A Combinatorial Strongly Polynomial Algorithm for Minimizing Submodular Functions

SATORU IWATA

University of Tokyo, Tokyo, Japan

LISA FLEISCHER

Carnegie Mellon University, Pittsburgh, Pennsylvania

AND

SATORU FUJISHIGE

Osaka University, Osaka, Japan

Abstract. This paper presents a 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 length of the largest absolute function value. The paper also presents a strongly polynomial version in which the number of steps is bounded by a polynomial in the size of the underlying set, independent of the function values.

A preliminary version has appeared in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing* (Portland, Ore.). ACM, New York, 2000, pp. 96–107.

The work of S. Iwata was done at Osaka University, Toyonaka, Japan and at the Fields Institute, Toronto, Ont., Canada.

The work of S. Iwata was partly supported by Grants-in-Aid for Scientific Research from the Ministry of Education, Science, Sports, and Culture of Japan.

The work of L. Fleischer was done while on leave at the Center for Operations Research and Econometrics, Universite Catholique de Louvain, Belgium and at the Fields Insititute, Toronto, Ont., Canada.

The work of L. Fleischer was supported in part by DONET, a European network supported by the European Community with the frame of the Training and Mobility of Researchers Programme (Contract number ERB TMRX-CT98-0202). Additional support was from the University of Waterloo, Dept. of Combinatorics and Optimization, and the National Science Foundation (NSF) through grants INT 99-02663 and EIA 99-73858.

The work of S. Fujishige was partly supported by Grants-in-Aid for Scientific Research from the Ministry of Education, Science, Sports, and Culture of Japan.

Authors' addresses: S. Iwata, Department of Mathematical Engineering and Information Physics, University of Tokyo, Tokyo 113-8656, Japan, e-mail: iwata@sr3.t.u-tokyo.ac.jp; L. Fleischer, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213, e-mail: lkf@andrew.cmu.edu; S. Fujishige, 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.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*

General Terms: Algorithms

Additional Key Words and Phrases: Discrete optimization, strongly polynomial algorithm, submodular function

1. *Introduction*

Let V be a finite nonempty set of cardinality n. A function f defined on all the subsets of 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 a combinatorial polynomial-time algorithm for finding a minimizer of a general submodular function, provided that an oracle for evaluating the function value is available. Throughout this paper, we assume without loss of generality that $f(\emptyset) = 0$ by subtracting the scalar $f(\emptyset)$ from every function value.

Submodularity is a discrete analog of convexity [Frank 1982; Fujishige 1984b; Lovász 1983], and submodular functions arise naturally in various fields, including combinatorial optimization, computational biology, game theory, scheduling, probability, and information theory. Examples include the matroid rank function, the cut capacity function, and the entropy function. Problems in diverse areas such as dynamic flows [Hoppe and Tardos 2000], facility location [Tamir 1993], and multiterminal source coding [Fujishige 1978; Han 1979] rely on algorithms for general submodular function minimization. Submodular function minimization is also used to solve submodular flow problems [Cunningham and Frank 1985; Edmonds and Karp 1972; Fujishige and Iwata 2000] which generalize network flow and matroid optimization problems, and model several graph augmentation and connectivity problems [Edmonds and Giles 1977; Frank and Tardos 1988; 1989]. For general background on submodular functions, see Frank and Tardos [1998], Fujishige [1991], and Lovász [1983].

There are two natural polyhedra in $\mathbf{R}^V$ associated with a submodular function f. The *submodular polyhedron* $\mathrm{P}(f)$ and the *base polyhedron* $\mathrm{B}(f)$ are defined by

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

where $x(X) = \sum_{v \in X} x(v)$. Linear optimization problems over these polyhedra can be solved efficiently by the greedy algorithm [Edmonds 1970].

The first polynomial-time algorithm for submodular function minimization is due to Grötschel et al. [1981]. They showed, in general, the polynomial-time equivalence of separation and optimization for polyhedra via the ellipsoid method. The separation problem of deciding whether $\mathbf{0} \in \mathrm{P}(f^{\mu})$, for the submodular function f^{μ} defined by subtracting scalar $\mu \leq 0$ from $f(X)$ for every nonempty $X \subseteq V$, is equivalent to determining if the minimum of the submodular function f is at least μ. This problem can be solved using the ellipsoid algorithm in conjunction with the greedy algorithm that solves the optimization problem over $\mathrm{P}(f^{\mu})$. Since the maximum value μ^* of μ with $\mathbf{0} \in \mathrm{P}(f^{\mu})$ equals the minimum value of f, embedding the ellipsoid algorithm in a binary search for μ^* yields a polynomial-time algorithm for submodular function minimization. However, the ellipsoid method is far from being efficient in practice and is not combinatorial.

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 [1984] for minimizing submodular functions that arise from the separation problem for matroid polyhedra. This was adapted for general submodular function minimization by Bixby et al. [1985] and improved by Cunningham [1985] to obtain the first combinatorial, pseudopolynomial-time algorithm. More recently, Narayanan [1995] introduced a rounding technique that improves Cunningham's algorithm for matroid polyhedra. Based on a minimum-norm base characterization of minimizers [Fujishige 1980; 1984a], Sohoni [1992] devised another pseudopolynomial-time algorithm. For a closely related problem of finding a nonempty proper subset that minimizes a symmetric submodular function f, Queyranne [1998] has described a combinatorial strongly polynomial algorithm. (A symmetric set function f satisfies $f(X) = f(V \backslash X)$ for all $X \subseteq V$.) Queyranne's algorithm extends the undirected minimum-cut algorithm of Nagamochi and Ibaraki [1992].

A fundamental tool in the above algorithms for general submodular function minimization [Bixby et al. 1985; Cunningham 1984; 1985; Narayanan 1995] 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 with a vertex set given by the underlying set of the submodular function, and with an arc set that represents a set of possible exchange operations. They progress by iteratively performing a sequence of exchange operations along an augmenting path. These algorithms are not known to be polynomial since the best-known lower bound on the amount of each augmentation is too small. The amount of augmentation is determined by exchange capacities multiplied by the convex combination coefficients. These coefficients may be as small as the reciprocal of the maximum absolute value of the submodular function.

To make a pseudopolynomial-time algorithm run in polynomial time, Edmonds and Karp [1972] introduced the scaling technique in the design of the first polynomial-time minimum cost flow algorithm. Since this initial success, there have been many polynomial-time scaling algorithms designed for various combinatorial optimization problems. However, a straightforward attempt to apply the scaling technique does not work for submodular function minimization. This is mainly because rounding a submodular function may violate the submodularity. More specifically, the set function f' defined by $f'(X) = \lfloor f(X) \rfloor$ is not necessarily submodular even if f is a 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 the scaling parameter δ. The complete directed graph serves as a relaxation of the submodular function f to another submodular function f_δ defined by $f_\delta(X) = f(X) + \delta\,|X| \cdot |V \backslash X|$. Note that the second term $\delta\,|X| \cdot |V \backslash X|$ is the cut function of this additional graph, and hence submodular.

The relaxation f_δ has a natural interpretation in the setting of network flows. In their cut-canceling algorithm for minimum cost flows, Ervolina and McCormick

[1993] relax the capacity of each flow arc by the scaling parameter δ. For submodular function minimization, the "graph" is the set of possible exchange arcs, which is really the complete directed graph on V.

The use of this additional graph was first introduced by Iwata [1997] as the first capacity-scaling algorithm for submodular flow. Since that paper was published, the submodular flow algorithms of Iwata et al. [1999] and Fleischer et al. [2001] have developed the techniques further. In particular, incorporating ideas from Iwata et al. [1999], the algorithm in Fleischer et al. [2001] introduces a method to avoid exchange operations on an augmenting path. This is done by carefully performing exchange operations during the search for an augmenting path of sufficient residual capacity. Our work in the present paper employs this technique to develop a capacity scaling, augmenting path algorithm for submodular function minimization.

The resulting algorithm uses $O(n^5 \log M)$ arithmetic steps and function evaluations, where $M = \max\{|f(X)| \mid X \subseteq V\}$. Even under the assumption that M is bounded by a constant, our scaling algorithm is faster than the best previous combinatorial, pseudopolynomial-time algorithm due to Cunningham [1985], which uses $O(n^6 M \log (nM))$ arithmetic steps and function evaluations.

We then modify our scaling algorithm to run in strongly polynomial time. A strongly polynomial algorithm for submodular function minimization performs a number of steps bounded by a polynomial in the size of the underlying set, independent of M. Grötschel et al. [1988] described the first such algorithm using the ellipsoid method. To make a polynomial-time algorithm run in strongly polynomial time, Frank and Tardos [1987] developed a generic preprocessing technique that is applicable to a fairly wide class of combinatorial optimization problems including submodular flow (assuming an oracle for computing exchange capacities) and testing membership in matroid polyhedra. However, this framework does not readily apply to our scaling algorithm for 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, a new element not contained in any minimizer, or a new ordered pair $(u, v) \in V$ such that any minimizer containing u also contains v. The resulting algorithm uses $O(n^7 \log n)$ arithmetic steps and function evaluations. Our approach is based on the general technique originated by Tardos [1985] in the design of the first strongly polynomial minimum cost flow algorithm.

Independently, Schrijver [2000] has also developed a combinatorial strongly polynomial algorithm for general submodular function minimization based on Cunningham's approach. Instead of designing an algorithm that uses provably large augmentations as we do here, Schrijver's complexity analysis depends on an algorithmic framework that uses paths whose lengths are provably nondecreasing. His algorithm can be shown to use $O(n^8)$ function evaluations and $O(n^9)$ arithmetic steps. A modification of this algorithm improves both of these quantities by a linear factor [Fleischer and Iwata 2000]. Both Schrijver's algorithm and ours use Gaussian elimination to maintain the representation of a vector in $\mathrm{B}(f)$ as the convex combination of a small number of extreme points. However, we do not require this step to establish the polynomial time complexity of our algorithm.

Schrijver [2000] poses an open problem to design a strongly polynomial algorithm for submodular function minimization that consists only of additions, subtractions, comparisons, and oracle calls. The symmetric submodular function minimization algorithm of Queyranne [1998] is "fully combinatorial" in this sense.

Iwata [2001] has very recently answered this question by describing a fully combinatorial implementation of the strongly polynomial algorithm in the present paper.

This paper is organized as follows: Section 2 provides background on submodular functions. Section 3 presents our scaling algorithm for submodular function minimization, and Section 4 gives a strongly polynomial algorithm. In Section 5, we discuss the variants of our algorithms without Gaussian elimination. Finally, we conclude with extensions in Section 6.

2. *Preliminaries*

We denote by **Z** and **R** the set of integers and the set of reals, respectively. For any vector $x \in \mathbf{R}^V$ and any subset $X \subseteq V$, the expression $x(X)$ denotes $\sum_{v \in X} x(v)$. For any vector $x \in \mathbf{R}^V$, we denote by x^+ and x^- the vectors in $\mathbf{R}^V$ defined by $x^+(v) = \max\{0, x(v)\}$ and $x^-(v) = \min\{0, x(v)\}$ for $v \in V$. For each $u \in V$, let χ_u denote the vector in $\mathbf{R}^V$ such that $\chi_u(u) = 1$ and $\chi_u(v) = 0$ for $v \in V \backslash \{u\}$.

A vector in the base polyhedron $\mathrm{B}(f)$ is called a *base*, and an extreme point of $\mathrm{B}(f)$ an *extreme base*. It is easy to see that for any base $x \in \mathrm{B}(f)$ and any subset $Y \subseteq V$ we have $x^-(V) \leq x(Y) \leq f(Y)$. The following fundamental lemma shows that these inequalities are in fact tight for appropriately chosen x and Y. Although the lemma easily follows from a theorem of Edmonds [1970] on the vector reduction of polymatroids, we provide a direct proof for completeness.

LEMMA 2.1. *For a submodular function $f \colon 2^V \to \mathbf{R}$, we have*

$$\mathit{max}\{x^-(V) \mid x \in \mathrm{B}(f)\} = \mathit{min}\{f(X) \mid X \subseteq V\}. \tag{2.1}$$

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

PROOF. Let x be a maximizer in the left-hand side. For any $s, t \in V$ with $x(s) < 0$ and $x(t) > 0$, there exists a subset X_{st} such that $s \in X_{st} \subseteq V \backslash \{t\}$ and $x(X_{st}) = f(X_{st})$. Then it follows from the submodularity of f that

$$X = \bigcup_{s:x(s)<0} \bigcap_{t:x(t)>0} X_{st}$$

satisfies $x(X) = f(X)$. Since $x(u) \leq 0$ for every $u \in X$ and $x(v) \geq 0$ for every $v \in V \backslash X$, we have $x^-(V) = x(X) = f(X)$, which establishes the min-max relation. The integrality assertion follows from the same argument starting with an integral base x that maximizes $x^-(V)$ over all integral bases. □

It is not completely obvious that Lemma 2.1 provides a good characterization of a minimizer of f. In fact, proving $x \in \mathrm{B}(f)$ by the definition would require exponential number of function evaluations. If y is an extreme base, however, there is a compact proof that $y \in \mathrm{B}(f)$ resulting from the greedy algorithm described below. However, the maximizer of (2.1) may not be an extreme base. To handle this, Cunningham [1984; 1985] suggested maintaining a base $x \in \mathrm{B}(f)$ as a convex combination of extreme bases, thus yielding a compact proof that $x \in \mathrm{B}(f)$ for any base x generated by his algorithm.

Let $L = (v_1, \ldots, v_n)$ be a linear ordering of V. For any $j \in \{1, \ldots, n\}$, we define $L(v_j) = \{v_1, \ldots, v_j\}$. The greedy algorithm of Edmonds [1970] and Shapley [1971] computes an extreme base $y \in \mathrm{B}(f)$ associated with L by

$$y(v) := f(L(v)) - f(L(v)\backslash\{v\}), \qquad \forall v \in V. \tag{2.2}$$

Thus, the linear ordering L provides a certificate that y is an extreme base. Conversely, any extreme base can be generated by applying the greedy algorithm to an appropriate linear ordering.

A fundamental tool in our algorithm is to move from a base x to another base by an *exchange operation* that increases one component and decreases another component by the same amount, that is, $x := x + \alpha(\chi_u - \chi_v)$. With $x = \sum_{i \in I} \lambda_i y_i$, a convex combination of extreme bases, this can be realized by applying an exchange operation on an extreme base y_i. An exchange amount β on y_i corresponds to an exchange amount $\lambda_i \beta$ on x. The following lemma shows that interchanging two consecutive elements in a linear ordering that generates y_i results in an exchange operation on y_i.

LEMMA 2.2. *Suppose u immediately succeeds v in a linear ordering L that generates an extreme base $y \in \mathrm{B}(f)$. Then the linear ordering L' obtained from L by interchanging u and v generates an extreme base $y' = y + \tilde{c}(y, u, v)(\chi_u - \chi_v)$ with*

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

PROOF. It is obvious from the greedy algorithm that y' can differ from y only at u and v. Namely, $y' = y + \beta(\chi_u - \chi_v)$ for some β. Since $L'(v) = L(u)$, it follows from (2.2) that $y'(v) = f(L(u)) - f(L(u)\backslash\{v\})$. Thus, we obtain $\beta = f(L(u)\backslash\{v\}) - f(L(u)) + y(v)$. □

The quantity $\tilde{c}(y, u, v)$ in Lemma 2.2 is called an *exchange capacity*. In general, an exchange capacity $\tilde{c}(x, u, v)$ is defined for any base $x \in \mathrm{B}(f)$ and any ordered pair of distinct $u, v \in V$ as the maximum amount of exchange operation that keeps x in the base polyhedron. Hence, the exchange capacity $\tilde{c}(x, u, v)$ is expressed as

$$\tilde{c}(x, u, v) = \min\{f(X) - x(X) \mid u \in X \subseteq V\backslash\{v\}\}. \tag{2.4}$$

However, there is nothing special that makes this computation easier than minimizing f. Our algorithm uses only those exchange capacities that can be computed via Lemma 2.2.

3. *A Scaling Algorithm*

In this section, we describe a combinatorial algorithm for minimizing an integer-valued submodular function f: $2^V \rightarrow \mathbf{Z}$ with $f(\emptyset) = 0$. We assume we have an evaluation oracle for the function value of f. Our algorithm is an augmenting path algorithm, embedded in a scaling framework. A formal description of this algorithm SFM appears in Figure 1.

3.1. THE SCALING FRAMEWORK. The algorithm consists of scaling phases with a positive parameter δ. The algorithm starts with an arbitrary linear ordering L on V and the extreme base $x \in \mathrm{B}(f)$ generated by L. The initial value of δ is given by $\delta := \xi/n^2$ with $\xi = \min\{|x^-(V)|, x^+(V)\}$. At the end of each scaling phase, the algorithm cuts δ in half. The algorithm ends with $\delta < 1/n^2$.

```
SFM($f$):

Initialization:
    $L \leftarrow$ a linear ordering on $V$
    $x \leftarrow$ an extreme base in $\mathrm{B}(f)$ generated by $L$
    $\delta \leftarrow \min\{|x^-(V)|, x^+(V)\}/n^2$
    $I \leftarrow \{k\}$, $y_k \leftarrow x$, $\lambda_k \leftarrow 1$, $L_k \leftarrow L$
    $\varphi \leftarrow \mathbf{0}$,
While $\delta \geq 1/n^2$ do                      [ Scaling phase ]
    $S \leftarrow \{v \mid x(v) + \partial\varphi(v) \leq -\delta\}$
    $T \leftarrow \{v \mid x(v) + \partial\varphi(v) \geq \delta\}$
    $W \leftarrow$ the set of vertices reachable from $S$ in $G^\circ$
    While $W \cap T \neq \emptyset$ or there is an active triple do
        While $W \cap T = \emptyset$ and there is an active triple do
            Apply Double-Exchange to an active triple $(i, u, v)$.
            Update $W$.
        If $W \cap T \neq \emptyset$ then          [ There is a $\delta$-augmenting path. ]
            Augment flow $\varphi$ along a $\delta$-augmenting path $P$.
            Update $G^\circ$, $S$, $T$, $W$.
        Apply Reduce($x, I$).
    $\delta \leftarrow \delta/2$
    $\varphi \leftarrow \varphi/2$
Return $W$.
End.
```

FIG. 1. A scaling algorithm for submodular function minimization. The algorithm finds a subset of V that minimizes submodular function f. It uses a directed graph $G^\circ = (V, A^\circ)$ where $A^\circ = \{(u, v) \mid u, v \in V,\ u \neq v,\ \varphi(u, v) = 0\}$.

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 $\varphi\colon V \times V \to \mathbf{R}$ be a flow in the complete directed graph $G = (V, A)$ with the vertex set V and the arc set $A = V \times V$. The *boundary* $\partial\varphi\colon V \to \mathbf{R}$ is defined by

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

That is, $\partial\varphi(u)$ is the net flow value emanating from u. A flow φ is called δ*-feasible* if it satisfies capacity constraints $0 \leq \varphi(u, v) \leq \delta$ for every $u, v \in V$. Our algorithm maintains φ such that at least one of $\varphi(u, v)$ and $\varphi(v, u)$ is equal to zero for any $u, v \in V$.

The algorithm maintains a base $x \in \mathrm{B}(f)$ as a convex combination $x = \sum_{i \in I} \lambda_i y_i$ of 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. Instead of trying to maximize $x^-(V)$ directly, the algorithm uses $z = x + \partial\varphi$ and seeks to increase $z^-(V)$, thereby increasing $x^-(V)$ via the δ-feasibility of φ. This z is a base in the base polyhedron of the submodular function $f_\delta(X) = f(X) + \delta|X| \cdot |V \backslash X|$.

3.2. A SCALING PHASE. Each δ-scaling phase maintains a δ-feasible flow φ and a subgraph $G^\circ = (V, A^\circ)$ with the arc set $A^\circ = \{(u, v) \mid u, v \in V,\ u \neq v,\ \varphi(u, v) = 0\}$. The δ-scaling phase aims at increasing $z^-(V)$ by sending flow along directed paths in G° 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*.

If there is no δ-augmenting path, let W denote the set of vertices currently reachable from S in G°. A triple (i, u, v) of $i \in I$, $u \in W$ and $v \in V \backslash W$ is called *active* if u immediately succeeds v in L_i. If there is an active (i, u, v), the algorithm performs an appropriate exchange operation, and modifies φ so that $z = x + \partial\varphi$ is invariant. We refer to this procedure as Double-Exchange(i, u, v). The detail of Double-Exchange is described below. As a result of Double-Exchange(i, u, v), either W remains unchanged or the vertex v and the set of vertices in $V \backslash W$ reachable from v by δ-capacity paths are added to W. The algorithm performs Double-Exchange as long as it is applicable, until a δ-augmenting path is found. Once a δ-augmenting path P is found, the algorithm augments the flow φ by δ along P by setting $\varphi(u, v) := \delta - \varphi(v, u)$ and $\varphi(v, u) := 0$ for each arc (u, v) in P. This increases $z^-(V)$ by δ since z changes only at the initial and terminal vertices of P. This is an extension of a technique developed in Fleischer et al. [2001] for finding δ-augmenting paths for submodular flows.

A δ-scaling phase ends when there is neither a δ-augmenting path nor an active triple. Then the algorithm cuts the value of δ in half and goes to the next scaling phase. To keep the δ-feasibility of φ, the algorithm halves the flow φ for each arc.

The first step of the procedure Double-Exchange(i, u, v) is to compute the exchange capacity $\tilde{c}(y_i, u, v)$. It then updates x and φ as $x := x + \alpha(\chi_u - \chi_v)$ and $\varphi(u, v) := \varphi(u, v) - \alpha$, so that $z = x + \partial\varphi$ remains unchanged. The amount α of this exchange operation is determined by taking the minimum of $\varphi(u, v)$ and $\lambda_i \tilde{c}(y_i, u, v)$. Note that $\varphi(u, v)$ is the maximum amount of feasible decrease of flow on (u, v) and that $\lambda_i \tilde{c}(y_i, u, v)$ 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) updates $y_i := y_i + \tilde{c}(y_i, u, v)(\chi_u - \chi_v)$ and $\lambda_i := \alpha/\tilde{c}(y_i, u, v)$. It also updates L_i by interchanging u and v, which maintains y_i as an extreme base generated by L_i. Double-Exchange(i, u, v) is called *saturating* if $\alpha = \lambda_i \tilde{c}(y_i, u, v)$. Otherwise, it is called *nonsaturating*. If Double-Exchange(i, u, v) is nonsaturating, the old y_i remains in the convex representation of x with coefficient $\lambda_i - \alpha/\tilde{c}(y_i, u, v)$. Thus, if Double-Exchange(i, u, v) is nonsaturating, then before updating y_i, it adds to I a new index k with $y_k := y_i$, $\lambda_k := \lambda_i - \alpha/\tilde{c}(y_i, u, v)$, and $L_k := L_i$. Double-Exchange(i, u, v) is summarized in Figure 2.

After each δ-augmentation, and at the end of the δ-scaling phase, the algorithm applies a procedure Reduce(x, I) that 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 a feasible solution into a basic feasible solution. 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 $\sum \mu_i y_i = \mathbf{0}$ 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

```
Double-Exchange$(i, u, v)$:
$\tilde{c}(y_i, u, v) \leftarrow f(L_i(u)\backslash\{v\}) - f(L_i(v)) + y_i(v)$
$\alpha \leftarrow \min\{\varphi(u, v),\ \lambda_i \tilde{c}(y_i, u, v)\}$
$x \leftarrow x + \alpha(\chi_u - \chi_v)$
$\varphi(u, v) \leftarrow \varphi(u, v) - \alpha$
If $\alpha < \lambda_i \tilde{c}(y_i, u, v)$ then
    $k \leftarrow$ a new index
    $I \leftarrow I \cup \{k\}$
    $\lambda_k \leftarrow \lambda_i - \alpha/\tilde{c}(y_i, u, v)$
    $\lambda_i \leftarrow \alpha/\tilde{c}(y_i, u, v)$
    $y_k \leftarrow y_i$
    $L_k \leftarrow L_i$
$y_i \leftarrow y_i + \tilde{c}(y_i, u, v)(\chi_u - \chi_v)$
Update $L_i$ by interchanging $u$ and $v$.
```

FIG. 2. Algorithmic description of the procedure Double-Exchange(i, u, v).

such i from I. We continue this procedure until we eventually obtain affine independence.

This same step is also used in the submodular function minimization algorithms of Cunningham [1985] and Schrijver [2000]. However, for the present algorithm, we do not need this step to obtain a polynomial bound on the complexity. We include this linear algebraic procedure because it significantly reduces the running time of the algorithm. For an analysis of the algorithm without Reduce, see Section 5.

3.3. CORRECTNESS. In the subsequent analysis, the end of a scaling phase refers to the point immediately before δ is cut in half. The following lemma establishes a relaxed strong duality, which plays a crucial role in the analysis of our algorithm.

LEMMA 3.1. *At the end of the δ-scaling phase, $z^-(V) \geq f(W) - n\delta$.*

PROOF. At the end of the δ scaling phase, there are no active triples, which implies for each $i \in I$ the first $|W|$ vertices in L_i must belong to W. Then it follows from (2.2) that $y_i(W) = f(W)$. Since $x = \sum_{i \in I} \lambda_i y_i$ and $\sum_{i \in I} \lambda_i = 1$, we obtain $x(W) = \sum_{i \in I} \lambda_i y_i(W) = f(W)$.

At the end of the δ scaling phase, the set W also satisfies $S \subseteq W \subseteq V \backslash T$ and $\partial\varphi(W) \geq 0$. By the definitions of S and T, we have $z(v) < \delta$ for every $v \in W$ and $z(v) > -\delta$ for every $v \in V \backslash W$. Therefore, we have $z^-(V) = z^-(W) + z^-(V \backslash W) \geq z(W) - \delta|W| - \delta|V \backslash W| = x(W) + \partial\varphi(W) - n\delta \geq f(W) - n\delta$. □

As an immediate consequence of Lemma 3.1, we obtain the following lemma, which leads us to the correctness of the scaling algorithm.

LEMMA 3.2. *At the end of the δ-scaling phase, $x^-(V) \geq f(W) - n^2\delta$.*

PROOF. By Lemma 3.1, the set W satisfies $z^-(V) \geq f(W) - n\delta$. Since $\partial\varphi(v) \leq (n-1)\delta$ for each $v \in V$, we have $x^-(V) \geq z^-(V) - n(n-1)\delta \geq f(W) - n^2\delta$. □

THEOREM 3.3. *The algorithm obtains a minimizer of f at the end of the δ-scaling phase with $\delta < 1/n^2$.*

PROOF. By Lemma 3.2, the output W of the algorithm satisfies $x^-(V) \geq f(W) - n^2\delta > f(W) - 1$. For any $Y \subseteq V$, the weak duality in Lemma 2.1 asserts $x^-(V) \leq f(Y)$, which implies $f(W) - 1 < f(Y)$. Hence, it follows from the integrality of f that W minimizes f. □

3.4. COMPLEXITY. We now investigate the number of iterations in each scaling phase.

LEMMA 3.4. *The number of augmentations per scaling phase is $O(n^2)$.*

PROOF. It follows from Lemma 3.1 that at the beginning of the δ-scaling phase except for the first one, $z^-(V)$ is at least $f(X) - 2n\delta$ for some $X \subseteq V$. Replacing φ by $\varphi/2$ could decrease $z(X)$ by at most $|X| \cdot |V \backslash X|\delta$ which is bounded above by $n^2\delta/4$ for any X, and hence the decrease of $z^-(V)$ is also bounded by $n^2\delta/4$. Hence, $f(X) - 2n\delta - n^2\delta/4 \leq z^-(V)$. On the other hand, since $x(X) \leq f(X)$ and $\partial\varphi(X) \leq \delta|X| \cdot |V \backslash X|$, we have $z^-(V) \leq z(X) \leq f(X) + n^2\delta/4$ throughout the δ-scaling phase. Since each δ-augmentation increases $z^-(V)$ by δ, the number of δ-augmentations per phase is at most $2n + n^2/2$, which is $O(n^2)$, for all phases after the first one.

In the first phase, let x denote the initial extreme base. Then $z = x$ at the start of the algorithm. Since $z^-(V) \leq f(\emptyset) = 0$ and $z^-(V) \leq f(V) = x(V)$ must hold throughout, possible increase of $z^-(V)$ during the first scaling phase is bounded by $\xi = \min\{|x^-(V)|,\ x^+(V)\}$. Thus, the initial setting $\delta = \xi/n^2$ guarantees that the number of augmentations in the first scaling phase is n^2. □

LEMMA 3.5. *Between δ-augmentations, $|I|$ grows by at most $n - 1$.*

PROOF. A new index is added to I only during a nonsaturating Double-Exchange. Since each nonsaturating Double-Exchange adds a new element to W, this happens at most $n - 1$ times before a δ-augmenting path is found. □

LEMMA 3.6. *Algorithm* SFM *performs the procedure* Double-Exchange *$O(n^3)$ times between δ-augmentations.*

PROOF. Once the algorithm applies Double-Exchange(i, u, v), the vertices u and v are interchanged in L_i, and the triple (i, u, v) never becomes active again until the next δ-augmentation or the end of the phase. By performing basis reduction after each augmentation, $|I| < 2n$ throughout the algorithm by Lemma 3.5. Hence, the number of times Double-Exchange is applied is bounded by the number of triples, which is at most $O(n^3)$. □

THEOREM 3.7. *Algorithm* SFM *is a polynomial-time algorithm that performs $O(n^5 \log M)$ function evaluations and arithmetic operations.*

PROOF. The algorithm starts with $\delta = \xi/n^2$ and ends with $\delta < 1/n^2$. For the initial extreme base x and $X = \{v \mid v \in V,\ x(v) > 0\}$, we have $\xi \leq x^+(V) = x(X) \leq f(X) \leq M$. Thus, SFM consists of $O(\log M)$ scaling phases. Each scaling phase performs $O(n^2)$ augmentations by Lemma 3.4.

Between δ-augmentations, there are $O(n^3)$ calls of Double-Exchange by Lemma 3.6. The procedure Double-Exchange consists of $O(1)$ calls of the function evaluation oracle. Therefore, the algorithm calls the oracle $O(n^5 \log M)$ times in total.

As a result of Double-Exchange(i, u, v), the vertex v and the set of vertices in $V \backslash W$ reachable from v by δ-capacity paths may be added to W. (This set may be determined by standard graph search on G.) Thus, over the course of an augmentation, updates to W take at most n^3 time. To find an active triple efficiently, we maintain a pointer for each index $i \in I$ that points to an element of W in an active triple for L_i. After a Double-Exchange that does not increase W, this takes at most linear time to update. After a Double-Exchange that increases W, this may need to be updated for all $i \in I$, and thus takes at most n^2 time. Thus, per augmentation, this takes $O(n^3)$ time. After augmenting φ on P, the endpoints of P may be removed from S or T.

After each augmentation, we also update the expression $x = \sum_{i \in I} \lambda_i y_i$ to recover the affine independence of y_i's. The bottleneck in this procedure is the time spent for computing the coefficients μ_i, as described in Section 3.2. Since $|I| < 2n$ by Lemma 3.5, this takes $O(n^3)$ arithmetic operations. If performed correctly, the encoding length of the numbers generated by Gaussian elimination is bounded by a polynomial in the size of the input (which includes the maximum encoding length of the function values) [Edmonds 1967]. In addition, since the resulting multipliers $\lambda = (\lambda_i \mid i \in I)$ are a basic solution to the system $H\lambda = x$ where the columns of H correspond to extreme bases y_i, their size is also bounded by a polynomial in the input size. Thus, SFM is a polynomial-time algorithm with $O(n^5 \log M)$ arithmetic steps. □

The previous best known pseudopolynomial time bound is $O(n^6 M \log(nM))$ due to Cunningham [1985]. Theorem 3.7 shows that our scaling algorithm is faster than this even if M is fixed as a constant.

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 $x \in \mathrm{B}(f)$ 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 $f\colon 2^V \to \mathbf{R}$, 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 $(1/\epsilon) f$ with an $O(n^5 \log(M/\epsilon))$ bound on the number of steps, where $M = \max\{|f(X)| \mid X \subseteq V\}$.

4. *A Strongly Polynomial Algorithm*

This section presents a strongly polynomial algorithm for minimizing a real-valued submodular function $f\colon 2^V \to \mathbf{R}$. The main idea is to show via Lemma 4.1 below that after $O(\log n)$ scaling phases, the algorithm detects either a new element that is contained in every minimizer of f, a new element that is not contained in any minimizer of f, or a new vertex pair (u, v) such that v is in every minimizer of f containing u. Since there are $O(n^2)$ such detections, after $O(n^2 \log n)$ scaling phases, the algorithm finds a minimizer of f.

LEMMA 4.1. *At the end of the δ-scaling phase in* SFM(f)*, the following hold:*

(*a*) *If $x(w) < -n^2\delta$, then w is contained in every minimizer of f.*

(*b*) *If $x(w) > n^2\delta$, then w is not contained in any minimizer of f.*

PROOF. By Lemma 3.2, $x^-(V) \geq f(W) - n^2\delta$ holds at the end of the δ-scaling phase. For any minimizer X of f, we have $f(W) \geq f(X) \geq x(X) \geq x^-(X)$. Thus, $x^-(V) \geq f(W) - n^2\delta \geq x^-(X) - n^2\delta$. Therefore, if $x(w) < -n^2\delta$, then $w \in X$. On the other hand, $x^-(X) \geq x^-(V) \geq f(W) - n^2\delta \geq x(X) - n^2\delta$. Therefore, if $x(w) > n^2\delta$, then $w \notin X$. □

The strongly polynomial algorithm, denoted $\mathsf{SPM}(f)$, maintains a subset $X \subseteq V$ that is included in every minimizer of f, a vertex set $U = \{u_1, \ldots, u_\ell\}$ corresponding to a partition $\{V_1, \ldots, V_\ell\}$ of $V \backslash X$ into pairwise disjoint nonempty subsets, a submodular function $\hat{f}$ defined on 2^U, and a directed acyclic graph $D = (U, F)$. Each arc in F is an ordered pair (u, w) of vertices u and w in U such that w is in every minimizer of $\hat{f}$ containing u. For $u \in U$, let $\Gamma(u)$ denote the corresponding set of the partition of $V \backslash X$. For example, $\Gamma(u_j) = V_j$. For $Y \subseteq U$, we also denote $\Gamma(Y) = \cup_{u \in Y} \Gamma(u)$. Throughout the algorithm, we keep a correspondence between minimizers of f and $\hat{f}$ so that any minimizer of f is represented as $X \cup \Gamma(W)$ for some minimizer W of $\hat{f}$. Initially, the algorithm assigns $U := V$, $F := \emptyset$, $\hat{f} := f$, 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. We denote by $\hat{f}_u$ the *contraction* of $\hat{f}$ by $R(u)$, that is, the submodular function on ground set $U \backslash R(u)$ defined by

$$\hat{f}_u(Y) = \hat{f}(Y \cup R(u)) - \hat{f}(R(u)), \qquad \forall Y \subseteq U \backslash R(u). \tag{4.1}$$

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

LEMMA 4.2. *Any consistent extreme base $y \in \mathrm{B}(\hat{f})$ satisfies $y(u) \leq \hat{f}(R(u)) - \hat{f}(R(u) \backslash \{u\})$ for each $u \in U$.*

PROOF. The consistent extreme base y satisfies $y(u) = \hat{f}(Y) - \hat{f}(Y \backslash \{u\})$ for some $Y \supseteq R(u)$. The claim then follows from the submodularity of $\hat{f}$. □

The building block of the strongly polynomial algorithm is the subroutine $\mathsf{Fix}(\hat{f}, D, \eta)$ which performs $O(\log n)$ scaling phases starting with $\delta = \eta$, and an extreme base $y \in \mathrm{B}(\hat{f})$ that is consistent with D for submodular function $\hat{f}$. The subroutine $\mathsf{Fix}(\hat{f}, D, \eta)$ is invoked only if $\hat{f}(U) \geq \eta/3$ or there is a subset $Y \subseteq U$ such that $\hat{f}(Y) \leq -\eta/3$. $\mathsf{Fix}(\hat{f}, D, \eta)$ performs scaling phases until $\delta < \eta/3n^3$. Then, if $\hat{f}(U) \geq \eta/3$, at least one element $w \in U$ satisfies $x(w) > n^2\delta$ at the end of the last scaling phase. By Lemma 4.1 (b), such an element w is not contained in any minimizer of $\hat{f}$. Otherwise, $\hat{f}(Y) \leq -\eta/3$ and at least one element $w \in Y$ satisfies $x(w) < -n^2\delta$ at the end of the last scaling phase. By Lemma 4.1 (a), such an element w is contained in every minimizer of $\hat{f}$.

The choice of η in each call to Fix is determined so that (i) Lemma 4.1 may be invoked for a new element w after $O(\log n)$ phases, and (ii) the number of augmentations in the first phase is not too large. This is accomplished by setting η as in (4.2). We explain why (i) holds below, and why (ii) holds in the proof of Theorem 4.3

$$\eta = \max\{\hat{f}(R(u)) - \hat{f}(R(u) \backslash \{u\}) \mid u \in U\}. \tag{4.2}$$

Lemma 4.2 implies that $y(v) \leq \eta$ for any $y \in \mathrm{B}(\hat{f})$ consistent with D.

If $\eta \leq 0$, then an extreme base $y \in \mathrm{B}(\hat{f})$ consistent with D satisfies $y(u) \leq 0$ for each $u \in U$. In this case, $y^-(U) = y(U) = \hat{f}(U)$, which implies that U minimizes $\hat{f}$ by the weak duality in Lemma 2.1. Therefore, the algorithm returns U as a minimizer of $\hat{f}$.

If $\eta > 0$, then let u be an element that attains the maximum in the right-hand side of (4.2). Then, since $\eta = \hat{f}(R(u)) - \hat{f}(R(u)\backslash\{u\}) = \hat{f}(U) - \hat{f}(R(u)\backslash\{u\}) + (\hat{f}(R(u)) - \hat{f}(U))$, at least one of the three values, $\hat{f}(U)$, $-\hat{f}(R(u)\backslash\{u\})$, and $\hat{f}(R(u)) - \hat{f}(U)$, is greater than or equal to $\eta/3$. Hence, we consider the following three cases:

If $\hat{f}(U) \geq \eta/3 > 0$, the algorithm applies $\mathsf{Fix}(\hat{f}, D, \eta)$ to find a new element w that is not in any minimizer of $\hat{f}$. In this case, it suffices to minimize the function $\hat{f}$ among those subsets that do not contain any vertex v with $w \in R(v)$. Thus, the algorithm deletes $\{v \mid w \in R(v)\}$ from U.

If $\hat{f}(R(u)\backslash\{u\}) \leq -\eta/3 < 0$, the algorithm applies $\mathsf{Fix}(\hat{f}, D, \eta)$ to find a new element w in every minimizer of $\hat{f}$. In this case, every minimizer of $\hat{f}$ includes $R(w)$. Thus, it suffices to minimize the submodular function $\hat{f}_w$, defined on the smaller underlying set $U\backslash R(w)$ as in (4.1); so the algorithm sets $\hat{f} := \hat{f}_w$.

Otherwise, $(\hat{f}_u(U\backslash R(u)) = \hat{f}(U) - \hat{f}(R(u)) \leq -\eta/3 < 0)$, the algorithm applies $\mathsf{Fix}(\hat{f}_u, D_u, \eta)$ where D_u is defined as D restricted to $U\backslash R(u)$. In this case, $\mathsf{Fix}(\hat{f}_u, D_u, \eta)$ finds an element $w \in U\backslash R(u)$ that is contained in every minimizer of $\hat{f}_u$. Thus, the algorithm adds (u, w) to F. If this creates a cycle in D, then the arcs of the cycle imply that either every element in the cycle is contained in a minimizer of $\hat{f}$, or every element in the cycle is not contained in any minimizer. Thus, the algorithm contracts the cycle to a single vertex and modifies U and $\hat{f}$ by regarding the contracted vertex set as a single vertex.

This algorithm is summarized in Figure 3.

THEOREM 4.3. *Algorithm* SPM *is a strongly polynomial algorithm that performs $O(n^7 \log n)$ function evaluations and arithmetic operations.*

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 Lemma 3.4, each phase after the first phase has $O(n^2)$ augmentations.

To bound the number of augmentations in the first phase, recall that the choice of η implies that $y(v) \leq \eta$ for any extreme base $y \in \mathrm{B}(\hat{f})$ consistent with D. By submodularity of $\hat{f}$, any extreme base $y \in \mathrm{B}(\hat{f}_u)$ consistent with D_u satisfies $y(t) \leq \hat{f}_u(R(t)) - \hat{f}_u(R(t)\backslash\{t\}) \leq \hat{f}(R(t)) - \hat{f}(R(t)\backslash\{t\}) \leq \eta$ for each $t \in U\backslash R(u)$. Thus, for $\mathsf{Fix}(\hat{f}, D, \eta)$ or $\mathsf{Fix}(\hat{f}_u, D_u, \eta)$, we have $y^+(V) \leq n\eta$. Since the number of augmentations in a δ-phase is bounded by $y^+(U)/\delta$, the number of augmentations in the first phase of any call to Fix is bounded by n.

Since the proof of Theorem 3.7 shows that the number of arithmetic operations and function evaluations per augmentation is $O(n^3)$, this yields an $O(n^7 \log n)$ bound on the total number of steps.

When applied to rational-valued submodular functions, SPM works in the space of polynomial size. In particular, as noted earlier, the encoding length of the numbers generated in the Gaussian elimination is bounded by a polynomial in the input size

```
SPM($f$):

Initialization:
    $X \leftarrow \emptyset$, $U \leftarrow V$, $\hat{f} \leftarrow f$, $F \leftarrow \emptyset$
While $U \neq \emptyset$ do
    $\eta \leftarrow \max\{\hat{f}(R(u)) - \hat{f}(R(u)\backslash\{u\}) \mid u \in U\}$
    Let $u \in U$ attain the maximum above.
    If $\eta \leq 0$ then
        $X \leftarrow X \cup \Gamma(U)$
        Return $X$.
    Else
        If $\hat{f}(U) \geq \eta/3$ then
            $w \leftarrow$ Fix($\hat{f}, D, \eta$)            [ $w$ not in any minimizer. ]
            Delete $\{v \mid w \in R(v)\}$ from $U$.
        Else if $\hat{f}(R(u)\backslash\{u\}) \leq -\eta/3$ then
            $w \leftarrow$ Fix($\hat{f}, D, \eta$)            [ $w$ in every minimizer. ]
            $U \leftarrow U\backslash R(w)$
            $\hat{f} \leftarrow \hat{f}_w$
            $X \leftarrow X \cup \Gamma(R(w))$
        Else
            $w \leftarrow$ Fix($\hat{f}_u, D_u, \eta$)          [ $w$ in every minimizer containing $u$. ]
            If $u \in R(w)$ then
                Contract $\{v \mid v \in R(w),\ u \in R(v)\}$ to a single vertex.
            Else $F \leftarrow F \cup \{(u, w)\}$
Return $X$.
End.
```

FIG. 3. A strongly polynomial algorithm for submodular function minimization.

(including the maximum encoding length of the function values) [Edmonds 1967], and so is the size of the resulting multipliers λ. Thus, SPM is a strongly polynomial algorithm. □

5. *Removing Gaussian Elimination*

The algorithms described in Sections 3 and 4 both employ Gaussian elimination to get a representation of x as a convex combination of a small number of extreme bases. This step is, however, not necessary to obtain the polynomiality. We explain here the effect of removing this step.

The size of the set I in the convex combination representation of x increases by at most $n - 1$ per augmentation, due to Lemma 3.5. The number of augmentations per scaling phase is not affected by the size of I (see the proof of Lemma 3.4), and hence remains $O(n^2)$. Thus, the total number of bases introduced during the algorithm is bounded by n times the number of augmentations. For the scaling algorithm SFM(f) described in Section 3, this is $O(n^3 \log M)$.

The size of I does affect the work per augmentation, however. In particular, it affects the number of calls to Double-Exchange during the search for an augmentation. In the proof of Lemma 3.6, it is explained that the number of Double-Exchange operations before augmentation per extreme base in I is at most n^2. Thus, the total work per augmentation in the algorithm without Reduce is $O(n^5 \log M)$. Thus, the number of arithmetic operations and function evaluations used by this more combinatorial version of SFM(f) is bounded by $O(n^7 \log^2 M)$.

The strongly polynomial algorithm SPM(f) described in Section 4 does not depend on reducing the size of I for strong polynomiality. If this step is omitted, the number of extreme bases in I may grow to $O(n^3 \log n)$ in an iteration of Fix. Since each call to Fix starts with a single extreme base, the size of I will remain bounded throughout SPM(f) by $O(n^3 \log n)$. This will increase the the work per augmentation to $O(n^5 \log n)$. Thus, an overall bound on the number of steps is $O(n^9 \log^2 n)$.

In contrast, if the linear algebraic step were omitted in the strongly polynomial algorithm described in Schrijver [2000], the size of I could become exponential in n.

6. *Conclusion*

This paper has presented a strongly polynomial algorithm for minimizing submodular functions defined on Boolean lattices: all subsets of the ground set V. Several related problems have been shown to require algorithms for minimizing submodular functions on restricted families of subsets [Goemans and Ramakrishnan 1995; Grötschel et al. 1988]. These problems have combinatorial solutions modulo an oracle for submodular function minimization on distributive lattices. Our algorithms can be extended to minimize submodular functions defined on distributive lattices.

Consider a submodular function $f \colon \mathcal{D} \to \mathbf{R}$ 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 [Fujishige 1991, p. 56], we can compute an upper-bound $\hat{M}$ on $|f(X)|$ among $X \in \mathcal{D}$. Let $\hat{f}$ be the rank function of the reduction by a vector with each component being equal to $\hat{M}$. The submodular function $\hat{f}$ is defined on 2^V and the set of minimizers of $\hat{f}$ coincides with that of f. Thus, we may apply our algorithms. However, each evaluation of the function value of $\hat{f}$ requires $O(n^2)$ elementary operations in addition to a single call for the evaluation of f. Schrijver [2000] describes a similar method to solve this problem.

Alternatively, 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 $O(n^5 \min\{\log n\hat{M}, n^2 \log n\})$ 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 [Ito et al. 1994; Iwata and Murota 1995]. 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.

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

REFERENCES

BIXBY, R. E., CUNNINGHAM, W. H., AND TOPKIS, D. M. 1985. Partial order of a polymatroid extreme point. *Math. Oper. Res. 10*, 367–378.

CUNNINGHAM, W. H. 1984. Testing membership in matroid polyhedra. *J. Combinat. Theory B36*, 161–188.

CUNNINGHAM, W. H. 1985. On submodular function minimization. *Combinatorica 5*, 185–192.

CUNNINGHAM, W. H., AND FRANK, A. 1985. A primal-dual algorithm for submodular flows. *Math. Oper. Res. 10*, 251–262.

EDMONDS, J. 1967. Systems of distinct representatives and linear algebra. *J. Res. NBS 71B*, 241–245.

EDMONDS, J. 1970. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and Their Applications*. R. Guy, H. Hanani, N. Sauer, and J. Schönheim, Eds. Gordon and Breach, pp. 69–87.

EDMONDS, J., AND GILES, R. 1977. A min-max relation for submodular functions on graphs. *Ann. Discrete Math. 1*, 185–204.

EDMONDS, J., AND KARP, R. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM 19*, 2 (Apr.), 248–264.

ERVOLINA, T. R., AND MCCORMICK, S. T. 1993. Two strongly polynomial cut canceling algorithms for minimum cost network flow. *Disc. Appl. Math. 46*, 13–165.

FLEISCHER, L., AND IWATA, S. 2000. Improved algorithms for submodular function minimization and submodular flow. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing* (Portland, Ore., May 21–23). ACM, New York, pp. 107–116.

FLEISCHER, L., IWATA, S., AND MCCORMICK, S. T. 2001. A faster capacity scaling algorithm for submodular flow. *Math. Prog.*, to appear.

FRANK, A. 1982. An algorithm for submodular functions on graphs. *Ann. Discrete Math. 16*, 189–212.

FRANK, A., AND TARDOS, É. 1987. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica 7*, 49–65.

FRANK, A., AND TARDOS, É. 1988. Generalized polymatroids and submodular flows. *Math. Prog. 42*, 489–563.

FRANK, A., AND TARDOS, É. 1989. An application of submodular flows. *Linear Alg. Appl. 114/115*, 329–348.

FUJISHIGE, S. 1978. Polymatroidal dependence structure of a set of random variables. *Inf. Contr. 39*, 55–72.

FUJISHIGE, S. 1980. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Math. Oper. Res. 5*, 186–196.

FUJISHIGE, S. 1984a. Submodular systems and related topics. *Math. Prog. Study 22*, 113–131.

FUJISHIGE, S. 1984b. Theory of submodular programs: A Fenchel-type min-max theorem and subgradients of submodular functions. *Math. Prog. 29*, 142–155.

FUJISHIGE, S. 1991. *Submodular Functions and Optimization*. North-Holland, Amsterdam, The Netherlands.

FUJISHIGE, S., AND IWATA, S. 2000. Algorithms for submodular flows. *IEICE Trans. Inform. Syst. E83-D*, 322–329.

GOEMANS, M. X., AND RAMAKRISHNAN, V. S. 1995. Minimizing submodular functions over families of subsets. *Combinatorica 15*, 499–513.

GRÖTSCHEL, M., LOVÁSZ, L., AND SCHRIJVER, A. 1981. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica 1*, 169–197.

GRÖTSCHEL, M., LOVÁSZ, L., AND SCHRIJVER, A. 1988. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, New York.

HAN, T.-S. 1979. The capacity region of general multiple-access channel with correlated sources. *Inf. Cont. 40*, 37–60.

HOPPE, B., AND TARDOS, É. 2000. The quickest transshipment problem. *Math. Oper. Res. 25*, 36–62.

ITO, H., IWATA, S., AND MUROTA, K. 1994. Block-triangularization of partitioned matrices under similarity/equivalence transformations. *SIAM J. Matrix Anal. Appl. 15*, 1226–1255.

IWATA, S. 1997. A capacity scaling algorithm for convex cost submodular flows. *Math. Prog. 76*, 299–308.

IWATA, S. 2001. A fully combinatorial algorithm for submodular function minimization. *J. Combinat. Theory*, in press.

IWATA, S., MCCORMICK, S. T., AND SHIGENO, M. 1999. A strongly polynomial cut canceling algorithm for the submodular flow problem. In *Proceedings of the 7th MPS Conference on Integer Programming and Combinatorial Optimization*. Springer-Verlag, Berlin, Germany, pp. 259–272.

IWATA, S., AND MUROTA, K. 1995. A minimax theorem and a Dulmage-Mendelsohn type decomposition for a class of generic partitioned matrices. *SIAM J. Matrix Anal. Appl. 16*, 719–734.

LOVÁSZ, L. 1983. Submodular functions and convexity. In *Mathematical Programming—The State of the Art*, A. Bachem, M. Grötschel and B. Korte, Eds. Springer-Verlag, New York, pp. 235–257.

NAGAMOCHI, H., AND IBARAKI, T. 1992. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Disc. Math. 5*, 54–64.

NARAYANAN, H. 1995. A rounding technique for the polymatroid membership problem. *Linear Alg. Appl. 221*, 41–57.

QUEYRANNE, M. 1998. Minimizing symmetric submodular functions. *Math. Prog. 82*, 3–12.

SCHRIJVER, A. 2000. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Combinat. Theory B80*, 346–355.

SHAPLEY, L. S. 1971. Cores of convex games. *Int. J. Game Theory 1*, 11–26.

SOHONI, M. A. 1992. Membership in submodular and other polyhedra. Tech. Rep. TR-102-92. Dept. Comput. Sci. Eng., Indian Institute of Technology, Bombay, India.

TAMIR, A. 1993. A unifying location model on tree graphs based on submodularity properties. *Disc. Appl. Math. 47*, 275–283.

TARDOS, É. 1985. A strongly polynomial minimum cost circulation algorithm. *Combinatorica 5*, 247–255.

RECEIVED JULY 1999; REVISED JANUARY 2001; ACCEPTED JANUARY 2001