アルゴリズム 83－6
（2002. 3. 15）

劣モジュラ関数最小化の高速スケーリング算法

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: 劣モジュラ関数の最小値を計算する組合せ的強多項式アルゴリズムが，最近，Iwata, Fleischer, Fujishige (IFF) と Schrijver によって与えられた．IFF アルゴリズムがスケーリング法を利用しているのに対して，Schrijver のアルゴリズムは距離ラベルを利用している．本研究では，両者の手法を融合して，より効率的な組合せ的アルゴリズムを開発する．

A Faster Scaling Algorithm for Minimizing Submodular Functions

Satoru IWATA
Department of Mathematical Informatics
University of Tokyo
iwata@sr3.t.u-tokyo.ac.jp

Abstract

Combinatorial strongly polynomial algorithms for minimizing submodular functions have been developed by Iwata, Fleischer, and Fujishige (IFF) and by Schrijver. The IFF algorithm employs a scaling scheme for submodular functions, whereas Schrijver's algorithm exploits distance labeling. This paper combines these two techniques to yield a faster combinatorial algorithm for submodular function minimization. The resulting algorithm improves over the previously best known bound by an almost linear factor in the size of the underlying ground set.

1 Introduction

Let V be a finite nonempty set of cardinality n. A set function f on V is *submodular* if it satisfies

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

Submodular functions are discrete analogues of convex functions [12]. Examples include cut capacity functions, matroid rank functions, and entropy functions.

The first polynomial-time algorithm for submodular function minimization is due to Grötschel–Lovász–Schrijver [7]. A strongly polynomial algorithm has also been described by Grötschel–Lovász–Schrijver [8]. These algorithms rely on the ellipsoid method, which is not efficient in practice.

Recently, combinatorial strongly polynomial algorithms have been developed by Iwata–Fleischer–Fujishige (IFF) [11] and by Schrijver [13]. Both of these algorithms build on the first combinatorial pseudopolynomial algorithm due to Cunningham [2]. The IFF algorithm employs a scaling scheme developed in capacity scaling algorithms for the submodular flow problem [5, 9]. In contrast, Schrijver [13] achieves a strongly polynomial bound by distance labeling argument similar to [1]. In this paper, we combine these two techniques to yield a faster combinatorial algorithm.

Let γ denote the time required for computing the function value of f and M the maximum absolute value of f. The IFF scaling algorithm minimizes an integral submodular function in $O(n^5\gamma \log M)$ time. The strongly polynomial version runs in $O(n^7\gamma \log n)$ time, whereas an improved variant of Schrijver's algorithm runs in $O(n^7\gamma + n^8)$ time [4].

The time complexity of our new scaling algorithm is $O((n^4\gamma + n^5)\log M)$. Since the function evaluation oracle has to identify an arbitrary subset of V as its argument, it is quite natural to assume γ is at least linear in n. Thus the new algorithm is faster than the IFF algorithm by a factor of n. The strongly polynomial version of the new scaling algorithm runs in $O((n^6\gamma + n^7)\log n)$ time. This is an improvement over the previous best bound by an almost linear factor in n.

These combinatorial algorithms perform multiplications and divisions, although the problem of submodular function minimization does not involve those operations. Schrijver [13] 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.' A very recent paper [10] settles this problem by developing a fully combinatorial variant of the IFF algorithm. Similarly, we can implement the strongly polynomial version of our scaling algorithm in a fully combinatorial manner. The resulting algorithm runs in $O(n^8\gamma \log^2 n)$ time, improving the previous bound by a factor of n.

This paper is organized as follows. Section 2 provides preliminaries on submodular functions. In Section 3, we describe the new scaling algorithm. Section 4 is devoted to its complexity analysis. Finally, in Section 5, we discuss its extensions as well as a fully combinatorial implementation.

2 Preliminary

This section provides preliminaries on submodular functions. See [6, 12] for more details and general background.

For a vector $x \in \mathbf{R}^V$ and a subset $Y \subseteq V$, we denote $x(Y) = \sum_{u \in Y} x(u)$. We also denote by x^- the vector in $\mathbf{R}^V$ with $x^-(u) = \min\{x(u), 0\}$. For each $u \in V$, let χ_u denote the vector in $\mathbf{R}^V$ with $\chi_u(u) = 1$ and $\chi_u(v) = 0$ for $v \in V \backslash \{u\}$.

For a submodular function $f : 2^V \to \mathbf{R}$ with $f(\emptyset) = 0$, we consider the *base polyhedron*

$$\mathrm{B}(f) = \{x \mid x \in \mathbf{R}^V,\ x(V) = f(V),\ \forall Y \subseteq V : x(Y) \leq f(Y)\}.$$

A vector in $\mathrm{B}(f)$ is called a *base*. In particular, an extreme point of $\mathrm{B}(f)$ is called an *extreme base*. An extreme base can be computed by the greedy algorithm of Edmonds [3] and Shapley [14] as follows.

Let $L = (v_1, \cdots, v_n)$ be a linear ordering of V. For any $v_j \in V$, we denote $L(v_j) = \{v_1, \cdots, v_j\}$. The greedy algorithm with respect to L generates an extreme base $y \in \mathrm{B}(f)$ by

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

Conversely, any extreme base can be obtained in this way with an appropriate linear ordering.

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 theorem shows that these inequalities are in fact tight for appropriately chosen x and Y.

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

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

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

This theorem is immediate from the vector reduction theorem on polymatroids due to Edmonds [3]. It has motivated combinatorial algorithms for minimizing submodular functions.

3 A scaling algorithm

This section presents a new scaling algorithm for minimizing an integral submodular function $f : 2^V \to \mathbf{Z}$.

The algorithm consists of scaling phases with a scale parameter $\delta \geq 0$. It keeps a set of linear orderings $\{L_i \mid i \in I\}$ of the vertices in V. We denote $v \preceq_i u$ if v precedes u in L_i or $v = u$. Each linear ordering L_i generates an extreme base $y_i \in \mathrm{B}(f)$ by the greedy algorithm. The algorithm also keeps a base $x \in \mathrm{B}(f)$ as a convex combination $x = \sum_{i \in I} \lambda_i y_i$ of the extreme bases. Initially, $I = \{0\}$ with an arbitrary linear ordering L_0 and $\lambda_0 = 1$.

Furthermore, the algorithm works with a flow in the complete directed graph on the vertex set V. The flow is represented as a skew-symmetric function $\varphi : V \times V \to \mathbf{R}$. Each arc capacity is equal to δ. Namely, $\varphi(u,v) + \varphi(v,u) = 0$ and $-\delta \leq \varphi(u,v) \leq \delta$ hold for any pair of vertices $u, v \in V$. 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 $z = x + \partial\varphi$. Given a flow φ, the procedure constructs an auxiliary directed graph $G_\varphi = (V, A_\varphi)$ with arc set $A_\varphi = \{(u,v) \mid u \neq v,\ \varphi(u,v) \leq 0\}$. Let $S = \{v \mid z(v) \leq -\delta\}$ and $T = \{v \mid z(v) \geq \delta\}$. A directed path in G_φ from S to T is called an *augmenting path.*

If there is an augmenting path P, the algorithm augments the flow φ along P by $\varphi(u,v) := \varphi(u,v) + \delta$ and $\varphi(v,u) := \varphi(v,u) - \delta$ for each arc (u,v) in P. This procedure is referred to as $\mathsf{Augment}(\varphi, P)$.

Each scaling phase also keeps a valid labeling d. A labeling $d : V \to \mathbf{Z}$ is *valid* if $d(u) = 0$ for $u \in S$ and $v \preceq_i u$ implies $d(v) \leq d(u) + 1$. A valid labeling $d(v)$ serves as a lower bound on the number of arcs from S to v in the directed graph $G_I = (V, A_I)$ with the arc set $A_I = \{(u,v) \mid \exists i \in I,\ v \preceq_i u\}$.

Let W be the set of vertices reachable from S in G_φ, and Z be the set of vertices that attains the minimum labeling in $V \backslash W$. A pair (u,v) of $u \in W$ and $v \in Z$ is called *active* for $i \in I$ if v is the first element of Z in L_i and u is the last element in L_i with $v \preceq_i u$ and $d(v) = d(u) + 1$. A triple (i,u,v) is also called *active* if (u,v) is active for $i \in I$. The procedure $\mathsf{Multiple\text{-}Exchange}(i,u,v)$ is applicable to an active triple (i,u,v).

For an active triple (i,u,v), the set of elements between v and u in L_i is called an *active interval.* Any element w in the active interval must satisfy $d(v) \leq d(w) \leq d(u)$. The active interval is divided into Q and R by $Q = \{w \mid w \in W,\ v \prec_i w \preceq_i u\}$ and $R = \{w \mid w \in V \backslash W,\ v \preceq_i w \prec_i u\}$.

The procedure $\mathsf{Multiple\text{-}Exchange}(i,u,v)$ moves the vertices in R to the place immediately after u in L_i, without changing the ordering in Q and in R. Then it computes an extreme base y_i generated by the new L_i. This results in $y_i(q) \geq y_i'(q)$ for $q \in Q$ and $y_i(r) \leq y_i'(r)$ for $r \in R$, where y_i' denotes the previous y_i.

Consider a complete bipartite graph with the vertex sets Q and R. The algorithm finds a flow $\xi : Q \times R \to \mathbf{R}_+$ such that $\sum_{r \in R} \xi(q,r) = y_i(q) - y_i'(q)$ for each $q \in Q$ and $\sum_{q \in Q} \xi(q,r) = y_i'(r) - y_i(r)$ for each $r \in R$. Such a flow can be obtained easily by the so-called northwest corner rule. Then the procedure computes $\eta = \max\{\xi(q,r) \mid q \in Q,\ r \in R\}$. If $\lambda_i \eta \leq \delta$, $\mathsf{Multiple\text{-}Exchange}(i,u,v)$ is called *saturating.* Otherwise, it is called *nonsaturating.*

In the nonsaturating Multiple-Exchange(i, u, v), a new index k is added to I. The associated linear ordering L_k is the previous L_i. The coefficient λ_k is determined by $\lambda_k := \lambda_i - \delta/\eta$, and then λ_i is replaced by $\lambda_i := \delta/\eta$. Whether saturating or nonsaturating, the procedure adjusts the flow φ by $\varphi(q, r) := \varphi(q, r) - \lambda_i \xi(q, r)$ and $\varphi(r, q) := \varphi(r, q) + \lambda_i \xi(q, r)$ for every $(q, r) \in Q \times R$.

Let h denote the number of vertices in the active interval. The number of function evaluations required for computing the new extreme base y_i by the greedy algorithm is at most h. The northwest corner rule can be implemented to run in $\mathrm{O}(h)$ time. Thus the total time complexity of Multiple-Exchange(i, u, v) is $\mathrm{O}(h\gamma)$.

If there is no active triple, the algorithm applies Relabel to each $v \in Z$. The procedure Relabel(v) increments $d(v)$ by one. Then the labeling d remains valid.

The number of extreme bases in the expression of x increases as a consequence of nonsaturating Multiple-Exchange. In order to reduce the complexity, the algorithm occasionally applies a procedure Reduce(x, I) that computes an expression of x as a convex combination of affinely independent extreme bases chosen from the currently used ones. This computation takes $\mathrm{O}(n^2|I|)$ time with the aid of Gaussian elimination.

We are now ready to describe the new scaling algorithm.

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 $x := y_0$, $\lambda_0 := 1$, $I := \{0\}$, and $\delta := |x^-(V)|/n^2$.

Step 1: Put $d(v) := 0$ for $v \in V$, and $\varphi(u, v) := 0$ for $u, v \in V$.

Step 2: Put $S := \{v \mid z(v) < -\delta\}$ and $T := \{v \mid z(v) > \delta\}$. Let W be the set of vertices reachable from S in G_φ.

Step 3: If there is an augmenting path P, then do the following.

- **(3-1)** Apply Augment(φ, P).
- **(3-2)** Apply Reduce(x, I).
- **(3-3)** Go to Step 2.

Step 4: Compute $\ell := \min\{d(v) \mid v \in V \backslash W\}$ and put $Z := \{v \mid v \in V \backslash W, d(v) = \ell\}$. If $\ell < n$, then do the following.

- **(4-1)** If there is an active triple (i, u, v), then apply Multiple-Exchange(i, u, v).
- **(4-2)** Otherwise, apply Relabel(v) for each $v \in Z$.
- **(4-3)** Go to Step 2.

Step 5: Determine the set X of vertices reachable from S in G_I. If $\delta \geq 1/n^2$, then apply Reduce(x, I), $\delta := \delta/2$, and go to Step 1.

We now intend to show that the scaling algorithm obtains a minimizer of f.

Lemma 3.1 *At the end of each scaling phase, $z^-(V) \geq f(X) - n(n+1)\delta/2$.*

Proof. At the end each scaling phase, $d(v) = n$ for every $v \in V \backslash W$. Since $d(v)$ is a lower bound on the number of arcs from S to v, this means there is no directed path from S to $V \backslash W$ in G_I. Thus we have $X \subseteq W \subseteq V \backslash T$, which implies $z(v) \leq \delta$ for $v \in X$. It follows from $S \subseteq X$ that $z(v) \geq -\delta$ for $v \in V \backslash X$. Since there is no arc in G_I emanating from X, we have $y_i(X) = f(X)$ for each $i \in I$, and hence $x(X) = \sum_{i \in I} \lambda_i y_i(X) = f(X)$. Therefore, we have $z^-(V) = z^-(X) + z^-(V \backslash X) \geq z(X) - \delta|X| - \delta|V \backslash X| = x(X) + \partial\varphi(X) - n\delta \geq f(X) - n(n+1)\delta/2$. ■

Lemma 3.2 *At the end of each scaling phase, $x^-(V) \geq f(X) - n^2\delta$.*

Proof. Since $z = x + \partial\varphi$, we have $x^-(V) \geq z^-(V) - n(n-1)\delta/2$, which together with Lemma 3.1 implies $x^-(V) \geq f(X) - n^2\delta$. ■

Theorem 3.3 *At the end of the last scaling phase, X is a minimizer of f.*

Proof. Since $\delta < 1/n^2$ in the last scaling phase, Lemma 3.2 implies $x^-(V) > f(X) - 1$. Then it follows from the integrality of f that $f(X) \leq f(Y)$ holds for any $Y \subseteq V$. ■

4 Complexity

This section is devoted to complexity analysis of the new scaling algorithm.

Lemma 4.1 *Each scaling phase performs* Augment $\mathrm{O}(n^2)$ *times.*

Proof. At the beginning of each scaling phase, the set X obtained in the previous scaling phase satisfies $z^-(V) \geq f(X) - 2n^2\delta$ by Lemma 3.2. For the first scaling phase, we have the same inequality by taking $X = \emptyset$. Note that $z^-(V) \leq z(Y) \leq f(Y) + n(n-1)\delta/2$ for any $Y \subseteq V$ throughout the procedure. Thus each scaling phase increases $z^-(V)$ by at most $3n^2\delta$. Since each augmentation increases $z^-(V)$ by δ, each scaling phase performs at most $3n^2$ augmentations. ■

Lemma 4.2 *Each scaling phase performs* Relabel $\mathrm{O}(n^2)$ *times.*

Proof. Each application of Relabel(v) increases $d(v)$ by one. Since Relabel(v) is applied only if $d(v) < n$, Relabel(v) is applied at most n times for each $v \in V$ in a scaling phase. Thus the total number of relabels in a scaling phase is at most n^2. ■

Lemma 4.3 *The number of indices in I is at most $2n$.*

Proof. A new index is added as a result of nonsaturating Multiple-Exchange. In a nonsaturating Multiple-Exchange(i, u, v), at least one vertex in R becomes reachable from S in G_φ, which means the set W is enlarged. Thus there are at most n applications of nonsaturating Multiple-Exchange between augmentations. Hence the number of indices added between augmentations is at most n. After each augmentation, the number of indices is reduced to at most n. Thus $|I| \leq 2n$ holds. ■

In order to analyze the number of function evaluations in each scaling phase, we now introduce the notion of reordering phase. A *reordering phase* consists of consecutive applications of Multiple-Exchange between those of Relabel or Reduce. By Lemmas 4.1 and 4.2, each scaling phase performs $\mathrm{O}(n^2)$ reordering phases.

Lemma 4.4 *There are $\mathrm{O}(n^2)$ function evaluations in each reordering phase.*

Proof. The number of function evaluation in Multiple-Exchange(i, u, v) is at most the number of vertices in the active interval for (i, u, v). In order to bound the total number of function evaluations in a reordering phase, suppose the procedure Multiple-Exchange(i, u, v) marks each pair (i, w) for w in the active interval. We now intend to claim that any pair (i, w) of $i \in I$ and $w \in V$ is marked at most once in a reordering phase.

In a reordering phase, the algorithm does not change the labeling d nor delete a vertex from W. Hence the minimum value of d in $V \backslash W$ is nondecreasing. After execution of Multiple-Exchange(i, u, v), there will not be an active pair for i until the minimum value of d in $V \backslash W$ becomes larger. Let Multiple-Exchange(i, s, t) be the next application of Multiple-Exchange to the same index $i \in I$. Then we have $d(t) > d(v) = d(u) + 1$, which implies $v \prec_i u \prec_i t \prec_i s$ in the linear ordering L_i before Multiple-Exchange(i, u, v). Thus a pair (i, w) marked in Multiple-Exchange(i, u, v) will not be marked again in the reordering phase.

Since $|I| \leq 2n$ by Lemma 4.3, there are at most $2n^2$ possible marks without duplications. Therefore, the total number of function evaluations in a reordering phase is $\mathrm{O}(n^2)$. ■

Theorem 4.5 *The algorithm performs $\mathrm{O}(n^4 \log M)$ function evaluations and $\mathrm{O}(n^5 \log M)$ arithmetic computations.*

Proof. Since $-2M \leq x^-(V)$ for $x \in \mathrm{B}(f)$, the initial value of δ satisfies $\delta \leq 2M/n^2$. Each scaling phase cuts the value of δ in half, and the algorithm terminates when $\delta < 1/n^2$. Thus the algorithm consists of $\mathrm{O}(\log M)$ scaling phases.

Since each scaling phase performs $\mathrm{O}(n^2)$ reordering phases, Lemma 4.4 implies that the number of function evaluations in a scaling phase is $\mathrm{O}(n^4)$. In addition, by Lemma 4.1, each scaling phase performs $\mathrm{O}(n^2)$ calls of Reduce, which requires $\mathrm{O}(n^3)$ arithmetic computations. Thus each scaling phase consists of $\mathrm{O}(n^4)$ function evaluations and $\mathrm{O}(n^5)$ arithmetic computations. Therefore, the total running time bound is $\mathrm{O}((n^4\gamma + n^5) \log M)$. ■

5 Discussions

A family $\mathcal{D} \subseteq 2^V$ is called a distributive lattice (or a ring family) if $X \cap Y \in \mathcal{D}$ and $X \cup Y \in \mathcal{D}$ for any pair of $X, Y \in \mathcal{D}$. A compact representation of $\mathcal{D}$ is given by a directed graph as follows. Let $D = (V, F)$ be a directed graph with the arc set F. A subset $Y \subseteq V$ is called an ideal of D if no arc enters Y in D. Then the set of ideals of D forms a distributive lattice. Conversely, any distributive lattice $\mathcal{D} \subseteq 2^V$ with $\emptyset, V \in \mathcal{D}$ can be represented in this way. Moreover, we may assume that the directed graph D is acyclic.

For minimizing a submodular function f on $\mathcal{D}$, we apply the scaling algorithm with a minor modification. The modified version uses the directed graph $G_\varphi = (V, A_\varphi \cup F)$ instead of $G_\varphi = (V, A_\varphi)$. The initial linear ordering L_0 must be consistent with D, i.e., $v \preceq_i u$ if $(u, v) \in F$. Then all the linear orderings that appear in the algorithm will be consistent with D. This ensures that the set X obtained at the end of each scaling phase belongs to $\mathcal{D}$. Thus the modification of our scaling algorithm finds a minimizer of f in $\mathcal{D}$.

Iwata–Fleischer–Fujishige [11] also describes a strongly polynomial algorithm that repeatedly applies their scaling algorithm with $\mathrm{O}(\log n)$ scaling phases. The number of iterations is $\mathrm{O}(n^2)$. Replacing the scaling algorithm by the new one, we obtain an improved strongly polynomial algorithm that runs in $\mathrm{O}((n^6\gamma + n^7) \log n)$ time.

A very recent paper [10] has shown that the strongly polynomial IFF algorithm can be implemented by using only additions, subtractions, comparisons, and oracle calls for function values. Similarly, the new strongly polynomial scaling algorithm can be made fully combinatorial as follows.

The first step towards a fully combinatorial implementation is to neglect Reduce. This causes growth of the number of extreme bases for convex combination. However, the number is still bounded by a polynomial in n. Since the number of indices added between augmentations is at most n, each scaling phase yields $O(n^3)$ new extreme bases. Hence the number of extreme bases through the $O(\log n)$ scaling phases is $O(n^3 \log n)$.

The next step is to choose an appropriate step length in Multiple-Exchange, so that the coefficients should be rational numbers with a common denominator bounded by a polynomial in n. Let σ denote the value of δ in the first scaling phase. For each $i \in I$, we keep $\lambda_i = \mu_i \delta / \sigma$ with an integer μ_i. We then modify the definition of saturating Multiple-Exchange. Multiple-Exchange(i, u) is now called saturating if $\lambda_i \xi(q, r) \leq \varphi(q, r)$ for every $(q, r) \in Q \times R$. Otherwise, it is called nonsaturating. In nonsaturating Multiple-Exchange(i, u), let ν be the minimum integer such that $\nu \xi(q, r) > \varphi(q, r)\sigma/\delta$ for some $(q, r) \in Q \times R$. Then the new coefficients λ_k and λ_i are determined by $\mu_k := \mu_i - \nu$ and $\mu_i := \nu$. Thus the coefficients are rational numbers whose common denominator is σ/δ, which is bounded by a polynomial in n through the $O(\log n)$ scaling phases. Then it is easy to implement this algorithm using only additions, subtractions, comparisons, and oracle calls for the function values.

Finally, we discuss time complexity of the resulting fully combinatorial algorithm. The algorithm performs $O(n^2)$ iterations of $O(\log n)$ scaling phases. Since it keeps $O(n^3 \log n)$ extreme bases, each scaling phase requires $O(n^6 \log n)$ oracle calls for function evaluations and $O(n^6 \log n)$ fundamental operations. Therefore, the total running time is $O(n^8 \gamma \log^2 n)$. This improves the previous $O(n^9 \gamma \log^2 n)$ bound in [10] by a factor of n.

Acknowledgements

The author is grateful to Lisa Fleischer, Satoru Fujishige, Yasuko Matsui, and Kazuo Murota for stimulating conversations and very helpful comments on the manuscript. The idea of Multiple-Exchange was originally suggested by Satoru Fujishige as a heuristics to improve the practical performance of the IFF algorithm.

References

[1] W. H. CUNNINGHAM, *Testing membership in matroid polyhedra*, J. Combin. Theory, Ser. B, 36 (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, pp. 69–87, 1970.

[4] L. FLEISCHER AND S. IWATA, *A push-relabel framework for submodular function minimization and applications to parametric optimization*, Discrete Appl. Math., to appear.

[5] L. FLEISCHER, S. IWATA, AND S. T. MCCORMICK, *A faster capacity scaling algorithm for submodular flow*, Math. Programming, to appear.

[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] S. IWATA, *A fully combinatorial algorithm for submodular function minimization*, J. Combin. Theory, Ser. B, to appear.

[11] S. IWATA, L. FLEISCHER, AND S. FUJISHIGE, *A combinatorial strongly polynomial algorithm for minimizing submodular functions*, J. ACM, 48 (2001), 761–777.

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

[13] A. SCHRIJVER, *A combinatorial algorithm minimizing submodular functions in strongly polynomial time*, J. Combin. Theory, Ser. B, 80 (2000), 346–355.

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