Degenerate Convex Hulls in High Dimensions Without Extra Storage

Günter Rote
Technische Universität Graz, Institut für Mathematik
Steyrergasse 30, A-8010 Graz, Austria
electronic mail: `rote@opt.math.tu-graz.ac.at`

1 Introduction

We present an algorithm for enumerating the faces of the convex hull of a given set P of n points in d dimensions. The main features of the algorithm are that it uses little extra storage and that it addresses degeneracy explicitly.

It is based on an idea that was recently introduced by Avis and Fukuda [1991] for their convex hull algorithm: The idea is to take a pivoting rule from linear programming and to "invert" the path that it takes to the optimal solution in all possible ways, thereby visiting all feasible bases in a depth-first search manner.

Theoretical considerations and computational tests have established that the method takes a long time for degenerate point sets. The reason is that, in the case of degenerate polytopes, the number of feasible bases may exceed the number of facets by far.

Therefore we propose a variation of the method that takes degeneracies into account explicitly: Instead of visiting all feasible bases, the algorithm visits all *facets*. The manner of visiting facets is analogous to the convex hull algorithm of Chand and Kapur [1970] as it is described and analyzed in Swart [1985].

Section 2 gives an overview of the method in a quite general way. Section 3 describes the method from a different point of view: as a seach of the face lattice. Section 4 describes specific pivoting rules and section 5 gives some implementation details. Section 6 gives a rough complexity analysis and proposes a hybrid algorithm that saves time in nondegenerate cases, thus becoming even more competitive with Avis and Fukuda's algorithm.

Motivation of the problem. Besides being one of the most prominent problems in computational geometry, the convex hull has a number of important applications.

Optimization. The minimum of a concave function over a polytope is achieved at a vertex. Many methods for global optimization are based on enumerating the vertices of a set of solutions that is described by linear inequalities. This enumeration problem is dual to the convex hull problem, and many algorithms have been proposed for it in the area of Operations Research, see Matheiss and Rubin [1980] or Chen, Hansen, and Jaumard [1991].

Mathematical research. In combinatorial optimization one is interested in describing the facets of polytopes whose vertices correspond to combinatorial objects, because this opens the possibility to optimize over these objects by linear programming and cutting plane algorithms ("polyhedral combinatorics", see for example Pulleyblank [1989]). Examples are the traveling salesman polytope or the cut polytope. In addition to theoretical considerations that lead to classes of facets, it is also useful to compute the facets explicitly for small problems (see e. g. Christof, Jünger and Reinelt [1991]).

Certain highly regular graphs can be described as the skeletons of regular polyhedra, see Brouwer, Cohen and Neumaier [1989].

The polytopes that arise in these areas are *highly degenerate.*

Why is it important to use little extra storage? Most algorithms for computing convex hulls (see e. g. Swart [1985] or Seidel [1991]) require to maintain a description of the whole convex hull, at least of all its facets. Since the number of facets has an explosive growth in higher dimensions, these algorithms soon exceed the capacity limits even of today's most powerful computers. In this respect, the storage limit is much more severe than the time limit, because it may not just make a problem take longer, but it may actually make it infeasible (see Christof, Jünger, and Reinelt [1991] for an account of the efforts that were undertaken to preprocess a problem by hand and split it into subproblems in order to make it tractable by computer.) In

---

[0] An earlier version of this extended abstract appeared in the Proceedings of the Eighth Annual Symposium on Computational Geometry, Berlin, June 10–12, 1992, published by the Association for Computing Machinery, 1992; pp. 26–32.

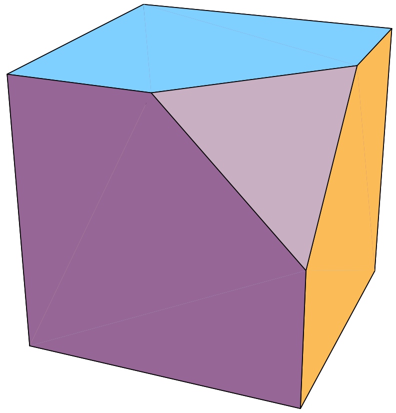

Figure 1: A cube with one trucated corner.

addition, through virtual storage and paging, the storage requirement indirectly affects the running time of the algorithm.

Therefore the method of Avis and Fukuda can be seen as a major breakthrough, since it makes larger problems amenable to a computer solution for the first time. The present algorithm is a further development of that method.

Comparison with the algorithm of Avis and Fukuda. The algorithm of Avis and Fukuda differs from the present algorithm in one main point: it visits feasible bases, and not facets. Geometrically, a *feasible basis* of a point set P can be defined as a $(d-1)$-dimensional simplex with vertices from P which is contained in a facet of the convex hull of P. A facet which is not a simplex contains many fasible bases, and it is possible to go from a feasible basis to an adjacent fasible basis and remain on the same facet. (This is called a degenerate pivot.) By an additional test the lexicographically smallest basis of each facet can be identified, and thus it is possible to output each facet only once.

A point on the boundary of $conv(P)$ which is not a vertex increases the number of feasible bases but it is ignored by the facet enumeration algorithm of the present paper (except for the additional overhead in the factor $O(n)$ for the pivot steps). (However, points which are not extreme can easily be removed beforehand by linear programming.)

Some illustrative calculations. Consider the d-dimensional cube. It has 2^d vertices and $2d$ facets. The total number of faces is 3^d. For example, the 4-cube has 16 vertices and only 8 facets, but it has 464 feasible bases, and the algorithm of Avis and Fukuda visits each of them. Perturbation of the vertices leads to a triangulation of the facets, and this helps a little: By perturbing the vertices of the 4-cube I generated a polytope with 47 simplicial facets. (Of course, this number depends on the perturbation.)

For the 5-cube, these numbers are even more striking: 32 vertices, 10 facets, and 30080 feasible bases. These figures have been obtained with the help of an implementation of different versions of Avis and Fukuda's algorithm.

Notation and definitions. We will assume without loss of generality that the convex hull $conv(P)$ of P is full-dimensional, with dimension d, and that we put the coordinate origin into the center of gravity of the points. The *facets* of a d-polytope are its $(d-1)$-dimensional faces, and its $(d-2)$-faces are called *ridges*.

2 Overview of the algorithm

We first specify an algorithm that starts from an arbitrary facet and moves to a (unique) adjacent facet, from there again to an adjacent facet and so on until a certain "target" facet is reached:

Algorithm: Pivot to the target facet
(∗) **while** F is not the target facet **do**
(∗∗) *select* a facet f of F;
(∗ f is a ridge of the polytope. ∗)
pivot about f to the (unique) other
facet F' containing f;
$F := F'$;
end while.

It should be clear what pivoting means: A hyperplane rotating about f has one degree of freedom. As a supporting hyperplane (containing P on one side) it has two extreme positions, corresponding to the two facets F and F'. We shall abbreviate the pivot step by a procedure call $F' := pivot(F, f)$.

To make the algorithm concrete we have to determine the *target facet* in step (∗) and specify the pivot selection step (∗∗). We will do this in section 4. For now we assume that a procedure $f := selectfacet(F)$ is available for step (∗∗). If the definition of the target facet and of *selectfacet* fit together in such a way that that the algorithm

(i) never visits a facet twice, and

(ii) always reaches the same target facet, regardless of the starting facet,

then the algorithm implicitly defines a directed tree on the set of facets, which is rooted at the target facet. By carrying out a depth-first search on this tree, starting at the root, it is now possible to visit every facet:

procedure *search*(F);
(∗ In the initial call, F is the target facet. ∗)
begin output facet F;

```
(†)     for every facet f of F do
            F' := pivot(F, f);
            if F' is not the target facet
                        and selectfact(F') = f
            then search(F');
        end for;
    end procedure;
```

Note that line (†) requires an enumeration of facets one dimension lower. Let us for the time being assume that we have procedures $f := \mathit{firstfacet}(F)$ and $f' := \mathit{nextfacet}(F, f)$ that do the job. With these procedures we can rewrite $\mathit{search}(F)$ as follows:

```
    procedure search(F);
    begin output facet F;
        f := firstfacet(F);
        repeat
            F' := pivot(F, f);
            if F' is not the target facet
                        and selectfacet(F') = f
            then search(F');
            f := nextfacet(F, f);
        until f = nil;
    end procedure;
```

In the following algorithm the recursion from the depth-first search procedure is removed. The local variables F and f are removed because they need not be remembered: After returning from $\mathit{search}(F')$, the original values of f and F can be recovered easily from F' because $f = \mathit{selectfacet}(F')$ anf $F = \mathit{pivot}(F', f)$.

Algorithm SEARCH:
Non-recursive search of all facets

```
    F := the target facet;
1:      output facet F;
        f := firstfacet(F);
        repeat
            F' := pivot(F, f);
(*)         if F' is not the target facet
                        and selectfacet(F') = f
            then F := F'; goto 1;
2:          f := nextfacet(F, f);
        until f = nil;
    (* backtrack: *)
    if F is the target facet then STOP;
    F' := F;
    f := selectfacet(F');
    F := pivot(F', f);
    goto 2;
```

The test in line $(*)$, whether F' is the target facet, can be carried out by testing whether $f = \mathit{selextfacet}(F)$. In order to use algorithm SEARCH recursively for enumerating facets of facets, we have to cast it into the form of the procedures $\mathit{firstfacet}(F)$ and $\mathit{nextfacet}(F, f)$:

```
procedure firstfacet(𝓕);
begin return the target facet of 𝓕;
end procedure;

procedure nextfacet(𝓕, F);
(* This procedure carries out a portion of      *)
(* program SEARCH between two successive        *)
(* executions of "output facet".                *)
begin f := firstfacet(F);
      repeat
        F' := pivot(F, f);
(‡)     if F' is not the target facet
                and selectfacet(F') = f
        then return F';
2:      f := nextfacet(F, f);
      until f = nil;
      if F is the target facet of 𝓕
          then return nil;
      f := selectfacet(F);
      F := pivot(F, f);
      goto 2;
end procedure;
```

When $\mathcal{F}$ has dimension 1, $\mathit{nextfacet}(\mathcal{F}, F)$ is computed directly.

There are no more output statements. Instead, they are in a new top level procedure:

```
  program enumerate facets;
  begin 𝓕 := the whole polytope;
      F := firstfacet(𝓕);
      repeat   output facet F;
               F := nextfacet(𝓕, F);
      until F = nil;
  end program;
```

The algorithm is now still recursive in the dimension of the faces. However, we see that the recursive procedures need no local storage, except for their parameters. Thus, an (implicit) stack of length d for the chain of faces corresponding to the chain of recursive calls is the only additional storage that is needed.

In the specific implementation described below, *firstfacet* and *selectfacet* are identical. This makes some simplifications possible. For example, in line (‡), F' can only be the target facet if $f = \mathit{selectfacet}(F') = \mathit{firstfacet}(F')$ is the facet of F through which we have just entered F. Thus we just skip over this statement if we have entered the repeat-loop from the first statement of the procedure (the "normal" entry into the loop), and we can omit the check whether F' is the target vertex.

Also, by the nature of depth-first search, when the algorithm has finished to explore a face F, it returns to the initial facet of F. This is the facet through which the algorithm has entered F; if the procedure *nextfacet* is modified to give this facet, the algorithm can just pivot back via this facet without calling *selectfacet*.

3 A different viewpoint: Exploring the face lattice

The *face lattice* of a polytope is a directed graph whose nodes correspond to the faces, including the empty face $\emptyset$ and the polytope $conv(P)$ itself, and which has an arc between two faces F and f if F is k-dimensional and f is $(k-1)$-dimensional and f is contained in F. Figure 2 shows the face lattice of a cube with a trucated corner (figure 1).

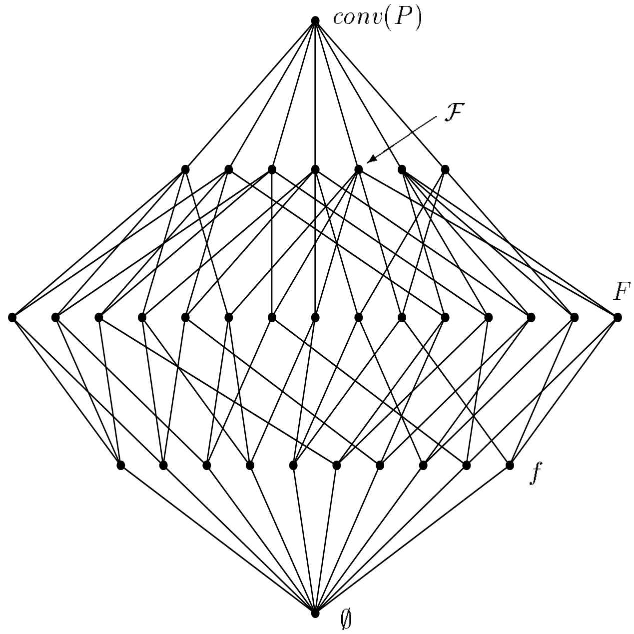

Figure 2: The face lattice for the polytope in figure 1. The arcs are directed from top to bottom.

Our algorithm now starts at the top level and visits every facet, every facet of every facet, and so on, recursively descending into the lower levels. For each facet F of a face $\mathcal{F}$, one facet f of F is selected. Pivoting about f (inside $\mathcal{F}$) corresponds to finding the fourth node on the unique 4-cycle through the nodes $\mathcal{F}$, F, and f in the graph of the face lattice (see figure 2, where the pivot is indicated by the arrow). Under conditions (i) and (ii), these pivots induce a tree on the facets of $\mathcal{F}$, rooted at the target facet of $\mathcal{F}$.

The analysis of the algorithm in section 6 will again refer to the face lattice.

4 The pivoting rule

The easiest way to get some order into the facets of a polytope is an optimization criterion.

Recall that we assume that the origin is contained in the interior of the polytope $conv(P)$. For a given facet F lying on a hyperplane h, let $x_d(F) = x_d(h)$ be the intercept of h, i. e., the intersection of h with the x_d-axis (see figure 3), and set

$$z(F) := z(h) := 1/x_d(h).$$

For a vertical hyperplane we define $z(h) = z(F) := 0$. In other terms, $z(F)$ is the d-th coordinate of the vertex corresponding to F in the polar polytope.

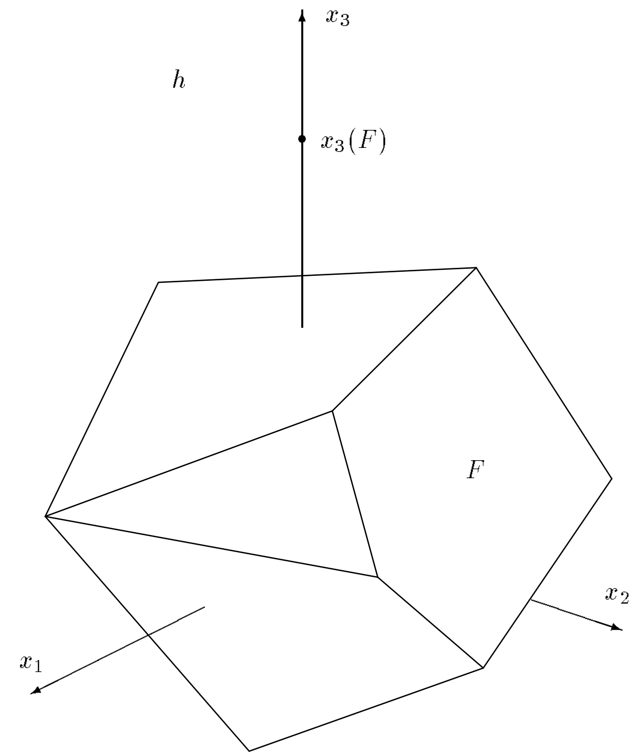

Figure 3: The objective function.

The facet which maximizes $z(F)$ is the facet where the positive x_d-axis pierces through the boundary of the polytope. By considering also the other coordinate axes in a lexicographic manner, we can assume that all facets have distinct values $z(F)$. In particular, the "optimal" facet is unique, and we can define it as our target facet. In the sequel, we will take the objective function z with this lexicographic meaning without mentioning it.

We have to show two things: Firstly, how to extend the ordering of the facets induced by the function z to facets of facets and to facets of arbitrary faces. Secondly, we must define the pivoting facet (procedure *selectfacet*) and show that pivoting about this facet leads to the target facet without getting caught in a cycle.

To order the facets f of a facet F lying on a hyperplane h we proceed as follows: we take the center of gravity C_F of the points of P lying in F, and we push it slightly away from the origin (see figure 4a). $conv(P \cup \{C_F\})$ will have a pyramid built on top of F, with new facets corresponding to the facets of F (see figure 4b). We can now use the ordering of these new facets with respect to z.

Of course, the movement of C_F is infinitesimally small and only conceptual. Computationally, when we look at facets of a face f at some level of the recursion, we have a sequence of points $C_{\mathcal{F}}, C_F, \ldots, C_f$, that in conjunction with any facet of f define a hyperplane h. The derivative of $z(h)$ when C_f moves outwards on a ray through the origin gives a ranking of the facets of f. (Any ray on which C_f moves outwards will give the same ranking.)

Having defined the order of facets by an optimization criterion, a natural choice for the pivoting facet is the (optimal) target facet itself:

> **procedure** *selectfacet*($\mathcal{F}$);
> (∗ identical to *firstfacet* ∗)
> **begin return** the target facet of $\mathcal{F}$, i. e.,
> the optimal facet;
> **end procedure**;

Lemma 1 *Let f be the optimal facet of a facet F which is not the target facet. Then $z(F)$ increases when we pivot about f.*

Proof. Consider the change of $z(h)$ as h rotates about a ridge f to an adjacent facet. $z(h)$ will increase for some ridges and decrease for other ridges. Since F is not optimal, there is a ridge for which $z(h)$ increases. For the optimal rigde, $z(h)$ must therefore also increase.

To make this argument precise, observe two things: The objective function of a ridge f, i. e., the *rate* of change of $z(h)$ as h rotates about f depends on the speed of the rotation, in particular on the position of the point C_F, but the *direction* of the change (increasing or decreasing) is independent of this. Secondly, for any adjacent facet F', we have $z(F') > z(F)$ if and only if $z(h)$ increases as h rotates about the common ridge of F and F'. □

From this lemma we conclude that a sequence of pivots with the pivot rule of selecting always the optimal ridge leads to a sequence of facets with strictly increasing z-values that eventually terminates at the target (optimal) facet. Thus the requirements (i) and (ii) from the introduction are fulfilled.

Lemma 2 *Let D denote the intersection of the x_d-axis with the hyperplane h in which a facet F is contained, and let l denote the ray from C_F towards D. Then the optimal facet f of F is the facet where this ray intersects the boundary of F.*

Proof. Let f be a facet of F and consider the intersection d of the line l with the affine hull of f (see Figure 5 where the situation of figure 4 is shown as it is seen on the hyperplane h through the current facet F). Let

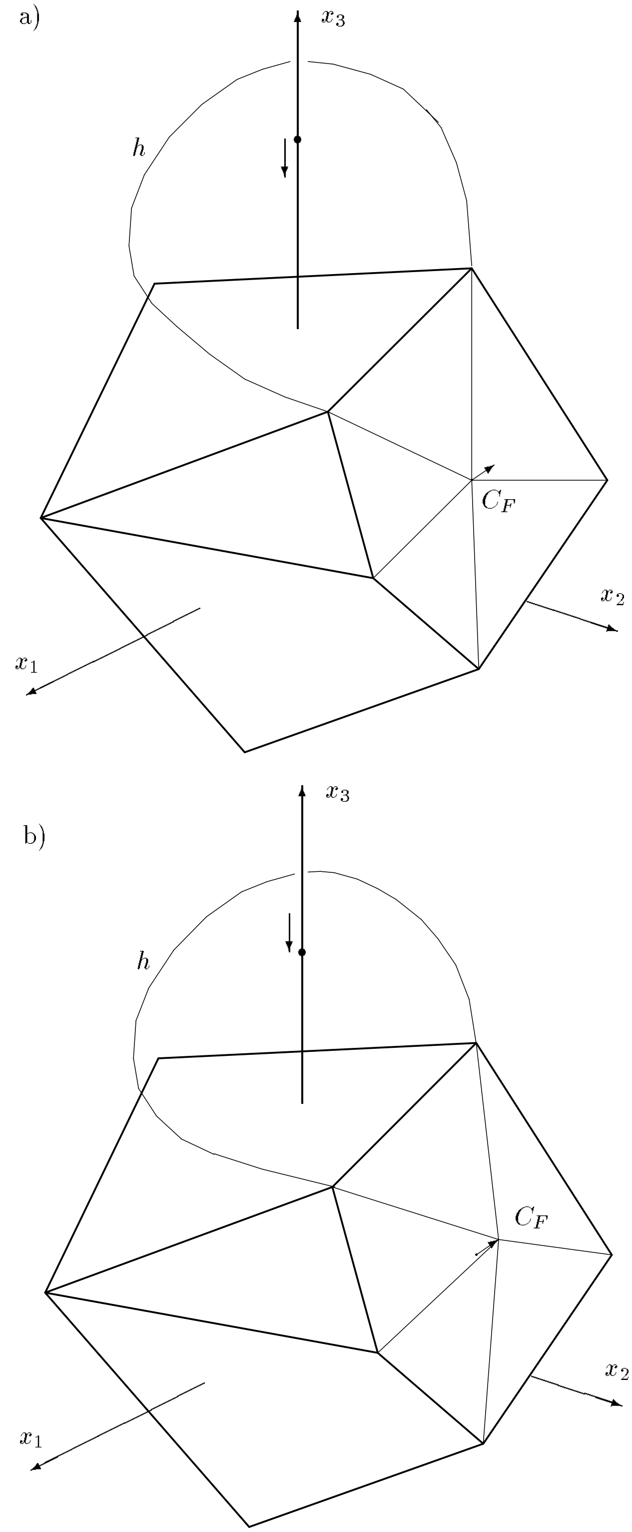

Figure 4: The new vertex C_F in the facet F, as it moves outwards.

us rotate h about f and watch the intersections of h with the ray through C_F and with the x_d-axis. The speeds by which these two points move are related like

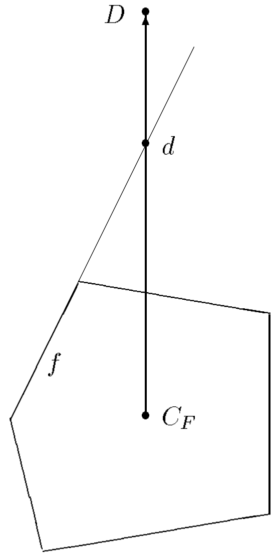

Figure 5: The situation in the hyperplane h of F.

const $\cdot\, \overline{C_F d}/\overline{dD}$, where the constant is independent of f. Thus the biggest rate of change at the x_d-axis for a fixed small movement of C_F is achieved when d is as close to C_F as possible. $\square$

This lemma implies that *the lower-dimensional enumeration problems,* when we restrict our attention to a certain face in the recursive procedure, *can be handled in the same way as the full-dimensional problem*, because the objective function is of the same type.

Note that the objective function for a given face depends on the sequence of faces $\mathcal{F}, F, f, \ldots$ in the stack of recursive calls through which that face was reached. These parameters were not explicitly mentioned in the description of the algorithm in section 2.

In the procedures *selectfacet* and *firstfacet* we need to compute the optimal facet of a given face. This can be done by solving a linear program, for example by the simplex algorithm, by the method of Seidel [1991] in $O(d!n)$ time, or by the method of Welzl and Sharir [1992]

in less than $O(2^d n)$ time.

In most calls of the procedure *selectfacet* we simply have to check whether a given face f is the optimal facet of some face F'. This optimality check is actually a linear program one dimension lower than the computation of the optimal facet from scratch.

5 Implementation

The algorithm can most naturally be implemented in the form of a simplex tableau, like the algorithm of Avis and Fukuda [1991]. Essentially we consider the linear program

$$\text{lex max } (x_d, x_{d-1}, \ldots, x_1)$$
$$\text{subject to } \textstyle\sum_{j=1}^{d} p_{ij} x_j \geq 1, \quad \text{for } i = 1, \ldots n,$$

where p_{ij} are the coordinates of the n given points P_i.

The rows of the tableau correspond to the points of P. We have d additional rows for the components of the lexicographic objective function. At any time during the algorithm, a set of d affinely independent points (a feasible basis) is selected, and the current solution is the hyperplane spanned by these points. The points that lie on the corresponding facet can be recognized as having right-hand sides 0. The points determining the current basis correspond to the columns of the tableau.

The operations described in section 4 can be carried out easily: Adding a point C_F corresponds to adding a row which is the average of the rows corresponding to the points on the face F, and putting this point into the basis. (Actually, it is more sensible to let C_F enter the basis right away when we move to face F.) Pushing C_F outwards means that we consider the column corresponding to C_F as an additional requirement on feasiblility: A basis is now feasible only if the (original) right-hand side together with the entry in the new column is lexicographically non-negative. Adding more points C_f will simply add more components to this lexicographic feasibility criterion. A face is thus implicitly determined by the sequence of points (columns) $C_{\mathcal{F}}, C_F, \ldots, C_f$. This means that the only space that is required for the algorithm is the space for d additional rows in the tableau.

The theorem in the next section bounds the number of pivots of the algorithm. Any pivot operation takes $O(nd)$ steps. Of course, in practice we will restrict the attention to the points that lie on the current face, and the value n in the expression $O(nd)$ is just the number of those points.

A natural improvement is to use the revised simplex algorithm, which operates on the original coordinates of the points and restricts the operations that change values to a $(d \times d)$-submatrix.

6 Analysis

We have seen in section 3 that the algorithm essentially carries out a depth-first search of the face lattice of the polytope $conv(P)$. Let A_k be the number of directed paths in the face lattice (see figure 2) which start at the top level (the polytope itself) and end at a k-dimensional face.

In the algorithm we carry out two types of pivots: Those which are only *tried* and immediately revoked, and the *successful* pivots which are actually performed.

Lemma 3 *The number of pivot operations which are tried at level $d-k$ of the recursion is at most A_{k-1}. The number of successful pivot operations at level $d-k$ of the recursion is less than $2A_k$.*

Proof. For the proof we look at the initial recursive formulation of procedure *search*(F): It walks through the depth-first search tree of all facets of F, and the number of walking steps (pivots) is less than twice the number of vertices visited. This proves the second statement. For each face that is searched, all its facets are tried as pivots. The first statement follows. $\square$

Theorem 1 *The total number of pivots in the algorithm is at most* $3A$, *where* A *is the number of directed paths in the face lattice starting at the top node.* $\square$

The number of linear programs that have to be solved in the procedures *firstface* and *selectfacet* is also bounded by this number.

For the example of d-cubes from the introduction, each k-cube has $2k$ facets. The number A of paths is therefore $\big(2 \cdot 4 \cdot 6 \cdots (2d)\big) + \big(4 \cdot 6 \cdot 8 \cdots (2d)\big) + \big(6 \cdot 8 \cdots (2d)\big) + \cdots + (2d)$. For the 5-cube this number is 6320, which compares favorably with the 30080 feasible bases.

The best case for the facet enumeration algorithm occurs when the facets are highly degenerate but the lower-dimensional faces are simplicial. Consider a "simplicial pyramid". It is built as the convex hull of a $(d-1)$-dimensional simplicial polytope with n vertices together with an additional point outside the hyperplane containing the polytope. Avis and Fukuda's basis enumeration method will find that it has $O(\binom{n}{d})$ feasible bases to visit, whereas the combinatorial complexity of the present facet enumeration algorithm is at most $O(n^{\lfloor (d-1)/2 \rfloor} \cdot d!)$. The term $d!$ can be omitted if the hybrid version described in the following subsection is used.

6.1 A hybrid algorithm

The algorithm of Avis and Fukuda enumerates feasible bases, i. e., simplices. For a simplex it is a trivial matter to enumerate its facets and hence its possible pivots. Thus the basis enumeration algorithm has no need for a recursion on the dimension. We can incorporate this into our algorithm by stopping the recursion whenever the current face is a simplex and enumerating its facets directly. In this way, the algorithm will become similar to the algorithm of Avis and Fukuda, except for the different pivoting strategy. This follows an idea of Swart [1985], who introduced the *abbreviated* face lattice as the combinatorial structure underlying this variation of the algorithm.

7 Further Research

The facet enumeration algorithm in section 2 is presented in a very general way. The realization of *firstfacet* and *selectfacet* by an optimization criterion is just one possibility. It would be nice if a simpler way of defining target facets were found, for example by just considering the indices of the points lying on the facet in some lexicographic way. This would eliminate the need to solve a linear program at every step of the algorithm.

It is an open question whether *selectfacet*(F) can be defined to depend only on F and not on the whole path from the top level to F. Another question to investigate is the connection between our algorithm and the simpler method of perturbing the vertices, which is also a way to deal with degeneracy.

8 References

David Avis and Komei Fukuda [1991]
A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, in: *Proc. 7th Ann. Symp. Computat. Geometry, June 1991*, pp. 98–104; revised version to appear in *Discrete and Computational Geometry* **8** (1992), 295–313.

Andries E. Brouwer, Arjeh M. Cohen, and Arnold Neumaier [1989]
Distance-regular graphs, Springer-Verlag.

D. R. Chand and S. S. Kapur [1970]
An algorithm for convex polytopes, *J. Assoc. Comput. Mach.* **17**, 78–86.

Pey-Chun Chen, Pierre Hansen, Brigitte Jaumard [1991]
On-line and off-line vertex enumeration by adjacency lists, *Operations Research Letters* **10**, 403–409.

T. Christof, M. Jünger, and G. Reinelt [1991]
A complete description of the traveling salesman polytope on 8 nodes, *Operations Research Letters* **10**, 497–500.

T. H. Matheiss and D. S. Rubin [1980]
A Survey and Comparison of Methods for Finding All Vertices of Convex Polyhedral Sets, *Mathematics of Operations Research* **5**, 167–185.

William R. Pulleyblank [1989]
Polyhedral combinatorics, in: *Optimization*, Handbooks in Operations Research and Management Science, vol. 1, eds. G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, North-Holland, pp. 371–446.

Raimund Seidel [1991]
Small-dimensional linear programming and convex hulls made easy, *Discrete and Computational Geometry* **6**, 423–434.

M. Sharir and E. Welzl [1992]
A combinatorial bound for linear programming and related problems, in: *STACS 92, Proc. 9th Ann. Symp. Theoretical Aspects of Computer Science, Febr. 13–15, 1992, Cachan, France*, eds. A. Finkel et al., Lecture Notes in Computer Science **577**, Springer-Verlag, pp. 569–579.

G. Swart [1985]
Finding the convex hull facet by facet, *J. Algorithms* **6**, 17–48.