# Primal–Dual Methods for Vertex and Facet Enumeration

David Bremner*
Department of Mathematics, University of Washington, Seattle, U.S.A.
bremner@math.washington.edu

Komei Fukuda
Department of Mathematics, Swiss Federal Institute of Technology, Lausanne and
Institute for Operations Research, Swiss Federal Institute of Technology,
Zurich, Switzerland
fukuda@ifor.math.ethz.ch

Ambros Marzetta
Institute for Theoretical Computer Science,
Swiss Federal Institute of Technology, Zurich, Switzerland
marzetta@inf.ethz.ch

March 8, 1998

**Abstract**

Every convex polytope can be represented as the intersection of a finite set of halfspaces and as the convex hull of its vertices. Transforming from the halfspace (respectively vertex) to the vertex (respectively halfspace) representation is called *vertex enumeration* (respectively *facet enumeration*). An open question is whether there is an algorithm for these two problems (equivalent by geometric duality) that is polynomial in the input size and the output size. In this paper, we extend the known polynomially solvable classes of polytopes by looking at the dual problems. The *dual* problem of a vertex (facet, respectively) enumeration problem is the facet (vertex) enumeration problem for the same polytope where the input and output are simply interchanged. For a particular class of polytopes and a fixed algorithm, one transformation may be much easier than its dual. In this paper, we propose a new class of algorithms that take advantage of this phenomenon. Loosely speaking, *primal–dual* algorithms use a solution to the easy direction as an oracle to help solve the seemingly hard direction.

## 1 Introduction

A *polytope* is the bounded intersection of a finite set of halfspaces in $\mathbb{R}^d$. The *vertices* of a polytope are those feasible points that do not lie in the interior of a line segment between two other feasible points. Every polytope $P$ can be represented as the intersection of a non-redundant set of halfspaces $\mathcal{H}(P)$ and as the convex hull of its vertices $\mathcal{V}(P)$. The

*Research supported by NSERC Canada, FCAR Québec, and the J.W. McConnell Foundation.

problem of transforming from $\mathcal{H}(P)$ to $\mathcal{V}(P)$ is called *vertex enumeration*; transforming from $\mathcal{V}(P)$ to $\mathcal{H}(P)$ is called *facet enumeration* or *convex hull*.

An algorithm is said to be *polynomial* if the time to solve any instance is bounded above by a polynomial in the size of input and output. We consider the input (respectively, output) size to be the number of real (or rational) numbers needed to represent the input (respectively output); in particular we do not consider the dimension to be a constant. We assume each single arithmetic operation takes a constant amount of time[1].. A *successively polynomial algorithm* is one whose $k$th output is generated in time polynomial in $k$ and the input size $s$, for each $k$ less than or equal to the cardinality of output. Clearly every successively polynomial algorithm is a polynomial algorithm. We assume that a polytope is full dimensional and contains the origin in its interior; under these conditions[2] vertex enumeration and facet enumeration are polynomially equivalent, that is, the existence of a polynomial algorithm for one problem implies the same for the other problem. Several polynomial algorithms (see e.g. [3, 6, 7, 9, 17, 18]) are known under strong assumptions of nondegeneracy, which restrict input polytopes to be simple in the case of vertex enumeration and simplicial in the case of facet enumeration. However, it is open whether there exists a polynomial algorithm in general.

In this paper, we extend the known polynomially solvable classes by looking at the dual problems. The *dual* problem of a vertex (facet, respectively) enumeration problem is the facet (vertex) enumeration problem for the same polytope where the input and output are simply interchanged. For a particular class of polytopes and a fixed algorithm, one transformation may be much easier than its dual. One might be tempted to explain this possible asymmetry by observing that the standard nondegeneracy assumption is not self-dual. Are the dual problems of nondegenerate vertex (face) enumeration problems harder? More generally, are the complexities of the primal and the dual problem distinct?

Here we will show in a certain sense that the primal and dual problems are of the same complexity. More precisely, we will show the following theorem: if there is a successively polynomial algorithm for the vertex (facet, respectively) enumeration problem for a hereditary class of problems, then there is a successively polynomial algorithm for the facet (vertex) enumeration problem for the same class, where a hereditary class contains all subproblems of any instance in the class. We propose a new class of algorithms that take advantage of this phenomenon. Loosely speaking, *primal–dual* algorithms use a solution to the easy direction as an oracle to help solve the seemingly hard direction.

From this general result relating the complexity of the primal and dual problems, and known polynomial algorithms for the primal-nondegenerate case, we arrive at a polynomial algorithm for vertex enumeration for simplicial polytopes and facet enumeration for simple polytopes. We then show how to refine this algorithm to yield an algorithm with time complexity competitive with the algorithms known for the primal-nondegenerate case.

The only published investigation of the dual-nondegenerate case the authors are aware of is in a paper by Peter Gritzmann and Victor Klee [12]. Their approach, most easily understood in terms of vertex enumeration, consists of intersecting the constraints with

[1] This assumption is merely to simplify our discussion. One can easily analyze the complexity of an algorithm in our primal–dual framework for the binary representation model and in general its binary complexity depends only on that of the associated "base" algorithm.

[2] We discuss these assumptions further in Section 2.2.

each defining hyperplane and, after removing the redundant constraints, finding the vertices lying on that facet by some brute force method. David Avis (private communication) has independently observed that this method can be extended to any polytope whose facets are simple (or nearly simple) polytopes. The method of Gritzmann and Klee requires solving $O(m^2)$ linear programs (where $m$ is the number of input halfspaces) to remove redundant constraints. Our approach does not rely on the polynomial solvability of linear programming if an interior point is known (as is always the case for facet enumeration).

### Notation

We start by defining some notation. Recall that $\mathcal{H}(P)$ (respectively $\mathcal{V}(P)$) is the non-redundant halfspace (respectively vertex) description of $P$. We use $m$ for $|\mathcal{H}(P)|$, $n$ for $|\mathcal{V}(P)|$, and $d$ for the dimension $\dim P$. The facets of $P$ are the intersection of the bounding hyperplanes of $\mathcal{H}(P)$ with $P$. We use $\mathbb{O}$ ($\mathbb{O}^k$) and $\mathbb{1}$ ($\mathbb{1}^k$) to denote the vector of all zeros (of length $k$) and all ones (of length $k$) respectively. We treat sets of points and matrices interchangeably where convenient; the rows of a matrix are the elements of the corresponding set. Given (row or column) vectors $a$ and $b$, we use $ab$ to denote the inner product of $a$ and $b$. Since we assume the origin is in the interior of each polytope, each facet defining inequality can be written as $hx \leq 1$ for some vector $h$. For a vector $h$, we use $h^+$, $h^-$, and $h^0$ respectively to denote the set of points $x$ such that $hx \leq 1$, $hx > 1$, and $hx = 1$, respectively. We sometimes identify the halfspace $h^+$ with the associated inequality $hx \leq 1$ where there is no danger of confusion. We use $\mathcal{P}(H)$ to denote the polyhedron $\{\, x \mid Hx \leq \mathbb{1} \,\}$. Similarly we use $\mathcal{H}(P)$ to mean the matrix $H$ where $P = \{\, x \mid Hx \leq \mathbb{1} \,\}$. For a set of points $V$ we use $\mathcal{H}(V)$ to mean $\mathcal{H}(\operatorname{conv} V)$; similarly for a set of halfspaces $H$, we use $\mathcal{V}(H)$ to mean $\mathcal{V}(\mathcal{P}(H))$. We say that $h^+$ is *valid* for a set of points $X$ (or $hx \leq 1$ is a *valid inequality*) if $X \subseteq h^+$. We make extensive use of duality of convex polytopes in what follows. The *proper faces* of a convex polytope are the intersection of some set of facets. By adding the two *improper faces*, the polytope itself and the empty set, the faces form a lattice ordered by inclusion. Two polytopes are said to be *combinatorially equivalent* if their face lattices are isomorphic and *dual* if their face lattices are anti-isomorphic (i.e. isomorphic with the direction of inclusion reversed). The following is well known (see e.g. [5]).

**Proposition 1** *If $P = \operatorname{conv} X$ is a polytope such that $\mathbb{O} \in \operatorname{int} P$, then $Q = \{\, y \mid Xy \leq \mathbb{1} \,\}$ is a polytope dual to $P$ such that $\mathbb{O} \in \operatorname{int} Q$.*

## 2 Primal–Dual Algorithms

In this section we consider the relationship between the complexity of the primal problem and the complexity of the dual problem for vertex/facet enumeration. We will fix the primal problem as facet enumeration in the rest of this paper, but the results can also be interpreted in terms of vertex enumeration. For convenience we assume in this paper that the input polytope is full dimensional and contains the origin as an interior point. While it is easy to see this is no loss of generality in the case of facet enumeration, in the case of vertex enumeration one might need to solve a linear program to find an interior point. We call a family $\Gamma$ of polytopes *facet-hereditary* if for any $P \in \Gamma$, for any $H' \subset \mathcal{H}(P)$, if

$\bigcap H'$ is bounded then $\bigcap H'$ is also in $\Gamma$. The main idea of this paper is summarized by the following theorem.

**Theorem 1** *If there is a successively polynomial vertex enumeration algorithm for a facet-hereditary family of polytopes then there is a successively polynomial facet enumeration algorithm for the same family.*

Simple polytopes are not necessarily facet-hereditary, but each simple polytope can be perturbed symbolically or lexicographically onto a combinatorially equivalent polytope whose facet defining halfspaces are in "general position", i.e. the arrangement of facet inducing hyperplanes defined by the polytope is simple. The family of polytopes whose facet inducing halfspaces are in general position is obviously facet-hereditary.

**Corollary 1** *There is a successively polynomial algorithm for facet enumeration of simple polytopes and for vertex enumeration of simplicial polytopes.*

Proof of Theorem 1 is constructive, via the following simple algorithm. Algorithm 1 takes as input a set $V$ of points in $\mathbb{R}^d$, and a subset $H_0 \subset \mathcal{H}(V)$ such that $\bigcap H_0$ is bounded. We show below how to compute such a set of halfspaces.

**Algorithm 1** PrimalDualFacets$(V, H_0)$

```
H_cur ← H_0
while ∃ṽ ∈ 𝒱(H_cur) \ V do                    FindWitness
    Find h ∈ ℋ(V) s.t. ṽ ∈ h⁻                  DeleteVertex
    H_cur ← H_cur ∪ { h }
endwhile
return H_cur.
```

At every step of the algorithm we maintain the invariant that $\operatorname{conv} V \subseteq \mathcal{P}(H_{\text{cur}})$. When the algorithm terminates, we know that $\mathcal{V}(H_{\text{cur}}) \subseteq V$. It follows that $\mathcal{P}(H_{\text{cur}}) \subseteq \operatorname{conv} V$. There are two main steps in this algorithm that we have labeled FindWitness and Delete-Vertex. The vertex $\tilde{v} \in \mathcal{V}(H_{\text{cur}}) \setminus V$ is a *witness* in the sense that for any such vertex, there must be a facet of $\mathcal{H}(V)$ not yet discovered whose defining halfspace cuts off $\tilde{v}$. From the precondition of the theorem there exists a successively polynomial algorithm to enumerate the vertices of $H_{\text{cur}}$. It follows that in time polynomial in $|V|$ we can find $|V|+1$ vertices of $\mathcal{P}(H_{\text{cur}})$, or discover $\mathcal{V}(H_{\text{cur}}) = V$. If we discover $|V|+1$ vertices, one of these vertices must be a witness. In order to find the facet cutting off a witness (the "DeleteVertex" step), we need to solve a separating hyperplane problem for a point and convex set. The separating hyperplane problem can be solved via the following linear program: maximize $\tilde{v}y$ subject to $Vy \leq \mathbb{1}$. If $y^*$ is a basic optimal solution (i.e. a solution corresponding to a vertex of the polar polytope $P^* = \{\, y \mid Vy \leq \mathbb{1} \,\}$) of the linear program then $y^*x \leq 1$ is the desired separating halfspace. While there are linear programming algorithms polynomial in the bit size of the input, there are not yet any known that are polynomial in $n = |V|$ and $d$, which is what we need for our theorem. It turns out that because we have a halfspace description of the convex hull of the union of our two sets, we can solve the separating hyperplane problem via a much simpler algorithm. The rest of this section is organized as follows. In Section 2.1 we discuss how to implement the DeleteVertex step without solving a linear program. In Section 2.2 we discuss how to preprocess to eliminate the various boundedness and full dimensionality assumptions made above. Taken together, the results of these two sections will establish the following stronger version of Theorem 1:

**Theorem 2** *For any facet hereditary family of polytopes $\Gamma$ if we can generate $k$ vertices of an $m$-facet $d$-polytope $P' \in \Gamma$ (or certify that $P'$ has less than $k$ vertices) in time $O(f(k,m,d))$ then we can enumerate the $m$ facets of an $n$-vertex $d$-polytope $P$ in time*

$$O(nd^3 + mnd^2 + m^2 d + \sum_{i=d+2}^{m} f(n+1, i-1, d)) .$$

In certain cases (such as the dual-nondegenerate case considered in Section 3), we may have a theoretical bound for $f(k,m,d)$ polynomial in $k$, $m$, and $d$. In other cases, such a theoretical bound may be difficult to obtain, but we may have experimental evidence that a certain method (e.g. some heuristic insertion order for an incremental algorithm) is efficient for vertex enumeration for $\Gamma$. In either case the techniques described in this section can be used to obtain an efficient method for facet enumeration as well. It is worth noting that there is no restriction of the input points to be in "convex position". Redundant (interior) input points will have no effect other than to slow down pivot operations and tests for membership in the input (i.e. $m$ will be the total number of input points, including redundant points).

## 2.1 Deleting Vertices without Linear Programming

Our main tool here will be the pivot operation of the simplex method of linear programming. Any inequality system

$$Hx \leq \mathbb{1} \tag{1}$$

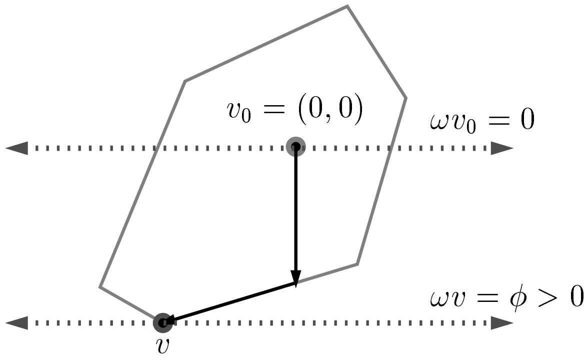
can be represented in the standard "dictionary" form (see e.g. [7]) as follows. We transform each inequality into an equality by adding a slack variable, to arrive at the following system of linear equations or *dictionary*:

$$s = \mathbb{1} - Hx \tag{2}$$

More precisely, a dictionary for (1) is a system obtained by solving (2) for some subset of $m$ slack and original variables (where $m$ is the row size of $H$). A solution to (2) is feasible for (1) if and only if $s \geq \mathbb{0}$. In particular since $H\mathbb{0} < \mathbb{1}$, $s = \mathbb{1}$ is a feasible solution to both. The variables are naturally partitioned into two sets. The variables appearing on the left hand side of a dictionary are called *basic*; those on the right hand side are called *cobasic*. A *pivot* operation moves between dictionaries by making one cobasic variable (the *entering variable*) basic and one basic variable (the *leaving variable*) cobasic.

If we have a feasible point for a polytope and a halfspace description, in $d$ pivot operations we can find a vertex of the polytope. If we ensure that each pivot does not decrease a given objective function, then we have the following.

**Figure 1** The raindrop algorithm




**Lemma 1 (Raindrop Algorithm)** *Given $H \in \mathbb{R}^{m\times d}$, $\omega \in \mathbb{R}^d$ and $v_0 \in \mathcal{P}(H)$, in time $O(md^2)$ we can find $v \in \mathcal{V}(H)$ such that $\omega v \geq \omega v_0$.*

*Proof.* We start by translating our system by $-v_0$ so that our initial point is the origin. As a final row to our dictionary we add the the equation $z = \omega x$ (the *objective row*). Note that by construction, $x = \mathbb{O}$ is a feasible solution. We start a pivot operation by choosing some cobasic variable $x_j$ to become basic. Depending on the sign of the coefficient of $x_j$ in the objective row, we can always increase or decrease $x_j$ without decreasing the value of $z$. As we change the value of $x_j$, some of the basic slack variables will decrease as we get closer to the corresponding hyperplane. By considering ratios of coefficients, we can find one of the first hyperplanes reached. By moving that slack variable to the right hand side (making it cobasic), and moving $x_j$ to the left hand side, we obtain a new dictionary in $O(md)$ time (see e.g. [7] for details of the simplex method). We can continue this process as long as there is a cobasic $x$-variable. After exactly $d$ pivots, all $x$-variables are basic. It follows that the corresponding basic feasible solution is a vertex (See Figure 1). ☐

The raindrop algorithm seems to be part of the folklore of Linear Programming; a generalized version is discussed in [16].

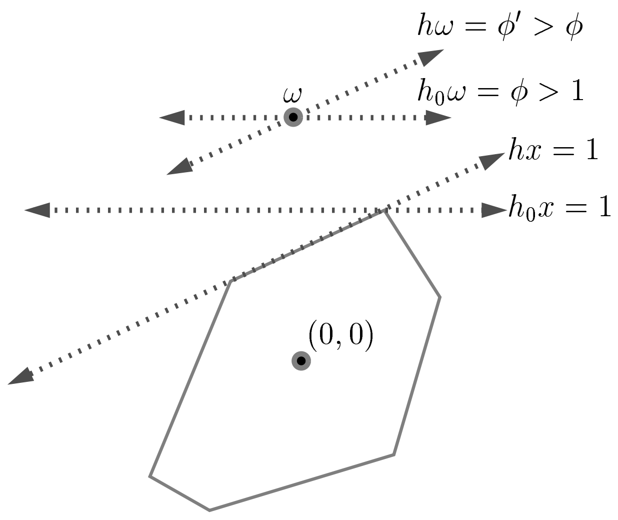
By duality of convex polytopes we have the following.

**Lemma 2 (Dual Raindrop Algorithm)** *Given $V \in \mathbb{R}^{n\times d}$, $\omega \in \mathbb{R}^d$ and $h_0$ such that $V \subset h_0^+$, in $O(nd^2)$ time we can find $h \in \mathcal{H}(V)$ such that $h\omega \geq h_0\omega$.*

Essentially this is the same as the initialization step of a gift-wrapping algorithm (see e.g. [6, 18]), except that we are careful that the point $\omega$ is on the same side of our final hyperplane as of the one we started with. Figure 2 illustrates the rotation dual to the pivot operation in Lemma 1.

We can now show how to implement the DeleteVertex step of Algorithm 1 without linear programming. A *basis* $B$ for a vertex $v \in \mathcal{V}(H)$ is a set of $d$ rows of $H$ such that $Bv = \mathbb{1}$ and rank $B = d$. We can obviously find a basis in polynomial time; in the pivoting based algorithms in the following sections we will always be given a basis for $v$.

**Figure 2** Pivoting from a valid inequality to a facet.



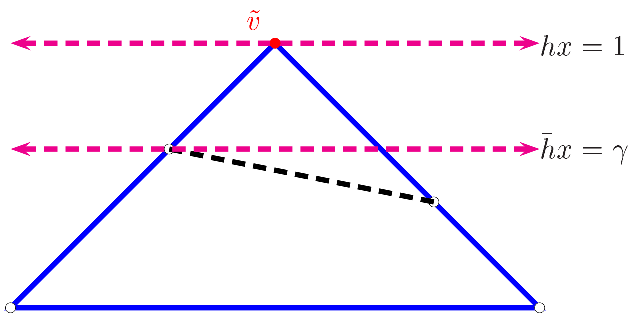

**Lemma 3 (DeleteVertex)** *Given $V \in \mathbb{R}^{n\times d}$, $H_0 \subset \mathcal{H}(V)$, $\tilde{v} \in \mathcal{V}(H_0) \setminus V$, and a basis $B$ for $\tilde{v}$, we can find $h \in \mathcal{H}(V)$ such that $\tilde{v} \in h^-$ in time $O(nd^2)$.*

*Proof.* Let $\bar{h} = \frac{1}{d}\sum_{b\in B} b$. The inequality $\bar{h}x \le 1$ is satisfied with equality by $\tilde{v}$ and with strict inequality by every $v \in V$ (since $\tilde{v}$ is the unique vertex of $\mathcal{P}(H_0)$ lying on $\bar{h}^0$; see Figure 3). Let $\gamma = \max_{v\in V} \bar{h}v$. Since $\mathbb{O} \in \operatorname{int}\operatorname{conv} V$, $\gamma > 0$. Let $h_0 = \bar{h}/\gamma$. The constraint $h_0x \le 1$ is valid for $\operatorname{conv} V$, but $h_0\tilde{v} > 1$. The lemma then follows from Lemma 2. ☐

If we are not given a basis for the vertex $\tilde{v}$ we wish to cut off, we can use the mean of the outward normals of all facets meeting at $\tilde{v}$ in place of the vector $\bar{h}$. This mean vector can be computed in time $O(|H_0|d)$ time.

**Corollary 2** *Given $V \in \mathbb{R}^{n\times d}$, $H_0 \subset \mathcal{H}(V)$, and $\tilde{v} \in \mathcal{V}(H_0) \setminus V$, we can find $h \in \mathcal{H}(V)$ such that $\tilde{v} \in h^-$ in time $O(nd^2 + |H_0|d)$.*

**Figure 3** Illustrating the proof of Lemma 3.



It will prove useful below to be able to find a facet of $\operatorname{conv} V$ that cuts off a particular extreme ray or direction of unboundedness for our current intermediate polyhedron.

**Lemma 4 (DeleteRay)** *Given $V \in \mathbb{R}^{n\times d}$ and $r \in \mathbb{R}^d \setminus \{\,\mathbb{O}\,\}$, in $O(nd^2)$ time we can find $h \in \mathcal{H}(V)$ such that $hr > 0$.*

*Proof.* The proof is similar to that of Lemma 3. Let $\gamma = \max_{v\in V} rv$ . Since $\mathbb{O} \in \operatorname{int}\operatorname{conv} V$, $\gamma > 0$. Let $h_0 = r/\gamma$. The constraint $h_0 x \leq 1$ is valid for $\operatorname{conv} V$, but $h_0 r = (r\cdot r)/\gamma > 0$. By Lemma 2, in $O(nd^2)$ time we can compute $h \in \mathcal{H}(V)$ such that $hr \geq h_0 r > 0$. $\square$

## 2.2 Preprocessing

We have assumed throughout that the input polytopes are full dimensional and contain the origin as an interior point. This is polynomially equivalent to assuming that along with a halfspace or vertex description of $P$, we are given a relative interior point point, i.e. an interior point of $P$ in aff $P$. Given a relative interior point, then (either representation of) $P$ can be shifted to contain the origin as an interior point and embedded in a space of dimension $\dim P$ in $O(Nd^2)$ time by elementary matrix operations, where $N$ is the number of input halfspaces or points.

Finding a relative interior point in a set of points requires only the computation of the centroid. On the other hand, finding a relative interior point of the intersection of a set of halfspaces $H$ requires solving at least one (and no more than $|H|$) linear programs. Since we are interested here in algorithms polynomial in $n$, $m$, and $d$, and there not yet any such linear programming algorithms, we thus assume that the relative interior point is given.

In order to initialize Algorithm 1, we need to find some subset $H_0 \subset \mathcal{H}(V)$ whose intersection is bounded. We start by showing how to find a subset whose intersection is pointed, i.e. has at least one vertex.

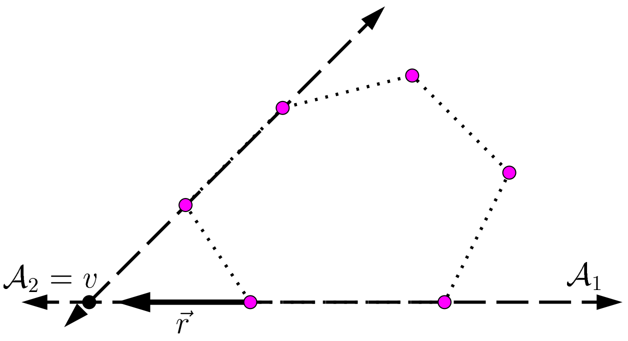
**Algorithm 2** FindPointedCone

```
H ← ∅. r⃗ ← x ∈ ℝ^d \ { 𝕆 }. 𝒜 ← ℝ^d.
while |H| < d do
    h ← DeleteRay(r⃗, V)
    H ← H ∪ { h }
    𝒜 ← 𝒜 ∩ h^0
    Let a and b distinct points in 𝒜.
    r⃗ ← a − b.
endwhile
return H
```

**Lemma 5** *Given $V \in \mathbb{R}^{n\times d}$, in $O(nd^3)$ time, Algorithm 2 computes subset $H \subset \mathcal{H}(V)$ such that $\bigcap H$ defines a vertex.*

*Proof.* We can compute a parametric representation of the affine subspace $\mathcal{A}$ defined by the intersection of all hyperplanes found so far in $O(d^3)$ time by Gaussian Elimination. With each DeleteRay call in Algorithm 2, we find a hyperplane that cuts off some ray in the previous affine subspace (see Figure 4). It follows that the dimension of $\mathcal{A}$ decreases with every iteration. $\square$

**Figure 4** Successive affine subspaces $\mathcal{A}_i$ computed by Algorithm 2




We now show how to augment the set of halfspaces computed by Algorithm 2 so that the intersection of our new set is bounded. To do so, we use a constructive proof of Carathéodory's Theorem. The version we use here is based on that presented by Edmonds [10]. Similar ideas occur in an earlier paper by Klee [14].

**Lemma 6 (The Carathéodory Oracle)** *Given $H \in \mathbb{R}^{m\times d}$ such that $\mathcal{P}(H)$ is a bounded d-polytope and $v_0 \in \mathcal{P}(H)$, in time $O(md^3)$ we can find $V \subset \mathcal{V}(H)$ such that $v_0 \in \operatorname{conv} V$ and $|V| \le d+1$.*

*Proof.* (sketch) Let $P = \mathcal{P}(H)$. Apply Lemma 1 to find $v \in \mathcal{V}(H)$. If $v = v_0$, return $v$. Otherwise, find the point $z$ at which the ray $\overrightarrow{vv_0}$ exits $P$. Intersect all constraints with the minimal face containing $z$ and recurse with $z$ as the given point in the face. The recursively computed set, along with $v$, will contain $v_0$ in its convex hull. $\square$

By duality of convex polytopes, we have the following:

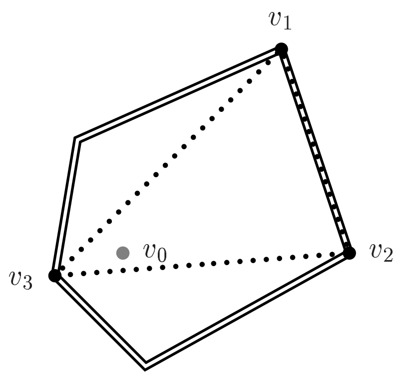
**Lemma 7 (The Dual Carathéodory Oracle)** *Given a d-polytope $P = \operatorname{conv} V$ and $h_0$ such that $V \subset h_0^+$, we can find in time $O(|V|d^3)$, some $H \subset \mathcal{H}(V)$ such that $h_0 \in \operatorname{conv} H$ and $|H| \le d+1$.*

Figure 5(a) illustrates the application of the Carathéodory oracle to find a subset of vertices of a polygon containing an interior point $v_0$ in their convex hull. In Figure 5(b) the equivalent dual interpretation of finding a set of facets that imply a valid inequality is shown. In order to understand the application of Lemma 7, we note the following:

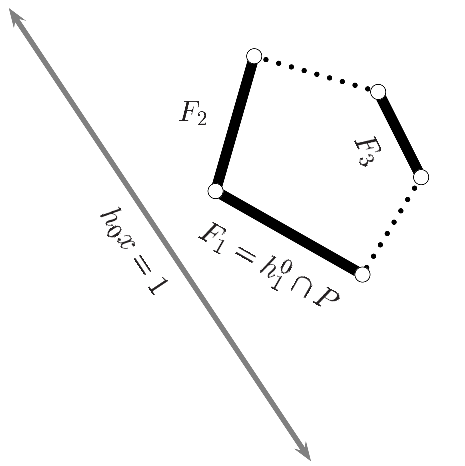
**Proposition 2** *Let $P = \{\, x \mid Ax \le \mathbb{1} \,\}$ and $Q = \{\, x \mid A'x \le \mathbb{1} \,\}$ be polyhedra such that each row $a'$ of $A'$ is a convex combination of rows of $A$. $P \subseteq Q$.*

Using Lemma 5 and Lemma 7, we can now find a subset of $\mathcal{H}(V)$ whose intersection is bounded.

**Figure 5** The primal and dual Carathéodory oracles.

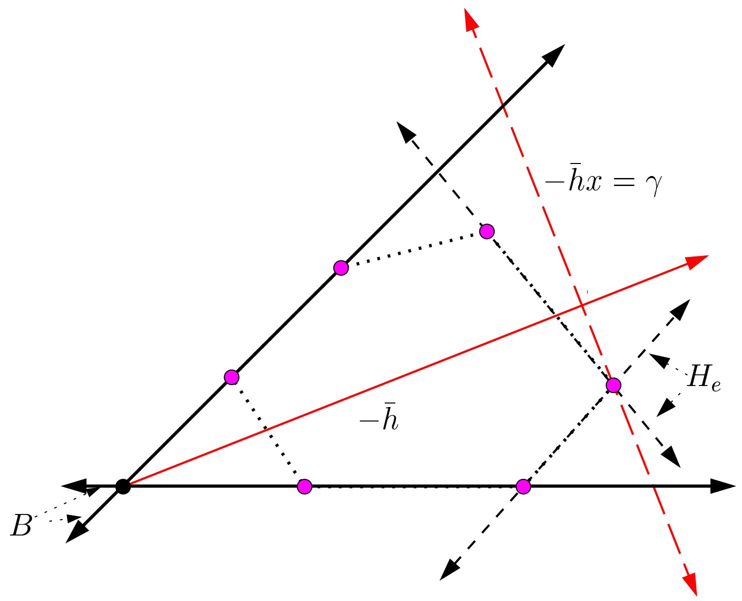



(a) Using the Carathéodory Oracle to find a set of points whose convex hull contains $v_0$.




(b) The dual problem of finding a set of facets that imply a given valid constraint $h_0x \leq 1$.

**Figure 6** Illustrating the proof of Lemma 8



**Lemma 8** *Given $V \in \mathbb{R}^{n \times d}$, in time $O(nd^3)$ we can compute a subset $H \subseteq \mathcal{H}(V)$ such that $\bigcap H$ is bounded and $|H| \leq 2d$.*

*Proof.* We start by computing set $B$ of $d$ facet defining halfspaces whose intersection defines a vertex, using Algorithm 2. The proof is then similar to that of Lemma 3 and Lemma 4. Compute the mean vector $\bar{h}$ of the normal vectors in $B$ (see Figure 6). Let $\gamma = \max_{v \in V} \ -\bar{h}v$ . Let $h_0 = -\bar{h}/\gamma$. Note that $h_0^+$ is valid for $V$, but any ray feasible for $\bigcap B$ will be cut off by this constraint; hence $\mathcal{P}(B) \cap h_0^+$ is bounded. Now by applying Lemma 7 we can find a set of halfspaces $H_e \subset \mathcal{H}(V)$ such that $h_0 \in \operatorname{conv} H_e$. Since $h_0^0$ contains at least one vertex of $V$, $|H_e| \leq d$. By Proposition 2, $\mathcal{P}(B \cup H_e)$ is bounded. □

# 3 The Dual-Nondegenerate Case

In this section we describe how the results of the previous section lead to a polynomial algorithm for facet enumeration of simple polytopes. We then give a refinement of this algorithm that yields an algorithm whose time complexity is competitive with the known algorithms for the primal nondegenerate case.

From the discussion above, we know that to achieve a polynomial algorithm for facet enumeration on a particular family of polytopes we need only have a polynomial algorithm for vertex enumeration for each subset of facet defining halfspaces of a polytope in the family. Dual-nondegeneracy (i.e. simplicity) is not quite enough in itself to guarantee this, but it is not difficult to see that the halfspaces defining any simple polytope can be perturbed so that they are in general position without affecting the combinatorial structure of the polytope. In this case each dual sub-problem is solvable by any number of pivoting methods (see e.g. [3, 7, 9]). Equivalently (and more cleanly) we can use lexicographic ratio testing (see Section 4.1) in the pivoting method. A basis is a subset of $\mathcal{H}(P)$ whose bounding hyperplanes define a vertex of $P$. Although a pivoting algorithm may visit many bases (or perturbed vertices) equivalent to the same vertex, notice that any vertex of the input is simple hence will have exactly one basis. It follows that we can again guarantee to find a witness or find all vertices of $\mathcal{P}(H_{\text{cur}})$ in at most $n+1$ bases (where $n = |V|$, as before) output by the pivoting algorithm. In the case where each vertex is not too degenerate, say at most $d + \delta$ facets meet at every vertex for some small constant $\delta$, we may have to wait for as many as $n \cdot \binom{d+\delta}{\delta} + 1$ bases. Of course this grows rather quickly as a function of $\delta$, but is polynomial for $\delta$ constant. In the rest of this section we assume for ease of exposition that the polytope under consideration is simple.

It is not completely satisfactory to perform a vertex enumeration from scratch for each verification (FindWitness) step since each succeeding input to the vertex enumeration algorithm consists of adding exactly one halfspace to the previous input. We now show how to avoid this duplication of effort. We are given some subset $H_{\text{cur}} \subset \mathcal{H}(V)$ such that $\mathcal{P}(H_{\text{cur}})$ is bounded and a starting vertex $v \in \mathcal{V}(H_{\text{cur}})$ (we can use the raindrop algorithm to find a starting vertex in $O(|H_{\text{cur}}|d^2)$ time).

Algorithm 3 is a standard pivoting algorithm for vertex enumeration using depth first search. The procedure ComputeNeighbour$(v, j, H_{\text{cur}})$ finds the $j$-th neighbour of $v$ in $\mathcal{P}(H_{\text{cur}})$. This requires $O(md)$ time to accomplish using a standard simplex pivot. To

**Algorithm 3** dfs($v, H_{\text{cur}}$)

```
for j ∈ 1 ... d do
    v' ← ComputeNeighbour(v, j, H_cur)
    if new(v') then
        dfs(v', H_cur)
    endif
endfor
```

check if a vertex is new (i.e. previously undiscovered by the depth first search) we can simply store the discovered vertices in some standard data structure such as a balanced tree, and query this structure in $O(d \log n)$ time.

We could use Algorithm 3 as a subroutine to find witnesses for Algorithm 1, but we can also modify Algorithm 3 so that it finds new facets as a side effect. We are given a subset $H_0 \subset \mathcal{H}(V)$ as before and a starting vertex $v \in \mathcal{V}(H_0)$ with the additional restriction that $v$ is a vertex of the input. In order to find a vertex of $\mathcal{P}(H_0)$ that is also a vertex of the input, we find an arbitrary vertex of $\mathcal{P}(H_0)$ using Lemma 1. If this vertex is not a vertex of the input then we apply DeleteVertex to find a new halfspace which cuts it off, and repeat. In what follows, we assume the halfspaces defining the current intermediate polytope are stored in some global dictionary; we sometimes denote this set of halfspaces as $H_{\text{cur}}$. We modify Algorithm 3 by replacing the call to ComputeNeighbour with a call to the procedure ComputeNeighbour2. In addition to the neighbouring vertex $v'$, ComputeNeighbour2 computes the (at most one) halfspace defining $v'$ not already known. Suppose we have found $v$ (i.e. $v$ is a vertex of the current intermediate polytope). Since $P$ is simple we must have also found all of the halfspaces defining $v$. It follows that we have a halfspace description of each edge leaving $v$. Since we have a halfspace description of the edges, we can pivot from $v$ to some neighbouring vertex $v'$ of the current intermediate polytope. If $v' \in V$ then we know $v'$ must be adjacent to $v$ in conv $V$; otherwise we can cut $v'$ off using our DeleteVertex routine. If $P$ is simple, then no perturbation is necessary, since we will cut off degenerate vertices rather than trying to pivot away from them. Thus ComputeNeighbour2 can be implemented as in Algorithm 4.

**Algorithm 4** ComputeNeighbour2($v, j, H_{\text{cur}}$)

```
repeat
    ṽ ← ComputeNeighbour(v, j, H_cur)
    If ṽ ∉ V then
        h ← DeleteVertex(ṽ, H_cur, V)
        AddToDictionary(h, H_cur)
    end if
until ṽ ∈ V
return ṽ
```

**Lemma 9** *With $O(mnd)$ preprocessing,* ComputeNeighbour2 *takes time $O(md + k(md + nd^2))$, where $k$ is the number of new halfspaces discovered.*

*Proof.* As mentioned above, ComputeNeighbour takes $O(md)$ time. The procedure AddToDictionary merges the newly discovered halfspace into the global dictionary. Since $P$ is simple, we know the new halfspace will be strictly satisfied by the current vertex $v$; it follows that we can merge it into the dictionary by making the slack variable basic. This amounts to a basis transformation of the bounding hyperplane, which can be done in $O(d^2)$ time.

Since the search problem is completely static (i.e. there are no insertions or deletions), it is relatively easy to achieve a query time of $O(d + \log n)$, with a preprocessing cost of $O(n(d + \log n))$ using e.g. $kd$-trees [15]. We claim that the inequality $n \leq 2^{md}$ follows from the Upper Bound Theorem. For $0 \leq d \leq 3$ this is easily verified. For $d \geq 4$,

$$\begin{aligned} n &\leq 2\binom{m - \lfloor d/2 \rfloor}{\lfloor d/2 \rfloor} && \text{Upper Bound Theorem} \\ &\leq \frac{2m^{\lfloor d/2 \rfloor}}{\lfloor d/2 \rfloor !} \\ &\leq m^{d/2} = 2^{(d \log m)/2} && d \geq 4 \end{aligned}$$

It follows that $d + \log n \leq 2md$, hence the query time is $O(md)$, and the preprocessing time is $O(nmd)$. Since each pivot in ComputeNeighbour2 that does not discover a vertex of $V$ discovers a facet of $\operatorname{conv} V$, we can charge the time for those pivots to the facets discovered. □

**Algorithm 5** pddfs$(v, H_0)$

$H_{\text{cur}} \leftarrow H_0$
For $j \in 1 \ldots d$ do
    $v' \leftarrow$ ComputeNeighbour2$(v, j, H_{\text{cur}})$    `add new halfspaces to` $H_{\text{cur}}$
    if new$(v')$ then
        $H_{\text{cur}} \leftarrow H_{\text{cur}} \cup$ pddfs$(v')$
    endif
endfor
return $H_{\text{cur}}$

A depth first search based primal–dual algorithm is given in Algorithm 5. Note that we do not need an additional data structure or query step to determine if a $v'$ is newly discovered. We simply mark each vertex as discovered when we search in ComputeNeighbour2. Furthermore, for $P$ simple, $m \leq n$. Thus we have the following:

**Theorem 3** *Given $V \in \mathbb{R}^{n \times d}$, if $\operatorname{conv} V$ is simple, we can compute $H = \mathcal{H}(V)$ in time $O(n|H|d^2)$.*

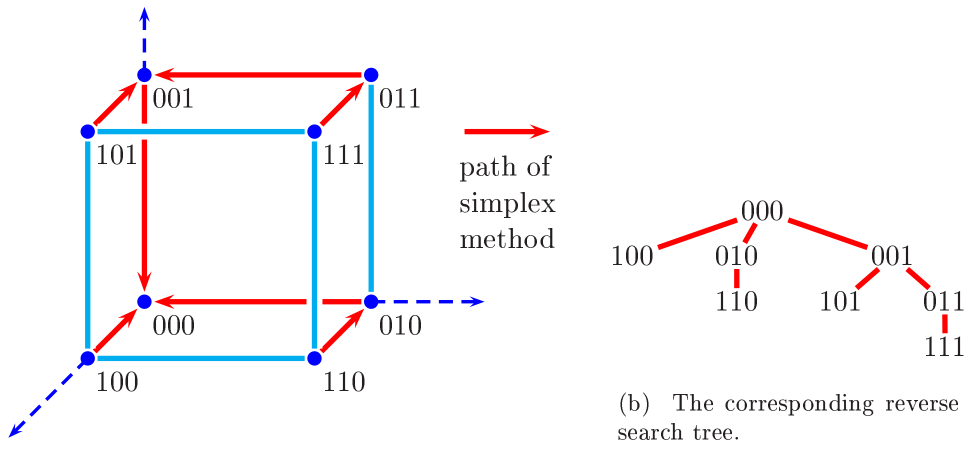
# 4 The Dual-Degenerate Case

We would like an algorithm that is useful for moderately dual-degenerate polytopes. In a standard pivoting algorithm for vertex enumeration based on depth or breadth first search, previously discovered bases must be stored. Since the number of bases is not necessarily polynomially bounded in the dual-degenerate case[3] we turn to *reverse search* [3] which allows us to enumerate the vertices of a non-simple polytope without storing the bases visited. The rest of this section is organized as follows. Section 4.1 explains how to use reverse search for vertex enumeration of non-simple polytopes via lexicographic pivoting. Section 4.2 shows how to construct a primal dual facet enumeration algorithm analogous to Algorithm 5 but with the recursion or stack based depth first search replaced by the "memoryless" reverse search.

## 4.1 Lexicographic Reverse Search

The essence of reverse search in the simple case is as follows. Choose an objective function (direction of optimization) so that there is a unique optimum vertex. Fix some arbitrary pivot rule. From any vertex of the polytope there is a unique sequence of pivots taken by the simplex method to this vertex (see Figure 7(a)). If we take the union of these paths to the optimum vertex, it forms a tree, directed towards the root. It is easy to see algebraicly that the simplex pivot is reversible; in fact one just exchanges the roles of the leaving and entering variable. Thus we can perform depth-first search on the "simplex tree" by reversing the pivots from the root (see Figure 7(b)). No storage is needed to backtrack, since we merely pivot towards the optimum vertex.

**Figure 7** Reverse search on a 3-cube




(a) The "simplex tree" induced by the objective $-\mathbb{1}$.

(b) The corresponding reverse search tree.

[3]Even if the number of bases is bounded by a small polynomial in the input size, any super-linear space usage may be impractical for large problems.

In this section we discuss a technique for dealing with degeneracy in reverse search. In essence what is required is a method for dealing with degeneracy in the simplex method. Here we use the method of *lexicographic pivoting*, which can be shown to be equivalent to a standard symbolic perturbation of the constant vector $b$ in the system $Ax \leq b$ (see e.g. [7] for discussion). Since the words "lexicographic" and "lexicographically" are somewhat ubiquitous in the remainder of this paper, we sometimes abbreviate them to "lex".

In order to present how reverse search works in the non-simple case, we need to discuss in more detail the notions of dictionaries and pivoting used in Section 2. We start by representing a polytope as a system of linear equalities where all of the variables are constrained to be nonnegative. Let $P$ be a $d$-polytope defined by a system of $m$ inequalities. As before, convert each inequality to an equality by adding a slack variable. By solving for the original variables along with some set of $m-d$ slacks and eliminating the original variables from the $m-d$ equations with slack variables on the left hand side, we arrive at the *slack representation* of $P$. Geometrically, this transformation can be viewed as coordinatizing each point in the polyhedron by its scaled distance from the bounding hyperplanes. By renaming slack variables, we may assume that the slack representation has the form

$$Ax = b \qquad \text{where} \qquad A = \begin{bmatrix} I & A' \end{bmatrix}, \qquad A' \in \mathbb{R}^{(m-d)\times d}. \tag{3}$$

For $J \subset \mathbb{Z}^+$ and vector $x$, let $x_J$ denote the vector of elements of $x$ indexed by $J$. Similarly, for matrix $A$, let $A_J$ denote the subset of columns of $A$ indexed by $J$. If $\operatorname{rank} A_J = |J| = \operatorname{rank} A$, we call $J$ a *basis* for $A$, and call $A_J$ a *basis matrix*. Suppose $B \subset \{1 \ldots m\}$ defines a basis of (3) (i.e. a basis for $A$). Let $C$ (the *cobasis*) denote $\{1 \ldots m\} \setminus B$. We can rewrite (3) as

$$b = A_B x_B + A_C x_C \,.$$

Rearranging, we have the familiar form of a dictionary

$$x_B = A_B^{-1} b - A_B^{-1} A_C x_C \,. \tag{4}$$

The solution $\beta = A_B^{-1} b$ obtained by setting the cobasic variables to zero is called a *basic* solution. If $\beta \geq \mathbb{O}$, then $\beta$ is a called a *basic feasible solution* or *feasible basis*. Each feasible basis of (3) corresponds to a basis of some vertex of the corresponding polyhedron, in the sense of an affinely independent set of $d$ supporting hyperplanes; by setting $x_i = 0$, $i \in C$ we specify $d$ inequalities to be satisfied with equality. If the corresponding vertex is simple, then the resulting values for $x_B$ will be strictly positive, i.e. no other inequality will be satisfied with equality. In the rest of this paper we use basis in the (standard linear programming) sense of a set of linearly independent columns of $A$ and reserve *cobasis* for the corresponding set of supporting hyperplanes incident on the vertex (or equivalently, the set of indices of the corresponding slack variables). A *pivot* operation moves between feasible bases by replacing exactly one variable in the cobasis with one from the basis. To pivot to a new basis, start by choosing some cobasic variable $x_k$ in $C$ to increase. Let $\beta = A_B^{-1} b$ and let $A' = A_B^{-1} A_C$. The standard simplex ratio test looks for the first basic variable forced to zero by increasing $x_k$, i.e. it looks for

$$\min_{a'_{ik} > 0} \frac{\beta_i}{a'_{ik}} \,.$$

In the general (non-simple) case, there will be ties for this minimum ratio. Define

$$L(B) \equiv \begin{bmatrix}\beta & A_B^{-1}\end{bmatrix}$$

$$L'(B)_{ij} \equiv \begin{cases} \infty & \text{if } a'_{ik} = 0 \\ L(B)_{i,j}/a'_{ik} & \text{otherwise} \end{cases}$$

To choose a variable to leave the basis, we find the lexmin row of $L'(B)$, i.e. we first apply the standard ratio test to $\beta$ and then break ties by applying the same test to successive columns of $L(B)$. Intuitively, this simulates performing the standard ratio test in a perturbed system $Ax \leq b + \vec{\varepsilon}$ where $\vec{\varepsilon}_i = \epsilon^i$ for some $0 < \epsilon \ll 1$. This perturbation is equivalent to sequentially perturbing the hyperplanes in index order, with each successive hyperplane pushed outward by a smaller amount. That there is a unique choice for the leaving variable (i.e. that the corresponding vertex of the perturbed polytope is simple) follows from the fact that $A_B^{-1}$ is non-singular.

A vector $x$ is called *lexicographically positive* if $x \neq \mathbb{O}$ and the lowest indexed nonzero entry is positive. A basis $B$ is called lexicographically positive if every row of $L(B)$ is lexicographically positive. Let $B$ be a basis set and let $C$ be the corresponding cobasis set. Given an objective vector $\omega$, the *objective row* of a dictionary is defined by

$$z = \omega x = \omega_B x_B + \omega_C x_C$$

substituting for $x_B$ from (4),

$$= \omega_B A_B^{-1} b + (\omega_C - \omega_B A_B^{-1} A_C) x_C \,.$$

The simplex method chooses cobasic variable to increase with a positive coefficient in the *cost row* $\omega_C - \omega_B A_B^{-1} A_C$ (i.e. a variable $x_j$ s.t. increasing $x_j$ will increase the objective value $z$). Geometrically, we know that increasing a slack variable $x_k$ will increase (resp. decrease) the objective function $\omega x$ iff the inner product of the objective vector with the outer normal of the corresponding halfspace $h_k^+$ is negative (resp. positive). Every cobasis $C$ for vertex $v$ defines a polyhedral cone $P_C$ with apex $v$ containing $P$. A cobasis is optimal for objective vector $\omega^* \in R^d$ if $(v + \omega^*) \in (v - P_C)$ (note that $\omega^*$ is the original objective vector before transforming to the slack representation). Reinterpreting in terms of the slack representation, we have the following standard result of linear programming (see e.g. [7]).

**Proposition 3** *If the cost row has no positive entry, then the current basic feasible solution is optimal.*

If the entering variable is chosen with positive cost row coefficient, and the leaving variable is chosen by the lexicographic ratio test, we call the resulting pivot a *lexicographic pivot*. A vector $v$ is *lexicographically greater* than a vector $v'$ if $v - v'$ is lexicographically positive. The following facts are known about lexicographic pivoting:

**Proposition 4** [13] *Let $S$ be lexicographically positive basis, $T$ be a basis arrived at from $S$ by a lexicographic pivot, and $\omega$ a nonzero objective vector.*

(a) *$T$ is lexicographically positive, and*

(b) *$\omega_T L(T)$ is lexicographically greater than $\omega_S L(S)$.*

A basis is called *lex optimal* if it is lexicographically positive, and there are no positive entries in the corresponding cost row. In order to perform reverse search, we would like a unique lex optimal basis. We claim that if $C = \{ m-d+1 \dots m \}$, we can fix $C$ as the unique lex optimal basis by choosing as the objective function $-\sum_{i \in C} x_i$. This is equivalent to choosing the mean of the outward normals of the hyperplanes in $C$ as objective direction. If we consider an equivalent perturbed polytope, the intuition is that all of the perturbed vertices corresponding to a single original vertex are contained in the cone defined by the lex maximal cobasis (see Figure 8).

**Lemma 10** *Let $S = \{ 1 \dots m-d \}$ denote the initial basis defined by the slack representation. For objective vector $\omega = \left[\mathbb{O}^{m-d}, -\mathbb{1}^{d}\right]$, a lex positive basis $B$ has a positive entry in the cost row if and only if $B \neq S$.*

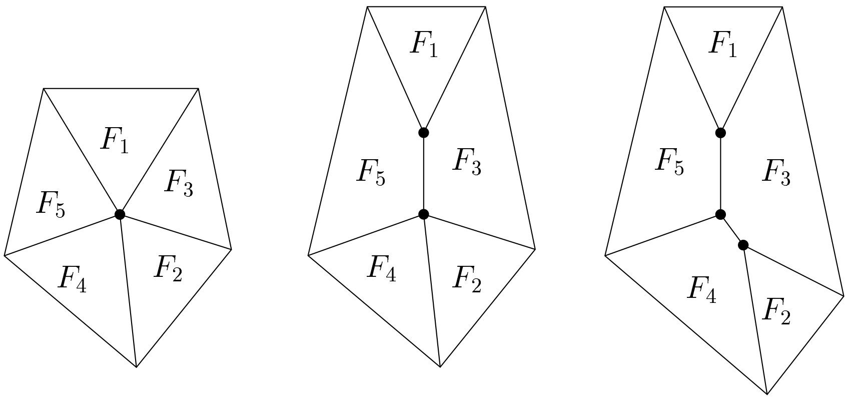
*Proof.* The cost row for $S$ is $-\mathbb{1}^d$. Let $B$ be a lex positive basis distinct from $S$, and let $\beta$ denote the basic part of the corresponding basic feasible solution. Let $k$ denote the number of non-identity columns in $A_B$. If $\omega_B \beta < 0$, then there must be some positive entry in the cost row since $\beta$ is not optimal. Suppose that $\omega_B \beta = 0$. It follows that $\beta = \left[\beta', \mathbb{O}^k\right]$ since $\omega_B = \left[\mathbb{O}^{m-d-k}, -\mathbb{1}^k\right]$ and $\beta \geq \mathbb{O}$. Let $j$ be the first column of $A_B$ that is not column $j$ of an $(m-d) \times (m-d)$ identity matrix. Let $a = \left[\mathbb{O}, \hat{a}\right]$ denote row $j$ of $A_B$. Since the first $m-d-k$ columns of $A_B$ are identity columns, $\hat{a}$ is a $k$-vector. Let $\alpha = \left[\alpha', \hat{\alpha}\right]$ be column $j$ of $A_B^{-1}$, where $\hat{\alpha}$ is also a $k$-vector. Since $\hat{a}\hat{\alpha} = 1$, we know $\hat{\alpha} \neq \mathbb{O}$. By the lex positivity of $L(B)$, along with the fact that $\beta = \left[\beta', \mathbb{O}^k\right]$ and the fact that the first $j-1$ columns of $A_B^{-1}$ are identity columns, it follows that $\hat{\alpha}$ has no negative entries. It follows that element $j$ of $\omega_B A_B^{-1}$ is negative. Since identity column $j$ is not in $A_B$, it must be present in $A_C$, in position $j' < k$. It follows that element $j'$ of $\omega_B A_B^{-1} A_C$ is negative, hence element $j'$ of the cost row is positive. □

From the preceding two lemmas, we can see that the lexicographically positive bases can be enumerated by reverse search from a unique lex optimal basis. The following tells us that this suffices to enumerate all of the vertices of a polytope.

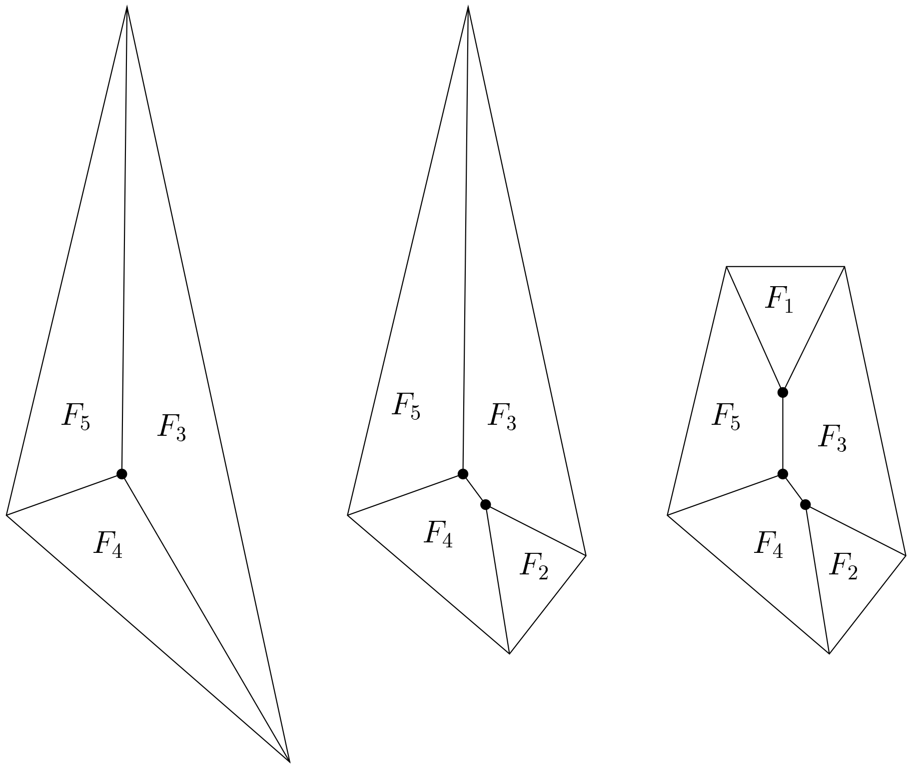
**Lemma 11** *Every vertex of a polytope has a lexicographically positive basis.*

*Proof.* Let $P$ be a polytope. Let $v$ be an arbitrary vertex of $P$. Choose some objective function so that $v$ is the unique optimum. Choose an initial lex positive basis. Run the simplex method with lexicographic pivoting. Since there are only a finite number of bases, and by Proposition 4 lexicographic pivoting does not repeat a basis, we must eventually reach some basis of $v$. Since lexicographic pivoting maintains a lex positive basis at every step, this basis must be lex positive. □

**Figure 8** Lexicographic perturbation and incremental construction




(a) Sequentially perturbing the halfspaces defining a vertex




(b) Intersecting the perturbed halfspaces in reverse order

Algorithm 6 gives a schematic version of the lexicographic reverse search algorithm. We rename variables so that $C_{\text{opt}} = \{m - d + 1 \ldots m\}$ is a cobasis of the initial vertex $v_0$. The routine PivotToOpt does a lexicographic pivot towards this cobasis with the objective function $\omega = [\mathbb{O}^{m-d}, -\mathbb{1}^d]$. If there is more than one cobasic variable with positive cost coefficient, then we choose the one with lowest index. PivotToOpt returns not only the new cobasis, but the index of the column of the basis that entered. The test IsPivot$(C', C)$ determines whether $(C, k) = \text{PivotToOpt}(C')$ for some $k$.

**Algorithm 6** ReverseSearch$(H, v_0)$

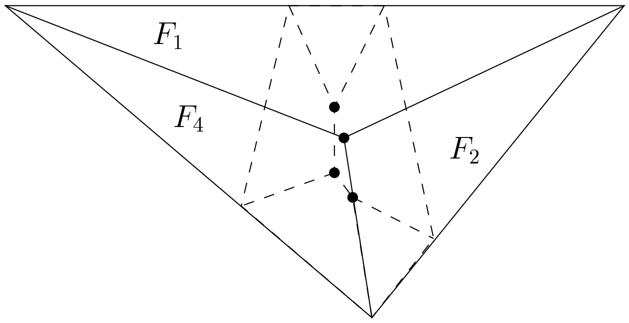
```
C ← C_opt, j ← 1, AddToDictionary(H_0, H_cur)
repeat
    while j ≤ d
        C′ ← ComputeNeighbour(C, j, H_cur)
        if IsPivot(C′, C) then
            C ← C′, j ← 1                          down edge
        else
            j ← j + 1                              next sibling
        end if
    end while
    (C, j) ← PivotToOpt(C)                         up edge
    j ← j + 1.
until j > d and C = C_opt
```

As before, we could use Algorithm 6 to implement a verification (FindWitness) step by performing a vertex enumeration from scratch. In the next section, we discuss how to construct an algorithm analogous to Algorithm 5 that performs only a single vertex enumeration, but which uses reverse search instead of a standard depth first search.

### 4.2 Primal–Dual Reverse Search

In this section we give a modification of Algorithm 6 that computes the facet defining halfspaces as a side effect. Define pdReverseSearch$(H_0, v_0)$ as Algorithm 6 with the call to ComputeNeighbour replaced with a call to ComputeNeighbour2. As in Section 3, we suppose that preprocessing steps have given us an initial set of facet defining halfspaces $H_0$ such that $\mathcal{P}(H_0)$ is bounded and there is some $v_0$ that is a vertex of the input and of $\mathcal{P}(H_0)$. It will turn out that the numbering of halfspaces is crucial. We number the $j$th halfspace discovered (including preprocessing) as $m - j$ (of course, we do not know what $m$ is until the algorithm completes, but this does not prevent us from ordering indices). This reverse ordering corresponds to pushing later discovered hyperplanes out farther, thus leaving the skeleton of earlier discovered vertices and edges undisturbed; compare Figure 8(b) where halfspaces are numbered as in pdReverseSearch, with Figure 9, where a different insertion order causes intermediate vertices to be cut off.

**Figure 9** A perturbed vertex cut off by later halfspaces.




The modified algorithm pdReverseSearch can be considered as a simulation of a "restricted" reverse search algorithm for vertex enumeration where we are given access only to a subset of the halfspaces, and where the "input halfspaces" are labeled in a special way. Since the lexicographic reverse search described in the previous section works for each labeling of the halfspaces, to show that the restricted reverse search correctly enumerates the vertices of $P$, we need only show that it visits the same set of cobases as the unrestricted algorithm would, if given the same labeling and initial cobasis.

Let $Ax = b$, $A \in \mathbb{R}^{(m-d)\times m}$ be the slack representation of a $d$-polytope. We can write the slack representation in homogeneous form $S = \begin{bmatrix} I & A' & -b \end{bmatrix}$ where $A' \in \mathbb{R}^{(m-d)\times d}$. Suppose at some step of the restricted reverse search the lowest index halfspace visited (including initialization) is $k+1$. The restricted reverse search therefore has access to all of $S$ except for the first $k-1$ rows and the first $k-1$ columns.

Let $K$ denote $\{\, k \ldots m \,\}$ for some $k \leq m-d$. For any cobasis $C \subset K$, let $\widehat{B}$ denote $K \setminus C$. We define the *$k$-restricted basis matrix* for $C$ as the last $m-d-k+1$ rows of $A_{\widehat{B}}$. Let $R$ denote the $k$-restricted basis matrix for $C$, and let $\rho$ denote $R^{-1}b_K$. By the *$k$-restricted lexicographic ratio test* we mean the lexicographic ratio test applied to the matrix $[\rho\ R^{-1}]$. By way of contrast we use the *unrestricted* lexicographic ratio test or basis matrix to mean the previously defined lexicographic ratio test or basis matrix.

We observe that the restricted basis matrix is a submatrix of the unrestricted basis matrix for a given cobasis, and that this property is preserved by matrix inversion. Let $R$ denote the $k$-restricted basis for $C$. Let $U$ denote the (unrestricted) basis matrix for $C$. Since $k \leq m-d$, and the first $m-d$ columns of the slack representation form an identity matrix, we know columns of $U$ before $k$ must be identity columns. It follows that

$$U = \begin{bmatrix} I & M \\ \mathbb{O} & R \end{bmatrix}$$

for some matrix $M$. The reader can verify the following matrix identity.

$$U^{-1} = \begin{bmatrix} I & M \\ \mathbb{O} & R \end{bmatrix}^{-1} = \begin{bmatrix} I & -MR^{-1} \\ \mathbb{O} & R^{-1} \end{bmatrix}. \tag{5}$$

The edges of the reverse search tree are pivots. Referring to our interpretation of lex pivoting as a perturbation, in order that both versions of the reverse search generate the same perturbed vertex/edge tree, they must generate the same set of pivots. We argue first that choosing the same hyperplane to leave the cobasis (i.e. edge to leave the perturbed vertex), yields the same hyperplane to enter the cobasis in both cases (i.e. the same perturbed vertex).

**Lemma 12** *Let $P$ be a $d$-polytope and let $Ax = b$ be the slack representation of $P$. Let $C \subset \{\, k \ldots m \,\}$ be a cobasis for $Ax = b$. For $k \leq m - d$ and for any entering variable $x_s$, if there is a candidate leaving variable $x_t$ with $t \geq k$ then the leaving variable chosen by the lexicographic ratio test is identical to that chosen by the $k$-restricted lexicographic ratio test.*

*Proof.* Let $\beta$ denote $U^{-1}b$. As above, let $\rho$ denote $R^{-1}b_K$. One consequence of (5) is that $\rho = \beta_K$. If there is exactly one candidate leaving variable, then by the assumptions of the lemma it must have index at least $k$, and both ratio tests will find the same minimum. If on the other hand there is a tie in the minimum ratio test applied to $\beta$ then a variable with index at least $k$ will always be preferred by the unrestricted lexicographic ratio test, since in the columns of $U^{-1}$ with index less than $k$, these variables will have ratio 0. □

The up (backtracking) edges in the reverse search tree consist of of lex pivots where the lowest indexed cobasic variable with positive cost coefficient is chosen as the entering variable (i.e. lowest indexed tight constraint that can profitably be loosened). The previous lemma tells us that for a fixed entering variable and cobasis, the restricted and unrestricted reverse search will choose the same leaving variable. It remains to show that in a backtracking pivot towards the optimum cobasis they will choose the same entering variable. Given a fixed set of halfspaces $\{\, h_1^+, h_2^+, \ldots, h_d^+ \,\}$ (a cobasis) and a fixed vector $\omega^*$ (direction of optimization), the signs of the cost vector depend only on the signs of $\omega^* h_i$, $1 \leq i \leq d$. We can in fact show something slightly stronger, since our objective vector $\omega$ (with respect the slack representation) does not involve hyperplanes with index less than $k$. As above, let $K = \{\, k \ldots m \,\}$ and $\widehat{B} = K \setminus C$. Analogous to the definition of a $k$-restricted basis matrix, we define the *$k$-restricted cost row* for cobasis $C$ as $\omega_C - \omega_{\widehat{B}} R^{-1} \widehat{A}_C$ where $R$ is the $k$-restricted basis matrix and $\widehat{A}_C$ is the last $m - d - k + 1$ rows of $A_C$.

**Lemma 13** *For objective vector $\omega = \left[\mathbb{O}^{m-d}, \omega'\right]$, for $k \leq m - d$, the cost row and the $k$-restricted cost row are identical.*

*Proof.* As before, let $R$ and $U$ be the restricted and unrestricted basis matrices respectively. From the form of the objective vector, we know $\omega_B = \left[\mathbb{O}^{k-1}, \omega_{\widehat{B}}\right]$. By (5),

$$
\begin{aligned}
\omega_B U^{-1} A_C &= \begin{bmatrix} \mathbb{O}^{k-1} & \omega_{\widehat{B}} \end{bmatrix} \begin{bmatrix} I & -MR^{-1} \\ \mathbb{O} & R^{-1} \end{bmatrix} \begin{bmatrix} A' \\ \widehat{A}_C \end{bmatrix} \\
&= \omega_{\widehat{B}} R^{-1} \widehat{A}_C
\end{aligned}
$$

□

In the case of down edges in the reverse search tree, each possible entering variable (hyperplane to leave the cobasis) is tested in turn, in order of increasing index. Thus if the previous backtracking pivot to a cobasis was identical in the two algorithms, the next down edge will be also. Reverse search is just depth first search on a particular spanning tree;

hence it visits the nodes of the tree in a sequence (with repetition) defined by the ordering of edges. The ordering of edges at any node in the reverse search tree is in turn determined by the numbering of hyperplanes.

**Lemma 14** *Let $P$ be a polytope. Let $H_0$ be a subset of $\mathcal{H}(P)$ with bounded intersection. Let $v_0 \in \mathcal{V}(H_0) \cap \mathcal{V}(P)$. The set of cobases visited by* pdReverseSearch$(H_0, v_0)$ *is the same as that visited by* ReverseSearch$(\mathcal{H}(P), v_0)$ *if* ReverseSearch *is given the same halfspace numbering.*

*Proof.* We can think of the sequences of cobases as chains connected by pivots. Let $\mathcal{C}_r = \langle C_1, C_2, \dots \rangle$ be the chain of cobases visited by pdReverseSearch$(H_0, v_0)$. Let $\mathcal{C}_u$ be the chain of cobases visited by ReverseSearch$(\mathcal{H}(P), v_0)$. Both sequences start at the same cobasis since the starting cobasis is the one with the lex maximum set of indices. Now suppose the two sequences are identical up to element $j$; further suppose that $\bigcup_{i \leq j} C_i = k+1 \dots m$. There are two cases. If the edge in $\mathcal{C}_r$ from $C_j$ to $C_{j+1}$ is a reverse (down) edge, then we start by pivoting from $C_j$ to $C_{j+1}$ by fixing some entering variable and choosing the leaving variable lexicographically. $C_{j+1}$ contains at most one variable not present in $C_i$, $i \leq j$; this variable is numbered $k$, if present. Let $s$ denote the position of the entering variable in $C_j$ (i.e. the column to leave the cobasis matrix). Since the cobasis in position $C_j$ will have occurred $s-1$ times in both sequences, we know that ReverseSearch and pdReverseSearch will choose the same entering variable. By Lemma 12, they will choose the same leaving variable. The test IsPivot$(C_{j+1}, C_j)$ depends only on the cost row, so by Lemma 13 the next cobasis in $\mathcal{C}_u$ will also be $C_{j+1}$. Suppose on the other hand the pivot from $C_j$ to $C_{j+1}$ is a forward pivot. We know from Lemma 13 that both invocations will choose the same entering variable, and we again apply Lemma 12 to see that they will choose the same leaving variable. □

**Theorem 4** *Given $V \in \mathbb{R}^{n \times d}$, let $m$ denote $|\mathcal{H}(V)|$, and let $\phi$ denote the number of lexicographically positive bases of $\mathcal{H}(V)$.*

(a) *we can compute $\mathcal{H}(V)$ in time $O(\phi m d^2)$ and space $O((m+n)d)$.*

(b) *we can decide if* $\operatorname{conv} V$ *is simple in time $O(n^2 d + n d^3)$.*

**Proof**

(a) Total cost for finding an initial set of halfspaces is $O(nkd^2)$, where $k$ is the size of the initial set. Since every DeleteVertex call finds a new halfspace, the total cost for these calls is $O(nmd^2)$. In every call to ComputeNeighbour2, each pivot except the last one discovers a new halfspace. Those which discover new halfspaces have total cost $O(m^2 d)$ which is $O(\phi m d)$; the other pivots cost $O(\phi m d^2)$ as there are $\phi d$ calls to ComputeNeighbour2. The $\phi$ forward pivots (PivotToOpt) cost $O(\phi m d)$.

(b) At any step of the reverse search, we can read the number of halfspaces satisfied with equality by the current vertex off the dictionary in $O(m)$ time. From the Lower Bound Theorem [4] for simple polytopes, if $P$ is simple then $m \leq 2(n/d + d)$ for $d \geq 2$. If we reach a degenerate vertex, or discover more than $2(n/d+d)$ facets, we stop. If the reverse search terminates, then in $O(nmd)$ time we can compute the number of facets meeting at each vertex. The total cost is thus $O(n(n/d+d)d^2) = O(n^2d + nd^3)$. □

Theorem 4b is of independent interest since the problem of given $H$, deciding whether $\mathcal{P}(H)$ is simple is known to be NP-complete in the strong sense [11].

# 5 Experimental Results

In order to test whether primal–dual reverse search is of practical value, we have implemented it and compared its performance to Avis' implementation of reverse search [1]. Both programs are written in C and use rational arithmetic, which allows for a fair comparison. We present experiments with two families of polytopes: (1) certain simple polytopes which show the best and the worst behavior of both programs and (2) products of cyclic polytopes which are degenerate for both programs.

The memory requirements of our implementation are twice the input size plus twice the output size, as the program stores four dictionaries: a constant vertex dictionary $V$ and a growing halfspace dictionary $H_{\text{cur}}$ in unpivoted form, and a working copy of both. The program uses an earlier version of the preprocessing step, with an upper bound of $O(nd^4)$, compared to the current bound of $O(nd^3)$. The source code is available at `http://wwwjn.inf.ethz.ch/ambros/pd.html`. In what follows, **pd** is our implementation of primal–dual reverse search and **lrs** is Avis' implementation of reverse search. All of the experiments have been performed on a Digital AlphaServer 4/233 with 256M of real memory and 512M of virtual memory.

Figure 10 compares the running time of the two programs on products of two simplices. These $2d$-dimensional polytopes have $2d+2$ facets and $(d+1)^2$ vertices. They are simple (which is ideal for vertex enumeration by **lrs** and facet enumeration by **pd**), but have extremely high triangulation complexity [2], which is bad for vertex enumeration by **pd** and facet enumeration by **lrs**, because the perturbation of the vertices made by the algorithms induces a triangulation of the polytope's boundary. On the plot, we show the times for enumerating both the facets and the vertices. As expected, **pd** is clearly superior to **lrs** for facet enumeration of these polytopes. Their very few facets are all found by the preprocessing of our current implementation; in fact this accounts for most of the time taken by **pd** on these examples.

A less asymmetric example is the product of cyclic polytopes $C_k(n)\times C_k(n)\times\cdots\times C_k(n)$. These polytopes are neither simple nor simplicial. Moreover, it is known [2] that both their primal and their dual triangulations are superpolynomial; nonetheless experimentally it seems that the dual triangulations are smaller than the primal ones. This is advantageous for **pd**, meaning that the perturbation made by **pd** for facet enumeration produces less bases than the one made by **lrs**. This difference is reflected in the relative performance of the two programs. The relation between the primal and dual triangulation sizes (number of bases computed by either algorithm) of $C_4(n)\times C_4(n)$ (eight-dimensional polytopes with $n^2$ vertices and $O(n^2)$ facets) shown in Figure 11 is similar to the relation of the running times shown in Figure 12.

**Figure 10** Running time for products of simplices $T_d \times T_d$

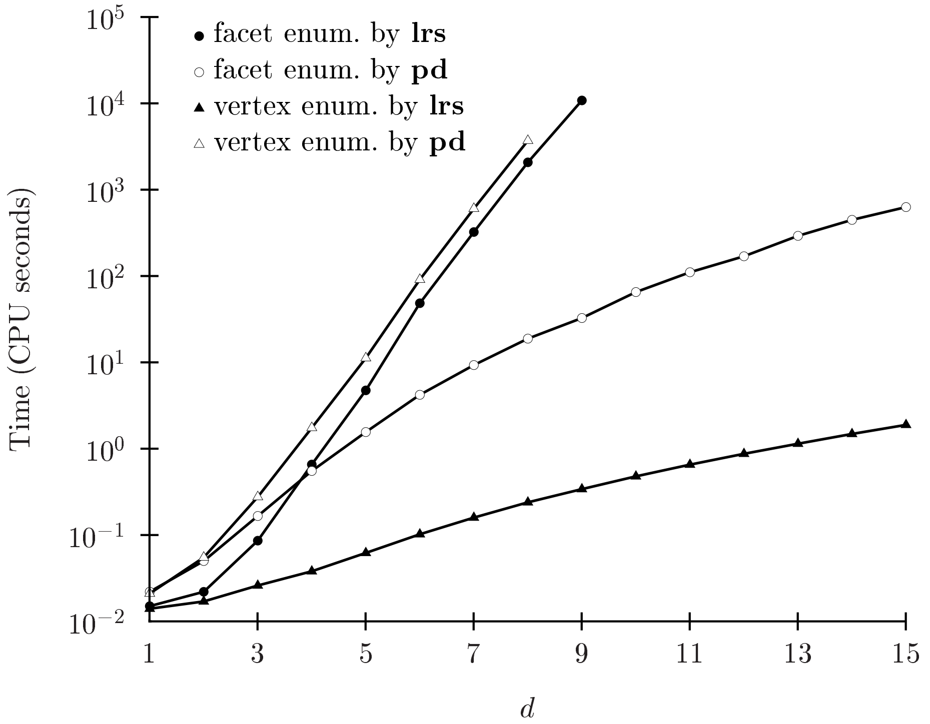



**Figure 11** Triangulation size (number of bases computed) of $C_4(n) \times C_4(n)$.

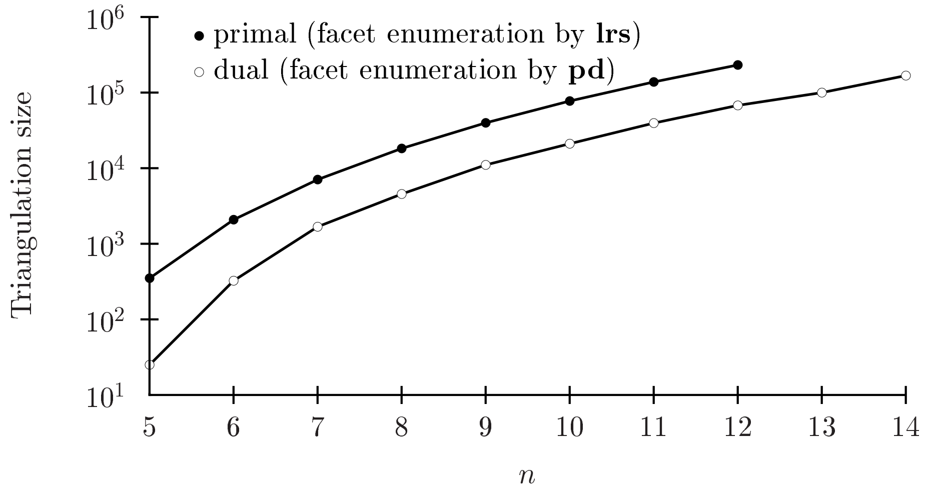
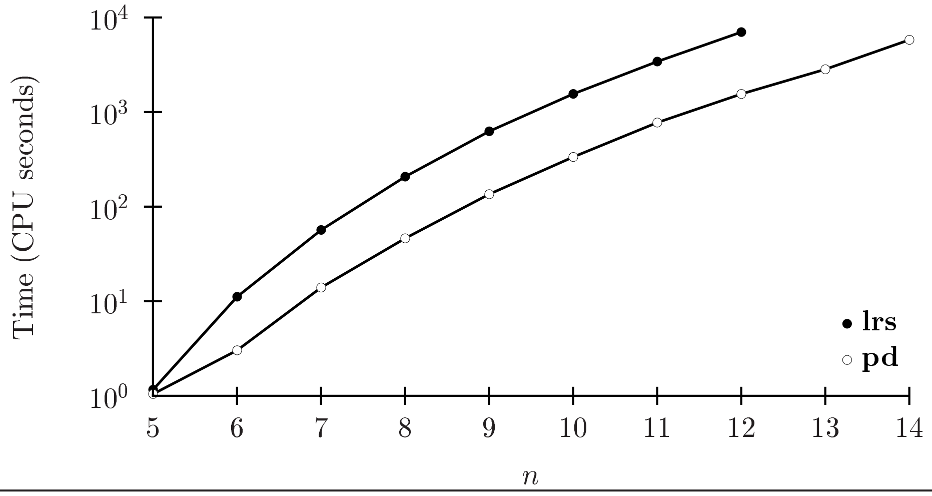


**Figure 12** CPU time for facet enumeration of $C_4(n) \times C_4(n)$.




## 6 Conclusions

An alternative approach to achieving an algorithm polynomial for the dual-nondegenerate case is to modify the method of Gritzmann and Klee [12]. An idea due to Clarkson [8] can be used to reduce the row size of each of these linear programs to $O(m')$ where $m'$ is the maximum number of facets meeting at a vertex. If we assume that $m' \leq d + \delta$ for some constant $\delta$ then we can solve each linear program by brute force in time polynomial in $d$. It seems that such an approach will be inherently quadratic in the input size since the entire set of input halfspaces is condered to enumerate the vertices of each facet.

It would be interesting to remove the requirement in Theorem 1 that the family be facet-hereditary, but it seems difficult to prove things in general about the polytopes formed by subsets of the halfspace description of a known polytope.

## Acknowledgments

The authors would like to thank David Avis for useful discussions on this topic, and for writing **lrs**. We would also like to thank an anonymous referee for a careful reading of this paper and several helpful suggestions.

## References

[1] D. Avis. A C implementation of the reverse search vertex enumeration algorithm. Technical Report RIMS Kokyuroku 872, Kyoto University, May 1994. (Revised version of: Technical Report SOCS-92.12, McGill University, School of Computer Science).

[2] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Comput. Geom.: Theory and Appl.*, 7(5–6):265–301, Apr. 1997.

[3] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Disc. Comput. Geom.*, 8:295–313, 1992.

[4] D. W. Barnette. The minimum number of vertices of a simple polytope. *Israel J. Math.*, 8:304–308, 1971.

[5] A. Brøndsted. *Introduction to Convex Polytopes.* Springer-Verlag, 1981.

[6] D. Chand and S. Kapur. An algorithm for convex polytopes. *J. ACM*, 17:78–86, 1970.

[7] V. Chvátal. *Linear Programming.* W. H. Freeman, New York, NY, 1983.

[8] K. L. Clarkson. More output-sensitive geometric algorithms. In *Proc. 35th IEEE Symp. Found. Comp. Sci.*, pages 695–702, 1994.

[9] M. Dyer. The complexity of vertex enumeration methods. *Math. Oper. Res.*, 8(3):381–402, 1983.

[10] J. Edmonds. Decomposition using Minkowski. Abstracts of the 14th International Symposium on Mathematical Programming, Amsterdam, 1991.

[11] K. Fukuda, T. M. Liebling, and F. Margot. Analysis of backtrack algorithms for listing all vertices and all faces of a convex polyhedron. *Comput. Geom.: Theory and Appl.*, 8:1–12, 1997.

[12] P. Gritzmann and V. Klee. On the complexity of some basic problems in computational convexity: II. Volume and mixed volumes. In T. Bisztriczky, P. McMullen, R. Schneider, and A. I. Weiss, editors, *Polytopes: Abstract, Convex, and Computational*, number 440 in NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci., pages 373–466. Kluwer Academic, Dordrecht, 1994.

[13] J. P. Ignizio and T. M. Cavalier. *Linear Programming*, pages 118–122. Prentice Hall International Series in Industrial and Systems Engineering. Prentice Hall, 1994.

[14] V. Klee. Extreme points of convex sets without completeness of the scalar field. *Mathematika*, 11:59–63, 1964.

[15] K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, volume 3 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Heidelberg, West Germany, 1984.

[16] K. Murty. The gravitational method for linear programming. *Opsearch*, 23:206–214, 1986.

[17] R. Seidel. *Output-size sensitive algorithms for constructive problems in computational geometry.* Ph.D. thesis, Dept. Comput. Sci., Cornell Univ., Ithaca, NY, 1986. Technical Report TR 86-784.

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