A Strongly Polynomial-Time Algorithm for Minimizing Submodular Functions

Satoru IWATA * Lisa FLEISCHER † Satoru FUJISHIGE ‡

July 1999

Abstract

This paper presents a combinatorial polynomial-time algorithm for minimizing submodular set functions. 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. These are the first combinatorial algorithms for submodular function minimization that run in (strongly) polynomial time.

Key words: submodular function, combinatorial optimization, strongly polynomial-time algorithm

*Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560-8531, Japan. E-mail: `iwata@sys.es.osaka-u.ac.jp`. Partly supported by Grants-in-Aid for Scientific Research from Ministry of Education, Science, Sports, and Culture of Japan.

†Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027, USA. E-mail: `lisa@ieor.columbia.edu`. This work done while on leave at Center for Operations Research and Econometrics, Université catholique de Louvain, Belgium.

‡Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560-8531, Japan. E-mail: `fujishig@sys.es.osaka-u.ac.jp`. Partly supported by Grants-in-Aid for Scientific Research from Ministry of Education, Science, Sports, and Culture of Japan.

1. Introduction

Grötschel, Lovász, and Schrijver [14] revealed the polynomial-time equivalence between the optimization and separation problems in combinatorial optimization via the ellipsoid method. Since then, many combinatorial problems have been shown to be polynomial-time solvable by means of their framework. The problem of minimizing submodular (set) functions is among these problems. Since the ellipsoid method is far from being efficient in practice and is not combinatorial, efficient combinatorial algorithms for submodular function minimization have been desired for a long time.

Submodular functions arise in various branches of mathematical engineering such as combinatorial optimization 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 base polyhedron associated with the relevant submodular function often plays an important role. See Lovász [21] and Fujishige [12] for fundamental results about submodular functions and for close connections to convexity.

Linear optimization problems over base polyhedra are efficiently solvable by the greedy algorithm of Edmonds [4]. Thus Grötschel, Lovász, and Schrijver [14] assert that the submodular function minimization, which is equivalent to the separation problem, is solvable in polynomial time by the ellipsoid method. Later, they also devised a strongly polynomial-time algorithm within their framework using the ellipsoid method [15].

A first step towards a combinatorial strongly polynomial-time algorithm was taken by Cunningham [2, 3], who devised a strongly polynomial-time algorithm for testing membership in matroid polyhedra as well as a pseudopolynomial-time algorithm for minimizing submodular functions. Recently, Narayanan [24] improved the running time bounds of these combinatorial algorithms by a rounding technique. Based on the minimum-norm base characterization of minimizers due to Fujishige [10, 11] (cf. [12, §7.1 (a)]), Sohoni [26] gave another combinatorial pseudopolynomial-time algorithm for submodular function minimization.

For the problem of minimizing a symmetric submodular function over proper nonempty subsets, Queyranne [25] presented a combinatorial strongly polynomial-time algorithm, extending the undirected minimum cut algorithm of Nagamochi and Ibaraki [22]. See also Nagamochi and Ibaraki [23] for a slight extension.

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 bases. 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 a pseudopolynomial-time algorithm.

These previous methods use the set of arcs of the Hasse diagrams of the partial orders

associated with the extreme bases. They are inefficient since the lower bound on the size 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 proved 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, which can be exponentially small in the size of the underlying set.

To overcome this difficulty, we augment the arc set of the Hasse diagrams with the complete directed graph on the underlying set, letting the capacity of this additional arc set depend directly on our scaling parameter. This technique was first introduced by Iwata [19], who used it to develop the first polynomial-time capacity-scaling algorithm for the submodular flow problem of Edmonds and Giles [5]. This algorithm was later refined by Fleischer, Iwata, and McCormick [7] into one of the fastest algorithms for submodular flow. Our work in this paper builds on ideas in this latter paper 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 undelying set and the largest length of the function value.

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 [8] 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 apply to submodular function minimization. Instead, we devise a combinatorial algorithm that repeatedly detects an element that belongs to every minimizer or an ordered pair of elements with the property that if the first belongs to a minimizer then the second does.

There are some practical problems, in dynamic flows [17], facility location [27], and multi-terminal source coding [9, 16], where the polynomial-time solvability relies on a submodular function minimization routine. Goemans and Ramakrishnan [13] discussed a class of submodular function minimization problems over restricted families of subsets. Their solution is combinatorial modulo 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. In Section 5, we give a brief discussion on extensions.

2. Preliminaries

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 $|V| = n$. The set of functions $x : V \to \mathbf{R}$ forms a linear space $\mathbf{R}^V$. A vector in $x \in \mathbf{R}^V$ is usually identified with a modular function $x : 2^V \to \mathbf{R}$ defined by $x(X) = \sum\{x(v) \mid v \in X\}$ $(X \subseteq V)$. For each $u \in V$, we denote by χ_u the unit vector in $\mathbf{R}^V$ such that $\chi_u(v) = 1$ $(v = u)$ and $= 0$ $(v \in V \backslash \{u\})$.

A function $f : 2^V \to \mathbf{R}$ is said to be *submodular* if it satisfies

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

We suppose that $f(\emptyset) = 0$ without loss of generality throughout this paper. For basic notation and facts about submodular functions, see Lovász [21] and Fujishige [12], for example.

We define the *submodular polyhedron* $\mathrm{P}(f)$ and the *base polyhedron* $\mathrm{B}(f)$ associated with the submodular function f by

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

A vector $x \in \mathrm{B}(f)$ is called a *base*. For any base $x \in \mathrm{B}(f)$ and any distinct $u, v \in V$ define the *exchange capacity*

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

The exchange capacity can be expressed as

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

We call an extreme point of $\mathrm{B}(f)$ an *extreme base*. For any extreme base $y \in \mathrm{B}(f)$ define

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

Then, $\mathcal{D}(y)$ is a distributive lattice, i.e., $X, Y \in \mathcal{D}(y)$ implies $X \cup Y, X \cap Y \in \mathcal{D}(y)$. A set $X \in \mathcal{D}(y)$ is called *tight* for y. Since y is an extreme base, $\mathcal{D}(y)$ is *simple*, i.e., a maximal chain in $\mathcal{D}(y)$ is of length $n = |V|$. Hence, there exists a unique poset $\mathcal{P}(y) = (V, \preceq_y)$ on V such that the set of (lower) ideals of $\mathcal{P}(y)$ coincides with $\mathcal{D}(y)$. Note that for distinct $u, v \in V$ we have $\tilde{\mathrm{c}}(y, u, v) > 0$ if and only if $v \preceq_y u$ in $\mathcal{P}(y)$. Denote by $H(y) = (V, A(y))$ the Hasse diagram of the poset $\mathcal{P}(y)$. Given an extreme base y, we can construct the Hasse diagram $H(y)$ in $\mathrm{O}(n^2)$ time by using the evaluation oracle [1] (cf. [12, pp. 62–63]).

A fundamental operation in our algorithm is to transform an extreme base $y \in \mathrm{B}(f)$ to another extreme base $y' = y + \tilde{\mathrm{c}}(y, u, v)(\chi_u - \chi_v)$ for $u, v \in V$ with $(u, v) \in A(y)$, i.e., (u, v) being an arc of the Hasse diagram $H(y)$. The new extreme base y' thus obtained is

adjacent to y in the base polyhedron [12, Theorem 3.47]. Computing exchange capacities in general is as hard as minimizing submodular functions. However, Lemma 2.2 given below shows that if $y \in \mathrm{B}(f)$ is an extreme base, then the exchange capacity $\tilde{c}(y,u,v)$ can be easily computed for all $(u,v) \in A(y)$. We denote by $J_y(u)$ the principal ideal of $\mathcal{P}(y)$ generated by u. That is, $J_y(u)$ is the unique minimal ideal of $\mathcal{P}(y)$ that contains u. Note that $J_y(u)$ is the same as $\mathrm{dep}(y,u)$ in [12]. We require the following easy technical lemma, which also appears in [1].

Lemma 2.1: *For an extreme base $y \in \mathrm{B}(f)$, if X is tight and u is maximal in X with respect to $\preceq_y$, then $X\backslash\{u\}$ is also tight for y.* ■

Lemma 2.2: *For an extreme base $y \in \mathrm{B}(f)$ let $(u,v) \in A(y)$. Then we have*

$$\tilde{c}(y,u,v) = f(J_y(u)\backslash\{v\}) - y(J_y(u)\backslash\{v\}). \tag{2.3}$$

Proof. Let X be a minimal minimizer in the right-hand side of (2.2). Since $y(J_y(u)) = f(J_y(u))$ and $y(X \cup J_y(u)) \leq f(X \cup J_y(u))$, it follows from the submodularity of f that

$$f(X \cap J_y(u)) - y(X \cap J_y(u)) \leq f(X) - y(X),$$

which implies by the definition of X that $X \subseteq J_y(u)$. Since u is maximal in $J_y(u)$, $J_y(u)\backslash\{u\}$ is tight by Lemma 2.1. Since $(u,v) \in A(y)$, v is maximal in $J_y(u)\backslash\{u\}$. Hence, $J_y(u)\backslash\{u,v\}$ is also tight. Since $y(X\backslash\{u\}) \leq f(X\backslash\{u\})$, the submodularity of f further implies

$$f(J_y(u)\backslash\{v\}) - y(J_y(u)\backslash\{v\}) \leq f(X) - y(X).$$

Thus we obtain (2.3). ■

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 (see [12, Corollaries 3.4 and 3.5]).

Lemma 2.3: *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. ■

We will not use the integrality property indicated in the latter half of this lemma. Lemma 2.3 shows a min-max relation of strong duality. A weak duality is described as follows: for any base $x \in \mathrm{B}(f)$ and any $X \subseteq V$ we have $x^-(V) \leq f(X)$. We call the difference $f(X) - x^-(V)$ a *duality gap*. Note that, if f is integer-valued and the duality gap $f(X) - x^-(V)$ is less than one for some $x \in \mathrm{B}(f)$ and $X \subseteq V$, then X minimizes f.

3. A Scaling Algorithm

The present section gives 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. Let M denote an upper bound on $|f(X)|$ $(X \subseteq V)$. Note that we can easily compute M by $O(n)$ calls for the evaluation oracle.

3.1. Algorithm Outline

As indicated earlier, our algorithm uses an augmenting path approach to submodular function minimization [1, 2, 3]. As with previous algorithms, we maintain a base $x \in B(f)$ as a convex combination of extreme bases $y_i \in B(f)$ $(i \in I)$, so that $x = \sum_{i \in I} \lambda_i y_i$. Roughly speaking, these previous algorithms seek to increase $x^-(V)$ by augmenting from vertices v with $x(v) < 0$ to vertices u with $x(u) > 0$ along paths of arcs in $\bigcup_{i \in I} A(y_i)$. The algorithms stop with the optimal x when there are no more augmenting paths. The corresponding minimizer X is determined by the set of vertices reachable from vertices v with $x(v) < 0$.

To adapt this procedure to a scaling framework, we augment the arc set $\bigcup_{i \in I} A(y_i)$ with the complete directed graph on V, let the capacity of this graph depend directly on our scaling parameter δ, an idea first introduced for submodular flows in [19]. In this regard, we actually concern ourselves with a vector $z = x - \partial\varphi$ where $\varphi : V \times V \to \mathbf{R}$ is maintained as *skew-symmetric*, i.e., $\varphi(u, v) + \varphi(v, u) = 0$ for $u, v \in V$, and *δ-feasible* in that it satisfies capacity constraints $-\delta \leq \varphi(u, v) \leq \delta$ for every $u, v \in V$. 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* $\partial\varphi : V \to \mathbf{R}$ of φ is defined by

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

We start with $x \in B(f)$ as an extreme point, which is easily obtainable using the greedy algorithm, and φ as the zero flow. Thus, initially $z^-(V) = x^-(V) \geq -nM$. We seek to increase $z^-(V)$, and in doing so, obtain improvements in $x^-(V)$, via the δ-feasibilty of φ.

The algorithm consists of scaling phases with a positive parameter δ. It starts with $\delta = M$, cuts δ in half at the beginning of each scaling phase, and ends with $\delta < 1/n^2$. 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.

A phase starts by preprocessing φ to make it δ-feasible, and then repeatedly searches to send flow along augmenting paths in $G(\varphi)$ from $S := \{v \mid v \in V, x(v) \leq \partial\varphi(v) - \delta\} = \{v \mid v \in V, z(v) \leq -\delta\}$ to $T := \{v \mid v \in V, x(v) \geq \partial\varphi(v) + \delta\} = \{v \mid v \in V, z(v) \geq \delta\}$. Such a directed path is called a *δ-augmenting path.*

If there are no δ-augmenting paths, then the algorithm checks the set of arcs in $\bigcup_i A(y_i)$ to try to augment along these arcs individually. Such an augmentation changes both x and φ together without changing z and may increase the set of vertices reachable from S in $G(\varphi)$. This is an extension of a technique for handling exchange capacity arcs in submodular flows first developed in [7]. 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 in one iteration.

3.2. Algorithm Details

We now describe the scaling algorithm more precisely. Figure 1 provides a formal description, where Δ^+W is the set of arcs in $\bigcup_{i\in I} A(y_i)$ that leave W.

At the beginning of the δ-scaling phase, after δ is cut in half, the current flow φ is 2δ-feasible. Then the algorithm modifies each $\varphi(u,v)$ to the nearest value within the interval $[-\delta,\delta]$ to make it δ-feasible. 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 augmenting flow along δ-augmenting paths.

Let W denote the set of vertices reachable by directed paths from S in $G(\varphi)$. For each $i \in I$, consider $U_i = \{u \mid u \in W, \exists v \in V\backslash W, v \preceq_{y_i} u\}$. Then U_i is empty if and only if no arc in $H(y_i)$ leaves W. A pair of $i \in I$ and $u \in W$ is called *admissible* if u is minimal in U_i with respect to $\preceq_{y_i}$.

If $W \cap T = \emptyset$, there is no δ-augmenting path in $G(\varphi)$. Then, as long as the Hasse diagram $H(y_i)$ for some $i \in I$ has an arc leaving W, the algorithm repeatedly picks up an admissible pair of $i \in I$ and $u \in W$. It *scans* the pair (i,u) by applying the operation $\mathsf{Push}(i,u,v)$ to each arc $(u,v) \in A(y_i)$ with $v \in V\backslash W$.

The operation $\mathsf{Push}(i,u,v)$, depicted in Figure 2, starts with reducing the flow through (u,v) by $\alpha = \min\{\delta, \lambda_i \tilde{c}(y_i,u,v)\}$. It is called *saturating* if $\alpha = \lambda_i\tilde{c}(y_i,u,v)$, and *nonsaturating* otherwise. A saturating $\mathsf{Push}(i,u,v)$ updates y_i as $y_i := y_i + \tilde{c}(y_i,u,v)(\chi_u - \chi_v)$ while nonsaturating one adds to I a new index k with $y_k = y_i := y_i + \tilde{c}(y_i,u,v)(\chi_u - \chi_v)$ and $\lambda_k = \alpha/\tilde{c}(y_i,u,v)$ and updates λ_i as $\lambda_i := \lambda_i - \alpha/\tilde{c}(y_i,u,v)$. Consequently, x moves to $x + \alpha(\chi_u - \chi_v)$ in either case. Thus $z = x - \partial\varphi$ is invariant.

Each time the algorithm applies the push operation, it updates the set W of vertices reachable from S in $G(\varphi)$. If $\mathsf{Push}(i,u,v)$ is nonsaturating, it makes v reachable from S in $G(\varphi)$, and hence W is enlarged. Therefore, we encounter at most n nonsaturating pushes before we find a δ-augmenting path or all the admissible pairs disappear. A scan continues until W increases, or all arcs $(u,v) \in A(y_i)$ with $v \in V\backslash W$ disappear. In the

SFM(f):

Input: $f : 2^V \to \mathbf{Z}$
Output: $X \subseteq V$ minimizing f

Initialization:
$x \leftarrow$ an extreme base in $\mathrm{B}(f)$
$I \leftarrow \{i\}$, $y_i \leftarrow x$, $\lambda_i \leftarrow 1$,
$\varphi \leftarrow \mathbf{0}$,
$\delta \leftarrow M$
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 $G(\varphi)$
While $S \neq \emptyset$, $T \neq \emptyset$ and $\Delta^+W \neq \emptyset$ **do**
While $W \cap T = \emptyset$ and $\Delta^+W \neq \emptyset$ **do**
Find an admissible pair (i,u) of $i \in I$ and $u \in W$.
$Z \leftarrow \{v \mid v \in V \backslash W, (u,v) \in A(y_i)\}$
Repeat
Find a vertex $v \in Z$ and Push(i,u,v).
Update W and Z.
until $Z = \emptyset$ or $|W|$ increases.
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 and W.
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** $X = \emptyset$ **else if** $T = \emptyset$ **then** $X = V$ **else** $X = W$
End.

Figure 1: A scaling algorithm for submodular function minimization.

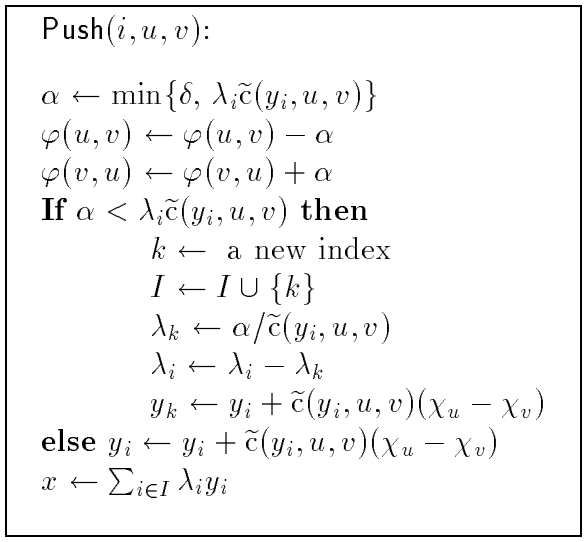

Figure 2: Algorithmic description of the operation Push(i,u,v).

first case, the scan is interrupted. Thus, if a scan is completed, all pushes are saturating.

If we find a δ-augmenting path, the algorithm augments δ units of flow along the path, which effectively increases $z^-(V)$ by δ. We also compute an expression for x as a convex combination of at most n affinely independent extreme bases y_i ($i \in I$), chosen from the current y_i's. This computation is a standard linear programming technique of transforming feasible solutions into basic ones by using Gaussian elimination. Since a new index k is added to I only as a result of a nonsaturating push, $|I| \leq 2n$ after finding an augmenting path. Thus, computing a new expression for x requires $O(n^3)$ time.

A δ-scaling phase ends when either $S = \emptyset$, $T = \emptyset$, or we find the set W of vertices reachable from S in $G(\varphi)$ being disjoint with T and having no leaving arcs in $\bigcup_{i \in I} A(y_i)$.

3.3. Correctness and Complexity

We first show that the use of the push operation results in correct and efficient augmentations.

For a saturating Push(i,u,v), we denote the new y_i by y_i' and the previous one by y_i. By Lemma 2.2, $J_{y_i}(u) \setminus \{v\}$ is tight for y_i'. Thus,

$$J_{y_i'}(u) \subseteq J_{y_i}(u) \setminus \{v\}. \tag{3.3}$$

For any $w \in W$ with $J_{y_i}(w) \subseteq W$, we have

$$J_{y_i'}(w) = J_{y_i}(w) \subseteq W. \tag{3.4}$$

These two facts are fundamental in the following argument.

Lemma 3.1: *After a saturating* $\mathsf{Push}(i,u,v)$*, if* $u \in U_i$ *and* $v \in V \backslash W$*, then* (i,u) *remains admissible.*

Proof. Note that no other vertex than v enters W after $\mathsf{Push}(i,u,v)$. We show that for any vertex $w \in W$ either $w \not\preceq_{y'_i} u$ or $w \notin U_i$ holds after the push. It follows from (3.3) that $w \not\preceq_{y_i} u$ implies $w \not\preceq_{y'_i} u$. On the other hand, since (i,u) is an admissible pair, $w \prec_{y_i} u$ implies $J_{y_i}(w) \subseteq W$. Then it follows from (3.4) that $J_{y'_i}(w) \subseteq W$. Thus, $w \prec_{y_i} u$ implies $w \notin U_i$ after the push. ■

Lemma 3.2: *For a vertex* $v \in V \backslash W$*,* $\mathsf{Push}(i,u,v)$ *is not repeated during a scan of* (i,u)*.*

Proof. This follows immediately from repeated applications of (3.3). ■

Lemma 3.3: *Once* (i,u) *is scanned, it does not become admissible again before the next augmentation.*

Proof. After scanning u, there are no arcs in $A(y_i)$ leaving both u and W. In addition, by the minimality of u, there is no path from u to a vertex $w \in U_i$. Hence $J_{y_i}(u) \subseteq W$. By (3.4) this property is maintained until the next augmentation along a directed path from S to T. Hence u does not reenter U_i before the next augmentation. ■

Lemmas 3.1 and 3.2 imply that there are at most $n-1$ pushes in a scan, whereas Lemma 3.3 implies that there are at most $2n^2$ scans before an augmenting path is found.

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

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

A relaxed strong duality is given as follows.

Lemma 3.5: *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* $x^-(V) \geq f(\emptyset) - n^2\delta$ *and* $z^-(V) \geq f(\emptyset) - n\delta$*.*

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

(iii) *If* $S \neq \emptyset$ *and* $T \neq \emptyset$*, then* $x^-(V) \geq f(W) - n^2\delta$ *and* $z^-(V) \geq f(W) - n\delta$*.*

Proof. When the δ-scaling phase finishes with $S = \emptyset$, we have $x(v) > \partial\varphi(v) - \delta \geq -n\delta$ for every $v \in V$, which implies $x^-(V) \geq f(\emptyset) - n^2\delta$ as well as $z^-(V) \geq f(\emptyset) - n\delta$. Similarly, when the δ-scaling phase finishes with $T = \emptyset$, we have $x(v) < \partial\varphi(v) + \delta \leq n\delta$ for every $v \in V$, 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 $S \neq \emptyset$ and $T \neq \emptyset$, we have a vertex subset $W \subseteq V$ such that $S \subseteq W \subseteq V \backslash T$ and there exists no arc leaving W in $G(\varphi)$ nor in $H(y_i)$ for any $i \in I$. Then we have $\partial\varphi(W) < 0$ and $y_i(W) = f(W)$ for every $i \in I$. Since $x \in \mathrm{B}(f)$ is the convex combination of y_i's, the latter implies $x(W) = f(W)$. 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 $z^-(V) = z^-(W) + z^-(V \backslash W) \geq x(W) - \partial\varphi(W) - |W|\delta - \delta|V \backslash W| \geq f(W) - n\delta$. ■

Lemma 3.5 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 $X \subseteq V$. Making the current flow δ-feasible decreases $z^-(V)$ by at most $\binom{n}{2}\delta$. Each δ-augmentation increases $z^-(V)$ by δ. Since $z^-(V)$ is at most $f(X) + \binom{n}{2}\delta$ at the end of a δ-phase by Lemma 3.4 the number of δ-augmentations per phase is at most $n^2 + n$ for all phases after the first. Since $z^-(V) = x^-(V) \geq -nM$ at the start of the algorithm, setting the initial $\delta = M$ is more than sufficient obtain a similar bound on the number of augmentations in the first phase.

As an immediate consequence of Lemmas 2.3 and 3.5, we also obtain the following.

Theorem 3.6: *The algorithm obtains a minimizer of f at the end of the δ-scaling phase with $\delta < 1/n^2$.*

Proof. By Lemma 3.5, the output X of the algorithm satisfies $x^-(V) \geq f(X) - n^2\delta > f(X) - 1$. It follows from Lemma 2.3 and the integrality of f that X is a minimizer of f. ■

Theorem 3.7: *Algorithm* SFM *runs in* $\mathrm{O}(n^7 \log(nM))$ *time.*

Proof. The algorithm starts with $\delta = M$ and ends with $\delta < 1/n^2$, so the algorithm consists of $\mathrm{O}(\log(nM))$ scaling phases. Each scaling phase finds $\mathrm{O}(n^2)$ δ-augmenting paths. To find an augmenting path, we scan each admissible pair (i, u) for $i \in I$ and $u \in V$ at most once, and there are $\mathrm{O}(n^2)$ admissible pairs, since $|I| \leq 2n$. In addition, we make at most $n - 1$ interrupted scans. Thus, the total number of scans per augmentation is $\mathrm{O}(n^2)$. Each scan uses at most $n - 1$ push operations. When a push operation is performed, we compute the associated exchange capacity (see Lemma 2.2) and construct the Hasse diagram for the new extreme base, which takes $\mathrm{O}(n^2)$ time. Thus the total time for a scan is $\mathrm{O}(n^3)$. After each scan or each non-saturating push, we must determine a new admissible pair, which takes $\mathrm{O}(n^3)$ time by topological sort. Thus, the time spent

per augmenting path is $O(n^5)$. After each augmentation, we also update the expression $x = \sum_{i \in I} \lambda_i y_i$, which takes $O(n^3)$ time and hence is not a bottleneck. Thus the overall complexity of SFM is $O(n^7 \log(nM))$. ■

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 the duality gap $f(X) - x^-(V)$ 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 $O(n^7 \log(nM/\epsilon))$ time, where M is an upper bound on $|f(X)|$ $(X \subseteq V)$.

4. A Strongly Polynomial-Time Algorithm

This section presents a strongly polynomial-time algorithm for minimizing submodular functions using the scaling algorithm in Section 3. The new algorithm exploits the following proximity lemma.

Lemma 4.1: *At the end of the δ-scaling phase, if $x(w) < -n^2\delta$, then w belongs to every minimizer of f.*

Proof. Let X be any minimizer of f. Since $x^- \in P(f)$, there exists a vector $y \in P(f)$ with $x^- \leq y \leq \mathbf{0}$ such that $y(V) = f(X)$. Note that $y(v) = 0$ for each $v \in V \backslash X$. By Lemma 3.5, there exists a subset $Y \subseteq V$ such that $x^-(V) \geq f(Y) - n^2\delta$. Then we have $y(w) - x(w) \leq y(V) - x^-(V) \leq f(X) - f(Y) + n^2\delta \leq n^2\delta$. This implies $y(w) < 0$ due to the assumption, and hence $w \in X$. ■

Let $f : 2^V \to \mathbf{R}$ be a submodular function and $x \in B(f)$ an extreme base whose components are bounded from above by $\eta > 0$. Assume that there exists a subset $Y \subseteq V$ such that $f(Y) \leq -\kappa$ for some positive parameter κ. We then apply the scaling algorithm starting with $\delta = \eta$ and the extreme base $x \in B(f)$. After $\lceil \log_2(n^3\eta/\kappa) \rceil$ scaling phases, δ becomes less than κ/n^3. Since $x(Y) \leq f(Y) \leq -\kappa$, at least one element $w \in Y$ satisfies $x(w) < -n^2\delta$. By Lemma 4.1, such an element w belongs to every minimizer of f. We denote this procedure by Fix(f, x, η).

We now discuss how to apply this procedure to design a strongly polynomial-time algorithm for minimizing a submodular function f. If $f(V) > 0$, we replace the value $f(V)$ by zero. The set of minimizers remains the same unless the minimum value is zero, in which case we may assert that $\emptyset$ minimizes f.

An ordered pair (u, v) of distinct vertices $u, v \in V$ is said to be *compatible* with f if $u \in X$ implies $v \in X$ for every minimizer X of f. Our algorithm keeps a directed acyclic

graph $D = (V, F)$ whose arcs are compatible with f. Initially, the arc set F is empty. Each time the algorithm finds a compatible pair (u, v) with f, it adds (u, v) to F. When this gives rise to a cycle in D, the algorithm contracts the strongly connected component $U \subseteq V$ to a single vertex and modifies the submodular function f by regarding U as a singleton.

For each $v \in V$, let $R(v)$ denote the set of vertices reachable from v in D and f_v the submodular function on the subsets of $V \backslash R(v)$ defined by

$$f_v(X) = f(X \cup R(v)) - f(R(v)) \qquad (X \subseteq V \backslash R(v)).$$

An ordering $(v_1, \cdots, v_n)$ of V is called *consistent* with D if $i < j$ implies $(v_i, v_j) \notin F$. Consider an extreme base $x \in \mathrm{B}(f)$ obtained by the greedy algorithm with a consistent ordering $(v_1, \cdots, v_n)$. That is, let $x(v_1) = f(v_1)$ and $x(v_j) = f(\{v_1, v_2, \ldots, v_j\}) - f(\{v_1, v_2, \ldots, v_{j-1}\})$ for $j = 2, \ldots, n$. The extreme base obtained from a consistent ordering is also called *consistent*. Then it follows from the submodularity of f that any consistent extreme base $x \in \mathrm{B}(f)$ satisfies $x(v) \leq f(R(v)) - f(R(v) \backslash \{v\})$ for each $v \in V$.

In each iteration, the algorithm computes

$$\eta = \max\{f(R(v)) - f(R(v) \backslash \{v\}) \mid v \in V\}. \tag{4.1}$$

If $\eta \leq 0$, an extreme base $x \in \mathrm{B}(f)$ consistent with D satisfies $x(v) \leq 0$ for each $v \in V$, which implies that V minimizes f. If in addition $f(V) = 0$, then the original function may have had a positive value of $f(V)$. Therefore, the algorithm returns $\emptyset$ or V as a minimizer, according to whether $f(V) = 0$ or $f(V) < 0$.

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

In the former case ($f(R(u)) \geq \eta/2$), we have $f_u(V \backslash R(u)) = f(V) - f(R(u)) \leq -\eta/2$. The algorithm finds a consistent extreme base $x \in \mathrm{B}(f_u)$ by the greedy algorithm with an ordering $(v_1, \cdots, v_k)$ of $V \backslash R(u)$ such that $i < j$ implies $(v_i, v_j) \notin F$. That is, let $x(v_1) = f(v_1)$ and $x(v_j) = f(\{v_1, v_2, \ldots, v_j\}) - f(\{v_1, v_2, \ldots, v_{j-1}\})$ for $j = 2, \ldots, k$. Then the extreme base $x \in \mathrm{B}(f_u)$ satisfies $x(v) \leq f(R(v)) - f(R(v) \backslash \{v\}) \leq \eta$. Thus we may apply the procedure $\mathsf{Fix}(f_u, x, \eta)$ to find an element $w \in V \backslash R(u)$ that belongs to every minimizer of f_u. Since $\kappa = \eta/2$, the procedure terminates within $\mathrm{O}(\log n)$ scaling phases. Consequently, we obtain a new pair (u, w) that is compatible with f. Hence the algorithm adds the arc (u, w) to F.

In the latter case ($f(R(u) \backslash \{u\}) < -\eta/2$), we compute an extreme base $x \in \mathrm{B}(f)$ consistent with D by the greedy algorithm, and then apply the procedure $\mathsf{Fix}(f, x, \eta)$ to find an element $w \in R(u)$ that belongs to every minimizer of f. Since $x(v) \leq \eta$ for every $v \in V$ and $\kappa = \eta/2$ again, the procedure terminates within $\mathrm{O}(\log n)$ scaling phases. Note that every minimizer of f includes $R(w)$. Thus it suffices to minimize the submodular

Input: $f : 2^V \to \mathbf{R}$
Output: $X \subseteq V$ minimizing f

Initialization:
    $X \leftarrow \emptyset$
    $F \leftarrow \emptyset$
While $V \neq \emptyset$ **do**
    If $f(V) > 0$ **then** $f(V) \leftarrow 0$
    $\eta \leftarrow \max\{f(R(v)) - f(R(v)\backslash\{v\}) \mid v \in V\}$
    If $\eta \leq 0$ **then break**
    Let $u \in V$ attain the maximum above.
    If $f(R(u)) \geq \eta/2$ **then**
        Find a consistent extreme base $x \in \mathrm{B}(f_u)$ by the greedy algorithm.
        $w \leftarrow \mathsf{Fix}(f_u, x, \eta)$
        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 $x \in \mathrm{B}(f)$ by the greedy algorithm.
        $w \leftarrow \mathsf{Fix}(f, x, \eta)$
        $V \leftarrow V\backslash R(w)$
        $f \leftarrow f_w$
        Find a subset Q of the original underlying set represented by $R(w)$.
        $X \leftarrow X \cup Q$
If $f(V) < 0$ **then**
        Find a subset Q of the original underlying set represented by V.
        $X \leftarrow X \cup Q$
End.

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

function f_w, which is now defined on a smaller underlying set. Figure 3 provides a formal description of the strongly polynomial-time algorithm.

Theorem 4.2: *The algorithm in* Figure 3 *computes the minimizer of a submodular function in* $O(n^9 \log n)$ *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 n^2 times. Thus the overall running time of the algorithm is $O(n^9 \log n)$, 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.

An easy way to minimize such a function f is to consider the reduction of f by a sufficiently large vector. As described in [12, p. 56], we can compute an upper bound $\hat{M}$ on $|f(X)|$ $(X \in \mathcal{D})$. Let f' be the rank function of the reduction by a vector with each component being equal to $\hat{M}$. The submodular function f' is defined on 2^V and the set of minimizers of f' coincides with that of f. Thus, we may apply our algorithms. However, each evaluation of the function value of f' requires $O(n^2)$ elementary operations in addition to a single call for the evaluation of f. Consequently, this approach takes $O(n^9 \min\{\log(n\hat{M}), n^2 \log n\})$ time.

Alternatively, we can slightly extend the algorithms in Sections 3 and 4 by keeping the base $x \in B(f)$ as a convex combination of extreme bases y_i's plus a vector in the characteristic cone of $B(f)$. 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 $O(n^7 \min\{\log(n\hat{M}), n^2 \log n\})$ time.

Submodular functions defined on modular lattices naturally arise in linear algebra. Minimization of such functions has a significant application to computing canonical forms of partitioned matrices [18, 20]. 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.

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

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

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

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

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

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

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

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

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

[17] B. Hoppe and É. Tardos: The quickest transsihpment problem, *Proceedings of 5th ACM/SIAM Symposium on Discrete Algorithms* (1995), 512–521.

[18] 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.

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

[20] 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.

[21] 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.

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

[23] H. Nagamochi and T. Ibaraki: A note on minimizing submodular functions, *Inform. Process. Lett.*, **67** (1998), 239–244.

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

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

[26] 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.

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