Mathematical Programming 12 (1977) 81–96.
North-Holland Publishing Company

AN ALGORITHM FOR DETERMINING ALL EXTREME POINTS OF A CONVEX POLYTOPE

M.E. DYER and L.G. PROLL
University of Leeds, Leeds, U.K.

Received 10 August 1975
Revised manuscript received 1 March 1976

An algorithm for determining all the extreme points of a convex polytope associated with a set of linear constraints, via the computation of basic feasible solutions to the constraints, is presented. The algorithm is based on the product-form revised simplex method and as such can be readily linked onto standard linear programming codes. Applications of such an algorithm are reviewed and limited computational experience given.

1. Introduction

The problem of enumerating all the extreme points of the convex polyhedron of feasible solutions to a finite set of linear inequalities in real n dimensional space has been discussed by several authors, a brief review of the literature being given in Section 1.1. below. This problem, together with the closely related one of ranking the extreme points in ascending or descending order of the value of a further linear function defined on the polyhedron, has a number of applications in mathematical programming and related areas. The interest in convex polyhedra undoubtedly arises from the ease with which linearly constrained programming problems are handled by the simplex method and its variants.

Applications of methods for determining extreme points fall into two main classes. The first class simply utilises the dual representation of a bounded polyhedron (polytope) by convex combinations of its extreme points to describe the set of solutions to the inequalities. The second class uses the fact that the set of extreme points may often contain as a subset some target set of points which, although the true objects of interest, are difficult to describe explicitly.

Among the first type of application, the most important is that of determining all optimal solutions to a linear program. This is discussed, for example, by Hadley [9]. Related problems are those of determining all optimal strategies to a matrix game (Motzkin et al. [14]) and of determining all 'near-optimal' solutions to a linear program, the need for which has been remarked upon, for example, by Koenigsberg [20] and Van de Panne [23]. Van de Panne [23] has also shown that extreme point enumeration methods can be used to solve certain problems in multiparametric linear programming. Vajda [22] has described a manpower planning problem which is also related to extreme point enumeration.

The second type of application usually involves extreme point ranking in the above sense. They were due originally to Murty [15] who applied extreme point

ranking to the fixed charge problem and later to the travelling salesman problem [16]. An alternative method has been proposed by Pollatschek and Avi-Itzhak [17]. They describe its application to zero-one mixed-integer programming.

Other applications are due to Cabot and Francis [4] in the field of nonconvex quadratic programming and to Kirby, Love and Swarup [10] to a generalised problem termed extreme point mathematical programming. Burdet [3] considers applications to general zero-one linear programming and to quadratic programming.

1.1. Review of the problem

While much research has been devoted to convex polytopes (see e.g. Grunbaum [8]), it appears that in general they have few useful properties which can be exploited in order to enumerate easily their extreme points.

The earliest published method appears to be that of Motzkin et al. [14]. Their approach is to add inequalities in a stepwise manner, noting at each step which "new" extreme points have been created and which "old" ones excluded. This has certain merits but the chief disadvantage is that, if many of the inequalities are redundant, much effort may be wasted calculating extreme points which will later be excluded.

It is well known that the problem of determining all the extreme points reduces to that of enumerating all the basic feasible solutions for the inequalities. In the nondegenerate case, in which each extreme point of the polytope is the intersection of exactly n hyperplanes in n dimensions, the correspondence between the two sets is one to one. In other cases the polytope may be "perturbed" by standard methods to restore this correspondence. This allows the use of the simplex tableau to represent the extreme point algebraically. Unfortunately it is well known [13] that it is not, in general, possible to generate a complete non-recurrent sequence of basic feasible solutions by simplex pivots.

Balinski [1] was the first to adapt the format of the simplex tableau to the problem. He devised a flexible search method based on successively relaxing the inequalities. The "book-keeping" for this algorithm is particularly simple but the method, like Motzkin's, may be inefficient in the presence of large numbers of irrelevant constraints since infeasible basic solutions are visited. Manas and Nedoma [12] also use the simplex tableau but generate only basic feasible solutions. The "book-keeping" for this algorithm requires a list of the indices of the basic variables at each basic feasible solution to be held. They attempt to trace an edge-path on the polytope terminating when each extreme point has been visited at least once. Silverman [19] has also devised a method along these lines which he calls the G-Path Method. The essential difference between the methods of Silverman and Manas and Nedoma is in the use made of this G-path construct. This is an edge path on the polytope such that each extreme point either lies on the path or is adjacent to some point on the path. It will be readily observed that, in order to enumerate all the extreme points of the polytope, it is sufficient to follow such a path. Silverman also discusses, in general terms, the

computational requirements of certain other algorithms.

Mattheis [13] adds a further "generalised-slack" variable to the inequalities. He then views the original polytope as a special facet of a polytope in $(n+1)$ dimensions. The search then takes place among the extreme points of the higher dimensional polytope which do not lie in the special facet. The points of the original polytope are then generated as "neighbours" of these points. He shows that redundant inequalities do not affect the process, but the main justification for the additional complication rests on an unproven conjecture that fewer points will be explicitly generated than in a search of the original polytope.

Burdet [3] presents an intuitively appealing approach using (basically) repeated applications of a Phase I linear programming method to construct a kind of branch and bound search for basic feasible solutions. This method will probably need more simplex pivoting operations than its competitors in the nondegenerate case but could be useful in cases of significant degeneracy. However, it seems that certain simplifications which he claims may be made (with respect to two-dimensional and simplicial faces) will have little power in the scheme proposed, and might well be omitted in a practical scheme. He gives no example.

Chernikova [6] gives an algorithm for determining the extreme edges of a convex polyhedral cone. This can be adapted in a straightforward manner to find the extreme points of a convex polyhedron. The idea is similar in many respects to that of Motzkin et al. [14] but is presented in a slightly different framework. It is discussed by Rubin [18], who also gives an application to "cardinality constrained" linear programming.

Greenberg [7] also presents a method for calculating all edges of a cone. His development is based on a theorem of Uzawa [21], strengthened by incorporating certain exclusion tests on candidate edges. The technique again has similarities with that of Motzkin et al. [14].

Other suggestions have been made by Hadley [9] and Charnes [5].

In general little attention has been paid in the papers referenced above to computer implementation; indeed several of the methods contain vague or ill-defined steps. In addition little computational experience has been cited.

The method presented here is based on the simplex method but uses the format of the product form revised simplex method [9]. Instead of using a single tableau and following a path on the edges of a polytope, the algorithm constructs a spanning tree in the graph of the polytope. Section 2 outlines the necessary concepts. Certain information is associated with each node of the tree, the nature of which is described in Section 3.1. The structure of the tree is exploited to compare directly the adjacency of each feasible basis with those previously found. This comparison occurs at most once between any two bases and allows us to avoid updating many columns of the simplex tableau in the later stages of the search. The working of the algorithm is described in Sections 3.2 and 3.3. The search strategy used to construct the tree is basically breadth-first and the resultant tree has a certain minimality property which is described in Section 3.4.

In Section 4 an example is presented and some computational experience with computer implementation of the algorithm is given in Section 5.

It is felt that the product-form orientation makes the method more easily compatible with standard linear programming procedures than other methods. The method can further be readily adapted to rank the extreme points and hence is equally useful for both types of application outlined in the previous section.

2. Notation and definitions

We consider the inequalities

$$\sum_{j=1}^{n} a_{ij}x_j \leq a_{i0} \qquad (i = 1, \ldots, m), \tag{1}$$

where

$$x_j \geq 0 \qquad (j = 1, \ldots, n)$$
$$a_{i0} \geq 0 \qquad (i = 1, \ldots, m).$$

It may be noted that any consistent set of linear inequalities can be manipulated into the form of (1). This is shown in most of the standard linear programming texts, for example, Hadley [9]. It will be assumed that (1) has a bounded solution set. This can easily be arranged by adding (if necessary) the additional inequality

$$\sum_{j=1}^{n} x_j \leq M \quad \text{for large enough } M.$$

If (1) has degenerate basic feasible solutions then it will be wise to incorporate some perturbation (implicit or explicit) to the a_{i0} in order to minimise the number of repetitions of degenerate extreme points.

We add slack variables to (1) in the usual way to give the system

$$\sum_{j=1}^{n} a_{ij}x_j + x_{n+i} = a_{i0} \qquad (i = 1, \ldots, m) \tag{2}$$

where

$$x_j \geq 0 \qquad (j = 1, \ldots, m+n).$$

The (column) m-vector of the coefficients of x_j $(j = 1, \ldots, m+n)$ in (2) will be referred to as column j and will be denoted by a_j $(j = 1, \ldots, n)$ or e_{j-n} $(j = n+1, \ldots, n+m)$. The right-hand sides of (2) form the vector a_0.

A basis B of (2) is a nonsingular $m \times m$ matrix of the columns of m basic variables. The remaining variables are nonbasic. We write $\bar{a}_j = B^{-1}a_j$ with obvious extensions to $\bar{a}_{ij}$, $\bar{a}_0$ etc. A basis is feasible if $\bar{a}_{i0} \geq 0$ $(i = 1, \ldots, m)$. The corresponding solution to (2) is a basic feasible solution. It will be assumed that the reader is familiar with these ideas and the notions of pivoting which are used to compute these updated columns, $\bar{a}_j$, in the product-form revised simplex algorithm. For further information, see Hadley [9]. We also use β to denote the set of indices of basic variables corresponding to B.

In the development of the algorithm we use the concept of a multiset [11] i.e. an entity like a set but allowed to contain repeated elements. For a given element, we define $N(A)$ to be the number of occurrences of the element in the

multiset A. The relations

$$N(A \cup^{+} B) = N(A) + N(B), \tag{3}$$

$$N(A - B) = N(A) - \min(N(A), N(B)) \tag{4}$$

then define the operators '$\cup^{+}$' and '$-$' on multisets. Clearly a set may regard as a multiset. We use $|A|$ to denote the sum of terms $N(A)$ over all elements in the multiset.

In the discussion below we borrow certain terminology from the theory of graphs. For an introduction to these concepts the reader should consult a standard text such as Berge [2]. The graph G of the polytope corresponding to (2) has a node for each feasible basis and an arc connecting two nodes if the two bases differ in exactly one column. We call the bases adjacent, or neighbours if they are connected by an arc of G. The node of G corresponding to basis B_0 containing columns $(n+1), \ldots, (n+m)$ is called the root node, r, of G. It will be noted that B_0 is an identity matrix. The corresponding basic feasible solution is $x_j = 0$ $(j = 1, \ldots, n)$ and $x_{n+1} = a_{i0}$ $(i = 1, \ldots, m)$.

The algorithm constructs a tree, T, containing r. If t is a node of T, this orientation defines the (unique) predecessor $P(t)$ of t, and t is a successor of $P(t)$. If $P(r)$ is defined to be r, then P can be considered to be a function from the set of nodes in T into itself, the tree function. The height $h(t)$ of t is the number of arcs in the (unique) path from r to t in T. In terms of P, we have

$$h(t) = \min\{k: P^k(t) = r\}.$$

The algorithm terminates when T is a spanning tree of G.

3. The algorithm

3.1. *Organization*

At a typical stage of the algorithm, the gth, a list of feasible bases $B_0, B_1, \ldots, B_u$ to (2) will have been constructed. B_0 corresponds to the root node r and hence is the $m \times m$ identity matrix. The nodes of T will be identified with the corresponding index of the associated basis in this list. The B_l are not retained explicitly, however, but sufficient information is stored to allow us to premultiply by their inverses as necessary. This information is, in principle, the eta-vectors of the product-form revised simplex method. We also list information on which of these bases are known to be adjacent. The data stored for each node l is as follows:

(i) p_l: The predecessor $P(l)$ of l in T. Such numbers determine the structure of T.

(ii) q_l: The index of the pivotal row in the pivotal operation transforming $B_{p_l}^{-1}$ to B_l^{-1}.

(iii) r_l: The index of the column entering the basis in the above operation.

(iv) s_l: The index of the column leaving the basis in the above operation.

(v) γ_l: A set of indices of columns such that, if $j \in \gamma_l$, the list is known to

contain a basis B containing column j and differing from B_l in at most one column. In particular, γ_l contains the indices of the basis variables in B_l.

(vi) a_l: The $(m \times 1)$ pivot column in the operation transforming $B_{p_l}^{-1}$ to B_l^{-1}. In fact $a_l = B_{p_l}^{-1} a_{r_l}$ and is virtually the eta-vector of the product-form method. This notation may appear initially confusing but no conflict arises since, for $l \leq n$, a_l is identical to column l of (2). This follows from the facts that B_0 is an identity matrix and, for $l \leq n$, $r_l = l$.

At the start of the algorithm, we will have stored $a_0, a_1, \ldots, a_n$ and we set

$$p_0 \leftarrow 0, \qquad g \leftarrow 0,$$
$$\gamma_0 \leftarrow \{n+1, n+2, \ldots, n+m\}.$$

3.2. *Outline discussion*

In this section the main features of the algorithm are outlined, a detailed description being given in Section 3.3.

At the start of the gth stage, the nodes of T may be thought of as being divided into two sets. The set of nodes

$$C = \{0, 1, \ldots, (g-1)\}$$

is called the set of considered nodes with the property that the current list contains all feasible bases adjacent to any basis associated with C. The remaining set of nodes

$$D = \{g, (g+1), \ldots, u\}$$

is called the set of unconsidered nodes.

At the gth stage, the first unconsidered node g is selected and becomes considered. From the information associated with the sequence of nodes

$$g, P(g), P^2(g), \ldots, 0$$

we can construct the basic set β_g of B_g and also deduce the eta-vector representation of B_g^{-1}. We determine $\bar{a}_0$, the updated right hand side, using the forward transformations of the product-form method. At this point the extreme point corresponding to B_g has been determined.

Currently γ_g contains, in addition to the indices of the basic columns of B_g, the indices of nonbasic columns which would generate a node in C if exchanged for a column of B_g. Clearly if γ_g contains the indices of all the columns of (2) then the list contains all feasible bases adjacent to B_g. If this is the case the gth stage is complete. Otherwise a comparison for adjacency of node g with the remaining nodes of D is commenced. For this purpose we use only the sets of numbers p_l, r_l, s_l. Since any basic feasible solution to (2) can be reached from any other basic feasible solution by a finite number of simplex pivots and since every basic feasible solution associated with T can be reached from node 0 by a sequence of feasible simplex pivots, it is clear that the path in T from g to any node $d \in D$ can be constructed using the p_l. The comparison test proceeds by tracing back the paths in T from g to 0 and d to 0 until they meet at a common node, u (say).

Then

$$P^j(g) = P^k(d) = u \text{ for some } j, k$$

and the path from g to d is

$$g, P(g), ---, P^{j-1}(g), u, P^{k-1}(d), ---, P(d), d.$$

Let

$$X_g = \{s_g, s_{P(g)}, ---, s_P^{j-1}(g)\},$$
$$Y_g = \{r_g, r_{P(g)}, ---, r_P^{j-1}(g)\},$$

i.e. X_g contains the indices of the variables leaving the successive bases on the path from u to g, Y_g contains similar information on the entering variables. X_d, Y_d can clearly be defined in a similar manner. Note that, in general, X_g, Y_g, X_d, Y_d are multi-sets and

$$\beta_g = (\beta_u \cup^+ Y_g) - X_g, \tag{5}$$
$$\beta_d = (\beta_u \cup^+ Y_d) - X_d. \tag{6}$$

Consider (5), by (3) and (4)

$$N(\beta_g) = N(\beta_u) + N(Y_g) - \min[N(\beta_u) + N(Y_g), N(X_g)].$$

Clearly, in order to leave a particular basis, a variable must have either been present in the initial basis β_u or have entered a previously encountered basis, i.e.,

$$N(X_g) \leq N(\beta_u) + N(Y_g)$$

hence

$$N(\beta_g) = N(\beta_u) + N(Y_g) - N(X_g).$$

Similarly,

$$N(\beta_d) = N(\beta_u) + N(Y_d) - N(X_d).$$

Hence

$$N(\beta_g - \beta_d) = [N(\beta_u) + N(Y_g) - N(X_g)] - \min[N(\beta_u) + N(Y_g) - N(X_g), N(\beta_u) + N(Y_d) - N(X_d)]. \tag{7}$$

Subtracting $N(\beta_u) - N(X_g) - N(X_d)$ from each term on the right-hand side of (7), we obtain

$$N(\beta_g - \beta_d) = N(Y_g) + N(X_d) - \min[N(Y_g) + N(X_d), N(Y_d) + N(X_g)].$$

Hence

$$\beta_g - \beta_d = (Y_g \cup^+ X_d) - (Y_d \cup^+ X_g) = \rho. \tag{8}$$

By symmetry,

$$\beta_d - \beta_g = (Y_d \cup^+ X_g) - (Y_g \cup^+ X_d) = \sigma. \tag{9}$$

ρ may be interpreted as the set of indices of variables which enter the basis but do not leave during the basis exchanges on the path from g to d, σ having a similar interpretation for the path from d to g. One method of testing the adjacency of nodes g and d, therefore, is to construct the sets ρ and σ and to test whether $|\rho| = 1 = |\sigma|$. In principle it is necessary to construct only one of these sets in order to test adjacency but it is little more work to construct both and, in the event of g and d being adjacent, ρ and σ define the (single) basis exchange which is necessary to get from g to d (and d to g).

The comparison test detailed in Section 3.3 constructs ρ and σ from (8) and (9) respectively in an efficient manner by inspecting arcs from the path from g to u and from d to u alternately. This allows

(a) u to be recognised as quickly as possible,

(b) the sets X_g, Y_g, X_d, Y_d to be built up element by element and the cancelation of common elements implied by (8) and (9) to be performed as quickly as possible,

thereby minimizing the space required to assemble ρ and σ. This strategy is possible because

$$d \in D \Rightarrow P(d) \in C$$

hence

$$P(d) \leq g < d.$$

It follows from Proposition 1 (Section 3.4) that

$$k - 1 \leq j \leq k,$$

i.e., g and d differ by at most one level in the tree.

Conducting the comparison in this form involves the storage of less information than retaining the entire set of indices of basic variables for each node as do Manas and Nedoma [12]. Furthermore no more operations will generally be involved than would be needed to compare two such sets; fewer operations will be needed when g and d are "close" in T.

If g and d are found to be adjacent the sets γ_g and γ_d are adjusted by adding to each the appropriate index. The gth stage will be abandoned should γ_g include all indices of columns in (2). If, however, the examination of D is completed without this condition being fulfilled, new bases are added to the list. For each $j \notin \gamma_g$, we form

$$\bar{a}_j = \begin{cases} B_g^{-1} a_j & (j \leq n), \\ B_g^{-1} e_{j-n} & (j > n). \end{cases}$$

and determine the pivotal row, t, and the outgoing column, k, assuming that column j enters the basis. In each case, a new node l is then added to D with the following information:

$$\begin{aligned} &p_l = g, \quad g_l = t, \quad r_l = j, \quad s_l = k, \\ &\gamma_l = \beta_g \cup \{j\}, \qquad a_l = \bar{a}_j. \end{aligned} \tag{10}$$

The gth stage is then complete and the $(g+1)$th commences. The algorithm terminates when the set of unconsidered nodes, D, is empty, all feasible bases having been enumerated.

3.3. *Detailed discussion*

In this section the main computational steps of a typical stage of the algorithm are described in more precise fashion. The functional notation $P(l)$ for the number p_l will be used throughout.

First $h(g)$ is determined from the condition $h(g) = \min\{k: P^k(g) = 0\}$.

The path in T from 0 to g passes through nodes $P^k(g)$ for $k = h(g), h(g)-1, \ldots, 0$. The set of basic variables for g is then obtained by a repeated substitution:

(A) *Basic set formation*

(0) Initialisation. $b_i \leftarrow n+i$ $(i = 1, \ldots, m)$, $k \leftarrow h(g)$.

(1) If $k = 0$ terminate.

(2) Substitution. $k \leftarrow k-1$, $l \leftarrow P^k(g)$, $b_{q_l} \leftarrow r_l$. Go to (1).

The vector b now contains the indices of the basic variables in "row-order".

The right-hand side vector $\bar{a}_0$ is now calculated. The following routine is used with $j = 0$, but is described for a general column j of (2) for later reference.

(B) *Column update*

(0) Initialisation. If $j \leqslant n$, $z \leftarrow a_j$ otherwise $z \leftarrow e_{j-n}$, $k \leftarrow h(g)$.

(1) If $k = 0$, set $\bar{a}_j \leftarrow z$ and terminate.

(2) Forward transformation. $k \leftarrow k-1$, $l \leftarrow P^k(g)$, $z_{q_l} \leftarrow z_{q_l}/a_{lq_l}$. For $i \neq q_l$, $z_i \leftarrow z_i - z_{q_l} a_{li}$. Go to (1).

The extreme point is now determined by $x_{b_i} = \bar{a}_{oi}$ $(i = 1, \ldots, m)$ and $x_j = 0$ $(j \notin \beta)$. The gth stage is complete if $\gamma_g = \{1, \ldots, m+n\}$, otherwise the comparison for adjacency with D is begun. It appears best to use an efficient implementation of the following routine for this purpose. The comparison of g with a given $d \in D$ is described.

(C) *Comparison*

(0) Initialisation. $\rho, \sigma \leftarrow \emptyset$. $k \leftarrow g$, $l \leftarrow d$.

(1) If $k > l$: $r \leftarrow r_k$, $s \leftarrow s_k$, $k \leftarrow P(k)$. Go to (2).
If $k < l$: $r \leftarrow s_l$, $s \leftarrow r_l$, $l \leftarrow P(l)$. Go to (2).
If $k = l$, go to (3).

(2) If $r \in \sigma$, $\sigma \leftarrow \sigma - \{r\}$, otherwise $\rho \leftarrow \rho \cup^+ \{r\}$.
If $s \in \rho$, $\rho \leftarrow \rho - \{s\}$, otherwise $\sigma \leftarrow \sigma \cup^+ \{s\}$. Go to (1).

(3) If $|\rho| = 1$, g and d are adjacent, otherwise not.
If g, d are found to be adjacent, $\gamma_g \leftarrow \gamma_g \cup \sigma$, $\gamma_d \leftarrow \gamma_d \cup \rho$.

If the gth stage is still not complete when D is exhausted, the generation of new feasible bases begins. For each variable index $j \notin \gamma_g$, update column j using the routine of (B) above. Determine the pivotal row t by the usual ratio method,

i.e., t is the index at which $\bar{a}_{oi}/\bar{a}_{ji}$ attains its minimum for $\bar{a}_{ji} > 0$ $(i = 1, \ldots, m)$. The variable which will leave the basis is $k = b_t$. The information for each new node is stored as indicated in (10). We set $g \leftarrow g + 1$ and (provided $g \leqslant u$) enter the next stage of the algorithm.

3.4. *Properties of the algorithm*

It will be clear from the above description that the algorithm will generate all feasible bases and terminate. It is also easy to see that each pair of extreme points will be compared at most once for adjacency. We can further establish the following:

Proposition 1. *If* $g < l$, *then* $h(g) \leqslant h(l)$.

Otherwise, suppose g is the lowest numbered node such that there is an $l > g$ with $h(g) > h(l)$. The structure of T clearly implies that $P(g) < g$, $P(l) < l$ and $P(g) \leqslant P(l)$. But $h(P(g)) = h(g) - 1$, $h(P(l)) = h(l) - 1$ and hence $h(P(g)) > h(P(l))$. This contradicts the assumption that g was the least such number.

Thus the algorithm numbers the nodes in non-decreasing order of height in T. This is sometimes called a bottom-up labelling of T.

Proposition 2. *If* $h^1(g)$ *is the number of arcs in any path from node* 0 *to node* g *in* G, *then* $h(g) \leqslant h^1(g)$.

Otherwise, suppose there is a path in G from node 0 containing a node which violates the assumption. Let g be the first such node encountered. Clearly $g > 0$ since $h^1(0) = h(0) = 0$. Suppose l is the node immediately before g on the path, then $h^1(l) = h^1(g) - 1$. But from the priority of g, $h(l) \leqslant h^1(l) = h^1(g) - 1 < h(g) - 1$. But $h(P(g)) = h(g) - 1$, so $h(l) < h(P(g))$. But then, from Proposition 1, we have $l < P(g)$. But since l is adjacent to g, the way T is constructed implies $l \geqslant P(g)$, giving a contradiction.

This implies that each feasible basis is generated in the smallest possible number of basis exchanges given that all intervening bases must be feasible. In graphical terms T is a tree of "shortest routes" from node 0 if all edges of G have unit weight.

The minimality property expressed by Proposition 2 is perhaps useful in a numerical sense since computational errors will tend to grow with the number of basis exchanges.

It will be noted that both properties derive from the method used to construct T.

4. Example

The following problem first appeared in Balinski [1] and was used by Mattheis [13] as an illustrative example:

Determine all the extreme points of the polytope represented by the equations

$$\begin{aligned}
x_4 &\quad + 3x_1 + 2x_2 - x_3 = 6,\\
x_5 &\quad + 3x_1 + 2x_2 + 4x_3 = 16,\\
x_6 &\quad + 3x_1 \quad\quad - 4x_3 = 3,\\
x_7 &\quad + \tfrac{9}{4}x_1 + 4x_2 + 3x_3 = 17,\\
x_8 &+ x_1 + 2x_2 + x_3 = 10.
\end{aligned}$$

(all variables non-negative).

Since the algorithm is designed for computer implementation, detailed computations are not given below. Rather, an outline of the progress of the method on this problem will be presented.

Initially we have

$$a_0 \leftarrow (6, 16, 3, 17, 10), \qquad a_1 \leftarrow (3, 3, 3, \tfrac{9}{4}, 1),$$
$$a_2 \leftarrow (2, 2, 0, 4, 2), \qquad a_3 \leftarrow (-1, 4, -4, 3, 1),$$
$$p_0 \leftarrow 0, \qquad \gamma_0 \leftarrow \{4, 5, 6, 7, 8\}, \qquad g \leftarrow 0, \quad u \leftarrow 0.$$

Stage 0. $h(0) = 0$; $b \leftarrow (4, 5, 6, 7, 8)$, $\bar{a}_0 \leftarrow (6, 16, 3, 17, 10)$. Extreme point 0 is determined. Create new entries for $j = 1, 2, 3$.

$$p_1 \leftarrow 0, \quad q_1 \leftarrow 3, \quad r_1 \leftarrow 1, \quad s_1 \leftarrow 6, \quad \gamma_1 \leftarrow \{1, 4, 5, 6, 7, 8\}.$$
$$p_2 \leftarrow 0, \quad q_2 \leftarrow 1, \quad r_2 \leftarrow 2, \quad s_2 \leftarrow 4, \quad \gamma_2 \leftarrow \{2, 4, 5, 6, 7, 8\}.$$
$$p_3 \leftarrow 0, \quad q_3 \leftarrow 2, \quad r_3 \leftarrow 3, \quad s_3 \leftarrow 5, \quad \gamma_3 \leftarrow \{3, 4, 5, 6, 7, 8\}.$$
$$g \leftarrow 1, \quad u \leftarrow 3.$$

Stage 1. $h(1) = 1$; $b \leftarrow (4, 5, 1, 7, 8)$, $\bar{a}_0 \leftarrow (3, 13, 1, 14\tfrac{3}{4}, 9)$. Extreme point 1 is determined. The comparison routine gives 1 nonadjacent with 2 or 3. Thus create new nodes for $j = 2, 3$.

$$\bar{a}_2 \leftarrow (2, 2, 0, 4, 2), \qquad \bar{a}_3 \leftarrow (3, 8, -\tfrac{4}{3}, 6, \tfrac{7}{3}),$$
$$p_4 \leftarrow 1, \quad q_4 \leftarrow 1, \quad r_4 \leftarrow 2, \quad s_4 \leftarrow 4, \quad \gamma_4 \leftarrow \{1, 2, 4, 5, 7, 8\}, \quad a_4 \leftarrow \bar{a}_2.$$
$$p_5 \leftarrow 1, \quad q_5 \leftarrow 1, \quad r_5 \leftarrow 3, \quad s_5 \leftarrow 4, \quad \gamma_5 \leftarrow \{1, 3, 4, 5, 7, 8\}, \quad a_5 \leftarrow \bar{a}_3.$$
$$g \leftarrow 2, \quad u \leftarrow 5.$$

Stage 2. $h(2) = 1$; $b \leftarrow (2, 5, 6, 7, 8)$, $\bar{a}_0 \leftarrow (3, 10, 3, 5, 4)$. The comparison routine gives 2 nonadjacent to 3, 5 but adjacent to 4. Then

$$\gamma_2 \leftarrow \{1, 2, 4, 5, 6, 7, 8\}, \qquad \gamma_4 \leftarrow \{1, 2, 4, 5, 6, 7, 8\}.$$

Create new node for $j = 3$.

$$\bar{a}_3 \leftarrow (-\tfrac{1}{2}, 5, -4, 5, 2),$$
$$p_6 \leftarrow 2, \quad q_6 \leftarrow 4, \quad r_6 \leftarrow 3, \quad s_6 \leftarrow 7, \quad \gamma_6 \leftarrow \{2, 3, 5, 6, 7, 8\}, \quad a_6 \leftarrow \bar{a}_3.$$
$$g \leftarrow 3, \quad u \leftarrow 6.$$

Stage 3. $h(3) = 1$; $b \leftarrow (4, 3, 6, 7, 8)$, $\bar{a}_0 \leftarrow (10, 4, 19, 5, 6)$. Comparison gives 3 nonadjacent with 4, 5, 6. Thus create new nodes for $j = 1, 2$.

$\bar{a}_1 \leftarrow (\frac{15}{8}, \frac{3}{4}, 6, 0, \frac{1}{4})$, $\bar{a}_2 \leftarrow (\frac{5}{2}, \frac{1}{2}, 2, \frac{5}{2}, \frac{3}{2})$,
$p_7 \leftarrow 3$, $q_7 \leftarrow 1$, $r_7 \leftarrow 1$, $s_7 \leftarrow 4$, $\gamma_7 \leftarrow \{1, 3, 4, 6, 7, 8\}$, $a_7 \leftarrow \bar{a}_1$.
$p_8 \leftarrow 3$, $q_8 \leftarrow 4$, $r_8 \leftarrow 2$, $s_8 \leftarrow 7$, $\gamma_8 \leftarrow \{2, 3, 4, 6, 7, 8\}$, $a_8 \leftarrow \bar{a}_2$.
$g \leftarrow 4$, $u \leftarrow 8$.

Stage 4. $h(4) = 2$; $b \leftarrow (2, 5, 1, 7, 8)$, $\bar{a}_0 \leftarrow (\frac{3}{2}, 10, 1, 8\frac{3}{4}, 6)$. Comparison gives 4 adjacent to 5. Then

$$\gamma_4 \leftarrow \{1, 2, 3, 4, 5, 6, 7, 8\}, \qquad \gamma_5 \leftarrow \{1, 2, 3, 4, 5, 7, 8\}.$$

Abandon stage 4. $g \leftarrow 5$.

Stage 5. $h(5) = 2$; $b \leftarrow (3, 5, 1, 7, 8)$, $\bar{a}_0 \leftarrow (1, 5, \frac{7}{3}, 8\frac{3}{4}, \frac{20}{3})$. Comparison gives 5 nonadjacent to 6, adjacent to 7. Hence

$$\gamma_5 \leftarrow \{1, 2, 3, 4, 5, 6, 7, 8\}, \qquad \gamma_7 \leftarrow \{1, 3, 4, 5, 6, 7, 8\}.$$

Abandon stage 5. $g \leftarrow 6$.

Stage 6. $h(6) = 2$; $b \leftarrow (2, 5, 6, 3, 8)$, $\bar{a}_0 \leftarrow (\frac{7}{2}, 5, 7, 1, 2)$. Comparison gives 6 nonadjacent to 7, adjacent to 8.

$$\gamma_6 = \{2, 3, 4, 5, 6, 7, 8\}, \qquad \gamma_8 = \{2, 3, 4, 5, 6, 7, 8\}.$$

Create new node for $j = 1$.

$\bar{a}_1 \leftarrow (\frac{9}{8}, \frac{15}{4}, 0, -\frac{3}{4}, -\frac{1}{2})$,
$p_9 \leftarrow 6$, $q_9 \leftarrow 2$, $r_9 \leftarrow 1$, $s_9 \leftarrow 5$, $\gamma_9 \leftarrow \{1, 2, 3, 5, 6, 8\}$, $a_9 \leftarrow \bar{a}_1$.
$g \leftarrow 7$, $u \leftarrow 9$.

Stage 7. $h(7) = 2$; $b \leftarrow (1, 3, 6, 7, 8)$, $\bar{a}_0 \leftarrow (\frac{8}{3}, 2, 3, 5, \frac{16}{3})$. Comparison gives 7 nonadjacent to 8, adjacent to 9, so

$$\gamma_7 \leftarrow \{1, 2, 3, 4, 5, 6, 7, 8\}, \qquad \gamma_9 \leftarrow \{1, 2, 3, 5, 6, 7, 8\}.$$

End stage 7, $g \leftarrow 8$.

Stage 8. $h(8) = 2$; $b \leftarrow (4, 3, 6, 2, 8)$, $\bar{a}_0 \leftarrow (5, 3, 15, 2, 3)$. Comparison gives 8 adjacent to 9 and

$$\gamma_8 \leftarrow \{1, 2, 3, 4, 5, 6, 7, 8\}, \qquad \gamma_9 \leftarrow \{1, 2, 3, 4, 5, 6, 7, 8\}.$$

End stage 8, $g \leftarrow 9$.

Stage 9. $h(9) = 3$; $b \leftarrow (2, 1, 6, 3, 8)$, $\bar{a}_0 \leftarrow (2, \frac{4}{3}, 7, 2, \frac{8}{3})$.

$$\gamma_9 = \{1, 2, 3, 4, 5, 6, 7, 8\}.$$

End stage 9.

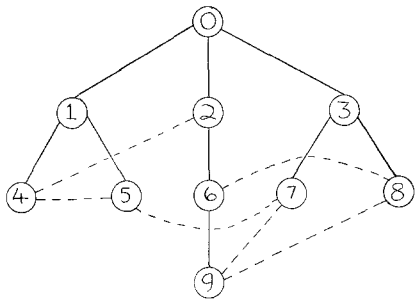

Fig. 1. Graph of polytope of example.

All nodes have now been considered, thus terminate. The ten feasible bases and corresponding adjacencies have been computed. The graph G for the polytope of this example is presented in Figure 1. The tree T constructed by the algorithm is shown in solid line, other arcs of G in broken line.

5. Computer implementation

As with any algorithm, efficient implementation is essential for practical use. In this respect the following points may be noted. Firstly the sets γ_l can be conveniently stored in a computer in "packed" binary form, since the only operations performed on any γ_l are the addition of single elements and a once-for-all "decoding" before the generation of new bases. A simple count of the number of indices currently in γ_l will then suffice for the test of termination in the comparison phase. Secondly, careful attention should be paid to the coding of the comparison routine, since this will be used often during the execution of the algorithm. There are several ways in which the outline routine (C) of 3.3 can be efficiently programmed. Thirdly, for unbounded polyhedra there is no real necessity to introduce the additional inequality of Section 2. Unbounded edges are indicated by a completely negative updated column in the basis-generation phase. These columns may be simply ignored or, if the unbounded edges are of interest, output at this point. No new entry is created in the list. It may be further remarked that for maximum efficiency advantage could be taken of sparsity in the coefficients of the inequality set. Should storage be a limitation, the vectors a_l $(l > n)$ could be located in backing store and only those few brought into central storage which are required at each stage.

This algorithm has been programmed in FORTRAN for an ICL 1905E computer. The program has been tested on a number of manufactured problems. Table 1 gives some information on the running of the program on six non-degenerate problems. In this table m and n are as in (1). The number of extreme points and unbounded edges of the polyhedron for each of the test problems is indicated together with the execution time on the 1905E. These computing times are only accurate to about one second either way and are given only as an

Table 1

Problems	m	n	Number of extreme points	Number of unbounded edges	Execution time on ICL 1905E (seconds)
1	6	6	54	0	8
2	6	7	78	0	12
3	6	8	111	0	17
4	6	9	134	0	25
5	6	10	247	0	56
6	20	3	12	0	5
7	25	4	25	0	10
8	9	8	170	102	45
9	6	10	130	214	42

indication of the performance of the method as programmed for this particular machine. In Table 1 Problems 1 to 5 were obtained by taking the first 6, 7, 8, 9, 10 columns respectively of the following set of inequalities:

$$
\begin{aligned}
-3x_1 - 3x_2 + 3x_3 + 4x_4 + 2x_5 + 3x_6 + 2x_7 + 5x_8 + 1x_9 + 2x_{10} &\leq 120,\\
3x_1 + 3x_2 - 3x_3 - 2x_4 + 2x_5 + 3x_6 + 4x_7 + 1x_8 + 1x_9 - 5x_{10} &\leq 121,\\
3x_1 + 3x_2 + 3x_3 + 4x_4 - 4x_5 - 3x_6 + 2x_7 + 0x_8 + 1x_9 + 4x_{10} &\leq 122,\\
-3x_1 + 3x_2 - 3x_3 + 4x_4 + 2x_5 + 3x_6 + 4x_7 + 5x_8 + 1x_9 - 2x_{10} &\leq 123,\\
3x_1 - 3x_2 + 3x_3 - 2x_4 + 2x_5 + 3x_6 - 3x_7 - 2x_8 + 1x_9 + 0x_{10} &\leq 124,\\
3x_1 + 3x_2 + 3x_3 - 2x_4 + 2x_5 - 3x_6 - 3x_7 - 3x_8 + 1x_9 + 1x_{10} &\leq 125.
\end{aligned}
$$

(All variables non-negative.)

Problems 6, 7 each contain a large proportion of redundant inequalities (15 and 16 respectively) and Problems 8, 9 are unbounded.

Experience with degenerate problems confirms the recommendations of Section 2 as regards perturbing the problem data.

There is little in the literature with which to compare these results. The papers of Motzkin [14] and Burdet [3] cite no computational experience. Silverman [19] gives only a count of the number of simplex pivots required by his method to solve eight degenerate three dimensional problems. Mattheis [13] gives some times for computing the vertices of n-dimensional simplices and hypercubes for $n = 1, \ldots, 30$. These are, however, rather special polytopes for the above (or any) algorithm. The simplex is simply the case $m = 1$ in (1), and for the hypercube problem a_l is a unit vector for each l so that all the pivoting is trivial. Also storage is a problem for n beyond about 10. Indeed it is difficult to see how Mattheis, who also maintains a list structure, could store so many entries.

Balinski [1] and Manas and Nedoma [12] each give a computing time for one problem. It is impossible to draw any comparisons, however, since neither paper presents the exact problem solved. In this context it is hoped that the problem

given above may act as a benchmark to compare the method presented here with possible alternatives.

6. Conclusions

A description has been given of an algorithm, compatible with standard linear programming methods, for the determination of all extreme points of a polytope associated with a set of linear inequalities. Some modifications have been discussed and limited computational experience with the algorithm has been presented. Further work in applying the method is being undertaken.

A main feature of the method is to conduct comparisons for adjacency directly rather than to perform unnecessary pivoting. The justification for this is that these comparisons use only fixed point operations on a computer, which are less costly in time than the floating point operations involved in pivoting. A useful side-benefit of this is that complete information on the graphical structure of the polytope is available with no additional effort. Information on the higher-dimensional face-structure (e.g. [3]) would require extra computation, however.

The search strategy we have used is breadth-first. An equivalent algorithm using a depth-first strategy can be devised. The advantage of using the product-form representation disappears, however, although the comparison routine can be somewhat streamlined. The method of Manas and Nedoma [12], although not expressed in these terms, is similar to a depth-first search.

A mention was made (in 1.1) that the algorithm can be easily adapted for the extreme-point ranking problem defined in 1.0. For this application the root node would be the minimum (say) of the linear function. The function value for each basic feasible solution generated would be stored. The only change required in the algorithm is that the nodes are not now considered in list order but in increasing order of the function. In other words the next node to be selected at each stage is not the first unconsidered node but the unconsidered node having least value of the linear function. Some work in applying the method (in a strengthened form) to the solution of zero–one integer programming problems is currently under way.

References

[1] M.L. Balinski, "An algorithm for finding all vertices of convex polyhedral sets", *Journal of the Society Industrial and Applied Mathematics* (1961) 72–88.

[2] C. Berge, *The theory of graphs and its applications* (Methuen, London, 1962).

[3] C.A. Burdet, "Generating all the faces of a polyhedron", *SIAM Journal on Applied Mathematics*, 26 (1974) 479–489.

[4] V.A. Cabot and R.L. Francis, "Solving certain nonconvex quadratic minimization problems by ranking the extreme points", *Operations Research*, 18 (1970) 82–86.

[5] A. Charnes, W.W. Cooper and A. Henderson, *An introduction to linear programming* (Wiley; New York, 1953).

[6] N.V. Chernikova, "An algorithm for finding a general formula for non-negative solutions of a system of linear inequalities", *U.S.S.R. Computational Mathematics and Mathematical Physics* 5 (1965) 228–233.
[7] H. Greenberg, "An algorithm for determining redundant inequalities and all solutions to polyhedra", *Numerische Mathematik*, 24 (1975) 19–26.
[8] B. Grunbaum, *Convex polytopes* (Wiley, New York, 1967).
[9] G. Hadley, *Linear programming* (Addison-Wesley, Reading, MA, 1962).
[10] M.J.L. Kirby, H.L. Love and Kanti Swarup, "Extreme point mathematical programming", *Management Science*, 18 (1972) 540–549.
[11] D.E. Knuth, *The art of computer programming*, Vol. 2: *Seminumerical algorithms* (Addison-Wesley, Reading, MA, 1968).
[12] M. Manas and J. Nedoma, "Finding all vertices of a convex polyhedron", *Numerische Mathematik*, 12 (1968) 226–229.
[13] T.H. Mattheis, "An algorithm for determining irrelevant constraints and all vertices in systems of linear inequalities", *Operations Research* 21 (1973) 247–260.
[14] T.S. Motzkin, H. Raiffa, G.L. Thompson and R.M. Thrall, "The double description method", in: H.W. Kuhn and A.W. Tucker, eds., *Contributions to the theory of games*, Vol. 2, (Princeton University Press, Princeton, RI, 1973).
[15] K.G. Murty, "Solving the fixed charge problem by ranking the extreme points", *Operations Research* 16 (1968) 268–279.
[16] K.G. Murty, "An algorithm for ranking all the assignments in increasing order of costs", *Operations Research* 16 (1969) 682–687.
[17] M.A. Pollatschek and B. Avi-Itshak, "Sorting feasible basic solutions of a linear program", presented at the 3rd Annual Israel Conference on Operations Research (1969).
[18] D.S. Rubin, "Vertex generation and cardinality constrained linear programs", *Operations Research* 23 (1975) 555–564.
[19] G.J. Silverman, "Computational considerations in extreme point enumeration". IBM Los Angeles Scientific Center, Report G320-2649 (1971).
[20] W.J. Sullivan and E. Koenigsberg, "Mixed integer programming applied to ship allocation", in: E.M.L. Beale, ed., *Applications of mathematical programming techniques*, (English Universities' Press, London, 1970).
[21] H. Uzawa, "A theorem on convex polyhedral cones", in: Arrow, Hurwicz and Uzawa, eds., *Studies in linear and nonlinear programming*, (Standford University Press, Stanford, CA, 1958).
[22] S. Vajda, "Manpower planning and mathematical programming", paper presented at the Operational Research Society annual conference, Brighton (1974).
[23] C. Van De Panne, "A node method for multiparametric linear programming", *Management Science* 21 (1975) 1014–1020.