The Minimum-Norm-Point Algorithm Applied to Submodular Function Minimization and Linear Programming

Satoru FUJISHIGE[*], Takumi HAYASHI[†], and Shigueo ISOTANI[‡]

September, 2006

Abstract

In the present paper we consider applications of the minimum-norm-point algorithm to submodular function minimization and linear programming. Although combinatorial polynomial algorithms for submodular function minimization (SFM) have recently been obtained and linear programming problems can be solved in polynomial time by interior-point algorithms, there still remain (open) problems of reducing the complexity of the SFM algorithms and of devising a strongly polynomial algorithm for linear programming. The present paper has been motivated by an attempt to solve these problems. We show some possible approaches to them by means of the minimum-norm-point algorithm. Computational results on submodular function minimization reveals that our algorithm outperforms the existing polynomial algorithms for SFM. We also carry out preliminary experiments on our algorithm for LP by using MATLAB, which shows some interesting behavior of our algorithm.

Keywords: The minimum-norm-point algorithm, submodular function minimization, linear programming, zonotopes

1. Introduction

Philip Wolfe [15] presented an algorithm for finding the minimum-norm point in the convex hull of a given finite set of points in the n-dimensional Euclidean space $\mathbf{R}^n$ (von

[*]Corresponding author. Research Institute for Mathematical Sciences, Kyoto University, Kyoto 606-8502, Japan. E-mail: fujishig@kurims.kyoto-u.ac.jp

[†]Research Institute for Mathematical Sciences, Kyoto University, Kyoto 606-8502, Japan. E-mail: thayashi@kurims.kyoto-u.ac.jp

[‡]Faculdade de Ciências Econômicas e Administrativas de Osasco – FAC-FITO, Osasco, SP, Brazil. E-mail: shigueo_isotani@yahoo.com

Hohenbalken [7] also gave essentially the same algorithm for more general objective functions). In the present paper we consider applications of the minimum-norm-point algorithm to submodular function minimization and linear programming. Combinatorial polynomial algorithms for submodular function minimization (SFM) have been obtained by [10, 9, 13], and linear programming problems can be solved in polynomial time by interior-point algorithms (see, e.g., [1]). However, there still remain (open) problems of reducing the complexity of the SFM algorithms and of devising a strongly polynomial algorithm for linear programming. The present paper has been motivated by an attempt to solve these problems. We will show some possible approaches to them by means of the minimum-norm-point algorithm.

The minimum-norm-point algorithm keeps a simplex (a set of affinely independent set of points) chosen from the given point set P. Updating such a simplex requires a solution of a linear optimization problem over the convex hull $\hat{P}$ of P, and the algorithm works if the linear optimization can (efficiently) be done over the polytope $\hat{P}$. In the original problem setting by Wolfe [15] the polytope $\hat{P}$ is expressed as the convex hull of the set P of given points. Hence, the linear optimization over $\hat{P}$ can be done trivially by the evaluation of a linear function on given points in P. For a general polytope Q, however, the number of extreme points of Q can be exponentially large with respect to dimension n, so that the minimum-norm-point algorithm cannot be used in the original, trivial way. However, there are interesting classes of polytopes on which linear optimization can efficiently be done, even if the number of extreme points of Q is exponentially large with respect to dimension n.

The following is one of such classes of polytopes. It is well known that the greedy algorithm [2] works for base polyhedra associated with submodular functions (see [6] for details about submodular functions and related polyhedra). Hence we can easily make linear optimization over base polyhedra (although the number of extreme points can be $n!$). This fact leads us to an algorithm for submodular function minimization by means of the minimum-norm-point algorithm, which will be discussed in Section 3.

Moreover, we can formulate linear programming problems in terms of zonotope (the Minkowski sum of line segments; see, e.g., [16]), which will be discussed in Section 4. Zonotopes are also typical polytopes on which linear optimization can be done in a greedy way. We will show how to adapt the minimum-norm-point algorithm, based on this fact, to solve linear programming problems. The algorithm consists of solving a sequence of minimum-norm-point problems with a multi-dimensional Newton-like algorithm to be given in this paper.

We examine the proposed algorithms for submodular function minimization and linear programming by computational experiments in Sections 3.3 and 4.3. Computational results for submodular function minimization will show that the minimum-norm-point algorithm outperforms the existing polynomial-time algorithms in [9, 10, 13]. Computational experiments for linear programming, though very preliminary, are also made by

using MATLAB, which will show some interesting behavior of the algorithm.

2. The Minimum-Norm-Point Algorithm

In this section we describe Wolfe's algorithm [15] for finding the minimum-norm point in a polytope for completeness.

2.1. Description of the minimum-norm-point algorithm

Consider the n-dimensional Euclidean space $\mathbf{R}^n$. Suppose that we are given a finite set P of points p_i $(i \in I)$ in $\mathbf{R}^n$. The problem is to find the minimum-norm point x^* in the convex hull $\hat{P}$ of points p_i $(i \in I)$.

Wolfe's algorithm [15] is given as follows.

The Minimum-Norm-Point Algorithm

Input: A finite set P of points p_i $(i \in I)$ in $\mathbf{R}^n$.
Output: The minimum-norm point x^* in the convex hull $\hat{P}$ of the points p_i $(i \in I)$.

Step 1: Choose any point p in P and put $S := \{p\}$ and $\hat{x} := p$.

Step 2: Find a point $\hat{p}$ in P that minimizes the linear function $\langle \hat{x}, p \rangle = \sum_{k=1}^{n} \hat{x}(k)p(k)$ in $p \in P$. Put $S := S \cup \{\hat{p}\}$.
If $\langle \hat{x}, \hat{p} \rangle = \langle \hat{x}, \hat{x} \rangle$, then return $x^* = \hat{x}$ and halt;
else go to Step 3.

Step 3: Find the minimum-norm point y in the affine hull of points in S.
If y lies in the relative interior of the convex hull of S, then put $\hat{x} := y$ and go to Step 2.

Step 4: Let z be the point that is the nearest to y among the intersection of the convex hull of S and the line segment $[y, \hat{x}]$ between y and $\hat{x}$. Also let $S' \subset S$ be the unique subset of S such that z lies in the relative interior of the convex hull of S'. Put $S := S'$ and $\hat{x} := z$. Go to Step 3.
(End)

The cycle formed by Step 2 and Step 3 is called a *major cycle*, and the one by Step 3 and Step 4 a *minor cycle*. Every major cycle increases the size of the simplex S by one, while every minor cycle decreases the size of the simplex S by at least one. A simplex S is called a *corral* if the minimum-norm point in the affine hull of S lies in the relative interior of the convex hull of S. When we go from Step 3 to Step 2 in a major cycle, the current simplex S is a corral. Note that every corral S uniquely determines the minimum-norm point $\hat{x}$ and that every time we get a new corral, the norm of the new $\hat{x}$ strictly decreases. Also note that at most $n - 1$ repetitions of Step 3 and Step 4 in a minor cycle

give a corral, so that the Wolfe algorithm described above terminates in a finite number of steps. (It is open to determine whether the Wolfe algorithm runs in polynomial time.)

In Step 3, for $S = \{p_i \mid i \in I\}$ we have $y = \sum_{i \in I} \mu_i p_i$ with $\sum_{i \in I} \mu_i = 1$. Note that y lies in the relative interior of the convex hull of S if and only if $\mu_i > 0$ for all $i \in I$, where recall that S is affinely independent. In Step 4, both $\hat{x}$ and y are expressed as $\hat{x} = \sum_{i \in I} \lambda_i p_i$ and $y = \sum_{i \in I} \mu_i p_i$. Then, the point z is determined in such a way that $z = (1 - \beta)\hat{x} + \beta y$, $(1 - \beta)\lambda_i + \beta\mu_i \geq 0$ for all $i \in I$, and β is as large as possible.

Remark: When implementing the Wolfe algorithm, we should take care of numerical errors by introducing small tolerance intervals for decisions such as '$\alpha = \beta$?'. Besides these, the algorithm is self-correcting, so that it is stable against numerical errors. □

2.2. Applicability of the algorithm

The Wolfe algorithm requires linear optimization in Step 2, which can be done by computing $\langle \hat{x}, p \rangle$ for all points p in P. If the number of points in P is exponential in the dimension of the space $\mathbf{R}^n$, then it becomes hard to perform the linear optimization in Step 2.

Now, suppose that the set P is implicitly given as the set of extreme points of a polytope Q in $\mathbf{R}^n$. Then the Wolfe algorithm works if linear optimization over Q can efficiently be made. There are classes of polytopes on which linear optimization can efficiently be done. For example, we have

(1) base polyhedra, associated with submodular functions, on which the so-called greedy algorithm finds optimal (extreme) points, and

(2) zonotopes on which every linear optimization can be done in a greedy way,

where a zonotope is the Minkowski sum of line segments (or an affine transformation of a unit hypercube).

Remark: A pointed polyhedron is called *edge-polynomial* [5] if the number of edge vectors of the polyhedron is polynomial in the dimension of the input data space, where edge vectors are identified up to nonzero multiples. Base polyhedra and zonotopes are typical edge-polynomial polyhedra. The number of edge vectors of base polyhedra is $\mathrm{O}(n^2)$ with n being the dimension of the space, and that of zonotopes is at most the number of the generators. It should be noted that linear optimization over any edge-polynomial polyhedron is easy (solvable in strongly polynomial time) under certain conditions, so that the minimum-norm-point algorithm works for edge-polynomial polyhedra. □

We shall show how the Wolfe algorithm works for base polyhedra (in Section 3), and how linear programming problems can be formulated in terms of zonotope and can be solved by the Wolfe algorithm (in Section 4).

3. Base Polyhedra and Submodular Function Minimization

In this section we show how the Wolfe algorithm can be used to minimize submodular functions.

3.1. Submodular functions and base polyhedra

Let E be a finite nonempty set and f be a submodular function on 2^E, i.e., $f : 2^E \to \mathbf{R}$ satisfies

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \tag{3.1}$$

for any $X, Y \subseteq E$. We suppose that $f(\emptyset) = 0$ without loss of generality. We then define polyhedra

$$\mathrm{P}(f) = \{x \mid x \in \mathbf{R}^E,\ \forall X \in 2^E : x(X) \leq f(X)\}, \tag{3.2}$$

$$\mathrm{B}(f) = \{x \mid x \in \mathrm{P}(f),\ x(E) = f(E)\}. \tag{3.3}$$

Here, $\mathrm{P}(f)$ is called the *submodular polyhedron* and $\mathrm{B}(f)$ the *base polyhedron*, associated with submodular function f on 2^E.

Remark: Since $\mathrm{B}(f)$ defined as above is bounded, it is also called a base polytope. Note that $\mathrm{P}(f)$ is always unbounded. In the general theory of submodular functions (see [6]) we consider a distributive lattice $\mathcal{D} \subseteq 2^E$ (a set of subsets of E that is closed with respect to set union $\cup$ and intersection $\cap$) and a submodular function f on $\mathcal{D}$. We assume that $\emptyset, E \in \mathcal{D}$ and $f(\emptyset) = 0$. Then $\mathrm{B}(f)$ is defined similarly as in (3.2), and is bounded only if $\mathcal{D} = 2^V$. □

The linear optimization over base polyhedron $\mathrm{B}(f)$ can easily be made by the greedy algorithm of Edmonds [2]. Here we assume that we are given an oracle for evaluation of the function value $f(X)$ for any $X \subseteq E$.

The Greedy Algorithm

Input A weight vector $w \in \mathbf{R}^E$.
Output: An optimal $x^* \in \mathrm{B}(f)$ that minimizes the linear objective function $\sum_{e \in E} w(e)x(e)$ in $x \in \mathrm{B}(f)$.

Step 1: Find a linear ordering $e_1, e_2, \cdots, e_n$ of elements of E such that

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_n). \tag{3.4}$$

Step 2: Compute

$$x^*(e_i) = f(\{e_1, e_2, \cdots, e_i\}) - f(\{e_1, e_2, \cdots, e_{i-1}\}) \quad (i = 1, 2, \cdots, n). \tag{3.5}$$

Return x^*.
(End)

We also have the following theorem that characterizes the minimizers of a submodular function $f : 2^E \to \mathbf{R}$ with $f(\emptyset) = 0$.

Theorem 3.1 ([2]): *We have*

$$\min\{f(X) \mid X \subseteq E\} = \max\{x^-(E) \mid x \in \mathrm{B}(f)\}, \tag{3.6}$$

where $x^-(e) = \min\{x(e), 0\}$ *for* $e \in E$.

Moreover, if f *is integer-valued, then the maximum in the right-hand side is attained by an integral base* $x \in \mathrm{B}(f)$. □

Note that for any $X \subseteq E$ and $x \in \mathrm{B}(f)$ we have $f(X) \geq x^-(E)$. The gap $f(X) - x^-(E)$ evaluates an upper bound for to what extent X is close to a minimizer of f. In particular, if f is integer-valued, the gap $f(X) - x^-(E)$ being less than one implies that X is a minimizer of f.

3.2. The minimum-norm point in a base polyhedron and submodular function minimization

We have the following theorem.

Theorem 3.2 ([4], [6, Sec. 7.1.(a)]): *Let* x^* *be the minimum-norm point in the base polyhedron* $\mathrm{B}(f)$ *given by* (3.3). *Define*

$$A_- = \{e \mid e \in E,\ x^*(e) < 0\}, \tag{3.7}$$
$$A_+ = \{e \mid e \in E,\ x^*(e) \leq 0\}. \tag{3.8}$$

Then, A_- *is the unique minimal minimizer of* f, *and* A_+ *the unique maximal minimizer of* f. □

Because of this theorem we can solve the submodular function minimization problem by finding the minimum-norm point in the base polyhedron $\mathrm{B}(f)$. The minimum-norm-point algorithm described in Section 2 can directly be employed to solve the submodular function minimization problem by means of the greedy algorithm of Edmonds. Computational results are given in Section 3.3.

3.3. Computational results

Combinatorial polynomial algorithms for submodular function minimization (SFM) were devised independently by Iwata, Fleischer, and Fujishige [10], and Schrijver [13]. Also Fleischer and Iwata [3] proposed a polynomial preflow-push algorithm, which has the same complexity as Schrijver's ([14]). Currently the fastest SFM algorithm has been obtained by Iwata [9]. See a nice survey [11] for more details about the developments in SFM algorithms (also see [6, Chapter VI]).

The following computational results on SFM algorithms are based on a report of [8].

3.3.1. Computational Setup

We used a Dynabook G6/X18PDE with an Intel Pentium 4, CPU 1.80GHz, 768MB of memory and running Linux RedHat version 2.4.18. All programs are written in C language and compiled with gcc using the -O4 optimization option.

We denote by FW the proposed SFM algorithm by means of the minimum-norm-point algorithm [4]. The Iwata-Fleischer-Fujishige algorithm [10] is denoted by SFM3 and Schrijver's algorithm [13] by LEX2. We also have Fleischer and Iwata's algorithm [3], denoted by PR. Moreover, HYBRID is an algorithm, proposed by Iwata [9], that combines techniques involved in SFM3 and PR.

The FW program was, first, written in FORTRAN language by Masahiro Nakayama (in his graduation thesis at the University of Tsukuba in February, 1985). We rewrote the program in C language and improved some part of it. The other programs were written by Satoru Iwata. We employed Quick Sort for the sorting algorithm required in the greedy algorithm.

We tested the algorithms using two kinds of submodular functions. One is proposed by Satoru Iwata and the other is a class of cut functions.

3.3.2. Iwata's Test Function

The submodular function suggested by Satoru Iwata is

$$f(X) = |X||V \setminus X| - \sum_{j \in X} (5j - 2n) \qquad (X \subseteq V)$$

where $V = \{1, 2, \cdots, n\}$.

The results on this function are shown in Table 1 and Table 2.

This class of test problems is very special for FW. Except for FW, HYBRID outperformed the others.

Table 1: Running times for Iwata's function

Running time (sec)					
n	FW	HYBRID	SFM3	LEX2	PR
100	0.00	0.41	1.00	2644.52	277.36
200	0.00	4.92	18.69		
300	0.00	21.77	115.44		
400	0.00	67.12	369.13		
500	0.00	166.73	894.33		
600	0.01	325.26	2820.83		
700	0.01	568.54			

Table 2: Number of generated extreme bases for Iwata's function

Number of bases					
n	FW	HYBRID	SFM3	LEX2	PR
100	2	1163	766	337348	373324
200	2	3732	4618		
300	2	6710	7309		
400	2	10803	9914		
500	2	16701	18835		
600	2	22011	33849		
700	2	28699			

3.3.3. Cut Functions

In the case of cut functions, we need to generate networks. We used the generator GENRMF available from DIMACS Challenge [17]. Each generated network has b grid-like frames of size $(a \times a)$. The number of vertices is a^2b and that of arcs $5a^2b - 4ab - a^2$. All vertices in each frame are connected to its grid neighbors and each vertex is connected by an arc to a vertex randomly chosen from the next frame.

All the running times reported here are in seconds, and we only report the user CPU time. We generated five instances for each problem family of specified size, using different random seeds. Each number shown in the tables is the averaged time over five runs.

We used GENRMF to produce two kinds of networks as follows:

- *Genrmf-long*. The number of vertices of a generated graph is $n = 2^x$. The parameters are $a = 2^{x/4}$ and $b = 2^{x/2}$.
- *Genrmf-wide*. The number of vertices of a generated network is $n = 2^x$. The parameters are $a = 2^{2x/5}$ and $b = 2^{x/5}$.

We used the submodular function minimization algorithms to compute minimum cuts. The running times for the computation are shown in Table 3 and Table 4, and numbers of generated extreme bases in Table 5 and Table 6. Figure 1 and Figure 2, respectively, represent Table 3 and Table 4.

For the genrmf-long networks LEX2 and PR were faster than HYBRID. However, for the genrmf-wide networks LEX2 was slower than HYBRID. In both cases FW outperformed the others.

Figures 3, 4, 5, and 6 show sample behaviors of iteration vs. duality gap for Genrmf-Long with $n = 63$ and $m = 222$. Here, one iteration means a generation of a new extreme base.

Table 3: Results on Genrmf-Long

Running time (sec)						
n	m	FW	HYBRID	SFM3	LEX2	PR
63	222	0.040	4.024	10.952	1.428	1.242
126	453	0.368	70.826	280.527	53.360	23.286
256	1008	3.792	7376.475		3209.700	3507.494
525	2180	46.052				
1008	4332	366.21				

Table 4: Results on Genrmf-Wide

Running time (sec)						
n	m	FW	HYBRID	SFM3	LEX2	PR
75	290	0.056	3.340	20.588	4.195	3.486
147	602	0.410	55.996	749.135	141.497	572.336
324	1395	4.596	4265.148		9607.360	2433.578
576	2544	27.170				
1024	4608	172.52				

Table 5: Results on Genrmf-Long

Number of bases						
n	m	FW	HYBRID	SFM3	LEX2	PR
63	222	98	23029	28288	526	1918
126	453	221	112328	140678	2280	5732
256	1008	515	690950		8757	14605
525	2180	1353				
1008	4332	2980				

Table 6: Results on Genrmf-Wide

Number of bases						
n	m	FW	HYBRID	SFM3	LEX2	PR
75	290	100	13564	18507	756	3519
147	602	196	80240	66346	3878	7694
324	1395	429	661802		14066	20553
576	2544	766				
1024	4608	1486				

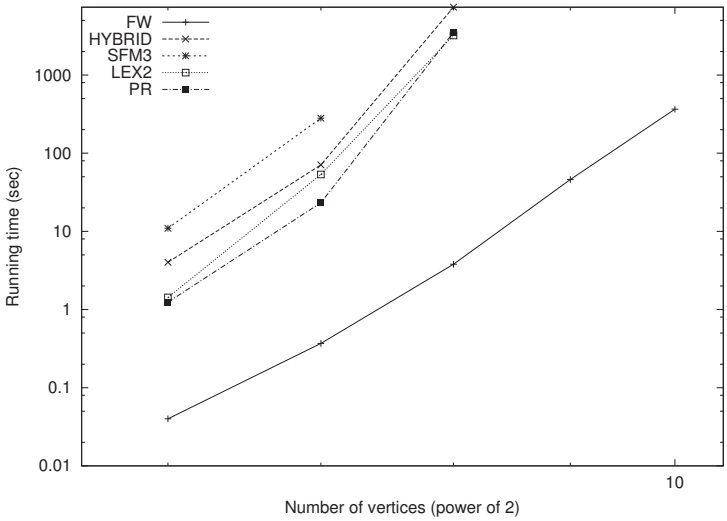

Figure 1: Number of vertices vs. time on Genrmf-Long

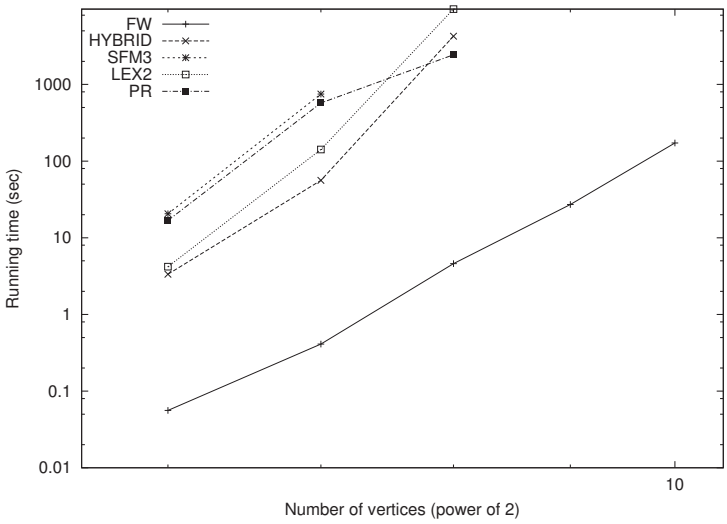

Figure 2: Number of vertices vs. time on Genrmf-Wide

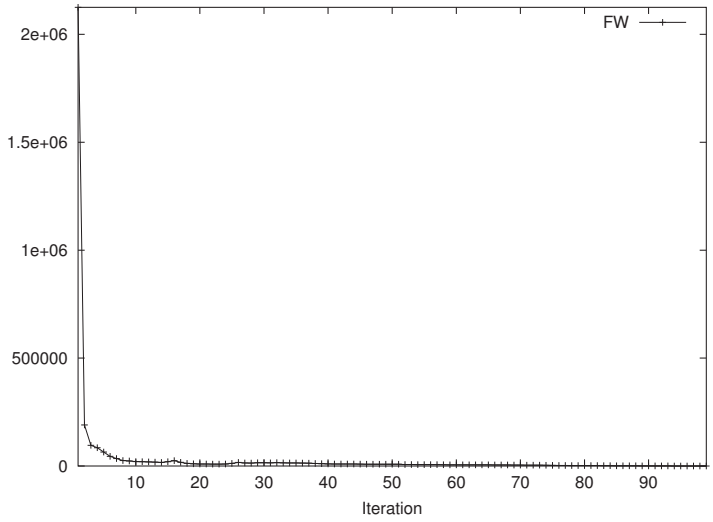

Figure 3: Iteration vs. duality gap by FW

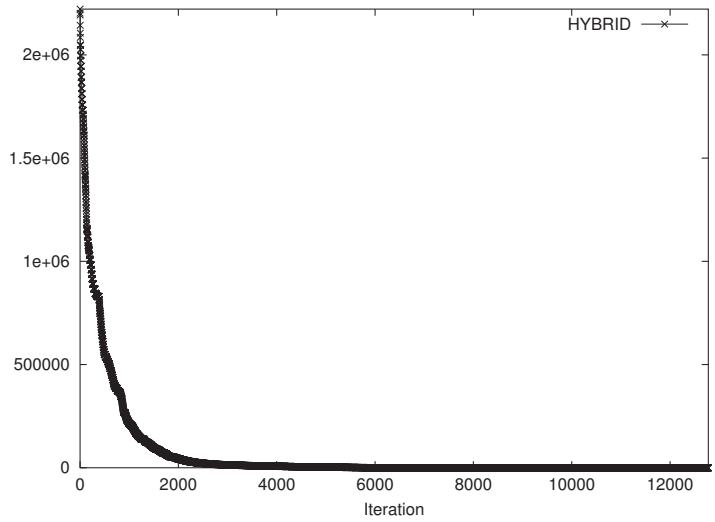

Figure 4: Iteration vs. duality gap by HYBRID

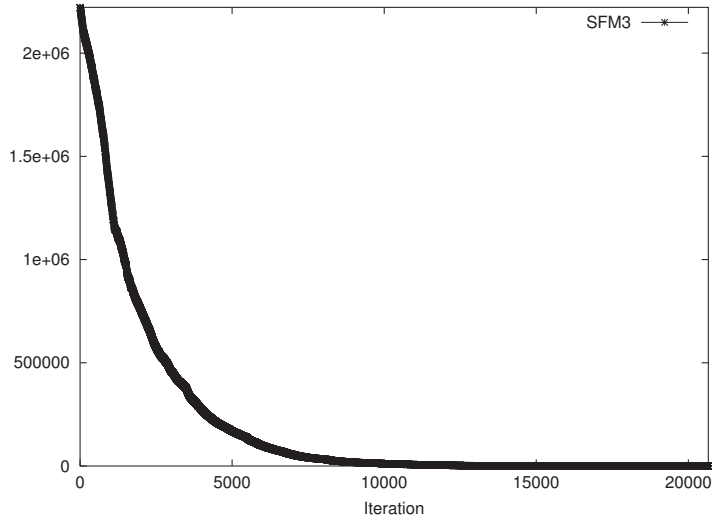

Figure 5: Iteration vs. duality gap by SFM3

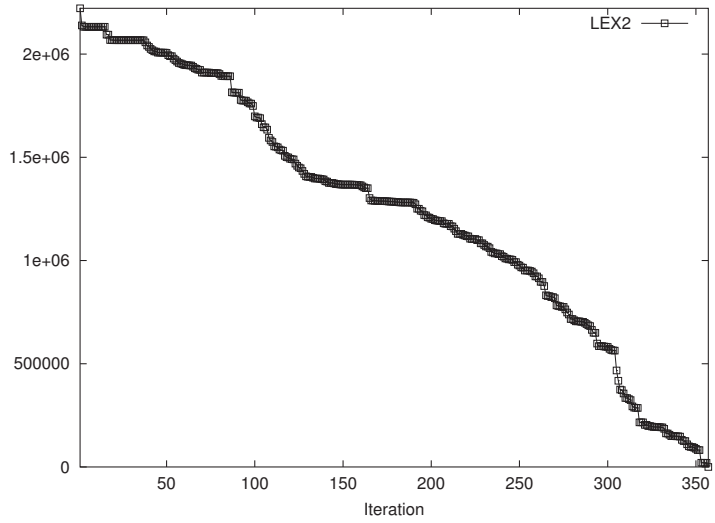

Figure 6: Iteration vs. duality gap by LEX2

4. Zonotopes and Linear Programming

In this section, we consider the linear programming (LP) problem and show how to adapt the Wolfe algorithm to solve LP.

4.1. Reformulation of linear programming problems

We consider the linear programming problem given in the following form:

$$\begin{aligned} (\mathrm{LP}) \quad \text{Maximize} \quad & cx = \sum_{j=1}^{n} c(j)x(j) \\ \text{subject to} \quad & Ax = b, \\ & l \leq x \leq u, \end{aligned} \tag{4.1}$$

where A is an $m \times n$ matrix, b an m-dimensional column vector, l and u n-dimensional column vectors, x an n-dimensional column variable vector (in $\mathbf{R}^n$), and c an n-dimensional row vector (in $(\mathbf{R}^n)^*$, the dual space of $\mathbf{R}^n$).

Remark: The ordinary standard form of LP is given by (4.1) with $l = \mathbf{0}$ and $u = (+\infty, +\infty, \cdots, +\infty)^{\mathrm{T}}$, where T denotes the matrix transpose. It should be noted that in our LP form (4.1) vectors l and u are finite-valued, so that our model is slightly restrictive. However, it suffices to consider such a bounded feasible region practically as well as theoretically (see [12, Theorem 2.2]). □

Define an $(m+1) \times n$ matrix

$$\bar{A} = \begin{pmatrix} A \\ c \end{pmatrix}. \tag{4.2}$$

Also define the polytope

$$\boldsymbol{Z} = \{z \mid z = \bar{A}x,\ l \leq x \leq u\}. \tag{4.3}$$

Note that polytope $\boldsymbol{Z}$ is, what is called, a zonotope (with possible translation).

The LP problem (4.1) can be reformulated as follows.

$$\begin{array}{lll} (\mathrm{LP})' & \text{Maximize} & \gamma \\ & \text{subject to} & \begin{pmatrix} b \\ \gamma \end{pmatrix} \in \boldsymbol{Z}, \end{array} \tag{4.4}$$

where γ is a scalar variable in $\mathbf{R}$.

Remark: Define

$$L = \left\{ \begin{pmatrix} b \\ \gamma \end{pmatrix} \;\middle|\; \gamma \in \mathbf{R} \right\}. \tag{4.5}$$

Note that L is a line parallel to the last coordinate space and that Problem $(\mathrm{LP})'$ is to find the point of maximum last coordinate in the intersection of line L and zonotope $\boldsymbol{Z}$. □

For any $\bar{c} \in (\mathbf{R}^{n+1})^*$ an optimal solution $\hat{z}$ of the problem of minimizing

$$\bar{c}z = \sum_{i=1}^{n+1} \bar{c}(i)z(i) \tag{4.6}$$

over zonotope $\boldsymbol{Z}$ in (4.3) can easily be computed as $\hat{z} = \bar{A}x$ with $d = \bar{c}\bar{A}$ and

$$x(j) = \begin{cases} u(j) & \text{if } d(j) < 0 \\ l(j) & \text{otherwise} \end{cases} \qquad (j = 1, 2, \cdots, n). \tag{4.7}$$

Therefore, the minimum-norm-point algorithm works for $\boldsymbol{Z}$.

4.2. The LP-Newton algorithm

In order to solve Problem $(\mathrm{LP})'$ (or Problem (LP)) we introduce a new optimization method which we call the *LP-Newton algorithm*. It consists of repeated minimum-norm-point algorithms as follows.

The LP-Newton Algorithm LPN

Input: Data A, b, c, l, u for Problem (LP).

Output: An optimal solution x^* of Problem (LP) or decision that Problem (LP) is infeasible.

Step 1: Compute

$$x(j) = \begin{cases} u(j) & \text{if } c(j) > 0 \\ l(j) & \text{otherwise} \end{cases} \tag{4.8}$$

for each $j = 1, 2, \cdots, n$. Put $\gamma := cx$.

Step 2: Put $\bar{b} := (b^{\mathrm{T}}, \gamma)^{\mathrm{T}}$ (where T denotes the matrix transpose). By using the minimum-norm-point algorithm find the point z in $\boldsymbol{Z}$ that is the nearest to $\bar{b}$, where z is expressed as a convex combination of affinely independent extreme points y_k $(k \in K)$ of $\boldsymbol{Z}$, i.e., $z = \sum_{k \in K} \lambda_k y_k$ (with $\sum_{k \in K} \lambda_k = 1$ and $\lambda_k > 0$ $(k \in K)$) and each y_k is given by $y_k = \bar{A} x_k$ with $l \leq x_k \leq u$ $(k \in K)$. Let $z = (\tilde{z}^{\mathrm{T}}, \zeta)^{\mathrm{T}}$.
If $z = \bar{b}$, then put $x^* = \sum_{k \in K} \lambda_k x_k$, return x^*, and halt;
else if $\zeta \geq \gamma$, then return 'Problem (LP) is infeasible' and halt.

Step 3: Compute $\gamma := \{(\tilde{z} - b)^{\mathrm{T}} b - (z - \bar{b})^{\mathrm{T}} z)\}/(\gamma - \zeta)$.
Go to Step 2.
(End)

Note that the initial x computed in Step 1 corresponds to $\bar{c} = (0, \cdots, 0, 1)$ in (4.6).

If the algorithm does not halt at the end of an execution of Step 2, we have a hyperplane H_z expressed by

$$(z - \bar{b})^{\mathrm{T}} (y - z) = 0 \tag{4.9}$$

in a variable vector y in $\mathbf{R}^{n+1}$. The new γ computed in Step 3 gives the point $\bar{b} = (b^{\mathrm{T}}, \gamma)^{\mathrm{T}}$ that is the intersection point of L and H_z (see Figure 7).

In Figure 7, z_0, z_1, z_2, z_3 represent the sequence of zs computed in Step 2, and $\bar{b}_0$, $\bar{b}_1$, $\bar{b}_2$=z_3 that of $\bar{b}$s.

Computational results about the behavior of the LP-Newton algorithm for linear programming are given in Section 4.3.

Remark: Linear programming problems can be reduced to finding feasible solutions in systems of linear inequalities, and the latter problems can further be reduced to minimum-norm-point problems for zonotopes. Here we are, however, interested in solving optimization problems directly. □

Remark: We can consider a metric other than the Euclidean one, such as $(\bar{A}\bar{A}^{\mathrm{T}})^{-1}$ when $\bar{A}$ is of row-full rank. It is left for future work to choose an appropriate metric in the space of the zonotope. □

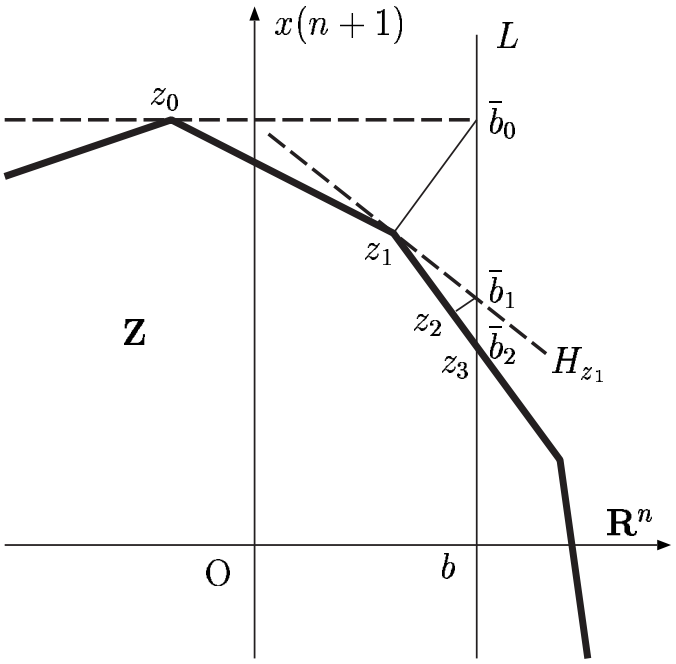

Figure 7: An illustration of the LP-Newton algorithm ($z_3 = \bar{b}_2$).

4.3. Computational results

Consider Problem

$$\begin{array}{lll} \text{(LP)} & \text{Maximize} & cx = \sum_{j=1}^{n} c(j)x(j) \\ & \text{subject to} & Ax = b, \\ & & l \leq x \leq u, \end{array}$$

in (4.1). We generate at random each component $a(i,j)$ of A uniformly from $[0,1]$, $b(i)$ of b from $[10,11]$, and $c(j)$ of c from $[-\frac{1}{2},\frac{1}{2}]$. We set $l = \mathbf{0}$ and $u(j) = 10$ $(j = 1,2,\cdots,n)$.

We programmed our LP-Newton algorithm (LPN) by MATLAB. We carried out computational experiments of LPN for LP on Sun Fire V440 with SPARC/Solaris 10(3/05), CPU 1.6GHz$\times$4, 8GB of memory, using MATLAB version 7.1.0.183 (R14) Service Package 3.

Table 7 shows 10-run averages of the running time of LPN, the number of the Newton steps (Step 2) of LPN, and the number of generated extreme points of zonotope $\boldsymbol{Z}$. Note that the major part of Step 2 of LPN is to carry out the minimum-norm-point procedure of Wolfe and that the number of generated extreme points of $\boldsymbol{Z}$ is equal to the total number

of executed major cycles of the minimum-norm-point procedure. (The last row of Table 7 gives the 10-run averages of the running time of the LP solver linprog available within the MATLAB package, for reference.) We observe that the running time of LPN depends on m but seems indifferent to values of n. The number of the Newton steps (Step 2) of LPN is relatively small and increases very slowly with respect to m.

Table 8 shows sample behaviors of objective function values computed by LPN. The objective function values converge to optimal values very quickly, as expected.

Table 7: Results for Algorithm LPN

Averaged running time, number of steps, and number of generated extreme points									
m	10	10	10	50	50	50	100	100	100
n	200	350	500	200	350	500	200	350	500
time (sec)	0.047	0.047	0.051	1.52	1.36	2.23	26.10	17.82	18.97
# Newton steps	3.80	3.90	3.90	5.70	5.20	5.00	9.00	6.90	6.10
# extreme points	8.1	8.8	8.7	65.2	52.4	48.6	418.7	274.4	305.7
linprog (sec)	0.225	0.137	0.186	0.30	0.48	0.74	0.79	1.38	2.03

Table 8: Sample behaviors of LPN

Iteration vs. (objective function value − optimal value)							
iteration	1	2	3	4	5	6	7
$(m, n) = (10, 500)$	618.89	6.458	0.0024	0.0	—	—	—
$(m, n) = (50, 500)$	585.67	1.948	0.0206	0.00004	0.0	—	—
$(m, n) = (100, 500)$	595.95	3.454	0.1706	0.00055	0.00002	0.0	—

5. Concluding Remarks

The computational results on submodular function minimization (SFM) have shown that the minimum-norm-point SFM algorithm FW runs very fast, and suggest that FW is strongly polynomial. It is, however, open to determine the complexity of FW for SFM.

We have also proposed a new algorithm LPN for LP by means of the minimum-norm-point algorithm and the LP-Newton algorithm. The present results indicate that the number of the Newton steps is small and that the computation time is seemingly indifferent to the dimension n of the original variable vector space. The running time of LPN implemented by MATLAB is, however, far from competitive with existing commercial codes such as CPLEX. Our computational experiments are only preliminary and require much more further to examine its behavior when implemented, say, by C or C++, which will be left for future work.

Acknowledgements

We are grateful to Satoru Iwata for providing us with his programs of the SFM algorithms, and to Hiroshi Hirai for his help in carrying out computational experiments for LP. The present research was supported partly by a Grant-in-Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan and by Japan International Cooperation Agency.

References

[1] A. Ben-Tal and A. Nemirovski: *Lectures on Modern Convex Optimization—Analysis, Algorithms, and Engineering Applications* (MPS/SIAM Series on Optimization) (SIAM, 2001).

[2] J. Edmonds: Submodular functions, matroids, and certain polyhedra. *Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications* (R. Guy, H. Hanani, N. Sauer and J. Schönheim, eds., Gordon and Breach, New York, 1970), pp. 69–87; also in: *Combinatorial Optimization—Eureka, You Shrink!* (M. Jünger, G. Reinelt, and G. Rinaldi, eds., Lecture Notes in Computer Science **2570**, Springer, Berlin, 2003), pp. 11–26.

[3] L. Fleischer and S. Iwata: A push-relabel framework for submodular function minimization and applications to parametric optimization. *Discrete Applied Mathematics* **131** (2003) 311–322.

[4] S. Fujishige: Submodular systems and related topics. *Mathematical Programming Study* **22** (1984) 113–131.

[5] S. Fujishige: Submodularity and polyhedra. 4th Japanese-Hungarian Symposium on Discrete Mathematics and Its Applications (Budapest, June 3–6, 2005).

[6] S. Fujishige: *Submodular Functions and Optimization*, (Second Edition) (Annals of Discrete Mathematics **58**) (Elsevier, Amsterdam, 2005).

[7] B. von Hohenbalken: A finite algorithm to maximize certain pseudoconcave functions on polytopes. *Mathematical Programming* **8** (1975) 189–206.

[8] S. Isotani and S. Fujishige: Submodular function minimization: Computational experiments. Unpublished manuscript, 2003.

[9] S. Iwata: A faster scaling algorithm for minimizing submodular functions. *SIAM Journal on Computing* **32** (2003) 833–840.

[10] S. Iwata, L. Fleischer, and S. Fujishige: A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of ACM* **48** (2001) 761–777.

[11] S. T. McCormick: Submodular function minimization. In: *Discrete Optimization* (Handbooks in Operations Research and Management Science **12**) (K. Aardal, G. L. Nemhauser, and R. Weismantel, eds., Elsevier, Amsterdam, 2005), Chapter 7, pp. 321–391.

[12] C. H. Papadimitriou and K. Steiglitz: *Combinatorial Optimization—Algorithms and Complexity* (Prentice-Hall, New Jersey, 1982).

[13] A. Schrijver: A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory*, Ser. B **80** (2000) 346–355.

[14] J. Vygen: A note on Schrijver's submodular function minimization algorithm. *Journal of Combinatorial Theory* **B88** (2003) 399–402.

[15] P. Wolfe: Finding the nearest point in a polytope. *Mathematical Programming* **11** (1976) 128–149.

[16] G. M. Ziegler: *Lectures on Polytopes* (Graduate Texts in Mathematics **152**) (Springer, Berlin, 1995).

[17] The First DIMACS international algorithm implementation challenge: The core experiments, 1990. Available at ftp://dimacs.rutgers.edu/pub/netflow/general-info/core.tex .