アルゴリズム 70－2
（1999．11．8）

A Strongly Polynomial-Time Algorithm for Minimizing Submodular Functions

Satoru IWATA * Lisa FLEISCHER † Satoru FUJISHIGE ‡

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.

劣モジュラ関数最小化の強多項式時間アルゴリズム

岩田 覚　Lisa Fleischer　藤重 悟

梗概: 劣モジュラ関数の最小化は，離散最適化における基本的な問題である．Grötschel–Lovász–Schrijver (1981) は，楕円体法を用いて，この問題が多項式時間で解けることを示した．しかし，楕円体法が実際には極めて非効率的であるため，組合せ的な多項式時間アルゴリズムの開発が長い間望まれてきた．

本論文では，劣モジュラ関数を最小化する最初の組合せ的な多項式時間アルゴリズムを提示する．このアルゴリズムは，各枝の容量が一様な完全有向グラフを用いた新たなスケーリング技法を利用しており，計算量が台集合の大きさと関数値の最大ビット長の多項式で押えられる．さらに，計算量が台集合の大きさの多項式で押えられる強多項式時間アルゴリズムも提示する．

*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 [7] 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. Later, they also devised a strongly polynomial-time algorithm within their framework using the ellipsoid method [8]. 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.

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. To develop a capacity-scaling, augmenting-path algorithm for submodular function minimization, our work in this paper builds on ideas by Iwata [9] and Fleischer, Iwata, and McCormick [5]. 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 is the technique first introduced by Iwata [9], who used it to develop the first polynomial-time capacity-scaling algorithm for the submodular flow problem. This algorithm was later refined by Fleischer, Iwata, and McCormick [5] into one of the fastest algorithms for submodular flow.

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 $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$ $(X, Y \subseteq V)$. We suppose that $f(\emptyset) = 0$ without loss of generality throughout this paper. For basic notation and facts about submodular functions, see Lovász [10] and Fujishige [6], for example.

We define the *submodular polyhedron* $\mathrm{P}(f)$ and the *base polyhedron* $\mathrm{B}(f)$ associated with the submodular function f by $\mathrm{P}(f) = \{x \mid x \in \mathbf{R}^V, \forall X \subseteq V : \ x(X) \leq f(X)\}$ and $\mathrm{B}(f) = \{x \mid x \in \mathrm{P}(f), x(V) = f(V)\}$, respectively. 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* as $\tilde{c}(x, u, v) = \max\{\alpha \mid \alpha \in \mathbf{R}, x + \alpha(\chi_u - \chi_v) \in \mathrm{B}(f)\}$.

We call an extreme point of $\mathrm{B}(f)$ an *extreme base*. It is known that for any extreme base $y \in \mathrm{B}(f)$ the set of the *tight* set $\mathcal{D}(y) = \{X \mid X \subseteq V, \ y(X) = f(X)\}$ is a distributive lattice,

which is equal to the set of (lower) ideals of a unique poset $\mathcal{P}(y) = (V, \preceq_y)$ on V, and that we can construct the Hasse diagram $H(y) = (V, A(y))$ of the poset $\mathcal{P}(y)$ in $O(n^2)$ time by using the evaluation oracle [1] (cf. [6, pp. 62–63]).

Computing exchange capacities in general is as hard as minimizing submodular functions. However, Lemma 2.1 given below shows that if $y \in B(f)$ 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 [6].

Lemma 2.1: *For an extreme base $y \in B(f)$ let $(u, v) \in A(y)$. Then we have $\tilde{c}(y, u, v) = f(J_y(u)\backslash\{v\}) - y(J_y(u)\backslash\{v\})$.* ■

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 [6, Corollaries 3.4 and 3.5]).

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

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

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 2.2 shows a min-max relation of strong duality. A weak duality is described as follows: for any base $x \in B(f)$ 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 $x \in B(f)$ 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 $x = \sum_{i \in I} \lambda_i y_i$.

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 [9]. 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 $\partial\varphi(v) = \sum_{u \in V} \varphi(u,v)$ $(v \in V)$.

We start with $x \in \mathrm{B}(f)$ 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 $E(\varphi) = \{(u,v) \mid u, v \in V,\ u \neq v,\ \varphi(u,v) \leq 0\}$.

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 [5]. 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.

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 $\mathsf{Push}(i,u,v)$, depicted in Figure 1, to each arc $(u,v) \in A(y_i)$ with $v \in V \backslash W$. The operation $\mathsf{Push}(i,u,v)$ 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 $y_k := y_i + \tilde{c}(y_i,u,v)(\chi_u - \chi_v)$ 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.

Push(i, u, v):

$\alpha \leftarrow \min\{\delta,\ \lambda_i \tilde{c}(y_i, u, v)\}$
$\varphi(u, v) \leftarrow \varphi(u, v) - \alpha$
$\varphi(v, u) \leftarrow \varphi(v, u) + \alpha$
If $\alpha < \lambda_i \tilde{c}(y_i, u, v)$ **then**
    $k \leftarrow$ a new index
    $I \leftarrow I \cup \{k\}$
    $\lambda_k \leftarrow \alpha / \tilde{c}(y_i, u, v)$
    $\lambda_i \leftarrow \lambda_i - \lambda_k$
    $y_k \leftarrow y_i + \tilde{c}(y_i, u, v)(\chi_u - \chi_v)$
else $y_i \leftarrow y_i + \tilde{c}(y_i, u, v)(\chi_u - \chi_v)$
$x \leftarrow \sum_{i \in I} \lambda_i y_i$

Figure 1: Algorithmic description of the operation Push(i, u, v).

Each time the algorithm applies the push operation, it updates the set W of vertices reachable from S in $G(\varphi)$. If 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 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 2.1, $J_{y_i}(u) \backslash \{v\}$ is tight for y_i'. Thus, $J_{y_i'}(u) \subseteq J_{y_i}(u) \setminus \{v\}$. For any $w \in W$ with $J_{y_i}(w) \subseteq W$, we have $J_{y_i'}(w) = J_{y_i}(w) \subseteq W$. These two facts are fundamental in the proof of the following lemmas.

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

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).* ∎

Lemma 3.3: *Once (i,u) is scanned, it does not become admissible again 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 $X \subseteq V$.* ∎

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 $z^-(V) \geq f(W) - n\delta$.* ∎

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-scaling 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 2.2 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 $\delta < 1/n^2$.* ∎

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

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 the duality gap $f(X) - x^-(V)$ is less than one, X is a minimizer of f.

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.* ■

Let $f : 2^V \to \mathbf{R}$ be a submodular function and $x \in \mathrm{B}(f)$ 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 $x \in \mathrm{B}(f)$. 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 $\mathsf{Fix}(f, x, \eta)$.

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 $f_v(X) = f(X \cup R(v)) - f(R(v))$ $(X \subseteq V \backslash R(v))$. 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 $x(v_1) = f_u(v_1)$ and $x(v_j) = f_u(\{v_1, v_2, \ldots, v_j\}) - f_u(\{v_1, v_2, \ldots, v_{j-1}\})$ 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 $O(\log n)$ 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 $x \in B(f)$ 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 $O(\log n)$ scaling phases. Note that every minimizer of f includes $R(w)$. Thus it suffices to minimize the submodular function f_w, which is now defined on a smaller underlying set.

Theorem 4.2: *The proposed algorithm computes a 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. ∎

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

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

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

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

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

[10] 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.