A FASTER SCALING ALGORITHM FOR MINIMIZING SUBMODULAR FUNCTIONS [*]

SATORU IWATA [†]

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.

Key words. submodular function, discrete optimization, algorithm

AMS subject classifications. 90C27

1. Introduction.

Let V be a finite nonempty set of cardinality n. A set function f defined on the subsets of 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 of submodular functions 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 [1]. In this paper, we combine these two techniques to yield a faster combinatorial algorithm.

[*]This research is supported in part by a Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan.

[†]Department of Mathematical Informatics, University of Tokyo, Tokyo 113-8656, Japan.

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 $\mathrm{O}(n^5 \gamma \log M)$ time. The strongly polynomial version runs in $\mathrm{O}(n^7 \gamma \log n)$ time, whereas an improved variant of Schrijver's algorithm runs in $\mathrm{O}(n^7 \gamma + n^8)$ time [4].

The time complexity of our new scaling algorithm is $\mathrm{O}((n^4 \gamma + n^5) \log M)$. 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 $\mathrm{O}((n^6 \gamma + n^7) \log n)$ time. This is an improvement over the previous best bound by an almost linear factor in n.

These combinatorial algorithms perform multiplications and divisions, despite 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 $\mathrm{O}(n^8 \gamma \log^2 n)$ 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 by 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*

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

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 by $v \preceq_i u$ if v precedes u in L_i. 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 $d(v) = 0$ for $v \in T$ and $v \preceq_i u$ implies $d(u) \leq d(v) + 1$. A valid labeling $d(v)$ serves as a lower bound on the number of arcs from v to T 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 u be a vertex that attains the minimum labeling $d(u)$ in W. An index i is called *active* for u if there is an element $v \in V$ such that $v \preceq_i u$ and $d(u) = d(v) + 1$. The procedure $\mathsf{Multiple\text{-}Exchange}(i, u)$ is applicable to a pair of such $i \in I$ and $u \in W$.

Let v be the first element in L_i with $d(u) = d(v)+1$. Since u attains the minimum labeling $d(u)$ in W, we have $v \in V \backslash W$. Note that every element w between v and u satisfies $d(v) \leq d(w) \leq d(u)$. Let Q and R be the set of elements respectively in W and in $V \backslash W$ between v and u in L_i, i.e., $Q = \{w \mid w \in W,\, v \prec_i w \preceq_i u\}$ and $R = \{w \mid w \in V \backslash W,\, v \preceq_i w \prec_i u\}$.

The procedure $\mathsf{Multiple\text{-}Exchange}(i, u)$ moves the vertices in Q to the place immediately before v 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 $\lambda_i \eta \leq \delta$, $\mathsf{Multiple\text{-}Exchange}(i, u)$ is called *saturating*. Otherwise, it is called *nonsaturating*.

In the nonsaturating $\mathsf{Multiple\text{-}Exchange}(i, u)$, 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 $(q,r) \in Q \times R$.

If there is no active index for the vertex u, the algorithm increment $d(u)$ by one. Then the labeling d remains valid. This procedure is referred to as $\mathsf{Relabel}(u)$.

The number of extreme bases in the expression of x increases as a result of nonsaturating $\mathsf{Multiple\text{-}Exchange}$. In order to reduce the complexity, the algorithm occasionally applies a procedure $\mathsf{Reduce}(x, I)$ that computes an expression of x as a convex combination of affinely independent extreme bases chosen from the currently used ones. This computation is a standard linear programming technique that takes $O(n^3)$ 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 $x := y_0$, $\lambda_0 := 1$, $I := \{0\}$, $\delta := |x^-(V)|/n^2$, and $\varphi(u,v) := 0$ for $u, v \in V$.

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 $\mathsf{Augment}(\varphi, P)$.

(3-2) Apply $\mathsf{Reduce}(x, I)$.

(3-3) Go to Step 2.

Step 4: Find a vertex $u \in W$ that attains the minimum $d(u)$. If $d(u) < n$, then do the following.

(4-1) If there is an active index i for u, then apply $\mathsf{Multiple\text{-}Exchange}(i, u)$.

(4-2) Otherwise, apply $\mathsf{Relabel}(u)$.

(4-3) Go to Step 2.

Step 5: Determine the set X of vertices reachable from W in G_I. If $\delta \geq 1/n^2$, then apply $\mathsf{Reduce}(x, I)$, $\delta := \delta/2$, 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 $v \in W$. Since $d(v)$ is a lower bound on an number of arcs from v to T, this means there is no directed path from W to T in G_I. Thus we have $T \cap X = \emptyset$, which implies $z(v) \leq \delta$ for

$v \in X$. It follows from $S \subseteq W \subseteq X$ 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 $x^-(V) \geq f(X) - 1$. 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 the 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 by the previous scaling phase satisfies $z^-(V) \geq f(X) - n(n+1)\delta$ by Lemma 3.1. For the first scaling phase, we have the same inequality by taking $X = \emptyset$. Note that $z^-(V) \leq z(Y) \leq f(Y) + n(n-1)\delta$ for any $Y \subseteq V$ throughout the procedure. Thus each scaling phase increases $z^-(V)$ by at most $2n^2\delta$. Since each augmentation increases $z^-(V)$ by δ, each scaling phase performs at most $2n^2$ augmentations. □

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

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

Lemma 4.3. *Each scaling phase performs nonsaturating* Multiple-Exchange $\mathrm{O}(n^3)$ *times.*

Proof. If Multiple-Exchange(i, u) is nonsaturating, 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. Since each scaling phase performs $\mathrm{O}(n^2)$ augmentations by Lemma 4.1, the number of nonsaturating applications of Multiple-Exchange in a scaling phase is $\mathrm{O}(n^3)$. □

Lemma 4.4. *After an application of* Multiple-Exchange(i,u)*, the index i remains inactive until* Relabel(u) *is applied.*

Proof. After Multiple-Exchange(i,u), we have $d(v) \geq d(u)$ for every vertex $v \preceq_i u$. While $d(u)$ remains the same, a new vertex w that becomes to precede u in L_i must satisfy $d(w) > d(u)$. Thus the index i remains inactive for u until Relabel(u) is applied. $\square$

Lemma 4.5. *Each scaling phase performs saturating* Multiple-Exchange $\mathrm{O}(n^3)$ *times.*

Proof. Let $\rho(u)$ denote the number of indices active for $u \in V$. It follows from $|I| \leq 2n$ that $\rho(u) \leq 2n$ for each $u \in V$. A saturating Multiple-Exchange(i,u) decreases $\rho(u)$ by one, while a nonsaturating Multiple-Exchange(i,u) does not change $\rho(u)$. By Lemma 4.4, $\rho(u)$ does not increase unless the algorithm performs Relabel(u). Since each scaling phase performs $\mathrm{O}(n^2)$ calls of Relabel by Lemma 4.2, the total number of saturating Multiple-Exchange in a scaling phase is $\mathrm{O}(n^3)$. $\square$

Theorem 4.6. *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.

By Lemmas 4.3 and 4.5, each scaling phase performs Multiple-Exchange $\mathrm{O}(n^3)$ times. The procedure Multiple-Exchange requires $\mathrm{O}(n)$ oracle calls for the function values and $\mathrm{O}(n)$ arithmetic computations. In addition, by Lemma 4.2, 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 $\mathrm{O}((n^4\gamma + n^5)\log M)$. $\square$

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 arcs 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 a new index is added in nonsaturating Multiple-Exchange, each scaling phase yields $\mathrm{O}(n^3)$ new extreme bases. Hence the number of extreme bases through the $\mathrm{O}(\log n)$ scaling phases is $\mathrm{O}(n^3 \log n)$.

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 $\mathrm{O}(\log n)$ 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 the running time of the resulting fully combinatorial algorithm. The algorithm performs $\mathrm{O}(n^2)$ iterations of $\mathrm{O}(\log n)$ scaling phases. Since the algorithm keeps $O(n^3 \log n)$ extreme bases, each scaling phase applies Multiple-Exchange $O(n^5 \log n)$ times. Each Multiple-Exchange consists of $O(n)$ oracle calls and $O(n)$ arithmetic operations. Therefore, the total running time is $O(n^8 \gamma \log^2 n)$. This improves the previous bound $O(n^9 \gamma \log^2 n)$ in [10] by a factor of n.

Acknowledgements. The author thanks Satoru Fujishige, Yasuko Matsui, and Kazuo Murota for stimulating conversations and helpful comments.

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, to appear.

[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.