社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－AL－96　(2)
2004／7／27

劣モジュラ多面体内直線探索問題に対する強多項式時間アルゴリズム

永野 清仁[1]

Abstract

本稿では劣モジュラ関数最小化の完全に組合せ的な強多項式時間アルゴリズムをもとにし，Megiddo により提案されたパラメトリック・サーチ法を枠組みとして用いることで，劣モジュラ多面体内の直線探索を行う強多項式時間アルゴリズムを与える．

A Strongly Polynomial Algorithm for Line Search in Submodular Polyhedra

Kiyohito NAGANO[1]

Abstract

This paper presents a strongly polynomial time algorithm for line search in submodular polyhedra with the aid of a fully combinatorial algorithm for submodular function minimization as a subroutine. The algorithm is based on the parametric search method proposed by Megiddo.

1 Introduction

Let V be a finite set with $|V| = n$. Let f be a *submodular function* on the subsets of V, that is,

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y), \quad \forall X, Y \subseteq V. \tag{1}$$

Iwata, Fleischer and Fujishige [6] and Schrijver [10] independently presented combinatorial, strongly polynomial time algorithms for submodular function minimization. Iwata [4] presented a *fully combinatorial* algorithm, which uses only additions, subtractions, comparisons, and the oracle calls for function values. For a vector $x \in \mathbf{R}^V$ and $u \in V$, we denote by $x(u)$ the component of x on u. For a submodular function $f : 2^V \to \mathbf{R}$ with $f(\emptyset) = 0$, the *submodular polyhedron* $\mathbf{P}(f)$ is defined by

$$\mathbf{P}(f) \quad = \quad \{\, x \in \mathbf{R}^V \,|\, x(X) \leq f(X) \ (\forall X \subseteq V) \,\}, \tag{2}$$

where $x(X) = \sum_{u \in X} x(u)$. In this paper we consider the following problem:

Problem　Line Search in Submodular Polyhedra (LSSP)
Instance:　A submodular function $f : 2^V \to \mathbf{R}$ with $f(\emptyset) = 0$, $x_0 \in \mathbf{P}(f)$ and $a \in \mathbf{R}^V$.
Task:　Find $t^* = \max\{\, t \in \mathbf{R} \,|\, x_0 + ta \in \mathbf{P}(f)\}$.

An example of Problem LSSP is illustrated in Fig. 1.

If $a \in \mathbf{R}^V_-$, Problem LSSP does not have an optimal solution. Hence throughout we assume that $a \notin \mathbf{R}^V_-$. We may assume $f(X) \geq 0$, $\forall X \subseteq V$, and $x_0 = \mathbf{0}$, by resetting $f(X) := f(X) - x_0(X)$ for all $X \subseteq V$. So throughout we assume that f is nonnegative, $f(\emptyset) = 0$ and $x_0 = \mathbf{0}$. It is easy to see that the optimal value t^* of Problem LSSP is equal to $\min\{f(X)/a(X) \,|\, X \subseteq V, a(X) > 0\}$. So Problem LSSP can be regarded as a minimum-ratio problem.

The Newton method (Section 3) is a simple approach to Problem LSSP. If a is nonnegative, it is shown that the number of iterations of the Newton method is at most $n + 1$ and Problem LSSP can be solved in strongly polynomial time. (See Fujishige [2, §7.2] for details.) If $a \in \mathbf{R}^V$and $a \notin \mathbf{R}^V_+$, however, only a weakly polynomial running time bound is given, and it is left open to verify if the Newton method for Problem LSSP runs in strongly polynomial time.

[1]東京大学大学院 情報理工学系研究科 数理情報学専攻, Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo. E-mail: nagano@simplex.t.u-tokyo.ac.jp

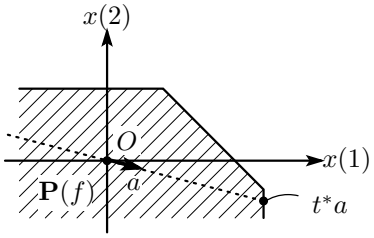

Figure 1: Problem LSSP ($n = 2$)

In this paper, we propose an algorithm for Problem LSSP, which is quite different from the Newton method. The algorithm uses a fully combinatorial algorithm for submodular function minimization [4, 5] as a subroutine, within the framework of the parametric search method proposed by Megiddo [7]. It solves Problem LSSP in strongly polynomial time.

The paper is organized as follows. In Section 2, we provide preliminaries for the following sections. In Section 3, we describe the Newton method using an algorithm for submodular function minimization as a subroutine. In Section 4, we present a strongly polynomial time algorithm for Problem LSSP using a fully combinatorial algorithm for submodular function minimization as a subroutine within the framework of the parametric search method proposed by Megiddo.

2 Preliminaries

Let V be a finite nonempty set and $|V| = n$. A family $\mathcal{D} \subseteq 2^V$ is said to be a *ring family* if it satisfies

$$X, Y \in \mathcal{D} \Rightarrow X \cup Y,\ X \cap Y \in \mathcal{D}.$$

Let $f : 2^V \to \mathbf{R}$ be a submodular function and let $\arg\min f$ denote a family of all the minimizers of f. It is not difficult to see that $\arg\min f$ forms a ring family. Suppose that $X, Y \in \arg\min f$ and $f(X) = f(Y) = \alpha$. Then, using submodularity (1), we have $f(X \cup Y) + f(X \cap Y) \leq 2\alpha$. Since $f(X \cup Y) \geq \alpha$ and $f(X \cap Y) \geq \alpha$, this implies $f(X \cup Y) = f(X \cap Y) = \alpha$, that is, $X \cup Y,\ X \cap Y \in \arg\min f$. As $\arg\min f$ is closed under union and intersection, there exists a *minimal minimizer* $X_{\mathsf{min}} = \bigcap\{\, X \mid X \in \arg\min f\} \in \arg\min f$ and exists a *maximal minimizer* $X_{\mathsf{max}} = \bigcup\{\, X \mid X \in \arg\min f\} \in \arg\min f$.

Let $f : 2^V \to \mathbf{R}$ be a submodular function with $f(\emptyset) = 0$ and let $x \in \mathbf{P}(f)$. A subset $X \subseteq V$ is said to be a tight set at x if $x(X) = f(X)$. We denote the family of tight sets at x by $\mathcal{D}(x)$. Namely,

$$\mathcal{D}(x) = \{\, X \subseteq V \mid x(X) = f(X)\}.$$

For any $y \in \mathbf{R}^V$, a function $f_y : 2^V \to \mathbf{R}$ defined by $f_y(X) = f(X) - y(X)$ $(X \subseteq V)$ is obviously a submodular function. As $x \in \mathbf{P}(f)$, $f_x(X) \geq 0\,,\ \forall X \subseteq V$, and $f_x(\emptyset) = 0$. Thus the minimum value of f_x is 0, which implies for any $X \subseteq V$,

$$X \in \mathcal{D}(x) \iff X \in \arg\min f_x\,.$$

So $\arg\min f_x = \mathcal{D}(x)$, therefore $\mathcal{D}(x)$ forms a ring family. Note that $\emptyset \in \mathcal{D}(x)$.

Let U be a finite set. A function $g : \mathcal{D} \to \mathbf{R}$ is said to be a *modular function* on a ring family $\mathcal{D} \subseteq 2^U$ if it satisfies

$$g(X) + g(Y) = g(X \cup Y) + g(X \cap Y), \quad \forall X, Y \in \mathcal{D}.$$

For a vector $b \in \mathbf{R}^U$ we denote $b(X) = \sum_{u \in X} b(u)$ for all $X \subseteq U$, so b can be regarded as a modular function on 2^U. For a ring family $\mathcal{D}$, a function $b_{\mathcal{D}} : \mathcal{D} \to \mathbf{R}$ defined by

$$b_{\mathcal{D}}(X) = \sum_{u \in X} b(u) \quad (X \in \mathcal{D}) \tag{3}$$

is a modular function on $\mathcal{D}$.

As an instance of Problem LSSP, without loss of generality, we assume that f is nonnegative, $f(\emptyset) = 0$, $x_0 = \mathbf{0}$, and $a \notin \mathbf{R}^V_-$. We explain that the optimal value t^* of LSSP(f, $\mathbf{0}$, a) is nonnegative and finite. The optimal value of LSSP(f, $\mathbf{0}$, a) is

$$t^* = \max\{t \,|\, ta \in \mathbf{P}(f)\}. \tag{4}$$

Since $\mathbf{0} \in \mathbf{P}(f)$, t^* is nonnegative. Let $A \subseteq V$ be a subset which satisfies $a(A) > 0$. If $t > f(A)/a(A)$, then $ta(A) > f(A)$ and hence $ta \notin \mathbf{P}(f)$. So $t^* \leq f(A)/a(A)$, therefore t^* is finite.

For any $t \in \mathbf{R}$ we consider deciding whether $ta \in \mathbf{P}(f)$ or $ta \notin \mathbf{P}(f)$. Since, for any $x \in \mathbf{R}^V$, $f(\emptyset) - x(\emptyset) = 0$, we have

$$\begin{aligned} x \in \mathbf{P}(f) \quad &\Longleftrightarrow \quad f_x(X) = f(X) - x(X) \geq 0, \ \forall X \subseteq V\,, \\ &\Longleftrightarrow \quad \min\{f_x(X) \,|\, X \subseteq V\} = 0, \end{aligned}$$

and if x can be represented as ta, using $ta(\emptyset) = 0$,

$$\begin{aligned} ta \in \mathbf{P}(f) &\iff \min\{f_{ta}(X) \,|\, X \subseteq V\} = 0, \\ ta \notin \mathbf{P}(f) &\iff \min\{f_{ta}(X) \,|\, X \subseteq V\} < 0. \end{aligned} \tag{5}$$

So we can decide whether $ta \in \mathbf{P}(f)$ or $ta \notin \mathbf{P}(f)$ by minimizing f_{ta}.

Now, for any $t \geq 0$, let us consider the conditions of "$t < t^*$", "$t = t^*$" and "$t > t^*$". See Figure 2 to understand each condition intuitively.

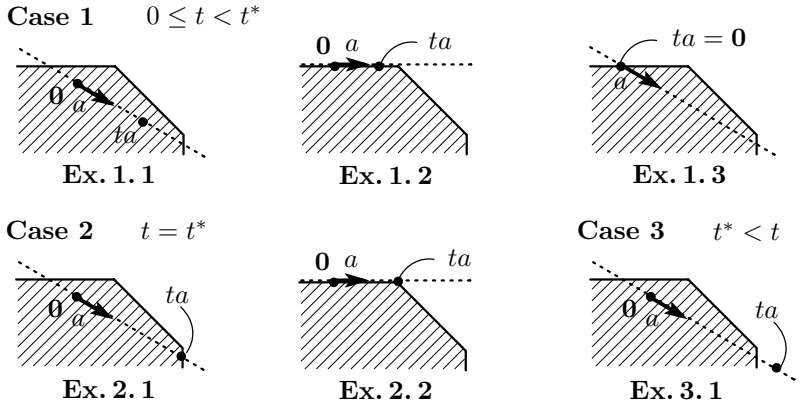

Figure 2: Relation between t and t^*

It follows from (4), (5), and the convexity of $\mathbf{P}(f)$ that

$$\begin{aligned} 0 \leq t \leq t^* \quad &\Longleftrightarrow \quad t \geq 0 \ \text{ and } \ \min\{f_{ta}(X) \,|\, X \subseteq V\} = 0, \\ t^* < t \quad &\Longleftrightarrow \quad t \geq 0 \ \text{ and } \ \min\{f_{ta}(X) \,|\, X \subseteq V\} < 0. \end{aligned} \tag{6}$$

The condition (6) is not sufficient to compare t with t^*, because we cannot decide whether $t < t^*$ or $t = t^*$. We consider the condition of $t = t^*$. Remark that $t = t^*$ and $ta \in \partial\mathbf{P}(f)$ are not equivalent (see Ex. 1. 2 and Ex. 1. 3 in Figure 2), where $\partial\mathbf{P}(f)$ is a "boundary" of $\mathbf{P}(f)$, that is, $\partial\mathbf{P}(f) = \{\, x \in \mathbf{P}(f) \,|\, \exists X \in 2^V \backslash \{\, \emptyset \,\} \text{ s.t. } x(X) = f(X)\}$. Equation (4) directly implies that

$$\begin{aligned} t = t^* \quad &\Longleftrightarrow \quad ta \in \mathbf{P}(f), \ \forall \varepsilon > 0 \ (t + \varepsilon)a \notin \mathbf{P}(f), \\ &\Longleftrightarrow \quad ta \in \mathbf{P}(f), \ \exists X \subseteq V \text{ s.t. } \forall \varepsilon > 0 \ \ \varepsilon a(X) > f_{ta}(X) (\geq 0), \\ &\Longleftrightarrow \quad ta \in \mathbf{P}(f), \ \exists X \in \mathcal{D}(ta) \ \text{ s.t. } \ a(X) > 0, \\ &\Longleftrightarrow \quad ta \in \mathbf{P}(f), \ \max\{a(X) \,|\, X \in \mathcal{D}(ta)\} > 0. \end{aligned} \tag{7}$$

For $ta \in \mathbf{P}(f)$, $\mathcal{D}(ta)$ always includes $\emptyset$, so $\max\{a(X) \,|\, X \in \mathcal{D}(ta)\} \geq 0$. Thus using (6) and (7), we

obtain

$$
\begin{aligned}
0 \le t < t^* \quad &\Longleftrightarrow \quad \begin{cases} t \ge 0, \\ \min\{f_{ta}(X) \,|\, X \subseteq V\} = 0, \\ \max\{a(X) \,|\, X \in \mathcal{D}(ta)\} = 0, \end{cases} \\
t = t^* \quad &\Longleftrightarrow \quad \begin{cases} t \ge 0, \\ \min\{f_{ta}(X) \,|\, X \subseteq V\} = 0, \\ \max\{a(X) \,|\, X \in \mathcal{D}(ta)\} > 0, \end{cases} \\
t^* < t \quad &\Longleftrightarrow \quad \begin{cases} t \ge 0, \\ \min\{f_{ta}(X) \,|\, X \subseteq V\} < 0. \end{cases}
\end{aligned} \tag{8}
$$

3 The Newton method for Problem LSSP

The Newton method is a simple approach to Problem LSSP with weakly polynomial running time bound. It is left open to verify if the Newton method runs in strongly polynomial time.

The Newton method for Problem LSSP uses an algorithm for submodular function minimization as a subroutine. Let $f : 2^V \to \mathbf{R}$ be a submodular function and $|V| = n$. Let γ denote the upper bound on the time to compute the function value of f. Combinatorial strongly polynomial time algorithms for submodular function minimization are given independently by Iwata, Fleischer and Fujishige (IFF) [6] and Schrijver [10]. Iwata [5] described an improved variant of the IFF algorithm and this algorithm achieves the best known bound on the running time, $\mathrm{O}(\gamma(n^6 \log n) + n^7 \log n)$.

Let Algorithm SFM be some algorithm which finds a minimizer of a submodular function $f : 2^V \to \mathbf{R}$ with $\mathrm{O}(\mathcal{T}^{\mathsf{O}}(n))$ oracle calls for function evaluation and $\mathrm{O}(\mathcal{T}^{\mathsf{A}}(n))$ arithmetic operations where $\mathcal{T}^{\mathsf{O}}(n)$ and $\mathcal{T}^{\mathsf{A}}(n)$ are some polynomials in n, for example, $\mathcal{T}^{\mathsf{O}}(n) = n^6 \log n$ and $\mathcal{T}^{\mathsf{A}}(n) = n^7 \log n$. For simplicity, we assume $n\,\mathcal{T}^{\mathsf{O}}(n) = \mathrm{O}(\mathcal{T}^{\mathsf{A}}(n))$. Let $\mathcal{T}(n) = \gamma\,\mathcal{T}^{\mathsf{O}}(n) + \mathcal{T}^{\mathsf{A}}(n)$. The running time of Algorithm SFM is $\mathrm{O}(\mathcal{T}(n))$.

Algorithm SFM (Submodular Function Minimization)
Input: A submodular function $f : 2^V \to \mathbf{R}$.
Output: A minimizer of f.
Operation: Oracle calls for function evaluation, arithmetic operations.
Running Time: $\mathrm{O}(\mathcal{T}(n))$ $(\,\mathcal{T}(n) = \gamma\,\mathcal{T}^{\mathsf{O}}(n) + \mathcal{T}^{\mathsf{A}}(n)\,)$.

We define a function $h : \mathbf{R} \to \mathbf{R}$ as

$$
h(t) = \min_{X \subseteq V}\{f_{ta}(X)\} = \min_{X \subseteq V}\{f(X) - ta(X)\}\,. \tag{9}
$$

It is obvious that h is a concave function. As $\mathbf{0} \in \mathbf{P}(f)$, $h(0) = 0$. Since $f_{ta}(\emptyset) = 0$ for any $t \in \mathbf{R}$, $h(t) \le 0$ for any $t \in \mathbf{R}$. Using (4), (5) and (9), we have $t^* = \max\{\, t \in \mathbf{R} \,|\, h(t) = 0\}$. The graph of h is illustraited in Figure 3 by a thick curve.

For any $t \in \mathbf{R}$ we can obtain the value $h(t)$ by running SFM$(f - ta)$. For each $v \in V$ we compute $ta(v)$ in advance. A function evaluation of $f - ta$ needs a function evaluation of f and at most n subtractions. Thus the time complexity of one iteration in the Newton method is $\mathrm{O}((\gamma + n)\,\mathcal{T}^{\mathsf{O}}(n) + \mathcal{T}^{\mathsf{A}}(n))$. Since $n\,\mathcal{T}^{\mathsf{O}}(n) = \mathrm{O}(\mathcal{T}^{\mathsf{A}}(n))$, $f - ta$ can be minimized in $\mathrm{O}(\mathcal{T}(n))$ time.

The Newton method is described below. The process of Newton method is illustrated in Figure 3.

The Newton method for Problem LSSP
Step 0: Find a set $X_0 \subseteq V$ such that $a(X_0) > 0$. Set $t_1 := f(X_0)/a(X_0)$ $(\ge t^*)$. Set $i := 1$.
Step 1: Obtain $X_i \subseteq V$ such that $h(t_i) = f(X_i) - ta(X_i)$ by running SFM$(f - t_i a)$.
Step 2: If $h(t_i) = 0$, return $t^* := t_i$ and stop. If $h(t_i) < 0$ then set $t_{i+1} := f(X_i)/a(X_i)$ and $i := i + 1$. Go to Step 1.

If $a \in \mathbf{R}^V_+$, it is known that the number of iterations of the Newton method for Problem LSSP is at most n. (See Fujishige [2, §7.2] for details.) It is left open to verify if the Newton method for Problem LSSP runs in a strongly polynomial number of iterations. An analysis based on Radzik [9]

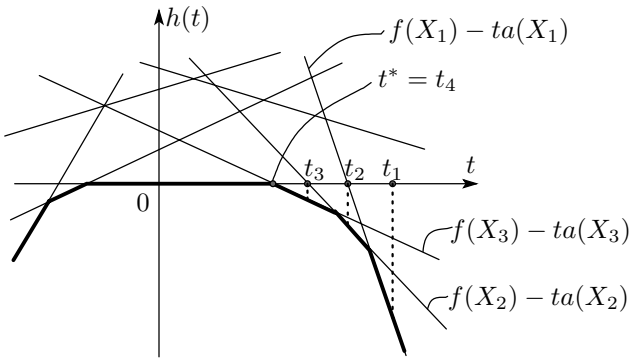

Figure 3: $h(t)$

gives a bound on the number of iterations of the Newton method for a special class of the LSSP problem with an integer-valued submodular function f and a integer vector a.

Theorem 3.1 *Let f be a integer-valued nonnegative submodular function, and let a be a integer vector which satisfies $a \notin \mathbf{R}^V_-$. If $\max_{X \subseteq V} |f(X)| \leq U_1$, $\max_{X \subseteq V} |a(X)| \leq U_2$, the Newton method for* LSSP$(f, \mathbf{0}, a)$ *runs in* $\mathrm{O}(\log U_1 + \log U_2)$ *iterations.*

4 A strongly polynomial algorithm

We present a combinatorial strongly polynomial time algorithm for Problem LSSP. We use a fully combinatorial strongly polynomial algorithm for submodular function minimization [4, 5] as a subroutine and the parametric search method proposed by Megiddo [7].

Framework

Later we will describe two procedures for Comparison with the Optimal Value; Procedure COV and Procedure L-COV. For any given nonnegative value $t \geq 0$, we can tell whether "$t < t^*$", "$t = t^*$" or "$t > t^*$" by running COV(t) in $\mathrm{O}(\gamma\, \mathcal{T}^{\mathsf{O}}_{\mathsf{COV}}(n) + \mathcal{T}^{\mathsf{A}}_{\mathsf{COV}}(n))$ time, where $\mathcal{T}^{\mathsf{O}}_{\mathsf{COV}}(n)$ and $\mathcal{T}^{\mathsf{A}}_{\mathsf{COV}}(n)$ are some polynomials in n. Procedure L-COV is a similar procedure. For any given $t \geq 0$, once $ta(v)$ is computed for each $v \in V$, it compares t to t^* with $\mathrm{O}(\mathcal{T}^{\mathsf{O}}_{\mathsf{L\text{-}COV}}(n))$ oracle calls for function evaluation of f, and $\mathrm{O}(\mathcal{T}^{\mathsf{FC}}_{\mathsf{L\text{-}COV}}(n))$ *fully combinatorial operations*, that is, additions, subtractions and comparisons, where $\mathcal{T}^{\mathsf{O}}_{\mathsf{L\text{-}COV}}(n)$ and $\mathcal{T}^{\mathsf{FC}}_{\mathsf{L\text{-}COV}}(n)$ are some polynomials in n. Moreover, if $t = t^*$, Procedure L-COV returns a subset $X \subseteq V$ such that $f(X) = t^* a(X)$ and $a(X) > 0$.

By running COV(0) we can tell whether $t^* = 0$ or $t^* > 0$. So we can assume that $t^* > 0$. If we knew the value of t^* and run L-COV(t^*), then it would return a subset $X \subseteq V$ s. t. $f(X) = t^* a(X)$ and $a(X) > 0$, that is, $t^* = f(X)/a(X)$. We try to run L-COV(t^*) *without knowing* the value of t^*. If we can run L-COV(t^*) successfully without knowing the value of t^*, we can obtain t^* by $f(X)/a(X)$ using $X \subseteq V$ s. t. $f(X) = t^* a(X)$ and $a(X) > 0$. The point is *how* to run L-COV(t^*) successfully without knowing the value of t^*. To achieve this goal, we use Megiddo's parametric search method.

Megiddo's parametric search

We give a strongly polynomial time algorithm for Problem LSSP using the parametric search technique of Megiddo [7]. We explain this technique in the following paragraphs.

Operations used in running L-COV(t^*) are additions, subtractions, comparisons, oracle calls for function evaluation of f, and only n multiplications to obtain $t^* a(v)$ for each $v \in V$. So each value which appears in running L-COV(t^*) can be represented as the form $p - qt^*$ where values p, q are known values and not functions of t^*. We consider trying to run L-COV(t^*) without knowing the value of t^* with all the values represented as linear functions of t^*. When values are represented as linear functions of t^*, each operation is done as follows:

$$
\begin{array}{lrcl}
\textit{An addition}: & (p_1 - t^*q_1) + (p_2 - t^*q_2) & \mapsto & (p_1 + p_2) - t^*(q_1 + q_2) \\
\textit{A subtraction}: & (p_1 - t^*q_1) - (p_2 - t^*q_2) & \mapsto & (p_1 - p_2) - t^*(q_1 - q_2) \\
\textit{A comparison}: & (p_1 - t^*q_1) \ ? \ (p_2 - t^*q_2) & \mapsto & ? \ = \ \text{`}>\text{'} \ \text{ or } \ \text{`}=\text{'} \ \text{ or } \ \text{`}<\text{'}
\end{array}
$$

Even though t^* is not known additions and subtractions do not change the assymptotic running time of the procedure. A comparison of two linear functions of t^* is not so easy as the other operations.

We now consider comparing two linear functions of t^*. Let p_1, p_2, q_1, q_2 be known values. Let us consider the comparison of $p_1 - t^*q_1$ and $p_2 - t^*q_2$. Setting $p = p_1 - p_2$, $q = q_1 - q_2$, we want to decide whether $p - t^*q > 0$, $p - t^*q = 0$ or $p - t^*q < 0$. Note that $t^* > 0$.

If $p\,q \leq 0$, it is easy to decide the sign of $p - t^*q$ using $t^* > 0$: $p = 0$ and $q = 0 \Longrightarrow p - t^*q = 0$, $p \geq 0$, $q \leq 0$, and $(p,\, q) \neq \mathbf{0} \Longrightarrow p - t^*q > 0$, $p \leq 0$, $q \geq 0$, and $(p,\, q) \neq \mathbf{0} \Longrightarrow p - t^*q < 0$.

Now let us assume that $p\,q > 0$. If $p > 0$ and $q > 0$, then $p/q > 0$, and hence $p - t^*q = 0 \iff p/q = t^*$, $p - t^*q > 0 \iff p/q > t^*$, $p - t^*q < 0 \iff p/q < t^*$. If $p < 0$ and $q < 0$, then $p/q > 0$ and hence $p - t^*q = 0 \iff p/q = t^*$, $p - t^*q > 0 \iff p/q < t^*$, $p - t^*q < 0 \iff p/q > t^*$. This analysis implies that we can obtain the sign of $p - t^*q$ if we can decide $p/q > t^*$, $p/q = t^*$ or $p/q < t^*$. So a comparison of two linear functions of t^* can be done by running Procedure COV if $p\,q > 0$. Thus we can run L-COV(t^*) successfully without knowing the value of t^*. We describe below Algorithm LSSP, which solves Problem LSSP within Megiddo's parametric search method.

Algorithm LSSP

Step 1: Decide whether "$t^* = 0$" or "$t^* > 0$" by running COV(0). If $t^* = 0$, then stop.

Step 2: Run L-COV(t^*) without knowing the value of t^* with all the values represented as linear functions of t^*. Each comparison of two linear functions of t^* encounterd during the computation can be evaluated (if necessary) by running Procedure COV. We can obtain $X \subseteq V$ s. t. $f(X) = t^*a(X)$ and $a(X) > 0$.

Step 3: Return $t^* := f(X)/a(X)$.

We will show that Algorithm LSSP solves Problem LSSP in strongly polynomial time.

Comparison of t with t^*

Now let us consider describing Procedure COV and Procedure L-COV using (8). As a preparation for describing them, we introduce four algorithms; Algorithm FC-SFM, Algorithm $\mathsf{SFM}_{\mathsf{min}}$, Algorithm $\mathsf{FC\text{-}SFM}_{\mathsf{min}}$, and Algorithm MFM.

An algorithm for submodular function minimization is said to be a *fully combinatorial* strongly polynomial time algorithm if the total number of oracle calls for function evaluation and *fully combinatorial operations*, that is, additions, subtractions and comparisons, is bounded by some polynomial in n. Iwata [4] presented a fully combinatorial strongly polynomial time algorithm for submodular function minimization as a variant of the IFF algorithm [6], and later, Iwata [5] described an improved algorithm, which runs in $\mathrm{O}(\gamma(n^8 \log^2 n) + n^9 \log^2 n)$ time.

Let Algorithm FC-SFM be some algorithm which finds a minimizer of a submodular function $f : 2^V \to \mathbf{R}$ with $\mathrm{O}(\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n))$ oracle calls for function evaluation of f and $\mathrm{O}(\mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n))$ fully combinatorial operations. For example, $\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n) = n^8 \log^2 n$ and $\mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n) = n^9 \log^2 n$. For simplicity, we assume $n\,\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n) = \mathrm{O}(\mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n))$. Let $\mathcal{T}_{\mathsf{FC}}(n) = \gamma\,\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n) + \mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n)$.

Algorithm FC-SFM (Fully Combinatorial algorithm for SFM)

Input:	A submodular function $f : 2^V \to \mathbf{R}$.
Output:	A minimizer of f.
Operation:	Oracle calls for function evaluation, *fully combinatorial* operations.
Running Time:	$\mathrm{O}(\mathcal{T}_{\mathsf{FC}}(n))$ $(\ \mathcal{T}_{\mathsf{FC}}(n) = \gamma\,\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n) + \mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n)\ )$.

Now we consider finding a minimal minimizer of f. It is known that the IFF algorithm [6] finds a maximal minimizer. And similarly, Iwata's combinatorial strongly polynomial time algorithm [5] and Iwata's fully combinatorial strongly polynomial time algorithm [4, 5], which are improved variants of the IFF algorithm, find maximal minimizers. If f is a submodular function, then a function f' defined as $f'(X) = f(V \backslash X)$ $(X \subseteq V)$ is also a submodular function. So we can construct a (fully)

combinatorial strongly polynomial algorithm which finds a minimal minimizer. Note that an oracle call for function evaluation of f' can be done in $O(\gamma + n)$ time.

Let Algorithm $\mathsf{SFM}_{\mathsf{min}}$ be some combinatorial strongly polynomial time algorithm which finds a minimal minimizer of a submodular function and let Algorithm $\mathsf{FC\text{-}SFM}_{\mathsf{min}}$ be some fully combinatorial strongly polynomial time algorithm which finds a minimal minimizer of a submodular function. For simplicity we assume the running time of $\mathsf{SFM}_{\mathsf{min}}$ is $O(\mathcal{T}(n))$ and that of $\mathsf{FC\text{-}SFM}_{\mathsf{min}}$ is $O(\mathcal{T}_{\mathsf{FC}}(n))$.

Algorithm $\mathsf{SFM}_{\mathsf{min}}$

Input: A submodular function $f : 2^V \to \mathbf{R}$.
Output: The minimal minimizer of f.
Operation: Oracle calls for function evaluation, arithmetic operations.
Running Time: $O(\mathcal{T}(n))$ $(\mathcal{T}(n) = \gamma\,\mathcal{T}^{\mathsf{O}}(n) + \mathcal{T}^{\mathsf{A}}(n))$.

Algorithm $\mathsf{FC\text{-}SFM}_{\mathsf{min}}$

Input: A submodular function $f : 2^V \to \mathbf{R}$.
Output: The minimal minimizer of f.
Operation: Oracle calls for function evaluation, *fully combinatorial* operations.
Running Time: $O(\mathcal{T}_{\mathsf{FC}}(n))$ $(\mathcal{T}_{\mathsf{FC}}(n) = \gamma\,\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n) + \mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n))$.

Let U be a finite set and let $\mathcal{D} \subseteq 2^U$ be a ring family. For a modular function $g : \mathcal{D} \to \mathbf{R}$ with $g(\emptyset) = 0$, g can be expressed as $g(X) = b(X)$, $\forall X \in \mathcal{D}$, using some vector $b \in \mathbf{R}^U$. Let $b \in \mathbf{R}^U$, and let us consider minimizing a modular function $b_{\mathcal{D}} : \mathcal{D} \to \mathbf{R}$ defined as (3). We can assume w.l.o.g. $\{\emptyset, U\} \subseteq \mathcal{D}$. We need to have some infomation on $\mathcal{D}$ in advance. We assume for each $v \in U$ the minimal set $M_v \in \mathcal{D}$ containing v is known. (This is enough information about $\mathcal{D}$. See, for example, Fujishige [2, §3.2].) Using a result of Picard [8] the modular function minimization problem can be reduced to the minimum cut problem of a network with $O(|U|)$ vertices in $O(|U|^2)$ time, and Cunningham [1] showed the equivalence between the modular function minimization problem and the minimum cut problem. Using, for example, the Goldberg-Tarjan algorithm [3] for solving the minimum cut problem, $b_{\mathcal{D}}$ can be minimized with $O(|U|^3)$ fully combinatorial operations. Let Algorithm MFM be some fully combinatorial strongly polynomial time algorithm which finds a minimizer of a modular function over a ring family $\mathcal{D} \subseteq 2^U$ with $\mathcal{T}_{\mathsf{MFM}}(|U|)$ fully combinatorial operations. For example, $\mathcal{T}_{\mathsf{MFM}}(|U|) = |U|^3$. We can assume $\mathcal{T}_{\mathsf{MFM}}(n) = O(\mathcal{T}(n))$ and $\mathcal{T}_{\mathsf{MFM}}(n) = O(\mathcal{T}_{\mathsf{FC}}(n))$.

Algorithm MFM (Modular Function Minimization)

Input: A vector $b \in \mathbf{R}^U$, and ring family $\mathcal{D} \subseteq 2^U$ with $\{\emptyset, U\} \subseteq \mathcal{D}$ ($\forall v \in U$, the minimal set $M_v \in \mathcal{D}$ containing v is known).
Output: A minimizer of $b_{\mathcal{D}}$.
Operation: *Fully combinatorial* operations.
Running time: $O(\mathcal{T}_{\mathsf{MFM}}(|U|))$.

We describe below Procedure COV, which decide, for any given nonnegative value $t \geq 0$, whether "$t < t^*$", "$t = t^*$" or "$t > t^*$" using conditions (8) directly. In Step 1, we examine whether $ta \in \mathbf{B}(f)$ or not. In Step 2, we obtain information about $\mathcal{D}(ta)$. Note that $\mathcal{D}(ta)$ always includes $\emptyset$ but not necessarily includes V. Hence, for some $v \in V$, there may not exist a subset X such that $v \in X$ and $X \in \mathcal{D}(ta)$. In Step 3, we maximize $a_{\mathcal{D}(ta)}$ and examine whether $t = t^*$ or not.

Procedure COV (Comparison with the Optimal Value)

Input: A nonnegative value $t \geq 0$.
Output: A decision whether "$t < t^*$", "$t = t^*$" or "$t > t^*$".
Operation: Oracle calls for function evaluation, arithmetic operations.

Step 1: Minimize f_{ta} by running $\mathsf{SFM}(f_{ta})$. If $\min\{f_{ta}(X) \mid X \subseteq V\} < 0$ then stop ($t > t^*$).

Step 2: For each $v \in V$, let $f_v : 2^{V\setminus\{v\}} \to \mathbf{R}$ be a submodular function defined by $f_v(X) = f_{ta}(X \cup \{v\})$ $(X \subseteq V\setminus\{v\})$. Find (if any) the minimal set $M_v \in \mathcal{D}(ta) = \arg\min f_{ta}$ containing v by running $\mathsf{SFM}_{\mathsf{min}}(f_v)$.

Step 3: Maximize $a_{\mathcal{D}(ta)} : \mathcal{D}(ta) \to \mathbf{R}$ by running $\mathsf{MFM}(-a, \mathcal{D}(ta))$.
If $\max\{a(X) \mid X \in \mathcal{D}(ta)\} = 0$ then stop ($t < t^*$).
If $\max\{a(X) \mid X \in \mathcal{D}(ta)\} > 0$ then return the maximizer of $a_{\mathcal{D}(ta)}$ and stop ($t = t^*$).

Let us consider the running time of Procedure COV. In Procedure COV we run Algorithm SFM once, Algorithm $\mathsf{SFM}_{\mathsf{min}}$ n times, and Algorithm MFM once. Note that for any given $X \subseteq V$ a function value $f_{ta}(X) = f(X) - \sum_{v \in X} ta(v)$ can be acquired by a function evaluation of $f(X)$ and at most n subtractions. (For each $v \in V$ we compute $ta(v)$ in advance.) So the running time of $\mathsf{SFM}(f_{ta})$ is $\mathrm{O}((\gamma + n)\,\mathcal{T}^{\mathsf{O}}(n) + \mathcal{T}^{\mathsf{A}}(n))$. Since $n\,\mathcal{T}^{\mathsf{O}}(n) = \mathrm{O}(\mathcal{T}^{\mathsf{A}}(n))$, f_{ta} can be minimized in $\mathrm{O}(\mathcal{T}(n))$ time. Thus, the total running time is $\mathrm{O}((n+1)\mathcal{T}(n) + \mathcal{T}_{\mathsf{MFM}}(n)) = \mathrm{O}(n\mathcal{T}(n))$. Let $\mathcal{T}^{\mathsf{O}}_{\mathsf{COV}}(n) = n\mathcal{T}^{\mathsf{O}}(n)$, $\mathcal{T}^{\mathsf{A}}_{\mathsf{COV}}(n) = n\mathcal{T}^{\mathsf{A}}(n)$, and let $\mathcal{T}_{\mathsf{COV}}(n) = n\mathcal{T}(n)$ $(= \gamma\mathcal{T}^{\mathsf{O}}_{\mathsf{COV}}(n) + \mathcal{T}^{\mathsf{A}}_{\mathsf{COV}}(n))$.

Let Procedure L-COV be a procedure which is obtained by replacing Algorithm SFM and Algorithm $\mathsf{SFM}_{\mathsf{min}}$ by Algorithm FC-SFM and Algorithm $\mathsf{FC\text{-}SFM}_{\mathsf{min}}$ respectively in Procedure COV. For any given $t \geq 0$, once $ta(v)$ is computed for each $v \in V$, Procedure L-COV compares t to t^* with $\mathrm{O}(\mathcal{T}^{\mathsf{O}}_{\mathsf{L\text{-}COV}}(n))$ oracle calls for function evaluation of f, and $\mathrm{O}(\mathcal{T}^{\mathsf{FC}}_{\mathsf{L\text{-}COV}}(n))$ fully combinatorial operations, where $\mathcal{T}^{\mathsf{O}}_{\mathsf{L\text{-}COV}}(n) = n\,\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n)$ and $\mathcal{T}^{\mathsf{FC}}_{\mathsf{L\text{-}COV}}(n) = n\,\mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n)$. And moreover if $t = t^*$, Procedure L-COV returns a subset $X \subseteq V$ s.t. $f(X) = t^* a(X)$ and $a(X) > 0$. Let $\mathcal{T}_{\mathsf{L\text{-}COV}}(n) = n\mathcal{T}_{\mathsf{FC}}(n)$ $(= \gamma\mathcal{T}^{\mathsf{O}}_{\mathsf{L\text{-}COV}}(n) + \mathcal{T}^{\mathsf{FC}}_{\mathsf{L\text{-}COV}}(n))$. The time complexity of Procedure L-COV is $\mathrm{O}(\mathcal{T}_{\mathsf{L\text{-}COV}}(n))$.

Complexity

Theorem 4.1 *Algorithm* LSSP *solves* LSSP(f, $\mathbf{0}$, a) *in strongly polynomial time.*

Proof The running time in Step 1 is $\mathrm{O}(\mathcal{T}_{\mathsf{COV}}(n))$. In Step 2, $\mathrm{O}(\mathcal{T}^{\mathsf{FC}}_{\mathsf{L\text{-}COV}}(n))$ comparisons of linear functions of t^* are evaluated and the running time of the other part is $\mathrm{O}(\mathcal{T}_{\mathsf{L\text{-}COV}}(n))$. So the total running time is $\mathrm{O}(\mathcal{T}_{\mathsf{COV}}(n) + \mathcal{T}_{\mathsf{L\text{-}COV}}(n) + \mathcal{T}^{\mathsf{FC}}_{\mathsf{L\text{-}COV}}(n)\mathcal{T}_{\mathsf{COV}}(n)) = \mathrm{O}(n\mathcal{T}_{\mathsf{FC}}(n) + n^2\,\mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n)\mathcal{T}(n))$. □

References

[1] W. H. Cunningham: Minimum cuts, modular functions, and matroid polyhedra. *Networks*, **15** (1985), pp. 205–215.

[2] S. Fujishige: *Submodular Functions and Optimization*. North-Holland, Amsterdam, 1991.

[3] A. V. Goldberg and R. E. Tarjan: A new approach to the maximum flow problem. *Journal of the ACM*, **35** (1988), pp. 921–940.

[4] S. Iwata: A fully combinatorial algorithm for submodular function minimization. *Journal of Combinatorial Theory (B)*, **84** (2002), pp. 203–212.

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

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

[7] N. Megiddo: Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, **4** (1979), pp. 414–424.

[8] J. C. Picard: Maximal closure of a graph and applications to combinatorial problems. *Management Science*, **22** (1976), pp. 1268–1272.

[9] T. Radzik: Fractional combinatorial optimization. In D. Z. Du and P. M. Pardalos, eds. , *Handbook of Combinatorial Optimization*, vol. 1, pp. 429–478, Kluwer Academic Publishers, Boston, 1998.

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