アルゴリズム 79−10
(2001. 7. 27)

劣モジュラ関数最小化の完全に組合せ的なアルゴリズム

岩田 覚
東京大学情報理工学系研究科
数理情報学専攻
iwata@sr3.t.u-tokyo.ac.jp

梗概: 加減算と大小比較のみを用いて, 劣モジュラ関数の最小値を計算する強多項式時間アルゴリズムを提示する.

A Fully Combinatorial Algorithm for Submodular Function Minimization

Satoru IWATA
Department of Mathematical Informatics
Graduate School of Information Science and Technology
University of Tokyo
iwata@sr3.t.u-tokyo.ac.jp

Abstract: This paper presents a strongly polynomial algorithm for general submodular function minimization using only additions, subtractions, comparisons, and oracle calls for function values.

1 Introduction

Let U be a finite nonempty set of cardinality n. A function f defined on the subsets of U is *submodular* if it satisifies

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

Examples of submodular functions include cut capacity functions, matroid rank functions, and entropy functions.

Grötschel–Lovász–Schrijver [9, 10] showed that submodular functions can be minimized in strongly polynomial time by the ellipsoid method. Combinatorial strongly polynomial algorithms are developed independently by Iwata–Fleischer–Fujishige (IFF) [12] and Schrijver [16]. Both of these algorithms are based on the first combinatorial pseudopolynomial-time algorithm due to Cunningham [2]. The IFF algorithm employs a scaling scheme for submodular functions developed in the design of capacity scaling algorithms for submodular flows [4, 11], while Schrijver's algorithm builds more directly on Cunningham's algorithm.

These combinatorial algorithms perform multiplications and divisions, despite the problem of submodular function minimization does not involve multiplications nor divisions. Schrijver [16] asks if one can minimize submodular functions in strongly polynomial time using only additions, subtractions, comparisons, and the oracle calls for function values. Such an algorithm is called 'fully combinatorial.' The present paper settles this problem by developing a fully combinatorial variant of the IFF algorithm.

A fully combinatorial algorithm consists of oracle calls for function evaluation and fundamental operations including additions, subtractions, and comparisons. Such an algorithm is strongly polynomial if the total number of oracle calls and fundamental operations is bounded by a polynomial in the dimension n of the problem. In the design of a fully combinatorial, strongly polynomial algorithm, we are allowed to multiply an integer which is bounded by a polynomial in n. We are also allowed to compute an integer rounding of a ratio of two numbers, provided that the answer is bounded by a polynomial in n.

Fully combinatorial, strongly polynomial algorithms are known for various combinatorial optimization problems, e.g., the minimum spanning tree, shortest path, maximum flow, and assignment problems. For the minimum cost flow problem, Fujishige [6] suggested a fully combinatorial implementation of the first strongly polynomial algorithm due to Tardos [18], which used Gaussian elimination. For the problem of testing membership in matroid polyhedra, which is a special case of submodular function minimization, Cunningham [1] devised a strongly polynomial algorithm and its fully combinatorial implementation. For finding a proper nonempty subset minimizing a symmetric submodular function, Queyranne [15] presented a fully combinatorial, strongly polynomial algorithm, extending a minimum cut algorithm of Nagamochi–Ibaraki [14] for undirected graphs.

An advantage of fully combinatorial algorithms is that they are easily extended to solve the problem over any totally ordered additive group. In fact, our algorithm as well as its analysis can be applied to submodular functions over an arbitrary totally ordered additive group.

This paper is organized as follows. Section 2 provides preliminaries on base polyhedra. In Section 3, we decribe an outline of our algorithm. The algorithm repeatedly applies a procedure Fix, which will be described in Section 4. Finally, in Section 5, we discuss the time complexity to show that the algorithm is strongly polynomial.

2 Base Polyhedra

This section provides preliminaries on submodular functions and base polyhedra. See [5, 7, 13] for more details and general background.

Let V be a finite nonempty set of cardinality ν. For a vector $x \in \mathbf{R}^V$ and a subset $X \subseteq V$, we denote $x(X) = \sum_{u \in X} x(u)$. We also denote by x^- a vector in $\mathbf{R}^V$ defined by $x^-(u) = \min\{x(u), 0\}$. For each $u \in V$, we denote by χ_u the vector in $\mathbf{R}^V$ with $\chi_u(u) = 1$ and $\chi_u(v) = 0$ for $v \in V \backslash \{u\}$.

Consider a directed acyclic graph $D = (V, F)$ on the vertex set V. A subset $Y \subseteq V$ is called an *ideal* if no arc leaves Y in D. The set $\mathcal{D}$ of all the ideals of D forms a distributive lattice.

For a submodular function $g : \mathcal{D} \to \mathbf{R}$ with $g(\emptyset) = 0$, we consider the *base polyhedron*

$$\mathrm{B}(g) = \{y \mid y \in \mathbf{R}^V,\, y(V) = g(V),\, \forall X \in \mathcal{D} : y(X) \leq g(X)\}.$$

A vector in $\mathrm{B}(g)$ is called a *base*. In particular, an extreme point of $\mathrm{B}(g)$ is called an *extreme base*. An extreme base can be computed by the following slight generalization of the greedy algorithm of Edmonds [3] and Shapley [17].

Let $L = (v_1, \cdots, v_\nu)$ be a linear ordering of V. For any $v_j \in V$, we denote $L(v_j) = \{v_1, \cdots, v_j\}$. The linear ordering L is called a *linear extension* of D if $L(v_j) \in \mathcal{D}$ for every v_j. The greedy algorithm with respect to a linear extension L generates an extreme base $y \in \mathrm{B}(g)$ by

$$y(u) := g(L(u)) - g(L(u) \backslash \{u\}).$$

Conversely any extreme base can be obtained by this way with an appropriate linear extension [8].

Suppose u immediately succeeds v in a linear extension L that generates an extreme base $y \in \mathrm{B}(g)$. If $(u, v) \notin F$, then the linear ordering L' obtained from L by interchanging u and v is also a linear extension. The extreme base $y' \in \mathrm{B}(g)$ generated by L' can differ from y only at u and v. More precisely, it satisfies $y' = y + \beta(\chi_u - \chi_v)$ with

$$\beta = g(L(u)\backslash\{v\}) - y(L(u)\backslash\{v\}) \geq 0.$$

This quantity β is called an *exchange capacity.*

3 A Fully Combinatorial Algorithm

This section presents an outline of our fully combinatorial algorithm for minimizing a submodular function $f : 2^U \to \mathbf{R}$. The algorithm consists of iterations. Each iteration calls a procedure Fix described in Section 4.

The algorithm works with a directed acyclic graph $D = (V, F)$ and a subset $Z \subseteq U$ that is included in every minimizer of f. The vertex set V of D corresponds to a partition of $U\backslash Z$. For a subset $Y \subseteq V$, we denote by $\Gamma(Y)$ the union of the subsets of U represented by the vertices in Y. Each arc $(u, v) \in F$ reflects an implication that a minimizer of f including $\Gamma(\{u\})$ must include $\Gamma(\{v\})$ as well. A subset $Y \subseteq V$ is called an ideal of D if no arc leaves Y in D. Thus any minimizer W of f is in the form of $W = \Gamma(Y) \cup Z$ for some ideal Y. Initially, $Z := \emptyset$, $V := U$ and $F := \emptyset$, which apparently satisfy the above properties.

Recall that $\mathcal{D}$ denotes the set of all the ideals of D. We now consider a function $\widehat{f} : \mathcal{D} \to \mathbf{R}$ defined by $\widehat{f}(Y) = f(\Gamma(Y) \cup Z) - \min\{f(Z), f(U)\}$ for $Y \in \mathcal{D}\backslash\{\emptyset, V\}$ and $\widehat{f}(\emptyset) = \widehat{f}(V) = 0$. It is easy to verify that $\widehat{f}$ is submodular on $\mathcal{D}$.

Lemma 3.1 *Any minimizer $W \subseteq U$ of f is represented as $W = \Gamma(Y) \cup Z$ by a minimizer $Y \in \mathcal{D}$ of $\widehat{f}$.*

Proof. Recall that any minimizer W of f is represented as $W = \Gamma(Y) \cup Z$ by some $Y \in \mathcal{D}$. Then $\widehat{f}(Y) = f(W) - \min\{f(Z), f(U)\} \leq 0$ and $\widehat{f}(Y) \leq \widehat{f}(X)$ for any $X \in \mathcal{D}\backslash\{V, \emptyset\}$. Thus Y is a minimizer of $\widehat{f}$. ■

For each vertex $v \in V$, let $R(v)$ denote the set of vertices reachable from v in D. At the beginning of each iteration, the algorithm computes

$$\alpha = \max\{\widehat{f}(R(v)) - \widehat{f}(R(v)\backslash\{v\}) \mid v \in V\} \tag{1}$$

and $\sigma = |V|\alpha$. If $\alpha \leq 0$, then the algorithm finds a minimizer of f by the following lemma.

Lemma 3.2 *If $\alpha \leq 0$, then U or Z is a minimizer of f.*

Proof. Let X be the unique maximal minimizer of $\widehat{f}$. If $X \neq V$, there is a vertex $v \in V\backslash X$ with $X \cup \{v\} \in \mathcal{D}$. By the submodularity of $\widehat{f}$, we have $\widehat{f}(X \cup \{v\}) - \widehat{f}(X) \leq \widehat{f}(R(v)) - \widehat{f}(R(v)\backslash\{v\}) \leq 0$, which contradicts the definition of X. Thus V is a minimizer of $\widehat{f}$, and hence $\widehat{f}(Y) \geq 0$ for $Y \in \mathcal{D}$.

By Lemma 3.1, any minimizer W of f is represented as $W = \Gamma(Y) \cup Z$ by a minimizer $Y \in \mathcal{D}$ of $\widehat{f}$. It follows from $\widehat{f}(Y) \geq 0$ that $f(W) = f(\Gamma(Y) \cup Z) \geq \min\{f(U), f(Z)\}$. Thus U or Z is a minimizer of f. ■

If $\alpha > 0$, let u be the vertex that attains the maximum in (1). Since $\alpha = \widehat{f}(R(u)) - \widehat{f}(R(u)\backslash\{u\})$, either $2\widehat{f}(R(u)) \geq \alpha$ or $2\widehat{f}(R(u)\backslash\{u\}) < -\alpha$.

If $2\widehat{f}(R(u)\backslash\{u\}) < -\alpha < 0$, the algorithm applies a procedure $\mathsf{Fix}(\widehat{f}, \alpha, \sigma)$ to find a vertex $w \in V$ that is contained in every minimizer of $\widehat{f}$. Since $\Gamma(\{w\})$ must be included in every minimizer of f by Lemma 3.1, the algorithm adds $\Gamma(\{w\})$ to Z and deletes w from D. The resulting Z and D continues to satisfy the required properties.

Lemma 3.3 *Any exchange capacity in* $\mathrm{B}(\widehat{f})$ *is at most* σ.

Proof. Let y be an arbitrary extreme base of $\mathrm{B}(\widehat{f})$ and L a linear extension that generates y. For each $v \in V$, we have $y(v) = \widehat{f}(L(v)) - \widehat{f}(L(v)\backslash\{v\}) \leq \widehat{f}(R(v)) - \widehat{f}(R(v)\backslash\{v\}) \leq \alpha$ by the submodularity of $\widehat{f}$. Since $y(V) = \widehat{f}(V) = 0$, we also have $-(|V|-1)\alpha \leq y(v) \leq \alpha$ for each $v \in V$. Thus any exchange capacity in $\mathrm{B}(\widehat{f})$ is at most $\sigma = |V|\alpha$. ■

If $2\widehat{f}(R(u)) \geq \alpha > 0$, the algorithm applies a procedure $\mathsf{Fix}(\widehat{f}_u, \alpha, \sigma)$ to find a vertex $w \in V\backslash R(u)$ that is contained in every minimizer of $\widehat{f}_u$ defined by

$$\widehat{f}_u(X) = \widehat{f}(X \cup R(u)) - \widehat{f}(R(u)) \qquad (X \subseteq V\backslash R(u)).$$

Note that $\widehat{f}_u$ is submodular and $2\widehat{f}_u(V\backslash R(u)) \leq -\alpha$. A subset $X \subseteq V\backslash R(u)$ is a minimizer of $\widehat{f}_u$ if and only if $X \cup R(u)$ minimizes $\widehat{f}$ among those ideals that contains u. Therefore, any minimizer of $\widehat{f}$ containing u must contain w. This implies by Lemma 3.1 that any minimizer of f including $\Gamma(\{u\})$ must include $\Gamma(\{w\})$. Then the algorithm adds a new arc (u, w) to F. If this yields a directed cycle Q, any minimizer of f must include all or none of the elements represented by the vertices in Q, and hence the algorithm contracts Q to a single vertex. The resulting Z and D continues to satisfy the required properties.

Lemma 3.4 *Any exchange capacity in* $\mathrm{B}(\widehat{f}_u)$ *is at most* σ.

Proof. Let y be an extreme base of $\mathrm{B}(\widehat{f}_u)$ generated by a linear extension L. For each $v \in V$, we have $y(v) = \widehat{f}_u(L(v)) - \widehat{f}_u(L(v)\backslash\{v\}) = \widehat{f}(R(u) \cup L(v)) - \widehat{f}(R(u) \cup L(v)\backslash\{v\}) \leq \widehat{f}(R(v)) - \widehat{f}(R(v)\backslash\{v\}) \leq \alpha$ by the submodularity of $\widehat{f}$. Since $y(V\backslash R(u)) = -\widehat{f}(R(u))$ and $\widehat{f}(R(u)) \leq \sum_{w \in R(u)}\{\widehat{f}(R(w)) - \widehat{f}(R(w)\backslash\{w\})\} \leq |R(u)|\alpha$, we obtain $y(V\backslash R(u)) \geq -|R(u)|\alpha$. Therefore, we have $-(|V|-1)\alpha \leq y(v) \leq \alpha$ for each $v \in V$. Thus any exchange capacity in $\mathrm{B}(\widehat{f}_u)$ is at most $\sigma = |V|\alpha$. ■

As a result of each iteration with $\alpha > 0$, the algorithm deletes a vertex from D or adds a new arc to D. Therefore, after at most n^2 iterations, the algorithm terminates with $\alpha \leq 0$, which provides a minimizer of f by Lemma 3.2.

4 The Fixing Procedure

This section describes the procedure $\mathsf{Fix}(g, \alpha, \sigma)$ for finding a vertex $w \in V$ that is contained in every minimizer of a submodular function $g : \mathcal{D} \to \mathbf{R}$ such that any exchage capacity in $\mathrm{B}(g)$ is at most σ. The distributive lattice $\mathcal{D}$ is the set of ideals of a directed graph $D = (V, F)$ on the vertex set V of cardinality ν. We also assume that there is a subset $Y \in \mathcal{D}$ such that $2g(Y) \leq -\alpha$. Whenever the algorithm calls Fix, these conditions are satisfied with $\sigma \leq n\alpha$.

The procedure consists of scaling phases with a scale parameter $p \in \mathbf{Z}$, which is initially set as $p := 1$. It keeps a set of linear orderings $\{L_i \mid i \in I\}$ of the vertices in V. Each linear ordering L_i generates an extreme base $y_i \in \mathrm{B}(g)$ by the greedy algorithm. The procedure also keeps a set of nonnegative integral coefficients $\{\mu_i \mid i \in I\}$ such that $\sum_{i \in I} \mu_i = p$. Initially, $I = \{0\}$ with an arbitrary linear ordering L_0 and $\mu_0 = 1$.

Furthermore, the procedure works with a flow in the complete directed graph with the vertex set V and the arc set $E = \{(u, v) \mid u, v \in V\}$. The flow is represented as a skew-symmetric function $\varphi : V \times V \to \mathbf{R}$. That is, $\varphi(u,v)+\varphi(v,u)=0$ holds for any pair of vertices $u,v \in V$. The capacity of each arc in $E\backslash F$ is equal to σ. Namely, $\varphi(u,v) \le \sigma$ holds for any arc $(u,v) \in E\backslash F$. The boundary $\partial\varphi$ is defined by $\partial\varphi(u) = \sum_{v\in V}\varphi(u,v)$ for $u \in V$. Initially, $\varphi(u,v)=0$ for any $u,v \in V$.

Each scaling phase aims at increasing $z^-(V)$ for $z = \partial\varphi + x$ with $x = \sum_{i\in I}\mu_i y_i$. Given a flow φ, the procedure constructs an auxiliary directed graph $G(\varphi) = (V, A(\varphi))$ with arc set $A(\varphi) = F \cup \{(u,v) \mid u \neq v,\ \varphi(u,v) \le 0\}$. Let $S = \{v \mid z(v) \le -\sigma\}$ and $T = \{v \mid z(v) \ge \sigma\}$. A directed path in $G(\varphi)$ from S to T is called an *augmenting path.*

Let W be the set of vertices reachable from S in $G(\varphi)$. Then W must be an ideal of D. 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. We now describe an operation Double-Exchange that is applicable to an active triple (i,u,v). This operation modifies x and φ without changing z. See Figure 1 for the formal description.

```
Double-Exchange$(i,u,v)$:

$\beta := g(L_i(u)\backslash\{v\}) - y_i(L_i(u)\backslash\{v\})$;
**If** $\varphi(u,v) < \mu_i\beta$ **then**
        $q := \lceil \varphi(u,v)/\beta \rceil$
        $k \leftarrow$ a new index;
        $I := I \cup \{k\}$;
        $\mu_k := \mu_i - q$;
        $\mu_i := q$;
        $y_k := y_i$;
        $L_k := L_i$;
Update $L_i$ by interchanging $u$ and $v$;
$y_i := y_i + \beta(\chi_u - \chi_v)$;
$\varphi(u,v) := \varphi(u,v) - \mu_i\beta$;
$\varphi(v,u) := \varphi(v,u) + \mu_i\beta$.
```

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

The first step of Double-Exchange(i,u,v) is to compute the exchange capacity

$$\beta = g(L_i(u)\backslash\{v\}) - y_i(L_i(u)\backslash\{v\}).$$

If $\varphi(u,v) \ge \mu_i\beta$, Double-Exchange$(i,u,v)$ is called *saturating.* Otherwise, it is called *nonsaturating.*

In the nonsaturating Double-Exchange(i,u,v), a new index k is added to I. The associated y_k and L_k are the previous y_i and L_i, respectively. Then it computes $q := \lceil \varphi(u,v)/\beta \rceil$ by repeatedly subtracting β from $\varphi(u,v)$. Since $\varphi(u,v) < \mu_i\beta$, the number of required subtractions is at most $\mu_i \le p$. More efficiently, we can carry out this computation of q by $O(\log^2 p)$ fundamental operations. The associated coefficient μ_k is given by $\mu_k := \mu_i - q$, and then μ_i is replaced by $\mu_i := q$. Note that the new μ_i satisfies $\varphi(u,v) \le \mu_i\beta \le \varphi(u,v) + \beta$.

Whether saturating or nonsaturating, Double-Exchange(i,u,v) interchanges u and v in L_i and replaces y_i by $y_i := y_i + \beta(\chi_u - \chi_v)$. The resulting y_i is an extreme base generated by the new linear ordering L_i. The final step of Double-Exchange is to adjust φ so that $z = \partial\varphi + \sum_{i\in I} \mu_i y_i$ is invariant. Namely, $\varphi(u,v) := \varphi(u,v) - \mu_i\beta$ and $\varphi(v,u) := \varphi(v,u) + \mu_i\beta$. The resulting φ satisfies $-\sigma \le -\beta \le \varphi(u,v) \le \sigma$. If Double-Exchange$(i,u,v)$ is nonsaturating, it satisfies $\varphi(u,v) \le 0$, which implies that v is now reachable from S in $G(\varphi)$.

We are now ready to describe the procedure Fix.

Procedure Fix(g,α,σ):

Step 0: Let L_0 be an arbitrary linear ordering. Compute an extreme base y_0 by the greedy algorithm with respect to L_0. Put $p := 1$, $\mu_0 := 1$, $I := \{0\}$, and $\varphi(u,v) := 0$ for $u, v \in V$.

Step 1: While there is an augmenting path or an active triple, repeat the following

- **(1-1)** If there is an augmenting path P, then augment the flow φ along P by putting $\varphi(u,v) := \varphi(u,v) + \sigma$ and $\varphi(v,u) := \varphi(v,u) - \sigma$ for each arc (u,v) in P.
- **(1-2)** Otherwise, apply Double-Exchange to an active triple (i,u,v).

Step 2: If $x = \sum_{i\in I} \mu_i y_i$ satisfies $x(w) < -\nu^2\sigma$ for some $w \in V$, then return w.

Step 3: Put $p := 2p$ and $\mu_i := 2\mu_i$ for each $i \in I$. Go to Step 1. ■

One execution of Step 1 is referred to as a scaling phase in the following analysis.

Lemma 4.1 *At the end of a scaling phase, $z^-(V) \ge pg(W) - \nu\sigma$ holds.*

Proof. Since there is no active triple, we have $y_i(W) = g(W)$ for each $i \in I$, and hence $x(W) = \sum_{i\in I} \mu_i y_i(W) = pg(W)$. Note that $z(v) < \sigma$ for $v \in W$ (since $T \cap W = \emptyset$) and $z(v) > -\sigma$ for $v \in V\backslash W$ (since $S \subseteq W$). Therefore, we have $z^-(V) = z^-(W) + z^-(V\backslash W) \ge z(W) - \sigma|W| - \sigma|V\backslash W| = x(W) + \partial\varphi(W) - \nu\sigma \ge pg(W) - \nu\sigma$. ■

Theorem 4.2 *If $x(w) < -\nu^2\sigma$ at the end of a scaling phase, w is contained in every minimizer of g.*

Proof. By Lemma 4.1, the set W satisfies $z^-(V) \ge pg(W) - \nu\sigma$. Since $\partial\varphi(v) \le (\nu-1)\sigma$ for each $v \in V$, we have $x^-(V) \ge z^-(V) - \nu(\nu-1)\sigma \ge pg(W) - \nu^2\sigma$. For any minimizer X of g, we have $pg(W) \ge pg(X) \ge x(X) \ge x^-(X)$. Thus we obtain $x^-(V) \ge x^-(X) - \nu^2\sigma$, which implies $w \in X$ if $x(w) < -\nu^2\sigma$. ■

5 Complexity

This section is devoted to complexity analysis of our fully combinatorial algorithm.

Lemma 5.1 *The procedure* Fix *consists of* $\mathrm{O}(\log n)$ *scaling phases.*

Proof. Recall $\sigma \le n\alpha$ and there is an ideal $Y \in \mathcal{D}$ with $2g(Y) \le -\alpha$. After $2 + \lfloor \log_2 \nu^3 n \rfloor$ scaling phases, the scale parameter p satisfies $p\alpha > 2\nu^3\sigma$. Therefore, we have $x(Y) = \sum_{i\in I} \mu_i y_i(Y) \le pg(Y) < -\nu^3\sigma$, which implies there exists a vertex $w \in Y$ such that $x(w) < -\nu^2\sigma$. Thus the procedure terminates after $\mathrm{O}(\log \nu^3 n)$ scaling phases. Since $\nu \le n$, we may conclude that Fix performs $\mathrm{O}(\log n)$ scaling phases. ■

Lemma 5.2 *The procedure* Fix *performs* $O(\nu^2 \log n)$ *augmentations.*

Proof. At the beginning of each scaling phase, the set W obtained by the previous scaling phase satisfies $z^-(V) \geq pg(W) - \nu\sigma$ by Lemma 4.1. For the first scaling phase, we have the same inequality by taking $W = V$. Note that $z^-(V) \leq z(X) \leq pg(X) + \nu(\nu - 1)\sigma$ for any X throughout the procedure. Thus each scaling phase increases $z^-(V)$ by at most $\nu^2\sigma$. Since each augmentation increases $z^-(V)$ by σ, each scaling phase performs at most ν^2 augmentations. Then it follows from Lemma 5.1 that the total number of augmentations in Fix is $O(\nu^2 \log n)$. ■

Lemma 5.3 *The procedure* Fix *performs nonsaturating* Double-Exchange $O(\nu^3 \log n)$ *times.*

Proof. If Double-Exchange(i, u, v) is nonsaturating, the vertex v becomes reachable from S in $G(\varphi)$, which means the set W is enlarged. Thus there are at most ν applications of nonsaturating Double-Exchange between augmentations. Since Fix performs $O(\nu^2 \log n)$ augmentations by Lemmas 5.2, the number of nonsaturating applications of Double-Exchange is $O(\nu^3 \log n)$. ■

Lemma 5.4 *The procedure* Fix *maintains* $O(\nu^3 \log n)$ *extreme bases.*

Proof. A new index k is added to I only as a result of nonsaturating Double-Exchange. Hence, it follows from Lemma 5.3 that $|I|$ is $O(\nu^3 \log n)$. ■

Lemma 5.5 *The procedure* Fix *performs* Double-Exchange $O(\nu^7 \log^2 n)$ *times.*

Proof. Once the procedure 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 augmentations or the end of the phase. Hence, for each $i \in I$, the procedure applies Double-Exchange $O(\nu^2)$ times between augmentations. Then it follows from Lemmas 5.1 and 5.4 that the procedure performs Double-Exchange $O(\nu^7 \log^2 n)$ times. ■

Theorem 5.6 *The fully combinatorial algorithm finds a minimizer of f by $O(n^9 \log^2 n)$ oracle calls and fundamental operations.*

Proof. The fully combinatorial algorithm calls the procedure Fix $O(n^2)$ times. Each application of Double-Exchange requires a function evaluation of g. Thus, by Lemma 5.5, the algorithm performs $O(n^9 \log^2 n)$ oracle calls and fundamental operations. ■

Acknowledgements

The author is very grateful to Yves Crama, Satoru Fujishige, Makoto Matsumoto, and Tom McCormick for their helpful comments.

References

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

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

[3] 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.

[4] L. Fleischer, S. Iwata, and S. T. McCormick: A faster capacity scaling algorithm for submodular flow, *Math. Programming*, to appear.

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

[6] S. Fujishige: A capacity-rounding algorithm for the minimum-cost circulation problem: A dual framework of the Tardos algorithm, *Math. Programming*, 35 (1986), 298–308.

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

[8] S. Fujishige and N. Tomizawa: A note on submodular functions on distributive lattices, *J. Oper. Res. Soc. Japan*, 26 (1983), 309–318.

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

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

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

[12] S. Iwata, L. Fleischer, and S. Fujishige: A combinatorial strongly polynomial algorithm for minimizing submodular functions, *J. ACM*, to appear.

[13] 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.

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

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

[16] A. Schrijver: A combinatorial algorithm minimizing submodular functions in strongly polynomial time, *J. Combin. Theory*, B80 (2000), 346–355.

[17] L. S. Shapley: Cores of convex games, *Int. J. Game Theory*, 1 (1971), 11–26.

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