Algorithms for the frame of a finitely generated unbounded polyhedron.

J.H. Dulá, The University of Mississippi, University, MS 38677, and
F.J. López, The University of Texas at El Paso, El Paso, TX 79968

February 2004

ABSTRACT. Consider two finite sets, \( A \) and \( V \) of points in \( m \)-dimensional space. The convex hull of \( A \) and the conical hull of \( V \) can be combined to create a finitely generated unbounded polyhedron. We explore the geometry of these polyhedral sets to design, implement, test, and compare two different algorithms for finding the frame, a minimal cardinality subset of \( A \) and \( V \), that generate the same polyhedron. One algorithm is a naive approach based on the direct application of the definition of these sets. The second algorithm is based on different principles erecting the frame geometrically one element at a time. Testing indicates that the second algorithm is faster with the difference becoming increasingly dramatic as the cardinality of the sets \( A \) and \( V \) increases and frame density decreases.

Key Words: Linear Programming, Extreme Points and Rays, Convex Analysis.

Introduction. The problem of identifying the extreme points of a polyhedron defined as the convex hull of a finite set of points in multidimensional space is known by various names. Fukuda (2002) refers to it as the “redundancy removal” problem for a point set, Edelsbrunner (1987) describes it without naming it, and for Rosen et al. (1992) it is a “convex hull” problem. We have opted to adopt the term used by Gerstenhaber (1951) and refer to it as the “frame” problem. It is important to stress from the outset that the frame problem is not the polyhedral facial decomposition problem from computational geometry also called the “convex hull” problem by many, e.g., Edelsbrunner (1987). This terminology creates confusion about two very different problems. In both cases, the input is a convex hull of a finite point set. The goal of the facial decomposition convex hull problem, however, is to express the resultant multidimensional polyhedron as an intersection of halfspaces. It is an interesting problem in its own right but it is combinatorial in nature and, in the general case, has no polynomial algorithms. This version of the convex hull problem is not the topic of this paper. Our interest is on the deterministic frame problem which is nowhere as difficult as facial decomposition. In fact, it is easily solved in polynomial time. As we shall see, the frame problem has many applications (including as a subproblem in facial decomposition) and presents its own theoretical and computational challenges, especially in large scale applications. In this paper we generalize the definition of the frame problem to the case of unbounded polyhedra and discuss, implement, and test solution algorithms on massive data sets. We introduce two algorithms, \textbf{Naive} and \textbf{PolyFrame}, to solve this new generalization of the frame problem.

Let \( P = \{p^1, \ldots, p^n\} \) be a finite set of \( n \) points in \( \mathbb{R}^m \). The set \( P \) can be used to generate different polyhedral objects in \( \mathbb{R}^m \) by combining its elements using constrained linear operations. The elements of \( P \) are the generators of the object they define. There are four fundamental objects: \( i) \) the linear hull, \( ii) \) the affine hull, \( iii) \) the conical hull, and \( iv) \) the convex hull of the points in

\[ 1 \] Corresponding Author: School of Business, The University of Mississippi, University, MS 38677; email: jdul@olemiss.edu.
Often, there exist proper subsets of $\mathcal{P}$ that suffice to describe the particular object. Generators that are not necessary to describe the objects are said to be *non-essential* or *redundant*. There exists a special interest in minimal cardinality subsets used to generate a given polyhedral set. The minimum cardinality subsets needed to generate the linear and affine hull are the familiar “bases.” The minimum cardinality subsets needed to generate the conical or convex hull are referred to as “frames” (Gestenhaber (1951), Wets and Witzgall (1967)). A *frame* is to a convex hull what a *basis* is to a linear combination.

Interest in frames occurs in several areas. The planar and three-dimensional frame problem is familiar in computational geometry and operations research because of its applications in optical recognition and routing and scheduling. Numerous specialized schemes have been proposed for the frame problem in these areas. By and large, standard schemes that work well such as “radial sweeping” and “gift wrapping” (see Preparata and Shamos (1985), and Graham and Yao (1990)) simply do not translate well beyond three dimensions due mostly to the “dimensionality curse.” There is a demand, though, for the solution of the frame problem in higher dimensions. Nonparametric ordering in multivariate statistics (Barnett (1976)) involves identifying frames of convex hulls in the space of the data points. Facial decomposition of polyhedral sets is familiar in operations research and computational geometry (Mattheiss and Rubin (1980)).

Until now, frames of convex hulls and conical hulls have been treated separately. Algorithms for finding frames are specifically available for either polyhedral set when finitely generated. The present work extends previous contributions by studying and developing a more general finitely generated polyhedral set; namely, an unbounded polyhedron specifically in the dimension of the data points. This is, in fact, a combination of a convex hull and a recession cone defined by a conical hull. In this sense, this work subsumes all previous work on frames. Our purpose is to study these objects directly and introduce two algorithms for finding their frame in the space where they reside. The first algorithm, *Naive*, is rather straightforward and obvious and it is based on the determination of feasibility of linear systems. The second algorithm, *PolyFrame*, is based on entirely different principles. It relies on the separation information obtained from the optimal solution of specially formulated linear programs to iteratively erect the frame one element at a time. As we shall see, *PolyFrame* is particularly effective for this purpose since the LPs depend on the cardinality of the frame.
The paper is organized as follows. The next section provides a brief background of the work that precedes our contribution and on which we build. Section 3 presents the perfunctory details of our notation and definitions but also discusses the assumptions. Section 4 presents the object of our interest: finitely generated unbounded polyhedra which we construct by combining a convex hull and a conical hull. Section 5 applies the definitions and results about finitely generated unbounded polyhedra to present an obvious algorithm for finding its frame. We call this algorithm Naive*. In Section 6 we begin to build the groundwork for the second of the algorithms proposed here, PolyFrame. We introduce a linear programming formulation with the important properties necessary for PolyFrame: feasibility and boundedness, and detection and separation of exterior points; all this without previous restrictive requirements on the dimension of the polyhedral sets. Section 7 presents PolyFrame. The results of tests comparing the two algorithms are reported and analyzed in Section 8. We explain the difference in performance between the two algorithms. The paper concludes the work in Section 9. A major point of this conclusion is that PolyFrame is substantially faster than Naive* and that it is the procedure of choice in large scale applications where the frame density is low.

2. Background. Our work is preceded by all the literature on finding bases of linear and affine hulls and a more directly relevant body of work on frames of conical and convex hulls. We focus here on papers about procedures for finding frames.

An early work specifically on finding frames is by Wets and Witzgall (1967) and applies to general finitely generated cones. Their idea is to arrange the vectors in a matrix which is transformed to a Jordan canonical form. A column different from a unit vector having only nonnegative components can be expressed as a nonnegative combination of the unit vectors and therefore is non-essential and, consequently, not a frame element. For columns with one or more negative components, Wets and Witzgall provide a set of rules to either determine whether the corresponding vector is a frame element or to obtain another canonical form of the matrix that will bring new information about the remaining vectors. The process is repeated until all vectors are conclusively classified. This procedure turns out to be equivalent to feasibility verification using LP.

Dulá, et al. (1992) explore the role of preprocessors to find some of the extreme points of convex hulls which, in this case, correspond to frame elements. Two preprocessors are based on the idea of maximizing arbitrary objective functions and another one on the fact that given any point, the point furthest away from it in Euclidean distance is necessarily extreme. The Frank-Wolfe algorithm is used on a quadratic program to resolve the status of points not uncovered by the preprocessors. The QP is formulated to obtain the projection of a point \( \hat{p} \) on the convex hull of a subset of the extreme points. When \( \hat{p} \) is exterior, the projection is used to define a hyperplane that can be translated to obtain one more extreme point. Repeating the process results in finding all the extreme points. This is where the idea of “erecting” the frame one element at a time was first introduced. To the best of the authors’ knowledge, Dulá, et al. (1992) presents the first output-sensitive frame algorithm.

Rosen, et al. (1992) explored presorting and preprocessing for finding extreme points of convex hulls. The process begins by finding the smallest hyperrectangle that contains all the points, and its center \( c \). Then, the Euclidean distance from the points to \( c \) is used to sort the points in descending order. This sorting is expected to position frame elements towards the top of the list. An iterative
method builds a list of points where an element is added if it cannot be conclusively classified as 
non-essential. The test for this proceeds from top to bottom of the sorting and involves feasibility 
verification of linear systems formulated to determine whether the point being tested is in the 
convex hull of the points already in the list. In the final phase, a naive feasibility test for extreme 
points is applied again to all the elements of the final list to conclusively separate extreme from 
non-essential.

Dulá and Helgason (1996) used an entirely different approach to find the frame of a convex hull. 
The procedure generates a nested sequence of hulls which expand one frame element at a time 
until the final hull is fully “erected.” Given any subset \( \mathcal{P} \) of \( \mathcal{P} \) such that its convex hull has full 
dimension and contains the origin in its interior, it is possible to verify with a specially formulated 
linear program whether or not any point in \( \mathcal{P} \) is in the convex hull of \( \mathcal{P} \). When the point is exterior, 
the LP solution provides the parameters for a hyperplane that supports this hull and separates 
it from the point. A translation of the hyperplane reveals one more extreme point of \( \mathcal{P} \) that is 
used to generate the next hull in the sequence. By repeating the process and updating the list of 
extreme points, the full frame will be identified. This new approach proved faster than traditional 
methods based on the feasibility of linear systems mainly because it involves linear programs with 
relatively few columns. Since the procedure is output-sensitive, the difference is accentuated when 
the frame is a relatively small subset of \( \mathcal{P} \). The procedure is initialized with a set of \( m + 1 \) affinely 
independent extreme points of the convex hull of the data points that must contain the origin in 
its interior. This initialization is computationally expensive.

output-sensitive algorithms for the frame problem of a convex hull that do not rely on naive 
feasibility verification. Clarkson (1994) provided the rough sketches of an idea for an algorithm 
that is essentially the same as in Dulá & Helgason (1996). Ottmann et al. in (1995) and Chan 
(1996) separately and independently proposed another output-sensitive frame algorithm. This 
algorithm differs from Clarkson and Dulá & Helgason in an important way. What Clarkson and 
Dulá & Helgason do with inner products, Ottmann et al. and Chan do with full-data LPs. It is 
not immediately clear how this approach compares to the inner product way since the full-data 
LPs actually have the potential of uncovering more than one new frame element every time they 
are solved. Ottmann’s et al. algorithm was cleaned up, formalized, and corrected in the Ottmann 
et al. (2001) paper. There are no computational results in any of the papers by Clarkson, Chan, 
and Ottmann, et al. so it is difficult to predict or compare performances.

The result in Dulá & Helgason (1996) was extended to finitely generated, pointed, conical hulls 
in Dulá, et al. (1997). Because of the pointedness assumption, frame elements are extreme rays. 
The method consists of generating a sequence of nested cones until the full frame is found. As in 
the convex hull case, a linear program determines if a vector belongs to a partial cone or is exterior. 
The solution to the LP provides a separating and supporting hyperplane for an exterior vector. 
This hyperplane is then “rotated,” rather than translated, away from the cone and towards the 
exterior vector until the last vector in the set is reached. This vector is necessarily an extreme 
ray. Initialization for this procedure is also expensive since it requires knowing a complement of 
\( m \) linearly independent extreme rays of the cone.
These recent works contain important ideas that are ripe for extension and generalization. We use these ideas as a starting point to develop the theory and the algorithms that we present in this work. Our results on obtaining the frame of a finitely generated unbounded polyhedron directly in the space where it resides subsumes and improves current available approaches and algorithms for the cases of conical and convex hulls.

3. Notation, Definitions, Terminology, and Assumptions. In this work, sets of points or vectors are identified by calligraphic letters; e.g., \( \mathcal{P}, \mathcal{A}, \mathcal{V}, \mathcal{E}, \mathcal{F} \). All points and resultant polyhedral objects are in \( \mathbb{R}^m \). For many of these sets there correspond matrices identified by the associated upper case letter, e.g., \( P, A, V, E, F \), constructed using the elements of the corresponding set as its columns. For example, the matrix \( P \) is constructed from the elements in \( P \) as its columns. The two sets \( A \) and \( V \) contain the \textit{generators} or \textit{generating elements} for the two polyhedral sets with which we are involved: polytopes (bounded polyhedra) and cones. These two sets constitute the basic data for our problem. The dimension of the points in \( A \) and \( V \) is \( m \) and their cardinalities are \( n_1 \) and \( n_2 \), respectively.

Points and vectors are two different words for the same thing, an ordered array of \( m \) coordinates; but we shall create a distinction. For the sake of clarity, we associate the word ‘point’ with the generating elements of convex hulls and ‘vector’ with those of conical hulls. Elements of cones are also called ‘rays.’ We shall use the convention that \( \langle a, b \rangle \) stands for the inner product of vectors \( a \) and \( b \); \( \langle P, x \rangle \) is the product of matrix \( P \) with the vector \( x \); and \( \langle \pi, P \rangle \) is the product of the transpose of vector \( \pi \) with matrix \( P \). The operation implies the dimension of the matrices and vectors must be compatible.

We present some definitions next.

**Definition 1.** The symbols ‘0’ and ‘\( e_i \)’ denote, respectively, the zero vector, and the \( i \)th unit vector when \( i = 1, \ldots, m \), with dimension determined by the context.

**Definition 2.** We shall use the symbol \( e_0 = (1, \ldots, 1)^T \) to denote a vector of all ones. Here too, the context will define the dimension of the vector.

**Definition 3.** The set \( \mathcal{H}(\pi, \beta) = \{ y \in \mathbb{R}^m | \langle \pi, y \rangle = \beta \} \) is the \textit{hyperplane} in \( \mathbb{R}^m \) with orthogonal vector \( \pi \) and level value \( \beta \).

**Definition 4.** A \textit{pointed cone} in \( \mathbb{R}^m \) is a cone for which there exists a hyperplane through the origin that intersects it only at the origin.

Let \( p^j \in \mathbb{R}^m, j = 1, \ldots, n, \mathcal{P} = \{ p^1, \ldots, p^n \} \) be a set of points, and \( P \) the associated matrix.

**Definition 5.** The \textit{positive} or \textit{conical} hull of \( \mathcal{P} \) is \( \text{pos}(\mathcal{P}) \) = \( \{ b \in \mathbb{R}^m | b = \langle P, x \rangle, x \in \mathbb{R}^n, x \geq 0 \} \).

**Definition 6.** The \textit{convex} hull of \( \mathcal{P} \) is \( \text{con}(\mathcal{P}) = \{ b \in \mathbb{R}^m | b = \langle P, x \rangle, \langle e_0, x \rangle = 1, x \in \mathbb{R}^n, x \geq 0 \} \).

In addition to these definitions, we have two assumptions for the remainder of this work.

**Assumption 1.** There is no duplication of elements in the sets \( \mathcal{A} \) and \( \mathcal{V} \). Moreover, the set \( \mathcal{V} \) cannot contain vectors which are multiples of each other.

**Assumption 2.** The cone \( \text{pos}(\mathcal{V}) \) is pointed.

\( \text{pos} \)\( \tilde{} \)\( \text{con} \)\( \text{pos} \)\( \text{con} \)\( \text{pos} \)\( \text{con} \)\( \text{pos} \)\( \text{con} \)

\( ^\dagger \) We shall also allow this, and other similar set operations, on the matrices constructed from point or vector sets; e.g. \( \text{pos}(\mathcal{V}) = \text{pos}(\mathcal{V}), \text{con}(\mathcal{A}) = \text{con}(\mathcal{A}), \text{etc.} \)
4. The Finitely Generated Unbounded Polyhedron. The combination of the convex hull of a set of points, \( A = \{a^1, \ldots, a^{n_1}\} \) (a polytope), with the conical hull of a set of vectors, \( V = \{v^1, \ldots, v^{n_2}\} \) (a cone) is an unbounded polyhedron, defined as

\[
pol(A, V) = \{ b \in \mathbb{R}^m \mid b = \langle A, x \rangle + \langle V, y \rangle; \; \langle e^0, x \rangle = 1; \; x \geq 0, \; y \geq 0, \; x \in \mathbb{R}^{n_1}, \; y \in \mathbb{R}^{n_2} \}. \tag{1}
\]

We refer to \( pol(A, V) \) (or, \( pol(A, V) \)) as the polyhedral hull of the sets (matrixes) \( A \) and \( V \). The operation “pol(·, ·)” is to be interpreted as the finitely generated polyhedral set resulting from the combination of the convex hull of the points in the first argument and the conical hull of the points in the second argument. The cone \( pos(V) \) is the set’s recession cone. Figures 1a and 1b depict the convex hull of a set of points, \( con(A) \), a polytope, and the conical hull of a set of vectors, \( pos(V) \), a cone, in two dimensions. Figure 1c shows how they can be combined to form \( pol(A, V) \), an unbounded polyhedron which is better appreciated in Figure 1d since non-essential elements are omitted leaving only the frame to describe it.

As with individual convex and conical hulls, and as illustrated in Figures 1c and 1d, it may be possible to define \( pol(A, V) \) using fewer than all the points in \( A \) and all the vectors in \( V \). The minimum cardinality subsets of \( A \) and \( V \) such that their polyhedral hull is \( pol(A, V) \) form a frame. A formal definition of the frame of \( pol(A, V) \) will be based on subsets of \( A \) and \( V \) as follows:

**Definition 7.** Let \( A \) be a set of \( n_1 \) points in \( \mathbb{R}^m \) and \( V \) a set of \( n_2 \) vectors in \( \mathbb{R}^m \). The frame of \( pol(A, V) \) is \( \{F_1, F_2\} \) where \( F_1 \subseteq A \) and \( F_2 \subseteq V \) such that: i) \( pol(F_1, F_2) = pol(A, V) \), ii) \( pol(F_1 \setminus \hat{a}, F_2) \neq pol(A, V) \) for any \( \hat{a} \in F_1 \), and iii) \( pol(F_1, F_2 \setminus \hat{v}) \neq pol(A, V) \) for any \( \hat{v} \in F_2 \).

We shall refer to the frame of a finitely generated polyhedral hull as \( \Phi \); thus \( \Phi = \{F_1, F_2\} \). Observe that since \( pos(V) \) is pointed, \( \Phi \) is the set of extreme elements of \( pol(A, V) \); that is, the elements of \( \Phi \) are the polyhedral hull’s extreme points and extreme rays. This means that, under our working assumptions, the frame of \( pol(A, V) \) is unique. Note that \( F_1 \) is not necessarily the frame of \( con(A) \); it is one of its subsets.
Previous treatment of convex and conical hulls (Dulá & Helgason (1996), Ottmann et al. (1995), and Dulá et al. (1997)) required a dimensionality assumption of the hulls. We shall relax this type of assumptions about the polytope con(\(A\)) or the cone pos(\(V\)). The convex hull of \(A\), con(\(A\)), resides in a subspace, \(L_1\), of dimension \(s_1\), \(1 \leq s_1 \leq m\), and the conical hull of \(V\), pos(\(V\)), resides in another subspace, \(L_2\), of dimension \(s_2\), \(1 \leq s_2 \leq m\). Any of the following is possible: i) one of the subspaces contains the other; i.e., \(L_1 \subseteq L_2\) or \(L_2 \subseteq L_1\); ii) no subspace contains the other, but they have some dimensions in common; or iii) the two subspaces share nothing but the origin.

A note about sequencing the identification of the frame elements. All procedures for identifying frames are essentially enumerative; they identify each frame element, sequentially, one at a time. The case of the polyhedral hull pol(\(A, V\)) presents choices about the enumeration sequence in \(\mathbb{R}^m\): 1) Find the frame, \(F_2\), of the recession cone first and then proceed to complete the frame of the entire polyhedral hull by identifying \(F_1\), its extreme points; 2) Reverse this order and find the frame of the convex hull first and then find the entire frame by identifying the extreme rays of the recession cone; and 3) Look for the extreme elements of pol(\(A, V\)) indiscriminately, without regard to whether they are points or rays. After some reflection it is clear that the last two alternatives are inefficient. This is due to the fact that the relation between the individual frames of pos(\(V\)) and con(\(A\)) and the whole frame, \(\Phi\), of pol(\(A, V\)) is different: the frame of pos(\(V\)) is \(F_2\) but the frame of con(\(A\)) is a superset of \(F_1\). If the second or third sequencing strategy is implemented, it may happen that, when trying to find the extreme points of pol(\(A, V\)) using a subset \(\hat{V}\) of \(V\), some points in \(A\) may emerge as temporarily extreme when they are actually not for the final hull pol(\(A, V\)). This is made obvious when we take \(\hat{V}\) to be empty (sequencing alternative 2 above and possibly in 3). Since the frame of con(\(A\)) is a superset of the set of extreme points of pol(\(A, \hat{V}\)), when a recession cone is added to con(\(A\)), it can absorb some of the extreme points in its umbra making them non-essential elements in the final unbounded polyhedral hull. All this can be verified in Figure 1. Notice how the three extreme points to the right of the vertical axis in Figure 1a are absorbed in the umbra of the recession cone of the final polyhedral hull in Figure 1d. This means that a search will have to be performed later for these points to expel them from the final frame; this represents additional computational effort. Therefore, any efficient algorithm to find the frame of pol(\(A, V\)) must start with the knowledge of the frame of pos(\(V\)).

In the next section we use the results above to design the first algorithm for finding the frame of pol(\(A, V\)); a simple and obvious naive application of the sufficient conditions from the definitions here.

5. A Naive Algorithm for the Frame of pol(\(A, V\)). An obvious algorithm emerges from the basic definitions above about finitely generated polyhedral hulls. We present and discuss this “naive” algorithm in this section.

The naive implementation follows the sequence prescribed at the end of the previous section; namely, begin by finding, the frame, \(F_2\), of pos(\(V\)) and proceed to complete the frame of pol(\(A, F_2\)) by finding \(F_1\). Since pos(\(V\)) is pointed, the frame is composed of the extreme rays of the cone. Therefore, vector \(v^k \in V\) belongs to the frame \(F_2\) if and only if the system
\[
S_1(V, k) : \begin{cases} 
(V \setminus v^k, y) = v^k, \\
y \geq 0.
\end{cases}
\] (2)

is infeasible. This way, finding \( F_2 \) requires testing the feasibility of system \( S_1(V, k) \) for \( k = 1, \ldots, n_2 \). Once \( F_2 \) is known we may proceed to the second stage which requires ascertaining the feasibility of the following system

\[
S_2(A, j) : \begin{cases} 
\langle A \setminus a^j, x \rangle + \langle F_2, y \rangle = a^j, \\
\langle e^0, x \rangle = 1, \\
x \geq 0, \\
y \geq 0;
\end{cases}
\] (3)

where \( a^j \in A \) is an element of \( \Phi \) if and only if the system is infeasible. This test needs to be repeated for \( j = 1, \ldots, n_1 \) to find all the extreme points of \( \text{pol}(A, V) \).

Note that performance can be enhanced if, every time a non-essential element is uncovered, it is excluded from further consideration in subsequent feasibility determinations. This improves performance since the size of the system is progressively reduced as non-essential elements become known. We implement this enhancement in a procedure called \textbf{Naive}*. What follows is a formal statement of \textbf{Naive}*. With the enhancement, each time a non-essential element is uncovered, the appropriate data matrix is updated by the removal of the redundant generator. The progressively more truncated matrices are denoted by \( \bar{V} \) and \( \bar{A} \).

\begin{center}
\begin{tabular}{|c|}
\hline
Procedure \textbf{Naive}* \\
[INPUT:] \( A, V, A, V \). \\
[OUTPUT:] \( F_1, F_2 \). \\
\textbf{Initialization.} \( \bar{A} \leftarrow A, \bar{V} \leftarrow V \). \\
\textbf{Phase 1.} \\
For \( k = 1 \) to \( n_2 \), Do: \\
\hspace{1em} If \( S_1(\bar{V}, k) \) feasible: \\
\hspace{2em} \( v^k \) is non-essential: \\
\hspace{3em} \( \bar{V} \leftarrow \bar{V} \setminus v^k \) (“Enhancement”). \\
\hspace{2em} Else: \( v^k \in F_2 \). \\
\hspace{1em} EndIf \\
\hspace{1em} Next \( k \). \\
\textbf{Phase 2.} \\
For \( j = 1 \) to \( n_1 \), Do: \\
\hspace{1em} If \( S_2(\bar{A}, j) \) feasible, \\
\hspace{2em} \( a^j \) is non-essential: \\
\hspace{3em} \( \bar{A} \leftarrow \bar{A} \setminus a^j \) (“Enhancement”). \\
\hspace{2em} Else \( a^j \in F_1 \). \\
\hspace{1em} EndIf \\
\hspace{1em} Next \( j \). \\
\textbf{Finalization.} \( \Phi = \{F_1, F_2\} \). \\
\hline
\end{tabular}
\end{center}
Two Notes about Naive$^*$.  

1. Determining the feasibility of a linear system can be done a number of ways. Computationally, this is equivalent to solving a linear program.

2. Computational complexity. Procedure Naive$^*$ is polynomial since it can be executed by solving $n_1 + n_2$ LPs.

6. An LP Formulation for Location and Separation of Points w.r.t. a Polyhedral Hull Directly in $\mathbb{R}^m$. In this section we formulate a linear program used to understand and manage polyhedral hulls of the sort of $\text{pol}(\mathcal{A}, \mathcal{V})$ directly in the space where they reside. This LP is an important component in the second of our algorithms presented here, PolyFrame.

In finding the frame of $\text{pol}(\mathcal{A}, \mathcal{V})$, PolyFrame will require the identification of interior and exterior generators to a sequence of nested cones and polyhedral hulls. Moreover, for exterior vectors and points, the procedure will require knowledge of a strictly separating hyperplane; all this directly in $\mathbb{R}^m$. The new LP is formulated to provide two important bits of information about an arbitrary vector or point in $\mathbb{R}^m$ and a given polyhedral hull $\text{pol}(\cdot, \cdot)$: \(i\) the location of the point (vector) with respect to the polyhedral hull (recession cone); more specifically does it belong to the set or is it external; and, if the element is in the exterior; \(ii\) a separating hyperplane between the generator and the set. The first item of information can be obtained by determining the feasibility of an appropriate linear system such as $S_1(\cdot, \cdot)$ or $S_2(\cdot, \cdot)$ above but this is unsuitable if the information on separation in the second item is also required since we cannot extract much from an infeasible system.

Linear program formulations for analogous purposes have been proposed before. Dulá & Helgason (1996) propose a formulation for convex hulls and Dulá, et al. (1997) one for conical hulls. These past LPs have been subject to restrictive dimensionality conditions. For example, in Dulá & Helgason (1996), the LP requires a full $m$-dimensional initializing convex hull containing the origin. The LP in Dulá et al. (1997) is subject to a similar dimensionality condition. The LP introduced here, generalizes previous formulations. It applies to cones, convex hulls, and unbounded polyhedral hulls with multiple extreme points. Unlike previous formulations, it is not subject to dimensionality requirements.

We present the LP formulation next. In the results that follow we demonstrate that this LP detects whether an arbitrary vector or point, $b \in \mathbb{R}^m$, is internal or external to a polyhedral hull, even if it is just a cone ($\mathcal{A} = \emptyset$) or a bounded polytope ($\mathcal{V} = \emptyset$), and that it provides information about a strictly separating hyperplane for any exterior point. All this, of course, requires that the LP be feasible and bounded which shall be our first result. Let us insert at this point some special notation.

The data for the LP are the two data sets needed to define a polyhedral hull, $\text{pol}(\cdot, \cdot)$. We shall work with general subsets of the original data sets, $\mathcal{A}$ and $\mathcal{V}$. We use the following notation for these subsets and their corresponding index sets:

- Define $\hat{\mathcal{A}}$ to be an arbitrary subset of $\mathcal{A}$ and $\hat{\mathcal{J}} \subseteq \mathcal{J}; \mathcal{J} = \{1, 2, \ldots, n_1\}$ its corresponding index set. The matrix $\hat{\mathcal{A}}$ contains the elements of $\hat{\mathcal{A}}$ as its columns.

- Define $\hat{\mathcal{V}}$ to be an arbitrary subset of $\mathcal{V}$ and $\hat{\mathcal{K}} \subseteq \mathcal{K}; \mathcal{K} = \{1, 2, \ldots, n_2\}$ its corresponding index set. The matrix $\hat{\mathcal{V}}$ contains the elements of $\hat{\mathcal{V}}$ as its columns.
The set \( \mathcal{E} = \{ -e^0, e^1, \ldots, e^m \} \) with the corresponding matrix being \( E \).

The set \( \mathcal{E} \) contains a collection of “auxiliary” vectors. These vectors positively span the space \( \mathbb{R}^m \); that is, their conical hull is the entire space. The LP formulations follow:

\[
P(\hat{A}, \hat{V}, b) : \begin{align*}
    \min_{x \geq 0, y \geq 0, w \geq 0} & \quad z = \langle e^0, w \rangle \\
    \text{s.t.} & \quad \langle \hat{A}, x \rangle + \langle \hat{V}, y \rangle + \langle E, w \rangle = b, \\
    & \quad \langle e^0, x \rangle = 1.
\end{align*}
\] (4)

The corresponding dual is:

\[
D(\hat{A}, \hat{V}, b) : \begin{align*}
    \max_{\pi, \beta} & \quad \omega = \langle \pi, b \rangle + \beta \\
    \text{s.t.} & \quad \langle \pi, \hat{A} \rangle + \beta e^0 \leq 0, \\
    & \quad \langle \pi, \hat{V} \rangle \leq 0, \\
    & \quad \langle \pi, e^0 \rangle \geq -1, \\
    & \quad \langle \pi, e^i \rangle \leq 1, \quad i = 1, \ldots, m;
\end{align*}
\] (5)

where \( b \) is any element of \( \mathbb{R}^m \). Let us make some observations about this primal dual pair (which we shall refer to as \( P/D \), for brevity). Notice that since \( \hat{A} \subseteq A \) and \( \hat{V} \subseteq V \), \( \text{pol}(\hat{A}, \hat{V}) \subseteq \text{pol}(A, V) \). This means that the LPs apply to data that generate polyhedral hulls that are subsets of the original object. Also, notice that when \( \hat{A} = \emptyset \), the primal dual pair \( P/D \) relates to a recession cone \( \text{pos}(\hat{V}) \) or, when \( \hat{V} = \emptyset \), to a bounded polytope \( \text{con}(\hat{A}) \). Finally, note how the vector \( e^0 \) has multiple duty. In the primal its dimension in the objective function is \( m + 1 \); in the constraint \( \langle e^0, x \rangle \) the dimension is \( |\hat{J}| \), the number of columns of the matrix \( \hat{A} \); and it appears hidden (and negated) in the matrix \( E \) as its first column with dimension \( m \). In the dual it appears twice, once with dimension \( m \) and another time with dimension \( |\hat{J}| \).

Let \( z^* = \omega^* \) be the value of the objective function at optimality. Let the optimal primal solution be composed of \( x^* \), \( y^* \) (dimensions defined by \( \hat{J} \) and \( \hat{K} \), respectively), and \( w^* \in \mathbb{R}^{m+1} \); and the optimal dual solution of \( \pi^* \in \mathbb{R}^m \) and \( \beta^* \in \mathbb{R} \).

We have the following four results about the pair \( P/D \).

**RESULT 1.** The linear program \( P(\hat{A}, \hat{V}, b) \) is feasible and bounded.

Proof. The proof for when \( b \) is a vector and the hull is a recession cone is immediate (i.e., \( \hat{A} = \emptyset \)). We present a proof for when \( \hat{A} \neq \emptyset \). Select an arbitrary point \( a^j \) such that \( \hat{j} \in \hat{J} \). Let \( x_j = 1 \) and \( x_j = 0 \) if \( j \neq \hat{j} \); \( j \in \hat{J} \). We use this to satisfy the unity constraint in \( P(\cdot) \). Then, for any \( b \in \mathbb{R}^m \) and \( y = 0 \), the linear program reduces to

\[
\begin{align*}
    \min_{x \geq 0, w \geq 0} & \quad z = \langle e^0, w \rangle \\
    \text{s.t.} & \quad \langle E, w \rangle = b - a^\hat{j} = \tilde{b} \in \mathbb{R}^m, \\
    & \quad \langle e^0, x \rangle = 1.
\end{align*}
\] (6)
This linear program is always feasible since the vectors in \( E \) positively span \( \mathbb{R}^m \). Since \( \pi = 0, \beta = 0 \), is a feasible solution, \( D(\cdot) \) is feasible and \( P(\cdot) \) is bounded by weak duality.

The linear program \( P(\hat{A},\hat{V},b) \) is formulated to penalize the use of the vectors in \( E \). Only if \( b \not\in \text{pol}(\hat{A},\hat{V}) \) are these vectors involved in a solution. This is formalized in the next result.

**Result 2.** \( z^* = \omega^* = 0 \) if and only if \( b \in \text{pol}(\hat{A},\hat{V}) \).

**Proof.** Suppose \( z^* = 0 \). Then, \( \langle e^0, w^* \rangle = 0 \) in \( P(\hat{A},\hat{V},b) \) implying \( w^* = 0 \); and \( x^* \) and \( y^* \) must be such that

\[
\begin{align*}
\langle \hat{A},x^* \rangle + \langle \hat{V},y^* \rangle &= b, \\
\langle e^0,x^* \rangle &= 1.
\end{align*}
\]

(7)

Since \( x^* \geq 0 \) and \( y^* \geq 0 \), this is enough to conclude that \( b \in \text{pol}(\hat{A},\hat{V}) \) by Expression (1) (Section 4). Conversely, suppose \( b \in \text{pol}(\hat{A},\hat{V}) \). Then the linear system \( \langle \hat{A},x \rangle + \langle \hat{V},y \rangle = b, \langle e^0,x \rangle = 1 \) has a solution \( \hat{x} \geq 0, \hat{y} \geq 0 \). This solution is feasible to \( P(\cdot) \) if we set \( w = 0 \). Therefore, there exists a feasible solution such that \( \langle e^0,w \rangle = 0 \) implying \( z^* \leq 0 \). On the other hand, the condition \( w \geq 0 \) implies \( z^* \geq 0 \); so \( z^* = 0 \). The proof is valid when \( \hat{A} = \emptyset \) meaning that the result applies if the test point is a vector and the polyhedral set is just the recession cone.

Note that the optimal objective function value, \( z^* \), is greater than zero, if and only if the point \( b \) is exterior. We need to ascertain that when the test point is exterior we shall not have to deal with the triviality that \( \pi^* = 0 \) in the optimal solution to \( D(\cdot) \). The next result establishes this.

**Result 3.** \( z^* > 0 \) implies \( \pi^* \neq 0 \).

**Proof.** Suppose \( \pi^* = 0 \). When \( \hat{A} \neq \emptyset \), then \( z^* = \omega^* = \beta^* > 0 \) by strong duality. But dual feasibility means, \( \langle \pi^*,a^j \rangle + \beta^* \leq 0 \), \( j \in \hat{J} \) implying \( \beta^* \leq 0 \); a contradiction. When \( \hat{A} = \emptyset, 0 < z^* = \omega^* = \langle \pi^*,b \rangle = 0 \); also a contradiction.

Result 4, next, shows how the solution to \( D(\cdot) \) provides the parameters to define a strictly separating hyperplane whenever the test vector or point is external to the hull generated by any subset of the data set. Knowing that \( \pi^* 
eq 0 \) whenever the test point is exterior to the polyhedral hull means that this hyperplane is never trivial. This makes this result effective.

**Result 4.** Let \( w^* = z^* > 0 \) and \( \pi^*,\beta^* \) be the optimal solution to \( (D(\hat{A},\hat{V},b)) \). Then the hyperplane \( \mathcal{H}(\pi^*,-\beta^*) \) strictly separates the polyhedron \( \text{pol}(\hat{A},\hat{V}) \) from \( b \) in the following two cases:

- **Case 1:** \( b \) is a point and \( \hat{A} \neq \emptyset \); and
- **Case 2:** \( b \) is a vector and \( \hat{A} = \emptyset \).

**Proof.** The proof for Case 1 has all the elements to show Case 2 as well. \( z^* > 0 \) means \( \langle \pi^*,b \rangle + \beta^* > 0 \). We proceed to show that every point, \( \bar{p} \in \text{pol}(\hat{A},\hat{V}) \) is such that \( \langle \pi^*,\bar{p} \rangle + \beta^* \leq 0 \). Let \( \bar{p} \in \text{pol}(\hat{A},\hat{V}) \). Then, by the definition of a polyhedral hull in Expression (1) (Section 4), there exist \( \bar{x} \geq 0 \) and \( \bar{y} \geq 0 \) such that \( \bar{p} = \langle \hat{A},\bar{x} \rangle + \langle \hat{V},\bar{y} \rangle, \langle e^0,\bar{x} \rangle = 1 \).

Now we have that

\[
\langle \pi^*, \bar{p} \rangle = \langle \pi^*, \langle \hat{A},\bar{x} \rangle + \langle \hat{V},\bar{y} \rangle \rangle,
\]

\[
= \langle \langle \pi^*, \hat{A} \rangle, \bar{x} \rangle + \langle \langle \pi^*, \hat{V} \rangle, \bar{y} \rangle.
\]

\[
\leq \beta^* e^0 \quad \langle \pi^*, \hat{A} \rangle \rangle + \langle \langle \pi^*, \hat{V} \rangle, \bar{y} \rangle \leq 0.
\]
since \( \langle \pi^*, a^j \rangle + \beta^* \leq 0 \); \( j \in \hat{J} \), and \( \langle \pi^*, v^k \rangle \leq 0 \); \( k \in \hat{K} \). Therefore:
\[
\leq -\beta^* \langle e^0, \hat{x} \rangle + 0 = -\beta^*.
\]

Having shown that \( \langle \pi^*, b \rangle > -\beta^* \) and \( \langle \pi^*, \tilde{p} \rangle \leq -\beta^* \) for any \( \tilde{p} \in \text{pol}(\bar{A}, \bar{V}) \), we establish that the hyperplane \( H(\pi^*, -\beta^*) \) separates \( \text{pol}(\bar{A}, \bar{V}) \) from \( b \). The proof for Case 2 (i.e., \( \bar{A} = \emptyset \) and the polyhedral hull is just a recession cone) is along the same lines but easier.

With these four results we establish that the LP formulations \( (P)/(D) \) are always feasible and bounded for any test point in \( \mathbb{R}^m \) (Result 1); we have necessary and sufficient conditions to identify exterior points, (Result 2); and, for any such point, we are guaranteed to obtain a non-trivial separating hyperplane (Results 3 and 4). The results apply equally to the case when we are dealing with a test vector and the recession cone (\( \hat{\text{trivial separating hyperplane} (\text{Results 3 and 4}) \)). We shall use this LP formulation in the design of the algorithm \textbf{PolyFrame} presented next.

7. \textbf{PolyFrame}: A New Algorithm for Erecting the Frame of a Polyhedral Hull. The second algorithm we develop, \textbf{PolyFrame}, for finding the frame of a finitely generated unbounded polyhedron is based on the LP formulated in the previous section.

\textbf{PolyFrame} follows the prescribed two-stage sequence established at the end of Section 4; namely, begin by finding the frame, \( \mathcal{F}_2 \), of \( \text{pos}(\mathcal{V}) \) and proceed to complete the frame, \( \Phi \), of the entire polyhedral hull, \( \text{pol}(\mathcal{A}, \mathcal{F}_2) \). The final result which we present next defines how, every time an exterior test generator is identified, the separating hyperplane provided by the solution to the LP can be used to identify a new frame element.

\textbf{RESULT 5.} \textit{The following two results apply to the solution of the primal/dual pair \( (P)/(D) \) when the optimal objective function value is not zero:}

\[ \text{i. Let } \emptyset \neq \hat{\mathcal{V}} \subset \mathcal{V} \text{ and } \pi^* \neq 0 \text{ be the optimal solution to } (D(\emptyset, \hat{\mathcal{V}})) \text{ with } z^* > 0. \text{ If there exists only one vector } v^* \in \mathcal{V} \text{ such that } v^* = \arg \max_{\{v^* \mid \langle \pi^*, v^* \rangle > 0\}} \langle \sigma, v^* \rangle, \text{ for a vector } \sigma \in \mathbb{R}^m \text{ linearly independent of } \pi^* \text{ such that } \langle \sigma, \mathcal{V} \rangle \leq 0, \text{ then } v^* \in \mathcal{F}_2. \]

\[ \text{ii. Let } \emptyset \neq \hat{\mathcal{A}} \subset \mathcal{A}. \text{ Let } b \in \mathcal{A} \text{ and } \hat{\mathcal{V}} = \mathcal{F}_2. \text{ If } z^* > 0 \text{ and there exists only one point } a^* \in \mathcal{A} \text{ such that } a^* = \arg \max_{j \in \hat{J}} \langle \pi^*, a^j \rangle, \text{ then } a^* \in \mathcal{F}_1. \]

Proof of Result i. We shall show that \( v^* \) is an extreme ray of \( \text{pos}(\mathcal{V}) \). \( \hat{\mathcal{A}} = \emptyset \) means we are dealing only with the recession cone, \( \text{pos}(\mathcal{V}) \), of the polyhedral hull. Set \( \zeta^* = \langle \hat{\sigma}, v^* \rangle / \langle \pi^*, v^* \rangle \) (note: \( \zeta^* \leq 0 \)). Then \( \langle \hat{\sigma}, v^k \rangle \leq \zeta^* \langle \pi^*, v^k \rangle \); \( \forall k \) s.t. \( \langle \pi^*, v^k \rangle \geq 0 \) (equality only when \( v^k = v^* \)). Consider the hyperplane \( \mathcal{H}(\zeta, 0) \) where \( \zeta = \hat{\sigma} - \pi^* \zeta^* \). To establish that \( v^* \) is extreme we shall show that \( \mathcal{H}(\zeta, 0) \) supports \( \text{pos}(\mathcal{V}) \) and that \( v^* \) is the only element in the support set. Let us begin by establishing the simple support result. Note that \( \langle \pi, v^k \rangle = \langle \hat{\sigma} - \pi^* \zeta^*, v^k \rangle = \langle \hat{\sigma}, v^k \rangle - \zeta^* \langle \pi^*, v^k \rangle \leq 0 \forall k \). To see this consider two cases. In the first case look only at the vectors \( v^k \) such that \( \langle \pi^*, v^k \rangle > 0 \). Here \( \langle \hat{\sigma}, v^k \rangle - \zeta^* \langle \pi^*, v^k \rangle \leq 0 \) by construction. The second case must treat the complement; all \( v^k \) such
that $\langle \pi^*, v^k \rangle \leq 0$. For this case we know $\langle \hat{\sigma}, v^k \rangle \leq 0$ by our requirement for $\hat{\sigma}$. Since $\xi^* \leq 0$ it follows that $\langle \hat{\sigma}, v^k \rangle - \xi^* \langle \pi^*, v^k \rangle \leq 0$ for this case too. Therefore, $\langle \hat{\pi}, v^k \rangle \leq 0$ for all $V^k$ implying the hyperplane $\mathcal{H}(\hat{\pi}, 0)$ supports $\text{pos}(\mathcal{V})$ and $v^*$ is its only element by the uniqueness assumption about the generator that attains the ‘argmax’. Note, $v^*$ is the last vector of $\mathcal{V}$ encountered by the separating hyperplane when this is rotated away from the recession cone while anchored to the origin.

Proof of Result \textit{ii}. The proof is analogous to the one above if we use the hyperplane $\mathcal{H}(\pi^*, \gamma^*)$ where $\pi^*$ is the optimal solution to $(D(\hat{A}, \hat{V}, b))$ and $\gamma^* = \langle \pi^*, a^* \rangle$. The hyperplane $\mathcal{H}(\pi^*, \gamma^*)$ supports $\text{pol}(A, \mathcal{F}_2)$ and $a^*$ is the singleton in the support set. Note, $a^*$ is the last point of $A$ contacted by the separating hyperplane when translated away from the polyhedral hull.

Result 5 squares with the indicated sequence discussed at the end of Section 4 for a procedure for finding the frame of $\text{pol}(A, \mathcal{V})$; namely find $\mathcal{F}_2$ first and then find the extreme points of the polyhedral hull $\text{pol}(A, \mathcal{F}_2)$. Procedure \textbf{PolyFrame} uses this sequencing in conjunction with Result 5. We present \textbf{PolyFrame} next.

**Procedure PolyFrame**

[INPUT:] $A, \mathcal{V}$.  
[OUTPUT:] $F_1, \mathcal{F}_2$.

\textbf{Phase 1.}

a. Initialization:  
$\mathcal{F}_2 \leftarrow$ any extreme ray of pos($\mathcal{V}$).  
$\hat{\sigma} \leftarrow$ Any feasible solution to $(D(\emptyset, V, b))$.

b. For $k = 1$ to $n_2$, Do:  
While $v^k$ is unclassified, Do:  
Solve $(P(\emptyset, \mathcal{F}_2, b))$.  
If $z^* = 0$: Next $k$. ($v^k$ not extreme)  
Else:  
$v^* = \argmax \{v^k \in \mathcal{V} | \langle \pi^*, v^k \rangle > 0 \}$.

EndIf  
If $v^*$ is unique, $\mathcal{F}_2 = \mathcal{F}_2 \cup \{v^*\}$.  
Else: Resolve tie to identify new frame element, $v^*$.  
$\mathcal{F}_2 = \mathcal{F}_2 \cup \{v^*\}$.  
EndIf  
Continue While.

Next $k$.

c. $\mathcal{F}_2 = \mathcal{F}_2$.

\textbf{Phase 2.}

a. Initialization.  
$\mathcal{F}_1 \leftarrow$ any extreme point of $\text{pol}(A, \mathcal{F}_2)$.

b. For $j = 1$ to $n_1$, Do:  
While $a^j$ is unclassified, Do:  
Solve $(P(\hat{\mathcal{F}}_1, \mathcal{F}_2, b))/D(\hat{\mathcal{F}}_1, \mathcal{F}_2, b))$.  
If $z^* = 0$: Next $j$. ($a^j$ not extreme)  
Else:  
$a^* = \argmax \{a^j \in A | \langle \pi^*, a^j \rangle > 0 \}$.

EndIf  
If $a^*$ is unique, $\hat{\mathcal{F}}_1 = \hat{\mathcal{F}}_1 \cup \{a^*\}$.  
Else: Resolve tie to identify new frame element, $a^*$.  
EndIf  
Continue While.

Next $j$.

c. $\mathcal{F}_1 = \mathcal{F}_1$.

\textbf{Finalization.} $\Phi = \{F_1, \mathcal{F}_2\}$.
Seven Notes about PolyFrame.

1. The Mechanics of the Procedure. Procedure PolyFrame generates a sequence of nested polyhedra. In Phase 1, this is a sequence of nested cones. Each new cone incorporates one new extreme ray until all the recession cone’s frame elements are identified. In Phase 2, a sequence of nested unbounded polyhedra is generated; all receding in the same set of directions found in Phase 1. The polyhedra grow one extreme point at a time. At any iteration in either of the two phases, if the iterant, \(v^k\) or \(a^j\), is not already known to be extreme, an LP will be solved that will either conclusively classify it as redundant or provide the parameters for a hyperplane that will identify a previously unclassified frame element. This frame element may or may not be the current iterant. The iteration index is incremented only when the status of the current iterate is resolved as either redundant or extreme. The LPs are augmented in either phase, by the inclusion in the coefficient matrix of the data of a new frame element. The increment occurs only if the pass identifies a frame element. Notice that each time the search for the argmax of inner products is executed in Part b of Phases 1 and 2, a new extreme element is identified. This means that the number of times these steps are executed is at most the number of extreme elements in the polyhedral hull. Since the number of operations depends, in part, on the density of extreme elements of the polyhedral hull, performance of procedure PolyFrame is output-sensitive in the sense of Clarkson (1994).

2. Validity of the algorithm. The procedure is finite because both iteration counters are finite and each pass in Phase 1 and Phase 2 necessarily produces a conclusive classification of a generator as either non-essential or extreme. To see this consider the three cases: the iterant, \(v^k\) or \(a^j\), is already classified; i.e., it has been identified as extreme in a previous iteration; ii) is non-essential (redundant); or iii) is exterior to the current partial hull. In the first two cases, the iteration counter is incremented immediately. In the third case, the iteration counter is increased after a finite sort identifies the generator, \(v^*\) or \(a^*\). This generator is necessarily a frame element. The only concern left is misclassification. We shall focus on Phase 2. Suppose that a frame element, \(a^k\) is misclassified. For this to happen, at one of the \(n_1\) iterations, it would have to satisfy the necessary and sufficient conditions for an interior point to a partial hull (Solution to LP is 0, Result 2). This, of course, is impossible if \(a^k\) is an extreme point of the polyhedral hull. The other case is that a nonframe element, \(a^k\), is misclassified. If this happened the algorithm will provide a supporting hyperplane for the entire polyhedral hull with a full recession cone with support set precisely at \(a^k\) (in the absence of ties) and this is a contradiction. In the presence of ties this argument is applied recursively to arrive at the same conclusion.

3. Computational Complexity. Let us look at Phase 1; Phase 2 is analogous. Set aside temporarily the possibility of ties in any iteration. The classification of every generator, whether extreme or redundant, is the consequence of the solution to an LP. Therefore, a total of \(n_2\) LPs will be solved in Phase 1 and their dimension is output-sensitive – bounded, essentially, by the cardinality of the frame. All other operations are inner products and sortings over finite sets and the iterations when this occurs is also output-sensitive. This is also the case in Phase 2. LPs can be solved in polynomial time. This establishes the polynomiality of the procedure. The possibility of ties does not affect this conclusion since ties are resolved by recursively applying the procedure to the set of points involved (see Note 5 below.)
4. **Initializations.** Initializing **PolyFrame** requires three different calculations: finding an initial extreme ray, finding a vector $\hat{\sigma}$, and finding an extreme point. Finding the extreme elements can be done in several ways. For example, one can use as a temporary “initializing” element any interior ray of the recession cone (e.g., a strictly positive combinations of elements of $V$) and iterating until $z^* > 0$. At this point the procedure will identify a true extreme ray in ‘Step b’ of Phase 1 which will be the real initializer and the interior point can be discarded. For Phase 2, with $F_2$ in hand, we take any point in the interior of the convex hull of $\text{con}(A)$ (e.g., its barycenter), as a temporary initializer and run the procedure until the first extreme point is identified. After this, the interior point can be discarded and the extreme point serves as the initializer. This was the procedure used in our implementations of **PolyFrame**. Other ideas can be based on simple preprocessing-type schemes such as those discussed in Dula, et al. (1992). The most expeditious way for finding a suitable $\hat{\sigma}$ is to find a feasible solution to the LP using $(D(\emptyset, V, b))$ for any objective function, $b$. One way to avoid the (rare) possibility that $\hat{\sigma}$ ever is not linearly independent to any, $\pi^*$, of the sequence of optimal solutions for LPs, $(D(\emptyset, \hat{V}, b))$, is to take two or more basic solutions to the LP $(D(\emptyset, V, b))$ with the full data set and two significantly different objective functions (e.g., $b$ and $-b$, etc.) and setting $\hat{\sigma}$ equal to a strict positive linear combination of the optimal solutions $\pi^*$. Our implementation used a $\hat{\sigma}$ initialized solving two LPs $(D(\emptyset, V, b))$ and $(D(\emptyset, V, -b))$.

5. **Resolving Ties in Phases 1 and 2.** If the points where $\max_{\{v^k \mid \langle \pi^*, v^k \rangle > 0\}} \langle \hat{\sigma}, v^k \rangle / \langle \pi^*, v^k \rangle$ in Phase 1 or $\max_{\{a^j \mid \langle \pi^*, a^j \rangle > 0\}} \langle \pi^*, a^j \rangle / \langle \pi^*, v^k \rangle$ in Phase 2 are not unique, more processing is needed to identify the requisite frame element for the procedure to make progress. In either phase, the problem reduces to finding any one element of the frame of the set of points involved in the tie. In our implementation this was done by a recursive application of **PolyFrame** on the points involved in the tie. This is permitted because of our flexibility in dealing with objects with fewer than $m$ dimensions. It may be more expeditious, however, to find an extreme point using some opportunistic preprocessor.

6. **A Procedure in $\mathbb{R}^{m+1}$.** It is clear that there is an equivalence between the unbounded polyhedron $\text{pol}(A, V)$ and the cone

$$ \text{pos}(C) = \{d \in \mathbb{R}^{m+1} \mid d = \langle C, z \rangle, \ z \geq 0, \ z \in \mathbb{R}^{n_1 + n_2} \}, \quad (8) $$

where $C = \begin{bmatrix} A & V \\ e^0 & 0 \end{bmatrix}$ is an $(m + 1) \times (n_1 + n_2)$ matrix. The extreme elements (points and rays) of $\text{pol}(A, V)$ correspond to extreme rays of the cone $\text{pos}(C)$, and vice-versa. Therefore, a third procedure based on Phase 1 of **PolyFrame** could be used to solve the problem of finding the frame, $\Phi$. The procedure would simply apply Phase 1 of **PolyFrame** to identify the extreme rays of $\text{pos}(C)$ and then proceed to separate extreme points and extreme rays of $\text{pol}(A, V)$. Either way, the number of iterations (LPs solved) must be the same (see Notes 1 and 2). A single phase procedure, however, has several obvious disadvantages compared to the two-phase approach in **PolyFrame**. One disadvantage is that all points and vectors are in $\mathbb{R}^{m+1}$. This affects directly calculations of inner products and LPs. We can expect to pay a higher price in terms of computational times for this step up in dimension. There is also an
adverse impact associated with the number of inner products calculated and tests performed. In ‘Step b’ of Phase 1, calculating inner products is more laborious than the analogous step in Phase 2. The first phase requires calculating extra inner products for the denominator in the ratio and testing their positivity. This work is not required for all those extreme elements that correspond to extreme points of pol(\(A, V\)) in Phase 2 of PolyFrame. Finally, both phases require sorting scalar values to identify the element where the maximum ratio of inner products (Phase 1) or just simple inner products (Phase 2) occurs. We can expect a comparable number of sorts in the one and two-phase approaches. In a single phase method in higher dimension, however, we can anticipate sorts over larger sets than in the two-phase method since in the latter the sorting is over sets of just rays or sets of just points. Sortings over smaller sets give PolyFrame a computational advantage. For these reasons, work on a third procedure based on a transformation of the problem into \(m + 1\) dimensions did not warrant further consideration.

7. Difference between Naive* and PolyFrame. The fundamental difference between Naive* and PolyFrame relates to the size of the LPs they process. Procedure Naive* starts with large LPs that get progressively smaller as new non-essential elements are identified, as provided by the enhancement. Algorithm PolyFrame starts with small LPs that grow as new frame elements are uncovered. These LPs only contain extreme elements (plus the full complement of “auxiliary” vectors in \(E\)). Note that PolyFrame, however, needs to calculate inner products every time an exterior point is identified; these operations are not required by Naive*.

The differences between Naive* and PolyFrame raise questions about how these factors interact and affect performance differences between the two algorithms. Understanding these factors becomes important in large scale applications. To test the two algorithms and address these issues, a problem suite was created. The next section reports on our tests and comparisons.

8. Computational Testing and Results. The two procedures Naive* and PolyFrame were implemented in a computer in order to verify and validate them, and to test and compare their performance. They were both coded in FORTRAN 90 and executed on a SGI Origin 2000 computer. This computer was equipped with eight R10000 processing units and two gigabytes of memory. The CPU clock speed was 195 MHz with a word size of 64 bits. To solve the LPs we used the subroutine ‘ddlprs’ from the IMSL Library (1994).

An extensive and varied problem suite was created to test the two algorithms. One reason why data sets were generated is that the authors are unaware of a standard test bank for experiments. Another reason is the need to be able to control characteristics and variety of polyhedral hulls that permit useful conclusions. Each polyhedral hull requires two different data files; one with the data set for the “convex hull” part and one for the recession cone; i.e., what corresponds to \(A\) and \(V\) of pol(\(A, V\)), respectively. The data sets were randomly generated. In order to make them more representative, they were further processed to introduce random distortions that reduced regularity of shape and diminished symmetries. Refer to López (1999) for a detailed account of the generation and distortion process for the problem suite used in our tests.

The data set contains files that generate test problems in 5, 10, and 20 dimensions and seven cardinalities: 250, 500, 1,000, 2,000, 5,000, 10,000, and 20,000. We say an object has low density when its extreme elements are up to 15% of the cardinality of the set; medium density corresponds
to objects with between 15% and 35% of extreme elements; and we classify objects with more than 35% of extreme elements as high density objects. The combination of three dimensions, seven cardinalities, and 3 basic density categories results in 63 test problems.

A “run” in this experiment consisted of the execution of either Naive* or PolyFrame using as input a “points” file and a “vectors” file. The output of each run was the frame $\Phi = \{F_1, F_2\}$. It was verified in every case that the two algorithms independently agreed exactly on the identification of the frames. Each run was repeated three times and we report the average of the three readings. No LP “hot-start” procedure was used since our interest focused on comparing the performance of the pure algorithms. For the same reason, no presorters, preprocessors, or other opportunistic schemes were applied. We may assume that such enhancements have a similar impact on the two procedures and therefore do not affect the comparison.

The relevant information about the data files used in our experiment along with the complete data sets analyzed here can be found in L´opez (1999).

Our Naive* results are quite predictable given what is known about solving LPs and what has been tried in finding frames of cones and polytopes with these types of methods. Figure 2.a shows how the average time required by Naive* to solve individual LPs in a frame problem behaves almost exactly linearly with respect to cardinality. Since a complete frame study requires an LP for each data point, total time is a factor of the square of the number of LPs solved. That is, the relation between time, $t$, and cardinality, $n$, is roughly $t = cn^2$, where $c$ is a factor that depends mainly on dimension and density and a certain overhead. This relation is confirmed in our tests and is depicted in Figure 2.b. Our graph is based on a problem in 5 dimensions with a density of about 5%. The factor, $c$, would be unaffected by density in a pure (unenhanced) naive implementation. Clearly the hardware and software used will affect this factor.

Figures 3 and 4 present selected comparisons between Naive* and PolyFrame using data from our experiments. Figure 3 helps understand the two extremes in the comparison of total execution times between the two algorithms. Figure 3a shows all test problems in 20 dimensions and high frame density. It was selected because it includes the case when the difference is least stark. This occurred at cardinality 250 where PolyFrame solved the problem in around 62% of the time it
took Naive*. The difference in total times was most accentuated when the generating set had the highest cardinality (20,000 generators) in dimension 20 and low frame density (4%). For this problem PolyFrame required around 7% the Naive* time. Figure 3b depicts the comparison for all cardinalities in 20 dimensions and low density.

Figure 4a provides insight into the impact of density on the relative performance of the two algorithms. It compares a typical case of ten dimensions at the highest cardinality: 20,000 generators. Almost all other cases will result in similar pictures. The figure shows how, as the density classification goes from ‘High’ to ‘Medium’ to ‘Low,’ the advantage of PolyFrame over Naive* steadily increases from $2.5 \times$ to $4 \times$ to $13 \times$ faster.

The data shows that PolyFrame improves with respect to Naive* as density decreases and the number of generating elements in the polyhedron increases. The reason why PolyFrame is better than Naive* is that the former solves small linear programs that grow as new frame elements are identified but are never larger than the cardinality of the frame. On the other hand, Naive* begins solving large linear programs that become progressively smaller. This difference becomes accentuated in problems with low density since, in this case, the linear programs solved by PolyFrame remain small.

Figure 4b elucidates how inner products affect the performance of PolyFrame. The figure depicts how much time PolyFrame spends calculating inner products compared with the time solving LPs as the cardinality of the data sets increases. When the number of elements is 250, PolyFrame uses around 4.5% of total time calculating inner products. As the number of elements in the polyhedron increases this percentage increases as well. For 20,000 elements, PolyFrame requires almost 11.50% of total time to calculate inner products. It is evident from this that inner products do not play an important role in explaining the difference in performance between Naive*
and PolyFrame. Since both procedures solve the same number of LPs, the difference is due to their sizes.

One potential application of the new procedure is Data Envelopment Analysis (DEA). The analysis in DEA requires the identification of “efficient” points and this involves identifying the frame of a finitely generated polyhedron with a recession cone that corresponds to one of the orthants in the space (see Dula and Thrall (2001)). A source of real DEA data has been compiled from the Report of Condition and Income publications of the Federal Financial Institutions Examination Council which contains yearly data about U.S. commercial banks. This data is also used in the articles by Berger, Hancock, and Humphrey (1993), Berger and Mester (1997), and Akhavin, Berger, and Berger (1997). From this data set we created three test problems:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Cardinality: $n$</th>
<th>Dimension: $m$</th>
<th>Frame Density: $d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,971</td>
<td>7</td>
<td>0.9%</td>
</tr>
<tr>
<td>2</td>
<td>12,456</td>
<td>8</td>
<td>1.08%</td>
</tr>
<tr>
<td>3</td>
<td>19,939</td>
<td>11</td>
<td>6.18%</td>
</tr>
</tbody>
</table>

The first problem is data for one of the years (2000) in the data set. Problems 2 and 3 are aggregates for three (1989, 1990, 1991) and four (1997, 1998, 1999, 2000) years respectively. Note how by increasing $m$ we are able to affect the frame density $d$. The point matrices, $A$, are fully dense (i.e., opposite of sparse). The results from these experiments are summarized in Figure 5†. As with our results with synthetic data, PolyFrame’s superiority is evident with real data as well. The most dramatic contrast between PolyFrame and Naive* occurs when the density and dimension are relatively low (Problem 2: $m = 8, d = 1.08$%); here PolyFrame is two orders of magnitude faster than Naive*. When these two factors increase, as occurs when going from Problem 2 to Problem 3, PolyFrame’s dominance compared to Naive* is less dramatic. This

† These tests were performed on a different machine than that used in the experiments with the simulated data and therefore the times cannot be used in comparisons with the other results in this article.
Problem 1: \( n=4,971, m=7, d=0.9\% \).

Problem 2: \( n=12,456, m=8, d=1.08\% \).

Problem 3: \( n=19,939, m=11, d=6.18\% \).

![Naive* vs PolyFrame on Banking Data.](image)

Figure 5.

The adverse effect of increased dimension and frame density is predicted by the results using simulated data. These tests show, however, that the procedures may behave less predictably when real data is involved. PolyFrame's relative performance is not as expected when going from Problem 1 to Problem 2. The second problem is roughly in the same category in terms of dimension and frame density and yet the increase in cardinality does not translate into an increase in relative performance with respect to Naive*.

We can conclude from all this that PolyFrame is almost always faster than Naive* for finding the frame of a finitely generated polyhedral hull. This difference becomes increasingly dramatic as the cardinality of the points increases and the frame "density" (proportion of frame elements to the total cardinality of the data set) decreases although we should be cautious about precise predictions when real data is involved. Indeed, in practice, applications tend to involve problems with small frame densities. The difference is almost entirely due to the fact that PolyFrame uses small LPs the number of columns of which never grow larger than the cardinalities of the frame of the polyhedron, whereas, Naive* involves LPs that start as large as the number of elements in the data set and then progressively grow smaller.

9. Concluding remarks. The frame problem is a fundamental problem in computational geometry, statistics, operations research, and, since it detects geometrical outliers in multi-dimensional data, it also has applications in data mining. We have studied finitely generated unbounded polyhedral sets described by the combination of a convex and a conical hull. This research is an extension and generalization of previous work on finding minimal cardinality subset of generating elements (i.e., frames) for convex and conical hulls which, until now, have been treated separately. We have presented two specialized algorithms, Naive* and PolyFrame, for finding frames of the resultant unbounded polyhedral set directly in the space where the object resides.

Procedure Naive* is based on a direct application of the definitions of the polyhedral sets and the property that frames are composed exclusively and entirely of extreme elements. Procedure PolyFrame applies the idea of gradually and systematically "erecting" the frame by generating a
sequence of nested hulls. **PolyFrame** required the formulation of a new and specialized LP with properties of universal feasibility, identification of external points, and availability of strictly separating hyperplanes. The two algorithms were validated and tested on a large and comprehensive problem suite.

Fukuda (2000) notes that applying naive-type procedures such as **Naive** to solve the frame problem “might end up in a very time consuming job for large $n$ (say $n > 1000$).” Our experiments attest to this. The new procedure, **PolyFrame**, however, extends the scale of frame problems that can be solved substantially. Our results show that **PolyFrame** is always faster than **Naive**. Sometimes the performance of the former is one order of magnitude better in terms of times. This occurs when size is large and frame density low. This makes **PolyFrame** particularly interesting since most applications fall into this category.

List of References.


