Basis Enumeration of Hyperplane Arrangements up to Symmetries

by

Aaron Moss

Bachelor of Computer Science, UNB, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s):	David Bremner, BSc, MSc, PhD, Computer Science
Examining Board:	Bradford Nickerson, BScE, MScE, PhD, PEng, Computer Science, Chair
	Eric Aubanel, BSc, PhD, Computer Science
	Andrew Gerber, BScE, PhD, BA, PEng, Mechanical Engineering

This thesis, dissertation or report is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

September, 2012

©Aaron Moss, 2012

Dedication

To Dr. Bremner and my examining committee, who have done me a major favour by reviewing this so quickly; and to my wife, who has had to attempt to explain "Basis Enumeration of Hyperplane Arrangements up to Symmetries" to other people for the last year.

Abstract

This thesis details a method of enumerating bases of hyperplane arrangements up to symmetries. I consider here automorphisms, geometric symmetries which leave the set of all points contained in the arrangement setwise invariant. The algorithm for basis enumeration described in this thesis is a backtracking search over the adjacency graph implied on the bases by minimum-ratio simplex pivots, pruning at bases symmetric to those already seen. This work extends Bremner, Sikirić, and Schürmann's method for basis enumeration of polyhedra up to symmetries, including a new pivoting rule for finding adjacent bases in arrangements, a method of computing automorphisms of arrangements which extends the method of Bremner *et al.* for computing automorphisms of polyhedra, and some associated changes to optimizations used in the previous work. I include results of tests on ACEnet clusters showing an order of magnitude speedup from the use of C++ in my implementation, an up to 3x speedup with a 6-core parallel variant of the algorithm, and positive results from other optimizations.

Acknowledgements

This research was partially supported by NSERC grants to both myself and Dr. Bremner; computational resources used in the experimental work were provided by the ACEnet consortium. I am also indebted to David Avis for his `lrslib` linear programming software and Thomas Rehn for his `permlib` group theory library, both of which were instrumental in the successful and swift completion of this project. I would also like to recognize Dr. Tariq Hasan of the UNB Applied Statistics Centre for his assistance with the statistical analysis of my data.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Though hyperplane arrangements are fundamental geometric structures with applications in many fields, in moderate to high dimensions they do not lend themselves easily to efficient programmatic manipulation. One potential application for software which can quickly find bases of hyperplane arrangements is computation of the vector partition function of a matrix, a fundamental operation in parametric integer programming and representation theory. Bases of hyperplane arrangements are equivalent to bases of systems of linear equations (minimal subsystems defining zero dimensional solutions). Basis enumeration of systems of linear equations is necessary for dual-type generating function approaches to computing the vector partition function, which research by Brion, Szenes, and Vergne [7, 22] suggests may be quicker than current approaches.

This thesis describes the design, implementation, and analysis of a program for basis enumeration of hyperplane arrangements up to symmetries, as well as the results of applying various programmatic and mathematical optimizations to this program. This program, called `Basil` (for "**Basi**s **l**ist") adapts the pivoting method of Bremner, Sikirić, and Schürmann [6] for basis enumeration of polyhedra (a closely related geometrical structure) up to symmetries, using a new pivot selection method to traverse hyperplane arrangements instead of polyhedra. The work of Bremner *et al.* is in turn related to the reverse-search method for basis enumeration of Avis and Fukuda [4] and earlier pivoting methods *e.g.* [9].

Sikirić's `Polyhedral` [21] and Rehn's `Sympol` [19] implement other solutions to the related problem of vertex enumeration up to symmetries of polyhedra; the approaches taken by both Sikirić and Rehn involve recursively decomposing the polyhedron into less complex polyhedra, and are quite different from the pivoting approach which is used in `Basil`. For a survey of approaches for vertex enumeration up to symmetries of polyhedra and the dual problem of facet enumeration up to symmetries, see [6], which covers both the recursive decomposition and pivoting approaches.

Chapter 2

Background

2.1 Arrangements and Polyhedra

2.1.1 Definitions and Properties

The structures considered in this thesis exist in d-dimensional real space, $\mathbb{R}^d$. A *point* $\boldsymbol{x}$ in $\mathbb{R}^d$ is the ordered list of *coordinates* $\boldsymbol{x} = [x_1\ x_2\ \cdots\ x_d]$, where each of the x_i is a real number. In this thesis, however, all explicitly defined vectors have rational coordinates, as it is not possible to exactly represent the real numbers in a computer system, but the rationals are closed under all operations used, and can be efficiently and easily represented in a digital computer. A *hyperplane* H is the set of points $\boldsymbol{x} \in \mathbb{R}^d$ which satisfy a linear equation $\boldsymbol{a}^\top \boldsymbol{x} = b, \boldsymbol{a} \in \mathbb{R}^d, b \in \mathbb{R}$, while a *hyperplane arrangement* $\mathcal{A}$ is the union of the points contained in a set of hyperplanes; the hyperplanes in the arrangement are typically indexed as $A_1, A_2, \cdots, A_n$. A *polyhedron* $\mathcal{P}$ is a closely related structure, the intersection of a set of *halfspaces* indexed as $P_1, P_2, \cdots, P_n$; a halfspace is the set of points $\boldsymbol{x} \in \mathbb{R}^d$ that satisfy a linear inequality $\boldsymbol{a}^\top \boldsymbol{x} \geq b$, $\boldsymbol{a}$ and b defined as above. The hyperplane for which this inequality is satisfied with equality is known as the *bounding hyperplane* of the halfspace, while the set of bounding hyperplanes of the halfspaces defining a polyhedron is called its *bounding hyperplane arrangement.* Complementing the concept of the bounding arrangement is the *interior* $\text{int}(\mathcal{P})$ of a polyhedron $\mathcal{P}$, the set of points in $\mathcal{P}$ which are not on the bounding arrangement of $\mathcal{P}$. The *size* of an arrangement is the number of hyperplanes n, while the *dimension* of an arrangement is the dimension of the smallest affine subspace containing the arrangement. Most arrangements discussed here are full rank and thus have dimension d, the dimension of the underlying space, and n is used throughout this thesis to denote the number of hyperplanes comprising the arrangement under consideration. Size and dimension of polyhedra are defined analogously.

A *cobasis* $\mathbf{B}$ of a hyperplane arrangement is a set of indices of d hyperplanes which intersect at a single point, a *vertex* of the arrangement. I use here the name *cobasis* from linear programming for what is typically called a *basis* in geometry for consistency with the terminology of my linear programming-based implementation. Figure 2.1 shows the cobases and vertices of an example arrangement. It should be noted that d hyperplanes meet in a single point if and only if the equations defining those hyperplanes are linearly independent. Hyperplanes which contain a vertex are said to be *incident* to

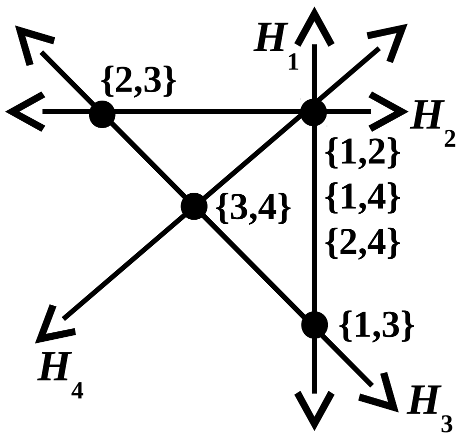


Figure 2.1: An arrangement in $\mathbb{R}^2$; dots denote vertices, cobases corresponding to each vertex are shown in set notation.

that vertex. Vertices of polyhedra may be defined as those vertices of the polyhedron's bounding hyperplane arrangement which are contained within the polyhedron, and cobases of a polyhedron as the cobases of the bounding arrangement which correspond to those vertices. A vertex of an arrangement or polyhedron may be defined by more than one cobasis in the case where more than d hyperplanes of the (bounding) arrangement meet at that point; such a vertex is called *degenerate*. Polyhedra and arrangements with no degenerate vertices are called *simple*; non-simple polyhedra and arrangements are *degenerate*.

Polyhedra, by virtue of being intersections of halfspaces, have the property of being *convex*; that is, for any two points $\boldsymbol{p}$ and $\boldsymbol{q}$ contained in a polyhedron $\mathcal{P}$, every point on the line segment $\overline{\boldsymbol{pq}}$ is also contained in $\mathcal{P}$. This property of convexity yields an alternate representation of a polyhedron $\mathcal{P}$ as the set of points $\boldsymbol{x}$ whose coordinates are linear combinations of the coordinates of the vertices $\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_k$ and *extreme rays* $\boldsymbol{v}_{k+1}, \boldsymbol{v}_{k+2}, \cdots, \boldsymbol{v}_m$ of $\mathcal{P}$ that satisfy $\boldsymbol{x} \in \{\sum_{i=1}^{m} \lambda_i \boldsymbol{v}_i : \sum_{i=1}^{k} \lambda_i = 1, \lambda_i \geq 0 \, \forall i\}$. If $k = m$, then $\mathcal{P}$ is bounded, and is called a *polytope*. This representation is called the *vertex representation* or *V-representation* of a polyhedron, while the previously described representation as an intersection of halfspaces is called the *halfspace representation* or *H-representation*.

A useful representation for arrangements is *homogeneous coordinates*, where an arrangement $\mathcal{A}$ in $\mathbb{R}^d$ is represented as a set of vectors in $\mathbb{R}^{d+1}$. To convert an arrangement to homogeneous coordinates, take each hyperplane $\boldsymbol{a}_i^\top \boldsymbol{x} = b_i$ in the arrangement, and produce a vector $\boldsymbol{v}_i = [b_i \; a_{i,1} \; a_{i,2} \; \cdots \; a_{i,d}]$. The $\boldsymbol{v}_i$ are the normal vectors to an arrangement $\mathcal{A}'$ in $\mathbb{R}^{d+1}$, and the original arrangement $\mathcal{A}$ is the intersection of $\mathcal{A}'$ with the $x_0 = 1$ hyperplane ($b_i = 1$ in the original coordinates). The arrangement $\mathcal{A}'$ in homogeneous coordinates has only one vertex, the origin, which is sometimes a useful property.

The *basis enumeration* problem for a polyhedron or arrangement $\mathcal{X}$ is to list all unique cobases of $\mathcal{X}$. If $\mathcal{X}$ is simple, the *vertex enumeration* problem, reporting each unique vertex, is equivalent to the basis enumeration problem. If $\mathcal{X}$ is degenerate, the vertex enumeration problem can be also solved by basis enumeration, though some method must be employed to filter out duplicate vertices. Solving the vertex enumeration problem for a polyhedron converts that polyhedron from H-representation to V-representation.

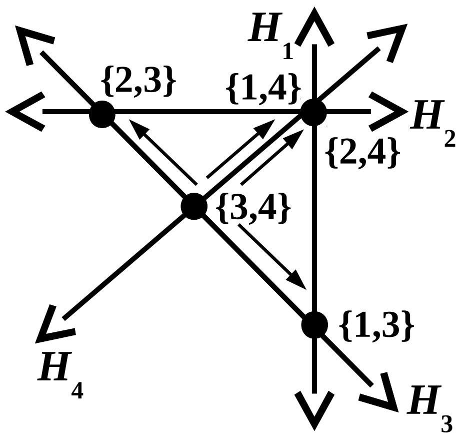


Figure 2.2: The arrows point to the cobases adjacent to $\{3, 4\}$.

The cobases of a polyhedron or hyperplane arrangement can be considered as the nodes of an implicit graph, where two cobases $\mathbf{B}_1$ and $\mathbf{B}_2$ are *adjacent* if they differ only by one element, that is, letting $\mathbf{B} = \mathbf{B}_1 \cap \mathbf{B}_2$, $\mathbf{B}_1 = \mathbf{B} \cup \{p\}$ and $\mathbf{B}_2 = \mathbf{B} \cup \{q\}$; here the $d - 1$ hyperplanes defining $\mathbf{B}$ intersect in a 1-dimensional line, an *edge* of the arrangement. Figure 2.2 illustrates adjacent cobases.

One further property of hyperplanes worth noting is that, being defined by an equation of the form $\boldsymbol{a}^\top \boldsymbol{x} = b$, the coefficients $a_1, a_2, \cdots, a_d$ and constant term b can be multiplied by any nonzero value k without changing the set of points included in the hyperplane. Halfspaces may be similarly scaled, with the added condition that the scale factor k be positive, as a negative k would reverse the inequality, producing the *complement* of the original halfspace (the union of the set of points not in the halfspace with the bounding hyperplane of that halfspace). To normalize the representation of halfspaces and hyperplanes in this thesis, either b or the first non-zero coefficient a_i is scaled to equal one. This scaling aids representation of the arrangement in homogeneous coordinates, as the coordinates of the intersections with the $x_0 = 1$ hyperplane may be derived without rescaling.

2.1.2 Linear Programming and the Simplex Method

A certain class of optimization problem involves finding a vertex $\boldsymbol{v}$ of a polyhedron which maximizes a linear objective function defined by a vector $\boldsymbol{c} = [c_1 \; c_2 \; \cdots \; c_d] \in \mathbb{R}^d$ as $c(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x}$. The field of *linear programming* has developed to solve this and related problems, with some of these related problems being defined on hyperplane arrangements. One of the oldest and most studied approaches to linear programming, the simplex method pioneered by Dantzig [10], is to find an initial cobasis and then repeatedly move (or *pivot*) to some adjacent cobasis corresponding to a vertex $\boldsymbol{v}'$ with an objective value $c(\boldsymbol{v}')$ at least as good as the objective value of the current vertex. This process is repeated, proceeding until either a cobasis of an optimal vertex is reached or it can be seen that no such optimal vertex exists. The fundamental data structure of the simplex method is the *simplex tableau*, $\mathrm{T}(\mathcal{P}, \mathbf{B})$, which re-expresses the linear inequalities defining a polyhedron $\mathcal{P}$ in terms of a cobasis $\mathbf{B}$. The tableau data structure is quite easily generalized to represent a hyperplane arrangement $\mathcal{A}$, so the discussion here will describe tableaux for arrangements. The first step to convert an arrangement to tableau form is to arbitrarily choose an "interior" side for each hyperplane (for polyhedra, the interior side of the bounding hyperplane of each halfspace is the side contained in the halfspace). After an interior side has been chosen for each of the n hyperplanes, n new *slack variables* $\{x_{d+1}, x_{d+2}, \cdots, x_{d+n}\}$ are added to the existing *decision variables* $\{x_1, x_2, \cdots, x_d\}$ which define points in $\mathbb{R}^d$. The slack variables represent the "distance" from each hyperplane in the arrangement to the point represented by the tableau; these distances have different values depending on the scaling of the input, but are always zero when the point is on the hyperplane, more positive the farther on the interior side of the hyperplane the point is, and more negative the farther on the opposite ("exterior") side of the hyperplane the point is. The slack variables are therefore always kept non-negative by the simplex algorithm when dealing with polyhedra, though when simplex tableaux are used to represent hyperplane arrangements the slack variables may be either positive or negative, as points in an arrangement may be on either side of any hyperplane in the arrangement. To add the slack variables, the equation $\boldsymbol{a}_i^\top \boldsymbol{x} = b_i$ defining each of the n hyperplanes A_i in $\boldsymbol{\mathcal{A}}$ is rewritten as $x_{d+i} = -b_i + \boldsymbol{a}_i^\top \boldsymbol{x}$, defining a matrix $A_{n \times d} = (a_{i,j})$ ($a_{i,j}$ being the j-th element of $\boldsymbol{a}_i$) and a vector $\boldsymbol{b} = [b_1\ b_2\ \cdots\ b_n]^\top$. These components are combined with the vector $\boldsymbol{c}$ defining the objective function in a matrix as follows, defining the initial simplex tableau[1]:

$$M = \begin{bmatrix} 0 & \boldsymbol{c}^{\top} \\ -\boldsymbol{b} & A_{n\times d} \end{bmatrix}$$

The row variables for a simplex tableau M are called *basic* variables, and are stored in a vector $\boldsymbol{x}_B$ initially valued $[z\ x_{d+1}\ x_{d+2}\ \cdots\ x_{d+n}]$, where z is a dummy variable for the objective row. The column variables are called *cobasic* variables[2], and are stored in a vector $\boldsymbol{x}_N$ initially valued $[x_0\ x_1\ x_2\ \cdots\ x_d]$; x_0 represents the constant terms b_i, as in the discussion of homogenized coordinates in Section 2.1.1. With these two vectors defined, the initial system of equations defining the arrangement $\boldsymbol{\mathcal{A}}$ can be written as $\boldsymbol{x}_B^{\top} = M\boldsymbol{x}_N$. The values of the cobasic variables of a simplex tableau are assumed to be zero, so that the value of any basic variable (or the objective function in the first row) can be read off from the constant term in the first column of M. After the initial setup of the tableau is complete, the decision variables are moved into the basis $\boldsymbol{x}_B$, with slack variables replacing them in the cobasis $\boldsymbol{x}_N$. When this process is completed, the tableau represents a basis of the arrangement, and the current vertex can be read off from the values of the decision variables in the first column.

In the context of linear programming, a *pivot* from a cobasis $\mathbf{B}_1$ to another

[1]The objective row $[0\ \boldsymbol{c}^{\top}]$, though integral to the simplex method of linear programming, is largely ignored for the solution to the basis enumeration problem described in this thesis, and $\boldsymbol{c}$ is set to a value of $\mathbf{1}$, a vector of all ones.

[2]Note that the cobasic variables of a simplex tableau, not the basic, correspond to a basis of the represented polyhedron or arrangement in the usual geometric definition.

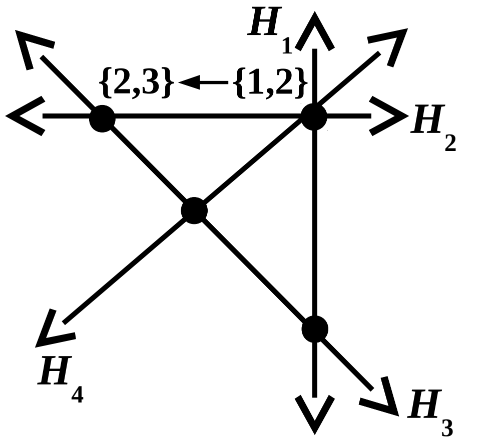


Figure 2.3: A pivot from cobasis $\{1, 2\}$ to $\{2, 3\}$.

adjacent cobasis $\mathbf{B}_2$ ($\mathbf{B}_1 = \mathbf{B}_2 \cup \{x_e\} \backslash \{x_l\}$) exchanges the *entering* slack variable, x_e for the *leaving* slack variable[3], x_l, traversing an edge of the arrangement or polyhedron. This pivot is accomplished by rewriting the equation $x_l = -b_l + \boldsymbol{a}_l^\top \boldsymbol{x}_N$ to isolate x_e (one of the elements of $\boldsymbol{x}_N$) on the left hand side, then substituting this modified equation into each other row of M to replace the x_e terms in those equations with new x_l terms.

As an example, consider the simplex tableau $\mathrm{T}(\mathcal{H}, \{1, 2\})$, which represents the cobasis $\{1, 2\}$ of the arrangement $\mathcal{H}$ pictured in Figure 2.3. The equations

[3]The variables are "entering" and "leaving" the linear programming basis, the complement of the cobasis.

for T are as follows:

$$\begin{array}{rcrcrcr} x_1 & = & 1 & + & x_3 & & \\ x_2 & = & 1 & & & + & x_4 \\ x_5 & = & 2 & + & x_3 & + & x_4 \\ x_6 & = & 0 & + & x_3 & - & x_4 \end{array}$$

If x_5 is chosen as leaving slack variable, and x_3 is chosen as the entering slack variable, the pivot performed will be from the cobasis $\{1, 2\}$ to the adjacent cobasis $\{2, 3\}$. To perform this pivot, the third equation is re-written with x_3 on the left hand side as $x_3 = -2 + x_5 - x_4$, and this modified equation is substituted into the remaining rows, resulting in the following tableau equations after the pivot:

$$\begin{array}{rcrcrcr} x_1 & = & -1 & + & x_5 & - & x_4 \\ x_2 & = & 1 & & & + & x_4 \\ x_5 & = & -2 & + & x_5 & - & x_4 \\ x_6 & = & -2 & + & x_5 & - & 2x_4 \end{array}$$

Many pivot rules used in the simplex method are based on the idea of the *minimum ratio test*. Geometrically, this test can be thought of as leaving one basis and sliding along an edge of a polyhedron or hyperplane arrangement until the first new (bounding) hyperplane is reached, forming a new basis. In a simplex tableau, distance from each hyperplane is represented by its associated slack variable, and moving from one hyperplane to another (equivalently, moving to an adjacent cobasis) is accomplished by allowing one cobasic variable to become non-zero while forcing some basic variable to zero. For a given pair x_e and x_l of cobasic and basic variables, the ratio between the value b_l of the leaving variable x_l and the coefficient $a_{l,e}$ of the entering variable x_e in the leaving variable's equation determines how much the entering variable can be increased or decreased while not changing the sign of x_l, which would move the current point across the hyperplane A_l in the arrangement. Since the cobasic variables in a simplex tableau are assumed to be zero, $x_l = -b_l + a_{l,e}x_e$; by setting x_e to $r = b_l/a_{l,e}$, x_l is then forced to zero. In polyhedra leaving and entering variables must be chosen such that $r \geq 0$, as slack variables cannot be negative, but this restriction does not hold for arrangements. Selecting x_l such that the absolute value of r is minimized (the "minimum ratio") finds the nearest adjacent cobasis to pivot to; if $r = 0$ (due to $\boldsymbol{b}_l = 0$), the pivot is *degenerate*, and represents a move to another cobasis of the same vertex (a degenerate vertex, as it has at least two distinct cobases).

2.2 Symmetries

2.2.1 Groups and Permutations

Mathematically, a *group* $(G, \bullet)$ is a set G of *elements* augmented with an *action* $\bullet$ which is a function of two elements of G which returns a group element; typical notation would be $\alpha \bullet \beta = \gamma$, for $\alpha, \beta, \gamma \in G$. Often, where the group action is clear from context, the group will be written as G and the $\bullet$ omitted in expressions (*e.g.* $\alpha\beta = \gamma$). Group elements are typically written in the opposite order to which they are applied, much like function composition. There is no requirement that the group action be commutative ($\alpha\beta$ is not necessarily equal to $\beta\alpha$), though it must be associative ($\forall \alpha, \beta, \gamma \in G, (\alpha\beta)\gamma = \alpha(\beta\gamma)$). The set of elements G must also be closed under the group action ($\forall \alpha, \beta \in G, \alpha\beta \in G$). All groups have an *identity* element ϵ such that $\forall \alpha \in G, \alpha\epsilon = \alpha = \epsilon\alpha$; each element $\alpha \in G$ also has a unique *inverse* α^{-1} such that $\alpha\alpha^{-1} = \epsilon = \alpha^{-1}\alpha$. A *subgroup* $(H, \bullet)$ of a group $(G, \bullet)$ is a group consisting of a set $H \subseteq G$ which is closed under the group action $\bullet$. It is often useful to represent a group G more compactly than by enumerating all its elements; in this case a *generating set* S may be found for G. S is a generating set for G (the elements of S are referred to as *generators*) if all the elements of G may be represented as combinations of elements of S or inverses of elements of S under the group action[4]. G generates itself trivially, but smaller generating sets are possible (*e.g.* $\{1\}$ generates $(\mathbb{Z}, +)$, the integers under addition, as does $\{2, 3\}$).

The groups dealt with most extensively in this thesis are *symmetry groups*. The elements of these groups are transformations such as rotations and reflections which may be applied to a given geometric structure while leaving it unchanged as a whole. As an example, consider the square in $\mathbb{R}^2$: rotating it about its center by any multiple of $90°$ will leave it unchanged, as will

[4]For a *finite group* $(G, \bullet)$, a group where G is a finite set, $\forall \alpha \in G\, \exists n, \alpha^n = \alpha^{-1}$, so we can say more concisely that all elements of G may be represented as combinations of elements of S. The groups dealt with in this thesis are all finite.

reflection about its horizontal, vertical, or diagonal axes. Furthermore, if the vertices of the square are labelled, then it can be seen that, for instance, two 90° rotations combine to form a 180° rotation, as does a vertical reflection followed by a horizontal reflection. The *image* of a vertex $\boldsymbol{x}$ under a symmetry α is the vertex to which $\boldsymbol{x}$ is mapped by α. In this setting, the *orbit* of a given vertex $\boldsymbol{v}$ under the symmetry group (denoted $\mathrm{Orb}_G(\boldsymbol{v})$) is the set of images of $\boldsymbol{v}$ under each element of the group. Orbits under a group G are an equivalence relation, partitioning the set X they act on, as shown in Theorem 2.2.1. For the square under its full symmetry group, the orbit of any vertex is all vertices of the square (by 0, 90, 180 and 270° rotations), but if the symmetry group containing only the identity symmetry and reflection about the square's vertical axis is considered, the top two vertices form one orbit, and the bottom two another.

Theorem 2.2.1. *Given a group G and a set X of objects acted on by that group, the orbit membership relation $\equiv_G = \{(x, y) : x, y \in X, y \in \mathrm{Orb}_G(x)\}$ is an equivalence relation.*

Proof. By definition, an equivalence relation is reflexive, symmetric, and transitive. Any element x of X is in its own orbit by the identity element of G ($x = \epsilon(x) \in \mathrm{Orb}_G(x)$), therefore $\equiv_G$ is reflexive. For any $y \in \mathrm{Orb}_G(x)$, $y = \alpha(x)$ for some $\alpha \in G$; it follows by inverses that $x = \alpha^{-1}(y) \in \mathrm{Orb}_G(y)$, thus $\equiv_G$ is symmetric. For any $z = \beta(y) \in \mathrm{Orb}_G(y)$, $\beta \in G$, $z = (\beta\alpha)(x)$. Since $\beta\alpha \in G$ by group closure, $z \in \mathrm{Orb}_G(x)$, showing $\equiv_G$ to be transitive as well. □

Another sort of group that is used in this thesis is the *permutation group*. A *permutation* is a function which takes a list of n elements, typically represented by the numbers 1 through n, and reorders them. The typical notation for a permutation σ is *cycle notation*, which lists the orbits of the numbers under the permutation as $(x_1\ \sigma(x_1)\ \sigma^2(x_1)\ \cdots\ \sigma^{-1}(x_1))\cdots(x_m\ \sigma(x_m)\ \sigma^2(x_m)\ \cdots\ \sigma^{-1}(x_m))$, omitting those orbits with only one element. As an example, the permutation (1 2 3)(5 6) maps 1 to 2, 2 to 3, and 3 to 1 and switches (*transposes*) 5 and 6, leaving 4 unmoved.

2.2.2 Automorphisms and Isometries

Many interesting hyperplane arrangements have a significant number of *automorphisms*: geometric symmetries which leave the points in the arrangement setwise invariant. These symmetries can also be considered as permutations of the list of hyperplanes included in the arrangement; automorphisms of polyhedra can be considered analogously, as either geometric transformations leaving the polyhedron setwise invariant, or permutations of the list of halfspaces defining the polyhedron. Given a group G of some set of these symmetries acting on a hyperplane arrangement, the problem of *basis enumeration up to symmetries* is listing exactly one cobasis from each orbit under the action of G. The symmetries this thesis is particularly concerned with are *isometries*, distance preserving symmetries.

2.2.3 Inner Products

Distance in $\mathbb{R}^d$ is defined by an *inner product*; an inner product on $\mathbb{R}^d$ is a map $\langle \cdot, \cdot \rangle : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ which satisfies the following conditions for all $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^d, a \in \mathbb{R}$:

1. *Symmetry*: $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle$
2. *Linearity*: $\langle a\boldsymbol{x}, \boldsymbol{y} \rangle = a\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ and $\langle \boldsymbol{x} + \boldsymbol{z}, \boldsymbol{y} \rangle = \langle \boldsymbol{x}, \boldsymbol{y} \rangle + \langle \boldsymbol{z}, \boldsymbol{y} \rangle$
3. *Positive-definiteness*: $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$, $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = 0$ iff $\boldsymbol{x} = \boldsymbol{0}$

Given an inner product, the *norm* of a vector $\boldsymbol{x}$ is defined as $\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$; the norm can be thought of intuitively as the "length" of the vector, as it is zero only for the zero vector, and positive otherwise. In this sense, the distance $d(\boldsymbol{p}, \boldsymbol{q})$ between two points $\boldsymbol{p}$ and $\boldsymbol{q}$ is $\|\boldsymbol{p} - \boldsymbol{q}\|$, defining a distance metric, and the angle θ between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined by the well-known formula $\cos\theta = \frac{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}{\|\boldsymbol{x}\| \ \|\boldsymbol{y}\|}$.

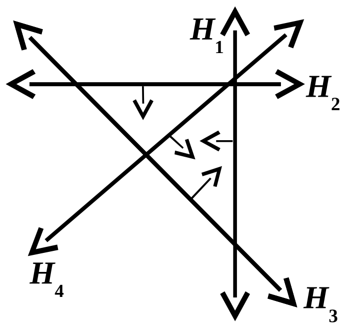
2.2.4 Gram Matrices

One property of isometric cobases is that, given some distance metric, the multiset of angles according to that metric between each hyperplane in the cobasis and all of the other hyperplanes in that cobasis is invariant under the action of any isometry. To use this property, the angles between all pairs of hyperplanes can be precomputed, and then each distinct angle can be represented by a unique integer. The *Gram matrix* $G = (g_{i,j})$ is constructed such that $g_{i,j}$ is the value corresponding to the angle between the hyperplanes indexed i and j. Gram matrices for polyhedra can be similarly constructed with respect to the angles between the bounding hyperplanes of the polyhedron. A submatrix of a Gram matrix uniquely representing the angles between pairs of hyperplanes in a given cobasis can be constructed by selecting only the elements in the rows and columns of the Gram matrix corresponding to the indices of the cobasic hyperplanes. If each row of this submatrix is sorted, and then the rows of the submatrix are lexicographically sorted, the resulting submatrix represents the angles between each pair of hyperplanes in the cobasis such that a test for matrix equality suffices to check equality of the multisets of angles for each hyperplane in the cobasis. Though inequality of Gram submatrices shows that the corresponding cobases are not symmetric, equality of Gram submatrices is not sufficient to show that the cobases are symmetric. As an example, consider the cube under the subgroup of its full symmetry group generated by the reflections about the cube's vertical and horizontal axes; the cobases on the front and back faces will be in separate orbits, and thus not symmetric, but the angles between each pair of planes in any cobasis are all right angles, so all the multisets will be equal.

An automorphism α of the Gram matrix $G = (g_{i,j})$ of a polyhedron may be defined by a permutation σ of the row and column indices of the matrix as $\alpha(G) = (g_{\sigma(i),\sigma(j)})$ such that $G = \alpha(G)$. Such an automorphism of the

Gram matrix corresponds to an automorphism of the polyhedron produced by permuting by σ the indices of the halfspaces defining the polyhedron. A proof of this can be found in [6], but intuitively the rows and columns of the Gram matrix correspond to the halfspaces defining the polyhedron. As the Gram matrix encodes the angles between each pair of bounding hyperplanes, any transformation which leaves the Gram matrix invariant will also not change the polyhedron, because the relative positions of each of the halfspaces have remained constant. If the Gram matrix is interpreted as the adjacency matrix of a graph, with the elements of the matrix representing colors of the edges, these automorphisms can also be expressed as edge-color preserving graph automorphisms, as shown in Proposition 3.1 of [6].

One problem I encountered in generating Gram matrices for hyperplane arrangements that does not arise in the polyhedral case is that any hyperplane $A = \{\boldsymbol{x} : \boldsymbol{a}^\top \boldsymbol{x} = b\}$ can be replaced by its *negation* $\widetilde{A} = \{\boldsymbol{x} : -\boldsymbol{a}^\top \boldsymbol{x} = -b\}$ without changing the arrangement, a special case of the scaling described in Section 2.1.1. However, the angle between the normal vectors of $\widetilde{A}$ and another hyperplane B is the supplement of the angle between the normal vectors of A and B, in general a different angle. When using the Gram matrix to eliminate non-symmetric cobases, this problem can be solved by simply using a unique up to supplements representation for each angle. This approach does not work when using the Gram matrix to determine the automorphisms of the arrangement, as spurious automorphisms are generated; essentially, these false automorphisms consider a hyperplane to be both itself



$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

Figure 2.4: An example arrangement, along with its constraint matrix. The small rays denote the interior side of each hyperplane.

and its negation simultaneously, causing the arrangement to be warped by some angles between pairs of hyperplanes being replaced by their supplements. If the arrangement is doubled such that each hyperplane is paired with its negation, then matrix automorphisms may replace a hyperplane by its negation by transposing the two in the permutation σ but this warping is prevented from occurring, and correct automorphisms may be derived after reversing the doubling process on the generated permutations. This does, however, quadruple the size of the Gram matrix used for automorphism generation.

As an example, Figure 2.4 shows a 2-dimensional arrangement, along with its constraint matrix. The Gram matrix for this arrangement is below, where the integer representation for angles is the angle in degrees in the range

$(-90, 90]$:

$$\begin{bmatrix} 0 & 90 & 45 & 45 \\ 90 & 0 & 45 & -45 \\ 45 & 45 & 0 & 90 \\ 45 & -45 & 90 & 0 \end{bmatrix}$$

As can be seen from the figure, the only true automorphism of this arrangement is (1 2), switching H_1 and H_2; this corresponds to a reflection across H_4, negating that hyperplane. Yet (1 2) does not correspond to an automorphism of the Gram matrix, as swapping rows (resp. columns) 1 and 2 of the Gram matrix changes the sign of the first and second values of the last column (row) of the matrix. However, constructing a unique up to supplements representation of this Gram matrix by ignoring the signs is also not a feasible option, as that introduces (3 4) as an automorphism of the matrix, even though (3 4) is not an automorphism of the arrangement. The doubling process described above solves both these issues because the pairing of each hyperplane H with its negation $\widetilde{H}$ is effectively a unique up to supplements representation of H which preserves information about the signs of the angles between H and the other hyperplanes.

To generate the Gram matrix I represent a hyperplane $A_i = \{\boldsymbol{x} : \boldsymbol{a}_i^\top \boldsymbol{x} = b_i\}$ of an arrangement $\mathcal{A}$ in homogeneous coordinates by the vector $\boldsymbol{v}_i = [-b_i\ \boldsymbol{a}_i]$. I then construct a map from angles in the range $[0, \pi)$ to the integers, such that supplementary angles have representations that are negations of each other; for this purpose $\cos\theta$ is a good starting point, as it has the desired negation property, and avoids the inverse cosine operation needed to calculate θ from $\cos\theta = \frac{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}{\|\boldsymbol{x}\|\ \|\boldsymbol{y}\|}$. As the square root operation in $\|\boldsymbol{x}\|\ \|\boldsymbol{y}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle \langle \boldsymbol{y}, \boldsymbol{y} \rangle}$ is expensive to calculate exactly, I wish to avoid computing it as well. If all the vectors $\boldsymbol{v}_i$ in the arrangement have the same norm k then the square root operation may simply be omitted, as $k\cos\theta = \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$ is a suitable unique representation for the angle θ between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$, while if the norms are not the same then the function

$$f(\theta) = \frac{\langle \boldsymbol{x}, \boldsymbol{y} \rangle \cdot |\langle \boldsymbol{x}, \boldsymbol{y} \rangle|}{(\|\boldsymbol{x}\|\ \|\boldsymbol{y}\|)^2}$$

uniquely represents θ, but does not require evaluation of a square root to compute. To see this, note that $f(\theta) = \pm(\cos\theta)^2$, the square cancels out the radical in $\|\boldsymbol{x}\|\ \|\boldsymbol{y}\|$, and keeping the sign on the numerator by multiplying by $|\langle \boldsymbol{x}, \boldsymbol{y} \rangle|$ rather than $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ distinguishes between the positive and negative square roots of $(\cos\theta)^2$.

Given rational input, applying one of the above functions for any $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ generates a rational value $r_{i,j}$ which uniquely represents the angle between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ and which satisfies the negation condition for supplementary angles. Since my algorithm only requires a test for equality on these angles, and it performs that test quite frequently, I then construct a map taking each distinct $|r|$ so constructed to a unique nonnegative integer $z(r)$, and store $g_{i,j} = \operatorname{sgn}(r_{i,j}) \cdot z(r_{i,j})$ in the Gram matrix.

The most obvious inner product to use is the Euclidean inner product

$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{d} x_i y_i$, though it is not the only inner product which may be used; one inner product that has functioned well on related problems [6] is based on a transformation defined as $g_{i,j} = \boldsymbol{v}_i^\top Q^{-1} \boldsymbol{v}_j$, where the matrix Q for an arrangement $\mathcal{A}$ is defined as $Q = \sum_{i=1}^{n} \boldsymbol{v}_i \boldsymbol{v}_i^\top$. For a polyhedron $\mathcal{P}$, any automorphism α of $G = (g_{i,j})$ corresponds to a permutation σ of the halfspaces of $\mathcal{P}$, where σ is an automorphism of $\mathcal{P}$. A rigorous proof is provided in Proposition 3.1 of [6], but essentially the multiplication by Q^{-1} maps each vector $\boldsymbol{v}_i$ of the polytope to an equivalent $\boldsymbol{w}_i = R\boldsymbol{v}_i$, where $RR = Q^{-1}$. This is well-defined because Q, being the sum of outer products of the rows of a full-rank matrix, is positive definite, and thus has a positive definite inverse with a unique square root R. The $\boldsymbol{w}_i$ are normalized to the Euclidean norm, as they have the same inner product $k = \langle \boldsymbol{w}_i, \boldsymbol{w}_i \rangle$, as shown in [6]. We saw earlier that this property eliminates the need to normalize the $\boldsymbol{w}_i$, as $k \cos\theta = \langle \boldsymbol{w}_i, \boldsymbol{w}_j \rangle$ is a suitable unique representation for θ, and thus $\langle \boldsymbol{w}_i, \boldsymbol{w}_j \rangle$ can be used to find automorphisms. For hyperplane arrangements the doubling process discussed above can be applied to G to yield valid automorphisms.

2.2.5 Fundamental Domain

A *fundamental domain* $\mathcal{F}$ of an isometry group G is a convex region which contains one representative point from each orbit of points in $\mathbb{R}^d$ under the action of G; for points in the interior of $\mathcal{F}$ there will be exactly one representative, though there may be more representatives of points on the boundary.

A vertex enumeration of a polytope or arrangement $\mathcal{X}$ to the vertices contained in a fundamental domain $\mathcal{F}$ will result in a vertex enumeration up to symmetries of $\mathcal{X}$, assuming $\mathcal{F}$ is constructed so that no vertices of $\mathcal{X}$ fall on the boundaries of the fundamental domain.

A fundamental domain for a group G can be constructed iteratively by starting with the whole of $\mathbb{R}^d$, choosing a *generic point* $\boldsymbol{x}$ (a point that is not its own image under any element of G but the identity element), and taking the intersection of the generated fundamental domain with the halfspace containing $\boldsymbol{x}$ whose bounding hyperplane is the normal bisector of the line segment $\overline{\boldsymbol{xy}}$ between $\boldsymbol{x}$ and each element $\boldsymbol{y} \neq \boldsymbol{x}$ of $\mathrm{Orb}_G(\boldsymbol{x})$. This structure is shown to be a fundamental domain by Theorem 2.2.2, below.

Lemma 2.2.1. *Given a point $\boldsymbol{x}$ and a polyhedron $\mathcal{P}$ such that $\mathcal{P}$ contains $\boldsymbol{x}$ and the bounding hyperplane arrangment of $\mathcal{P}$ is the set of normal bisectors of $\overline{\boldsymbol{xy}}$ for each $\boldsymbol{y} \in Y \subset \mathbb{R}^d, \boldsymbol{x} \notin Y$, a point $\boldsymbol{z}$ is in the interior of $\mathcal{P}$ if and only if $\boldsymbol{z}$ is closer to $\boldsymbol{x}$ than to any point in Y.*

Proof. The normal bisector B of a line segment $\overline{\boldsymbol{xy}}$ can also be expressed as the set of points equidistant from $\boldsymbol{x}$ and $\boldsymbol{y}$ ($B = \{\boldsymbol{b} : d(\boldsymbol{x}, \boldsymbol{b}) = d(\boldsymbol{y}, \boldsymbol{b})\}$); by this definition any point on the side of B containing $\boldsymbol{x}$ is closer to $\boldsymbol{x}$ then $\boldsymbol{y}$. Therefore, any point $\boldsymbol{z}$ in $\mathrm{int}(\mathcal{P})$ is closer to $\boldsymbol{x}$ than it is to any point $\boldsymbol{y} \in Y$ ($d(\boldsymbol{x}, \boldsymbol{z}) < d(\boldsymbol{y}, \boldsymbol{z})$). The converse is true as well; that is, if a point $\boldsymbol{z}$ is closer to $\boldsymbol{x}$ than to any point in Y, than it must be in $\mathrm{int}(\mathcal{P})$, as it must be on the side containing $\boldsymbol{x}$ of each of the bounding hyperplanes of $\mathcal{P}$. □

Theorem 2.2.2. *Given a point $\boldsymbol{x}$ and an isometry group G, and constructing a polyhedron $\mathcal{F}$ containing $\boldsymbol{x}$ with a bounding hyperplane arrangement consisting of the normal bisectors of the line segments $\overline{\boldsymbol{xy}}$ for all $\boldsymbol{y} \neq \boldsymbol{x} \in \mathrm{Orb}_G(\boldsymbol{x})$, $\mathcal{F}$ is a fundamental domain.*

Proof. Take any point $\boldsymbol{u}$ in $\mathbb{R}^d$, and consider the pairs of points $\{(\boldsymbol{u}', \boldsymbol{x}') : \boldsymbol{u}' \in \mathrm{Orb}_G(\boldsymbol{u}), \boldsymbol{x}' \in \mathrm{Orb}_G(\boldsymbol{x})\}$; let $(\boldsymbol{u}^*, \boldsymbol{x}^*)$ be a pair with minimal distance between them, where $s = d(\boldsymbol{u}^*, \boldsymbol{x}^*)$. There exists an isometry α which transforms $\boldsymbol{x}^*$ to $\boldsymbol{x}$, by definition of orbit. Applying α to $\boldsymbol{u}^*$ yields a point $\boldsymbol{u}^\dagger \in \mathrm{Orb}_G(\boldsymbol{u})$ which is distance s from $\boldsymbol{x}$, as isometries preserve distances. Since s is the shortest distance between a member of $\mathrm{Orb}_G(\boldsymbol{x})$ and $\mathrm{Orb}_G(\boldsymbol{u})$, $\boldsymbol{u}^\dagger$ is at least as close to $\boldsymbol{x}$ than to any other point in $\mathrm{Orb}_G(\boldsymbol{x})$. If $\boldsymbol{u}^\dagger$ is strictly closer to $\boldsymbol{x}$ than every other point in $\mathrm{Orb}_G(\boldsymbol{x})$ then it is in the interior of $\mathcal{F}$ by Lemma 2.2.1, but no other member $\boldsymbol{u}^\ddagger = \beta(\boldsymbol{u}^\dagger)$ of $\mathrm{Orb}_G(\boldsymbol{u})$ is in $\mathrm{int}(\mathcal{F})$, as $\boldsymbol{u}^\ddagger$ is distance s from $\boldsymbol{x}^\ddagger = \beta(\boldsymbol{x})$, a shorter distance than $d(\boldsymbol{x}, \boldsymbol{u}^\ddagger)$ by the distance preserving property of isometries, so $\boldsymbol{u}^\ddagger$ is not in $\mathrm{int}(\mathcal{F})$ by Lemma 2.2.1. In the other case, where $\boldsymbol{u}^\dagger$ is equidistant between $\boldsymbol{x}$ and some other $\boldsymbol{x}^\dagger \in \mathrm{Orb}_G(\boldsymbol{x})$ then $\boldsymbol{u}^\dagger$ is on the bisector of $\overline{\boldsymbol{x}\boldsymbol{x}^\dagger}$, part of the bounding hyperplane arrangement of $\mathcal{F}$ by construction; as there is no requirement for points on the boundary of a fundamental domain to be uniquely represented, it suffices to show that $\boldsymbol{u}^\dagger \in \mathcal{F}$, as done here. Finally, $\mathcal{F}$ is convex because it is a polyhedron. □

Chapter 3

Algorithms

3.1 Pivoting Search Algorithm

The essential idea of the algorithm presented here for basis enumeration up to symmetries is to recursively explore the hyperplane arrangement outward from an initial cobasis to its adjacent cobases, pruning this search tree when a cobasis symmetric to one already found is reached; the algorithm will be discussed in more detail below. For proof that this algorithm will report representatives from each orbit of cobases, so long as the adjacency graph of those cobases is connected, see Theorem 3.1.1.

Lemma 3.1.1. *Given an automorphism group G of a hyperplane arrangment $\mathcal{A}$, the set S of cobases adjacent to a cobasis $\mathbf{B}$ of $\mathcal{A}$ on the adjacency graph of cobases of $\mathcal{A}$ is equivalent up to symmetries to the set of cobases adjacent to any cobasis $\mathbf{B}' \in \mathrm{Orb}_G(\mathbf{B})$*

Proof. Consider some cobasis $\mathbf{C}$ adjacent to $\mathbf{B}$; let $\boldsymbol{b}$ be the vertex corresponding to $\mathbf{B}$ and $\boldsymbol{c}$ be the vertex corresponding to $\mathbf{C}$. The $d-1$ hyperplanes in $\mathbf{B} \cap \mathbf{C}$ meet on the edge $\overline{\boldsymbol{bc}}$ of the arrangement. Since all α in G are automorphisms, they preserve hyperplane incidence, so that for each hyperplane H incident to a point $\boldsymbol{v}$, $\alpha(H)$ will be incident to $\alpha(\boldsymbol{v})$. Therefore, all hyperplanes in $\mathbf{B}' = \alpha(\mathbf{B})$ are incident to vertex $\boldsymbol{b}' = \alpha(\boldsymbol{b})$ and all hyperplanes in $\mathbf{C}' = \alpha(\mathbf{C})$ are incident to $\boldsymbol{c}' = \alpha(\boldsymbol{c})$, so all the hyperplanes in $\mathbf{B}' \cap \mathbf{C}'$ also meet in an edge $\overline{\boldsymbol{b}'\boldsymbol{c}'}$ of the arrangement, and $\mathbf{B}'$ and $\mathbf{C}'$ are therefore adjacent. This shows that the adjacency structure of the graph of cobases is preserved under the automorphism group, and thus the set $S' = \{\alpha(\mathbf{C}) : \mathbf{C} \in S\}$ is equivalent to S up to symmetries. $\square$

Theorem 3.1.1. *Given an initial cobasis* $\mathbf{B}$ *of a hyperplane arrangement* $\mathcal{A}$ *with a connected adjacency graph of cobases, if all neighbours of* $\mathbf{B}$ *are visited and this visitation is repeated recursively for one previously visited member of each orbit of cobases under some automorphism group* G *of* $\mathcal{A}$*, then at least one member of each of the orbits of cobases of* $\mathcal{A}$ *under* G *will be visited.*

Proof. Assume no cobasis in some orbit X is visited. This implies that none of the cobases adjacent to cobases in X were chosen as representatives of their orbits to recurse from, since a representative of X would have been found adjacent to the chosen cobasis. However, this implies by Lemma 3.1.1 that for any orbit Y containing a cobasis adjacent to a member of X, no representative of Y has been visited, as any such representative $\mathbf{R}$ would be adjacent to a cobasis $\mathbf{M} \in X$ by symmetry and $\mathbf{M}$ would have been visited from **R**. Applying this inductively, if the graph of cobases is connected then no representative of any orbit has been visited. However, it is given that an initial cobasis **B** (which is an orbit representative) has been visited, a contradiction, therefore some cobasis in every orbit must have been visited.

□

Algorithm 1 Basis orbit enumeration algorithm

```
function SymmetricBasisSearch(void)
    $\mathbf{B} \leftarrow$ InitialCobasis()          ▷ find a cobasis of the arrangement
    $S \leftarrow$ a stack of cobases, initially empty
    Report($\mathbf{B}$)
    PushCobasis($S, \mathbf{B}$)
    repeat                                ▷ explore outward from this cobasis
        $\mathbf{B} \leftarrow$ PopCobasis($S$)
        for all $\mathbf{B}_i \in$ Adjacent($\mathbf{B}$) do          ▷ search adjacent cobases
            if InNewOrbit($\mathbf{B}_i$) then
                Report($\mathbf{B}_i$)
                PushCobasis($S, \mathbf{B}_i$)
            end if
        end for
    until Empty($S$)
end function
```

Having shown that representatives of each orbit will be discovered by this basis enumeration method, Algorithm 1 describes it in more detail. In the listing the subroutine InitialCobasis() returns an arbitrary cobasis of the arrangment; Adjacent(**B**) returns a list of all cobases $\mathbf{B}_i$ which are adjacent to a cobasis **B**. InNewOrbit(**B**) tests whether a cobasis **B** is in an orbit already discovered, while Report(**B**) is used to output a newly discovered cobasis **B** as well as adding **B** to the list of orbit representatives

InNewOrbit($\mathbf{B}'$) requires to work. The subroutines PushCobasis($S, \mathbf{B}$) and PopCobasis(S), which push and pop a cobasis to or from a stack S, updating internal structures to be consistent with that cobasis, complete the high level description of the algorithm.

To asymptotically analyze this algorithm, let n be the number of hyperplanes in the arrangement, d be the dimension of the underlying space, and k be the number of basis orbits. The outer loop of Algorithm 1 will be repeated k times. For each of the d elements of these k representative cobases, Adjacent may find up to $(n - d)$ adjacent cobases, one for every other hyperplane in the arrangement, while for simple arrangements, only 2 adjacent cobases may be found for each of the d elements, as there are no degenerate pivots. For each of these adjacent cobases, InNewOrbit must check if it is in the same orbit as any of the $O(k)$ previously discovered orbit representatives; as the complexity of the group theoretic calculations is outside the scope of this thesis, I will consider this check to be a call to an oracle that takes time $S(n, d)$. Combining these results, this algorithm has a runtime in $O(k^2 dnS(n, d))$; for simple arrangements this can be improved to $O(k^2 dS(n, d))$. To express these results in terms of input size only, note that there can be no more than $\binom{n}{d}$ cobases of an arrangement; this is $O(n^d)$. Since each of these cobases is an orbit representative under the trivial automorphism group containing only the identity transformation, $k \in O(n^d)$, and the previous results can be re-expressed as $O(n^{2d+1} dS(n, d))$ for the general case, and $O(n^{2d} dS(n, d))$ for simple arrangements. Fortunately, the results in Chapter 4 show practical performance well below these uninspiring worst-case bounds.

Practical pivoting algorithms for vertex enumeration use some form of perturbation or equivalent pivot rule (*e.g.* [3]) to reduce the number of bases reported per vertex. In this thesis the goal is to find all orbits of bases, so standard symbolic perturbation schemes that ignore the symmetry group are unlikely to work well. In [6] Bremner *et al.* describe an explicit *orbitwise* perturbation scheme that preserves the orbits of bases of the original input, possibly shattered into several orbits. Since this can be implemented as a preprocessor, it is not discussed here; some of the experimental data (the E7-j examples) in Table 4.1 is of this preprocessed type.

All the required subroutines for Algorithm 1 can be defined to act on a simplex tableau. Most of these subroutines have been known since Dantzig's original formulation of the simplex algorithm, and can be found in most linear programming textbooks, though some simple modifications may be needed to convert processes intended for use on polyhedra to work with arrangements. For PushCobasis($S, \mathbf{B}$) and PopCobasis(S), my implementation keeps an internal stack of pivots performed, reversing those pivots as necessary to return to an earlier cobasis.

The reverse search algorithm of Avis and Fukuda [4] which this approach is based on differs in that it does not keep an internal stack of pivots. Instead, the reverse search algorithm applies an objective function to the vertices of the arrangement, as in the simplex method, and evaluates a pivoting rule such as the minimum ratio test at each cobasis to determine a "parent" cobasis on the path from the optimal cobasis, which the reverse search algorithm is started at[1]. Where my algorithm would pivot up its stored stack of cobases, reverse search pivots to this parent cobasis, and where my algorithm would move to a sibling of its current cobasis, reverse search pivots to the parent, re-calculates the parent cobasis' children, and pivots to the next cobasis on that list. The advantage of the approach taken by the reverse search algorithm is that it requires only a constant amount of memory, but this advantage is offset by the extra computational burden of repeatedly recalculating the parent and sibling cobases. Furthermore, the traversal order used by reverse search is not compatible with the pruning process described above, as Theorem 3.1.1 depends on *all* neighbours of a reported cobasis being visited from there, while reverse search visits only the neighbours that are children of the reported cobasis according to the pivoting rule.

My implementation of ADJACENT($\mathbf{B}$), detailed in Algorithm 2, is based on the minimum ratio test. The pivoting rule included tries all the variables x_j in the cobasis $\mathbf{B}$ as entering variables, attempting to find valid leaving variables for each. Given an entering variable x_e, this method reports all the basic variables that are already zero as possible leaving variables as these represent degenerate pivots, in addition to all the basic variables x_i which have a ratio $b_i/a_{i,e}$ which has minimal absolute value in either the positive

[1]If there is more than one optimal cobasis, the reverse search algorithm is repeated by choosing each in turn as the initial cobasis.

or negative direction. Taking both positive and negative ratios ensures that new adjacent cobases are found on either side of the hyperplane corresponding to x_e.

Algorithm 2 Adjacent Cobasis Algorithm

```
function ADJACENT($\mathbf{B}$)
    $S \leftarrow$ a set of cobases, initially empty          ▷ try all entering indices
    for all $e \in \mathbf{B}$ do
                                  ▷ track positive, negative, and zero pivots
        $P, N, Z \leftarrow$ sets of cobases, initially empty
        $r_P \leftarrow \infty, r_N \leftarrow -\infty$
        for all $l \notin \mathbf{B}, a_{l,e} \neq 0$ do               ▷ try all leaving indices
            $r \leftarrow b_l / a_{l,e}$                              ▷ check minimum ratio
            $\mathbf{B}' \leftarrow \mathbf{B} \cup \{l\} \backslash \{e\}$
            if $r = 0$ then
                $Z \leftarrow Z \cup \{\mathbf{B}'\}$
            else if $0 < r < r_P$ then
                $P \leftarrow \{\mathbf{B}'\}$
                $r_P \leftarrow b_l / a_{l,e}$
            else if $r = r_P$ then
                $P \leftarrow P \cup \{\mathbf{B}'\}$
            else if $r_N < r < 0$ then
                $N \leftarrow \{\mathbf{B}'\}$
                $r_N \leftarrow b_l / a_{l,e}$
            else if $r = r_N$ then
                $N \leftarrow N \cup \{\mathbf{B}'\}$
            end if
        end for
        $S \leftarrow S \cup P \cup N \cup Z$
    end for
    return $S$
end function
```

For the InNewOrbit($\mathbf{B}$) method, the algorithm must determine whether or not $\mathbf{B}$ is in an orbit that has already been discovered or not. The simplest way to do this is to check if $\mathbf{B}$ is in the same orbit as each of the known orbit representatives individually. However, the group theoretic calculations to perform this check are computationally expensive, so it is best to avoid performing them where possible. Toward this end, the `Basil` software discussed in this thesis can utilize various invariant checks to swiftly rule out orbit representatives that cannot possibly be symmetric. The simplest such invariant is to check that the vertices represented by $\mathbf{B}$ and the orbit representative to check have the same number of incident hyperplanes, as no symmetric cobasis will have a different hyperplane incidence. A related but more powerful invariant is that the sets of angles between the hyperplanes meeting in a given cobasis will remain the same under isometries; `Basil` represents these sets of angles by the Gram submatrix described in Chapter 2, storing the discovered orbit representatives in a hash table indexed by Gram submatrix. This optimization allows Basil to only compare orbit representatives that have similar local structure, rather than the entire set of orbit representatives. `Basil` also provides an option to construct a subset of the constraints of the fundamental domain, and only investigate cobases which correspond to a vertex satisfying these constraints.

3.2 Parallel Variant

As this algorithm is essentially a depth-first search across the adjacency graph of cobases, and both traversal of an edge of this graph and determining the neighbours of a given cobasis are quite computationally expensive, a natural way to attempt to speed up the process is to have multiple threads searching different parts of the graph simultaneously. I chose a shared memory model for parallelism over a message passing model since the threads needed to share a relatively large amount of information regarding the cobases and vertices they had already found to prevent duplication of work, and must share a significant amount of internal state to share work with each other, namely the stack of pivots performed to reach their current basis. Using a shared memory model with one thread per physical processor core also has the benefit of lowering synchronization overhead by enforcing a fixed small number of cooperating threads.

Having chosen a shared memory model of parallelism with one thread per processor core, two modifications needed to be made to Algorithm 1. Firstly, the algorithm required a method to divide work among threads, discussed in Section 3.2.1; desirable qualities for this method include high processor utilization, limited duplication of work, and low synchronization overhead. Secondly, the threads needed a way to share results among themselves with minimal overhead, discussed in Section 3.2.2; this required careful construction of critical sections to synchronize access to global data structures without unnecessarily blocking other work.

3.2.1 Assigning Cobases to Threads

The simplest way to solve the problem of splitting work between threads would be to simply modify the PushCobasis($S, \mathbf{B}$) and PopCobasis(S) methods from Algorithm 1 to store the path of pivots used to reach the basis $\mathbf{B}$ of the stack S as well, synchronize access to S between threads, and have all threads perform the outer loop in parallel. This approach is functional, but has some disadvantages, as I explain next, namely that it squanders the advantage in minimizing costly pivot operations between cobases that is conferred by depth first search and requires an unnecessarily high amount of synchronization between threads.

In a single-threaded depth first search, often only one or two pivots are needed to move from the current cobasis to the next one; one if the next cobasis is a child of the current cobasis, two if it is a sibling. In multi-threaded depth-first search, the next cobasis may have been placed on the stack by a different thread in a more distant part of the graph, requiring a greater number of pivots to reach. Furthermore, having all the threads work directly off a common stack will tend to cluster their current cobases in a relatively small portion of the graph, leading to significant duplicated work as multiple threads find the same cobasis from different edges frequently, though the fact that the adjacency graph of cobasis orbits is often much smaller than the adjacency graph of cobases reduces the negative effect of such clustering.

To mitigate the factors discussed above, I have chosen to give each thread t

Algorithm 3 Load-balancing algorithm

```
function GetCobasis($S$, $S_t$, $nWaiting$, $waiting_t$, $\mathbf{B}$)
    if Empty($S_t$) then                                    ▷ Get cobasis from global stack
        begin critical section
            if Empty($S$) then
                if $waiting_t$ = false then
                    $waiting_t \leftarrow$ true
                    $nWaiting \leftarrow nWaiting + 1$
                end if
            else
                $\mathbf{B} \leftarrow$ PopCobasis($S$)
                if $waiting_t$ = true then
                    $waiting_t \leftarrow$ false
                    $nWaiting \leftarrow nWaiting - 1$
                end if
            end if
        end critical section
    else                                                    ▷ Get cobasis from local stack
        $\mathbf{B} \leftarrow$ PopCobasis($S_t$)
        if Empty($S_t$) = false then
            begin critical section
                if $nWaiting > 0$ then                      ▷ Unload local stack to global
                    $S \leftarrow$ Tail($S_t$)
                    $S_t \leftarrow$ Head($S_t$)
                end if
            end critical section
        end if
    end if
end function
```

its own local work stack S_t. Threads always push new cobases to S_t, instead of the global stack S, and so long as there are cobases left to explore in S_t, the thread t pops new cobases from S_t rather than the global stack S. If some thread has no work (*i.e.* S_t is empty), a thread with extra work will donate some cobases to the global stack for redistribution; I have chosen to move all but one of the cobases in S_t to S in this case, but other approaches could be taken. Threads with no cobases to explore poll the global stack S until they can pop a new cobasis from it and restart their search; the algorithm terminates when no thread has any cobases left to explore. Algorithm 3 explains the management of the local and global stacks in more detail, while Algorithm 4 shows the modifications which must be made to Algorithm 1 to incorporate these changes.

This approach to work distribution is quite similar to the parallel depth-first search for shared memory architectures proposed by Rao and Kumar in [17, 13], with the difference that their approach involves a rotating "donor" thread, while my approach uses the global work stack for this purpose. This approach also bears similarities to the dynamic load balancing implementation employed by Brüngger *et al.* [8] in `ZRAM`, their framework for parallel search algorithms, though `ZRAM` is based on a message passing architecture, and thus cannot use a global work stack. Instead of a global work stack, threads in `ZRAM` that exhaust their local work stack request new work units from a randomly chosen peer. If this peer lacks work to share, the work request is passed on to a third peer thread, chosen in a round-robin fashion.

Algorithm 4 Parallel basis orbit enumeration algorithm

```
function ParallelBasisSearch(void)
    $nThreads \leftarrow$ the number of threads
    $nWaiting \leftarrow 0$
    $\mathbf{B} \leftarrow$ InitialCobasis()
    $S \leftarrow$ a stack of cobases, initially empty
    Report($\mathbf{B}$)                                ▷ Start at initial cobasis
    PushCobasis($S, \mathbf{B}$)
    begin parallel block
        $S_t \leftarrow$ a stack of cobases, initially empty
        $waiting_t \leftarrow$ false
        repeat
            GetCobasis($S, S_t, nWaiting, waiting_t, \mathbf{B}$)
            if $waiting_t =$ true then continue     ▷ Skip iteration if no work
            for all $\mathbf{B}_i \in$ Adjacent($\mathbf{B}$) do       ▷ Search adjacent cobases
                if InNewOrbit($\mathbf{B}_i$) then
                    Report($\mathbf{B}_i$)
                    PushCobasis($S_t, \mathbf{B}_i$)
                end if
            end for
        until $nWaiting = nThreads$
    end parallel block
end function
```

Weibel [24] has presented another similar scheme for parallelizing a reverse-search based algorithm, but his message passing based architecture includes a complex protocol to assign a “boss” thread to provide work and propagate knowledge of the current boss thread to the other threads, as opposed to the the simpler global work stack for the shared memory system described here.

3.2.2 Sharing Results

InNewOrbit($\mathbf{B}$) requires knowledge of all the basis orbits already discovered to determine if the cobasis $\mathbf{B}$ is in fact in a new orbit. However, the group theoretic calculations to check the symmetry of $\mathbf{B}$ with the existing orbit representatives are very expensive, dominating the runtime of the algorithm in my experiments, so it is infeasible to block the other threads from reporting new orbit representatives while one thread tests a new basis. However, if other threads are allowed to report new orbit representatives, InNewOrbit($\mathbf{B}$) must be adjusted to account for these new representatives as they are seen.

The approach I took to solve this problem was to have each thread keep a local copy L of the global list G of basis orbit representatives, and lazily update L from G as needed. Algorithm 5 shows how the body of the inner loop of Algorithm 1 is rewritten to check if a basis $\mathbf{B}_i$ is new, and report it if so. In this description InNewOrbit($\mathbf{B}, L$) is a boolean function which checks if $\mathbf{B}$ is in a new orbit with respect to the orbit representatives in L, After(L, i) returns a list containing the elements of L with index at least

i, L being zero-indexed, and AddAll(L, C) adds all the elements of a list C into a list L.

Algorithm 5 Parallel algorithm to find and report new cobases

```
$isNew$ = InNewOrbit($\mathbf{B}_i, L$)                    ▷ check the local list
while $isNew$ do
    begin critical section                      ▷ check if the local list up to date
        if Size($L$) = Size($G$) then
            Append $\mathbf{B}_i$ to $G$
            Report($\mathbf{B}_i$)
            $C \leftarrow$ an empty list
        else
            $C \leftarrow$ After($G$, Size($L$))
        end if
    end critical section
    if Empty($C$) then return
    else                                                   ▷ update local list
        $isNew$ = InNewOrbit($\mathbf{B}_i, C$)
        AddAll($L, C$)
    end if
end while
```

Chapter 4

Implementation and Results

4.1 Implementation

In order to achieve good performance, Algorithm 1 needs an efficient pivot implementation. Previous experiments by Avis [3] suggest a significant advantage for the integer pivoting method of Edmonds [11]. The program described in this thesis, `Basil`, was built using David Avis' `lrslib` [2], which uses Edmonds' integer pivoting.

The design of `Basil` is quite closely based on the `Symbal` software of Bremner *et al.* [6], which performs basis enumeration up to symmetries on polyhedra. However, where `Symbal` is implemented in the `GAP` [23] computer algebra system with calls to C libraries wrapping `lrslib` for simplex operations and McKay's `Nauty` [14] for graph automorphism calculations, `Basil` has been re-implemented in C++, using Rehn's `permlib` [18] library to replace both the group theoretic capabilities of `GAP` used by `Symbal` and the automorphism code in `Nauty` with matrix automorphism routines. `Basil`, like `Symbal`, uses `lrslib` for its simplex tableau implementation.

4.2 Test Data and Results

4.2.1 Comparison with Symbal

As `Basil` is designed as an extension of `Symbal`, it is also capable of performing basis enumeration of polyhedra up to symmetries. Though this functionality is not the focus of this thesis, the experimental results shown in Table 4.1 compare the relative performance of `Basil` and `Symbal` for basis enumeration up to symmetries on a set of polyhedra. The Ey instances discussed are the Dirchlet-Voronoi-cells (DV-cells) of the root lattices E_y, while the Dx instances are the DV-cells of the root lattices D_x, as described in section 7.2 of [6]; the E7-j instances are E7, perturbed as described in section 7.2.3 of [6]. These polyhedra come from group theory, and have been studied by Schürmann [20] in the contexts of sphere packing and the Kepler conjecture; they are also used as test data for `Symbal`, and are available in the source distributions of both `Symbal` and `Basil`. As can be seen from these results, `Basil` is generally about two orders of magnitude faster than `Symbal`, attributable to the lower overhead of C++ execution than the `GAP` interpreter and the more sophisticated data structures available in C++ compared to `GAP`. These numbers represent only the CPU time of both programs; this

Table 4.1: Comparison of Basil & Symbal (bases & vertices count *orbits*)

				time(s)	
Problem	n:d	$\|G\|$	bases:vertices	Basil	Symbal
E7	126:8	2903040	32:2	1.82	282.16
E7-3	126:8	336	82:58	0.59	12.41
E7-7	126:8	720	1195:106	19.88	1507.72
E7-65	126:8	23040	356:14	7.88	1308.07
E7-102	126:8	103680	41:7	1.27	223.97
E8	240:8	696729600	2:2	0.41	2.13

is a fairly accurate representation of `Basil`'s runtime, but underestimates `Symbal`'s by about half due to overhead from the interprocess communication needed by `GAP` to interact with the external C libraries used.

4.2.2 Test Data

Table 4.2 provides some information about the named problem instances which results are discussed for here, including the number n of hyperplanes in the arrangement, dimension d of the underlying space, order $|G|$ of the symmetry group, and the number of basis and vertex orbits of the problem instance, as calculated by `Basil`. The column labelled "TRD" is a measure of problem size I have called "total representative degree", the sum of the degree (number of adjacent bases) of one representative basis from each orbit; this value is well defined, as each orbit representative will have the same number of neighbours by symmetry. The TRD value for these problems is also calculated by `Basil`. In this table, Problems DxA and EyA are the bounding hyperplane arrangements for the DV-cells of the root lattices D_x and E_y

Table 4.2: Test Data (bases & vertices count *orbits*)

Problem	n:d	$\|G\|$	bases:vertices	TRD
D4A	24:4	1152	12:7	206
D5A	40:5	3840	104:25	3296
C5-6-3a	16:5	120	51:16	926
C5-6-3b	25:5	120	291:36	7890
C6-6-3a	36:6	720	1394:91	74057
C6-6-3b	50:6	1440	5342:157	302294
C7-6-3a	48:7	2304	18720:140	1508673
C7-6-3b	50:7	1440	41735:1	8568000
E7A	126:8	2903040	12399:227	1213622

discussed above in the description of the Symbal test data, while the Cx-y-z instances are generated by choosing z vertices of the x-cube and acting on them with a subgroup of the hyperoctohedral group (the full automorphism group of the x-cube) with at least y orbits. The construction method for the Cx-y-z instances was employed because it was able to generate arrangements with a wide variety of distinct structures that still had a significant number of automorphisms; the Cx-y-z instances discussed here are included in the `Basil` source distribution, along with a script to generate such instances. All timing results reported in this thesis are from the Fundy and Placentia ACEnet clusters [1], which have 2.6–3.2 GHz AMD Opteron processors.

4.2.3 Results and Analysis

Table 4.3 shows some performance results for `Basil` on the above mentioned problem instances. `Basil` was run 9 times for each of these instances and the median time reported. The maximum time for each instance was less than

Table 4.3: Basil Timing Results (bases count *orbits*)

Problem	n:d	bases	time(s)
D4A	24:4	12	0.02
D5A	40:5	104	0.49
C5-6-3a	16:5	51	0.06
C5-6-3b	25:5	291	0.69
C6-6-3a	36:6	1394	11.38
C6-6-3b	50:6	5342	63.08
C7-6-3a	48:7	18720	456.59
C7-6-3b	50:7	41735	4262.75
E7A	126:8	12399	1603.56

Table 4.4: Non-Symmetric Timing Results (*all* bases & vertices counted, slowdown is factor relative to symmetric calculations)

Problem	n:d	bases:vertices	time(s)	slowdown
D4A	24:4	5028:863	24.30	1620.0
C5-6-3a	16:5	3005:234	4.39	79.8
C5-6-3b	25:5	24444:852	123.27	179.4

7.3% higher, and the minimum time no more than 1.9% lower, so this is a fair representation of `Basil`'s runtime.

Table 4.4 shows the benefits of considering symmetries for basis enumeration; the values in this table are the results of using `Basil` to enumerate all the cobases of the given test cases. The test cases used for this example are all very small, since, due to the large increase in runtime incurred by ignoring symmetries, none of the larger table instances completed in the five hours allotted. The timing results shown here are also medians of 9 runs, with the maximum time being no more than 17.2% higher.

To perform a more rigorous analysis of the parameters which affect the `Basil` runtime, I generated 100 C6-6-3 instances and 100 C7-6-3 instances and ran them with a 5 hour runtime limit and a 2GB virtual memory limit on the Fundy cluster, as earlier testing proved C6-6-3 and C7-6-3 to be good parameter values for generating many distinct problem instances and 5 hours and 2GB virtual memory reasonable limits on the resources needed to solve such instances. 165 of these 200 problem instances ran to completion in more than the resolution of the system timer, but less than the 5 hour cap; these runs are discussed in the analysis below.

To analyze these timing results, I attempted to fit a curve for runtime t in milliseconds in terms of $n \times d$, the size of the input matrix; $|G|$, the order of the symmetry group; the number of basis orbits b; and the total representative degree (TRD) of the problem. While $n \times d$ and $|G|$ may be swiftly calculated from the problem input, the number of basis orbits and the TRD are part of the output of `Basil`, and cannot be known before running the program. Since the total number of orbits of cobases may in general be exponential in the input size, the typical algorithm analysis approach of measuring runtime in terms of input size is not very useful here. A reasonable approach instead is to use the output size to measure the difficulty of a given problem instance. Since TRD is closely correlated with the number of basis orbits, I attempted to fit curves depending on TRD separately from curves depending on the number of basis orbits.

Attempts to fit a linear curve to this data showed variance increasing with problem size, violating the assumption of the regression model that the variance be roughly constant. To fix this issue I employed a Box-Cox transformation [5] to normalize the time data - this transformation fits a linear curve to t^λ for some λ rather than fitting to t. This transformation resulted in a much more consistent variance across the range of runtimes in the sample, though at the cost of a direct fit for runtime. As the runtimes in this sample varied over four orders of magnitude, the λ values fit by the Box-Cox regression are necessarily very small to collapse the variances of the data points down to a constant range, so the fits produced are not useful for approximating the asymptotic properties of the problem, as they have extremely small (or large) constant factors when expressed in terms of t rather than t^λ.

$$t^{0.176} = 0.494 + 3.14 \times 10^{-5} b + 43.9 \frac{1}{|G|} + 0.0219(n \times d) \tag{4.1}$$

$$t^{0.160} = 0.736 + 2.35 \times 10^{-7} \mathrm{TRD} + 36.8 \frac{1}{|G|} + 0.0175(n \times d) \tag{4.2}$$

$$t^{0.0694} = 1.04 + 7.83 \frac{1}{|G|} + 0.00386(n \times d) \tag{4.3}$$

Equation 4.1 fits the input parameters and the number of basis orbits under the Box-Cox transformation with correlation coefficient R^2 of 85.9%, while Equation 4.2 fits TRD similarly ($R^2 = 84.0\%$); Equation 4.3 fits the data in terms of only the input parameters, but it has a much lower correlation coefficient ($R^2 = 76.1\%$). All parameter coefficients of each of these curves are significant with 99.9% probability, thus each of the parameters is statistically significant. These fits demonstrate that larger symmetry groups decrease the

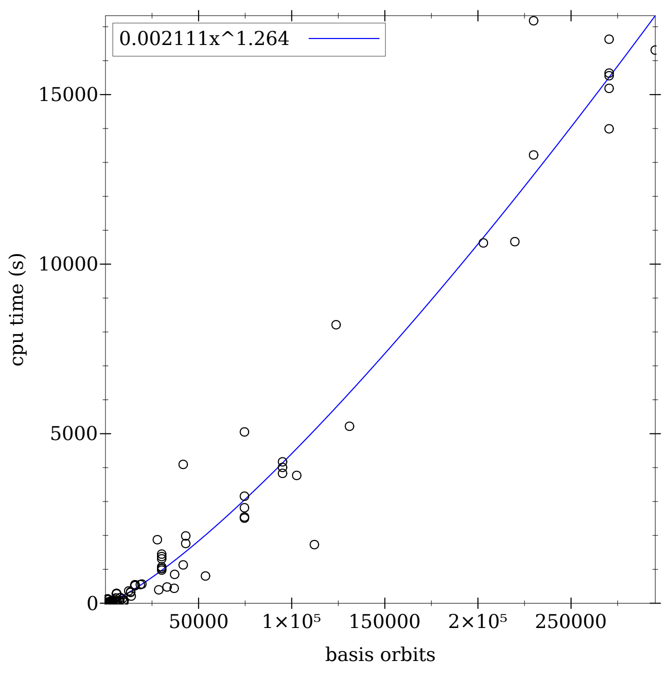


Figure 4.1: Time for Cd-6-3 instances by # basis orbits.

runtime (the curves were fit to $\frac{1}{|G|}$ to get positive coefficients), while larger input matrices, more basis orbits, and higher TRD all increase runtime. This is reasonable, as larger input matrices should lead to larger problems, and the number of basis orbits and the TRD are both measures of problem size from the output, so one would expect that all of these factors would increase runtime. On the other hand, a larger symmetry group will have larger orbits of cobases, and therefore fewer orbits with the same number of cobases, so the runtime is likely to decrease with a larger symmetry group.

Given the effectiveness of the Box-Cox power transformation for normalizing the runtime values to get a linear fit, I attempted to fit power functions of basis orbits and TRD to runtime as well. I was not able to achieve a sta-

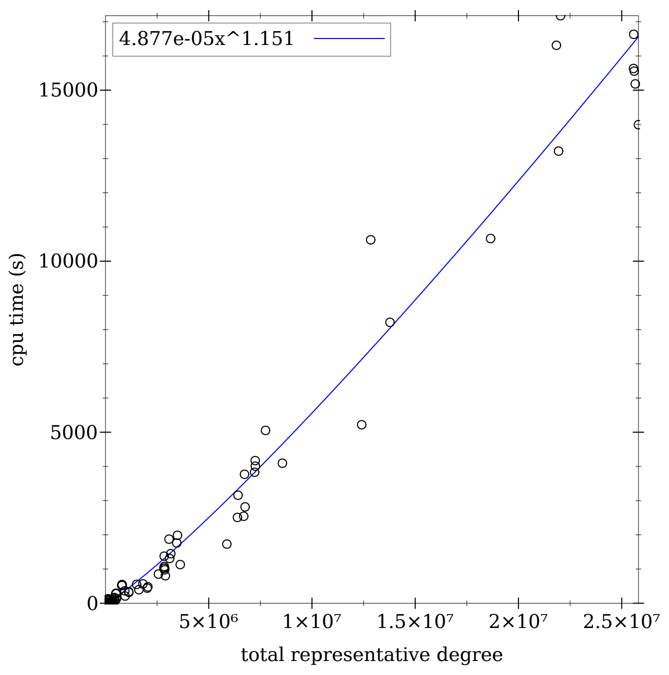


Figure 4.2: Time for C*d*-6-3 instances by total representative degree.

tistically valid fit, as the variances again increased with runtime, but for the range of times tested (10 milliseconds to 5 hours), the curves fit reasonably well, seeming to approximate the average runtime, and thus should provide some indication of the real-world performance of `Basil`. Figure 4.1 shows a super-linear but sub-quadratic fit in the number of basis orbits, consistent with a depth-first traversal of a moderately dense connected graph. Figure 4.2 provides some support to this hypothesis with a near-linear fit in the total representative degree, which is the number of graph nodes visited in the traversal.

4.2.4 Optimizations

4.2.4.1 Memory-less Reverse Search

As discussed in Chapter 3, the pivoting approach implemented in `Basil` and `Symbal` differs from the reverse search algorithm proposed by Avis and Fukuda [4] for the non-symmetric vertex enumeration problem in that the reverse search algorithm does not maintain state describing cobases already found or the path from the initial cobasis to the cobasis currently under consideration. That approach has the benefit of requiring a relatively small constant amount of memory, but also requires more simplex computations, increasing running time. Because it is unclear how to prune cobases symmetric to those already seen from the search without storing the cobases already seen, I have not yet investigated a memory-less reverse search for this problem. However, there are usually few orbits of cobases relative to the total number of cobases, allowing `Basil` to keep representatives of each in a feasible amount of memory. Additionally, the limiting factor on the size of instances `Basil` can currently solve is the computational expense of the group theoretic calculations required to check symmetry (encapsulated in InNewOrbit in Algorithm 1), which dominate the running time of `Basil` to a significant degree. Profiling results show that tests for orbit membership take about 60% of the runtime of `Basil`, while the only other individual operation which significantly contributes to runtime is simplex pivoting, contributing about 20% of the execution time.

4.2.4.2 Simple Invariants

Because the group theoretic computations involved in checking if two cobases are in the same orbit under the group action are so expensive, `Basil` utilizes some cheaper invariants of symmetric cobases to reduce the number of required symmetry tests. The simplest of these invariants is to check that the number of hyperplanes incident to the vertices defined by the two cobases is the same, as an automorphism of the hyperplane arrangement preserves the number of hyperplanes which meet at any given vertex. `Basil` also keeps a cache of the thousand most recently seen cobases to avoid the need to retest previous cobases such as the cobasis that was pivoted from to reach the current cobasis or near neighbours of the current cobasis.

4.2.4.3 Gram Submatrix

Additionally, `Basil` uses the Gram submatrix to differentiate cobases; representatives of known cobasis orbits are stored in a hash table indexed by the Gram submatrix corresponding to each cobasis. Comparing each newly discovered cobasis only to the cobases having Gram submatrices which are equivalent under the sorting procedure described in Chapter 2 greatly reduces the number of expensive group theoretic tests which must be performed. If the Gram submatrix invariant is turned off in `Basil`, execution time on a given instance increases dramatically, particularly for larger instances, while when activated the Gram matrix computations consume only about 5% of the execution time of `Basil`. The speedup numbers in Figure 4.3 compare

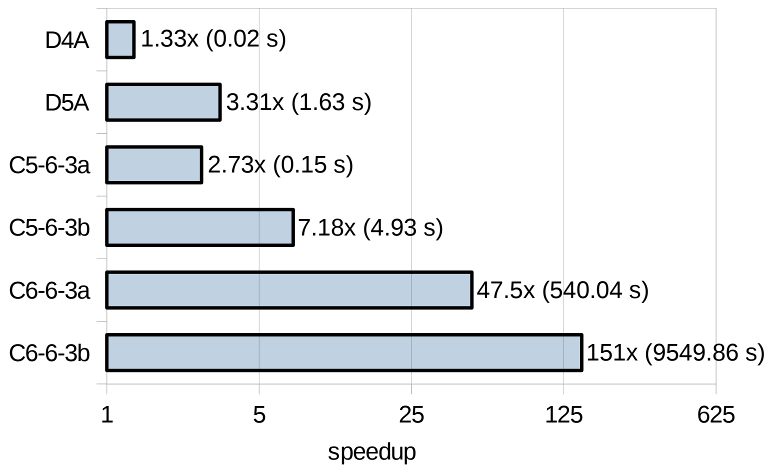


Figure 4.3: Speedup from using Gram matrix, log-scaled to fit all data (bar labels are speedup and runtime *without* Gram matrix)

the median of 9 runs without the Gram matrix optimization to the runtimes with the optimization reported in Table 4.3; the omitted instances did not complete within a 5 hour limit without the Gram matrix optimization.

4.2.4.4 Fundamental Domain

I also investigated use of constraints from the fundamental domain to prune the search space of the search algorithm. As generating the entire fundamental domain is computationally expensive, the approach taken in this experiment was to compute a polyhedron which is a superset of the full fundamental domain, comprising only some subset of the constraints of the full fundamental domain described in Theorem 2.2.2. This polyhedron is constructed as follows: take the initial cobasis of the arrangement, transform it by each of the generators of the automorphism group, and then find the vertices corresponding to these transformed cobases. For each unique vertex $\boldsymbol{v}$ so constructed which is distinct from the initial vertex $\boldsymbol{s}$, generate a constraint halfspace which contains $\boldsymbol{s}$ but not $\boldsymbol{v}$; the bounding hyperplane of this halfspace bisects the line segment $\overline{\boldsymbol{sv}}$ as in Lemma 2.2.1. This method generated relatively few of the constraints of the complete fundamental domain, as the initial vertex is rarely a generic point, typically being its own image under multiple automorphisms in the group. This optimization approach seems to be of less utility than the others I have tried, however, providing only about 5% speedup for sufficiently large instances, and pruning very few cobases relative to the number of basis orbits. As each added constraint should reduce the feasible space by about half its previous size, I would expect diminishing returns from adding more constraints, and thus that even if more constraints were generated they would not greatly reduce runtime. Table 4.5 gives counts of generated constraints and pruned cobases, as well as speedups relative to Table 4.3, omitting problems C7-6-3b and E7A, for which no constraints were generated.

4.2.5 Parallelization

I used the OpenMP framework [16] for the parallel version of `Basil`, preferring it over MPI [15] for its shared memory architecture, and over PThreads [12] due to its ease of programming and high-level operations. Table 4.6 shows the timing results of running this parallel implementation with 2, 4, 6, and 8 processor cores. It should be noted that these are wall-

Table 4.5: Fundamental Domain Timing Results (bases count *orbits*, pruned is the number cobases pruned for falling outside the fundamental domain)

Problem	n:d	bases	constraints	pruned	time(s)	speedup
D4A	24:4	12	3	0	0.02	0.75
D5A	40:5	104	1	16	0.51	0.96
C5-6-3a	16:5	51	1	2	0.06	0.92
C5-6-3b	25:5	291	2	7	0.66	1.04
C6-6-3a	36:6	1394	1	14	10.61	1.07
C6-6-3b	50:6	5342	3	63	59.81	1.05
C7-6-3a	48:7	18720	1	22	440.82	1.03

clock times rather than CPU times such as were reported Table 4.3; as `Basil` is CPU-bound, the wall times should be comparable to the CPU times. Table 4.7 shows how memory usage increases with number of cores and instance size. Both Table 4.6 and Table 4.7 report the median results of 9 runs. Figure 4.4 shows the parallel timing results for the Cd-6-3 instances discussed above; some runs did not complete for some numbers of parallel cores, so there are different numbers of data points for sequential, 2, 4, and 6 core runs. I omitted the 8 core runs because the results were basically identical to the 6-core runs, and overly cluttered the plot.

Table 4.6 and Figure 4.4 show that `Basil` does derive some runtime benefit from parallelization up to 6 cores, though synchronization overhead appears to outweigh the benefits of using 8 cores over 6 for most of the problems tested. As shown in Table 4.7, memory usage is approximately the same whether threads are given local work stacks or not, increasing roughly linearly in the number of cores, though there seems to be an execution time benefit from use of local stacks. Table 4.8 shows higher efficiency (speedup per

Table 4.6: Parallel Basil Timing Results (bases count *orbits*, p is # cores)

			time(s) with local stacks				
Problem	n:d	bases	$p = 1$	$p = 2$	$p = 4$	$p = 6$	$p = 8$
C6-6-3a	36:6	1394	12	7	5	5	13
C6-6-3b	50:6	5342	65	38	28	27	85
C7-6-3a	48:7	18720	473	352	208	191	800
C7-6-3b	50:7	41735	4247	3395	1941	1512	5864
E7A	126:8	12399	1604	903	573	474	496
			time(s) without local stacks				
C6-6-3a	36:6	1394	12	13	6	6	7
C6-6-3b	50:6	5342	65	51	32	34	39
C7-6-3a	48:7	18720	473	462	258	250	236
C7-6-3b	50:7	41735	4247	3465	1983	1557	1609
E7A	126:8	12399	1604	1102	651	580	563

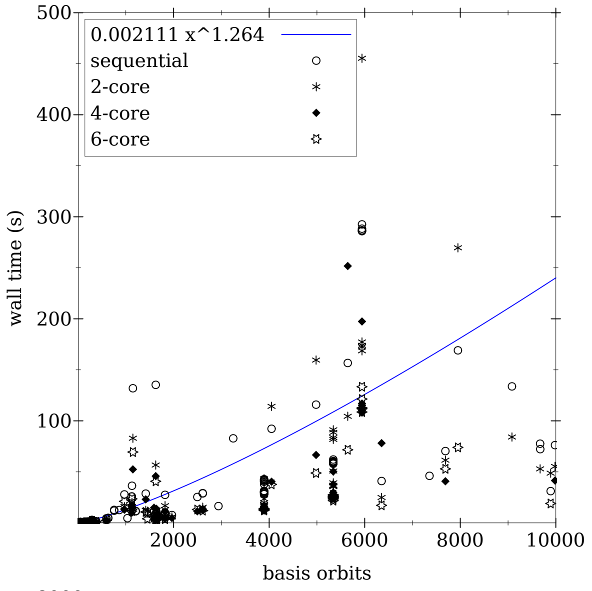
Table 4.7: Parallel Basil Memory Usage (bases count *orbits*, p is # cores)

			memory(MB) with local stacks				
Problem	n:d	bases	$p = 1$	$p = 2$	$p = 4$	$p = 6$	$p = 8$
C6-6-3a	36:6	1394	71	99	208	399	75
C6-6-3b	50:6	5342	82	148	263	472	165
C7-6-3a	48:7	18720	145	499	634	918	675
C7-6-3b	50:7	41735	553	1555	2524	3632	1492
E7A	126:8	12399	119	211	410	700	1079
			memory(MB) without local stacks				
C6-6-3a	36:6	1394	71	95	210	400	669
C6-6-3b	50:6	5342	82	154	270	477	1238
C7-6-3a	48:7	18720	145	526	640	928	1316
C7-6-3b	50:7	41735	553	1549	2422	3495	4696
E7A	126:8	12399	119	219	420	695	1064

Table 4.8: Parallel Basil Speedup & Efficiency Results (bases count *orbits*, p is # cores)

			speedup[%efficiency] with local stacks			
Problem	n:d	bases	$p = 2$	$p = 4$	$p = 6$	$p = 8$
C6-6-3a	36:6	1394	1.71[86%]	2.40[60%]	2.40[40%]	0.92[12%]
C6-6-3b	50:6	5342	1.71[86%]	2.32[58%]	2.41[40%]	0.76[10%]
C7-6-3a	48:7	18720	1.34[67%]	2.27[57%]	2.48[41%]	0.59[7%]
C7-6-3b	50:7	41735	1.25[63%]	2.19[55%]	2.81[47%]	0.72[9%]
E7A	126:8	12399	1.78[89%]	2.80[70%]	3.38[56%]	3.23[40%]
			speedup[%efficiency] without local stacks			
C6-6-3a	36:6	1394	0.92[46%]	2.00[50%]	2.00[33%]	1.71[21%]
C6-6-3b	50:6	5342	1.27[64%]	2.03[51%]	1.91[32%]	1.67[21%]
C7-6-3a	48:7	18720	1.02[51%]	1.83[46%]	1.89[32%]	2.00[25%]
C7-6-3b	50:7	41735	1.23[61%]	2.14[54%]	2.73[45%]	2.64[33%]
E7A	126:8	12399	1.46[73%]	2.46[62%]	2.77[46%]	2.85[36%]

thread) for 4 cores than for 6 cores for all instances, though the speedup for the 6-core runs is generally slightly higher than the speedup for 4-core runs. For many problem instances the overhead of syncing threads seems to outweigh the benefits of parallelization; notably, though 2 threads may give near-linear speedup on small instances, for larger instances the performance falls off somewhat, sometimes running more slowly than the sequential code.



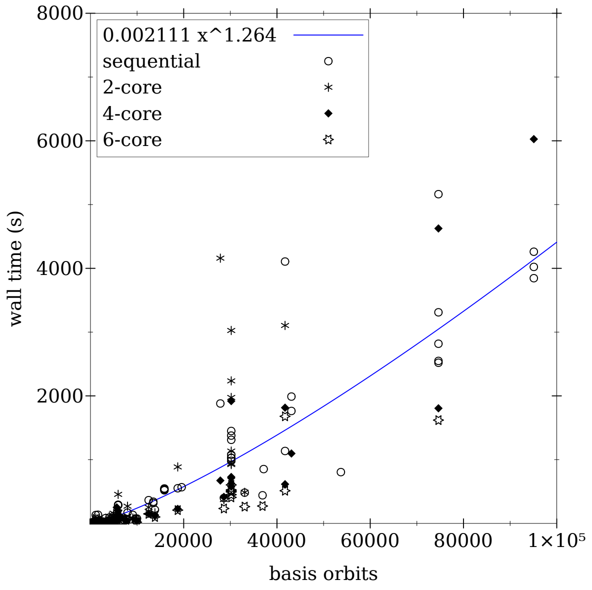


Figure 4.4: Plots up to 10,000 and 100,000 basis orbits of parallel times for Cd-6-3 instances, including sequential, 2, 4, and 6-cores with local stacks.

Chapter 5

Conclusion

This thesis has described a method of performing basis enumeration up to symmetries of hyperplane arrangements, and also experimentally tested the usefulness of various optimizations to this method. I have developed a new program, dubbed `Basil`, based on the approach of Bremner *et al.* [6] for basis enumeration of polyhedra up to symmetries by depth-first search of the graph of cobases constructed by minimum-ratio simplex pivots. This adaptation required definition of a new minimum-ratio test that would work on arrangements instead of polyhedra, and also a new procedure to compute the automorphism group of an arrangement.

Experimental results from running `Basil` show a multiple order of magnitude speedup over the first-generation `Symbal` software of Bremner *et al.* for basis enumeration of polyhedra up to symmetries; this speedup is chiefly due to the use of compiled C++ code instead of the interpreted `GAP` scripts used in

`Symbal`. I have also tested the utility of various programmatic optimizations and mathematical invariants for reducing runtime; use of Gram matrices to compare the angles between hyperplanes in a cobasis has proved to be very effective, and a parallel variant gave decent speedup on 4-6 cores, while gains from using fundamental domain constraints to bound the search space were more modest.

One open problem relating to this research is whether using more of the constraints of the fundamental domain to bound the search space would yield better speedup; elements of the group other than the generators could be used in the existing method to get some more constraints, but to really explore the possibilities of fundamental domain based pruning would require a way to generate a generic point and initial vertex that both fall within the same fundamental domain, instead of just using the initial vertex to generate the fundamental domain constraints.

Another interesting avenue of investigation would be to find an adjacency rule which preserves the neighbourhood adjacency property of Lemma 3.1.1 but returns fewer cobases than the minimum-ratio adjacency presented here; such a rule, if it exists, could significantly speed up `Basil` runtime while decreasing memory usage. If this hypothetical adjacency rule could define the adjacency graph of cobasis orbits rather than cobases, a memory-less reverse search following the model of Avis and Fukuda [4] could be implemented for basis enumeration of hyperplane arrangements up to symmetries.

References

[1] *ACEnet*, `http://www.ace-net.ca/wiki/ACEnet`, September 2011.

[2] David Avis, *lrs home page*, `http://cgm.cs.mcgill.ca/~avis/C/lrs.html`, accessed 26 January 2012.

[3] ———, *Computational experience with the reverse search vertex enumeration algorithm*, Optimization Methods and Software **10** (1998), no. 2, 107–124.

[4] David Avis and Komei Fukuda, *A pivoting algorithm for convex hulls and vertex enumeration of arrangments and polyhedra*, Discrete & Computational Geometry **8** (1992), no. 1, 295–313.

[5] G. E. P. Box and D. R. Cox, *An analysis of transformations*, Journal of the Royal Statistical Society. Series B (Methodological) **26** (1964), no. 2, 211–252.

[6] David Bremner, Mathieu Dutour Sikirić, and Achill Schürmann, *Polyhedral representation conversion up to symmetries*, Polyhedral Compu-

tation (David Avis, David Bremner, and Antoine Deza, eds.), CRM Proceedings & Lecture Notes, American Mathematical Society, 2009, pp. 45–71.

[7] Michel Brion and Michèle Vergne, *Residue formulae, vector partition functions and lattice points in rational polytopes*, Journal of the American Mathematical Society **10** (1997), no. 4, 797–833.

[8] Adrian Brüngger, Ambros Marzetta, Komei Fukuda, and Jurg Nievergelt, *The parallel search bench ZRAM and its applications*, Ann. Oper. Res. **90** (1999), 45–63. MR 1708017

[9] A. Charnes, *The simplex method: optimal set and degeneracy*, An introduction to Linear Programming, Wiley, New York, 1953, pp. 62–70.

[10] George B. Dantzig, *Maximizing a linear function of variables subject to linear inequalities*, Activity Analysis of Production and Allocation (1951), 339–347.

[11] Jack Edmonds and J.-F. Maurras, *Note sur les Q-matrices d'Edmonds*, RAIRO. Recherche opérationnelle **31** (1997), no. 2, 203–209.

[12] IEEE, *POSIX.1c, threads extensions (IEEE std. 1003.1c-1995)*, 1995.

[13] Vipin Kumar and V. Nageshwara Rao, *Parallel depth first search. part ii. analysis*, International Journal of Parallel Programming **16** (1987), 501–519, 10.1007/BF01389001.

[14] Brendan McKay, *The nauty page*, `http://cs.anu.edu.au/~bdm/nauty/`, accessed 26 January 2012.

[15] Message Passing Interface Forum, *MPI: A message-passing interface standard version 2.2*, `http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf`, September 2009.

[16] OpenMP Architecture Review Board, *OpenMP*, `http://openmp.org/wp/`, accessed 16 December 2011.

[17] V. Nageshwara Rao and Vipin Kumar, *Parallel depth first search. part i. implementation*, International Journal of Parallel Programming **16** (1987), 479–499, 10.1007/BF01389000.

[18] Thomas Rehn, *User's guide for PermLib*, `http://www.math.uni-rostock.de/~rehn/software/permlib.html`, October 2011, accessed 26 January 2012.

[19] ———, *User's guide for SymPol*, `http://www.math.uni-rostock.de/~rehn/software/sympol.html`, October 2011, accessed 16 February 2012.

[20] Achill Schürmann, *Computational geometry of positive definite quadratic forms: Polyhedral reduction theories, algorithms, and applications*, 2008.

[21] Mathieu Dutour Sikirić, *Polyhedral home page*, `http://drobilica.irb.hr/~mathieu/Polyhedral/`, accessed 10 May 2012.

[22] András Szenes and Michèle Vergne, *Residue formulae for vector partitions and Euler–Maclaurin sums*, Advances in Applied Mathematics **30** (2003), 295–342.

[23] The GAP Group, *Gap system for computational discrete algebra*, `http://www.gap-system.org`, September 2008, accessed 26 January 2012.

[24] Christophe Weibel, *Implementation and parallelization of a reverse-search algorithm for Minkowski sums*, Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments, 2010, pp. 34–42.

Appendix A

Software Distribution

Code and test data employed in this thesis are available by request from Dr. David Bremner at `bremner@unb.ca`. `Basil` can be compiled by running `make basil` from the main source directory, while the parallel variant can be compiled with `make basilp`. `Basil` depends on the GMP multi-precision arithmetic library and its C++ bindings, as well as the Boost C++ libraries (known to work with version 1.42); `Basil` also packages the `lrslib` and `permlib` libraries it depends on in its source distribution. The test instances used in the tables in Chapter 4 are stored in the `test/table` sub-directory of the source distribution, while the Cd-6-3 instances used for the larger tests are stored in the `test/cube` sub-directory.

Glossary

Arrangement See *hyperplane arrangement*.

Automorphism A symmetry which leaves the points in an object setwise invariant.

Basic variables Variables of a simplex tableau not included in the current cobasis.

Basil A program by the author for basis enumeration up to symmetries of hyperplane arrangements.

Basis enumeration Listing all unique cobases of an arrangement.

Bounding hyperplane The hyperplane for which the linear inequality defining a halfspace is satisfied with equality.

Bounding hyperplane arrangement The set of bounding hyperplanes of the halfspaces defining a polyhedron.

Cobasic variables Variables of a simplex tableau included in the current cobasis.

Cobasis A set of d distinct hyperplanes in an arrangement which meet at a single point.

Degenerate arrangement An arrangement that has at least one degenerate vertex.

Degenerate vertex A vertex with more than d incident hyperplanes.

Edge A 1-dimensional intersection of $d-1$ hyperplanes of an arrangement.

Fundamental domain A convex region of $\mathbb{R}^d$ which contains exactly one representative from each orbit of points under some isometry group.

GAP A freely available computer algebra system including group theoretic functions.

Generating set A subset of a group that the entire group may be expressed as combinations of under the group action.

Generic point A point that is not its own image under any non-identity isometry in a given group.

Gram matrix A matrix encoding the angles between the hyperplanes in an arrangement.

Group A set of elements closed under some action, including an identity element and unique inverses under the group action.

Group action A function combining two members of a group to produce another member of the group. A group is closed under its action.

Group generators The elements of a generating set of a group.

Halfspace The set of points in $\mathbb{R}^d$ satisfying a linear inequality.

Hyperplane The set of points in $\mathbb{R}^d$ satisfying a linear equation.

Hyperplane arrangement The union of a set of hyperplanes.

Identity element The group element which leaves all other group elements invariant under the group action.

Image The result of applying a group element to an object.

Incident hyperplane A hyperplane which contains a given vertex.

Inverse element The group element which maps a given group element to the identity under the group action.

Isometry A distance preserving symmetry.

Linear programming A field of computer science concerned with optimization problems with linear constraints.

Lrslib A C library by David Avis for construction and manipulations of simplex tableax.

Minimum ratio test A test to determine the nearest of multiple cobases with a simplex tableau.

Nauty A program by Brendan McKay for computing graph automorphisms.

Normal vector A vector perpendicular to a hyperplane.

Orbit The set of images of an object under a group action.

Permlib A C++ library by Thomas Rehn for group theoretic computations.

Permutation A one-to-one function of n items onto the same n items.

Permutation group A group consisting of permutations.

Pivot A move between two cobases which share an edge.

Polyhedron The intersection of a set of halfspaces.

Polytope A bounded polyhedron.

Reverse search A memory-less graph traversal algorithm devised by Avis and Fukuda.

Proper subgroup A subgroup which does not contain all the elements of the full group.

Simple arrangement An arrangement with no degenerate vertices.

Simplex tableau A representation from linear programming of a polyhedron or arrangement which simultaneously represents a cobasis of the (bounding) arrangement.

Slack variables Variables introduced in a simplex tableau to represent distance from the hyperplanes of the arrangement.

Subgroup A group S consisting of a subset of the elements of some other group G where S is closed under the group action of G.

Symbal A program by David Bremner for basis enumeration up to symmetries of polyhedra.

Symmetry A geometric transformation leaving the object acted upon invariant in some sense.

Symmetry group A group consisting of symmetries.

Total representative degree (TRD) The sum of the degrees of a representative of each orbit of cobases.

Vertex The point of intersection of at least d distinct hyperplanes of a hyperplane arrangement.

Vita

Candidate's full name: Aaron Moss
University attended (with dates and degrees obtained): University of New Brunswick, Bachelor of Computer Science, 2006-2011
Publications: Aaron Moss and David Bremner. "Basis Enumeration of Hyperplane Arrangements Up to Symmetries". *24th Proceedings of the Canadian Conference on Computational Geometry.* 151–156 (2012)
Aaron Moss, Sandy Liu and Rene Richard. "A Unified Authentication Framework for Accessing Heterogeneous Web Services". *International Journal of Web Services Practices.* 3 : 185–190 (2008) [earlier published in proceedings of 4th International Conference on Next Generation Web Services Practices. p. 117–122]
Conference Presentations: Aaron Moss. Basis enumeration of hyperplane arrangements up to symmetries. Science Atlantic Mathematics, Statistics and Computer Science Conference. (2011)