ELSEVIER

Information Processing Letters 73 (2000) 137–143

Information Processing Letters

www.elsevier.com/locate/ipl

Estimating the number of vertices of a polyhedron ☆

David Avis, Luc Devroye *

School of Computer Science, McGill University, 3480 University, Montréal, Québec, Canada H3A 2A7

Received 2 August 1998; received in revised form 21 July 1999
Communicated by F. Dehne

Abstract

Given a polyhedron P by a list of inequalities we develop unbiased estimates of the number of vertices and bases of P. The estimates are based on applying tree estimation methods to the reverse search technique. The time to generate an unbiased estimate is essentially bounded by the time taken to solve a linear program on P with the simplex method. Computational experience is reported. The method can be applied to estimate the output size of other enumeration problems solvable by reverse search. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Randomized algorithms; Tree estimation; Vertex estimation; Reverse search

1. Introduction

Let A be an m by n matrix with $m \geqslant n$ and let b be an n-vector. The *convex polyhedron* P is defined by

$$P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}. \tag{1.1}$$

For definitions not given here and further information, the reader is referred to Chapter 2.3, written by Bayer and Lee, of the handbook [12]. We assume throughout that A has full column rank, that P is non-empty and that there are no redundant inequalities. A bounded polyhedron is called a *polytope*. In this note we will deal with polytopes rather than possibly unbounded polyhedra. A point $\overline{x} \in P$ is a *vertex* of P if there is some n by n submatrix B of A such that $\overline{x}$ is the unique solution of $Bx = b$. The matrix B is called a *basis* for $\overline{x}$. The vertex is *degenerate* if more than n of the inequalities in (1.1) are satisfied as equations by $\overline{x}$. In other words it can be represented by more than one basis. A polytope with no degenerate vertices is called *simple*.

The well-known Upper Bound Theorem of McMullen (see [12]) states that the polyhedron P has at most

$$f(m,n) = \binom{m - \lfloor \frac{n+1}{2} \rfloor}{m-n} + \binom{m - \lfloor \frac{n+2}{2} \rfloor}{m-n}.$$

vertices. On the other hand, the Lower Bound Theorem of Barnette (see [12]) states that a non-empty simple polytope P has at least

$$g(m,n) = m(n-1) - (n+1)(n-2)$$

vertices. Non-simple polyhedra can have even fewer vertices, and general lower bounds were obtained by Deza and Fukuda [9] by inverting McMullen's condition. The bounds are tight and for most values of m and n there is a very large gap between the lower and upper bounds.

☆ Research supported by grants from N.S.E.R.C. and F.C.A.R. Appears in: *Snapshots of Computational Geometry*, Vol. 3, Technical Report SOCS-94.50, School of Computer Science, McGill University.

* Corresponding author. Email: luc@cs.mcgill.ca.

PII: S0020-0190(00)00011-9

An important computational problem is to generate all of the vertices of P. There are essentially two main approaches to this problem, both with their origins in the 1950s. The double description (or Motzkin) method [16] involves building the polytope sequentially by adding the defining inequalities one at a time. Recent algorithms of this type have been developed by Seidel, see [10], and Chazelle [7]. This method seems very powerful for degenerate polytopes but has the disadvantage that it can require a lot of memory. A practical implementation has been developed by Fukuda [11]. The second method of vertex generation involves pivoting around the edge skeleton of the polytope (for general references, see [8]). Specifically, a graph is defined on the bases of P. Two bases are adjacent if they differ in exactly one row. A search of this graph generates all bases and hence all vertices. An efficient practical method using this approach is the reverse search method of Avis and Fukuda [2]. Their approach is to compute directly a spanning tree of the basis graph. The method has time complexity depending on the number of bases of P and so is very slow for highly degenerate polytopes. The advantage of the reverse search method is that it requires only the space required to store the polytope P, and does not require the bases to be stored. A practical implementation of this method has been developed by Avis [3].

Let V and N denote respectively the number of vertices and bases of P. At present there is no known method of computing the vertices of P in time polynomial in m, n and V. The pivoting methods are, however, polynomial in m, n and N. For example, the reverse search method generates all bases in time $O(n(m-n)\min(m-n,n)N)$ and space $O(mn)$. In order to determine the feasibility of a vertex generation problem it is therefore necessary to have good estimates of N and V.

Due to the availability of the practical implementations of vertex generation algorithms various large problems have now been solved. For instance, in [6] all vertices of a non-simple polytope with $m = 729$ and $n = 8$ were found. Using the Upper Bound Theorem we can see that this polytope has at least 17 and at most 11 479 694 949 vertices. This is of little use in determining the feasibility of generating the vertices of the given polyhedron. In fact the polytope has 4862 vertices and is highly degenerate. This result was obtained by both the double description [11] and reverse search [3] methods, requiring many days of computation on a SPARC work station. The time taken by reverse search is proportional to the number of bases of the polytope. For a degenerate polytope, the number of bases is reduced by perturbation of the defining half-spaces. For this polytope, a perturbation yielded a polytope with 477 471 bases which was generated on a SPARC work station in about three weeks, using exact integer arithmetic.

The above example shows the need to be able to estimate the number of vertices and bases of a polytope, both before and after perturbation, and the need to estimate the running time of an algorithm. In this paper we give a method for estimating the number of vertices and bases of a polytope, and for estimating the running time of the reverse search method. We describe a class of randomized algorithms that give estimates whose expected value is the correct answer. Although the variance is high, methods are suggested for reducing it. The basic idea is to estimate the size of the reverse search tree estimation techniques. These are described in the next section, which also includes a discussion of how to reduce the variance. In Section 3 we discuss the application of these results to the vertex estimation problem for convex polytopes and give some experimental results on estimates for configuration polytopes.

2. Tree estimation

Perhaps the simplest estimate of the size of a tree is the Hall–Knuth (HK) estimator [13] which is analyzed in detail by Knuth [15]. This estimate is formed as follows. For a vertex v in a tree T, let $n(v)$ be the number of children of v. A random path

$$v_0, v_1, \ldots, v_k$$

is generated such that:

(i) v_0 is the root and v_k is a leaf of T, and

(ii) for $i = 1, \ldots, k$, v_i is a child of v_{i-1} chosen randomly with probability $1/n(v_{i+1})$.

An estimate $\boldsymbol{N}$ of the number of vertices of T is given by

$$\boldsymbol{N} = 1 + \sum_{i=0}^{k-1} n(v_0)n(v_1)\cdots n(v_i).$$

This estimate can be understood as the size of a tree where each vertex at depth i from the root has the same number of children as v_i. Hall and Knuth showed that the expected value $\overline{N}$ of N is the number of nodes in T, showing that the estimate is *unbiased*. The *cost* of an estimator is defined to be the number of tree nodes evaluated in obtaining the estimate. The cost of the HK estimator is $k+1$, the number of nodes on the random path. In the worst case it is one more than the height of the tree.

In order to use this estimator, it is necessary to find a random path in a tree. In general, given a graph G it is not possible to find a random path in the depth-first or breadth-first spanning tree of G without actually computing the tree. The HK estimator has been used primarily for a relatively limited number of well structured backtracking examples. Reverse search, however, allows a random path to be generated. This is because, given any node of the tree, it is possible to generate all children of this node without any other information. A direct application of the HK estimator to the basis graph of a polytope using reverse search to generate a random path gives an estimate of the number of bases. This is discussed further in the next section.

The major problem with the HK estimator is that the estimate has a high variance. Some methods to reduce the variance are described in [15]. In the next three subsections, we discuss ways to reduce the variance.

2.1. Multiple probes from the root

The simplest way to reduce the variance is simply to make a number q of independent estimates $N_1, N_2, \ldots, N_q$ from the root. The estimate $M(q)$ for T is

$$M(q) = \frac{1}{q}\sum_{i=1}^{q} N_i$$

which is clearly unbiased. If the probes are independent the variance of $M(q)$ is given by

$$\operatorname{Var} M(q) = \frac{\operatorname{Var} N}{q}.$$

The cost of computing $M(q)$ is q times the cost of computing a single estimate N.

2.2. Single probe with look-ahead

The HK estimator tends to underestimate the size of the tree with fairly high probability. Consider a large binary tree with one of the two nodes attached to the root a leaf. With probability $\frac{1}{2}$ the HK estimate is 3. In order to avoid these low estimates it is possible to "look-ahead" in the tree and avoid paths that lead to "small" subtrees. This can be done as follows.

Led d be a non-negative look-ahead parameter. The path is generated from the root by choosing the next vertex at random from among those children which have descendants at depth d. The selection is biased towards the child with the most descendants at depth d. If no such children exist the path is terminated, otherwise the procedure is repeated from the selected child. This procedure combines features of methods studied by Knuth [15], and the proof that the estimate is unbiased can be derived along similar lines.

For a tree T and any vertex v in the tree, let $n_d(v)$ be the number of descendants at exactly depth d from v. If T has depth at least d, a random path

$$v_0, v_1, \ldots, v_k$$

is generated such that

(i) v_i is a child of v_{i-1} and is chosen with probability $n_{d-1}(v_i)/n_d(v_{i-1})$ from the children of v_{i-1}, where $i = 1, \ldots, k$, and

(ii) v_k is the first vertex in the path with $n_d(v_k) = 0$.

Let

$$m(v) = 1 + \sum_{\substack{w \text{ is a child of } v \\ n_{d-1}(w)=0}} |\text{subtree rooted at } w|.$$

Note that in particular $m(v_k)$ is the size of the subtree rooted at v_k. The estimate $L(d)$ of the size of the tree is given by

$$L(d) = m(v_0) + \sum_{i=1}^{k} \frac{n_d(v_0)n_d(v_1)\cdots n_d(v_{i-1})}{n_{d-1}(v_1)n_{d-1}(v_2)\cdots n_{d-1}(v_i)} m(v_i).$$

For $d = 1$ this is the HK bound. The cost of computing $L(d)$ increases greatly with d in general, and depends on the average degree of nodes in T.

2.3. Single probes from depth s

Using the notation of the previous section, T has $n_s(v_0)$ nodes at depth s. As an alternative to making

$t = n_s(v_0)$ probes from the root v_0, we may make one probe from each node at depth s. Let $\boldsymbol{N}(v)$ be the HK estimate obtained for the subtree rooted at node v. Then the depth s estimate $\boldsymbol{D}(s)$ is

$$\boldsymbol{D}(s) = \sum_{i=0}^{s-1} n_i(v_0) + \sum_{v \text{ is a node at depth } s} \boldsymbol{N}(v).$$

For $s = 0$ this is the HK estimate. With $t = n(v_0)$ the cost of computing an estimate $\boldsymbol{M}(t)$ is comparable to that of computing $\boldsymbol{D}(1)$ since the same number of random probes are computed. However the latter estimate has lower variance, as made precise in the following theorem.

Theorem 2.1. *Set $t = n(v_0)$ and let $\boldsymbol{M}(t)$ and $\boldsymbol{D}(1)$ be estimates of the size of a tree T.*

(i) $\operatorname{Var} \boldsymbol{M}(t) \geqslant \operatorname{Var} \boldsymbol{D}(1)$ *with equality holding if and only if all subtrees of T at depth* 1 *have the same cardinality.*

(ii) $E \operatorname{cost} \boldsymbol{M}(t) \geqslant E \operatorname{cost} \boldsymbol{D}(1)$ *with equality if and only if the root has degree* 1.

Proof. Label the children of the root $u_1, u_2, \ldots, u_t$. Now

$$\boldsymbol{D}(1) = 1 + \sum_{i=1}^{t} \boldsymbol{N}(u_i),$$

where $\boldsymbol{N}(u_i)$ is the HK estimate of the size of the subtree rooted at u_i. Since the random variables $\boldsymbol{N}(u_i)$ are independent,

$$\begin{aligned}\operatorname{Var} \boldsymbol{D}(1) &= \operatorname{Var} \sum_{i=1}^{t} \boldsymbol{N}(u_i) \\ &= \sum_{i=1}^{t} E\big[\boldsymbol{N}(u_i) - \overline{\boldsymbol{N}}(u_i)\big]^2.\end{aligned}$$

Now let $\boldsymbol{N}$ be a HK estimate from the root v_0. Conditioned on the event that u_i is on the random path, $\boldsymbol{N}$ has the same distribution as $1 + t\boldsymbol{N}(u_i)$. From this we observe that the expected value $\overline{\boldsymbol{N}}$ of $\boldsymbol{N}$ is given by

$$\overline{\boldsymbol{N}} = 1 + \sum_{i=1}^{t} \overline{\boldsymbol{N}}(u_i).$$

The variance of $\boldsymbol{N}$ is given by

$$\begin{aligned}&E[\boldsymbol{N} - \overline{\boldsymbol{N}}]^2 \\ &= \sum_{i=1}^{t} E\big[(\boldsymbol{N} - \overline{\boldsymbol{N}})^2 \mid u_i \text{ is on path}\big] \Pr(u_i \text{ is on path}) \\ &= \frac{1}{t} \sum_{i=1}^{t} E\big[1 + t\boldsymbol{N}(u_i) - \overline{\boldsymbol{N}}\big]^2 \\ &= t \sum_{i=1}^{t} E\left[\boldsymbol{N}(u_i) - \frac{1}{t}\sum_{j=1}^{t} \overline{\boldsymbol{N}}(u_j)\right]^2 \\ &\geqslant t \sum_{i=1}^{t} E\big[\boldsymbol{N}(u_i) - \overline{\boldsymbol{N}}(u_i)\big]^2 \\ &\geqslant t \operatorname{Var} \boldsymbol{D}(1).\end{aligned}$$

Now the result follows since

$$\operatorname{Var} \boldsymbol{M}(t) = \frac{\operatorname{Var} \boldsymbol{N}}{t} \geqslant \operatorname{Var} \boldsymbol{D}(1).$$

For the costs, we observe that

$$\begin{aligned}E \operatorname{cost} \boldsymbol{M}(t) &= t \operatorname{cost} \boldsymbol{N} \\ &= t\left(1 + \frac{1}{t}\sum_{i=1}^{t} E \operatorname{cost} \boldsymbol{N}(u_i)\right) \\ &\geqslant 1 + \sum_{i=1}^{t} E \operatorname{cost} \boldsymbol{N}(u_i) \\ &= E \operatorname{cost} \boldsymbol{D}(1). \qquad \square\end{aligned}$$

2.4. Tail estimates

A tail estimate is useful in determining the feasibility of a vertex generation problem. If $\boldsymbol{X}$ is an unbiased estimate of the size of a tree T, then by Markhov's inequality, for every positive constant c,

$$\Pr\left(|T| > \frac{\boldsymbol{X}}{c}\right) > 1 - \frac{1}{c}. \tag{1}$$

This tail estimate can be used with any of the preceding estimators as a way of discarding hopeless problems. However, it does not give assurances that when $\boldsymbol{X}$ is small the problem is probably tractable. For this one requires bounds on the other tail of $\boldsymbol{X}$. At present we have no such bounds for any of the above estimators.

2.5. Red–blue trees

For the vertex generation application, a modified estimation problem is also of interest. Assume that T

has nodes colored either red or blue. It is required to estimate the number of red vertices. Again, let

$v_0, v_1, \ldots, v_k$

be a random path in the tree. Then the HK estimate for the number of red nodes, $\boldsymbol{R}$, is given by

$$\boldsymbol{R} = \mathbf{1}_{\{v_0 \text{ red}\}} + \sum_{i=0}^{k-1} \mathbf{1}_{\{v_{i+1} \text{ red}\}} n(v_0) \ldots n(v_i),$$

where $\mathbf{1}_{\{\}}$ is the indicator function. It is easy to show that this is an unbiased estimate of the number of red nodes. Similar modifications apply to the three variance reduction methods described above.

3. Vertex estimation

To apply the methods of the previous section, one must be able to compute all children of any given tree vertex. The original applications of the HK estimator were for trees generated by standard backtracking algorithms for well structured problems. For less structured graph search problems the method cannot be applied: unfortunately, there is no easy way to compute even the degree of a vertex in a tree generated by depth-first search of an arbitrary graph. However, the reverse search method [4] is a method of generating search trees which allow the children of an arbitrary node in the tree to be generated. In fact, it is not even necessary to keep track of the path in the search tree to the given node. The code *lrs* [3] implements the reverse search method for the problem of generating all of the vertices of a polytope, and is a modification of the method originally described in [2]. For the vertex generation problem, *lrs* generates a spanning tree of a set of bases of an input polytope P, as described in more detail below. If P is simple, each basis corresponds to a unique vertex. Hence the estimates of the previous section immediately give estimates for the number of vertices of P. Note that the worst case cost of obtaining a HK estimate is proportional to the maximum path length $+1$ in the reverse search tree. This path length is the worst case number of pivots required by the simplex method to go from a feasible basis to the optimum basis. In *lrs*, the pivoting rule is the least subscript rule (Bland's rule) for entering variable, with lexicographic perturbation to resolve ties for leaving variable (in the case of degeneracy). The Klee–Minty examples are known to generate an exponential number of pivots under Bland's rule [1], and so a HK estimate in this case could involve the generation of a random search path of exponential depth. This pathological behaviour is unlikely to be observed in practice.

In the case where P is non-simple, the same vertex may be represented by many bases. Since a basis is a subset of n inequalities from the m inequalities defining P, we can represent it by an n element subset of integers from $1, \ldots, m$. For a degenerate vertex, the *lex-min* basis is the lexicographically smallest subset of indices that define the vertex. *lrs* does not generate all bases of P: it generates the bases of a symbolic perturbation of P to a simple polytope. The bases generated are known as *lex-positive bases* (see, for example, [14, pp. 118–122]). The lex-positive bases do however contain all *lex-min* bases, and the property of being a *lex-min* basis can be easily checked from the corresponding dictionary which is available during the reverse search. In the reverse search tree generated by *lrs* for a non-simple polytope P, we can distinguish two types of nodes; those that correspond to *lex-min* bases (red) and those that do not (blue). The number of vertices in P is exactly the number of red nodes in T. Using the techniques of the last section, we can in estimate the number of vertices of non-simple polytopes. For the highly degenerate polytopes that are often encoutered in practice, there may be very few red nodes in a tree with a very large number of blue nodes, so the vertex estimates may have enormous variance.

We can use the HK estimates of the number of bases of P to estimate the running time of *lrs* for the complete vertex generation of P. This is due to the fact that *lrs* runs in time linear in the number of bases. For any given problem, one can run *lrs* and quickly determine the running time per basis. From this an unbiased estimate of the total running time is readily obtained. Then it is possible to decide whether or not it is likely that *lrs* will be able to complete the calculation.

In Table 1 we give some experimental results on configuration polytopes that were supplied by Gerardo Garbulsky. These polytopes are described in [6] and the input files for *lrs* can be found at the ftp site given in [3]. The polytopes arise in the study of ternary alloys. Each problem is numbered $m - n$ giving the

Table 1
Configuration polytopes

Problem	Upper bound	Actual		Estimate				
		Vertices	Bases	Vertices	Bases	Probes	Nodes	Time
729-9	1.15e+10	4862	477 421	2112	134 690	3-20	9918	790.3 s
31-20	587 860	18 553	169 272	11 596	120 743	2-20	7519	146.3 s
41-16	8 544 096	29 108	287 806	69 827	204 209	2-20	5917	52.7 s
71-61	5.47e+09	3 149 579	57 613 364	2.32e+06	5.18e+07	2-5	25 765	6418 s
90-86	3 067 078	323 188	1 621 760	393 712	1.73e+06	1-5	1300	4416 s
288-281	9.16e+12	?	?	2.33e+10	1.10e+12	1-5	5289	24 days

number of rows and columns of the input file. Since each row corresponds to an inequality, the polytope therefore is defined by m inequalities in dimension $n-1$. The problems are all highly degenerate. Column 2 gives the upper bound on the number of vertices given by the Upper Bound Theorem. Where known, the actual number of vertices is given in column 3. Column 5 gives an estimate of the number of vertices of the polytope. Columns 4 and 6 give respectively the actual and estimated number of lex-positive bases. These are the bases that *lrs* will generate in a complete vertex enumeration. The estimates were obtained by the depth s probe method, by *mutt* a DEC1000 AlphaServer 4/233 using version 2.5i of *lrs* compiled for 64-bit integers. The entry s-x in the "Probes" column indicates that the estimate was obtained by averaging x depth s estimates. For example, we made 20 probes from every node at depth 3 in the tree for problem 729-9, and averaged the results. The "Nodes" column gives the total number of tree nodes explored to obtain the given estimate. By comparing this with the actual number of bases (where known), we can determine what fraction of the tree was explored in making the estimate. For example, the estimate for problem 71-61 was made by looking at roughly 0.045% of the search tree. The final column is the CPU time taken in seconds (except for the problem 288-281) for the estimate.

Considering the high degree of degeneracy, we were surprised at the accuracy of the vertex estimates (compare columns 3 and 5). The largest problem completely solved, 71-61, used a parallel version of *lrs* running on a NEC Cenju-3 system with 64 processors in 4.5 days [5]. Incidentally, *mutt* estimates that she would need about six million years to solve 288-281 with *lrs*, but this is probably optimistic. As an application of the tail inequality (1) of the previous section, with probability at least 0.999 it would take more than six thousand years to solve 288-281 in this way. For those looking for harder problems, Geraldo Garbulsky has problems 554-545 and 842-743 available on request (email: gdg@lanai.mit.edu).

References

[1] D. Avis, V. Chvatal, Notes on Bland's pivoting rule, Math. Programming Study 8 (1978) 24–34.
[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. ftp://mutt.cs.mcgill.ca/pub/doc/avis/AF92b.ps.gz.
[3] D. Avis, A C implementation of the reverse search vertex enumeration algorithm, in: H. Imai (Ed.), RIMS Kokyuroku 872, Kyoto University, May 1994. ftp://mutt.cs.mcgill.ca/pub/doc/avis/Av94a.ps.gz.
[4] D. Avis, K. Fukuda, Reverse search for enumeration, Discrete Appl. Math. 65 (1–3) (1996) 21–46. ftp://mutt.cs.mcgill.ca/pub/doc/avis/AF96a.ps.gz.
[5] A. Brüngger, A. Marzetta, K. Fukuda, J. Nievergelt, The Parallel Search Bench ZRAM and its Applications, 1996. ftp://ifor13.ethz.ch/pub/fukuda/reports/zrampoc96.
[6] G. Ceder, G.D. Garbulsky, D. Avis, K. Fukuda, Ground states of a ternary lattice model with nearest and next-nearest neighbor interactions, Phys. Rev. B 49 (1994) 1–7.
[7] B. Chazelle, An optimal convex hull algorithm in any fixed dimension, Discrete Comput. Geom. 10 (1993) 377–409.
[8] V. Chvátal, Linear Programming, W.H. Freeman, New York, 1983.

[9] A. Deza, K. Fukuda, McMullen's conditions and some lower bounds for general convex polytopes, Geom. Dedicata 52 (1994) 165–173.
[10] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer, Berlin, 1987.
[11] K. Fukuda, cdd+ Reference Manual, Version 0.74, ETH-Zentrum and Graduate School of Systems Management, Tsukuba University, 17 June 1995. http://www.ifor.math.ethz.ch/ifor/staff/fukuda/cddplus_man/cddman.html.
[12] P.M. Gruber, J.M. Wills, Handbook of Convex Geometry, North-Holland, Amsterdam, 1993.
[13] M. Hall, D.E. Knuth, Combinatorial analysis and computers, Amer. Math. Monthly 72 (1965) 21–28.
[14] J.P. Ignizio, T. Cavalier, Linear Programming, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[15] D.E. Knuth, Estimating the efficiency of backtrack programs, Math. Comp. 29 (1975) 121–136.
[16] T.S. Motzkin, H. Raiffa, G.L. Thompson, R.M. Thrall, The double description method, Ann. of Math. Stud. 8 (1953) 51–73.