Journal of the Operations Research
Society of Japan
Vol. 40, No. 3, September 1997

A POLYNOMIAL ALGORITHM FOR ENUMERATING ALL VERTICES OF A BASE POLYHEDRON

Ping Zhan
Edogawa University

(Received January 17, 1996; Final March 18, 1997)

Abstract In general, it is difficult to enumerate all vertices of a polytope in polynomial time. Here we present a polynomial algorithm which enumerates all vertices of a submodular base polyhedron in $O(n^3|V|)$ time and in $O(n^2)$ space, where V is the vertex set of a base polyhedron and n the dimension of the underlying Euclidean space. Our algorithm is also polynomial delay, and a generalization of several enumeration algorithms.

1. Introduction

We denote the set of reals by $\mathbf{R}$. Given a finite set $N = \{1, 2, \cdots, n\}$, function f: $2^N \to \mathbf{R}$ is called a *submodular function* if

$$(1.1) \qquad \forall X,\ Y \subseteq N: \ f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y).$$

The problem considered here is to enumerate all vertices of the *base polyhedron*

$$(1.2) \qquad \mathrm{B}(f) = \{ x \mid x \in \mathbf{R}^N, \forall X \subset N: \ x(X) \leq f(X),\ x(N) = f(N)\},$$

where $x(X) = \sum_{e \in X} x(e)$.

For enumeration problems, if the complexity of an algorithm can be expressed as

$$p(m)l(n),$$

where $p(m)$ is a polynomial function of input size m and $l(n)$ is a linear function of output size n, we call the algorithm *a polynomial enumerating algorithm.* For a polytope defined by a system of linear inequalities, it is well known that it is difficult to enumerate all vertices of the polytope in polynomial complexity [3] [6] [1] [5]. In this paper, we introduce a polynomial algorithm which enumerates all vertices of a base polyhedron in $O(n^3|V|)$ time and $O(n^2)$ space, where V is the vertex set of the base polyhedron and n the dimension of the underlying Euclidean space.

We also introduce another concept about enumerating algorithms. An enumerating algorithm is called *polynomial delay* if every additional extreme point is outputted in polynomial time and polynomial space of the input size. Compare it with the polynomial algorithm defined above, here polynomial algorithm means the total time and space needed in enumerating all objects, not time and space needed between two outputs. Put it in the other way, for an enumerating algorithm, if the complexity between outputs is a linear function of output size, it may be polynomial, but is not polynomial delay; while a polynomial delay enumerating algorithm is always a polynomial enumerating algorithm. We will show that our enumerating algorithm is also polynomial delay.

Let E be a finite set and $\mathcal{I}$ be a family of subsets of E, called *independent sets.* We say that $M = (E,\mathcal{I})$ is a *matroid* if the following axioms are satisfied:

(M0) $\emptyset \in \mathcal{I}$;

(M1) If $I' \subseteq I \in \mathcal{I}$, then $I' \in \mathcal{I}$;

(M1) For every $X \subseteq E$, every maximal independent subset of X has the same cardinality.

As analyzing algorithms that work on matroids we assume that each matroid is represented by a subroutine that answers questions about independence of sets in the matroids. Here we assume that there is an oracle computing the value of a submodular function.

It should be noted that our algorithm does not need the assumption of non-degeneracy (used in the simplex method). While for a general polyhedron, degeneracy is a crucial problem even for a non-polynomial enumerating algorithm.

It should also be noted that the algorithm of this paper is a generalization of several enumeration algorithms dealing with the problems such as:

(1) Spanning tree enumeration problem: For a graph $G(V, E)$, without the loss of generality, we assume $G(V, E)$ is connected. Let X be an edge subset of E. If $f(X) + 1$ equals to the number of vertices connected to at least one edge in X, then enumerating all vertices of a base polyhedron of f is equivalent to enumerating all spanning trees of the given graph $G(V, E)$.

(2) Matroid base enumeration problem: Let M be a matroid $(E,\mathcal{I})$ on E. For $X \subseteq E$, if define $f(X)$ to be the rank of X, i.e., the cardinality of a maximum independent set $I \in \mathcal{I}$ contained in X, then the vertex enumeration problem of $\mathrm{B}(f)$ here is also the matroid base enumeration problem.

We point out that the algorithm here can be applied to global minimization problems of a concave objective function on a polytope $B(f)$ defined above. Spanning tree enumerating algorithms can be applied to routing algorithms of computer networks [7]

2. Definitions and Preliminaries

A sequence of monotone increasing subsets

$$\mathcal{C} : S_0 \subset S_1 \subset \cdots \subset S_k \tag{2.3}$$

is called a *chain.* If there exists no chain which contains chain $\mathcal{C}$ as a proper subsequence, $\mathcal{C}$ is called a *maximal chain.*

First we introduce the following well known theorem.

Theorem 2.1 (Extreme point theorem [4, 3.22]): *A vector v is a vertex of a base polyhedron of* $\mathrm{B}(f)$ *if and only if for a maximal chain*

$$\mathcal{C} : \emptyset = S_0 \subset S_1 \subset \cdots \subset S_n = N, \tag{2.4}$$

we have

$$v(j) = f(S_i) - f(S_{i-1}) \ \ (i = 1, 2, \cdots, n), \tag{2.5}$$

where $\{j\} = S_i - S_{i-1}$.

From the extreme point theorem, it is clear that we can enumerate all vertices of a base polyhedron by finding all possible chains. Such a naive way can not be a polynomial algorithm since the number of vertices varies from one (when all equations (1.1) are satisfied with equality, called *modular*) to $n!$ (when all equations (1.1) of nontrivial cases are satisfied with inequality). To overcome this difficulty, we introduce following structures.

We introduce briefly certain poset and lattice related to a vertex of $\mathrm{B}(f)$. Given a vertex v of $\mathrm{B}(f)$, define

(2.6) $$\mathcal{D}(v) = \{ X \mid X \subseteq N,\ v(X) = f(X) \}.$$

By the submodular property of f, we can show that $\mathcal{D}(v)$ is closed under intersection and union, i.e., $\mathcal{D}(v)$ is a *distributive lattice.* For the distributive lattice, it is well known that there exists a poset $\mathcal{P}(\mathcal{D}(v)) = (N, \preceq)$ such that $\mathcal{D}(v)$ is isomorphic to $\mathcal{P}(\mathcal{D}(v))$. Now we describe a way to construct $\mathcal{P}(\mathcal{D}(v))$. For a vertex v of a base polyhedron and an element i in N, define *a dependence function* as

(2.7) $$\mathrm{dep}(v,i) = \bigcap\{X \mid i \in X \subseteq N,\ v(X) = f(X)\},$$

and define $j \preceq_v i$ if and only if $j \in \mathrm{dep}(v,i)$. We say that i *covers* j if and only if $j \prec_v i$ and there is no $k(\neq i,j)$ such that $j \prec_v k \prec_v i$. We use (i,j) to denote that i covers j. From the definition of dependence function, we know that if $j \prec_v i$, we have $\mathrm{dep}(v,j) \bigcup \mathrm{dep}(v,i) = \mathrm{dep}(v,i)$. This fact will be frequently used later, especially in the case when i covers j. For the above poset, its *Hasse diagram* is defined as following. For the vertex set N, a line running downward from i to j if and only if i covers j. Hasse diagram of $\mathcal{P}(\mathcal{D}(v)) = (N, \preceq_v)$ is written by G_v below for simplicity. Conversely, given a G_v of f, we may also form a chain giving v of $\mathrm{B}(f)$ as following. Put $S_0 = \emptyset$, $S_{i+1} = S_i \cup \{j\}$ $(i = 0, 1, \cdots, n-1)$, where $j \in N$ is a minimum element of $N \setminus S_i$. We use this fact to compute adjacent vertices (see Theorem 2.2).

For convenience and completeness, we give here a procedure [2] [4] to compute $\mathrm{dep}(v,i)$ $(i \in N)$ and construct G_v for a vertex $v \in \mathrm{B}(f)$.

```
(0) Put $G_v := (N, \emptyset)$ and $S := \emptyset$. Put $j := 0$, repeat (1) and (2) until $j = n$.
(1) Put $j := j+1$. Finding an element $i_j \in N - S$ with $v(S+i_j) = f(S+i_j)$. Put $S := S+i_j$.
(2) Put $T := S$, $W := \emptyset$. For each $k := j-1, j-2, \cdots, 1$, repeat the following (2-1) and (2-2).
    (2-1) If $i_k \notin W$ and $v(T - i_k) = f(T - i_k)$, then put $T := T - i_k$.
    (2-2) If $i_k \notin W$ and $v(T - i_k) < f(T - i_k)$, then add to $G_v$ a line running downward from $i_j$ to $i_k$.
          Put $W := W \cup \mathrm{dep}(v, i_k)$.
    Put $\mathrm{dep}(v, i_j) := T$.
```

The following theorem is not explicitly stated in [4], it is a result of two theorems.

Theorem 2.2 ([4, 3.46-3.47]): *For a base polyhedron of* $\mathbf{B}(f)$, *a vertex* u *is adjacent to vertex* v *if and only if we may obtain a chain giving* u *from Hasse diagram* G_v *with a pair* (i,j) *in* G_v *reversed.*

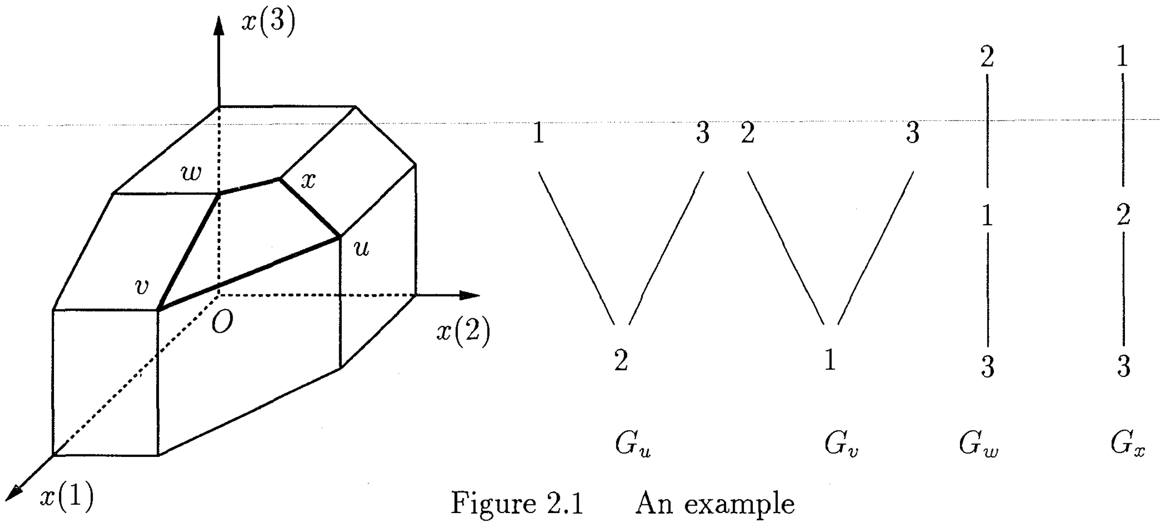

Figure 2.1 An example

Now we outline the basic idea of *reverse search algorithm* for enumeration [1]. Let $G(V, A)$ be a connected graph whose vertices are precisely the objects to be enumerated, and suppose that there is a vertex v^*, such that we can travel from every vertex v to v^* via an unique directed path of $G(V, A)$. It is clear that these paths constitute a spanning tree. If we reverse the directions of these paths, then we obtain a spanning tree with root v^*. From root v^*, we can reach every vertex of graph $G(V, A)$ by some search algorithms, say, the depth first algorithm.

In the reverse search algorithm with $G(V, A)$, we emphasis following points: (1) Set exactly one root vertex v^* of V. (2) Define a *local search function* $g: V \backslash v^* \to V$. (3) For each vertex $v \in V$, all vertices adjacent to v must be known.

3. An Enumerating Algorithm

In our reverse search algorithm, we define vertex $v^* = (v_1^*, v_2^*, \cdots, v_n^*)$ of a base polyhedron by $v_i^* = f(\{n, n-1, \cdots, i\}) - f(\{n, n-1, \cdots, i+1\})$ $(i = n-1, \cdots, 1)$ and $v_n^* = f(\{n\})$. Define *local search function* g: $V \backslash v^* \to V$

$$g(v) := v + \tilde{c}(v, j^*, i^*)(\chi_{j^*} - \chi_{i^*}), \tag{3.8}$$

where $i^* = \min\{i \mid \exists j, (j, i) \in G_v \text{ and } i < j\}$, $j^* = \max\{j \mid (j, i^*) \in G_v\}$, $\chi_i \, (i \in N)$ is a characteristic vector and $\tilde{c}(v, j^*, i^*)$ called *exchange capacity* associated with v, j^*, i^* is defined by

$$\tilde{c}(v, j^*, i^*) = \min\{f(X) - v(X) \mid j^* \in X \subseteq N, \; i^* \notin X\}. \tag{3.9}$$

The computation of $\tilde{c}(v, j^*, i^*)$ is not an easy work, but when v is a vertex, it can be done easily. We show the formula of computing $\tilde{c}(v, j^*, i^*)$ in Section 4.

If we assign weights w_i $(i \in N)$ to elements i $(i \in N)$ with $w_1 >> w_2 >> \cdots >> w_n$, we know from submodular property that moving from v in the direction defined by g leads to the adjacent vertex which decreases the value $\sum_{i \in N} w_i v_i$ $(v = (v_1, v_2, \cdots, v_n)^{\mathrm{T}} \in B(f))$ mostly and v^* is its minimum optimal solution.

In Hasse diagram G_v of vertex v, the pair of i and j with $j > i$ is said to be *true* if $(i, j) \in G_v$ and there is no k in N with $k > j$ which satisfies

(1) $(k, j) \in G_v$ with

$$f(S + i + j) + f(S + i + k) = f(S + i) + f(S + i + j + k)$$

or
(2) $(k,i) \in G_v$ with

$$f(S+i+j)+f(S+i+k)=f(S+i)+f(S+i+j+k)$$
$$v(S)=f(S),$$

where $S = \text{dep}(v,k) \bigcup \text{dep}(v,i) - j - i - k$.

An Algorithm: enumerating all vertices of a base polyhedron

Input: A finite set $N=\{1,2,\cdots,n\}$, a submodular function f: $2^N \to \mathbf{R}$.
Output: All vertices of the base polyhedron.
Step 0: Assume the optimum vertex $v^*=(v_1^*,v_2^*,\cdots,v_n^*)$ is determined by $v_{i'}^* = f(\{n,n-1,\cdots,i'\}) - f(\{n,n-1,\cdots,i'+1\})$ $(i'=n-1,\cdots,1)$, $v_n^*=f(\{n\})$. For v^*, compute dep(v^*,i') for each i' $(i' \in N)$ and construct its Hasse diagram. Get the output of vertex v^*. Let $v:=v^*$; $i:=n-1$; $j:=i+1$. Go to Step 1.
Step 1: (reverse search)
1-1: If the pair of i and j is true, let $T=\text{dep}(v,i)-i-j$, compute $\tilde{c}(v,j,i)=f(T+i)+f(T+j)-f(T+i+j)-f(T)$, get the output $u:=v+\tilde{c}(v,j,i)(\chi_j-\chi_i)$. Put $v:=u$, compute dep(v,i') for each i' $(i' \in N)$ and construct its Hasse diagram. Also compute l with $l:=\min\{i' \mid \exists j, (j',i') \in G_v \text{ and } i'<j'\}$. Put $i:=l$; $j:=i+1$, go to the beginning of Step 1.
1-2: Else if $j<n$, put $j:=j+1$, go to the beginning of Step 1.
Else if $i>1$, put $i:=i-1$ and $j:=i+1$, go to the beginning of Step 1.
Else if $v=v^*$, stop.
Else go to Step 2.
Step 2: (forward traverse)
Let $i^*=\min\{i' \mid \exists j, (j',i') \in G_v \text{ and } i'<j'\}$, and $j^*=\max\{j' \mid (j',i^*) \in G_v\}$.
Let $T=\text{dep}(v,j^*)-i^*-j^*$, compute $\tilde{c}(v,i^*,j^*)=f(T+i^*)+f(T+j^*)-f(T+i^*+j^*)-f(T)$, put $u:=v+\tilde{c}(v,i^*,j^*)(\chi_i^*-\chi_j^*)$. Put $v:=u$, compute dep(v,i') for each i' $(i' \in N)$ and construct its Hasse diagram. Put $i:=i^*$ and $j:=j^*$. Go to 1-2 of Step 1.
(End)

Let us illustrate the algorithm by an example shown in Figure 2.1. The graph G of Figure 3.1 describes the adjacent relations of vertices, and the heavy-lines are the spanning tree of search paths. Table 3.1 shows how vertices are searched by the algorithm. Note when we reach the vertex u, we do not need to traverse to the vertex v by judging whether the edge (u,v) is the edge of the spanning tree.

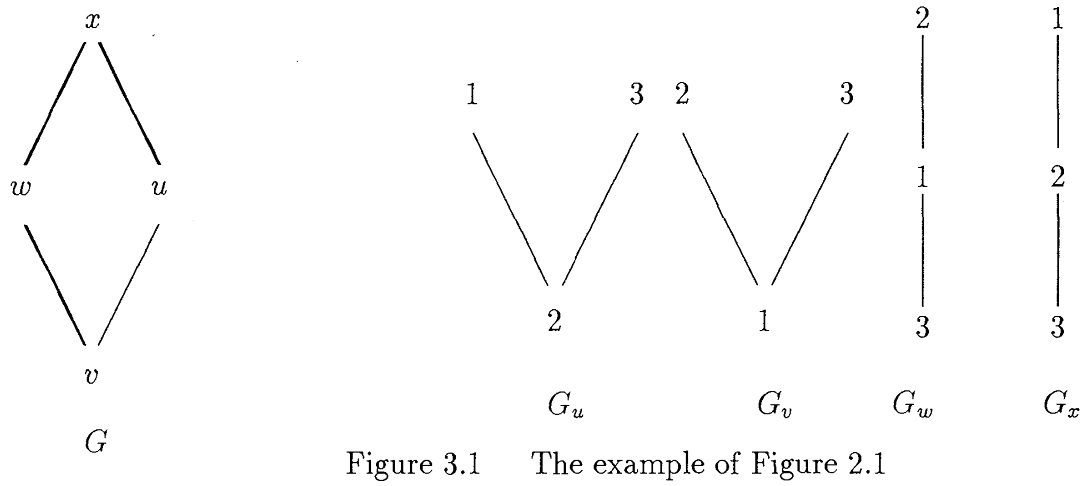

Figure 3.1 The example of Figure 2.1

Table 3.1. The enumerating process of the example

number	Step	exchanging pair (checking pair)	vertex v	G_v	output
1	0		x	G_x	x
2	1	(2,3)((1,2) (1,3))	u	G_u	u
3	2	(3,2)	x	G_x	
4	1	(1,2)	w	G_w	w
5	1	(1,3)	v	G_v	v
6	2	(3,1)	w	G_w	
7	2	(2,1)((1,3))	x	G_x	
8	1	Stop	x	G_x	

4. The Validity of the Algorithm

Lemma 4.1: *Assume that v is a vertex of $\mathrm{B}(f)$ obtained from adjacent vertex u by $v=u+\tilde{c}(u,j,i)(\chi_j-\chi_i)$. Let S'_k and S_k be the sets which are satisfied with*

(4.10) $$S'_k = \bigcap\{T \mid u(T+i+j+k) = f(T+i+j+k)\}$$

and

(4.11) $$S_k = \bigcap\{T \mid v(T+j+i+k) = f(T+j+i+k)\}.$$

Then we have $S'_k = S_k$ and also $S_k = \mathrm{dep}(v,i) \cup \mathrm{dep}(v,k) - i - j - k$.

Denote $\mathrm{dep}(v,i) - i - j$ by T. Then $\tilde{c}(u,j,i) = -\tilde{c}(v,i,j) = f(T+j) + f(T+i) - f(T) - f(T+i+j) > 0$.

Proof. Since $v(S'_k+i+j+k) = u(S'_k+i+j+k) = f(S'_k+i+j+k)$, we have $S'_k \supseteq S_k$. Conversely, $u(S_k+j+i+k) = v(S_k+j+i+k) = f(S_k+i+j+k)$, we have $S_k \supseteq S'_k$. i.e., $S'_k = S_k$.

Since $v(S_k+j+i+k) = f(S_k+j+i+k)$, from the definition of the dependence function, we have $S_k+j+i+k \supseteq \mathrm{dep}(v,i) \cup \mathrm{dep}(v,k)$, i.e., $S_k \supseteq \mathrm{dep}(v,i) \cup \mathrm{dep}(v,k) - j - i - k$. Conversely, note $\mathrm{dep}(v,t) \in \mathcal{D}(v)$ $(t \in N)$ and $\mathcal{D}(v)$ is closed under union operation, we know

(4.12) $$v(\mathrm{dep}(v,i) \cup \mathrm{dep}(v,k)) = f(\mathrm{dep}(v,i) \cup \mathrm{dep}(v,k)).$$

From the minimum property of S_k, we have $\mathrm{dep}(v,i) \cup \mathrm{dep}(v,k) \supseteq S_k+j+i+k$, i.e., $\mathrm{dep}(v,i) \cup \mathrm{dep}(v,k) - j - i - k \supseteq S_k$. Thus, we showed $\mathrm{dep}(v,i) \cup \mathrm{dep}(v,k) - j - i - k = S_k$.

The result $\tilde{c}(u,j,i) > 0$ comes from the fact that j covers i in G_u. The equation $\tilde{c}(u,j,i) = f(T+j) + f(T+i) - f(T) - f(T+i+j)$ comes from following equations

(4.13) $$u(i) = f(T+i) - f(T),$$

(4.14) $$u(j) = f(T+i+j) - f(T+i),$$

and

(4.15) $$v(j) = f(T+j) - f(T),$$

(4.16) $$v(i) = f(T+j+i) - f(T+j)$$

and the relation $v = u + \tilde{c}(u,j,i)(\chi_j - \chi_i)$, while the other entries of vectors v and u remain unchanged. □

It should be noted that in the above lemma if i and j are the maximum elements in the set $S_k + i + j$, i.e., there is no element t in dep(v,k) which satisfies $t \succ_v i$ and $t \succ_v j$, then S_k can be replaced by T. We replace S_k by S below for simplicity

Now we give the necessary and sufficient conditions which are related to the definition of g. First we have:

Lemma 4.2: *Let u be a vertex obtained from G_v with a pair (i,j) in G_v reversed. Then j=max$\{ j' \mid (j',i) \in G_u$ and $i < j'\}$ if and only if the pair of i and j is true, i.e., $(i,j) \in G_v$ and there is no k in N with $k > j$ which satisfies*

(1) $(k,j) \in G_v$ with

$$f(S+i+j)+f(S+i+k)=f(S+i)+f(S+i+j+k) \tag{4.17}$$

or

(2) $(k,i) \in G_v$ with

$$f(S+i+j)+f(S+i+k)=f(S+i)+f(S+i+j+k), \tag{4.18}$$

$$v(S)=f(S), \tag{4.19}$$

where $S =$ dep$(v,i) \bigcup$ dep$(v,k) - i - j - k$.

Note: Conditions (1) and (2) of the lemma are shown as $G_v(1)$ and $G_v(2)$ in Figure 4.1.

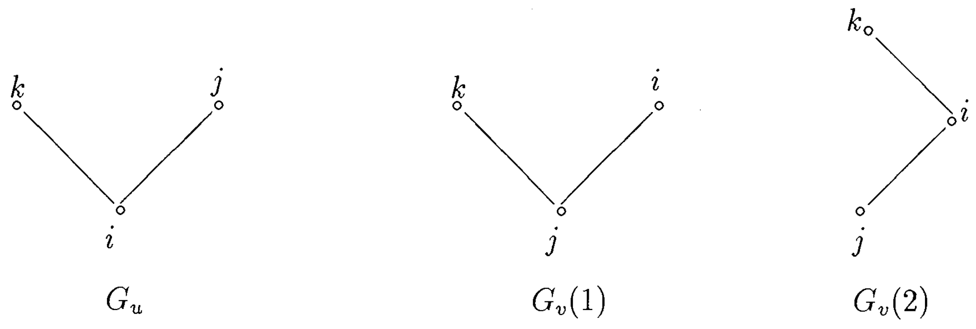

Figure 4.1.

Proof. First we show necessity. The fact that i covers j in G_v is clear.

We show condition (1) by contradiction. If there is a k which is satisfied with (1), it must satisfy $k \succ_u i$. Otherwise, by Lemma 4.1, we have

$$f(S+j+i+k)=u(S+j+i+k). \tag{4.20}$$

If k does not satisfy $k \succ_u i$, from the definition of S, we know that j and i are the maximum elements in the set $S+k+i+j$ of G_u. Hence we have

$$u(S+k)=f(S+k). \tag{4.21}$$

Note that k is the maximum element in the set $S+k$ of G_u, hence $u(S)=f(S)$. It is clear that because $v(k)=u(k)$, and j, k and i are the maximum elements in the set $S+i+j+k$ of G_v, we obtain $v(S)=f(S)$. Substituting these equations into (4.21), we obtain

$$v(S+k)=f(S+k), \tag{4.22}$$

which contradicts to $j \prec_v k$. Thus we proved $k \succ_u i$.

Now there are two possibilities. One is that k covers i in G_u, this is impossible since j is the maximum index element among the elements covering i by the assumption of necessity. ~~The other case is that there is a t with $k \succ_u t$ and $t \succ_u i$. It is clear that only j can be such~~ a t, but it is impossible since k satisfies modular equation (4.17). So k covers i. This is the same as the first case. Hence there is no k satisfying condition (1).

For condition (2) (note the equation $v(S) = f(S)$ of condition (2) means that there is no t other than k and i in $\text{dep}(v, k)$ which satisfies $t \succ_v j$), if we suppose that there is a k which is satisfied with condition (2), by the same argument as (1), we have

$$v(S+k) = f(S+k), \tag{4.23}$$

which is contradiction to $j \prec_v i \prec_v k$ and therefore also contradiction to the necessity assumption.

For sufficiency, it is trivial if there is no k other than j covering i in G_u. If there is such a k covering i in G_u (then $f(S+i+j) + f(S+i+k) = f(S+i) + f(S+i+j+k)$ and $v(S) = f(S)$ are satisfied), we show that it must cover j or i in G_v.

Replace i by j and u by v, as almost the same argument in the proof of necessity, we can prove $j \prec_v k$. Therefore, we know that there are two possibilities: k covers j, or k covers i (and i covers j) in G_v. The former is possible when the equation

$$f(S+j+i) + f(S+j+k) \geq f(S+j) + f(S+j+i+k) \tag{4.24}$$

holds with equality. The latter possibility occurs when k holds with inequality. This is a contradiction to sufficient conditions of (1) and (2). Hence j is the maximum index element among elements which covers i in G_u.

The validity of the expression of S is immediately followed from Lemma 4.1 □

It should be noted (it has been used in the proof of the necessity) that $v(S) = f(S)$ is implied by condition (1).

Lemma 4.3: *Let u be a vertex obtained from G_v with a pair (i, j) $(i < j)$ in G_v reversed. Then there is no pair $(m, l) \in G_u$ with $m > l$ and $l < i$ if and only if there is no pair $(k, l) \in G_v$ with $k > l$ (here k may be m).*

Proof. First we show sufficiency by contradiction. The outline of proof is: Assume that there exists a pair (m, l) in G_u with $m > l$, then in G_v at least one of the following three cases occurs, m covers l, j covers l or i covers l in G_v. Since all of m, i and j are greater than l, which contradicts to the assumption of sufficiency.

Note the relation $m > l < i < j$, there are several cases.

Case 1: $j = m$. In this case, j covers i and also j covers l in G_u, see Figure 4.2.

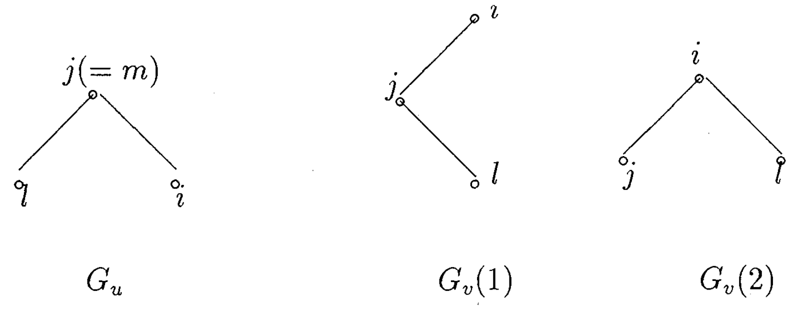

Figure 4.2.

First we show $i \succ_v l$. Otherwise, let $S = \{t \mid t \prec_u j\} - i$, $S' = \{t \mid t \prec_v i\} - j$, as shown in Lemma 4.1, we can prove $S = S'$, which contradicts to $l \in S$ and $l \notin S'$. Thus we proved $i \succ_v l$. Here it is clear that l is covered else by i or by j in G_v, a contradiction.

In Cases 2 $\sim$ 6: Suppose $S = \{t \mid t \prec_u m\}$.

Case 2: Both i and j are not in S $(i \neq m)$. It is clear that $(m, l) \in G_u$ also means $(m, l) \in G_v$.

Case 3: Both i and j are in S. It is clear that the pair (m, l) in G_u does not satisfy $m \prec_v l$. We also know that there is no t which satisfies $m \succ_v t$ and $t \succ_v l$. From m covering l in G_u, we have strict inequality $f(S + m - l) + f(S) > f(S - l) + f(S + m)$. Hence m covers l in G_v.

Case 4: $i \in S$ and $j \notin S$ $(j \neq m)$. The argument is similar to that of Case 3.

Case 5: $i \notin S$ and $j \in S$ $(j \neq m)$. The fact $(j, i) \in G_u$ implies that this case does not occur.

Case 6: $m = i$, i.e., we have j covering $i(= m)$ and i covering l in G_u. With the same argument as that in Case 1, we can show $i \succ_v l$. When $S' = \{t \mid t \prec_u j\} - m = S$, then either i or j covers l in G_v, as shown in Figure 4.3. Otherwise, there is a t $(t \notin \{i, j\})$ with j covering t in G_u. If $t < l$, it is the same as in Case 1. If $t > l$, two possibilities occur. One with t covering l is the same as in Case 2. The other is that t does not cover l in G_u, then l is covered by i or j, also as shown in Figure 4.3.

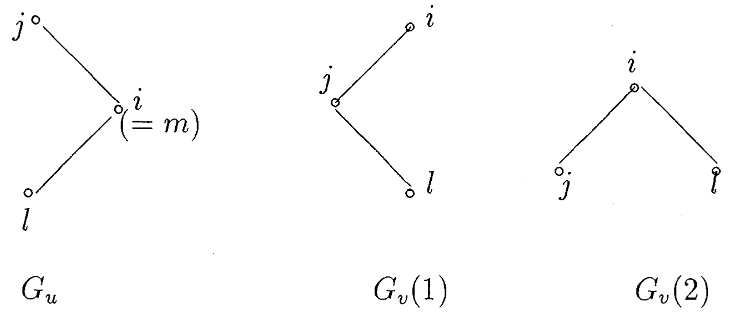

Figure 4.3.

For necessity, we show if there is a pair (m, l) with $l < i$ and $m > l$ in G_v, then $(k, l) \in G_u$ with $k > l$ (note that k may be m here). It should be noted that the proof is symmetric to that of sufficiency if we exchange j with i.

Case 1: $m = i$ in G_v. Then $i(= m)$ covers l and also i covers j in G_v. As in Case 1 of sufficiency, there are two possibilities: j covers l or i covers l in G_u.

Case 2: Neither $m = i$ nor $m = j$. By the same argument in the proof of sufficiency, l must be covered by m in G_u.

Case 3: $m = j$. Then i covers $j(= m)$ and j covers l in G_v. Again, as in Case 6 of sufficiency, we can prove $(k, l) \in G_u$ with $k > l$. □

When we consider a pair (i, j) in Lemma 4.3, we have to check every edge in G_v to make sure whether the condition of Lemma 4.3 is satisfied. But it can be done without any local information around i and j. This observation is important to the complexity of the algorithm. In the algorithm, computing $l := \min\{i', \mid \exists j', (i', j') \in G_v \text{ with } i' < j'\}$ and searching only i with $i \leq l$ is based on Lemma 4.3. Taking this fact and true judgment into account considerably saves the time complexity of the algorithm.

Lemma 4.4: *In Step 1 of the algorithm,* $g(v+\tilde{c}(v,j,i)(\chi_j-\chi_i))=v$.

Proof. The condition of Lemma 4.3 is satisfied since we only consider $i \leq l$, where ($l = \min\{ i' \mid \exists j',\ (j',i') \in G_v \text{ and } i' < j'\}$). We exchange i and j of the pair (i,j) with $j > i$ only when the condition of Lemma 4.2 is satisfied. □

It should be noted that the only edges which are omitted in reverse search algorithm are those which do not satisfy the condition of Lemma 4.3. Therefore those edges can not be the edges of the spanning tree defined by g. This fact together with Lemma 4.4 and the validity of reverse search show that the outputs of the above algorithm are all vertices of a base polyhedron.

Theorem 4.5: *There is an implementation of reverse search for enumerating vertices of a base polyhedron with time complexity* $O(n^3|V|)$ *and space complexity* $O(n^2)$.

Proof. First we consider the time complexity. Suppose that we are given submodular function. Then the time for computing $f(X)$ for any $X \subseteq N$ is constant since our oracle assumption.

In Step 0, the time for computing v^* is $O(n)$, and the time for computing $\mathrm{dep}(v^*,i)$ for each $i=(1,2,\cdots,n)$ and its Hasse diagram is known to be bounded by $O(n^2)$ (see the algorithm given in Section 2).

In the reverse search of the algorithm, the time to check whether the pair of i and j is true is bounded by $O(n)$. Note that there can be at most n^2-1 such pairs (the number of edges) in a Hasse diagram and these are done for each vertex of the base polyhedron. Hence the total time for this implementation is bounded by $O(n^3|V|)$. Now consider the time for computing $\mathrm{dep}(v,i)$ $(i=1,2,\cdots,n)$, a Hasse diagram and the index l defined in Step 1-1. As described in Step 0, the time is bounded by $O(n^2)$. Since these are also done for each vertex, the total time bound is $O(n^2|V|)$. It is clear that the time complexity in Step 1-2 is $O(n^2|V|)$

In the forward search, the time for computing i^* is bounded by $O(n^2)$ and the time for j^* is $O(n)$, the time for computing dependence functions and a Hasse diagram is bounded by $O(n^2)$. Hence the total time for the forward search is bounded by $O(n^2|V|)$.

Summarizing above arguments, the time for enumerating all vertices of a base polyhedron is bounded by $O(n^3|V|)$.

Now we consider the space complexity. In Step 0, we need $O(n)$ space for vertex v^*, and $O(n^2)$ for dependence functions and the Hasse diagram of vertex v^*, and also some constant space for l, i and j. In Steps 1 and 2, the space $O(n^2)$ remains unchanged. Hence the total space complexity is $O(n+n^2)$ or $O(n^2)$. □

Note vertex number $|V|$ is not contained in the space complexity above. So, it does not mean the space needed to save all outputs V, which in general is not bounded by $O(n^2)$. $O(n^2)$ is just the space needed in executing our enumerating algorithm. It should be noted that although we need $O(n)$ space to save a vertex, the total space needed to save all vertices of a base polyhedron is $O(|V|+n)$ not $O(n|V|)$ because exchanging property, $u=v+\tilde{c}(v,i,j)(\chi_i-\chi_j)$.

We point out again that the complexity given in Theorem 4.5 is based on the assumption that a submodular function $f: \mathbf{2}^N \to \mathbf{R}$ is defined, or there is an oracle that computes the value $f(X)$ for each $X \subseteq N$.

Note the complexity above does not contain the number of linear inequality, which generally is much greater than the dimension n (here we have 2^n inequalities).

Lemma 4.6: *The depth of the search path of g is less than n^2.*

Proof. Let v be a vertex of a base polyhedron and G_v its Hasse digram. Exchanging i with j of a pair (i,j) $(i < j) \in G_v$ creates a new adjacent vertex, say u. Considering sets S and S' defined as $S = \{t \,|\, t \prec_v i\}$ and $S' = \{t \,|\, t \prec_u i\}$, for each k with $k \notin S$, we also have $k \notin S'$. Exchange i with j of the pair (i,j) decreases the number $|S|$ at least one element j. Since there are only n elements, we have the conclusion. □

It is easy to see that the above proof can be stated in a symmetric way, i.e., we can also consider sets $S = \{t \,|\, t \succ_v j\}$ and $S' = \{t \,|\, t \succ_u j\}$.

Now we can have a conclusion about polynomial delay. The crucial points that our enumerating algorithm is also a polynomial delay algorithm are listed below: (1) the lattice structure of an extreme point which gives in polynomial time and space the exact information about all adjacent points, (2) for each point, the number of adjacent points is bounded by n^2, (3) the depth of the search path of g is less than n^2 (Lemma 4.6), and (4) we never traverse each edge of the search spanning tree more than two times. These arguments imply the following theorem.

Theorem 4.7: *The enumerating algorithm in Section 3 is also polynomial delay.*

In above arguments, we do not consider connected components (separators)[4]. Although we can considerably reduce the complexity, the upper bound complexity of the algorithm retains unchanged.

Except for the current vertex, no information about the searched vertices is needed in the reverse search algorithm. This property makes parallelization of the reverse search possible. One trivial implementation is to assign some free processor a child of the root whose branch is not yet traversed. This can be done recursively [1]. Lemma 4.6 tells us that the parallel implementation is powerful.

5. An Unbounded Base Polyhedron

Here, we make a brief outline about how the reverse search algorithm can be generalized to a more general set family than 2^N.

As has been mentioned in Section 2, given a set N, a collection of subsets of N forms a distributive lattice $\mathcal{D}$ if it is closed under set union and intersection. If a submodular function is defined on $\mathcal{D} \neq 2^N$, it is known that the base polyhedron related to $\mathcal{D}$ is unbounded (Theorem 3.12 of [4]). For such a distributive lattice, there exists a poset $\mathcal{P}(\mathcal{D}) = (N, \preceq)$ (obtained by the decomposition method [4]) isomorphic to $\mathcal{D}$. If the base polyhedron is *pointed* (there exists at least one extreme point), the edges (or lines) of its Hasse diagram $\mathcal{P}(\mathcal{D})$ are corresponding to the extreme rays of the base polyhedron (Theorem 3.26 of [4]). These edges will appear in every $\mathcal{P}(\mathcal{D}(v))$ $(v \in V)$ and can not be changed during enumerating vertices.

Acknowledgements The author would like to thank the referees sincerely for their many valuable suggestions and comments, which greatly improve the structure and contents of the manuscript. And also Prof. K. Fukuda and Doctors Z. Yang, X. Zhang and Y. Dai for their carefully reading the manuscript and useful comments.

References

[1] D. Avis, K. Fukuda, Reverse search for enumeration, Discrete Applied Mathematics, **65** (1996) 21-46.

[2] R. E. Bixby, W. H. Cunningham and D. M. Topkis, Partial order of a polymatroid extreme point, Mathematics of Operations Research **10** (1985) 365-378.
[3] V. Chvatal, Linear Programming, Freeman, New York, 1983.
[4] S. Fujishige, Submodular Function and Optimization, Ann. Discrete Math., Vol. 47, North-Holland, Amsterdam, 1991.
[5] P. Gritzmann, V. Klee, On the complexity of some basic problems in computational Convexity: I. Containment problems, Discrete Mathematics **136** (1994) 129-174.
[6] A. Schrijver, Theory of Linear and Integer Programming, Wiley-Interscience, New York, 1986.
[7] A. S. Tanenbaum, Computer Networks 3rd, Prentice Hall, Amsterdam, The Netherlands, 1996.

Ping Zhan
Department of Environmental Information
Edogawa University
474, Komaki, Nagareyama-shi, Chiba-Ken,
270-01, Japan