lue84] David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, Reading, Massachusetts, second edition, 1984.

[Mat73] Theodore H. Mattheiss. An algorithm for determining irrelevant constraints and all vertices in systems of linear inequalities. *Operations Research*, 21(1):247–260, 1973.

[Mat83] Theodore H. Mattheiss. A method for finding redundant constraints of a system of linear inequalities. In M. Beckmann and W. Krelle, editors, *Redundancy in Mathematical Programming: A State-of-the-Art Survey*, volume 206 of *Lecture Notes in Economics and Mathematical Systems*, chapter 7. Springer-Verlag, Berlin, 1983.

[McM70] P. McMullen. The maximum numbers of faces of a convex polytope. *Mathematika*, 17:179–184, 1970.

[Meg84] Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.

[MS80] Theodore H. Mattheiss and B.K. Schmidt. Computational results on an algorithm for finding all vertices of a polytope. *Mathematical Programming*, 18:308–329, 1980.

[Mul89] Ketan Mulmuley. On obstructions in relation to a fixed viewpoint. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 592–597, 1989.

[Mul90] Ketan Mulmuley. Output sensitive construction of levels and Voronoi diagrams in $\Re^d$ of order 1 to k. In *Proceedings of the* 22^{nd} *Annual ACM Symposium on the Theory of Computing*, pages 322–330, Seattle, 1990.

[Num91] Numerical Algorithms Group, Limited, Oxford. *Fortran Library Manual, Mark 15*, first edition, June 1991.

[OKM86] Joseph O'Rourke, S. Rao Kosaraju, and Nimrod Megiddo. Computing circular separability. *Discrete & Computational Geometry*, 1:105–113, 1986.

[Ray70] H. Raynaud. Sur l'enveloppe convexe des nuages de points aleatoires dans $\Re^n$. I. *Journal of Applied Probability*, 7:35–48, 1970.

[Rub83] David S. Rubin. Finding redundant constraints in sets of linear inequalities. In M. Beckmann and W. Krelle, editors, *Redundancy in Mathematical Programming: A State-of-the-Art Survey*, volume 206 of *Lecture Notes in Economics and Mathematical Systems*, chapter 6. Springer-Verlag, Berlin, 1983.

[Sei86] Raimund Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proceedings of the* 18nd *Annual ACM Symposium on the Theory of Computing*, pages 404–413, Berkeley, 1986.

[SH75] M.I. Shamos and D. Hoey. Closest-point problems. In *IEEE Symposium on Foundations of Computer Science*, pages 151–162, October 1975.

[Tel83] Jan Telgen. Identifying redundancy in systems of linear constraints. In M. Beckmann and W. Krelle, editors, *Redundancy in Mathematical Programming: A State-of-the-Art Survey*, volume 206 of *Lecture Notes in Economics and Mathematical Systems*, chapter 5. Springer-Verlag, Berlin, 1983.

[Yap88] Chee-Keng Yap. A geometric consistency theorem for a symbolic perturbation scheme. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pages 134–142, Urbana-Champaign, Illinois, June 1988. Association for Computing Machinery.

[Zio80] Stanley Zionts. Identifying efficient vectors: some theory and computational results. *Operations Research*, 28(3):785–793, 1980.

[ZW83] Stanley Zionts and Jyrki Wallenius. A method for identifying redundant constraints and extraneous variables in linear programs. In M. Beckmann and W. Krelle, editors, *Redundancy in Mathematical Programming: A State-of-the-Art Survey*, volume 206 of *Lecture Notes in Economics and Mathematical Systems*, chapter 3. Springer-Verlag, Berlin, 1983.