Basis Enumeration of Hyperplane Arrangements Up to Symmetries*

Aaron Moss† David Bremner‡

Abstract

Given a symmetry group acting on the hyperplanes of an arrangement, our goal is to report a single basis from each orbit of bases induced by this group. In this paper we extend previous techniques for finding the (feasible) bases of polyhedra up to symmetry, and for computing the symmetry groups of polyhedra, to the setting of hyperplane arrangements. We present some preliminary experiments with a C++ implementation of these techniques called `Basil`. These results show substantial speedups compared to a previous polyhedra only system using the computer algebra system `GAP`. We also measure the speedup due to a Gram matrix invariant, and show that the overhead of symmetry testing, while substantial, is dominated by the savings in reduced pivoting.

1 Introduction

Because of the large number of vertices and bases of a hyperplane arrangement, it is natural to consider generating these objects up to symmetry. One application for finding (orbits of) bases of hyperplane arrangements is the 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 [6, 14] suggests may be quicker than current approaches.

This paper describes the design and implementation of a program for basis enumeration of hyperplane arrangements up to symmetries. This program, called `Basil` ("**Basi**s **l**ist") adapts the pivoting method of Bremner, Sikirić, and Schürmann [5] for basis enumeration of polyhedra 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.* [7].

Sikirić's `Polyhedral` [13] and Rehn's `Sympol` [12] contain 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 smaller polyhedra, and are quite different from our pivoting approach. For a survey of approaches for vertex enumeration up to symmetries of polyhedra (and the dual problem of facet enumeration up to symmetries), see [5], which covers both the recursive decomposition and pivoting approaches.

2 Background

2.1 Arrangements & Polyhedra

The structures discussed here 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 (though in this paper all explicitly defined vectors have rational coordinates for reasons of efficiency and ease of computation). 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, 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* $P_1, P_2, \cdots, P_n$; 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 a halfspace, while the set of bounding hyperplanes of the halfspaces defining a polyhedron is called its *bounding hyperplane arrangement*. The *size* of an arrangement is the number of hyperplanes n, while all arrangments considered will be full rank and thus have dimension d, the dimension of the underlying space. 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 (We use here the name *cobasis* from linear programming for what is typically called a *basis* in geometry for consistency with the terminology of our linear programming-based implementation). Hyperplanes which contain a vertex are

*Research partially supported by NSERC. Computational resources provided by ACEnet.

†Faculty of Computer Science, University of New Brunswick, moss.aaron@unb.ca

‡Faculty of Computer Science, University of New Brunswick, bremner@unb.ca

said to be *incident* to that vertex. It should be noted that d hyperplanes meet in a single point if and only if the equations defining those hyperplanes are linearly independent. The problem of *basis enumeration* is thus to list all the unique cobases of a hyperplane arrangement. 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 (*i.e.* if more than d hyperplanes of the (bounding) arrangement meet at that point); such a vertex is called *degenerate*. A polyhedron or arrangement with no degenerate vertices is *simple*, for such a polyhedron the *vertex enumeration* problem (reporting each unique vertex) is equivalent to the basis enumeration problem. For non-simple (*degenerate*) polyhedra and arrangements, the vertex enumeration problem can be solved by basis enumeration, though some method must be employed to filter out duplicate vertices.

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.

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, 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 [8], 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 first step to convert a polyhedron to tableau form is to add n new *slack variables* $\{x_{d+1}, x_{d+2}, \cdots, x_{d+n}\}$ 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" between the bounding hyperplane of each halfspace in the polyhedron and the vertex represented by the tableau; 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, each of the inequalities $\boldsymbol{a}_i^\top \boldsymbol{x} \geq b_i$ defining the halfspace P_i in $\mathcal{P}$ is rewritten as an equation $x_{d+i} = -b_i + \boldsymbol{a}_i^\top \boldsymbol{x}$, 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:

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

The *basic* variables of the tableau are the set of variables x_i which are defined by the equations represented by the rows of the tableau; the set of variables x_j which are the column variables those equations are written in terms of are the *cobasic* variables of the tableau[1]. With the addition of auxiliary structures to the tableau matrix M to remember the current sets of basic and cobasic variables, the data structures needed for the simplex method are complete. 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. After the initial setup of the tableau is complete, the decision variables are moved into the basis, with slack variables replacing them in the cobasis. When this process is completed, the current vertex represented by the tableau 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 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, x_l,[2] traversing an edge of the arrangement or polyhedron.

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 the 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 constant term $\boldsymbol{b}_l$ of the

[1] 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.

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

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. By setting x_e to $r = \boldsymbol{b}_l/a_{l,e}$, x_l is 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 magnitude 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*, moving to another cobasis of the same vertex.

2.2 Symmetries

Many interesting hyperplane arrangements have a significant number of *automorphisms*: geometric symmetries (*e.g.* reflections and rotations) 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[3]. Taking the 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 we are particularly interested in are *isometries*, distance preserving symmetries.

One property of isometric cobases is that, given some distance metric, the set of angles according to that metric between each pair of hyperplanes which meet in a single cobasis is setwise invariant under the action of any symmetry in the automorphism group. 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* $A = (a_{i,j})$ is constructed such that $a_{i,j}$ 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 cobasis hyperplanes. If each row of this submatrix is sorted, and then the rows of the submatrix are lexicographically sorted, the resulting submatrix uniquely represents the angles between each pair of hyperplanes in the cobasis, and pairs of such matrices can be compared for equality swiftly. Equality of Gram submatrices does not guarantee that the corresponding cobases are symmetric, but inequality of Gram submatrices does show that the cobases are not symmetric.

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 the indices of the halfspaces defining the polyhedron by σ. A full proof of this can be found in [5], but intuitively the rows and columns of the Gram matrix correspond to the halfspaces defining the polyhedron. As the Gram matrix encodes the distances between each pair of bounding hyperplanes as angles, 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.

One problem we encountered in generating Gram matrices for hyperplane arrangements that does not occur in the polyhedral case is that any hyperplane $A = \{\boldsymbol{x} \mid \boldsymbol{a}^\top \boldsymbol{x} = b\}$ can be replaced by its *negation* $\bar{A} = \{\boldsymbol{x} \mid -\boldsymbol{a}^\top \boldsymbol{x} = -b\}$ without changing the arrangement. However, the angle produced by $\bar{A}$ and another hyperplane B is the supplement of the angle produced by A and B, in general a distinct angle. When using the Gram matrix to detect 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 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 symmetry 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.

3 Algorithms

The essential idea of our algorithm for basis enumeration up to symmetries is to explore the hyperplane arrangement outward, moving from an initial cobasis to its adjacent cobases, pruning this search tree when a cobasis symmetric to one already found is reached; a full description is in Algorithm 1. The subroutine InitialCobasis() returns any coba-

[3] Automorphisms on polyhedra can be considered analogously.

sis of the arrangment; ADJACENT($\mathbf{B}$) returns a list of all cobases $\mathbf{B}_i$ which are adjacent to a cobasis $\mathbf{B}$. INNEWORBIT($\mathbf{B}$) tests whether a cobasis $\mathbf{B}$ is in an orbit already discovered, while REPORT($\mathbf{B}$) is used to output a newly discovered cobasis $\mathbf{B}$. 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 description of the algorithm.

Algorithm 1 Basis orbit enumeration algorithm

```
function SymmetricBasisSearch(void)
                    ▷ find a cobasis of the arrangement
    B ← InitialCobasis()
                    ▷ explore outward from this cobasis
    S ← a stack of cobases, initially empty
    Report(B)
    PushCobasis(S, B)
    repeat
        B ← PopCobasis(S)
  ▷ search for new orbit representatives adjacent to B
        for all B_i ∈ Adjacent(B) do
            if InNewOrbit(B_i) then
                Report(B_i)
                PushCobasis(S, B_i)
            end if
        end for
    until Empty(S)
end function
```

The reader familiar with pivoting algorithms will remark upon the absence of perturbation from Algorithm 1. 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. Here our goal is to find all orbits of bases, so standard symbolic perturbation schemes that ignore the symmetry group are unlikely to work well. In [5] the authors 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, we do not discuss it here; some of our experimental data (the E7-j examples in Table 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 derived from most linear programming textbooks, though some simple modifications may be needed to convert processes intended for use on polyhedra to work with arrangements (such as our implementation of ADJACENT($\mathbf{B}$), detailed below). For PUSHCOBASIS($S, \mathbf{B}$) and POPCOBASIS(S), our implementation keeps an internal stack of pivots performed, reversing those pivots as necessary to return to an earlier cobasis.

Our implementation of ADJACENT($\mathbf{B}$) is based on the minimum ratio test. Our rule 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, our method reports all the basic variables that are already zero as possible leaving variables (these represent degenerate pivots), as well as all the basic variables x_i which have a minimal magnitude ratio $\boldsymbol{b}_i/a_{i,e}$ in both the positive and negative directions. Taking both positive and negative ratio ensures that new adjacent cobases are found on either side of the hyperplane corresponding to x_e.

4 Implementation & Results

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 [9]. The implementation described in this paper, `Basil`[4], 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.* [5], which performs basis enumeration up to symmetries on polyhedra. However, where `Symbal` is implemented in the `GAP` [15] computer algebra system with calls to C libraries wrapping `lrslib` for simplex operations and McKay's `Nauty` [10] for graph automorphism calculations, `Basil` has been re-implemented in C++, using Rehn's `permlib` [11] library to replace the both the group theoretic capabilities of `GAP` used by `Symbal` (which it should be pointed are relatively simple orbit membership tests) and the automorphism code in `Nauty` with matrix automorphism routines. `Basil` also uses `lrslib` for its tableau implementation.

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 paper, the experimental results shown in Table 1 compare the relative performance of `Basil` and `Symbal` for basis enumeration up to symmetries of a set of polyhedra. The Ey instances discussed are the Dirchlet-Voronoi-cells (DV-cells) of the root lattices E_y, as described in section 7.2 of [5]. As can be seen from these results, `Basil` is generally about two orders of magnitude faster, attributable to the lower overhead of C++ execution than the `GAP` interpreter and the more sophisticated and efficient data structures available in C++ than `GAP`. These numbers represent only the CPU time of both programs; this is a fairly accurate representation of `Basil`'s runtime, but underestimates `Symbal`'s

[4] Software and test input available by request.

Table 1: Comparison of Basil & Symbal
(bases & vertices count *orbits*)

Problem	n:d	bases:verts	Bas(s)	Sym(s)
E7	126:8	32:2	1.82	282.16
E7-3	126:8	82:58	0.59	12.41
E7-7	126:8	1195:106	19.88	1507.72
E7-65	126:8	356:14	7.88	1308.07
E7-102	126:8	41:7	1.27	223.97
E8	240:8	2:2	0.41	2.13

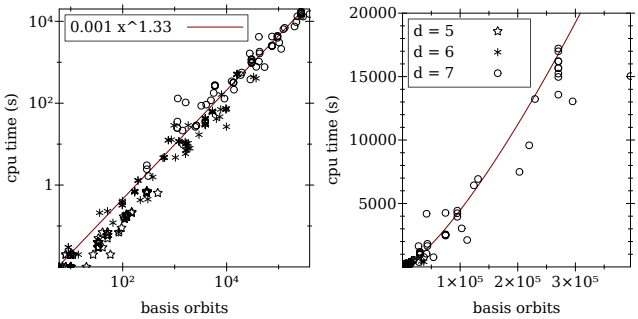

Figure 1: Time for Cd-6-3 instances by # basis orbits, on both log-log and linear plots.

by about half due to overhead from the interprocess communication needed to connect `GAP` to the external C libraries used. Timing results reported are from the Placentia ACEnet cluster [1], which has 2.3–3.0 GHz AMD Opteron processors.

Table 2 shows some early performance results for `Basil` on the arrangements. Problems DxA and EyA are the bounding hyperplane arrangements for the DV-cells of the root lattices D_x and E_y, 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 with at least y orbits. Figure 1 plots runtime for 268 Cx-6-3 instances. At least for these examples, it seems that suggests that `Basil`'s runtime is super-linear but sub-quadratic in the number of orbits output (as opposed to the total number of bases, which can be exponentially larger). Table 3 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 pivoting approach implemented in `Basil` (and `Symbal`) differs from that proposed by Avis and Fukuda [4] for the non-symmetric vertex enumeration problem in that their *reverse search* 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. For the symmetric case, we expect there to be

Table 2: Basil Timing Results
(bases & vertices count *orbits*)

Problem	n:d	bases:verts	Bas(s)
D4A	24:4	12:7	0.02
D5A	40:5	104:25	0.50
C5-6-3a	25:5	291:36	0.70
C5-6-3b	16:5	51:16	0.05
C6-6-3a	15:6	9:1	0.03
C6-6-3b	36:6	1394:91	13.90
C6-6-3c	50:6	5342:157	63.65
C7-6-3a	48:7	18720:140	456.59
E7A	126:8	12399:227	1570.66

Table 3: Non-Symmetric Timing Results
(*all* bases & vertices counted)

Problem	n:d	bases:verts	Bas(s)
D4A	24:4	5028:863	23.62
C5-6-3a	25:5	24444:852	120.80
C5-6-3b	16:5	3005:234	4.37
C6-6-3a	15:6	2530:1	16.31

relatively few orbits of cobases, allowing `Basil` to keep representatives of each in memory, and thus have not yet investigated a memory-less reverse search for this problem. Additionally, the limiting factor on the size of instances we 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. Our 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.

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 shrink the set of cobases that must be tested for symmetries. 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 recently seen cobases to avoid needing to retest previous cobases (for instance, the cobasis that was pivoted from to reach the current cobasis).

`Basil` also uses the Gram submatrix to differentiate cobases; representatives of known cobasis orbits are stored in a hash table indexed by the corresponding Gram submatrix. Comparing each newly discovered cobasis only to the cobases having Gram submatrices

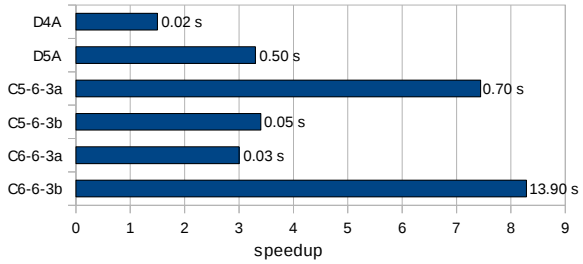

Figure 2: Speedup from using Gram matrix (bar labels are runtime *without* Gram matrix)

which are equivalent under the sorting procedure described earlier 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, as seen in Figure 2, while when activated the Gram matrix computations consume about 5% of the execution time of `Basil`.

5 Conclusion & Future Work

Basis enumeration seems to be an easier problem than the closely related problem of vertex enumeration. A pivoting algorithm can effectively explore the graph of adjacent bases. The main practical difficulty is the typically enormous output size from even moderate sized input. In certain applications, it suffices to generate one basis from each orbit under some natural symmetry group. In this paper we have described the enhancement of the symmetric pivoting software `Symbal` to produce a second generation symmetric pivoting software `Basil`. `Basil` is a native C++ application, and the speedup compared to `Symbal` can be seen as a validation of the use of C++ instead of the computer algebra system `GAP`, enabled by use of the `permlib` C++ library for group theoretic computations. The main motivation for developing `Basil` was to be able to generate orbit representatives of bases in hyperplane arrangements, based on a perceived need for this capability in certain novel approaches to integer programming. The extension from polyhedra to arrangements required defining a new ratio-test, and a modified procedure compute the symmetry group.

As the expense of the group theoretic calculations is the current limiting factor on the problem size that is feasible to solve, future directions for this research include a parallel implementation of `Basil` to bring greater computational power to bear on the problem, as well as research into invariants which may be cheaper to test than cobasis isomorphism.

Another way to reduce group theoretic calculations is to construct or approximate a *fundamental domain*, a convex cell F such that each orbit of cobases has exactly one representative in F. Such a cell can be constructed by techniques closely related to Voronoi diagrams, and could be used to prune the search for adjacent bases.

References

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

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

[3] D. Avis. Computational experience with the reverse search vertex enumeration algorithm. *Optimization Methods and Software*, 10(2):107–124, 1998.

[4] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangments and polyhedra. *Discrete & Computational Geometry*, 8(1):295–313, 1992.

[5] D. Bremner, M. D. Sikirić, and A. Schürmann. Polyhedral representation conversion up to symmetries. In D. Avis, D. Bremner, and A. Deza, editors, *Polyhedral Computation*, pages 45–71. CRM Proceedings & Lecture Notes, American Mathematical Society, 2009.

[6] M. Brion and M. Vergne. Residue formulae, vector partition functions and lattice points in rational polytopes. *Journal of the American Mathematical Society*, 10(4):797–833, October 1997.

[7] A. Charnes. The simplex method: optimal set and degeneracy. In *An introduction to Linear Programming*, Lecture VI, pages 62–70. Wiley, New York, 1953.

[8] G. B. Dantzig. Maximizing a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, pages 339–347, 1951.

[9] J. Edmonds and J.-F. Maurras. Note sur les Q-matrices d'Edmonds. *RAIRO. Recherche opérationnelle*, 31(2):203–209, 1997.

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

[11] T. Rehn. *User's Guide for PermLib*. `http://www.math.uni-rostock.de/~rehn/software/permlib.html`, October 2011. accessed 26 January 2012.

[12] T. Rehn. *User's Guide for SymPol*. `http://www.math.uni-rostock.de/~rehn/software/sympol.html`, October 2011. accessed 16 February 2012.

[13] M. D. Sikirić. *Polyhedral home page*. `http://drobilica.irb.hr/~mathieu/Polyhedral/`. accessed 10 May 2012.

[14] A. Szenes and M. Vergne. Residue formulae for vector partitions and Euler–Maclaurin sums. *Advances in Applied Mathematics*, 30:295–342, January 2003.

[15] The GAP Group. *GAP System for Computational Discrete Algebra*. `http://www.gap-system.org`, September 2008. accessed 26 January 2012.