ELSEVIER

Computational Geometry 11 (1998) 103–109

Computational Geometry
Theory and Applications

The vertex set of a 0/1-polytope is strongly $\mathcal{P}$-enumerable

Michael R. Bussieck, Marco E. Lübbecke *

Department of Mathematical Optimization, Braunschweig University of Technology, Pockelsstraße 14, D-38106 Braunschweig, Germany

Communicated by D. Avis; submitted 24 September 1997; accepted 28 February 1998

Abstract

In this paper, we discuss the computational complexity of the following enumeration problem: given a rational convex polyhedron P defined by a system of linear inequalities, output each vertex of P. It is still an open question whether there exists an algorithm for listing all vertices in running time polynomial in the input size and the output size. Informally speaking, a *linear* running time in the output size leads to the notion of *$\mathcal{P}$-enumerability* introduced by Valiant (1979). The concept of *strong $\mathcal{P}$-enumerability* additionally requires an output independent space complexity of the respective algorithm. We give such an algorithm for polytopes all of whose vertices are among the vertices of a polytope combinatorially equivalent to the hypercube. As a very important special case, this class of polytopes contains all 0/1-polytopes. Our implementation based on the commercial LP solver CPLEX [1] is superior to general vertex enumeration algorithms. We give an example how simplifications of our algorithm lead to efficient enumeration of combinatorial objects.

Keywords: Vertex enumeration; Convex polyhedra; 0/1-polytopes; Strong $\mathcal{P}$-enumerability; Computational complexity

1. Preliminaries

In this paper, we discuss the computational complexity of the following problem which is known as the *vertex enumeration* problem: given a rational convex polyhedron $P = \{x \in \mathbb{Q}^n \mid Ax \leqslant b\}$ defined by a system of m linear inequalities, compute the vertex set of P.

The framework for the computational complexity analysis of *counting* problems dates back to the late seventies when it was formally introduced by Valiant [10]. In a recently electronically published classification of *enumeration* problems Fukuda [4] improves on these basic concepts in order to take into account the explicit *generation* of objects. We emphasize the distinction between counting and enumeration because we do not ask for the bare number of vertices but for *listing* each particular vertex.

* Corresponding author. E-mail: M.Luebbecke@tu-bs.de.

[1] CPLEX is a registered trademark of ILOG, Inc.

0925-7721/98/$ – see front matter
PII: S0925-7721(98)00021-2

Following the lines of [4,10], we will now state fundamental definitions needed in the paper. Denote by Σ an alphabet, say $\{0, 1\}$, and by Σ^* the set of all strings over Σ, i.e., Σ^* consists of all finite sequences of zeros and ones, and the empty string. The notation $|x|$ is used for the size of x if x is a set, and its length if x is a string (size and cardinality of a set are conceptually distinct, but can be thought of as the same in this context). Formally, a *relation* R is a Boolean function $R: \Sigma^* \times \Sigma^* \to \{0, 1\}$. An instance of the *enumeration problem associated with a relation* R is to list all members of the *enumeration set* $\{y \mid R(x, y) = 1\}$, for a given x, assuming that this set is finite.

In general, the size of $\{y \mid R(x, y) = 1\}$ is exponential in $|x|$ so it is reasonable to take into account both, the input size *and* the output size, when talking about computational complexity issues in the context of enumeration problems. An algorithm for solving the enumeration problem associated with a relation R is called *polynomial* (respectively *linear*) if and only if its running time is polynomial in $|x|$ and polynomial (respectively linear) in $|\{y \mid R(x, y) = 1\}|$ for all x. Obviously, one cannot do better than linear in the output size. We will use the notion of *$\mathcal{P}$-enumerability of a relation* R, respectively of the corresponding enumeration sets, if there exists a linear algorithm which solves the enumeration problem associated with R. Fukuda et al. [5] strengthened this concept in order to consider the by no means trivial requirement that already output elements of an enumeration set must not be kept in memory. A *compact* algorithm is one which solves the enumeration problem associated with R for all x in space complexity polynomial in $|x|$ and the largest size $|y|$ of output objects y.

Definition 1. A relation R, respectively the enumeration set for a given x, is called *strongly $\mathcal{P}$-enumerable* if there exists a linear, compact algorithm which solves the enumeration problem associated with R.

As in [5], we define a relation that embeds the vertex enumeration problem in the more general context. Given a system $Ax \leqslant b$ of m linear inequalities, we denote by $a_i x \leqslant b_i$ the ith row. For the reason of convenience we identify the ith inequality with its index i. We say that a subset $V \subseteq \{1, \ldots, m\}$ of inequalities is *vertex defining* for the polyhedron $P = \{x \in \mathbb{Q}^n \mid Ax \leqslant b\}$ if and only if the set

$$\{x \in P \mid a_i x = b_i \text{ for all } i \in V,\ a_i x < b_i \text{ for all } i \notin V\}$$

consists of a single point.

The *vertex relation* $R_{\mathcal{V}}$ is defined by $R_{\mathcal{V}}([A, b], V) = 1$ if and only if V is vertex defining for the associated polyhedron P. The vertex enumeration problem is the enumeration problem associated with $R_{\mathcal{V}}$. We remark that it is not known whether there exists a polynomial algorithm for this enumeration problem in the general case, let alone whether $R_{\mathcal{V}}$ is (strongly) $\mathcal{P}$-enumerable. The first $\mathcal{P}$-enumerability result is by Valiant [10] who described a general algorithm which was recently adapted to the vertex enumeration problem by Fukuda et al. [5]. Unfortunately, their backtracking approach needs an $\mathcal{NP}$-complete decision problem to be solved for each output. Provan [8] gives algorithms to list the vertices of polyhedra associated with network linear programs and their duals. Each of his algorithms has running time which is quadratic in the output, and does not require that the polyhedron be either simple or bounded. A major result is the so-called *reverse search technique* by Avis and Fukuda [2] which constitutes a constructive proof for the strong $\mathcal{P}$-enumerability of $R_{\mathcal{V}}$ when P is simple.

2. Vertex enumeration for a special class of polytopes

In this section we describe a property of the input that enables us to prove strong $\mathcal{P}$-enumerability of the associated vertex relation. We denote by $\mathbf{P}^{\square}$ the class of polytopes which are *combinatorially equivalent* to a hypercube of appropriate dimension. In other words, there is a bijection between the faces of a hypercube and $P^{\square} \in \mathbf{P}^{\square}$ that preserves the inclusion relation, or, loosely speaking, each $P^{\square}$ is a *squashed cube*. We subsume in a class Π of polytopes all polytopes that can be generated using the following procedure: given an arbitrary $P^{\square} \in \mathbf{P}^{\square}$, take the *convex hull* of an arbitrary subset of the vertices of $P^{\square}$. An example construction in dimension three is depicted in Fig. 1. Similarly to the linear description $\{x \in \mathbb{Q}^n \mid 0 \leqslant x \leqslant 1\}$ of a hypercube we can describe every $P^{\square} \in \mathbf{P}^{\square}$ as

$$\{x \in \mathbb{Q}^n \mid Lx \leqslant l,\ Ux \leqslant u\}, \quad \text{where } L, U \in \mathbb{Q}^{n \times n},$$

and each vertex of $P^{\square}$ is uniquely determined by an appropriate choice of exactly n inequalities satisfied with equality. Note that this requirement implies the assumption that $P^{\square}$ has dimension n. Geometrically, we have n pairs of inequalities that represent *opposite* facets of $P^{\square}$. For the sake of simplicity, let us assume without loss of generality that two rows with common index form such a pair. Algebraically, every *partition* $\mathcal{L} \cup \mathcal{U}$ of $\{1, \ldots, n\}$ defines a vertex v of $P^{\square}$ via $L_{\mathcal{L}} v = l_{\mathcal{L}}$ and $U_{\mathcal{U}} v = u_{\mathcal{U}}$, i.e.,

$$\operatorname{rank}\begin{pmatrix} L_{\mathcal{L}} \\ U_{\mathcal{U}} \end{pmatrix} = n, \tag{1}$$

where $L_{\mathcal{L}}$ denotes the rows of L indexed by $\mathcal{L}$. Considering a polytope $P = \{x \in \mathbb{Q}^n \mid Ax \leqslant b\}$ in class Π we can always, at least theoretically, find a *boundary* $P^{\square} \in \mathbf{P}^{\square}$ and add the corresponding linear description to that of P. This *redundant* input becomes necessary because our algorithm relies on the description of $P^{\square}$ which we are not able to specify efficiently in advance. By construction, the *location* of each vertex of P is already determined by $Lx \leqslant l$, $Ux \leqslant u$, whereas its *feasibility* is solely defined by $Ax \leqslant b$. To put it more formally, we require that

$$R_{\mathcal{V}}\left(\begin{bmatrix} A & b \\ L & l \\ U & u \end{bmatrix}, V\right) = 1 \Rightarrow R_{\mathcal{V}}\left(\begin{bmatrix} L & l \\ U & u \end{bmatrix}, V \cap \{m+1, \ldots, m+2n\}\right) = 1 \tag{2}$$

for all $V \subseteq \{1, \ldots, m+2n\}$. This is the key property we will exploit in our algorithm. Given a linear description of both, P and a corresponding $P^{\square} \in \mathbf{P}^{\square}$, the pseudo code reads as follows.

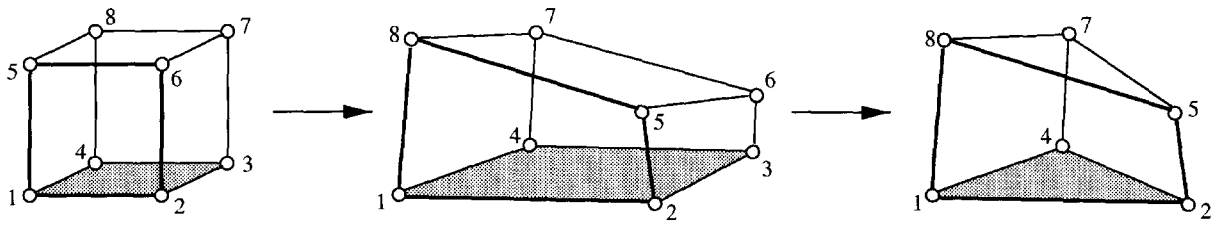

Fig. 1. From a hypercube via a polytope in class $\mathbf{P}^{\square}$ to a polytope in class Π.

Algorithm 1 (Vertex enumeration for polytopes in class Π)
begin enum($\mathcal{L}, \mathcal{U}, j$)
  /* initial call with $\mathcal{L} \equiv \mathcal{U} \equiv \emptyset$, $j = 0$ */
  if $\{x \in \mathbb{Q}^n \mid Ax \leqslant b, L_{\mathcal{L}}x = l_{\mathcal{L}}, U_{\mathcal{U}}x = u_{\mathcal{U}}\} \neq \emptyset$ **then**
    if $j = n$ **then**
      output the vertex defined by $\mathcal{L}$ and $\mathcal{U}$;
    else /* $j < n$ */
      call enum($\mathcal{L} \cup \{j+1\}, \mathcal{U}, j+1$);
      call enum($\mathcal{L}, \mathcal{U} \cup \{j+1\}, j+1$);
    endif
  endif
end.

Theorem 2 (Correctness). *Given the linear description $\{x \in \mathbb{Q}^n \mid Ax \leqslant b\}$ of a polytope $P \in \Pi$ and the linear description $\{x \in \mathbb{Q}^n \mid Lx \leqslant l, Ux \leqslant u\}$ of an appropriate $P^{\square} \in \mathbf{P}^{\square}$ Algorithm 1 called with $\mathcal{L} \equiv \mathcal{U} \equiv \emptyset$ outputs each vertex of P exactly once.*

Proof. Observe the following *hereditary* property of polytopes $P \in \Pi$: the intersection of P with a facet of a corresponding $P^{\square}$ gives again a polytope $F \in \Pi$. Since F is bounded, checking the existence of a vertex v_F of F is equivalent to checking the existence of a *feasible* point in F which can be done by solving a linear program. By construction, v_F is a vertex of P itself.

At the very depth of the recursion, when $j = n$, the sets $\mathcal{L}$ and $\mathcal{U}$ clearly form a partition of $\{1, \ldots, n\}$. It follows by (1) and $Av \leqslant b$ that $L_{\mathcal{L}}v = l_{\mathcal{L}}$, $U_{\mathcal{U}}v = u_{\mathcal{U}}$ defines a vertex v of P. Since all feasible partitions are constructed in a *binary tree* manner, each partition, and consequently each vertex, is reached only once.

On the other hand, let v be a vertex of P. Algorithm 1 obtains the corresponding partition $\mathcal{L}, \mathcal{U}$ since for all $\mathcal{L}' \subseteq \mathcal{L}, \mathcal{U}' \subseteq \mathcal{U}$ the set $\{x \in \mathbb{Q}^n \mid Ax \leqslant b, L_{\mathcal{L}'}x = l_{\mathcal{L}'}, U_{\mathcal{U}'}x = u_{\mathcal{U}'}\}$ is nonempty. □

Theorem 3 (Complexity). *Algorithm 1 has running time $O(v \cdot n \cdot T_{LP})$, where v denotes the cardinality of the output set $\{x \mid R_{\mathcal{V}}(\cdot, x) = 1\}$ and T_{LP} denotes the time complexity of performing a feasibility check. Apart from some indices Algorithm 1 only needs to store the linear description of P and $P^{\square}$ which is $O((m+n)n)$.*

Proof. As stated above, each vertex is reached exactly once, so the running time of Algorithm 1 is linear in v. Since the maximal depth of recursion is n, for each output we need to solve at most n linear programs of size $[A|L|U, b|l|u]$ in binary representation for the feasibility tests. No additional memory apart from the input size is needed. □

Considering polynomial time algorithms for linear programming (cf. [9]) our central result now follows immediately from Theorems 2 and 3.

Corollary 4. *Given the linear description $\{x \in \mathbb{Q}^n \mid Ax \leqslant b\}$ of a polytope $P \in \Pi$ and the linear description $\{x \in \mathbb{Q}^n \mid Lx \leqslant l, Ux \leqslant u\}$ of an appropriate $P^{\square} \in \mathbf{P}^{\square}$ the vertex relation $R_{\mathcal{V}}([A|L|U, b|l|u], \cdot)$ is strongly $\mathcal{P}$-enumerable.*

This corollary does not only generalize former special case $\mathcal{P}$-enumerability results, e.g., for the bipartite matching polytope [6] or certain 0/1-polytopes described by equalities [7] (a 0/1-polytope is one, all of whose vertices have coordinates zero or one). It also applies to the huge class of polytopes associated with combinatorial optimization problems. Taking the unit hypercube $\{x \in \mathbb{Q}^n \mid 0 \leqslant x \leqslant 1\}$ as boundary $P^{\square}$, whose linear description is of size polynomial in the size of $[A, b]$, we immediately have a special case result which is of interest by itself.

Corollary 5. *The vertex relation $R_{\mathcal{V}}([A, b], \cdot)$ is strongly $\mathcal{P}$-enumerable for all linear descriptions $[A, b]$ defining an arbitrary 0/1-polytope.*

Note that in this case we completely circumvent all trouble with artificially enlarged, and hence redundant input. We emphasize that these statements do *not* measure the complexity of combinatorial optimization or enumeration problems, since the input size of such problems is totally different from that discussed in this paper. To set an example, consider the problem #HAM of listing all Hamiltonian cycles in a graph G with n nodes and m edges. Even if we *had* the linear description of the corresponding *traveling salesperson polytope*, which is a 0/1-polytope, Corollary 5 would have said *nothing* about an efficient solution to #HAM. Not only that the size of this linear description is exponential in n and m, even worse, we know from computational complexity theory that it is most unlikely that one ever finds a complete such linear description.

An implementation of Algorithm 1 is straight forward, especially when a fast LP code is available. Our special case program `zerone` that dumps the vertex set of a 0/1-polytope is based on the commercial LP solver CPLEX. It is publicly available via anonymous ftp. [2]

We prefer not to give comparative computational results because a comparison with more general algorithms such as the *double description method* or the *reverse search technique* is obviously unfair. On an HP9000/735-125 workstation we listed, e.g., the 1,048,580 vertices of the unit hypercube of dimension 20 in less than twenty minutes and the 5,040 vertices of the bipartite matching polytope for a graph with two times seven nodes in about eight minutes. The benefit of our implementation clearly is its memory usage being independent from the output. Furthermore it is well suited to performing a frequent task when designing combinatorial optimization models, namely enumerating all integral vertices of polytopes P lying in a unit hypercube even when nothing is known about the 0/1-property of P.

3. Enumeration of particular combinatorial objects

Hitherto, we considered an algorithm that takes as input linear descriptions of polytopes. In this section we illustrate by example the derivation of algorithms for the enumeration of combinatorial objects that work directly on the *problem* rather than on the associated polytope.

Let $G = (N, A)$ be a connected directed graph with n nodes and m arcs. Introducing a variable x_{ij} for each arc (i, j) a *0/1 network flow* in G is a function $x : A \to \mathbb{Q}$ that satisfies *flow conservation* and *feasibility*,

$$\sum_{(i,k)} x_{ik} - \sum_{(k,j)} x_{kj} = b_k \quad \text{for all } k \in N \quad \text{and} \quad 0 \leqslant x \leqslant 1, \tag{3}$$

[2] The relevant URL is `ftp://ftp.math.tu-bs.de/pub/software/zerone.tar.gz`.

where $b_k \in \mathbb{Z}$ denotes the *demand* in node $k \in N$, see [1] for details on network flows. It is well known [9] that each feasible integral flow corresponds to a vertex of the polytope P defined by (3). Consequently, the vertex enumeration problem for P is equivalent to listing all integral 0/1 flows in G. An adaptation of Algorithm 1 to the latter problem is sketched in the following pseudo code. We label the variables x_{ij} by 0, 1 or *free*, i.e., $0 \leqslant x_{ij} \leqslant 1$, depending on the flow value on the corresponding arc already fixed to zero/one or not.

Algorithm 2 (Enumeration of integral 0/1 flows in a graph G)
begin enum(G, b, x)
  /* initial call with x labeled *free*, G is given with node demand b */
  if $A = \emptyset$ **then**
    output x;
  else if (G, b) contains an integral 0/1 flow **then**
    (i, j) := next arc labeled *free*;
    call enum(update(G, b, x, (i, j), 0));
    call enum(update(G, b, x, (i, j), 1));
  endif
end.

function update(G, b, x, (i, j), k)
/* construction of a stage dependent auxiliary graph */
begin
  label x_{ij} with k;
  $A := A \setminus \{(i, j)\}$; /* delete arc (i, j) from G */
  $b_i := b_i + k$; $b_j := b_j - k$; /* update demand */
  return (G, b, x);
end;

Corollary 6. *Given as input a graph G with n nodes and $m > 0$ arcs and node demand vector b, Algorithm 2 enumerates each integral 0/1 flow in G exactly once in time* $O(f \cdot m \cdot \min\{mn^{2/3}, m^{3/2}\})$, *where f denotes the number of output flows.*

Proof. We observe that each fixation of an x_{ij} to zero or one, and thus the construction of the resulting auxiliary graph, corresponds to the restriction of a flow feasibility constraint $0 \leqslant x_{ij}$ or $x_{ij} \leqslant 1$ in (3) to equality. Then, the correctness of the algorithm follows from Theorem 2. Finding out whether a graph contains an integral 0/1 flow can be done solving a maximum flow problem in $O(\min\{mn^{2/3}, m^{3/2}\})$ [1]. For each output we have to perform $O(m)$ such existence tests, and the assertion follows. □

4. Conclusions and open problems

In this paper we presented an algorithm for solving the *vertex enumeration* problem for polytopes P with the property that all vertices are among the vertices of a polytope that is combinatorially equivalent to a hypercube. We proved that the time complexity of the algorithm is polynomial in the input size and linear in the number of output vertices while its space complexity is polynomial in the input size only. In

other words, we proved the strong $\mathcal{P}$-enumerability of the vertex set of all polytopes with the property described above.

The simplicity of our algorithm arises from the fact that we theoretically know a linear description of the convex hull $P^{\square}$ of all *possible* vertices to be listed and only need a feasibility test for each output which amounts to solving a linear program. An efficient procedure to actually construct the linear description on an appropriate $P^{\square}$ would result in strong $\mathcal{P}$-enumerability results being independent from the particular description L, U and l, u but is not known to the authors. However, for 0/1-polytopes there is no difficulty at all since we *always* have an appropriate $P^{\square}$. Beyond this, our algorithm allows simplifications that lead to efficient enumeration of combinatorial objects.

Unfortunately, we have no immediate generalization of Algorithm 1 at hand. We conjecture that a backtracking approach does not allow proving more general $\mathcal{P}$-enumerability results. Such proofs require the use of *strict* inequalities in the algorithm which seem to constitute a major difficulty, see [7] for example problems and [5] for a general result. Particularly in combinatorial optimization, it is desirable to characterize strong $\mathcal{P}$-enumerable *facet* sets. Considering the polar polyhedron P^0 of P is of no help here because the property $P \in \Pi$ generally does not imply $P^0 \in \Pi$ as one easily checks in dimension three. Recently, Bremner et al. [3] proposed a polynomial algorithm for facet enumeration on a particular family of polytopes when a polynomial algorithm for vertex enumeration for each subset of facet defining halfspaces of a polytope in the family is known. It would be interesting if their method, in combination with Algorithm 1, led to an efficient implementation for listing all facets of polytopes in class Π.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows – Theory, Algorithms and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] D. Avis, K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, Discrete Comput. Geom. 8 (1992) 295–313.

[3] D. Bremner, K. Fukuda, A. Marzetta, Primal-dual methods for vertex and facet enumeration, in: Proceedings of the 13th Annual ACM Symposium on Computational Geometry, 1997.

[4] K. Fukuda, Note on new complexity classes $\mathcal{ENP}$, $\mathcal{EP}$ and $\mathcal{CEP}$, Paper electronically available at URL http://www.ifor.math.ethz.ch/ifor/ staff/fukuda/ENP_home/ENP_note.html, June 1996.

[5] K. Fukuda, T.M. Liebling, F. Margot, Analysis of backtrack algorithms for listing all vertices and all faces of a convex polyhedron, Computational Geometry 8 (1997) 1–12.

[6] K. Fukuda, T. Matsui, Finding all the perfect matchings in bipartite graphs, Appl. Math. Lett. 7 (1994) 15–18.

[7] M.E. Lübbecke, Algorithmen zur Enumeration aller Ecken und Facetten konvexer Polyeder, Master's Thesis, Department of Mathematical Optimization, Braunschweig University of Technology, July 1996.

[8] J.S. Provan, Efficient enumeration of the vertices of polyhedra associated with network LP's, Math. Programming 63 (1994) 47–64.

[9] A. Schrijver, Theory of Linear and Integer Programming, Wiley, Chichester, 1986.

[10] L.G. Valiant, The complexity of enumeration and reliability problems, SIAM J. Comput. 8 (3) (1979) 410–421.