A Push-Relabel Framework for Submodular Function Minimization and Applications to Parametric Optimization *

Lisa FLEISCHER † Satoru IWATA ‡

August 16, 2001

Abstract

Recently, the first combinatorial strongly polynomial algorithms for submodular function minimization have been devised independently by Iwata, Fleischer, and Fujishige and by Schrijver. In this paper, we improve the running time of Schrijver's algorithm by designing a push-relabel framework for submodular function minimization (SFM). We also extend this algorithm to carry out parametric minimization for a strong map sequence of submodular functions in the same asymptotic running time as a single SFM. Applications include an efficient algorithm for finding a lexicographically optimal base.

1 Introduction

A function f defined on all the subsets of a finite ground set V is *submodular* if it satisfies for all $X, Y \subseteq V$,

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y).$$

Submodular functions arise in combinatorial optimization and various other fields. Examples include cut capacity functions and matroid rank functions. *Submodular function minimization* (SFM) is the problem of finding a subset $X \subseteq V$ with $f(X) \leq f(Y)$ for all $Y \subseteq V$. The first (strongly) polynomial-time algorithms for SFM were introduced by Grötschel, Lovász, and Schrijver [10, 11]. These algorithms use the ellipsoid method.

Only recently, the first combinatorial polynomial-time algorithms were developed by Iwata, Fleischer, and Fujishige [13] and by Schrijver [17]. These algorithms build on Cunningham's work to design a combinatorial storngly polynomial algorithm for testing membership in matroid polyhedra as well as a combinatorial pseudopolynomial-time[1] algorithm for general SFM [1, 2]. Iwata, Fleischer, and Fujishige [13] design both weakly and strongly polynomial algorithms employing scaling techniques used in the design of algorithms for minimum cost submodular flow [5, 12, 14]. Schrijver [17] describes a combinatorial strongly polynomial algorithm that builds more directly

*Part of this work was done at the Fields Institute, University of Toronto, Toronto, Ontario, Canada. A part of this work has appeared in *Proceedings of the 32nd ACM Symposium on Theory of Computing* (2000), 107–116.

†Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Partially supported by NSF through grants EIA-0049084 and CCR-0049071. Email: `lkf@andrew.cmu.edu`.

‡Department of Mathematical Engineering and Information Physics, Graduate School of Engineering, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan. Email: `iwata@sr3.t.u-tokyo.ac.jp`.

[1]An algorithm is said to run in *pseudopolynomial time* if its run time depends polynomially on the absolute value of the largest number appearing in the input. In the case of SFM, this is the absolute value of the largest number used to represent a function value.

on Cunningham's algorithm [1] for testing membership in matroid polyhedra. The algorithms in [13] have worst-case complexity $O(n^5\gamma \log M)$ and $O(n^7\gamma \log n)$, where $n = |V|$ and M denotes the maximum absolute value of the function values, and γ denotes the time for one function evaluation, i.e., the time to determine $f(X)$ given X. The algorithm in [17] runs in $O(n^8\gamma + n^9)$ time.

In this paper, we present an improved version of Schrijver's algorithm. Schrijver's algorithm depends on the use of his novel subroutine Reduce-Interval described in the next section. It uses this subroutine in a lexicographic framework, using a layered network in a manner similar to augmenting path algorithms of Tardos, Tovey, and Trick [18]. We design a simpler push-relabel framework for SFM that reduces the number of subroutine calls by a factor of n. The resulting algorithm runs in $O(n^7\gamma + n^8)$ time.

The push-relabel framework was introduced by Goldberg and Tarjan [9] for the maximum flow problem. Subsequently, it was applied to polymatroid intersection by Fujishige and Zhang [7]. Gallo, Grigoriadis, and Tarjan [8] extended the push-relabel algorithm to solve monotone parametric maximum flow problems with no increase in time complexity. Iwata, Murota, and Shigeno [15] discussed an extension of the result in [8] to polymatroid intersection. They showed that a strong map sequence of submodular functions plays a similar role to that of a monotone parametric network. Analogously, we extend the push-relabel algorithm for SFM to solve the parametric minimization problem for a strong map sequence.

We then show how to use the parametric submodular function minimization algorithm to solve related problems in the same asymptotic time as solving SFM via the push-relabel algorithm: minimizing $f(X)/w(X)$ for a positive vector w, and finding a lexicographically optimal base. The concept of a lexicographically optimal base was introduced by Fujishige [6] as a generalization of the lexicographically optimal flow earlier defined by Megiddo [16].

Before the existence of combinatorial polynomial-time algorithms for general SFM, it was common to present algorithms that required minimizing submodular functions as having access to an oracle. This was done for two reasons: 1) the ellipsoid method was viewed as a tool for proving polynomial solvability rather than a practically efficient algorithm; 2) for specific submodular functions, there may be efficient specialized algorithms for minimization.

In this manner, Fujishige [6] devised an algorithm for finding the lexicographically optimal base that uses $O(n)$ calls to an oracle for SFM and an algorithm for finding a lexicographically optimal flow by solving $O(k)$ maximum flow problems, where k designates the number of terminal vertices. Gallo, Grigoriadis, and Tarjan [8] described how to find a lexicographically optimal flow in the same asymptotic running time as a single maximum flow computation using a push-relabel algorithm.

In our paper, we show how to exploit the structure of combinatorial algorithm for submodular function minimization to find a lexicographically optimal base in $O(n^7\gamma + n^8)$ time, the same asymptotic time as our improved version of Schrijver's algorithm for SFM. This improves by a factor of n the analysis of the algorithm of Fujishige [6] obtained by simply using our algorithm for SFM as the oracle.

Notation and Definitions

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 $|V| = n$. For a vector in $x \in \mathbf{R}^V$ and a set $X \subseteq V$ we define $x(X) = \sum_{v \in X} x(v)$. For each $u \in V$, we denote by χ_u the unit vector in $\mathbf{R}^V$ such that $\chi_u(v) = 1$ if $v = u$ and $\chi_u(v) = 0$ if $v \neq u$.

Throughout this paper, we assume without loss of generality that $f(\emptyset) = 0$. With such a submodular function f, we associate the *base polyhedron* $\mathrm{B}(f)$ defined by

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

A vector $x \in \mathrm{B}(f)$ is called a *base*. For a base $x \in \mathrm{B}(f)$, a set $X \subseteq V$ is called *x-tight* if $x(X) = f(X)$ holds. An extreme base is an extreme point of $\mathrm{B}(f)$. Given a total order $\preceq$ in V, the greedy algorithm [4] produces an extreme base y by setting $y(v) = f(L(v)) - f(L(v)\backslash\{v\})$ for each $v \in V$, where $L(v) = \{u \mid u \in V, u \preceq v\}$. Note that this implies $L(v)$ is y-tight for each v.

Let I be a set of indices for total orders in V. For $i \in I$, we denote by y_i the extreme base generated by $\preceq_i$ via the greedy algorithm. For $s, t \in V$ and $i \in I$, let $(s,t]_i := \{v \mid s \prec_i v \preceq_i t\}$ be the *interval* between s and t in $\preceq_i$. Note that $(s,t]_i$ may be empty. For $r \in (s,t]_i$ define $\preceq_i^{s,r}$ to be the total order obtained from $\preceq_i$ by moving r to just before s. That is, $r \preceq_i^{s,r} v$ if $s \preceq_i v \prec_i r$, and all other relations remain as with $\preceq_i$. We denote by $y_i^{s,r}$ the extreme base generated by $\preceq_i^{s,r}$ via the greedy algorithm.

2 Submodular Function Minimization

2.1 Schrijver's Algorithm

The combinatorial algorithms for submodular function minimization are based on a dual characterization of a minimizer. For $x \in \mathbf{R}^V$ define x^- by $x^-(v) := \min\{0, x(v)\}$ for $v \in V$. A theorem of Edmonds [4] on vector reduction of polymatroids implies

$$\max\{x^-(V) \mid x \in \mathrm{B}(f)\} = \min\{f(X) \mid X \subseteq V\}. \tag{1}$$

It is important to note that a maximizer of the left hand side might not be an extreme base. If $x \in \mathrm{B}(f)$ is extreme, it is easy to show this: simply exhibit the total order that generates x via the greedy algorithm. However, if x is not extreme, the problem of verifying $x \in \mathrm{B}(f)$ is the problem of determining if $f - x \geq \mathbf{0}$. Unfortunately, no efficient algorithm is known to do this without relying on general SFM. [2] To avoid this problem, Cunningham [2] maintains a representation of a base $x \in \mathrm{B}(f)$ as a convex combination of extreme bases: $x = \sum_{i\in I} \lambda_i y_i$, $\lambda_i \geq 0$, $\sum_{i\in I} \lambda_i = 1$. All of the subsequent algorithms for general SFM do the same.

Since the greedy algorithm returns a base in $\mathrm{B}(f)$, a natural idea is to start from such a base and "move towards" a maximizer of the left hand side of (1). One way to move from one base x to another base x' is to increase x for an element v and decrease x for an element u by the same amount. To determine a maximum feasible step size $\alpha(x,v,u)$ so that the new vector $x' = x + \alpha(x,v,u)(\chi_v - \chi_u)$ is in $\mathrm{B}(f)$ is to determine the minimum value of $f(X) - x(X)$ over all sets $v \in X \subseteq V\backslash\{u\}$. This quantity is called the *exchange capacity*, and computing it is again a problem of minimizing a submodular function. [3]

To avoid the obstacle of computing exact exchange capacities, Schrijver proposes a method of computing lower bounds on exchange capacities, and devises a framework in which performing such exchanges leads to a strongly polynomial algorithm for SFM [17]. Given $x = \sum_{i\in I} \lambda_i y_i$ and

[2] Given an algorithm that solves the membership problem, it is possible to find the minimum of an arbitrary submodular function f via the binary search method. By asking whether $0 \in \mathrm{B}(f^\beta)$ where the function is defined as $f^\beta(X) = f(X) - \beta$ for nonempty X, it is possible to determine if the minimum value of f is less than or greater than $\beta \leq 0$. Good upper and lower bounds on the minimum value of f may be obtained from any base $x \in \mathrm{B}(f)$.

[3] Let β be a lower bound of an arbitrary submodular function $f : 2^V \to \mathbf{R}$. Define $f' : 2^U \to \mathbf{R}$ on $U = V \cup \{u, v\}$ by $f'(X) = f(X \cap V) - \beta$ for $\emptyset \neq X \subset U$ and $f'(\emptyset) = f'(U) = 0$. Then f' is submodular and $0 \in \mathrm{B}(f)$. Computing $\alpha(0, v, u)$ is equivalent to finding the minimum value of f.

a pair (s,t) with $(s,t]_i \neq \emptyset$ for some $i \in I$, Schrijver introduces a subroutine that moves from x while reducing either $\max_{i\in I} |(s,t]_i|$ or the number of total indices $i \in I$ that attain this maximum. By maintaining an affinely independent representation of x, after at most n^2 such applications, $(s,t]_i = \emptyset$ for all $i \in I$, and thus the base x in this last iteration satisfies $\alpha(x,s,t) = 0$.

Schrijver's subroutine is now described as follows.

> Reduce-Interval(i,s,t)
>
> **Input:** A total order $\preceq_i$ and $s,t \in V$ such that $s \prec_i t$.
>
> **Output:** A constant $\mu \geq 0$ and a decomposition of $y_i + \mu(\chi_t - \chi_s)$ as a convex combination of $y_i^{s,r}$ for $r \in (s,t]_i$.

Suppose Reduce-Interval(i,s,t) returns μ and $\sum_{u\in(s,t]_i} \mu_r y_i^{s,r}$. The total orders $\preceq_i^{s,r}$ generated by the subroutine satisfy two properties:

- For each $r \in (s,t]_i$, the set $(s,t]_i^{s,r} := \{v \mid s \prec_i^{s,r} v \preceq_i^{s,r} t\}$ is strictly contained in $(s,t]_i$.
- If $\mu > 0$, then $\chi_t - \chi_s = \sum_r \frac{\mu_r}{\mu}(y_i^{s,r} - y_i)$. Otherwise, $y_i^{s,r} = y_i$ for some $r \in (s,t]_i$.

Let i attain the maximum $|(s,t]_i|$ in I. If λ_i is the coefficient of y_i in the convex representation of x then by replacing $\lambda_i y_i$ with $\lambda_i \sum_{r\in(s,t]_i} \mu_r y_i^{s,r}$, the new base $x' = x + \lambda_i \sum_r \mu_r y_i^{s,r} = x + \lambda_i y_i + \lambda_i \mu(\chi_t - \chi_s)$ with new index set of total orders I' has a convex representation with either smaller $\max_{i\in I'} |(s,t]_i|$ or fewer total orders that obtain this value.

It takes $O(n^2\gamma)$ time to determine this expression of $y_i + \mu(\chi_t - \chi_s)$. After this operation, the number of total orders in the expression of the new base x' may have increased by at most $|(s,t]_i| < n$. Gaussian elimination can then be used to reduce the total number of bases to at most n in $O(n^3)$ time. Thus Reduce-Interval takes $O(n^2\gamma + n^3)$ time.

Schrijver [17] describes a modification of a layered network algorithm to find a minimizer of f by calling Reduce-Interval $O(n^6)$ times. Below, we describe how to embed Reduce-Interval in a push-relabel framework to minimize a submodular function with $O(n^5)$ calls to Reduce-Interval. This speedup mirrors the improvement in the run time of push-relabel algorithms over layered network algorithms for the maximum flow algorithm. Besides giving a faster algorithm, the push-relabel framework is more adaptable to generalizations of maximum flow such as parametric maximum flow. We show in Sections 3 and 4 of this paper that this is also the case for submodular function minimization.

2.2 Push-Relabel Framework

Our push-relabel algorithm works on a graph with vertex set V and arc set $A_I := \bigcup_{i\in I} A_i$, where $A_i := \{(s,t) \mid s,t \in V,\ s \prec_i t\}$, and maintains distance labels d on the vertices. Let $P = \{v \mid v \in V,\ x(v) > 0\}$ and $N = \{v \mid v \in V,\ x(v) < 0\}$, where $x = \sum_{i\in I} \lambda_i y_i$.

Definition 2.1 *The labeling $d : V \to \mathbf{Z}$ is* valid *for $x \in \mathrm{B}(f)$ if it satisfies $d(v) = 0$ for $v \in N$ and $d(s) \leq d(t) + 1$ for all $(s,t) \in A_I$.*

The push-relabel algorithm maintains a valid labeling. Initially, $d(s) = 0$ for $s \in V$, which is clearly valid. Note that for a valid distance labeling d, $d(s)$ is a lower bound on the minimum number of arcs from s to N. For a valid labeling d, we define $Q = \{s \mid s \in P,\ d(s) < n\}$.

The push-relabel algorithm for maximum flow maintains a preflow: a flow that satisfies capacity constraints but not conservation. Instead, vertices are allowed to have excess: more flow coming in than going out. The operations push and relabel apply to a vertex with excess. In the setting of submodular function minimization, the algorithm will simply maintain a base x as a convex combination of extreme bases. The operations push and relabel will apply to elements s with $x(s) > 0$.

Operation $\mathsf{Relabel}(s)$ applies if $s \in Q$ and $d(s) \leq d(t)$ for every $(s,t) \in A_I$. It updates $d(s) := d(s) + 1$. If the new $d(s) = n$, then s is removed from Q. Thus, $d(s) \leq n$ holds for $s \in V$ throughout the algorithm.

Operation $\mathsf{Push}(s,t)$ applies if $s \in Q$, $(s,t) \in A_I$ and $d(s) = d(t) + 1$ and results in either $x(s) = 0$ or $(s,t) \notin A_I$. It accomplishes this by repeatedly selecting $i \in I$ with the largest interval $(s,t]_i$, and applying the subroutine $\mathsf{Reduce\text{-}Interval}(i,s,t)$ to get $\mu \geq 0$ and a convex decomposition $\sum_{r \in (s,t]_i} \mu_r y_i^{s,r}$ of $y_i + \mu(\chi_t - \chi_s)$. Then it updates $x := x + \varepsilon(\chi_t - \chi_s)$ with $\varepsilon = \min\{x(s), \lambda_i \mu\}$. If $\mu = 0$, this update does not change x, but $\preceq_i$ is replaced by $\preceq_i^{s,r}$ for some $r \in (s,t]_i$. Otherwise, the new convex combination coefficients are updated as $\lambda_i := \lambda_i - \varepsilon/\mu$, $\lambda_{i_r} := \varepsilon \mu_r / \mu$ for $r \in (s,t]_i$; and the set I is augmented by the indices i_r of those total orders $\preceq_i^{s,r}$ with nonzero coefficients, and the final coefficients of indices i_r that are already in I is obtained by adding the existing coefficient to the coefficient of i_r computed above. By a standard linear programming technique, the I can be reduced to an affinely independent set of at most n members in $O(n^3)$ time. This entire sequence is repeated until $x(s) = 0$ or $(s,t) \notin A_I$, which occurs when $(s,t]_i = \emptyset$ for all $i \in I$. If $(s,t) \notin A_I$, we call $\mathsf{Push}(s,t)$ *saturating*. Otherwise, $\mathsf{Push}(s,t)$ is *nonsaturating*. After each call to the subroutine $\mathsf{Reduce\text{-}Interval}$, the maximum size of the intervals $(s,t]_i$ decreases or the number of total orders that attain the maximum decreases. Thus $\mathsf{Push}(s,t)$ performs at most $O(n^2)$ calls to $\mathsf{Reduce\text{-}Interval}$.

These two operations are used in the algorithm as follows. The algorithm starts by fixing an arbitrary total order $\leq_\circ$ on the vertices. The algorithm repeatedly selects a vertex $s \in Q$ with highest $d(s)$ to apply a procedure $\mathsf{Scan}(s)$. The goal of $\mathsf{Scan}(s)$ is to either obtain $x(s) = 0$, or certify that no Push operation is applicable for s, in which case a relabel operation is applicable. The procedure $\mathsf{Scan}(s)$ repeatedly picks a vertex $t \in V$ in order of $\leq_\circ$ and applies $\mathsf{Push}(s,t)$ if possible, until $x(s) = 0$ or it has examined every $t \in V$. If $\mathsf{Scan}(s)$ ends with a non-saturating $\mathsf{Push}(s,t)$, the next time $\mathsf{Scan}(s)$ is invoked, it starts at t. This is done by keeping a pointer $\tau(s)$ that indicates the current vertex to be examined in $\mathsf{Scan}(s)$ for each $s \in V$. The algorithm increments $\tau(s)$ if it performs a saturating $\mathsf{Push}(s,\tau(s))$ or it finds $\mathsf{Push}(s,\tau(s))$ is not applicable. If $\tau(s)$ is the last vertex in $\leq_\circ$, this invocation of $\mathsf{Scan}(s)$ ends and the algorithm performs $\mathsf{Relabel}(s)$ and resets $\tau(s)$ to be the first vertex in $\leq_\circ$.

The algorithm terminates when either Q or N is empty. This algorithm is summarized in Figure 1.

Correctness and Complexity

Lemma 2.2 *The operations* Push *and* $\mathsf{Relabel}$ *maintain d valid.*

Proof. At the start d is valid. The operation $\mathsf{Relabel}$, if applicable, maintains that d is valid. Suppose d is valid before $\mathsf{Reduce\text{-}Interval}(i,s,t)$ that introduces a new arc (u,v) to A_I. If (u,v) is a new arc, it is not in A_i but is in A_{i_u} for some $u \in (s,t]_i$. Thus, $s \preceq_i v \preceq_i u \preceq_i t$, which implies that $d(u) \leq d(t) + 1 = d(s) \leq d(v) + 1$, where the equality follows by choice of (s,t). Thus d remains valid after a Push operation. ■

```
Push-Relabel(f, V):

Fix a total order ≤∘ on V, so V = {1, 2, ..., n}.
x ∈ B(f) obtained by greedy algorithm.
d(v) = 0 for all v ∈ V.
τ(v) = 1 for all v ∈ V.
P := {v | v ∈ V, x(v) > 0};
N := {v | v ∈ V, x(v) < 0};
Q := P;
While Q ≠ ∅ and N ≠ ∅,
    s ← element in Q with max d(s).
      [ Start Scan(s). ]
    While x(s) > 0 and τ(s) ≤∘ n,
        If d(s) = d(τ(s)) + 1 then
            Push(s, τ(s)).
              [ Updates Q, P, N, x ]
        τ(s) := τ(s) + 1;
      [ End Scan(s). ]
    If x(s) ≠ 0,
        Relabel(s).
          [ Updates Q ]
        τ(s) := 1.
    else Q ← Q\{s}.
W ← the set of vertices from which N is reachable.
Return W.
```

Figure 1: Description of a push-relabel algorithm for finding a minimizer of a submodular function.

Lemma 2.3 *At termination, the set W of vertices from which there is a directed path to N is a minimizer of f.*

Proof. If $N \neq \emptyset$, then $x(v) \leq 0$ for $v \in W$ and $x(v) \geq 0$ for $v \in V \backslash W$. This implies $x^-(V) = x(W)$. Since no arc in A_I enters W, the set W is y_i-tight for each $i \in I$, which implies $x(W) = f(W)$. Thus by (1), the set W is a minimizer of f.

If $N = \emptyset$, then $f(X) \geq x(X) \geq 0$ holds for every $X \subseteq V$, which implies that $\emptyset$ is a minimizer of f. ∎

By the same argument as in Proof of Lemma 2.3, any subset X with $N \subseteq X \subseteq V \backslash P$ such that there is no arc from $V \backslash X$ to X in A_I is a minimizer of f.

Since the algorithm never relabels a vertex s with $d(s) = n$, $d(s) \leq n$ for every $s \in V$. Thus, the algorithm performs at most n^2 relabel operations in total. The following sequence of lemmas bounds the number of push operations.

Lemma 2.4 Relabel(u) *is applicable when the algorithm resets* $\tau(u)$ *in* Scan(u).

Proof. It suffices to establish that when the algorithm resets $\tau(u)$, that there is no arc (u, v) in A_I with $d(v) < d(u)$. We do this by showing that $v <_\circ \tau(u)$ and $(u, v) \in A_I$ imply that $d(u) \leq d(v)$. Suppose by induction that the statement holds before Reduce-Interval(i, s, t) is applied when $(u, v) \notin A_I$. Suppose Reduce-Interval(i, s, t) introduces (u, v) into A_I. From the proof of Lemma 2.2, we have that $d(u) \leq d(t) + 1 = d(s) \leq d(v) + 1$. Now, if $t <_\circ \tau(u)$, then the first inequality can be tightened to be $d(u) \le d(t)$. On the other hand, if $v <_\circ \tau(u) \le_\circ t$, then $d(s) \le d(v)$. In either case, we have $d(u) \le d(v)$. ■

Corollary 2.5 *The algorithm performs at most n^3 saturating pushes.*

Proof. After a saturating $\mathsf{Push}(s,t)$, $\tau(s)$ is incremented by 1. Thus there are at most n saturating pushes before s is relabeled. Since no label exceeds n and there are at most n^2 saturating pushes per element, there are at most n^3 saturating pushes in total. ■

Lemma 2.6 *Between a non-saturating* $\mathsf{Push}(s,t)$ *and the next* $\mathsf{Scan}(s)$, *the algorithm performs* $\mathsf{Relabel}(u)$ *for some $u \in V$.*

Proof. As a consequence of a non-saturating $\mathsf{Push}(s,t)$, we have $x(s) = 0$. Before applying $\mathsf{Scan}(s)$ again, the algorithm must increase $x(s)$ via a $\mathsf{Push}(v,s)$ for some $v \in V$ with $d(v) = d(s) + 1$. This implies by the highest label selection rule that there must be a relabel operation some time before $\mathsf{Push}(v,s)$ is invoked. ■

Corollary 2.7 *The number of non-saturating pushes is at most n^3.*

Proof. Since there are at most n^2 relabel operations over the course of the algorithm, the number of times $\mathsf{Push}(s,t)$ is non-saturating for s is at most n^2. Over all vertices, this implies at most n^3 nonsaturating pushes. ■

Thus the algorithm performs $O(n^2)$ relabel and $O(n^3)$ push operations. Since each push operation calls Reduce-Interval $O(n^2)$ times, Reduce-Interval is invoked $O(n^5)$ times in total. Therefore, the push-relabel algorithm runs in $O(n^7\gamma + n^8)$ time.

In the above algorithm, we could reverse the direction of arcs in A_I and replace the roles of P and N with each other. In this case, a push operation is performed from a vertex s with negative $x(s)$. This algorithm ends if $P = \emptyset$ or $Q = \emptyset$. If $P = \emptyset$, we have $x^-(V) = x(V) = f(V)$, which implies V is a minimizer of f. Otherwise, the set W of vertices reachable from N by the arcs in A_I is a minimizer of f. We call this variant Reverse-Push-Relabel.

3 Parametric Submodular Function Minimization

Gallo, Grigoriadis, and Tarjan [8] modify the maximum flow push-relabel algorithm of Goldberg and Tarjan [9] to solve a parametric network flow problem. They consider a flow network with arc capacities c_θ that are functions of a parameter θ: For arc a leaving the source, $c_\theta(a)$ is increasing in θ; for a entering the sink, $c_\theta(a)$ is decreasing in θ; all other arcs have constant capacities. This is called a *monotone* parametric network. They show that for a sequence of parameter values $\theta_1 < \theta_2 < \cdots < \theta_k$, the minimum cuts and maximum flows can be computed for all values in the same asymptotic time as one push-relabel maximum flow computation.

In the setting of submodular functions, we consider a generalization of this special parametric flow problem. A submodular function $\widehat{f}$ is said to be a *strong quotient* of f if $Z \supseteq Y$ implies

$$f(Z) - f(Y) \ge \widehat{f}(Z) - \widehat{f}(Y) \tag{2}$$

for $Y, Z \subseteq V$. We denote this relation by $f \to \widehat{f}$, and say that the relation $f \to \widehat{f}$ is a strong map.

Lemma 3.1 (Topkis [19]) *If $f \to \widehat{f}$ then the minimal (maximal) minimizer of f is contained in the minimal (maximal) minimizer of $\widehat{f}$.*

To show that the parametric flow problem is indeed a special case of strong maps, consider any fixed value θ of the parameter and denote by $\delta(X)$ the set of arcs leaving X. The cut function κ_θ defined on subsets of $V \backslash \{s,t\}$ by $\kappa_\theta(A) = c_\theta(\delta(A \cup \{s\}))$ is a submodular function. For $\theta_1 < \theta_2$, it is easy to check that $\kappa_{\theta_1} \to \kappa_{\theta_2}$.

Another special case of strong map sequence is the set of functions obtained from a submodular function f and a nonnegative vector $w \in \mathbf{R}^V$: the set of functions $\{f + \theta_\ell w\}$ for an increasing sequence of θ_h. Then $f + \theta_{\ell+1} w \to f + \theta_\ell w$.

We show that the minimizer of all submodular functions in a strong map sequence $f_1 \to f_2 \to \cdots \to f_k$ can be found in the same asymptotic time as a single submodular function minimization using the push-relabel algorithm.

The algorithm consists of k iterations. Iteration ℓ finds a minimizer of f_ℓ. The first iteration starts with a valid labeling $d(s) = 0$ for $s \in V$ and applies the push-relabel algorithm for SFM until it terminates. Each subsequent iteration starts with the final distance labeling from the previous iteration and an appropriately defined base $x \in \mathrm{B}(f_\ell)$ such that the current labeling d is valid with respect to x. It resumes the push-relabel algorithm with these inputs.

To obtain the initial base $x \in \mathrm{B}(f_\ell)$ in iteration ℓ, let $\widehat{x} = \sum_{i \in I} \lambda_i \widehat{y}_i$ be the convex combination of extreme bases in $\mathrm{B}(f_{\ell-1})$ obtained at the end of the previous push/relabel iteration. For each of the bases $\widehat{y}_i \in \mathrm{B}(f_{\ell-1})$, we generate a base $y_i \in \mathrm{B}(f_\ell)$ and set $x = \lambda_i y_i$.

While the extreme bases in the convex combination have changed from $\widehat{y}$ to y, the total orders generating them have not changed. Thus $s \prec_i t$ still implies that $d(s) \leq d(t) + 1$. Furthermore, Lemma 3.2 below implies that $y_i \geq \widehat{y}_i$. Thus, $x = \sum_{i \in I} \lambda_i y_i \geq \sum_{i \in I} \lambda_i \widehat{y}_i = \widehat{x}$ so that $x(v) < 0$ implies $\widehat{x}(v) < 0$ which implies $d(v) = 0$. Thus we have that d is a valid labeling with respect to x.

Lemma 3.2 ([15]) *Let y_i and $\widehat{y}_i$ denote respectively the extreme points of $\mathrm{B}(f)$ and $\mathrm{B}(\widehat{f})$ obtained by applying the greedy algorithm to $\preceq_i$. Then $f \to \widehat{f}$ if and only if $y_i \geq \widehat{y}_i$ for every total order $\preceq_i$ of V.* ∎

We now discuss the time complexity of the algorithm. Since the validity of d implies $d(s) \leq n$ for every $s \in V$, the total number of the relabel operations is at most n^2. The total number of push operations is $O(n^3)$. To generate the initial base $x \in \mathrm{B}(f_\ell)$ in iteration ℓ, we apply greedy $|I| \leq n$ times to generate $|I| \leq n$ extreme bases. This takes a total of $O(n^2)$ arithmetic steps and function evaluations per function in the strong map sequence. Therefore the algorithm requires $O(n^7 + kn^2)$ oracle calls and $O(n^8)$ additional arithmetic steps. Thus the algorithm runs within the same time complexity as the push/relabel algorithm for a single submodular function minimization as long as $k = O(n^5)$.

Note that this algorithm can be run even if the f_i are obtained in an on-line manner during the course of the algorithm. We will use this fact in the following section.

Finally, we may be interested in computing the minimizers in the opposite order, i.e. first the minimizer of f_k, then of f_{k-1}, etc. To do this, we need to invoke Reverse-Push-Relabel. When moving from one function to the next, we obtain the new base in the *same* way as before. To check the labeling is valid, we are just in the opposite case of showing the new base $x \leq \widehat{x}$. But this holds by Lemma 3.2 since the new function f is a strong quotient of the old function $\widehat{f}$, i.e. $\widehat{f} \to f$.

4 Applications

Finding a Weighted Minimizer

One application is finding the minimizer of $f(X)/w(X)$ for a positive vector w. To do this, we seek the smallest value of α such that there is a set X with $f(X) = \alpha w(X)$. Such α can be computed by Dinkelbach's [3] discrete Newton method as follows.

Start with $\alpha := f(V)/w(V)$, which serves as an upper bound. Find a minimizer Y of $f - \alpha w$. If $f(Y) - \alpha w(Y) \geq 0$, then the current α is the optimal value, since decreasing α will only make $f - \alpha w$ more positive. Otherwise, the set Y is strictly contained in V and we update $\alpha := f(Y)/w(Y)$, which gives an improved upper bound. Repeating this, we will eventually obtain the optimal α. Since $\alpha \leq \widehat{\alpha}$ implies $(f - \alpha w) \rightarrow (f - \widehat{\alpha} w)$ we may apply the algorithm for strong map submodular functions, and thus solve the problem in the same asymptotic time as a single push-relabel submodular function minimization. By Lemma 3.1, the number of α visited by the algorithm is at most n.

Finding a Lexicographically Optimal Base

Another related application is finding a lexicographically optimal base. This concept was first introduced by Megiddo [16] for multi-terminal network flow. Fujishige [6] generalized it to the framework of polymatroids. Let $w \in \mathbf{R}^V$ be a weight vector satisfying $w(v) > 0$ for all $v \in V$. For any base $x \in \mathrm{B}(f)$, we denote by $\theta(x, w)$ the sequence of the numbers $x(v)/w(v)$ for $v \in V$ arranged in the increasing order. A base x^* is said to be lexicographically optimal w.r.t. w if $\theta(x^*, w)$ is lexicographically maximum among all the bases in $\mathrm{B}(f)$. A lexicographically maximum base may not be an extreme base. Fujishige [6] showed the uniqueness of the lexicographically optimal base and described algorithms to find it.

For any base $x \in \mathrm{B}(f)$, let $\xi_1 < \cdots < \xi_\ell$ denote the distinct values of $x(v)/w(v)$ for $v \in V$, and put $H_j = \{v \mid x(v) \leq \xi_j w(v)\}$. Fujishige [6] proved that x is the lexicographically optimal base if and only if $x(H_j) = f(H_j)$ holds for $j = 1, \ldots, \ell$. Therefore, if x is lexicographically optimal and $\xi_j \leq \alpha < \xi_{j+1}$, we have $f(X) - \alpha w(X) \geq x(X) - \alpha w(X) \geq x(H_j) - \alpha w(H_j) = f(H_j) - \alpha w(H_j)$ for any $X \subseteq V$. This suggests finding an appropriate H_i as a minimizer of $f - \alpha w$ for some parameter α. In fact, Fujishige [6] presented the following recursive algorithm for computing the lexicographically optimal base.

Minimize the submodular function $f - \alpha w$ for $\alpha := f(V)/w(V)$. If $f - \alpha w \geq 0$, then $x^*(v) := \alpha w(v)$ for each $v \in V$. In this case, x^* is the lexicographically optimal base since any base x must satisfy $x(V) = \alpha w(V)$. Otherwise, let W be the unique minimal minimizer of $f - \alpha w$. The minimal minimizer is the set of elements that can reach N in A_{x^*}. Let f^W be the *restriction* of f with respect to W defined by $f^W(X) := f(X)$ for $X \subseteq W$. Let f_W denote the *contraction* of f with respect to W defined by $f_W(X) := f(W \cup X) - f(W)$ for $X \subseteq V \backslash W$. Compute the lexicographically optimal base x^W of f^W with respect to w and the lexicographically optimal base x_W of f_W with respect to w. Then $x^* := x^W \oplus x_W$ defined by $x^*(v) := x^W(v)$ for $v \in W$ and $x^*(v) := x_W(v)$ for $v \in V \backslash W$ is the lexicographically optimal base of f.

We now discuss an efficient implementation of this recursive algorithm, similar to the lexicographically optimal flow algorithm of Gallo, Grigoriadis, and Tarjan [8]. Let α_1 be a sufficiently small number such that $f - \alpha_1 w$ is nonnegative. For instance, select $\alpha_1 = \min\{y(v)/w(v) \mid v \in V\}$ for some $y \in \mathrm{B}(f)$. Apply the reverse push/relabel algorithm to $f - \alpha_1 w$ to obtain a base x and valid labeling d_1. Similarly, let α_3 be a sufficiently large number such that V minimizes $f - \alpha_3 w$. For instance, select $\alpha_3 = \max\{y(v)/w(v) \mid v \in V\}$ for some $y \in \mathrm{B}(f)$. Apply the ordinary push/relabel algorithm to obtain a base x_3 and valid labeling d_3. We then perform the following recursive procedure $\mathsf{Slice}(f, \alpha_1, \alpha_3, x_1, x_3, d_1, d_3)$.

In the procedure $\mathsf{Slice}(f, \alpha_1, \alpha_3, x_1, x_3, d_1, d_3)$, we compute the unique minimal minimizer W of $f - \alpha_2 w$ for $\alpha_2 = f(V)/w(V)$ by applying both ordinary and reverse push/relabel algorithms. The ordinary one starts with $x := x_3 + (\alpha_3 - \alpha_2)w$ and d_3, while the reverse one starts with $x := x_1 - (\alpha_2 - \alpha_1)w$ and d_1. We concurrently execute these algorithms and stop when one of them terminates. Suppose the ordinary one terminates the first with a base x_2 and valid labeling d_2. (The other case is symmetric.) If $W = \emptyset$, then return $x^*(v) = \alpha_2 w(v)$ for each v. If $|W| > n/2$, then compute x^W and x_W by applying respectively $\mathsf{Slice}(f^W, \alpha_1, \alpha_2, x_1, x_2, d_1, d_2)$ and $\mathsf{Slice}(f_W, \alpha_2, \alpha_3, x_2, x_3, d_0, d_3)$, where $d_0(v) = 0$ for each v. If $|W| \leq n/2$, continue the reverse algorithm to replace x_2 and d_2 by the resulting ones, and then apply $\mathsf{Slice}(f^W, \alpha_1, \alpha_2, x_1, x_2, d_1, d_0)$ for finding x^W and $\mathsf{Slice}(f_W, \alpha_2, \alpha_3, x_2, x_3, d_2, d_3)$ for finding x_W. The lexicographically optimal base is obtained by $x^* := x^W \oplus x_W$.

When we divide the ground set into W and its complement, we introduce new labelings d_0 for at most half of the ground set. Therefore, the entire algorithm deals with at most $2n$ labelings, and hence it performs $O(n^2)$ relabel operations in total. Thus the algorithm finds the lexicographically optimal base in $O(n^7\gamma + n^8)$ time. This is the same as the running time of our push/relabel algorithm for SFM, whereas the previous best algorithm due to Fujishige [6] requires $O(n)$ calls to an oracle for SFM.

Acknowledgement

We are grateful to Tom McCormick for useful comments and the referees for suggestions which improved the presentation of this paper.

References

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

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

[3] W. Dinkelbach, On nonlinear fractional programming, *Management Sci.*, 13 (1967), 492–498.

[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, 1970), 69–87.

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

[6] S. Fujishige, Lexicographically optimal base of a polymatroid with respect to a weight vector, *Math. Oper. Res.*, 5 (1980), 186–196.

[7] S. Fujishige and X. Zhang, New algorithms for the intersection problem of submodular systems, *Japan J. Indust. Appl. Math.*, 9 (1992), 369–382.

[8] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, A fast parametric maximum flow algorithm and applications, *SIAM J. Comput.*, 18 (1989), 30–55.

[9] A. V. Goldberg and R. E. Tarjan, A new approach to the maximum flow problem, *J. ACM*, 35 (1988), 921–940.

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

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

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

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

[14] S. Iwata, S. T. McCormick, and M. Shigeno, A strongly polynomial cut canceling algorithm for the submodular flow problem, *Proceedings of the Seventh MPS Conference on Integer Programming and Combinatorial Optimization* (1999), 259–272.

[15] S. Iwata, K. Murota, and M. Shigeno, A fast parametric submodular intersection algorithm for strong map sequences, *Math. Oper. Res.*, 22 (1997), 803–813.

[16] N. Megiddo, Optimal flows in networks with multiple sources and sinks, *Math. Programming*, 7 (1974), 97–107.

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

[18] É. Tardos, C. A. Tovey, and M. A. Trick, Layered augmenting path algorithms. *Math. Oper. Res.*, 11 (1986), 362–370.

[19] D. M. Topkis, Minimizing a submodular function on a lattice, *Oper. Res.*, 26 (1978), 305–321.

This research was sponsored in part by National Science Foundation (NSF) grant no. CCR-0122581.