A Faster Strongly Polynomial Time Algorithm for Submodular Function Minimization

James B. Orlin

Sloan School of Management, MIT
Cambridge, MA 02139
jorlin@mit.edu

Abstract. We consider the problem of minimizing a submodular function f defined on a set V with n elements. We give a combinatorial algorithm that runs in $O(n^5 \text{ EO} + n^6)$ time, where EO is the time to evaluate $f(S)$ for some $S \subseteq V$. This improves the previous best strongly polynomial running time by more than a factor of n.

1 Introduction

Let $V = \{1, 2, \ldots, n\}$. A set function f on V is said to be *submodular* if the following is true:

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \quad \text{for all subsets } X, Y \subseteq V. \tag{1}$$

Here we consider the problem of Submodular Function Minimization (SFM), that is, determining a subset $S \subseteq V$ that minimizes $f(\ )$. Our contribution is to develop a strongly polynomial time algorithm for SFM that improves upon the best previous time bounds by a factor greater than n.

For a given subset $X \subseteq V$, let $f_X(Y) = f(X \cup Y) - f(X)$. It is elementary and well known that for fixed X, the function $f_X(\ )$ is submodular whenever $f(\ )$ is submodular. An equivalent way of defining submodularity is as follows.

For all subsets X, Y of V, and for each element $v \notin (X \cup Y)$, if $X \subseteq Y$ then $f_Y(v) \leq f_X(v)$.

In this way, submodular functions model decreasing marginal returns, and are economic counterparts of concave functions. Nevertheless, Lovasz [11] showed that they behave algorithmically more similarly to convex functions, and provided analysis on why this is true.

Examples of submodular functions include cut capacity functions, matroid rank functions, and entropy functions. For additional examples of submodular functions and for applications of SFM see McCormick [12], Fleischer [4], Fushishige [6], and Schrijver [14].

We assume without loss of generality that $f(\emptyset) = 0$. Otherwise, if $f(\emptyset) \neq 0$, we can subtract $f(\emptyset)$ from $f(S)$ for all $S \subseteq V$.

Grotschel, Lovasz, and Schrijver [7] and [8] gave the first polynomial time and strongly polynomial time algorithms for minimizing a submodular function. Their

M. Fischetti and D.P. Williamson (Eds.): IPCO 2007, LNCS 4513, pp. 240–251, 2007.

algorithms rely on the ellipsoid algorithm. Schrijver [13] and Iwata, Fleischer, and Fujishige [10] independently developed strongly polynomial time combinatorial algorithms for minimizing a submodular function. Both algorithms build on the work of Cunningham [1], who developed a pseudo-polynomial time algorithm for minimizing a submodular function.

Let EO be the maximum amount of time it takes to evaluate $f(S)$ for a subset $S \subseteq V$. EO stands for evaluation of the oracle function, as per McCormick [12]. In general, one expects EO to be at least n since the input size is $\Omega(n)$; however, this running time can sometimes be improved in an amortized sense if one is evaluating EO multiple times consecutively, as is done by many of the SFM algorithms including the one presented here. Let M be an upper bound on $|f(S)|$ for all $S \subseteq V$.

The running times of the algorithms of Schrijver [13] and Iwata, Fleischer, and Fujishige [10] were shown to be $O(n^8 \text{ EO} + n^9)$. Fleischer and Iwata [5] improved the running time of the combinatorial algorithms to $O(n^7 \text{ EO} + n^8)$. Vygen [15] showed that the running time of Schrijver's original algorithm was also $O(n^7 \text{ EO} + n^8)$. Subsequently Iwata [9] developed a scaling based algorithm whose running time is $O(n^4 \text{ EO} \log M + n^5 \log M)$. To date, the best strongly polynomial time combinatorial algorithm for SFM was the strongly polynomial version of Iwata's algorithm, which runs in $O((n^6 \text{ EO} + n^7) \log n)$ time.

We present a new approach for solving submodular minimization. As have previous approaches, our algorithm relies on expressing feasible points in the base polyhedron as a convex combination of extreme points. However, our algorithm works directly with vectors of the base polyhedron rather than relying on an auxiliary network, or on augmenting paths, or on flows.

We present a strongly polynomial time algorithm that runs in $O(n^5 \text{ EO} + n^6)$ steps, thus improving upon Iwata's time bound by a factor of $n \log n$. This also improves upon the best strongly polynomial time implementation of the ellipsoid algorithm for SFM, which runs in $\tilde{O}(n^5 \text{EO} + n^7)$ as reported by McCormick [12], where $\tilde{O}$ indicates that factors of $\log n$ may have been omitted from the time bound. Most of the proofs in this manuscript are omitted. A complete draft including the proofs is available on the author's website.

2 The Base Polyhedron

For a vector $x \in \mathbb{R}^{|V|}$, let $x(v)$ denote the v-th component. We let $x^-(v) = \min\{0, x(v)\}$. For a subset $S \subseteq V$, , we let $x(S) = \sum_{v \in S} x(v)$.

The *base polyhedron* is

$$B(f) = \{x \mid x \in \mathbb{R}^n, x(V) = f(V), \forall S \subseteq V : x(S) \leq f(S)\}.$$

A vector in $B(f)$ is called a *base*. An extreme point of $B(f)$ is called an *extreme base*. Edmonds 3 established the following duality theorem, which Cunningham 1 used to develop a pseudo-polynomial time algorithm for SFM. Subsequently all other efficient algorithms for SFM use the following duality theorem or a closely related result.

Theorem 1 (*Edmonds*). *For a submodular function* $f: 2^V \rightarrow \mathbb{R}$.

$$\max\{x^-(V): x \in B(f)\} = \min\{f(S): S \subseteq V\}. \tag{2}$$

The function $x^-(\,)$ is not linear, and the optimizer of $\max\{x^-(V): x \in B(f)\}$ is not, in general, an extreme point of the base polyhedron. The polynomial time algorithms in [9] and [10] proceed by representing vectors in the base polyhedron as a convex combination of extreme bases of the base polyhedron.

An extreme base can be computed by the greedy algorithm of Edmonds and Shapley [3] as follows: Let $L = \{v_1, \ldots, v_n\}$ be any linear ordering (permutation) of the elements of V. In our notation, for each j, v_j is in the j-th position of the permutation. The extreme base y_L *induced* by L is obtained by letting

$$y_L(v_j) = f(\{v_1, \ldots, v_j\}) - f(\{v_1, \ldots, v_{j-1}\}) \text{ for } j = 1 \text{ to } n.$$

If $P(j) = \{v_1, \ldots, v_j\}$, then we can also write $y_L(v_j) = f_{P(j-1)}(v_j)$, which is the marginal contribution for f of adding v_j to $\{v_1, \ldots, v_{j-1}\}$.

3 Distance Functions and Optimality Conditions

A *distance function* is a mapping $d: V \rightarrow \{0,1,\ldots,n\}$. Each distance function d induces a linear order $L(d)$ (denoted as $\prec_d$) of V as follows: $u \prec_d v$ if $d(u) < d(v)$ or if $d(u) = d(v)$ and $u < v$. The extreme base induced by the order $L(d)$ will be denoted as y_d.

In the algorithm presented in Section 5, at each iteration of the algorithm, we will maintain a collection D of $O(n)$ different distance functions of V, a vector x in the base polyhedron, and a vector λ. The vectors x and λ satisfy the following:

$$x = \sum_{d \in D} \lambda_d y_d \,, \sum_{d \in D} \lambda_d = 1, \text{ and } \lambda \geq 0. \tag{3}$$

We also write this as $x = \lambda_D y_D$, where $y_D = \{y_d : d \in D\}$. We let $D_{\min}(v)$ be shorthand for $\min\{d(v) : d \in D\}$. We say that the triple (x, λ, D) is *valid* if the following is true:

1. If $x(v) < 0$, then $d(v) = 0$ for all $d \in D$;
2. $d(v) \leq D_{\min}(v) + 1$ for all $d \in D$ and $v \in V$;

The algorithm will maintain a valid triple (x, λ, D) throughout all iterations. Sometimes, we will just say that the collection D of distance functions is valid.

Definition. We say that the quadruple (D, λ, x, S) satisfies the *optimality conditions* if it satisfies (3) and if it satisfies (4–6).

$$x(v) \leq 0 \quad \text{for } v \in S \tag{4}$$

$$x(v) \geq 0 \quad \text{for } v \in V \setminus S \tag{5}$$

$$v \prec_d w \text{ for all } d \in D, v \in S \text{ and } w \in V \setminus S. \tag{6}$$

We will also say that the triple (D, λ, x) satisfies the optimality conditions if there is a subset $S \subseteq V$ such that (D, λ, x, S) satisfies the optimality conditions. By (6), given any element $d \in V$, one can narrow the choice of S to n possibilities.

Lemma 1 (Sufficiency of Optimality Conditions). If the quadruple (D, λ, x, S) for SFM satisfies the optimality conditions, then S is a minimum cost set, and x is an optimal base in the base polyhedron.

Proof. By assumption, x is in the base polyhedron. Moreover, suppose without loss of generality that the elements are reordered so that $S = \{1, 2, \ldots, |S|\}$. Then

$$x^-(V) = x^-(S) = x(S) = \sum_{d \in D} \lambda_d y_d(S) = \sum_{d \in D} \lambda_d f(S) = f(S). \tag{7}$$

Thus $x^-(V) = f(S)$, and by Theorem 1, S is optimal. ♦

Lemma 2 (Existence of Optimality Conditions). If S is a minimum cost set for SFM, then there is a quadruple (D, λ, x, S) for SFM that satisfies the optimality conditions.

Proof. Let x be an optimal base in the base polyhedron. Moreover, suppose without loss of generality that the elements are reordered so that $S = \{1, 2, \ldots, |S|\}$. Then

$$x^-(V) \le x^-(S) \le x(S) = \sum_{d \in D} \lambda_d y_d(S) = \sum_{d \in D} \lambda_d f(S) = f(S). \tag{8}$$

Since $x^-(V) = f(S)$, it follows that (D, λ, x, S) satisfies the optimality conditions. (We have not established that D is valid, but our algorithm will produce a valid D as well). ♦

Definition. We say that **t**he quadruple (D, λ, x, S) satisfies the *partial optimality conditions* if it satisfies (3) and if it satisfies (5) and (6).

Lemma 3 (Partial Optimality Conditions). If the quadruple (D, λ, x, S) for SFM satisfies the partial optimality conditions, then there is a minimum cost set $S^* \subseteq S$.

We first claim that if the partial optimality conditions are satisfied, then $\varnothing$ is an optimal set for f_S among subsets of $V \backslash S$. If the claim is true then for any subset T of V,

$$f(T) \ge f(S \cap T) + f(S \cup T) - f(S) = f(S \cap T) + f_S(T \backslash S) \ge f(S \cap T).$$

So, if the claim is true, then the Lemma is true. We next prove the claim.

For each $d \in D$, let y'_d be the extreme base induced by d for the base polyhedron $B(f_S)$ defined over the set of elements $u \in V \backslash s$, and let $x' = \sum_{d \in D} \lambda_d y'_d$. We will show that for each $d \in D$ and for each $u \in V \backslash S$, $y'_d(u) = y_d(u)$. If this statement is true, it follows that $x'(u) = x(u)$ for $u \in V \backslash S$, and thus (D, λ, x', $\varnothing$) satisfies the optimality conditions for f_S over the set V\S and thus the claim is true.

So, suppose that $d \in D$ and $u \in V \backslash S$. Let $P(d,u) = \{v \in V : v \, \mathsf{p}_d \, u\}$. By (6), $S \subseteq P(d, u)$. Thus

$$y'_d(u) = f_S(u + P(d,u) \setminus S) - f_S(P(d,u) \setminus S)$$
$$= [f(u + P(d,u)) - f(S)] - [f(P(d,u)) - f(S)] = y_d(u).$$

This establishes that the claim is true, and thus the lemma is true. ♦

We will also say that the triple (D, λ, x) satisfies the partial optimality conditions if there is a subset $S \subseteq V$ such that (D, λ, x, S) satisfies the partial optimality conditions.

A sufficient condition for the partial optimality conditions to hold for valid distance functions D is the presence of a *distance gap at level k*, which is value k with $0 < k < n$ such that

1. there is some v with $D_{\min}(v) = k$, and
2. there is no u with $D_{\min}(u) = k$ -1.

By letting $S = \{u \in V$ with $D_{\min}(u) < k\}$, it is easy to verify that (D, λ, x, S) will satisfy the partial optimality conditions. In such a case, we will eliminate all elements in $V \setminus S$ from the problem. It would be possible to maintain these elements if we wanted to determine an optimal base, but they are not needed if we just want to determine a minimum cost set.

4 Distance Functions and Extreme Vectors

Suppose that d is a distance function. We let INC(d, v) be the distance function obtained by incrementing the distance label of v by 1 and keeping all other distance labels the same. That is, if d' = INC(d, v), then

$$d'(u) = \begin{cases} d(v)+1 & \text{if } u = v \\ d(u) & \text{if } u \neq v \end{cases}.$$

Lemma 4. Suppose that d' = INC(d, v). Then

1. $y_{d'}(v) \leq y_d(v)$,
2. $y_{d'}(u) \geq y_d(u)$ if $u \neq v$.

Proof. For each $u \in V$, let $P(u) = \{w \in V : w \prec_d u\}$. Let $P'(u) = \{w \in V : w \prec_{d'} u\}$. Note that $u \notin P(u)$, and $u \notin P'(u)$. Then for all $u \in V$, $y_{d'}(u) - y_d(u) = f_{P'(v)}(u) - f_{P(v)}(u)$.

Since $P(v) \subseteq P'(v)$, it follows from the submodularity of f that $f_{P'(v)}(v) \leq f_{P(v)}(v)$, and so $y_{d'}(v) - y_d(v) \leq 0$. Similarly, for $u \neq v$, $P'(u) \subseteq P(u)$, and so $f_{P(v)}(u) \leq f_{P'(v)}(u)$. ♦

For any subset $S \subseteq V$, We let $d(S) = \sum_{v \in S} d(v)$. We will maintain the distance functions in D in non-decreasing order of $d(V)$.

For each $v \in V$, we will maintain a *primary distance function* $p(v) \in D$, which is the first element d of D such that $d(v) = D_{\min}(v)$. By the way that we ordered the

elements of D, the primary distance function for v will minimize $d(V)$ among all $d \in D$ with $d(v) = D_{\min}(v)$. In addition, for every $v \in V$, we will maintain a *secondary distance function* $s(v) = \mathrm{INC}(p(v), v)$. Our algorithm modifies x by increasing $\lambda_{s(v)}$ and simultaneously decreasing $\lambda_{p(v)}$ for $v \in V$.

We maintain the order of D, and the functions $p(v)$ and $s(v)$ for all v by running the Procedure Update as follows:

Procedure Update(D, p, s)
begin
$D := \{d : \lambda_d > 0\}$;
order the vectors in D in non-decreasing order of $d(V)$;
for each $v \in V$, let $p(v)$ be the first element of D with $d(v) = D_{\min}(v)$;
for each $v \in V$, let $s(v) = \mathrm{INC}(p(v), v)$;
end

5 A Strongly Polynomial Algorithm for SFM

In this section, we present the strongly polynomial time algorithm for SFM. But first, we point out that occasionally the size of D grows too large and we want to decrease its size. Accordingly, we run a procedure called Reduce(x, λ, D) to reduce the size of D without affecting the base vector x.

Procedure Reduce(x, λ, D)

INPUT: a collection D of distance functions, a non-negative vector λ such that $\sum_{d \in D} \lambda_d = 1$. Let $x = \sum_{d \in D} \lambda_d y_d$.

OUTPUT: a subset $D' \subseteq D$ and a vector λ' such that

1. $\sum_{d \in D'} \lambda'_d = 1$ and $\lambda' \geq 0$, and $x = \sum_{d \in D'} \lambda'_d y_d$, and
2. the set $\{y_d : d \in D'\}$ is linear independent.

We will call the procedure when $3n \leq |D| < 4n$, and so the running time will be $O(n^3)$ using standard techniques from linear programming. For details on how to carry out Reduce, see Schrijver [13] or McCormick [12].

In the following procedure, let $V^0 = \{v \in V : x(v) = 0\}$. Let $V^+ = \{v \in V : x(v) > 0\}$.

Algorithm SFM
begin
$d := 0$;
$D = \{d\}$; $\lambda_d := 1$; $x := y_d$;
while the optimality conditions are not satisfied
begin
choose an element $v^* \in V^+$;
choose a vector $\gamma \geq 0$ with $\gamma \neq 0$ so that

$$\sum_{v \in V^0 + v^*} \gamma(v)[y_{s(v)}(u) - y_{p(v)}(u)] = 0 \text{ for all } u \in V^0;$$

let $x' := \sum_{v \in V^0 + v^*} \gamma(v)[y_{s(v)} - y_{p(v)}]$;
choose α maximum so that $x(u) + \alpha x'(u) \geq 0$ for all $u \in V^+$, and
$\alpha \sum_{u:p(u)=d} \gamma(u) \leq \lambda_d$ for all $d \in D$;
$x := x + \alpha x'$;
$\lambda_d := \lambda_d + \alpha \sum_{u:s(u)=d} \gamma(u) - \alpha \sum_{u:p(u)=d} \gamma(u)$ for all $d \in D \cup \{s(u): u \in V\}$;
Update(*D*, *p*, *s*);
if $|D| \geq 3n$, **then Reduce(*x*, λ, *D*)**;
if there is a distance gap at level k, then $V := \{v \in V : D_{\min}(v) \leq k\}$;
end while
end

The algorithm initializes by letting $x = y_d$, where $d(v) = 0$ for all $v \in V$. Subsequently, the algorithm continues until the optimality conditions are satisfied.

At each iteration, the algorithm selects a non-zero vector $\gamma \geq 0$ with the property that one can modify x by increasing $y_{s(v)}$ by $\gamma(v)$ and decreasing $y_{p(v)}$ by $\gamma(v)$ for all v so that the following is true: if $x(v) = 0$ prior to the modification, then $x(v) = 0$ after the modification. It is not obvious that such a vector γ exists. We prove its existence in the next section, and show that it can be determined easily by solving a system of linear equations.

Once we determine the vector γ, we modify λ and x. After the modification, at least one of the following changes takes place: either V^0 increases in size or there is some primary vector $p(v)$ that leaves D because $d_{p(v)} = 0$ after the modification. In fact, α is chosen sufficiently large so that one of these two events occur and so that no element ever leaves V^0, and so that any element leaving V^+ must enter V^0.

We reduce the size of D whenever $|D| \geq 3n$, and we eliminate elements from V whenever a distance gap is found.

In Section 7, we will show that the algorithm terminates in $O(n^6)$ steps with an optimal set. The proof of the time bound relies on a potential function argument.

6 The Auxiliary Matrix and How to Choose γ

In this section, we show how to choose γ by solving a system of at most n equations.

One of the key steps of the algorithm is as follows: choose a vector $\gamma \geq 0$ with $\gamma \neq 0$ so that

$$\sum_{v \in V^0 + v^*} \gamma(v)[y_{s(v)}(u) - y_{p(v)}(u)] = 0 \text{ for all } u \in V^0;$$

We consider two separate cases.

Case 1. $\gamma(v^*) = 0$.

In this case, we need to solve $\sum_{v \in V^0} \gamma(v)[y_{s(v)}(u) - y_{p(v)}(u)] = 0$ for all $u \in V^0$;

Case 2. $\chi(v^*) = 1$. (We can always scale γ so that this is true whenever $\chi(v^*) \neq 0$). In this case, we need to solve $\sum_{v \in V^0} \gamma(v)[y_{s(v)}(u) - y_{p(v)}(u)] = y_{p(v^*)}(u) - y_{s(v^*)}(u)$ for all $u \in V^0$.

Suppose that the rows and columns of the constraint matrix are both indexed by the elements of V^0. Then the constraint matrices for Cases 1 and 2 are identical. The right hand side b in Case 2 may be non-zero; however, by Lemma 4, $b \leq 0$.

We refer to the constraint matrix A^* for Cases 1 and 2 as the *auxiliary matrix*. By Lemma 4, the auxiliary matrix satisfies the following properties:

6.1. A^* is an $|V^0| \times |V^0|$ matrix.
6.2. The diagonal elements of A^* are non-positive.
6.3. All non-diagonal elements of A^* are non-negative.
6.4. Each column sum of A^* is non-positive.

In the case that A^* is invertible, it is the negative of what is known in the literature as an M-matrix, and thus the inverse of A^* is non-positive. See, for example, [1] for results on M-matrices.

Theorem 2. Let A^* be an auxiliary matrix. If A^* is singular, then there is a vector $w' \neq 0$, such that $w' \geq 0$, and $A^*w' = 0$. If A^* is non-singular then $(A^*)^{-1} \leq 0$, and thus the solution to $A^*w' = b$ is non-positive whenever b is non-negative.

Proof. The second half of the theorem is well known. The first half can easily be derived from [1], but we include a proof for completeness. Suppose that A^* is singular. Choose $w \neq 0$ so that $Aw = 0$. If $w \geq 0$, there is nothing to prove. Similarly if $w \leq 0$, then we can replace w by $-w$ and there is nothing to prove. So, suppose that there are $k < n$ positive coefficients of w. Without loss of generality assume that $w(v) > 0$ for $v = 1$ to k. (Otherwise, one can simultaneously reorder the rows and columns so that this is true.)

Let us write $A^* = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, where A_{11} denotes the first k rows and columns of A^*. Let us rewrite w as $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$, where w_1 denotes the first k components of w. By assumption, $A_{11}w_1 + A_{12}w_2 = 0$ and $A_{11}w_1 + A_{21}w_2 = 0$. By 6.3, $A_{12} \geq 0$. By assumption, $w_2 \leq 0$. Therefore, $A_{11}w_1 \geq 0$. We will next show that $A_{11}w_1 = 0$.

Let 1 denote a row vector of k ones. Then $1A_{11} \leq 0$ by 6.3 and 6.4. If $1A_{11} \neq 0$, then $1A_{11}w_1 < 0$, contradicting that $A_{11}w_1 \geq 0$. We conclude that $1A_{11} = 0$. It follows that $1A_{11}w_1 = 0$, which combined with $A_{11}w_1 \geq 0$ shows that $A_{11}w_1 = 0$. In addition, by 6.1c and 6.1d, $A_{21} = 0$.

Finally, we extend w to a vector w' of $|V^0|$ components by letting

$$w' = \begin{bmatrix} w_1 \\ 0 \end{bmatrix}.$$

Then $Aw' = 0$, which is what we wanted to prove. ♦

By Theorem 2, the solution for γ in cases 1 and 2 can both be found by solving a system of equations on the auxiliary matrix, which takes $O(|V^0|^3) = O(n^3)$ time. Moreover, the running time is faster when the auxiliary matrix only changes by q columns in an iteration. In this case, the time to solve the system of equations at a given iteration is $O(qn^2)$.

We note that occasionally a column of the auxiliary matrix is 0, in which case it is trivial to find a non-zero vector w' with $Aw' = 0$. However, this speedup does not affect the worst case analysis.

7 Proof of Correctness and Time Bound

In this section we establish the correctness of the SFM algorithm and show that it runs in $O(n^5 \text{ EO} + n^6)$ time.

We first establish that the following remain true throughout the execution of the algorithm:

7.1. At each iteration, there is a set D of valid distance functions, an element $x \in B(f)$ and a vector λ such that (3) is satisfied.
7.2. If $x(v) = 0$ at some iteration, then $x(v) = 0$ at all subsequent iterations;
7.3. $D_{\min}(v)$ is non decreasing over all iterations for all $v \in V$.
7.4. If $x(v) < 0$, then $D_{\min}(v) = 0$;

Theorem 3. Conditions 7.1 to 7.4 are satisfied at each stage of the algorithm SFM.

Proof. Conditions 7.1 to 7.4 are all satisfied immediately subsequent to the initialization. Suppose inductively that they are satisfied at some iteration of the algorithm, and we consider what happens after some procedure is called.

We first consider the procedure **Reduce**. This procedure maintains (3) and eliminates a number of elements of D. It is easy to verify that 7.1-7.4 remain true subsequent to the call of **Reduce**.

Next, we consider eliminating elements when a distance gap is found. This results in eliminating components from y_d for all d and from x, and also changes the base polyhedron. However, it is easy to see that 7.1 to 7.4 remain satisfied with respect to the new base polyhedron.

Finally, we consider changes that occur in Procedure SFM. When we modify λ, note that every increase in $\lambda_{s(v)}$ is matched by a decrease in $\lambda_{p(v)}$. For this reason, if $\sum_{d \in D} \lambda_d = 1$ holds prior to modifying λ, it also holds afterwards. Also, by our choice of α we modify λ in such a way that it is always non-negative, and so (3.1) is still satisfied.

The solution to the system of linear equations yields a vector x' with the property that $x'(v) = 0$ for all $v \in V^0$. So, 7.2 is true after we replace x by $x + \alpha x'$.

We next consider 7.3. The only distance functions added to D are of the form $s(v)$. If $u \neq v$, then $D_{\min}(u)$ is unchanged if $s(v)$ is added to D. As for $D_{\min}(v)$, the vector $d = p(v)$ is chosen so that $D_{\min}(v) = d(v)$. Accordingly, if $d' = s(v)$, then $d'(v) = D_{\min}(v) + 1$, and so 7.3 remains satisfied.

7.4 also remains satisfied. If $x(v) < 0$, then we do not create any distance functions with $d(v) \geq 1$. This completes the proof. ♦

Theorem 4. The SFM algorithm terminates with a set S that minimizes the submodular function and finds an optimum solution x in the base polyhedron. The algorithm runs in $O(n^5 EO + n^6)$ time.

Prior to proving the main theorem, we state our potential function, and prove three lemmas.

For $v \in V$, let $h(v) = d(V)$, where $d = p(v)$. Thus $h(v)$ is the sum of the distances in $p(v)$.

Let $\hat{h}(v) = \sum_{u \in V} (d(u) - D_{\min}(u))$. Since D is valid, it follows that $0 \le \hat{h}(v) \le n$ for all $v \in V$. Moreover, $h(v) - \hat{h}(v) = \sum_{v \in V} D_{\min}(v)$.

Let $H(v) = \{d \in D: d(V) = h(v) \text{ and } D_{\min}(v) = d(v)\}$. Note that any distance functions in $H(v)$ could have been chosen as a primary distance function for v if we had broken ties differently in ordering the elements of D.

We define the potential function Φ as follows:

$$\Phi(v) = |H(v)| \text{ and } \Phi = \sum_{v \in V^{in}} \Phi(v).$$

The next two lemmas concern $h(v)$ and $H(v)$.

Lemma 5. For each $v \in V$, the number of times that $h(v)$ changes over all iterations of the algorithm is $O(n^2)$.

Proof. We will actually bound the number of changes in $\hat{h}(v)$. Note that it is possible for $h(v)$ to change while $\hat{h}(v)$ stays constant if $D_{\min}(u)$ increases. But the number of changes in $D_{\min}(\ )$ over all iterations is $O(n^2)$. If the number of changes of $\hat{h}(v)$ is $O(n^2)$, then so is the number of changes of $h(v)$.

Recall that $0 \le \hat{h}(v) \le n$. We first consider changes in $\hat{h}(v)$ in between successive changes in $D_{\min}(v)$, and we refer to this set of iterations as a *phase*. The value $\hat{h}(v)$ cannot decrease during a phase unless $D_{\min}(u)$ increases for some $u \in V$, in which case $\hat{h}(v)$ can decrease by at most 1. All other changes in $h(v)$ during the phase are increases. So the total number of changes in $\hat{h}(v)$ is at most n plus the two times the number of increases in $D_{\min}(u)$ for some u. Suppose that we "charge" the latter changes in $\hat{h}(v)$ to changes in $D_{\min}$. In this case, the number of charged changes in $\hat{h}(v)$ over all iterations is $O(n^2)$, and the number of other changes in $\hat{h}(v)$ is at most n per phase. So the number of changes in $\hat{h}(v)$ is $O(n^2)$ over all phases. ♦

Lemma 6. The distance function $s(v) \notin H(u)$ for any $u \in V$.

Proof. Let $d = p(v)$, and let $d' = s(v)$. We note that $d' \notin H(v)$ because $d'(V) = h(v) + 1$. So, we consider $u \ne v$. If $D_{\min}(u) = d'(u)$, then $D_{\min}(u) = d(u)$. In this case $h(u) \le d(V) < d'(V)$, and so $d' \notin H(u)$. ♦

We next prove a lemma concerning the potential function Φ. We note that Φ decreases at some iterations and increases at others. By the *total decrease* in Φ over all iterations, we mean the sum of the decreases in Φ as summed over all iterations at which Φ decreases. We define *total increase* analogously.

Lemma 7. The total increase in Φ over all iterations is $O(n^4)$, and the total decrease in Φ over all iterations is also $O(n^4)$.

Proof. Given that $\Phi = O(n^2)$, it suffices to show that the total increase over all iterations is $O(n^4)$ after which the $O(n^4)$ bound on the total decrease will follow.

We first note that the only vectors that are added to D are vectors $d = s(v)$ for some $v \in V^0$. By Lemma 6, these additions to D do not change the potential function (until $p(v)$ is deleted from D). The potential function changes only when one of the following two steps takes place:

1. changes in $H(v)$ while $h(v)$ remains constant;
2. changes in $H(v)$ when $h(v)$ also changes.

By Lemma 6, each change in $H(v)$ while $h(v)$ remains constant can only result in a decrease in $\Phi(v)$. So, we only need to bound increases in changes in Φ when $h(v)$ changes for some v.

Each change in $h(v)$ can lead to an increase of at most $|D| = O(n)$ in $\Phi(v)$. By Lemma 5, there are $O(n^2)$ changes in $h(v)$ for each v and thus the total increase in Φ over all iterations due changes in $h(\ )$ is $O(n^4)$. ♦

We are now ready to prove Theorem 4.

Proof of Theorem 4. We first note that if the algorithm terminates, then it must terminate with an optimal solution since satisfying the optimality conditions is the only termination criterion.

The bottlenecks of the algorithm are the following:

1. Adding columns $A(v) = s(v) - p(v)$ to the auxiliary matrix.
2. Solving a system of equations $A^*w = b$ or $A^*w = 0$;
3. Reducing the number of columns in D via Procedure Reduce.

We add a column to $A(v)$ only when $p(v)$ was deleted from D. A deletion of $p(v)$ for some v either leads to a change in $h(v)$ or else it leads to a decrease in $|H(v)|$. The former can happen $O(n^3)$ times by Lemma 5. We now consider the latter case.

Deleting a single element $d = p(v)$ can result in several columns needing to be added to A^*. In particular, it is possible that $d = p(u)$ for a subset $U \subseteq V$. If d is deleted from D, then we need to replace $|U|$ different columns of A^*. But in this case, deleting d from D reduces $|H(u)|$ for all $u \in U$, and thus reduces Φ by $|U|$. We conclude that the number of columns added to A^* is at most the total decrease in Φ over all iterations, which is $O(n^4)$ by Lemma 7.

Thus the running time for adding columns to the auxiliary matrix is $O(n^5\ \mathrm{EO})$ since determining the values for a column takes $O(n\ \mathrm{EO})$ steps. For each column added to the auxiliary matrix, it takes $O(n^2)$ time to carry out elementary row operations to get A^* into canonical form for solving the system of equations. This takes $O(n^6)$ time over all iterations. Thus the running time for adding columns to A^* and carrying out elementary row operations is $O(n^5\ \mathrm{EO} + n^6)$.

We call the procedure **Reduce** when $|D| \geq 3n$, and we eliminate at least $2n$ elements of D. The running time is thus $O(n^3)$ for each call of **Reduce**. Each distance function d that is deleted from D must have been added as a vector of the form $s(v)$ at

some iteration, and this happens only $O(n^4)$ times. Thus the total time to carry out Reduce is $O(n^6)$.

We conclude that the total running time is $O(n^5 \, EO + n^6)$ time. ♦

We have developed a strongly polynomial time algorithm for SFM that dominates previous strongly polynomial time algorithms by a factor greater than n. Moreover, whereas other algorithms rely on the combinatorics of paths and flows, our algorithm relies on an iterative local search plus a combinatorial potential function argument.

Acknowledgments. This research was supported by the Office of Naval Research under Grant N00014-98-1-0317. I also thank Professors Satoru Iwata and Satoru Fujishige for their constructive comments on an earlier draft of this manuscript.

References

1. Berman, A., and Plemmons R. J.: *Nonnegative Matrices in the Mathematical Sciences*, SIAM, 1994.
2. Cunningham, W. H.. On Submodular Function Minimization: *Combinatorica* 3 (1985) 185-192.
3. Edmonds, J.: Submodular Functions, Matroids, and Certain Polyhedra. In Combinatorial Structures and their Applications, R. Guy, H. Hanani, N. Sauer, and J. Schönheim, eds., Gordon and Breach (1970) 69-87.
4. Fleischer, L. K.: Recent Progress in Submodular Function Minimization. *Optima*, 2000, 1-11.
5. Fleischer, L.K. and Iwata, S.: Improved Algorithms for Submodular Function Minimization and Submodular Flow. Proceedings of the 32th Annual ACM Symposium on Theory of Computing (2000) 107–116.
6. Fujishige, S.: *Submodular Functions and Optimization*. Second Edition. North-Holland (2005).
7. The Ellipsoid Algorithm and its Consequences in Combinatorial Optimization. *Combinatorica*, 1 (1981), 499–513.
8. Grötschel, M. Lovász, L.and Schrijver, A.: *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag. (1988).
9. Iwata, S.. A Faster Scaling Algorithm for Minimizing Submodular Functions. *SIAM J. on Computing* 32 (2002) 833–840.
10. Iwata, S., Fleischer, L., and. Fujishige, S: A Combinatorial, Strongly Polynomial-Time Algorithm for Minimizing Submodular Functions. *J. ACM* 48 (2001) 761–777.
11. Lovász, L.: Submodular Functions and Convexity. In *Mathematical Programming — The State of the Art*, A. Bachem, M. Gr¨otschel, B. Korte eds., Springer, Berlin (1983) 235–257.
12. McCormick, S.T.: Submodular Function Minimization. In *Discrete Optimization*, K. Aardal, G. Nemhauser, and R. Weismantel, eds. Handbooks in Operations Research and Management Science, Volume 12. Elsevier. (2005).
13. Schrijver, A.: A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time. *J. Combin. Theory Ser. B* 80 (2000) 346–355.
14. Schrijver, A.: *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin (2003).
15. Vygen, J.. A Note on Schrijver's Submodular Function Minimization Algorithm. *Journal of Combinatorial Theory B* 88 (2003) 399-402.