
Proceedings Paper:
Abdullahi, SD, Dyer, ME and Proll, LG (2003) Listing Vertices of Simple Polyhedra Associated with Dual LI(2) Systems. In: Calude, CS, Dinneen, MJ and Vajnovszki, V, (eds.) Discrete Mathematics and Theoretical Computer Science. 4th International Conference, DMTCS 2003, 07-12 Jul 2003, Dijon, France. Lecture Notes in Computer Science (2731). Springer-Verlag , Berlin, Germany , pp. 89-96. ISBN 978-3-540-40505-4

Listing Vertices of Simple Polyhedra Associated with Dual *LI(2)* Systems

S D Abdullahi, M E Dyer and L G Proll

School of Computing,
University of Leeds, Leeds, LS2 9JT, UK
{sammani, dyer, lgp}@comp.leeds.ac.uk

Abstract. We present an $O(nv)$ Basis Oriented Pivoting (BOP) algorithm for enumerating vertices of simple polyhedra associated with dual $LI(2)$ systems. The algorithm is based on a characterization of their graphical basis structures, whose set of edges are shown to consist of vertex-disjoint components that are either a tree or a subgraph with only one cycle. The algorithm generates vertices via operations on the basis graph, rather than by simplex transformations.

1 Introduction

The Vertex Enumeration (VE) problem is that of determining all the vertices of a convex polyhedron described by a set of linear inequalities or equations. This problem, together with its dual, that of finding the convex hull of a set of points, is of interest due to its application to problems in optimization and computational geometry [8]. The VE problem has been the object of substantial research effort and a considerable number of algorithms have been proposed for its solution, (see [1, 8, 11, 15] for useful reviews). The most successful empirically appear to be those of Avis and Fukuda [4], Chen et al. [5] and Dyer [9].

Dyer [9] shows that, at least for simple polytopes, *basis oriented pivoting* (BOP) algorithms have better worst-case time complexity bounds than do other approaches. The main inspiration of BOP algorithms is the simplex method of linear programming (see [7]). Starting from an arbitrary vertex, BOP methods examine the columns of the associated simplex tableaux to discover new vertices adjacent to those found so far. They proceed iteratively until a spanning tree of the edge-vertex graph of the polyhedron has been constructed. Pivoting methods exploit the correspondence between the vertices and basic feasible solutions (BFS); indeed what they actually list are BFS's. This correspondence is *1-1* for simple polytopes, and *one-to-many* for degenerate ones. Hence degeneracy is an issue for pivoting methods.

Most VE algorithms are applicable to general constraint systems. However, optimisation algorithms which take advantage of special structures in the coefficient matrix, for example in network LP's, can heavily outperform the simplex method, see for example [13]. It is natural to explore whether this might also be the case for the VE problem. Provan's work on network polyhedra [17] suggests that this could be a useful line of

research. Provan has provided a VE algorithm for network polyhedra, which are inherently degenerate, and shows that vertices can be listed in time $O(Ev^2)$, where E is the number of edges in the network and v is the number of vertices, plus the time to find the first vertex of the network polyhedron. An implementation of, and computational experience with, his primal algorithm is discussed in [1].

Linear systems of inequalities with at most two non-zeros per constraint, called *LI(2)*, are known to be related to network systems. Several algorithms have been developed for the *LI(2)* feasibility problem, examples are [3, 6, 12, 18]. These algorithms either show that the *LI(2)* system has no feasible solution or provide a single feasible solution. It is not obvious that these methods can be extended to determine all basic feasible solutions. However, some of these methods, such as Hochbaum and Naor's [12] whose backbone is the Fourier-Motzkin elimination method, provide a useful platform to explore. For example, Williams [19] suggests one of such avenues, the dual Fourier-Motzkin method, to enumerate vertices of polyhedra. An algorithmic description of Williams' method and its implementation is discussed in [1]. This method is not computationally promising or attractive performance-wise because the number of intermediate variables may grow exponentially in the process of eliminating constraints. Also, a lot of energy can be wasted in eliminating constraints that may not be important after all, i.e. redundant constraints.

In this paper we show that exploiting the properties of the coefficient matrix associated with linear system of inequalities with at most two non-zeros per columns (which we call dual *LI(2)*) brings substantial advantage for vertex listing. We present an algorithm that can be classified as a BOP method for simple polyhedra, (see [8]), whose running time is *O(nv)* where n is the number of edges (variables) in the constraint graph and v is the number of vertices, plus the time to find the initial vertex. The backbone of our algorithm is a proposition which we present below:

2 Basis Structure for Dual *LI(2)*

Network programs have coefficient matrices with at most two non-zero coefficients per column, both of magnitude 1 and if there are two, they are of opposite sign. It is well known that the bases of such matrices can be represented as spanning trees of a related graph [13]. This fact is at the core of Provan's method for listing vertices of network polyhedra. It seems reasonable to ask: what does a basis of a constraint system with no more than two general coefficients per column look like?

Proposition: Let $B = \{x \in \mathbb{R}^n : Ax = b, b \in \mathbb{R}^m, \ \ x \geq 0\}$ be a linear program with at most two non-zeros per column in A, where A is an $m \times n$ matrix; x an $n \times 1$ matrix and b an $m \times 1$ matrix, with $n > m$. Let G be a graph on m vertices corresponding to the rows of A, with an edge corresponding to each column of A (i.e. an edge between i and j if these are the non-zero positions in the column). Then the set of edges in G corresponding to any basis of A has vertex-disjoint components which are either:

1. A tree OR
2. Contain exactly one cycle.

$Proof$:

Suppose A is the matrix with 2 nonzeros per column. Each column of A can be represented as follows:

$$c_{i,j} = \begin{bmatrix} 0 \\ 0 \\ . \\ a_{i,j} \\ . \\ d_{i,j} \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

We can start building up components of the basis by rearranging rows and columns (using only the edges that correspond to the basis). At each stage we add a column which has one non-zero in a new-row, but may also have a non-zero in a row already used. Obviously, the set of edges that form the basis in this case cannot have more than 2 edges (variables) that are connected to 2 adjacent nodes, otherwise we have 3 vectors spanning $\mathbb{R}^2$.

Rearranging the rows and columns of B we obtain a matrix of the form:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & . & 0 & 0 & . & 0 \\ d_{2,1} & 0 & . & 0 & 0 & . & 0 \\ 0 & d_{3,2} & . & a_{3,9} & 0 & . & a_{3,n} \\ 0 & 0 & . & 0 & 0 & . & 0 \\ 0 & 0 & . & d_{5,9} & a_{5,10} & . & d_{5,n} \\ 0 & 0 & . & 0 & d_{6,10} & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & . & 0 & 0 & . & 0 \end{bmatrix} \tag{2}$$

As we can see from above matrix, we reach an $upper\ triangular\ matrix$ with one additional row as the component, when a stage is reached where one can add no more columns.

If we have r rows, then we have $r-1$ linearly independent vectors, and the structures of the arranged matrix yield a tree with $r-1$ edges plus (possibly) one edge giving r spanning vectors. This is a component in G with one cycle (if we terminate with a tree then these rows have rank deficiency 1). We can remove this component and start again. We obtain a collection of vertex disjoint components, each having at most one cycle.

Indeed, the basis corresponds to a collection of vertex disjoint components of G each of which may be a tree or contains at most one cycle. □

Corollary: If i is the number of component trees for any basis, then the rank of the matrix A is given by:

$$Rank(A) = m - i \tag{3}$$

(i.e i components are missing rank 1).

$Proof$:
Follows from the proof of above Proposition.

3 The Algorithm

Suppose we have a system of the form:

$$Ax = b, \;\; x \geq 0 \tag{4}$$

where A is $m \times n$ matrix with no more than 2 nonzeros per column. The graph $G(A)$ associated with (4) has:

- m nodes, labelled R_1, R_2,...., R_m, each associated with a row of A;
- n (undirected) edges, labelled E_1, E_2,....,E_n, each associated with a column of A;
- each edge E_j connecting nodes R_i, R_k if $a_{ij} \neq 0$ and $a_{kj} \neq 0$ or forming a loop at node R_i if $a_{ij} \neq 0$ and $a_{kj} = 0, \forall k$.

The algorithm as described here is valid for simple bounded polytopes, but is later extended to deal with unbounded polyhedra. The description uses a_j to represent the j^{th} column of A, B to represent a basis and β to represent its index set, $G(B)$ to represent the subgraph of $G(A)$ corresponding to B and V to represent a vertex of (4). We use *gamma sets* to control the enumeration. These record the set of edges that may lead from a vertex to previously unknown vertices. **EnumerateVertices** uses a subsidiary routine **SOLVE**, whose purpose is to solve sets of linear equations via the components of the basis graph. **SOLVE** is described separately.

EnumerateVertices

1. Compute a basic feasible solution to (4), by LP if necessary. Let,

$$\beta_1 \leftarrow \{\text{indices of basic edges}\}, \;\; \gamma_1 \leftarrow \{\text{indices of non} - \text{basic edges}\}$$

$$L \leftarrow \{(\beta_1, \gamma_1)\}, \quad p \leftarrow 1, \quad r \leftarrow 1$$

2. Construct $G(B_r)$ from β_r and determine its components
3. Determine V_r using $\mathbf{SOLVE}(G(B_r), b, x)$ and output it

4. $\forall j \in \gamma_r$,
   - determine $a_j^{'}$ using $\mathbf{SOLVE}(G(B_r), a_j, a_j^{'})$
   - Perform the simplex ratio test, so that if $k = argmin\{\frac{V_{ri}}{a_{ji}^{'}} : a_{ji}^{'} > 0\}$,
     the basic edge x_k leaves basis B_r
   - $\beta \leftarrow \beta_r \cup \{j\} - \{k\}$
   - $\gamma \leftarrow \{1, 2, ..., n\} - (\beta \cup \{k\})$
   - $\gamma_r \leftarrow \gamma_r - \{j\}$
   - If $\exists t \in \{1, 2, ..., p\}$ such that $\beta \equiv \beta_t$, $\gamma_t \leftarrow \gamma_t - \{k\}$
   - else $p \leftarrow p + 1, \quad L \leftarrow L \cup \{(\beta, \gamma)\}$
5. $r \leftarrow r + 1$
   If $r \leq p$, goto 2
   Stop. $\square$

SOLVE(G, w, x)

Let G consist of nodes $i = 1, 2, ...v$ labelled with w_i and edges $(i, j, k),\ i \leq j$, associated with x_k and labelled with (a_{ik}, a_{jk}).

For each component of G:

(a) if the component is a simple loop comprising edge (i, i, k), $x_k \leftarrow w_i / a_{ik}$

(b) if the component is a tree
repeat until all tree edges deleted
if i is a leaf node with incident edge (i, j, k) or (j, i, k)
$x_k \leftarrow w_i / a_{ik}, \quad w_j \leftarrow w_j - a_{jk} x_k$
delete incident edge

(c) if the component contains a cycle,
1. remove nodes of degree 1 as in (b)
2. if a simple loop remains, apply (a)
3. if a cycle $(i_1, i_2, k_1), (i_2, i_3, k_2), ..., (i_{t-1}, i_t, k_{t-1})$, where $i_t = i_1$, remains, solve parametrically for $x_{k_1}, x_{k_2}, ..., x_{k_{t-1}}$ as follows:
   $x_{k_1} \leftarrow \lambda$
   for s = 2,t
   $w_{i_s} \leftarrow w_{i_s} - a_{i_s k_{s-1}} x_{k_{s-1}}$
   $x_{k_{s-1}} \leftarrow w_{i_s} / a_{i_s k_{s-1}}$
   Solve $x_{k_t} = \lambda$ for λ, and hence determine $x_{k_2}, ..., x_{k_{t-1}}$. $\square$

The algorithm enumerates vertices of the polyhedron associated with (4) in a similar fashion to that of Dyer [9] for more general polyhedra. The principal differences are:

(a) vertices are generated via operations on the basis graph, rather than by simplex transformations;

(b) the spanning tree of the feasible-basis graph of the polyhedron is constructed breadth-first rather than depth-first;

(c) the adjacency test is performed via a hash table.

The advantage of using breadth-first search is that, for each basis, the components of its associated graph need be determined once only. The empirical efficiency of any VE algorithm depends on the accounting procedure, or adjacency test, used to ensure that vertices are neither omitted nor repeated in the enumeration procedure. Dyer and Proll [10] observed that approximately 90% of the execution time of their algorithm was spent in this phase. Dyer's algorithm [9] employs an AVL tree [14] to provide an elegant method for checking adjacency. However Ong et al [16] have shown that a cruder method based on a hash table data structure gives improved empirical performance. In this context, hashing can be used within Step 4 of our algorithm to test equivalence of bases. We use a binary encoding of the basis index set, i.e. $i \in \beta \Leftarrow r_{i-1} = 1$, (where r is the encoding of the vertex v and r_i is its ith bit) which can be hashed using, for example, a function of modulo type $h(r) = r \ mod \ p + 1$ (where h is the hash value of r and p is a prime number). Clearly if a basis hashes to an empty cell in the hash table, it must represent an undiscovered basis. On the other hand, if a basis hashes to an occupied cell, it may or may not represent a newly discovered basis. We can establish this by comparing the basis index set with those of other bases occupying this cell.

4 Unbounded Polyhedra

EnumerateVertices as described above fails at Step 4 if $\exists i \in \gamma_r$ such that $a^{'}_{ji} \leq 0, \forall i$ because there is then no valid primal simplex pivot. This characterises the existence of an unbounded edge incident at V_r. We can deal with this by modifying Step 4 to be:

$\forall j \in \gamma_r$,
determine $a^{'}_{j}$ using **SOLVE**$(G(B_r), a_j, a^{'}_{j})$
if $\{a^{'}_{ji} : a^{'}_{ji} > 0\} = \emptyset$ then
    $\gamma_r \leftarrow \gamma_r - j$
    output edge details
else
    $<$ as before $>$. $\square$

The complexity analysis of above algorithm is described below:

5 Complexity Analysis

The complexity of **EnumerateVertices**, in the absence of degeneracy, can be analysed as follows. In each loop, all computations are bounded by the number of edges in $G(B)$, i.e. $O(m)$, except for the step which identifies the edge to enter the new basis. This computation is $O(n)$, since n is the number of edges in $G(A)$. Assuming a good hash function [2] for the test for adjacency, this step will also be $O(m)$ at worst. Thus each loop computation is $O(n)$ (assuming $n \geq m$). If there are v nondegenerate vertices in total, the complexity of the algorithm will therefore be $O(nv)$. As normal with this type of analysis, we exclude the time to find an initial feasible basis.

6 Conclusion

In this paper we have proved an important proposition which characterizes the basis structure of dual *LI(2)* systems. The proposition was used to develop a new BOP algorithm for enumerating vertices of simple polyhedra associated with dual *LI(2)* systems. We have shown that the running time of the algorithm is linear per vertex *O(nv)*. This has answered an open problem raised in [17]. Like all BOP algorithms, the algorithm described here may experience difficulty with non-simple, or degenerate, polyhedra. Explicit perturbation of the constraints can often provide a pragmatic solution to these difficulties for mildly degenerate polyhedra but not for highly denerate ones. In [1], we show how the algorithm can be modified to deal properly with the issues arising from degeneracy.

References

1. S D Abdullahi. *Vertex Enumeration and Counting for Certain Classes of Polyhedra*. PhD thesis, School of Computing, The University of Leeds, 2002.
2. A V Aho, J E Hopcroft, and J D Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, Mass, 1982.
3. B Aspvall and Y Shiloach. Polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 9:827–845, 1980.
4. D Avis and K Fukuda. A pivoting algorithm for convex hulls and vertex enumeration for arrangements and polyhedra. *Discrete Computational Geometry*, 8:295–313, 1992.
5. P-C Chen, P Hansen and B Jaumard. On-line and off-line vertex enumeration by adjacency lists. *Operations Research Letters*, 10:403–409, 1991.
6. E Cohen and N Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 23:1313–1347, 1994.
7. G B Dantzig. *Linear Programming and Extensions*. Princeton University Press, N.J., 1963.
8. M E Dyer. *Vertex Enumeration in Mathematical Programming: Methods and Applications*. PhD thesis, The University of Leeds, October 1979.
9. M E Dyer. The complexity of vertex enumeration methods. *Mathematics of Operations Research*, 8:381–402, 1983.
10. M E Dyer and L G Proll. Vertex enumeration in convex polyhedra: a comparative computational study. In T B Boffey, editor, *Proceedings of the CP77 Combinatorial Programming Conference*, pages 23–43, University of Liverpool, Liverpool, 1977.
11. M E Dyer and LG Proll. An algorithm for determining all extreme points of a convex polytope. *Mathematical Programming*, 12:81–96, 1977.
12. D S Hochbaum and J Naor. Simple and fast algorithms for linear and integer program with two variables per inequality. *SIAM Journal of Computing*, 23:1179–1192, 1994.
13. J L Kennington and R V Helgason. *Algorithms for Network Programming*. John Wiley and Sons, Inc., 1980.
14. D E Knuth. *The Art of Computer Programming Vol. 3: Sorting and Searching*. Addison-Wesley, 1973.
15. T H Mattheiss and D S Rubin. A survey and comparison of methods for finding all vertices of convex polyhedral sets. *Mathematics of Operations Research*, 5:167–185, 1980.
16. S B Ong, M E Dyer, and L G Proll. A comparative study of three vertex enumeration algorithms. Technical report, School of Computer Studies, University of Leeds, 1996.
17. J Scott Provan. Efficient enumeration of the vertices of polyhedra associated with network LP's. *Mathematical Programming*, 64:47–64, 1994.

18. R Shostak. Deciding linear inequalities by computing loop residues. *Journal of the Association for Computing Machinery*, 28:769–779, 1981.
19. H P Williams. Fourier's method of linear programming and its dual. *American Mathematical Monthly*, 93:681–694, 1986.