A Combinatorial, Strongly Polynomial-Time Algorithm for Minimizing Submodular Functions

Satoru Iwata *
Grad. School of Eng. Science
Osaka University
Toyonaka, Osaka 560-8531, Japan
iwata@sys.es.osaka-u.ac.jp

Lisa Fleischer †
Dept. of Ind. Eng. & Oper. Res.
Columbia University
New York, NY 10027, USA
lisa@ieor.columbia.edu

Satoru Fujishige ‡
Grad. School of Eng. Science
Osaka University
Toyonaka, Osaka 560-8531, Japan
fujishig@sys.es.osaka-u.ac.jp

Abstract

This paper presents the first combinatorial polynomial-time algorithm for minimizing submodular functions, answering an open question posed in 1981 by Grötschel, Lovász, and Schrijver. The algorithm employs a scaling scheme that uses a flow in the complete directed graph on the underlying set with each arc capacity equal to the scaled parameter. The resulting algorithm runs in time bounded by a polynomial in the size of the underlying set and the largest length of the function value. The paper also presents a strongly polynomial-time version that runs in time bounded by a polynomial in the size of the underlying set independent of the function value.

1. Introduction

A function f defined on all the subsets of a finite set V is called *submodular* if it satisfies

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

This paper presents the first combinatorial polynomial-time algorithm for minimizing general submodular functions, provided that an oracle for evaluating the function value is available. Without loss of generality, we assume $f(\emptyset) = 0$ throughout this paper. Because of close connections to convexity [8, 14, 25], submodular function minimization has been regarded as a fundamental problem in discrete optimization. Readers are referred to [10, 15, 25] for general background on submodular functions.

Submodular functions arise in various branches of mathematical engineering such as combinatorial optimization, game theory and information theory. Examples include the matroid rank function, the cut capacity function, and the entropy function. In each of these and other applications, the *submodular polyhedron* and the *base polyhedron* defined by

$$\begin{aligned} \mathrm{P}(f) &= \{x \mid x \in \mathbf{R}^V, \forall X \subseteq V : x(X) \leq f(X)\}, \\ \mathrm{B}(f) &= \{x \mid x \in \mathrm{P}(f), x(V) = f(V)\} \end{aligned} \tag{1.1}$$

often play an important role. Linear optimization problems over these polyhedra are efficiently solvable by the greedy algorithm of Edmonds [4].

Grötschel, Lovász, and Schrijver [17] revealed the polynomial-time equivalence between the optimization and separation problems in combinatorial optimization via the ellipsoid method. Since the separation problem for $\mathrm{P}(f)$ is equivalent to the submodular function minimization, they asserted that the minimizer of a submodular function can be found in polynomial time using the ellipsoid method. Later, they also devised a strongly polynomial-time algorithm for this problem also using the ellipsoid method [18], based on the connection to convexity [25]. However, the ellipsoid method is far from being efficient in practice, and is not combinatorial. Hence an efficient combinatorial algorithm for submodular function minimization has been desired for a long time.

A first step towards a combinatorial polynomial-time algorithm was taken by Cunningham [2, 3], who

*A part of this work is done while on leave at the Fields Institute, Toronto, Canada. Partly supported by Grants-in-Aid for Scientific Research from Ministry of Education, Science, Sports, and Culture of Japan.

†This work done while on leave at Center for Operations Research and Econometrics, Université catholique de Louvain, Belgium, and at the Fields Institute, Toronto, Canada. Partially supported by NSF grants INT-9902663 and EIA-9973858.

‡Partly supported by Grants-in-Aid for Scientific Research from Ministry of Education, Science, Sports, and Culture of Japan.

devised a strongly polynomial-time algorithm for the separation problem for matroid polyhedra as well as a pseudopolynomial-time algorithm for integer-valued submodular function minimization. Narayanan [27] improved the running time bound of the former algorithm and extended the applicability of the latter by introducing a rounding technique. Based on the minimum-norm base characterization of minimizers [12, 13], Sohoni [30] gave another combinatorial pseudopolynomial-time algorithm for submodular function minimization.

For the problem of minimizing a symmetric submodular function over proper nonempty subsets, Queyranne [28] presented a combinatorial strongly polynomial-time algorithm, extending the undirected minimum cut algorithm of Nagamochi and Ibaraki [26].

In this paper, we present a combinatorial polynomial-time algorithm for submodular function minimization. Our algorithm uses an augmenting path approach with reference to a convex combination of extreme points of the associated base polyhedron. Such an approach was first introduced by Cunningham for minimizing submodular functions that arise from the separation problem for matroid polyhedra [2]. This was adapted for general submodular function minimization by Bixby, Cunningham, and Topkis [1] and improved by Cunningham [3] to obtain the first combinatorial, pseudopolynomial time algorithm.

A fundamental tool in these algorithms is to move from one extreme point of the base polyhedron to an adjacent extreme point via an exchange operation that increases one coordinate and decreases another coordinate by the same quantity. This quantity is called the exchange capacity. These previous methods maintain a directed graph on the underlying set that represents the possible exchange operations. They are inefficient since the lower bound on the amount of each augmentation is too small. In traditional network flow problems, it is possible to surmount this difficulty by augmenting only on paths of sufficiently large capacity [6]. However, it has been difficult to adapt this scaling approach to work in the setting of submodular function minimization, mainly because the amount of augmentation is determined by exchange capacities multiplied by the convex combination coefficients. These coefficients can be as small as the reciprocal of the maximum absolute value of the submodular function.

To overcome this difficulty, we employ a scaling framework that uses the complete directed graph on the underlying set, letting the capacity of this arc set depend directly on our scaling parameter δ. The complete directed graph serves as a relaxation of the submodular function f to another submodular function f_δ defined by $f_\delta(X) = f(X) + \delta\,|X| \cdot |V \backslash X|$.

This additional network was introduced by Iwata [22] in the design of the first capacity scaling algorithm for the submodular flow problem of Edmonds and Giles [5]. It is also used in the cut canceling algorithm of Iwata, McCormick, and Shigeno [23]. However, a direct application of the scaling framework in [22] to the submodular function minimization does not resolve the above difficulty.

Incorporating ideas from [23], Fleischer, Iwata, and McCormick [7] improved the capacity scaling algorithm by introducing a method to augment on paths that consist only of flow arcs and do not contain arcs corresponding to possible exchange operations. Instead, exchange operations are performed during the search for a shortest augmenting path of sufficient capacity. Our work in the present paper employs this technique in [7] to develop a capacity scaling, augmenting path algorithm for submodular function minimization.

The running time of the resulting algorithm is weakly polynomial, i.e., bounded by a polynomial in the size of the underlying set and the largest length of the function value. Even under the assumption that the largest absolute value of the function is bounded by a constant, our scaling algorithm is faster than the best previous combinatorial, pseudopolynomial-time algorithm due to Cunningham [3].

We then modify our scaling algorithm to run in strongly polynomial time, i.e., in time bounded by a polynomial in the size of the underlying set, independently of the largest length of the function value. To make a weakly polynomial-time algorithm run in strongly polynomial time, Frank and Tardos [9] developed a generic preprocessing technique that is applicable to a fairly wide class of combinatorial optimization problems including the submodular flow problem and testing membership in matroid polyhedra. However, this framework does not readily apply to submodular function minimization. Instead, we establish a proximity lemma, and use it to devise a combinatorial algorithm that repeatedly detects either a new element contained in every minimizer, or a new ordered pair $(u, v) \in V$ with the property that any minimizer containing u also contains v. Our approach is based on the general technique originated by Tardos [32] in the design of the first strongly polynomial-time minimum cost flow algorithm.

There are some practical problems, in dynamic flows [20], facility location [31], and multi-terminal source coding [11, 19], where the polynomial-time solvability relies on a submodular function minimization routine. Goemans and Ramakrishnan [16] discussed a class of submodular function minimization problems over restricted families of subsets. Their solution is combinatorial modulo an oracle for submodular function minimization on distributive lattices. Our algorithm can be used to provide combinatorial, strongly polynomial-time algorithms for these problems.

This paper is organized as follows. Section 2 provides preliminaries on submodular functions. Section 3 presents a scaling algorithm for submodular function minimization, which runs in weakly polynomial time. Section 4 is devoted to the strongly polynomial-time algorithm. Finally, we discuss extensions in Section 5.

2. Preliminaries

We denote by $\mathbf{Z}$ and $\mathbf{R}$ the set of integers and the set of reals, respectively. Let V be a finite nonempty set of cardinality $|V| = n$. For each $u \in V$, we denote by χ_u the unit vector in $\mathbf{R}^V$ such that $\chi_u(v) = 1$ if $v = u$ and 0 otherwise.

Given a submodular function f with $f(\emptyset) = 0$ and its associated base polyhedron $\mathrm{B}(f)$ as defined in (1.1), we call a vector $x \in \mathrm{B}(f)$ a *base*. An extreme point of $\mathrm{B}(f)$ is called an *extreme base*. A fundamental step in submodular function minimization algorithms is to move from one base x to another base x' via an *exchange operation* that increases one coordinate while decreasing another coordinate by the same amount, e.g., $x' := x + \alpha(\chi_u - \chi_v)$. The maximum value of α that ensures $x' \in \mathrm{B}(f)$ is called the *exchange capacity*. More precisely, for any base $x \in \mathrm{B}(f)$ and any distinct $u, v \in V$ the exchange capacity is

$$\widetilde{\mathrm{c}}(x, u, v) = \max\{\alpha \mid \alpha \in \mathbf{R},\ x + \alpha(\chi_u - \chi_v) \in \mathrm{B}(f)\}. \tag{2.1}$$

The exchange capacity $\widetilde{\mathrm{c}}(x, u, v)$ can also be expressed as

$$\widetilde{\mathrm{c}}(x, u, v) = \min\{f(X) - x(X) \mid u \in X \subseteq V \backslash \{v\}\}. \tag{2.2}$$

In general, computing $\widetilde{\mathrm{c}}(x, u, v)$ is as hard as submodular function minimization, even when x is an extreme base. However, if x is an extreme base, then for special pairs of vertices u and v, the exchange capacity $\widetilde{\mathrm{c}}(x, u, v)$ can be computed with one function evaluation as follows.

Let $L = (v_1, \cdots, v_n)$ be a linear ordering of V. For any $j \in \{1, \cdots, n\}$, we define $L(v_j) = \{v_1, \cdots, v_j\}$. Given such a linear ordering, the greedy algorithm of Edmonds [4] computes an extreme base $y \in \mathrm{B}(f)$ associated with L as

$$y(v_j) := f(L(v_j)) - f(L(v_{j-1})) \qquad \forall j = 1, \cdots, n, \tag{2.3}$$

where $L(v_0) = \emptyset$. Any extreme base can be generated by applying the greedy algorithm to an appropriate linear ordering. Note that a linear ordering $L = (v_1, \cdots, v_n)$ generates an extreme base y if and only if $y(L(v_j)) = f(L(v_j))$ for $j = 1, \cdots, n$. The following lemma describes when we can efficiently compute $\widetilde{\mathrm{c}}(x, u, v)$.

Lemma 2.1: *Let $y \in \mathrm{B}(f)$ be an extreme base generated by a linear ordering L of V in which u immediately succeeds v. Let L' be the linear ordering obtained from L by interchanging u and v. Then the extreme base y' generated by L' satisfies*

$$y' = y + \beta(\chi_u - \chi_v) \tag{2.4}$$

with

$$\beta = f(L(u)\backslash\{v\}) - f(L(u)) + y(v). \tag{2.5}$$

Moreover, we have $\widetilde{\mathrm{c}}(y, u, v) = \beta$.

Proof. Equations (2.4) and (2.5) follow from the greedy algorithm (see (2.3)). By the definition (2.1) of the exchange capacity, we have $\beta \leq \widetilde{\mathrm{c}}(y, u, v)$. Since $y(L(u)) = f(L(u))$, it follows from equations (2.2) and (2.5) that $\beta \geq \widetilde{\mathrm{c}}(y, u, v)$. Thus we obtain $\beta = \widetilde{\mathrm{c}}(y, u, v)$. ■

We will use Lemma 2.1 to transform one extreme base into another and to update the corresponding linear ordering.

For any vector $x \in \mathbf{R}^V$, we denote by x^- the vector in $\mathbf{R}^V$ defined by $x^-(v) = \min\{0, x(v)\}$ for $v \in V$. The following fundamental lemma easily follows from a theorem of Edmonds [4] on the vector reduction of polymatroids.

Lemma 2.2: *For a submodular function $f : 2^V \to \mathbf{R}$ we have*

$$\max\{x^-(V) \mid x \in \mathrm{B}(f)\} = \min\{f(X) \mid X \subseteq V\}.$$

If f is integer-valued, then the maximizer x can be chosen from among integral bases. ■

In fact, we do not rely on this min-max relation, which can be viewed as a strong duality theorem, but on the weak duality: For any base $x \in \mathrm{B}(f)$ and any $X \subseteq V$ we have $x^-(V) \leq x(X) \leq f(X)$. In particular, the following immediate corollary of this statement is crucial in our scaling algorithm.

Corollary 2.3: *If f is integer-valued and $f(X) - x^-(V)$ is less than one for some $x \in \mathrm{B}(f)$ and $X \subseteq V$, then X minimizes f.*

3. A Scaling Algorithm

In this section, we describe a combinatorial algorithm for minimizing an integer-valued submodular function $f : 2^V \to \mathbf{Z}$ with $f(\emptyset) = 0$. We assume an evaluation oracle for the function value of f. Our algorithm is an augmenting path algorithm, embedded in a scaling framework.

The previous augmenting path algorithms for submodular function minimization [1, 2, 3] maintain a base $x \in \mathrm{B}(f)$ as a convex combination of extreme bases

$y_i \in \mathrm{B}(f)$, so that $x = \sum_{i \in I} \lambda_i y_i$. This is because in general, the base $x \in \mathrm{B}(f)$ that attains the maximum in the left hand side of the min-max relation in Lemma 2.2 may not be an extreme point of $\mathrm{B}(f)$. However, in order to use Lemma 2.1 to compute exchange capacities, it is necessary to deal with extreme bases. Roughly speaking, these previous algorithms use a directed graph with the arc set defined by the pairs of vertices consecutive in some linear ordering that generates some y_i. They seek to increase $x^-(V)$ by performing exchange operations along a path of arcs from vertices s with $x(s) < 0$ to vertices t with $x(t) > 0$. The algorithms stop with an optimal x when there are no more augmenting paths. The corresponding minimizer X is determined by the set of vertices reachable from vertices s with $x(s) < 0$. Our algorithm builds on ideas developed in these previous algorithms.

3.1. The Scaling Framework

The algorithm consists of scaling phases with a positive parameter δ. The algorithm starts with $\delta = M/n^2$, where M is an upper bound on $|f(X)|$, $X \subseteq V$, specified below. It then cuts δ in half at the beginning of each scaling phase, and ends with $\delta < 1/n^2$.

To adapt the augmenting path approach to this scaling framework, we use a complete directed graph on V with arc capacities that depend directly on our scaling parameter δ. Let $\varphi : V \times V \to \mathbf{R}$ be *skew-symmetric*, i.e., $\varphi(u,v) + \varphi(v,u) = 0$ for $u, v \in V$. The function φ can be regarded as a flow in the complete directed graph $G = (V, E)$ with the vertex set V and the arc set $E = V \times V$. The *boundary* $\partial\varphi : V \to \mathbf{R}$ is defined by

$$\partial\varphi(v) = \sum_{u \in V} \varphi(u,v), \qquad \forall\, v \in V. \tag{3.1}$$

That is, $\partial\varphi(v)$ is the net flow entering v. A flow φ is called *δ-feasible* if it satisfies capacity constraints $-\delta \leq \varphi(u,v) \leq \delta$ for every $u, v \in V$.

Instead of trying to maximize $x^-(V)$ directly, the algorithm uses $z = x - \partial\varphi$ and seeks to maximize $z^-(V)$, thereby increasing $x^-(V)$ via the δ-feasibility of φ. As mentioned in the introduction, this z can be viewed as a base in the base polyhedron of a relaxed submodular function $f_\delta(X) = f(X) + \delta\,|X| \cdot |V - X|$.

The algorithm adopts the idea of maintaining a base $x \in \mathrm{B}(f)$ as a convex combination extreme bases $y_i \in \mathrm{B}(f)$. For each index $i \in I$, the algorithm also maintains a linear ordering L_i that generates y_i.

The algorithm starts with an arbitrary linear ordering L on V and the extreme base $x \in \mathrm{B}(f)$ generated by L. For any $X \subseteq V$, we have $x^-(V) \leq x(X) \leq f(X) \leq \sum_{v \in V} \max\{0, f(\{v\})\}$. Hence we adopt $M = \max\{|x^-(V)|, \sum_v \max\{0, f(\{v\})\}\}$ as an upper bound on $|f(X)|$. In addition, the algorithm starts with the zero flow $\varphi = \mathbf{0}$. Thus, initially $z^-(V) = x^-(V) \geq -M$.

3.2. A Scaling Phase

Each δ-scaling phase maintains a δ-feasible flow φ, and uses the *residual graph* $G(\varphi) = (V, E(\varphi))$ with the arc set

$$E(\varphi) = \{(u,v) \mid u, v \in V,\, u \neq v,\, \varphi(u,v) \leq 0\}. \tag{3.2}$$

Intuitively, $E(\varphi)$ consists of the arcs through which we can augment the flow φ by δ without violating the capacity constraints, i.e., the relaxed constraints for $z \in \mathrm{B}(f_\delta)$.

A δ-scaling phase starts by preprocessing φ to make it δ-feasible. At the beginning of the δ-scaling phase, after δ is cut in half, the current flow φ is 2δ-feasible. The algorithm modifies each $\varphi(u,v)$ to be δ-feasible by setting $\varphi(u,v)$ to the closest value in the interval $[-\delta, \delta]$. This may decrease $z^-(V)$ for $z = x - \partial\varphi$ by at most $\binom{n}{2}\delta$.

The rest of the δ-scaling phase aims at increasing $z^-(V)$ by sending flow along paths in $G(\varphi)$ from $S = \{v \mid v \in V,\, z(v) \leq -\delta\}$ to $T = \{v \mid v \in V,\, z(v) \geq \delta\}$. Such a directed path is called a *δ-augmenting path*.

A key feature of the algorithm is what it does when there are no δ-augmenting paths. In this case, let W denote the set of vertices currently reachable from S in $G(\varphi)$. If there is a pair (u,v) of vertices $u \in W$, $v \notin W$ such that v immediately succeeds u in some L_i, then the algorithm performs an appropriate exchange operation, and modifies φ so that $z = x - \partial\varphi$ is invariant. This has the affect of reducing exchange capacity $\tilde{c}(y_i, u, v)$ while creating residual flow capacity on (u,v). We refer to this procedure as Double-Exchange(i,u,v). It is an extension of a subroutine introduced in [7]. The details of Double-Exchange are described below and in Figure 1. Note that Double-Exchange(i,u,v) may add v to W. Otherwise, W remains invariant. The algorithm performs Double-Exchange as long as it is applicable, until a δ-augmenting path is found. Once a δ-augmenting path is found, the algorithm augments the flow φ by δ through the path without changing x. As a consequence, $z^-(V)$ increases by δ. This is an extension of a technique developed in [7] for finding δ-augmenting paths for submodular flows. We give details below. A formal description of the algorithm appears in Figure 2.

We call a vertex $v \in V \backslash W$ *active* in ordering L_i if v is the last vertex in L_i among vertices in $V \backslash W$ that satisfy $W \backslash L_i(v) \neq \emptyset$. We call (i,v) an *active pair*, and denote by Z the set of the current active pairs.

If $W \cap T = \emptyset$, there is no δ-augmenting path in $G(\varphi)$. Then, as long as there is an active pair (i,v), that is $Z \neq \emptyset$, the algorithm repeatedly picks an active pair $(i,v) \in Z$ and applies Double-Exchange(i,u,v) to (i,v) and the vertex u that immediately succeeds v in L_i. Since v is active in L_i, we have that $u \in W$.

The first step of the routine Double-Exchange(i,u,v) is to determine the amount α of the exchange opera-

Double-Exchange(i,u,v):

$\alpha \leftarrow \min\{\delta,\ \lambda_i \widetilde{c}(y_i,u,v)\}$
If $\alpha < \lambda_i \widetilde{c}(y_i,u,v)$ **then**
  $k \leftarrow$ a new index
  $I \leftarrow I \cup \{k\}$
  $\lambda_k \leftarrow \lambda_i - \alpha/\widetilde{c}(y_i,u,v)$
  $\lambda_i \leftarrow \alpha/\widetilde{c}(y_i,u,v)$
  $y_k \leftarrow y_i$
  $L_k \leftarrow L_i$
$y_i \leftarrow y_i + \widetilde{c}(y_i,u,v)(\chi_u - \chi_v)$
Update L_i by interchanging u and v.
$x \leftarrow \sum_{i\in I} \lambda_i y_i \qquad [x \leftarrow x + \alpha(\chi_u - \chi_v)]$
$\varphi(u,v) \leftarrow \varphi(u,v) - \alpha.$
$\varphi(v,u) \leftarrow \varphi(v,u) + \alpha.$

Figure 1: Algorithmic description of the procedure Double-Exchange(i,u,v).

tion. Since Double-Exchange(i,u,v) will modify both x and φ, this amount is set to be the minimum of δ and $\lambda_i\widetilde{c}(y_i,u,v)$, where δ is an amount of feasible decrease of flow on (u,v), and $\lambda_i\widetilde{c}(y_i,u,v)$, is the maximum exchange possible to effect in $x = \sum_i \lambda_i y_i$ by performing an exchange operation on y_i and keeping all the other extreme bases in I fixed. The procedure Double-Exchange(i,u,v) is called *saturating* if $\alpha = \lambda_i\widetilde{c}(y_i,u,v)$. Otherwise, it is called *nonsaturating*. A nonsaturating Double-Exchange(i,u,v) adds to I a new index k with $y_k := y_i$, $\lambda_k := \lambda_i - \alpha/\widetilde{c}(y_i,u,v)$, and $L_k := L_i$. Whether the Double-Exchange(i,u,v) is saturating or not, it updates y_i as $y_i := y_i + \widetilde{c}(y_i,u,v)(\chi_u - \chi_v)$, $\lambda_i := \alpha/\widetilde{c}(y_i,u,v)$, and L_i by interchanging u and v, moving v toward the end of L_i. This fact is crucial to get a polynomial-time bound of the algorithm, which will be discussed in detail below. The effect of Double-Exchange(i,u,v) is to move the current base from x to $x+\alpha(\chi_u-\chi_v)$. Then, to maintain $z = x-\partial\varphi$, the boundary $\partial\varphi$ is moved to $\partial\varphi + \alpha(\chi_u - \chi_v)$ by reducing flow on (u,v) by α.

Each time the algorithm applies Double-Exchange, it updates W and Z. If Double-Exchange(i,u,v) is nonsaturating, it makes v reachable from S in $G(\varphi)$, and hence W is enlarged. Thus there are at most n nonsaturating calls of Double-Exchange before a δ-augmenting path is found or all the active pairs disappear.

If a δ-augmenting path is found, the algorithm augments δ units of flow along the path. This increases $z^-(V)$ by δ since z changes only at the initial and terminal vertices of the path.

After each δ-augmentation, the algorithm computes an expression for x as a convex combination of at most n affinely independent extreme bases y_i, chosen from the current y_i's. This computation is a standard linear programming technique of transforming feasible solutions into basic feasible solutions. If the set of extreme points are not affinely independent, there is a set of coefficients μ_i for $i \in I$ that is not identically zero and satisfies $\sum \mu_i y_i = 0$ and $\sum \mu_i = 0$. Using Gaussian elimination, we can start computing such μ_i until a dependency is detected. At this point, we eliminate the dependency by computing $\theta := \min\{\lambda_i/\mu_i \mid \mu_i > 0\}$ and updating $\lambda_i := \lambda_i - \theta\mu_i$ for $i \in I$. At least one $i \in I$ satisfies $\lambda_i = 0$. Delete such i from I. We continue this procedure until we eventually obtain affine independence.

A δ-scaling phase ends when either $S = \emptyset$, $T = \emptyset$, or $Z = \emptyset$. In the last case, the set W of vertices that are reachable from S in $G(\varphi)$ is disjoint with T.

Lemma 3.1: *If $Z = \emptyset$, then $x(W) = f(W)$.*

Proof. If $Z = \emptyset$, for each $i \in I$ the first $|W|$ vertices in L_i must belong to W. Then it follows from (2.3) that $y_i(W) = f(W)$. Since $x = \sum_{i\in I} \lambda_i y_i$ and $\sum_{i\in I} \lambda_i = 1$, this implies $x(W) = \sum_{i\in I} \lambda_i y_i(W) = f(W)$. ■

3.3. Correctness and Complexity

We now investigate the number of iterations in each δ-scaling phase. To do this, we prove relaxed weak and strong dualities. The next lemma shows a relaxed weak duality.

Lemma 3.2: *For any base $x \in \mathrm{B}(f)$ and any δ-feasible flow φ, the vector $z = x - \partial\varphi$ satisfies $z^-(V) \le f(X) + \binom{n}{2}\delta$ for any $X \subseteq V$.*

Proof. For any $X \subseteq V$ we have $x(X) \le f(X)$ and $\partial\varphi(X) \ge -\binom{n}{2}\delta$, and hence $z^-(V) \le z(X) \le f(X) + \binom{n}{2}\delta$. ■

A relaxed strong duality is given as follows.

Lemma 3.3: *At the end of each δ-scaling phase, the following* (i)–(iii) *hold for x and $z = x - \partial\varphi$.*

(i) *If $S = \emptyset$, then $x^-(V) \ge f(\emptyset) - n^2\delta$ and $z^-(V) \ge f(\emptyset) - n\delta$.*

(ii) *If $T = \emptyset$, then $x^-(V) \ge f(V) - n^2\delta$ and $z^-(V) \ge f(V) - n\delta$.*

(iii) *If $x(W) = f(W)$, then $x^-(V) \ge f(W) - n^2\delta$ and $z^-(V) \ge f(W) - n\delta$.*

Proof. When the δ-scaling phase finishes with $S = \emptyset$, we have $x(v) > \partial\varphi(v) - \delta \ge -n\delta$ for every $v \in V$, which implies $x^-(V) \ge f(\emptyset) - n^2\delta$ as well as $z^-(V) \ge f(\emptyset) - n\delta$. Similarly, when the δ-scaling phase finishes with $T = \emptyset$, we have $x(v) < \partial\varphi(v) + \delta \le n\delta$ for every

SFM(f):

Initialization:
$L \leftarrow$ an linear ordering on V
$x \leftarrow$ an extreme base in B(f) generated by L
$I \leftarrow \{i\}$, $y_i \leftarrow x$, $\lambda_i \leftarrow 1$, $L_i \leftarrow L$
$\varphi \leftarrow \mathbf{0}$,
$\delta \leftarrow M$
While $\delta \geq 1/n^2$ **do**
$\delta \leftarrow \delta/2$
For $(u,v) \in E$ **do**
If $\varphi(u,v) > \delta$ **then** $\varphi(u,v) \leftarrow \delta$
If $\varphi(u,v) < -\delta$ **then** $\varphi(u,v) \leftarrow -\delta$
$S \leftarrow \{v \mid x(v) \leq \partial\varphi(v) - \delta\}$
$T \leftarrow \{v \mid x(v) \geq \partial\varphi(v) + \delta\}$
$W \leftarrow$ the set of vertices reachable from S in $G(\varphi)$
$Z \leftarrow$ the set of active pairs (i,v) of $i \in I$ and $v \in V$
While $S \neq \emptyset$, $T \neq \emptyset$ and $Z \neq \emptyset$ **do**,
While $W \cap T = \emptyset$ and $Z \neq \emptyset$ **do**,
Find an active pair $(i,v) \in Z$.
Let u be the vertex succeeding v in L_i.
Apply Double-Exchange(i,u,v).
Update W and Z.
If $W \cap T \neq \emptyset$ **then**
Let P be a directed path from S to T in $G(\varphi)$.
For $(u,v) \in P$ **do** $\varphi(u,v) \leftarrow \varphi(u,v) + \delta$, $\varphi(v,u) \leftarrow \varphi(v,u) - \delta$
Update S, T, W, and Z.
Express x as $x = \sum_{i \in I} \lambda_i y_i$ by possibly smaller affinely independent subset I and positive coefficients $\lambda_i > 0$ for $i \in I$.
If $S = \emptyset$ **then** $X \leftarrow \emptyset$ **else if** $T = \emptyset$ **then** $X \leftarrow V$ **else** $X \leftarrow W$
Return X
End.

Figure 2: A scaling algorithm for submodular function minimization.

$v \in V$, which implies $x^-(V) \geq x(V) - n^2\delta = f(V) - n^2\delta$ as well as $z^-(V) \geq x(V) - n\delta$.

When the δ-scaling phase ends with $x(W) = f(W)$, then $S \subseteq W \subseteq V \backslash T$ and $\partial\varphi(W) < 0$. By the definitions of S and T, we also have $x(v) > \partial\varphi(v) - \delta \geq -n\delta$ for every $v \in V \backslash W$ and $x(v) < \partial\varphi(v) + \delta \leq n\delta$ for every $v \in W$. Therefore we have $x^-(V) = x^-(W) + x^-(V \backslash W) \geq x(W) - n\delta|W| - n\delta|V \backslash W| = f(W) - n^2\delta$ as well as $z^-(V) = z^-(W) + z^-(V \backslash W) \geq x(W) - \partial\varphi(W) - |W|\delta - \delta|V \backslash W| \geq f(W) - n\delta$. ■

Theorem 3.4: *The number of augmentations per scaling phase is at most $n^2 + n$.*

Proof. Lemma 3.3 implies that at the beginning of the δ-scaling phase, after δ is cut in half, $z^-(V)$ is at least $f(X) - 2n\delta$ for some $X \subseteq V$. Preprocessing the flow φ to make it δ-feasible decreases $z^-(V)$ by at most $\binom{n}{2}\delta$. On the other hand, $z^-(V)$ is at most $f(X) + \binom{n}{2}\delta$ at the end of a δ-scaling phase by Lemma 3.2. Since each δ-augmentation increases $z^-(V)$ by δ, the number of δ-augmentations per phase is at most $n^2 + n$ for all phases after the first.

Since $z^-(V) = x^-(V) \geq -M$ at the start of the algorithm, setting $\delta = M/n^2$ is sufficient to obtain a similar bound on the number of augmentations in the first scaling phase. ■

As an immediate consequence of Lemmas 2.2 and 3.3, we also obtain the following.

Theorem 3.5: *The algorithm obtains a minimizer of f at the end of the δ-scaling phase with $\delta < 1/n^2$.*

Proof. By Lemma 3.3, the output X of the algorithm satisfies $x^-(V) \geq f(X) - n^2\delta > f(X) - 1$. For any $Y \subseteq V$, the weak duality in Lemma 2.2 asserts $x^-(V) \leq$

$f(Y)$. Thus we have $f(X) - 1 < f(Y)$, which implies by the integrality of f that X minimizes f. ■

Theorem 3.6: SFM *runs in* $O(n^5 \log M)$ *time.*

Proof. The algorithm starts with $\delta = M/n^2$ and ends with $\delta < 1/n^2$, so the algorithm consists of $O(\log M)$ scaling phases. Each scaling phase finds $O(n^2)$ augmenting paths.

We now claim that the algorithm performs the procedure Double-Exchange $O(n^2)$ times for each $i \in I$ between δ-augmentations. Each time the algorithm picks an active pair (i, v) and applies Double-Exchange(i, u, v), the vertex v shifts towards the end of L_i. If $v \notin W$, then it only moves towards the end of L_i. Once $v \in W$, it stays in W until the next δ-augmentation or the end of the phase. Hence the algorithm picks an active pair (i, v) at most n times between augmentations. Therefore, for each $i \in I$, the algorithm applies Double-Exchange to active vertices in L_i at most $O(n^2)$ times.

A new index k is added to I only as a result of a nonsaturating Double-Exchange, and there are less than n nonsaturating exchanges between δ-augmentations. Hence we have $|I| \leq 2n$, and the algorithm performs $O(n^3)$ saturating exchanges per δ-augmentation. A saturating Double-Exchange requires $O(1)$ time while a nonsaturating one $O(n)$ time. Therefore, the time spent in Double-Exchange per augmenting path is $O(n^3)$.

After each augmentation, we also update the expression $x = \sum_{i \in I} \lambda_i y_i$. The bottleneck in this procedure is the time spent computing the coefficients μ_i. Since $|I| \leq 2n$, this also takes $O(n^3)$ time.

Thus the overall complexity is $O(n^5 \log M)$. ■

The previous best known pseudopolynomial time bound is $O(n^6 M \log(nM))$ [3]. Theorem 3.6 shows that our scaling algorithm is faster than this even if M is fixed as a constant.

We note that we could relax the definition of an active vertex to include any vertex $v \in V \backslash W$ whose immediate successor in L_i belongs to W. The correctness argument would apply without modifications. However, care is needed to obtain an efficient implementation.

In this section, we have shown a weakly polynomial-time algorithm for minimizing integer-valued submodular functions. The integrality of a submodular function f guarantees that if we have a base $x \in B(f)$ and a subset X of V such that $f(X) - x^-(V)$ is less than one, X is a minimizer of f. Except for this we have not used the integrality of f. It follows that for any real-valued submodular function $f : 2^V \to \mathbf{R}$, if we are given a positive lower bound ϵ for the difference between the second minimum and the minimum value of f, the present algorithm works for the submodular function $(1/\epsilon)f$ and runs in $O(n^5 \log(nM/\epsilon))$ time, where M is an upper bound on $|f(X)|$ among $X \subseteq V$.

4. A Strongly Polynomial-Time Algorithm

This section presents a strongly polynomial-time algorithm for minimizing a submodular function $f : 2^V \to \mathbf{R}$ using the scaling algorithm in Section 3. The main idea behind the algorithm is to show via Lemma 4.1 that after $O(\log n)$ scaling phases, the algorithm detects either a new vertex that that is contained in every minimizer of f, or a new vertex pair (u, v) such that v is in every minimizer of f containing u. Since there are at most $O(n^2)$ such detections, after $O(n^2 \log n)$ scaling phases, the algorithm finds a minimizer of f.

Lemma 4.1: *If* $x(w) < -n^2\delta$ *at the end of the* δ*-scaling phase, then* w *is contained in every minimizer of* f*.*

Proof. By Lemma 3.3, at the end of the δ-scaling phase, there exists a subset $Y \subseteq V$ such that $x^-(V) \geq f(Y) - n^2\delta$. For any minimizer X of f, we have $f(Y) \geq f(X) \geq x(X) \geq x^-(X)$. Thus $x^-(V) \geq x^-(X) - n^2\delta$; so if $x(w) < -n^2\delta$, then $w \in X$. ■

The new algorithm maintains a subset $X \subseteq V$ that is included in every minimizer of f, a vertex set U corresponding to a partition of $V \backslash X$ into pairwise disjoint subsets, a directed acyclic graph $D = (U, F)$, and a submodular function $\widehat{f}$ on the subsets of U. Each arc in F is an ordered pair (u, w) of vertices in U such that w is in every minimizer of $\widehat{f}$ containing u. We call such a pair *compatible* with $\widehat{f}$. A component of $V \backslash X$ represented by a vertex of U corresponds to a set of mutually compatible pairs for f. For any vertex subset $Y \subseteq U$, we denote by $\Gamma(Y)$ the union of those components represented by the vertices in Y. Thus for any minimizer W the set $\Gamma(\{u\}) \subseteq V$ for any $u \in U$ is either completely contained in W, or completely excluded from W. Throughout the algorithm, any minimizer of f is represented as $X \cup \Gamma(W)$ for some minimizer W of $\widehat{f}$. Initially, the algorithm assigns $U := V$, $F := \emptyset$, $\widehat{f} := f$, and $X := \emptyset$, which clearly satisfy the above properties.

Let $R(u)$ denote the set of the vertices reachable from $u \in U$ in D. Then (u, w) is compatible with $\widehat{f}$ for every $w \in R(u)$. We denote by $\widehat{f}_u$ the *contraction* of $\widehat{f}$ by $R(u)$, i.e.,

$$\widehat{f}_u(Y) = \widehat{f}(Y \cup R(u)) - \widehat{f}(R(u)), \qquad \forall\, Y \subseteq U \backslash R(u).$$

When the algorithm detects a new vertex w that is contained in every minimizer of $\widehat{f}$, then it deletes $R(w)$ from D, adds $\Gamma(R(w))$ to X, and contracts $\widehat{f}$ by $R(w)$. When the algorithm detects a new compatible pair (u, w), then it adds (u, w) to F. If this creates a cycle in D, the algorithm contracts the cycle to a single vertex and modifies $\widehat{f}$ by regarding the contracted vertex set as a singleton.

The algorithm applies the proximity lemma at the end of $O(\log n)$ scaling phases that start with an appropriate scale parameter. This works as follows. Let $x \in B(\widehat{f})$ be an extreme base whose components are bounded from above by $\eta > 0$. Let $Y \subseteq U$ be a subset such that $\widehat{f}(Y) \leq -\eta/2$. After applying the scaling algorithm starting with $\delta = \eta$ and the extreme base $x \in B(\widehat{f})$ for $\lceil \log_2(2n^3) \rceil$ scaling phases, the new δ is less than $\eta/2n^3$; and since $x(Y) \leq \widehat{f}(Y) \leq -\eta/2$, at least one element $w \in Y$ satisfies $x(w) < -n^2\delta$. By Lemma 4.1, such an element w is contained in every minimizer of $\widehat{f}$. We denote this procedure by $\mathsf{Fix}(\widehat{f}, x, \eta)$.

The procedure Fix is applied either directly to $\widehat{f}$, or to $\widehat{f}_u$ for some $u \in U$. When applied to $\widehat{f}$, Fix identifies a new element that is contained in every minimizer of $\widehat{f}$. When applied to $\widehat{f}_u$, Fix identifies a new compatible arc leaving u. Below, we describe the framework in which Fix is applied to yield a strongly polynomial-time algorithm. First we require a few definitions. A formal description of the algorithm is given in Figure 3.

A linear ordering $(u_1, \cdots, u_k)$ of U is called *consistent* with D if $i < j$ implies $(u_i, u_j) \notin F$. The extreme base generated by a consistent linear ordering is also called *consistent*.

Lemma 4.2: *Any consistent extreme base $x \in B(\widehat{f})$ satisfies $x(u) \leq \widehat{f}(R(u)) - \widehat{f}(R(u)\backslash\{u\})$ for each $u \in U$.*

Proof. The consistent extreme base x satisfies $x(u) = \widehat{f}(Y) - \widehat{f}(Y\backslash\{u\})$ for some $Y \supseteq R(u)$. The claim then follows from the submodularity of $\widehat{f}$. ■

To start each iteration, whenever $\widehat{f}(U) > 0$, the algorithm replaces the value $\widehat{f}(U)$ by zero. The set of minimizers remains the same unless the minimum value is zero, in which case $\emptyset$ minimizes $\widehat{f}$. This modification is done so that if $\widehat{f}(R(u))$ is sufficiently positive, then $\widehat{f}_u(U\backslash R(u)) = \widehat{f}(U) - \widehat{f}(R(u))$ is sufficiently negative, providing a witness for the applicability of Fix to $\widehat{f}_u$.

Before each application of Fix, the algorithm computes

$$\eta = \max\{\widehat{f}(R(u)) - \widehat{f}(R(u)\backslash\{u\}) \mid u \in U\}. \qquad (4.1)$$

If $\eta \leq 0$, then an extreme base $x \in B(\widehat{f})$ consistent with D satisfies $x(u) \leq 0$ for each $u \in U$. In this case $x^-(U) = x(U) = \widehat{f}(U)$, which implies that U minimizes $\widehat{f}$ by the weak duality in Lemma 2.2. If in addition $\widehat{f}(U) = 0$, then the original function may have had a positive value of $\widehat{f}(U)$. Therefore, the algorithm returns $\emptyset$ or U as a minimizer of $\widehat{f}$, according to whether $\widehat{f}(U) = 0$ or $\widehat{f}(U) < 0$. In the former case X minimizes f, while in the latter case V does.

If $\eta > 0$, then let u be an element that attains the maximum in the right-hand side of (4.1). Then we have $\widehat{f}(R(u)) = \widehat{f}(R(u)\backslash\{u\}) + \eta$, which implies either $\widehat{f}(R(u)) \geq \eta/2 > 0$ or $\widehat{f}(R(u)\backslash\{u\}) < -\eta/2 < 0$ holds.

Lemma 4.3: *If $\widehat{f}(R(u)) \geq \eta/2 > 0$, then $\mathsf{Fix}(\widehat{f}_u, x, \eta)$ applies to any $x \in B(\widehat{f}_u)$ consistent with D restricted to $U\backslash R(u)$, and it finds a new pair (u, w) compatible with $\widehat{f}$.*

Proof. If $\widehat{f}(R(u)) \geq \eta/2$, then $\widehat{f}_u(U\backslash R(u)) = \widehat{f}(U) - \widehat{f}(R(u)) \leq -\eta/2$. If $x \in B(\widehat{f}_u)$ is an extreme base generated by a linear ordering of $U\backslash R(u)$ consistent with D, then x satisfies $x(t) \leq \widehat{f}(R(t)) - \widehat{f}(R(t)\backslash\{t\}) \leq \eta$ for each $t \in U\backslash R(u)$ by Lemma 4.2. Thus we may apply the procedure $\mathsf{Fix}(\widehat{f}_u, x, \eta)$ to find an element $w \in U\backslash R(u)$ that is contained in every minimizer of $\widehat{f}_u$. This (u, w) is a new pair compatible with $\widehat{f}$ (*new* since we contracted $R(u)$), and is added to D. ■

Lemma 4.4: *If $\widehat{f}(R(u)\backslash\{u\}) < -\eta/2 < 0$, then the subroutine $\mathsf{Fix}(\widehat{f}, x, \eta)$ applies to any x consistent with D.*

Proof. If $\widehat{f}(R(u)\backslash\{u\}) < -\eta/2$, then $\mathsf{Fix}(\widehat{f}, x, \eta)$ applies to x consistent with D obtained by the greedy algorithm, by choice of η and Lemma 4.2. ■

Once Fix finds a new element w in every minimizer of $\widehat{f}$, then every minimizer of $\widehat{f}$ includes $R(w)$. Thus it suffices to minimize the submodular function $\widehat{f}_w$, which is now defined on a smaller underlying set, and thus the algorithm redefines $\widehat{f} := \widehat{f}_w$.

Theorem 4.5: *The algorithm in* Figure 3 *computes the minimizer of a submodular function in $O(n^7 \log n)$ time, which is strongly polynomial.*

Proof. Each time we call the procedure Fix, the algorithm adds a new arc to D or deletes a set of vertices. This can happen at most $O(n^2)$ times. Each call to Fix takes $O(\log n)$ phases. By Theorem 3.4, each phase has $O(n^2)$ augmentations. Since the proof of Theorem 3.6 shows that the amount of work per augmentation is $O(n^3)$, this yields an overall run time of $O(n^7 \log n)$, which is strongly polynomial. ■

5. Concluding Remarks

This paper presents a strongly polynomial-time algorithm for minimizing submodular functions defined on Boolean lattices. We now briefly discuss minimizing submodular functions defined on more general lattices.

Consider a submodular function $f : \mathcal{D} \to \mathbf{R}$ defined on a distributive lattice $\mathcal{D}$ represented by a poset $\mathcal{P}$ on V. Then the associated base polyhedron is unbounded in general.

Initialization:
$X \leftarrow \emptyset$
$U \leftarrow V$, $\widehat{f} \leftarrow f$
$F \leftarrow \emptyset$
While $U \neq \emptyset$ **do**
If $\widehat{f}(U) > 0$ **then** $\widehat{f}(U) \leftarrow 0$
$\eta \leftarrow \max\{\widehat{f}(R(u)) - \widehat{f}(R(u)\backslash\{u\}) \mid u \in U\}$
If $\eta \leq 0$ **then break**
Let $u \in U$ attain the maximum above.
If $\widehat{f}(R(u)) \geq \eta/2$ **then**
Find a consistent extreme base $x \in \mathrm{B}(\widehat{f}_u)$ by the greedy algorithm.
$w \leftarrow \mathsf{Fix}(\widehat{f}_u, x, \eta)$
If $u \in R(w)$ **then**
Contract $\{v \mid v \in R(w),\ u \in R(v)\}$ to a single vertex.
Else $F \leftarrow F \cup \{(u, w)\}$
Else
Find a consistent extreme base $x \in \mathrm{B}(\widehat{f})$ by the greedy algorithm.
$w \leftarrow \mathsf{Fix}(\widehat{f}, x, \eta)$
$U \leftarrow U\backslash R(w)$
$\widehat{f} \leftarrow \widehat{f}_w$
$X \leftarrow X \cup \Gamma(R(w))$
If $\widehat{f}(U) < 0$ **then** $X \leftarrow V$
Return X
End.

Figure 3: A strongly polynomial-time algorithm for submodular function minimization.

In order to minimize such a function f, we can slightly extend the algorithms in Sections 3 and 4 by keeping the base $x \in \mathrm{B}(f)$ as a convex combination of extreme bases y_i's plus a vector in the characteristic cone of $\mathrm{B}(f)$. The latter can be represented as a boundary of a nonnegative flow in the Hasse diagram of $\mathcal{P}$. This extension enables us to minimize f in $\mathrm{O}(n^5 \min\{\log \hat{M}, n^2 \log n\})$ time, where $\hat{M}$ is an upper bound on $|f(X)|$ among $X \in \mathcal{D}$.

Submodular functions defined on modular lattices naturally arise in linear algebra. Minimization of such functions has a significant application to canonical forms of partitioned matrices [21, 24]. It remains an interesting open problem to develop an efficient algorithm for minimizing submodular functions on modular lattices, even for those specific functions that arise from partitioned matrices.

Independently, Schrijver [29] has also developed a combinatorial, strongly polynomial-time algorithm for submodular function minimization. This algorithm is also based on Cunningham's approach, although the two algorithms are quite different.

Acknowledgements

We are grateful to Bill Cunningham, Michel Goemans, and Maiko Shigeno for their useful comments.

References

[1] R. E. Bixby, W. H. Cunningham, and D. M. Topkis: Partial order of a polymatroid extreme point, *Math. Oper. Res.*, **10** (1985), 367–378.

[2] W. H. Cunningham: Testing membership in matroid polyhedra, *J. Combinatorial Theory*, **B36** (1984), 161–188.

[3] W. H. Cunningham: On submodular function minimization, *Combinatorica*, **5** (1985), 185–192.

[4] J. Edmonds: Submodular functions, matroids, and certain polyhedra, *Combinatorial Structures and Their Applications*, R. Guy, H. Hanani, N. Sauer, and J. Schönheim, eds., Gordon and Breach, 69–87, 1970.

[5] J. Edmonds and R. Giles: A min-max relation for submodular function on graphs, *Ann. Discrete Math.*, **1** (1977), 185–204.

[6] J. Edmonds and R. Karp: Theoretical improvements in algorithmic efficiency for network flow problems, *J. ACM*, **19** (1972), 248–264.

[7] L. Fleischer, S. Iwata, and S. T. McCormick: A faster capacity scaling algorithm for submodular flow, 1999.

[8] A. Frank: An algorithm for submodular functions on graphs, *Ann. Discrete Math.*, **16** (1982), 189–212.

[9] A. Frank and É. Tardos: An application of simultaneous Diophantine approximation in combinatorial optimization, *Combinatorica*, **7** (1987), 49–65.

[10] A. Frank and É. Tardos: Generalized polymatroids and submodular flows, *Math. Programming*, **42** (1988), 489–563.

[11] S. Fujishige: Polymatroidal dependence structure of a set of random variables, *Information and Control*, **39** (1978), 55–72.

[12] S. Fujishige: Lexicographically optimal base of a polymatroid with respect to a weight vector, *Math. Oper. Res.*, **5** (1980), 186–196.

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

[14] S. Fujishige: Theory of submodular programs: A Fenchel-type min-max theorem and subgradients of submodular functions, *Math. Programming*, **29** (1984), 142–155.

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

[16] M. X. Goemans and V. S. Ramakrishnan: Minimizing submodular functions over families of subsets, *Combinatorica*, **15** (1995), 499–513.

[17] M. Grötschel, L. Lovász, and A. Schrijver: The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica*, **1** (1981), 169–197.

[18] M. Grötschel, L. Lovász, and A. Schrijver: *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, 1988.

[19] T.-S. Han: The capacity region of general multiple-access channel with correlated sources, *Information and Control*, **40** (1979), 37–60.

[20] B. Hoppe and É. Tardos: The quickest transshipment problem, *Proceedings of Fifth ACM/SIAM Symposium on Discrete Algorithms* (1995), 512–521.

[21] H. Ito, S. Iwata, and K. Murota: Block-triangularization of partitioned matrices under similarity/equivalence transformations, *SIAM J. Matrix Anal. Appl.*, **15** (1994), 1226–1255.

[22] S. Iwata: A capacity scaling algorithm for convex cost submodular flows, *Math. Programming*, **76** (1997), 299–308.

[23] S. Iwata, S. T. McCormick, and M. Shigeno: A strongly polynomial cut canceling algorithm for the submodular flow problem, *Proceedings of the Seventh MPS Conference on Integer Programming and Combinatorial Optimization* (1999), 259–272.

[24] S. Iwata and K. Murota: A minimax theorem and a Dulmage-Mendelsohn type decomposition for a class of generic partitioned matrices, *SIAM J. Matrix Anal. Appl.*, **16** (1995), 719–734.

[25] L. Lovász: Submodular functions and convexity. *Mathematical Programming — The State of the Art*, A. Bachem, M. Grötschel and B. Korte, eds., Springer-Verlag, 1983, 235–257.

[26] H. Nagamochi and T. Ibaraki: Computing edge-connectivity in multigraphs and capacitated graphs, *SIAM J. Discrete Math.*, **5** (1992), 54–64.

[27] H. Narayanan: A rounding technique for the polymatroid membership problem, *Linear Algebra Appl.*, **221** (1995), 41–57.

[28] M. Queyranne: Minimizing symmetric submodular functions, *Math. Programming*, **82** (1998), 3–12.

[29] A. Schrijver: A combinatorial algorithm minimizing submodular functions in strongly polynomial time, *J. Combinatorial Theory*, Ser. B, submitted.

[30] M. A. Sohoni: Membership in submodular and other polyhedra. Technical Report TR-102-92, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, India, 1992.

[31] A. Tamir: A unifying location model on tree graphs based on submodularity properties, *Discrete Appl. Math.*, **47** (1993), 275–283.

[32] É. Tardos: A strongly polynomial minimum cost circulation algorithm, *Combinatorica*, **5** (1985), 247–255.

This research was sponsored in part by National Science Foundation (NSF) grant no. CCR-0122581.