A Faster Capacity Scaling Algorithm for Minimum Cost Submodular Flow

Lisa Fleischer [*] Satoru Iwata [†] S. Thomas McCormick [‡]

August 1999; July 2001

Abstract

We describe an $O(n^4 h \min\{\log U, n^2 \log n\})$ capacity scaling algorithm for the minimum cost submodular flow problem. Our algorithm modifies and extends the Edmonds–Karp capacity scaling algorithm for minimum cost flow to solve the minimum cost submodular flow problem. The modification entails scaling a relaxation parameter δ. Capacities are relaxed by attaching a complete directed graph with uniform arc capacity δ in each scaling phase. We then modify a feasible submodular flow by relaxing the submodular constraints, so that complementary slackness is satisfied. This creates discrepancies between the boundary of the flow and the base polyhedron of a relaxed submodular function. To reduce these discrepancies, we use a variant of the successive shortest path algorithm that augments flow along minimum cost paths of residual capacity at least δ. The shortest augmenting path subroutine we use is a variant of Dijkstra's algorithm modified to handle exchange capacity arcs efficiently. The result is a weakly polynomial time algorithm whose running time is better than any existing submodular flow algorithm when U is small and C is big. We also show how to use maximum mean cuts to make the algorithm strongly polynomial. The resulting algorithm is the first capacity scaling algorithm to match the current best strongly polynomial bound for submodular flow.

[*]Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Email: `lkf@andrew.cmu.edu`. This research was done in part while visiting RIMS, Kyoto University, with support by Grant-in-Aid for Scientific Research No. 10750053 from Ministry of Education, Science, Sports, and Culture of Japan, and while on leave at CORE, Université catholique de Louvain, Belgium. Additional support from NSF through grant EIA-0049084.

[†]Department of Mathematical Engineering and Information Physics, University of Tokyo, Tokyo 113-8656, Japan. Email: `iwata@sr3.t.u-tokyo.ac.jp`. Partially supported by Grant-in-Aid for Scientific Research No. 10750053 from Ministry of Education, Science, Sports, and Culture of Japan.

[‡]Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, BC V6T 1Z2 Canada. Email: `stmv@adk.commerce.ubc.ca`. Research supported by an NSERC Operating Grant; part of this research was done during visits to SORIE at Cornell University and to LIMOS at Université Blaise Pascal, Clermont-Ferrand.

1 Introduction

The submodular flow problem, introduced by Edmonds and Giles [4], is one of the most important frameworks of efficiently solvable combinatorial optimization problems. It includes minimum cost flow, graph orientation, polymatroid intersection, and directed cut covering problems as special cases. To talk about running time we use n for the number of nodes, h for the time to compute a single exchange capacity (defined below), C for the maximum absolute value of the arc costs, and U for the maximum absolute value of the capacities. When we use C or U we assume that those data are integral.

A number of minimum cost flow algorithms have been extended to solve the submodular flow problem. The first such polynomial algorithm is due to Cunningham and Frank [2]. It generalizes a cost-scaling primal-dual method. A variant of this algorithm, combined with the push/relabel algorithm of Fujishige and Zhang [18], runs in $\mathrm{O}(n^4 h \log C)$ time [24]. This is the previous best weakly polynomial bound for submodular flow. A different cost-scaling approach is based on cycle canceling [22], and achieves a running time of $\mathrm{O}(n^4 h \min\{\log(nC), n^2 \log n\})$. See [15] for a recent survey of submodular flow algorithms.

The dual approach to scaling costs is to instead scale the capacities. The prototype capacity scaling min cost flow algorithm is the Edmonds–Karp [5] algorithm that scales and then rounds the capacities. Generalizing this to submodular flow is difficult because rounding a scaled submodular function does not preserve submodularity. To obviate this difficulty, the algorithm in [19] adds a small multiple of the strictly submodular function b (which is the symmetric cut function of a complete directed graph with unit arc capacity) before rounding. However, the scaling scheme discussed in that paper calls maximum submodular flow algorithms repeatedly, and hence requires $\mathrm{O}(n^7 h \log U)$ time.

An easier way to do capacity scaling for submodular flow was introduced in [23]. There the submodular bounds are again relaxed by a multiple of b, but now only the relaxation parameter is scaled instead of the data. The result was the first strongly polynomial capacity scaling algorithm for submodular flow, but it again calls maximum submodular flow algorithms repeatedly and so runs in $\mathrm{O}(n^6 h \min\{\log(nU), n^2 \log n\})$ time.

In this paper, we modify and extend the Edmonds–Karp capacity scaling algorithm to an algorithm for submodular flow. A key innovation of our algorithm is a modification of Dijkstra's algorithm to efficiently handle exchange capacity arcs inherent in submodular flow. As in [19], capacities are relaxed by attaching a complete directed graph with uniform arc capacity δ in each scaling phase. As in [23], we relax the problem by a parameter δ, which is scaled. We then modify our current feasible submodular flow so that complementary slackness is satisfied. This creates discrepancies between the boundary of the flow and a base in the base polyhedron. To reduce these discrepancies, we use a variant of the successive shortest path algorithm that augments flow along min cost paths of residual capacity at least δ. Our shortest path algorithm is further specialized to dynamically modify flows so that we can handle exchange capacity arcs efficiently. This yields a weakly polynomial algorithm with a new run time, $\mathrm{O}(n^4 h \log U)$, that is faster than all other known submodular flow algorithms when U is smaller than C and $n^{\mathrm{O}(n^2)}$.

As in [23] we also show how to use (approximations of) maximum mean cuts to get a strongly polynomial version of the algorithm. The resulting algorithm runs in $\mathrm{O}(n^6 h \log n)$ time, matching the best current strongly polynomial bound. We also show how our algorithms can be extended to work for the case of crossing submodular functions.

The first strongly polynomial algorithm for the submodular flow problem is due to Frank and Tardos [9]. This is an application of simultaneous Diophantine approximation and substantially generalizes the first strongly polynomial minimum cost flow algorithm of Tardos [33]. A more direct generalization of the Tardos algorithm to the submodular flow problem is described by Fujishige, Röck, and Zimmermann [16] with the aid of the tree-projection method of Fujishige [13]. The latter algorithm combined with the improved primal-dual algorithm in [24] runs in $O(n^6 h \log n)$ time, the current best strongly polynomial bound, which is also achieved by the lexicographic cycle canceling algorithm in [22].

Our algorithm depends crucially on the sophisticated new technique of modifying Dijkstra's algorithm to deal with the exchange arcs inherent in submodular flow, while ignoring small residual capacity arcs. This novel technique is further extended in [21] to obtain the first combinatorial, polynomial-time algorithm for the fundamental problem of minimizing a submodular function, and used again in [20] to get the first fully combinatorial algorithm for the same problem.

In the same way that cost scaling is associated with cycle canceling, capacity scaling is associated with cut canceling (see [32] for a survey of this for min cost flow). Although our algorithm was developed as an extension of the Edmonds–Karp algorithm, it is also possible to see it as a way of speeding up the cut canceling algorithm for submodular flow described in [23]. That is, it is well-known that updating dual variables via shortest path distances from Dijkstra's algorithm can be seen as simultaneously canceling several cuts at once. From this perspective, our new algorithm is a much faster version of the algorithm of [23], thus answering the open question of finding such an algorithm. Thus the new algorithm merges the two seemingly different strands of research of [19] and [23].

2 Preliminaries on Submodular Functions

In this section, we review some well-known facts about submodular functions, base polyhedra, and exchange capacities. We re-prove many of the results for completeness, and cite the relevant results in the monograph by Fujishige [14] for readers interested in more details.

2.1 Base Polyhedra

Let V be a finite set, and $\mathcal{D} \subseteq 2^V$ be a distributive lattice with $\emptyset, V \in \mathcal{D}$. A function $f : \mathcal{D} \to \mathbf{R}$ is *submodular* if

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \tag{1}$$

holds for all $X, Y \in \mathcal{D}$. We assume throughout the paper that $f(\emptyset) = 0$. Then the *base polyhedron* $\mathrm{B}(f)$ of f is defined by

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

and $x \in \mathrm{B}(f)$ is called a *base*. For $p \in \mathbf{R}^V$, consider the linear program to maximize $\sum_{v \in V} p(v)x(v)$ on the base polyhedron $\mathrm{B}(f)$. An optimal solution is called a *p-maximum base*. Let $p_1 > \cdots > p_k$ be the distinct values of $p(v)$, and put $L_i = \{v \mid p(v) \geq p_i\}$, the ith *level set* of p. A p-maximum base exists if and only if $L_i \in \mathcal{D}$ for every i. Let $f_p : \mathcal{D} \to \mathbf{R}$ be defined by

$$f_p(X) = \sum_{i=1}^{k} \{f((X \cap L_i) \cup L_{i-1}) - f(L_{i-1})\},$$

where $L_0 = \emptyset$. The following theorems give properties of $\mathrm{B}(f_p)$ that we will need.

Theorem 2.1 ([2, Theorem 6]) *This f_p is a submodular function with $f_p \leq f$, and $x \in \mathrm{B}(f_p)$ if and only if $x \in \mathrm{B}(f)$ and $x(L_i) = f(L_i)$ for every i.* ■

Theorem 2.2 ([14, Theorem 3.15]) *A base $x \in \mathrm{B}(f)$ is p-maximum if and only if $x \in \mathrm{B}(f_p)$.* ■

2.2 Exchange Capacity

For each $v \in V$, we define $\vec{v} \in \mathbf{R}^V$ such that $\vec{v}(v) = 1$ and $\vec{v}(w) = 0$ for $w \neq v$. For a base $x \in \mathrm{B}(f)$ and a pair of distinct elements $v, w \in V$, we define the *exchange capacity* $\sigma(x, v, w)$ by

$$\sigma(x, v, w) = \max\{\alpha \mid \alpha \in \mathbf{R},\ x + \alpha(\vec{v} - \vec{w}) \in \mathrm{B}(f)\}.$$

Equivalently, the exchange capacity can be written as

$$\sigma(x, v, w) = \min\{f(X) - x(X) \mid v \in X,\ w \notin X,\ X \in \mathcal{D}\}.$$

Given $x \in \mathrm{B}(f)$, the *exchangeability graph* w.r.t. x is the directed graph with node set V and arc set $E_x = \{(w, v) \mid \sigma(x, v, w) > 0\}$. An arc $(w, v) \in E_x$ is called an *exchange arc.* For $0 < \alpha \leq \sigma(x, v, w)$ the operation $x = x + \alpha(\vec{v} - \vec{w})$ is called *augmenting* (w, v) by α. If $\alpha = \sigma(x, v, w)$, then we say that this operation *saturates* (w, v).

In common with most other submodular flow algorithms we assume that we have an *exchange capacity oracle*: the oracle takes input x, v, w and outputs $\sigma(x, v, w)$ in h time. Thus our running times will include a term in h for the number of calls to this oracle. Recent combinatorial algorithms for submodular function minimization (see Schrijver [31] or Iwata, Fleischer, and Fujishige [21]) have shown that an exchange oracle can be implemented using only an evaluation oracle (whose input is $X \in \mathcal{D}$ and whose output is $f(X)$). This has in turn led to a new submodular flow algorithm which needs only an evaluation oracle, see Fleischer and Iwata [7].

Lemma 2.3 ([14, Theorem 3.16]) *A base $x \in \mathrm{B}(f)$ is p-maximum if and only if $p(w) \geq p(v)$ holds for any $(w, v) \in E_x$.*

Proof. The "only if" part is immediate from the definition of the exchangeability graph. To prove the "if" part, Theorem 2.1 says that it suffices to show that $x(L_i) = f(L_i)$ holds. For any pair of $w \in V - L_i$ and $v \in L_i$, it follows from $(w, v) \notin E_x$ that there exists a subset $X_{wv} \in \mathcal{D}$ such that $v \in X_{wv}$, $w \notin X_{wv}$, and $x(X_{wv}) = f(X_{wv})$. Then by the submodularity of f and

$$L_i = \bigcup_{v \in L_i} \bigcap_{w \in V - L_i} X_{wv},$$

we obtain $x(L_i) = f(L_i)$. ■

Lemma 2.4 *The exchangeability graph is transitive. Namely, $(s, v) \in E_x$ and $(v, t) \in E_x$ imply $(s, t) \in E_x$.*

Proof. Suppose to the contrary $(s, t) \notin E_x$. Then there must be an $X \in \mathcal{D}$ that satisfies $t \in X$, $s \notin X$, and $x(X) = f(X)$. If $v \in X$ this contradicts $(s, v) \in E_x$, but if $v \notin X$ it contradicts $(v, t) \in E_x$. ■

Lemma 2.5 ([14, Figure 4.4]) *Suppose $(s, t) \notin E_x$ and $(s, t) \in E_y$ hold for a base y obtained from $x \in \mathrm{B}(f)$ by $y = x + \alpha(\vec{v} - \vec{w}) \in \mathrm{B}(f)$ with $0 < \alpha \leq \sigma(x, v, w)$. Then $s \neq v$ implies $(s, v) \in E_x$ and $t \neq w$ implies $(w, t) \in E_x$.*

Proof. Since $(s,t) \notin E_x$, there must be an $X \in \mathcal{D}$ that satisfies $t \in X$, $s \notin X$, and $x(X) = f(X)$. If $s \neq v$ and $(s,v) \notin E_x$, then there exists an $Y \in \mathcal{D}$ such that $v \in Y$, $s \notin Y$, and $x(Y) = f(Y)$. Submodularity of f and $v \in X \cup Y$ imply that $y(X \cup Y) = x(X \cup Y) = f(X \cup Y)$, which together with $t \in X \cup Y$ and $s \notin X \cup Y$ contradicts $(s,t) \in E_y$. The $t \neq w$ case is similar. ■

Lemma 2.6 ([14, Theorem 4.4]) *For distinct nodes $s,t,v,w \in V$ and a base $x \in \mathrm{B}(f)$, suppose $(w,v) \in E_x$, $(s,t) \in E_x$, and $(w,t) \notin E_x$. Then $\sigma(y,t,s) = \sigma(x,t,s)$ holds for a base y obtained by $y = x + \alpha(\vec{v} - \vec{w}) \in \mathrm{B}(f)$ with $0 < \alpha \leq \min\{\sigma(x,t,s), \sigma(x,v,w)\}$.*

Proof. Since $f(X) - y(X) < f(X) - x(X)$ only if $v \in X$ and $w \notin X$, and $\sigma(\cdot,t,s)$ only depends on sets X with $t \in X$, $s \notin X$, it suffices to show that $v,t \in X$ and $s,w \notin X$ imply $f(X) - x(X) \geq \sigma(x,t,s) + \alpha$. Since $(w,t) \notin E_x$, there exists a subset $Y \subseteq V$ such that $t \in Y$, $w \notin Y$, and $x(Y) = f(Y)$. Then $t \in X \cap Y$ and $s \notin X \cap Y$ imply $f(X \cap Y) - x(X \cap Y) \geq \sigma(x,t,s)$. It follows from $v \in X \cup Y$ and $w \notin X \cup Y$ that $f(X \cup Y) - x(X \cup Y) \geq \sigma(x,v,w)$ holds. Therefore we obtain $f(X) - x(X) \geq \sigma(x,t,s) + \sigma(x,v,w) \geq \sigma(x,t,s) + \alpha$ by the submodularity of f. ■

Lemma 2.7 ([14, Theorem 4.5]) *Let x be a base of f. Suppose $(w_1,v_1),(w_2,v_2),\cdots,(w_k,v_k) \in E_x$ have distinct end-nodes and that $i < j$ implies $(w_i,v_j) \notin E_x$. If $\alpha \leq \sigma(x,v_i,w_i)$ holds for every $i = 1,2,\ldots,k$, then $y = x + \alpha\sum_{i=1}^{k}(\vec{v}_i - \vec{w}_i)$ is also a base of f.*

Proof. By Lemma 2.6, $x' = x + \alpha(\vec{v}_1 - \vec{w}_1) \in \mathrm{B}(f)$ satisfies $\sigma(x',v_i,w_i) = \sigma(x,v_i,w_i)$ for $i = 2,\ldots,k$. It follows from Lemma 2.5 that $i < j$ implies $(w_i,v_j) \notin E_{x'}$. Therefore the result follows by induction on k. ■

3 The Submodular Flow Problem

Let $G = (V,A)$ be a directed graph with *lower* and *upper bounds* (*capacities*) $l : A \to \mathbf{R} \cup \{-\infty\}$ and $u : A \to \mathbf{R} \cup \{+\infty\}$ with $l \leq u$, and costs $c \in \mathbf{R}^A$. A *flow* is a function φ obeying $l(a) \leq \varphi(a) \leq u(a)$, $\forall a \in A$. We assume there are no parallel arcs, nor loops, nor isolated nodes in G, so that $\Theta(n) \leq m \leq n(n-1)$. Given arc $(w,v) = a \in A$, define $\partial^+ a = w$ and $\partial^- a = v$. Given a set $X \subset V$, define $\Delta^+ X := \{a \in A \mid \partial^+ a \in X, \partial^- a \notin X\}$ (the set of arcs leaving X), and $\Delta^- X := \{a \in A \mid \partial^+ a \notin X, \partial^- a \in X\}$ (the set of arcs entering X). Given a flow φ and $X \subseteq V$, the *boundary* of φ on X is $\partial\varphi(X) = \varphi(\Delta^+ X) - \varphi(\Delta^- X)$, the net flow leaving X. Let f be a submodular function on a distributive lattice $\mathcal{D} \subseteq 2^V$ with $f(\emptyset) = f(V) = 0$. Then the *submodular flow problem* is:

$$\begin{array}{ll} \text{Minimize} & c^\top \varphi \\ \text{subject to} & l \leq \varphi \leq u \\ & \partial\varphi \in \mathrm{B}(f). \end{array}$$

A feasible solution is called a *submodular flow.*

Theorem 3.1 (Frank [8]) *There exists a submodular flow if and only if*

$$l(\Delta^+ X) - u(\Delta^- X) \leq f(X) \tag{2}$$

holds for every $X \in \mathcal{D}$. If in addition l, u, f are integer valued, then there exists an integral submodular flow. ■

Given a *price function* (or *node potentials*) $p \in \mathbf{R}^V$, we define the *reduced cost* w.r.t. p as $c_p(a) = c(a) + p(\partial^+ a) - p(\partial^- a)$ for each $a \in A$. The following optimality conditions for the submodular flow problem are due to Fujishige [14, Theorem 5.3]. Edmonds and Giles [4] show that the submodular flow problem is totally dual integral, which implies the integrality claim.

Theorem 3.2 *A submodular flow φ is optimal if and only if there exists $p \in \mathbf{R}^V$ such that:*
⟨a⟩ *For any $a \in A$, $c_p(a) > 0$ implies $\varphi(a) = l(a)$, and $c_p(a) < 0$ implies $\varphi(a) = u(a)$, and*
⟨b⟩ *The boundary $\partial\varphi$ is a p-maximum base in* $\mathrm{B}(f)$.
Moreover, if c is integral, then we may restrict the above p to be integral. ■

Edmonds and Giles [4] originally formulated the submodular flow problem for crossing submodular functions (defined below), which are more general than what we have assumed here. They used crossing submodular functions because one of their aims was to generalize the Lucchesi–Younger theorem on the directed cut covering problem [26], and crossing functions provide a natural model in that case.

A pair of subsets $X, Y \subseteq V$ is called *crossing* if $X \cap Y \neq \emptyset$, $X - Y \neq \emptyset$, $Y - X \neq \emptyset$, and $X \cup Y \neq V$. A family $\mathcal{F} \subseteq 2^V$ is a *crossing family* if $X \cap Y \in \mathcal{F}$ and $X \cup Y \in \mathcal{F}$ hold for any crossing pair $X, Y \in \mathcal{F}$. A *crossing submodular function* is a function $f : \mathcal{F} \to \mathbf{R}$ that satisfies the submodular inequality (1) for any crossing pair $X, Y \in \mathcal{F}$.

For a crossing submodular function $f : \mathcal{F} \to \mathbf{R}$ with $\emptyset, V \in \mathcal{F}$ and $f(\emptyset) = 0$, the base polyhedron $\mathrm{B}(f)$ is defined by

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

This base polyhedron may be empty. Fujishige [12] has provided a necessary and sufficient condition for $\mathrm{B}(f)$ to be nonempty, and showed that a nonempty base polyhedron $\mathrm{B}(f)$ of a crossing submodular function f is identical to a base polyhedron $\mathrm{B}(\tilde{f})$ of a submodular function $\tilde{f}$ defined on a distributive lattice that contains $\mathcal{F}$, also called a *fully submodular* function.

Thus, theorems and algorithms that rely only on geometric properties of the base polyhedron carry over to crossing submodular functions. For example, Theorem 3.2 is valid for the submodular flow problem for crossing submodular functions since it refers only to base polyhedra, and does not require an algorithm to compute a value for $\tilde{f}$, whereas Theorem 3.1 is not geometric because it requires a suitable modification that replaces f in (2) by the fully submodular function $\tilde{f}$. The equivalent theorem for the crossing case was obtained by Frank [8].

An exchange oracle must minimize $f(X) - x(X)$ over sets X containing v but not w. For crossing submodular functions these sets form a distributive lattice, so it is reasonable for us to assume that we have an exchange oracle in this case as well.

In the rest of this paper, we assume f to be fully submodular for the sake of simplicity, but we indicate at the appropriate places that our results in fact extend also to the crossing case.

4 The Successive Shortest Path Algorithm

This section describes the successive shortest path (SSP) algorithm for the submodular flow problem, which is the basis for our algorithms. As with the successive shortest path algorithm for minimum cost flows, this algorithm is not polynomial, but only pseudopolynomial since the number of iterations is linear in U. The algorithm was originally presented by Fujishige [11] for solving the minimum cost independent flow problem, which turns out to be equivalent to the submodular flow problem [14, §5.2].

In order to deal with the distributive lattice $\mathcal{D}$ algorithmically, we need to know a compact representation of $\mathcal{D}$. As Birkhoff's representation theorem [1] suggests, we represent $\mathcal{D}$ by a directed graph with the vertex V and the arc set $E^\circ = \{(w, v) \mid \forall X \in \mathcal{D} : v \in X \Rightarrow w \in X\}$. Note that (V, E°) is transitive, i.e., $(s, v) \in E^\circ$, $(v, t) \in E^\circ$ implies $(s, t) \in E^\circ$, and that $(w, v) \in E^\circ$ implies that for any $x \in \mathrm{B}(f)$, $\sigma(x, v, w) = \infty$, so that $(w, v) \in E_x$.

If (V, E°) had a directed cycle, by transitivity there would be $w, v \in V$ with both (w, v) and (v, w) in E°, and so in E_x for any $x \in \mathrm{B}(f)$. Hence by Lemma 2.3 and Theorem 3.2, any optimal price p^* must satisfy $p^*(v) = p^*(w)$, implying that if $a = (w, v) \in A$, any optimal flow φ^* must satisfy $\varphi^*(a) = l(a)$ if $c(a) > 0$, or $\varphi^*(a) = u(a)$ if $c(a) < 0$. Therefore, shrinking such v and w to a single node does not change the problem. Thus we will assume in the rest of this paper that (V, E°) is acyclic.

We initialize by looking for a flow φ and a price vector p that satisfy condition $\langle a \rangle$ of Theorem 3.2. We first check whether there exists a dual feasible solution p. To do this we construct a directed graph $G^\circ = (V, A^\circ)$ with the arc set $A^\circ = F^\circ \cup B^\circ \cup E^\circ$ defined by $F^\circ = \{a \mid u(a) = +\infty\}$ and $B^\circ = \{\overline{a} \mid l(a) = -\infty\}$, where $\overline{a}$ is the reversal of a. Each arc in A° has length $c(a)$ if $a \in F^\circ$, $-c(\overline{a})$ if $a \in B^\circ$, and 0 if $a \in E^\circ$. We use the $\mathrm{O}(mn)$ Bellman–Ford–Moore shortest path algorithm to try to compute shortest path distances p in G°. If the algorithm finds a negative cycle in G°, then the instance is dual infeasible [14, Theorem 5.5]. Otherwise, the shortest path distances p satisfy $c_p(a) > 0$ implies $l(a) > -\infty$ and $c_p(a) < 0$ implies $u(a) < +\infty$. Then we can construct a flow φ that satisfies condition $\langle a \rangle$ of Theorem 3.2 in $\mathrm{O}(m)$ time.

We also start with a base $x \in \mathrm{B}(f_p)$, which can be obtained using the greedy algorithm [3, 17]. If $x = \partial\varphi$, then x satisfies condition $\langle b \rangle$ of Theorem 3.2, proving that φ is optimal (and that p is dual optimal). The algorithm will iteratively modify φ, x, and p so that $\langle a \rangle$ is preserved for φ and p, $\langle b \rangle$ is preserved for x and p, and so that $\partial\varphi$ and x eventually converge.

We construct an auxiliary network $\widehat{G}(\varphi, x) = (V, A_\varphi \cup E_x)$ with respect to flow φ and base $x \in \mathrm{B}(f_p)$. The arc set $A_\varphi := F_\varphi \cup B_\varphi$ is defined by $F_\varphi := \{a \mid a \in A,\ \varphi(a) < u(a)\}$ and $B_\varphi := \{\overline{a} \mid a \in A,\ \varphi(a) > l(a)\}$ (i.e., A_φ is just the ordinary residual graph of flow φ). As defined in Section 2.2, E_x is the arc set of the exchangeability graph with respect to $\mathrm{B}(f)$ (we emphasize that this is $\mathrm{B}(f)$ rather than $\mathrm{B}(f_p)$ to allow for changes to p). For each arc $a \in A_\varphi \cup E_x$, we define the residual capacity $r(a)$ and the cost $c(a)$ by

$$r(a) := \begin{cases} u(a) - \varphi(a) & (a \in F_\varphi) \\ \varphi(a) - l(a) & (a \in B_\varphi) \\ \sigma(x, v, w) & (a = (w, v) \in E_x), \end{cases} \qquad c(a) := \begin{cases} c(a) & (a \in F_\varphi) \\ -c(\overline{a}) & (a \in B_\varphi) \\ 0 & (a \in E_x). \end{cases}$$

The algorithm computes augmentations of φ and x with respect to r on $A_\varphi \cup E_x$. This will ensure that φ remains a flow. Together with the corresponding update to p, it will also ensure that x remains in $\mathrm{B}(f_p)$.

Each augmentation is determined as follows. We regard the reduced cost $c_p(a) = c(a) + p(\partial^+ a) - p(\partial^- a)$ as the length of $a \in A_\varphi \cup E_x$. Define $S^+ = \{v \mid x(v) > \partial\varphi(v)\}$ and $S^- = \{v \mid x(v) < \partial\varphi(v)\}$ as the sets of nodes where $\partial\varphi$ differs from x. The algorithm augments along a shortest path (with respect to c_p) from S^+ to S^- with a minimum number of arcs. Since φ and p satisfy $\langle a \rangle$, which is equivalent to having $c_p(a) \geq 0$ for all $a \in A_\varphi$, and since $x \in \mathrm{B}(f_p)$, which is equivalent to having $c_p(a) \geq 0$ for $a \in E_x$, we can compute shortest path distances $d(v)$ from S^+ to each $v \in V$ using Dijkstra's algorithm. Among the nodes in S^- with minimum $d(v)$, the algorithm selects a path P with a minimum number of arcs, and augments φ on P by $\alpha := \min\{x(s) - \partial\varphi(s),\ \partial\varphi(t) - x(t),\ \min\{r(a) \mid a \in P\}\}$. Since φ is augmented only on P and not on any longer paths from S^+ to S^-, the algorithm updates $p(v)$ by $\min(d(v), c_p(P))$ for each

```
SSP $(x, \varphi, p)$
Repeat the following Steps (1-1)–(1-4), until $S^+ = \emptyset$.
(1-1) Compute the shortest distance $d(v)$ from $S^+$ to each $v \in V$ in $\widehat{G}(\varphi, x)$ with respect
      to the arc length $c_p$. Among the shortest paths from $S^+$ to $S^-$, let $P$ be one with a
      minimum number of arcs. Let $s$ be the initial node of $P$ and $t$ be the final node of $P$.
(1-2) For each $v \in V$, put $p(v) := p(v) + \min\{d(v), c_p(P)\}$.
(1-3) Put $\alpha := \min\{x(s) - \partial\varphi(s),\ \partial\varphi(t) - x(t),\ \min\{r(a) \mid a \in P\}\}$.
(1-4) For each arc $a \in P$,
        • if $a \in F_\varphi$, then $\varphi(a) := \varphi(a) + \alpha$;
        • if $a \in B_\varphi$, then $\varphi(\overline{a}) := \varphi(\overline{a}) - \alpha$;
        • if $a \in E_x$, then $x(\partial^+ a) := x(\partial^+ a) - \alpha$, $x(\partial^- a) := x(\partial^- a) + \alpha$.
```

Figure 1: Successive Shortest Path algorithm.

$v \in V$. This augmentation of φ and update of p preserves ⟨a⟩ and ⟨b⟩, decreases $|\partial\varphi(v) - x(v)|$ by the augmentation amount for the initial and final nodes of P, and does not change $|\partial\varphi(v) - x(v)|$ for any other node. The details of this algorithm SSP are in Figure 1.

If there is no path from S^+ to S^-, then the problem is infeasible. In fact, the set X of nodes not reachable from S^+ satisfies $l(\Delta^+ X) - u(\Delta^- X) = \varphi(\Delta^+ X) - \varphi(\Delta^- X) = \partial\varphi(X) > x(X) = f(X)$, which proves infeasibility by Theorem 3.1.

Since P has a minimum number of arcs among the shortest paths, it follows from Lemma 2.4 that there is no consecutive pair of exchangeability arcs in P. The reduced cost c_p is nonnegative after Step (1-2) of Figure 1. In particular, each arc a in P satisfies $c_p(a) = 0$. The following Lemma 4.1 makes it possible to apply Lemma 2.7 to show that x is a base after Step (1-3) of Figure 1. Furthermore, Lemma 2.5 can be used to show that c_p remains nonnegative.

Lemma 4.1 *There exists a numbering $(w_1, v_1), (w_2, v_2), \cdots, (w_k, v_k)$ of the arcs in $P \cap E_x$ such that $i < j$ implies $(w_i, v_j) \notin E_x$.*

Proof. If no such numbering exists, there is a subset $\{(w_j, v_j) \mid j = 1, \ldots, q\}$ of $P \cap E_x$ such that $e_j = (w_{j-1}, v_j) \in E_x$ holds for each j, where $w_0 \equiv w_q$. After Step (1-2) of Figure 1, we have $p(w_j) = p(v_j)$, which implies $\sum_{j=1}^{q} c_p(e_j) = 0$. Therefore $c_p(e_j) = 0$ holds for each $j = 1, \ldots, q$, which contradicts the minimality of P. ■

If l, u and f are integer valued, then initial x and φ are integer, hence α is always integer and at least 1, so that $\sum_{v \in V} |x(v) - \partial\varphi(v)|$ is always a nonnegative integer and decreases by at least two with each augmentation. Hence the algorithm terminates in a finite number of iterations. Since the number of arcs in $\widehat{G}(\varphi, x)$ is $|A_\varphi| + |E_x| = \mathrm{O}(m) + \mathrm{O}(n^2) = \mathrm{O}(n^2)$, the bottleneck Dijkstra computation takes $\mathrm{O}(n^2 h)$ time, so each augmentation requires $\mathrm{O}(n^2 h)$ time.

Most of the successive shortest path algorithm works for crossing submodular functions. The only difference is we need to compute an initial base x using the bi-truncation algorithm of Frank and Tardos [10] (cf. [28]) instead of the greedy algorithm.

5 A Capacity Scaling Algorithm

This section develops our new capacity scaling algorithm for solving submodular flow problems. We assume throughout this section that l, u and f are integer valued.

5.1 Algorithm Overview

To obtain a polynomial-time algorithm, we embed a variant of the successive shortest path algorithm within a capacity scaling framework. For the minimum cost flow problem, Edmonds and Karp [5] propose a capacity scaling algorithm that maintains reduced cost optimality conditions while rounding the arc capacities. We modify the Edmonds–Karp algorithm by attaching a complete directed graph on the node set, which effectively relaxes the submodular constraints.

Without loss of generality, we assume that the graph restricted to infinite capacity arcs is strongly connected. With this assumption, between any pair of nodes there is always an infinite capacity path in the residual graph of any flow with non-negative reduced cost. This assumption may be enforced by introducing high cost, infinite capacity arcs between all node pairs. If these arcs are used in the final solution, then the problem is infeasible.

Neither the minimum cost flow version of SSP, nor our submodular flow version described in Section 4, runs in polynomial time, because the number of iterations might depend linearly on U. The Edmonds–Karp algorithm effectively deals with this by scaling capacities in the residual graph by a parameter $\delta > 0$, so that each augmentation is by at least δ units of flow. To modify this for submodular flow, we instead relax capacities by δ. We do this by adding to our initial graph the arc set of a complete directed graph (V, D) with arc set $D = \{(w, v) \mid w \neq v \in V\}$ with capacity δ. For all $a \in D$ we extend c, l, and u to D by setting $c(a) = 0$, $l(a) = 0$, and $u(a) = \delta$. Define the submodular function $b : 2^V \to \mathbf{R}$ by $b(X) = |X| \cdot |V - X|$. Thus $\delta b(X)$ is the capacity of the cut X in (V, D). This can be thought of as either relaxing the condition that $\partial\varphi \in \mathrm{B}(f)$ to $\partial\varphi \in \mathrm{B}(f + \delta b)$, as relaxing the capacities l and u by δ (viewing A as a complete directed graph with some 0-capacity arcs), or both.

To avoid confusion we will always use φ for a flow on the original set of *flow arcs* A, we will always use ψ for a flow on the *relaxation arcs* D, and we will always use ξ for a flow on the combined arc set $I := A \cup D$. Note that there is also a third arc set, the *exchange* arcs. There is no actual flow on an exchange arc, but augmenting along an exchange arc (w, v) by δ corresponds to changing base x by $x(w) = x(w) - \delta$ and $x(v) = x(v) + \delta$.

At any given point in the algorithm we will have a flow φ on A, and flow ψ on D, a price vector p, and a base $x \in \mathrm{B}(f_p + \delta b)$. We maintain x as the sum of $y \in \mathrm{B}(f_p)$ and $-\partial\psi \in \mathrm{B}(\delta b)$. We compute exchange capacities with respect to y and with respect to the original function f, not f_p (as in SSP), so we denote the set of exchange arcs by E_y, as defined in Section 2.2. We measure progress in the algorithm via the *discrepancy* between x and $\partial\varphi$, defined by $\Phi = \sum_v |x(v) - \partial\varphi(v)|$. In a sense, we relax the submodular constraints on φ twice: once in relaxing f to $f + \delta b$, and once in letting $\partial\varphi$ and x differ. Recall that, because the algorithm maintains $\langle a \rangle$ of Theorem 3.2 for φ and p, and $\langle b \rangle$ of Theorem 3.2 for y and p, if we can drive Φ to zero and ψ to zero so that $x = y = \partial\varphi$, we are optimal.

The algorithm RCS-MCSF is described in Figure 2. It embeds a modified version of SSP in a scaling framework that depends on parameter δ. We call the iterations with a fixed value of δ a *δ-scaling phase*, or just a phase for short. Inside a phase, we find shortest paths among paths from $S^+(\delta) := \{v \mid x(v) - \partial\varphi(v) \geq \delta\}$ to $S^-(\delta) := \{v \mid \partial\varphi(v) - x(v) \geq \delta\}$ with residual capacity at least δ (by our assumption on the subgraph on infinite capacity arcs, there is always an augmenting path between any two such nodes). The lower bound of δ on the residual capacity of an augmenting path will ensure a small number of iterations within a phase, since we are making "large" changes at each augmentation. At the end of a phase we set $\delta := \delta/2$ and continue until $\delta = 1$, at which point we can finish using SSP.

We start with the same initial flow φ, base x, and price function p as in SSP. For each $v \in V$, let $R(v) = \{w \mid (w,v) \in E^\circ\}$. Then $R(v) \cup \{v\}$ is the unique minimal member of $\mathcal{D}$ containing v. Define U as

$$U = \max\{\max\{|u(a)| \mid u(a) < +\infty\}, \max\{|l(a)| \mid l(a) > -\infty\}, \max_{v \in V}\{f(R(v) \cup \{v\}) - f(R(v))\}\}.$$

Since $|\varphi(a)| \leq U$ for all $a \in A$, $|\partial\varphi(v)| \leq (n-1)U$ for all $v \in V$. A p-maximum base x comes from the greedy algorithm, which by Lemma 2.3 can order $R(v)$ before v, so we have that $x(v) \leq f(R(v) \cup \{v\}) - f(R(v)) \leq U$ for each $v \in V$. Since $x(V) = f(V) = 0$, we have $|x(v)| \leq (n-1)U$. Thus the initial discrepancy between x and $\partial\varphi$ is at most $2n^2U$.

For any distinct $w, v \in V$, we may assume that at least one of $\psi(w,v)$ and $\psi(v,w)$ is zero, so that at least one of $r(v,w)$ and $r(w,v)$ equals δ. Given such a flow ψ, let D_ψ denote the set of relaxation arcs with residual capacity equal to δ, i.e., $D_\psi = \{(w,v) \mid \psi(w,v) = 0\}$. Define $A_\varphi(\delta) := \{a \mid a \in A_\varphi,\ r(a) \geq \delta\}$. Subroutine SubmodularDijkstra of Figure 3 adapts Dijkstra's algorithm to find a shortest path from any node in $S^+(\delta)$ to any node in $S^-(\delta)$ in the network with arc set $A_\varphi(\delta) \cup D_\psi$, while maintaining the δ reduced cost optimality condition that

$$c_p(a) \geq 0 \text{ holds for every } a \in A_\varphi(\delta) \cup D_\psi \cup E_y. \tag{3}$$

We call such a path a δ-*augmenting* path. It might seem that the algorithm should also include the subset of arcs of E_y with exchange capacity at least δ in its list of arcs eligible for a δ-augmenting path. Instead, SubmodularDijkstra performs exchange operations on the E_y arcs in the search procedure, adjusting flow ψ on the parallel D arcs to maintain $x = y - \partial\psi$ without changing x, so that δ-augmenting paths contain only flow and relaxation arcs. This procedure is described in detail in Section 5.2. Thus the only time that flow is "augmented" on exchange arcs is through this search procedure. It might also seem that on E_y, (3) should be maintained only for the subset of arcs of E_y with exchange capacity at least δ. However, the exchanges between parallel D and E_y arcs makes it possible to include *all* arcs of E_y in (3), which is important for proving that $y \in \mathrm{B}(f_p)$ is preserved, see Corollary 5.7.

Since $x = y - \partial\psi$ and we augment by exactly δ, each augmentation decreases the discrepancy by δ at both endpoints of the augmenting path, and maintains the discrepancy of all other nodes. A phase ends when one of $S^+(\delta)$, $S^-(\delta)$ is empty. Since $\sum_v x(v) - \partial\varphi(v) = 0$, at the end of a phase we have $\Phi \leq 2n\delta$.

At the start of a new phase, we modify ψ to satisfy the capacity constraints for the new value of δ, and modify φ to satisfy the reduced cost optimality constraints (3) for arcs a with residual capacity $\delta \leq r(a) < 2\delta$. After these adjustments, the initial discrepancy in a δ-phase is at most $4n\delta + 2n^2\delta + 4m\delta$. Thus the total number of augmentations in any δ phase is at most $3n^2 + 2n = \mathrm{O}(n^2)$, since each augmentation decreases the discrepancy by 2δ. Note that the total discrepancy is roughly halved in each phase.

Initially, we set $\delta = 2^i$ for $i = \lceil \log_2(\Phi/2n^2) \rceil \leq \lceil \log U \rceil$. Thus, after at most $\log U$ scaling phases, $\delta = 1$ and $\partial\varphi = x \in \mathrm{B}(f_p + b)$. We then remove the flow ψ and its contribution to x so that $x = y \in \mathrm{B}(f_p)$ and the discrepancy Φ is at most $2n^2$. At this point, we can call SSP to obtain an optimal solution in at most n^2 additional augmentations. The following theorem summarizes this discussion.

Theorem 5.1 *Algorithm* RCS-MCSF *calls* SubmodularDijkstra $\mathrm{O}(n^2 \log U)$ *times.* ■

```
Relaxed Capacity Scaling for Minimum Cost Submodular Flow
RCS-MCSF$(G, f)$

Input: Directed graph $G = (V, A, l, u, c)$ and submodular function $f : \mathcal{D} \to \mathbf{Z}$
Output: Flow $\varphi$ minimizing $c^\top \varphi$ over all $l \le \varphi \le u$ and $\partial\varphi \in \mathrm{B}(f)$

    Initialization:
        $p :=$ shortest path distances in $G^\circ$
        $\varphi :=$ flow that satisfies condition $\langle a \rangle$ of Theorem 3.2
        $x := p$-maximum base in $\mathrm{B}(f)$, i.e., $x \in \mathrm{B}(f_p)$
        $y := x$, $\psi \equiv 0$
        $\delta := 2^{\lceil \log_2(\Phi/2n^2) \rceil}$
    while $\delta \ge 1$
        for all $a \in D$, $\psi(a) = \min\{\psi(a), \delta\}$.
        $x := y - \partial\psi$
        for all $a \in A_\varphi(\delta)$,
            if $c_p(a) < 0$ then send $r(a)$ units of flow through $a$, and update $\varphi$ accordingly.
        while $S^+(\delta) \ne \emptyset$ and $S^-(\delta) \ne \emptyset$
            $(P, d) :=$ SubmodularDijkstra$(\widehat{G}(\varphi, \psi), \delta, p, y)$
            [ $P$: shortest path from $S^+(\delta)$ to $S^-(\delta)$ of residual capacity $\ge \delta$. ]
            [ $d$: shortest distance labels. ]
            Augment by $\delta$ on $P$.   [ Update $\varphi$, $\psi$, $x$. ]
            for all $v \in V$, $p(v) := p(v) + \min\{d(v), c_p(P)\}$
        $\delta := \delta/2$
    $\varphi :=$ SSP$(y, \varphi, p)$
    return $\varphi$.

    Invariants:
        $c_p(a) \ge 0$ for all $a \in A_\varphi(\delta) \cup D_\psi \cup E_y$.
        $x = y - \partial\psi$.
        $y \in \mathrm{B}(f_p)$.
```

Figure 2: High level capacity scaling algorithm.

5.2 Maintaining Optimality Conditions

The keys to the correctness of the algorithm are maintaining the reduced cost optimality conditions (3) for all flow arcs with residual capacity at least δ and all exchange capacity arcs, and maintaining $y \in \mathrm{B}(f_p)$ for the current value of p. This requires extra care in choosing an augmenting path. In the subroutine SubmodularDijkstra (see Figure 3), we modify Dijkstra's algorithm to maintain (3) for exchange capacity arcs. The modification entails a more careful treatment of the arcs emanating from the node w that is added to the set R of permanently labeled nodes in an iteration.

For each zero reduced cost exchange capacity arc $a = (w, v)$ with $v \notin R$, we do a *double-exchange* that reduces the exchange capacity on a and the flow ψ on its parallel relaxation arc so as to preserve the invariant $x = y - \partial\psi$ without changing x. This has two possible outcomes, both of which make a irrelevant to future shortest path operations in this iteration.

```
SubmodularDijkstra($\widehat{G}(\varphi,\psi),\delta,p,y$)

Input: Residual graph $\widehat{G}(\varphi,\psi)$ of $\varphi$ and $\psi$; scale factor $\delta$; dual prices $p$; and $y \in \mathrm{B}(f_p)$.
Output: A $\delta$-augmenting path $P$ from $S^+(\delta)$ to $S^-(\delta)$.

    Initialization:
        $d(v) := +\infty \quad \forall v \in V - S^+(\delta)$
        $d(v) := 0 \quad \forall v \in S^+(\delta)$
        $R := \emptyset$   [ set of permanently labeled nodes ]
    while $R \cap S^- = \emptyset$ do
        $w :=$ node in $V - R$ with smallest label.
        $R := R \cup \{w\}$
        [ scan $w$ ]
        for all $a = (w,v) \in E_y$ with $v \in V - R$, $c_p(a) = 0$, and $\psi(w,v) > 0$ do
            [ do double exchange on parallel $(w,v)$ arcs in $E_y$ and $D$ ]
            $\alpha := \min\{\psi(w,v), \sigma(y,v,w)\}$
            $y := y + \alpha(\vec{v} - \vec{w})$
            $\psi(w,v) := \psi(w,v) - \alpha$
        for all $a = (w,v) \in A_\varphi(\delta) \cup D_\psi$ with $v \in V - R$ do
            $d(v) := \min\{d(v), d(w) + c_p(a)\}$
    return path $P$ from $s$ to $w$ on nodes in $R$.

    Invariants:
        $d(\partial^- a) \le d(\partial^+ a) + c_p(a)$ for all $a \in A_\varphi(\delta) \cup D_\psi \cup E_y$
        $x = y - \partial\psi$
```

Figure 3: Subroutine to find a δ-augmenting path.

One outcome happens when $\psi(w,v) \le \sigma(y,v,w)$. In this case we drive $\psi(w,v)$ to zero, so that the residual capacity of $(w,v) \in D$ becomes δ, and we can use $(w,v) \in D$ instead of a in any δ-augmenting path. The other outcome happens when $\psi(w,v) \ge \sigma(y,v,w)$. In this case we saturate a, making $\sigma(y,v,w) = 0$, so that a cannot be used in any shortest path. The double-exchange operation is in some sense a variant of the AdjustFlow routine from [23]. The correctness of SubmodularDijkstra mostly follows in the same way as the use of Dijkstra's Algorithm for ordinary SSP, except that we need to be careful about exchange arcs.

The double-exchange operation is applied only to exchange capacity arcs with 0 reduced costs. If the reduced cost of an exchange capacity arc is positive, then its opposite relaxation arc cannot have residual capacity δ (otherwise it would be a negative reduced cost arc in D_ψ, violating (3)). Hence its parallel relaxation arc has residual capacity δ, with the same reduced cost. This relaxation arc can always be chosen instead of the exchange capacity arc in any δ-augmenting path.

The double-exchange operation can lead to the creation of new exchange capacity arcs (see Lemma 2.5). We must therefore argue that after the operation: $\langle$A$\rangle$ the new exchangeability graph maintains the nonnegative reduced cost condition (3); $\langle$B$\rangle$ the labels assigned to nodes by Dijkstra's algorithm continue to reflect shortest path distances; and $\langle$C$\rangle$ the algorithm terminates, i.e., we do not pursue a cycle of double-exchange operations. In the following sequence of lemmas, R is the set of permanently labeled nodes from SubmodularDijkstra.

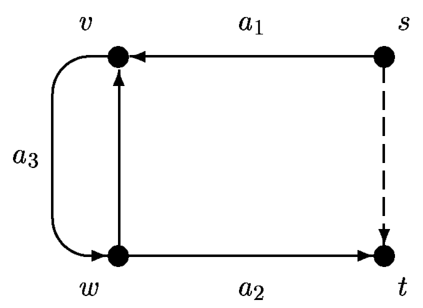

Figure 4: The path a_1, a_3, a_2 in the proof of Lemma 5.2.

Lemma 5.2 *When* RCS-MCSF *applies the double-exchange operation on a 0-reduced cost exchange arc,* ⟨A⟩ *the reduced cost optimality condition* (3) *is maintained, and* ⟨B⟩ *the distance labels d computed so far remain valid.*

Proof. We must check (3) for any new exchange arcs created by the double-exchange applied to (w, v). Let y and r denote values *before* the double-exchange.

The reverse exchange arc (v, w) clearly has reduced cost zero. If exchange arc (s, t) appears, then, by Lemma 2.5, exchange arcs $a_1 = (s, v)$ (if $s \neq v$) and $a_2 = (w, t)$ (if $w \neq t$) existed before the double-exchange. Since SUBMODULARDIJKSTRA applied double-exchange to (w, v), the relaxation arc (w, v) had residual capacity $r(w, v) < \delta$, which implies that the reverse relaxation arc, $a_3 = (v, w)$, had residual capacity $r(v, w) = \delta$. Hence there is a directed path a_1, a_3, a_2 from s to t in the auxiliary graph $\widehat{G}(\varphi, \psi, y) = (V, A_\varphi(\delta) \cup D_\psi \cup E_y)$ before the double-exchange (see Figure 4), and this path must have non-negative reduced cost. Since all arcs in this path have original cost 0, and the new exchange arc also has original cost 0, its reduced cost equals the reduced cost of this path, which is nonnegative. Thus ⟨A⟩ is shown.

The new exchange capacity arc has reduced cost equal to the length of the path a_1, a_3, a_2, thus its creation cannot make an existing label invalid, showing ⟨B⟩. ■

Lemma 5.3 *In a call to* SUBMODULAR DIJKSTRA, *a double-exchange operation is applied to each arc at most once. Hence* ⟨C⟩ *we do not pursue a cycle of double-exchange operations.*

Proof. Arc (w, v) participates in a double exchange only when its tail w is being scanned in the while loop of SUBMODULAR DIJKSTRA. The only way for (w, v) to participate twice is if the first double exchange causes (w, v) to leave E_y and a later double exchange of (w, t) causes (w, v) to re-enter E_y. But Lemma 2.5 shows that this cannot happen. ■

The above two lemmas ensure that the path P found by SUBMODULARDIJKSTRA is a shortest path from $S^+(\delta)$ to $S^-(\delta)$ in the auxiliary graph $\widehat{G}(\varphi, \psi, y) = (V, A_\varphi(\delta) \cup D_\psi \cup E_y)$ for the current ψ and y.

Theorem 5.4 *Subroutine* SUBMODULARDIJKSTRA *runs in* $O(n^2 h)$ *time.*

Proof. Each iteration adds one node to R and calls the exchange capacity oracle at most n times. ■

To complete the proof of correctness, we show that the algorithm maintains the claimed invariants that $x = y - \partial\psi$, the reduced cost optimality condition (3), and $y \in B(f_p)$.

Lemma 5.5 *The algorithm maintains $x = y - \partial\psi$.*

Proof. Every time ψ changes, x is updated. The only changes to y occur in the double-exchange operation in SubmodularDijkstra, which includes a modification to ψ to compensate for the corresponding change in y. ∎

Lemma 5.6 *The reduced cost δ-optimality condition* (3) *is maintained after each augmentation.*

Proof. We must show that all flow arcs with residual capacity at least δ, and all exchange capacity arcs with exchange capacity at least δ have non-negative reduced cost after the augmentation and update of p using the distance labels d obtained from SubmodularDijkstra. Let p be the old potential, and q the new potential. For any previously existing arc a with non-negative reduced cost, we have $c_q(a) = c_p(a) + \min\{d(\partial^+ a), c_p(P)\} - \min\{d(\partial^- a), c_p(P)\}$. Since the distance labels satisfy $c_p(a) + d(\partial^+ a) \geq d(\partial^- a)$, we have $c_p(a) + \min\{d(\partial^+ a), c_p(P)\} \geq \min\{c_p(a) + d(\partial^+ a), c_p(P)\} \geq \min\{d(\partial^- a), c_p(P)\}$, which implies $c_q(a) \geq 0$. If there is an arc with residual capacity newly at least δ, it is the reverse arc of an arc a on P. Since P is a shortest path, $c_q(a) = 0$, so $c_q(\overline{a}) = 0$ also. ∎

Corollary 5.7 *The condition that $y \in \mathrm{B}(f_p)$ is maintained after each augmentation.*

Proof. Lemma 2.3 says that $y \in \mathrm{B}(f_p)$ if $p(w) \geq p(v)$ for all $(w, v) \in E_y$, which is equivalent to saying that $c_p(w, v) \geq 0$ for all $(w, v) \in E_y$, which Lemma 5.6 established. ∎

Lemma 5.8 *The reduced cost optimality condition* (3) *is maintained at the start of each δ-phase.*

Proof. At the start of each δ-phase, we modify both ψ and φ. The modification of ψ does not create any new arcs, so (3) is maintained. We modify φ precisely to enforce (3) for all flow arcs, by augmenting arcs for which (3) does not hold. ∎

The above lemmas and theorems imply the main result of this section:

Theorem 5.9 *Algorithm* RCS-MCSF *computes a minimum cost submodular flow in* $\mathrm{O}(n^4 h \log U)$ *time.* ∎

The capacity scaling algorithm applies to crossing submodular flow problems. The only required modification is again the use of the bi-truncation algorithm for an initial base x.

6 A Strongly Polynomial Algorithm

To make the capacity scaling algorithm described in the preceding section strongly polynomial we need to create large decreases in δ to avoid the $\Theta(\log U)$ number of scaling phases. To do this, we obtain a sequence of feasible flows and a decreasing sequence of values of δ for which these flows are respectively *δ-optimal*: A triple $(\xi = (\varphi, \psi), y, p)$ is called δ-optimal if $c_p(a) \geq 0$ for all arcs $a \in A_\varphi(\delta) \cup D_\psi \cup E_y$ (i.e. (3) holds) and the discrepancy $\Phi \leq 2n\delta$. This definition exactly matches the conditions at the end of a δ-scaling phase.

Given a price vector p, the minimum value of δ for which there exists a δ-optimal triple with prices p can be obtained by computing a maximum mean cut, from which we can also obtain the corresponding flow ξ and base y in the triple. The definition and computation of maximum mean cuts is discussed below in Section 6.1.

The broad outline of the strongly polynomial version of our algorithm is similar to other strongly polynomial algorithms that use maximum mean cuts [6, 23]. We divide the δ-scaling phases of RCS-MCSF into *blocks*, so that each block contains $O(\log n)$ phases. We start each block by computing a δ-optimal triple using our current price vector p. In Section 6.2, we show that after performing $O(\log n)$ scaling phases, this value of δ decreases enough so that we can *restrict the sign* of at least one reduced cost in the optimal solution ("restrict the sign" of $c_p(a)$ means that we can prove that $c_{p^*}(a) \geq 0$ or $c_{p^*}(a) \leq 0$ for every optimal p^*). Since there are only $m' := m + n(n-1) = O(n^2)$ reduced costs (one for each arc of $I = A \cup E$), all reduced costs are fixed to zero after $2m' = O(n^2)$ blocks. At this point, we can compute an optimal flow with one maximum submodular flow computation. Thus the algorithm requires only $O(n^2 \log n)$ phases overall. As in the weakly polynomial algorithm described in Section 5, the time per phase is strongly polynomial, $O(n^4 h)$.

6.1 Maximum Mean Cuts and Approximations

In this section, we define maximum mean cuts, explain their usefulness in obtaining δ-optimal triples, and describe how to compute them. Finally, we define a most positive cut, which is easier to compute and can be used to approximate a maximum mean cut. First some preliminary definitions.

Given a price vector p, for the flow arcs we define *modified bounds* l_p and u_p as follows: For $a \in A$, if $c_p(a) > 0$ then $l_p(a) = u_p(a) = l(a)$; if $c_p(a) = 0$ then $l_p(a) = l(a)$ and $u_p(a) = u(a)$; and if $c_p(a) < 0$ then $l_p(a) = u_p(a) = u(a)$. (For $a \in D$, $l_p(a) = l(a)$ and $u_p(a) = u(a)$.) This definition is consistent with Theorem 3.2 in that φ and p are primal and dual optimal if and only if $l_p \leq \varphi \leq u_p$ (equivalent to condition $\langle a \rangle$ of the theorem) and $\partial\varphi \in \mathrm{B}(f_p)$. We will sometimes refer to the network restricted to arc set A, and at other times to the network with arc set $I = A \cup D$. We subscript Δ^+, Δ^-, and ∂ to indicate the relevant arc set.

A *cut* is a proper nonempty subset $X \in \mathcal{D}$ of V. The *value* of cut X with respect to prices p is defined as $\kappa_p(X) := l_p(\Delta_I^+ X) - u_p(\Delta_I^- X) - f_p(X)$. If δ were zero, then Theorems 3.1 and 3.2 would imply that p is dual optimal if and only if there is no cut having positive value, so positive cut value represents the amount of dual infeasibility. Note that as δ increases, $\kappa_p(X)$ decreases, making dual infeasibility less likely. We want the smallest δ such that no positive cuts remain; as is well-known, this ends up being the ratio between cut value and the amount of relaxation crossing the cut. Thus we define the *mean value* of a cut X using a denominator of $b(X) := |X| \cdot |V - X|$, or as $\overline{\kappa}_p(X) = \kappa_p(X)/b(X)$. A *maximum mean cut* T satisfies $\overline{\kappa}_p(T) = \max_{X \in \mathcal{D}} \overline{\kappa}_p(X)$. For fixed p, we denote this value by $\overline{\kappa}_p$. This definition of maximum mean cut differs from the definition in [23] because we relax capacities differently.

The algorithm contains $O(n^2)$ blocks, and hence computes $O(n^2)$ maximum mean cuts. A maximum mean cut can be computed by discrete Newton's algorithm [27, 29] using a maximum submodular flow subroutine for the data l_p, u_p, and f_p, with variable δ. (Cunningham and Frank [2, Theorem 7] show how to simulate an exchange oracle for f_p in $O(1)$ calls to an exchange oracle for f.) This algorithm requires $O(m') = O(n^2)$ calls to submodular maximum flow. Each maximum submodular flow can be computed in $O(n^3 h)$ time using Fujishige and Zhang's algorithm [18, 24], for a total of $O(n^5 h)$ time per max mean cut, and $O(n^7 h)$ time for computing max mean cuts overall. This would dominate the time for the rest of the algorithm, which we will show is only $O(n^6 h \log n)$, so we use a trick of Radzik from [6] to remove this bottleneck: Just one call to submodular max flow (with δ fixed to zero) produces a cut T° maximizing $\kappa_p(T)$, a *most positive cut*. If T^* is a max mean cut, then $\kappa_p(T^\circ) \geq \kappa_p(T^*)$, which implies that $\overline{\kappa}_p = \overline{\kappa}_p(T^*) \leq \overline{\kappa}_p(T^\circ)\frac{b(T^\circ)}{b(T^*)} \leq n\overline{\kappa}_p(T^\circ)$, so that $\overline{\kappa}_p(T^\circ) \geq \overline{\kappa}_p/n$. Thus T° approximates a max mean cut within a factor of n. Using most positive cuts in place of max mean cuts slightly increases the number of iterations in a block (but not enough to inflate the bound), and decreases the total time spent computing approximations to max mean cuts from $O(n^7 h)$ to $O(n^5 h)$, so that this is no longer a bottleneck.

6.2 Proximity Lemmas

In this section we prove the proximity lemmas which enable us to show that after a large enough decrease in δ, the sign of the reduced cost of some arc in the optimal solution is determined. Since the number of phases required to obtain this sufficiently large decrease is strongly polynomial, and the run time of each phase is strongly polynomial, this implies a strongly polynomial algorithm.

Lemma 6.1 (Primal Proximity) *Suppose that $(\xi = (\varphi, \psi), y, p)$ is δ-optimal. Then there exists an optimal flow φ^* to the original instance such that*
$\langle$a$\rangle$ $|\varphi(a) - \varphi^*(a)| \le 3n^2\delta$ *for all $a \in A$, and*
$\langle$b$\rangle$ $|\partial\varphi^*(X) - y(X)| \le 3n^3\delta$ *for any cut X.*

Proof. First we convert ξ to a flow on A exactly satisfying complementary slackness. We do this by saturating all residual A arcs with $c_p(a) < 0$; since all such arcs have $r(a) < \delta$, this increases the discrepancy Φ by at most $2m\delta \le 2n(n-1)\delta$. Then we delete all the D arcs and their flows ψ. Since such arcs have flow at most δ, this increases Φ by at most $n(n-1)\delta$. Since we started out with $\Phi \le 2n\delta$, we now have $\Phi \le (3n^2 - n)\delta$. Call the resulting flow φ'. Note that $|\varphi'(a) - \varphi(a)| \le \delta$ for all $a \in A$. For any set X, saturating negative reduced cost arcs in A and removing flow ψ from D implies that $|\partial_I \xi(X) - \partial_A \varphi'(X)| \le \frac{n^2}{2}\delta$. Since we had $|\partial\varphi(X) - y(X) + \partial\psi(X)| \le \Phi \le 2n\delta$, this implies that $|\partial_A \varphi'(X) - y(X)| \le n^2\delta$ for all $X \subset V$.

Now use SSP to convert φ' into an optimal flow φ^*. (As originally stated, SSP is finite only for integer data, but the lexicographic techniques of Schönsleben [30] and Lawler and Martel [25] show that it can be made finite for any data.) Since $\Phi \le (3n^2 - n)\delta$, the total amount of flow pushed by SSP is at most $(3n^2 - n)\delta$. This implies that for each $a \in A$, $|\varphi'(a) - \varphi^*(a)| \le (3n^2 - n)\delta$. Since $|\varphi'(a) - \varphi(a)| \le \delta$ we get $|\varphi(a) - \varphi^*(a)| \le 3n^2\delta$, verifying $\langle$a$\rangle$.

Suppose that P is an augmenting path chosen by the algorithm, and that flow is augmented by amount τ_P along P. By transitivity (Lemma 2.4), P contains at most $n/2$ arcs in E_y. Thus the flow boundary of any $X \subset V$ changes by at most $\frac{n}{2}\tau_P$ due to P. Since $\sum_P \tau_P \le (3n^2 - n)\delta$, we have that $|\partial_A \varphi'(X) - \partial_A \varphi^*(X)| \le \frac{3n^3}{2}\delta$. Since $|\partial_A \varphi'(X) - y(X)| \le n^2\delta$, this implies that $|y(X) - \partial_A \varphi^*(X)| \le 3n^3\delta$, verifying $\langle$b$\rangle$. ■

Corollary 6.2 *Suppose that (ξ, y, p) is δ-optimal. Then for all optimal prices p^*,*
$\langle$a1$\rangle$ *if there is an arc $a \in A$ with $\varphi(a) < u(a) - 3n^2\delta$, then $c_{p^*}(a) \ge 0$;*
$\langle$a2$\rangle$ *if there is an arc $a \in A$ with $\varphi(a) > l(a) + 3n^2\delta$, then $c_{p^*}(a) \le 0$;*
$\langle$b$\rangle$ *if there is a pair (w, v) with $\sigma(y, v, w) > 3n^3\delta$, then $p^*(v) \le p^*(w)$.*

Proof. For $\langle$a1$\rangle$, $\varphi(a) < u(a) - 3n^2\delta$ and Lemma 6.1 $\langle$a$\rangle$ imply that $\varphi^*(a) < u(a)$. Then complementary slackness implies that $c_{p^*}(a) \ge 0$ for all optimal prices p^*. A symmetric argument implies $\langle$a2$\rangle$.

For $\langle$b$\rangle$, suppose that X^* determines $\sigma(\partial\varphi^*, v, w)$, so that $\sigma(\partial\varphi^*, v, w) = f(X^*) - \partial\varphi^*(X^*)$. Then $\sigma(y, v, w) > 3n^3\delta$ implies that $f(X^*) - y(X^*) > 3n^3\delta$, and Lemma 6.1 $\langle$b$\rangle$ implies that $\sigma(\partial\varphi^*, v, w) = f(X^*) - \partial\varphi^*(X^*) > 0$. Then Theorem 3.2 implies $\langle$b$\rangle$. ■

Let p° be the prices at the beginning of a block. We first compute a most positive cut T°. If $\kappa_p(T^\circ) \le 0$, then we are optimal. Otherwise, $\overline{\kappa}_p(T^\circ) := \delta^\circ > 0$. Since $\delta^\circ = \overline{\kappa}_{p^\circ}(T^\circ) \ge \overline{\kappa}_{p^\circ}/n$, we can compute a feasible submodular flow $\xi^\circ = (\varphi^\circ, \psi^\circ)$ in the network with data l_{p°, u_{p°, f_{p°, using $\delta = n\delta^\circ$. With $y^\circ = \partial\varphi^\circ$ and $x^\circ = y^\circ - \partial\psi^\circ$, we have that ξ°, y°, and p° are almost $n\delta^\circ$-optimal, except that the discrepancy $\Phi = \sum_v |x^\circ(v) - \partial\varphi^\circ(v)| = \sum_v |\psi^\circ(v)|$ could be as large as $n^2(n\delta^\circ)$. So we start out our block of $\mathrm{O}(\log n)$ phases with $\delta = n^2\delta^\circ$ and these values. Let ξ', y', and p' be the corresponding values that are δ'-optimal after the last phase in the block. We first need a lemma on the proximity of f_{p° and f.

Theorem 6.4 below will show that at the end of each block, there is some $a \in \Delta_A^+ T^\circ$ satisfying Corollary 6.2 $\langle$a1$\rangle$, or some $a \in \Delta_A^- T^\circ$ satisfying $\langle$a2$\rangle$, or some $a = (w, v)$ and a level set L° of p° with $v \in L^\circ$, $w \notin L^\circ$ satisfying $\langle$b$\rangle$. We restrict the sign of a's reduced cost as follows: In case $\langle$a1$\rangle$, replace $u(a)$ by $+\infty$. In case $\langle$a2$\rangle$, replace $l(a)$ by $-\infty$. In case $\langle$b$\rangle$, add a new arc $\widehat{a} = (w, v)$ with $l(\widehat{a}) = 0$, $u(\widehat{a}) = \infty$, and $c(\widehat{a}) = 0$. Since an optimal submodular flow φ^* for the original problem remains optimal (with $\varphi^*(\widehat{a}) = 0$), these changes do not affect the set of optimal prices. We first need a lemma on the proximity of f_{p° and f.

Lemma 6.3 *Suppose that we have scalar $\gamma > 0$, $y' \in \mathrm{B}(f)$ and prices p° such that for all level sets L of p°, and all (w, v) with $v \in L$ and $w \notin L$ we have $\sigma(y', v, w) \le \gamma$. Then for any T°, $y'(T^\circ) - f_{p^\circ}(T^\circ) \le n^3\gamma$.*

Proof. First, $\sigma(y', v, w) \le \gamma$ implies that there exists a set X_{wv} with $v \in X_{wv}$, $w \notin X_{wv}$, and $y'(X_{wv}) \ge f(X_{vw}) - \gamma$. Since $L = \bigcup_{v \in L} \bigcap_{w \notin L} X_{wv}$, by submodularity of $f - y'$, $y' \in \mathrm{B}(f)$, and $f(V) = 0$, we get $y'(L) \ge f(L) - b(L)\gamma \ge f(L) - n^2\gamma$. Then modularity of y' and $y' \in \mathrm{B}(f)$ imply

$$\begin{aligned} y'(T^\circ) &= \sum_{i=1}^{k} [y'((T^\circ \cap L_i) \cup L_{i-1}) - y'(L_{i-1})] \\ &\le \sum_{i=1}^{k} [f((T^\circ \cap L_i) \cup L_{i-1}) - (f(L_{i-1}) - n^2\gamma)] \le f_{p^\circ}(T^\circ) + n^3\gamma. \end{aligned}$$

■

Theorem 6.4 *After a block of $\lceil \log_2(5n^8) \rceil = \mathrm{O}(\log n)$ scaling phases, at least one of the following holds:*

(a1) $\exists\, a \in \Delta_A^+ T^\circ$ *with* $l_{p^\circ}(a) - \varphi'(a) > 3n^2\delta'$,

(a2) $\exists\, a \in \Delta_A^- T^\circ$ *with* $\varphi'(a) - u_{p^\circ}(a) > 3n^2\delta'$,

(b) $\exists\, (w, v)$ *with* $\sigma(y', v, w) > 3n^3\delta'$ *and with* $v \in L^\circ$, $w \notin L^\circ$ *for some level set* L° *of* p°.

The reduced cost sign restriction resulting from applying Corollary 6.2 *is new.*

Proof. Since each scaling phase cuts δ in half,

$$\delta' \le n^2\delta^\circ/5n^8 = \delta^\circ/5n^6. \tag{4}$$

Conservation of flow for ξ' implies that $\xi'(\Delta_I^+ T^\circ) - \xi'(\Delta_I^- T^\circ) - \partial_I\xi'(T^\circ) = 0$. Subtracting this from $\kappa_{p^\circ}(T^\circ)$ yields

$$\delta^\circ b(T^\circ) = \kappa_{p^\circ}(T^\circ) = (l_{p^\circ} - \xi')(\Delta_I^+ T^\circ) + (\xi' - u_{p^\circ})(\Delta_I^- T^\circ) + (\partial_I\xi'(T^\circ) - f_{p^\circ}(T^\circ)). \tag{5}$$

If none of (a1), (a2), or (b) holds, then from (5), the bound on Φ, and Lemma 6.3 with $\gamma = 3n^3\delta'$ (noting that for $a \in D$, $|l_{p^\circ}(a) - \psi'(a)| \leq \delta'$ and $|\psi'(a) - u_{p^\circ}(a)| \leq \delta'$), we get $\delta^\circ < \delta^\circ b(T^\circ) = \kappa_{p^\circ}(T^\circ) \leq 3n^2|\Delta_A(T^\circ)|\delta' + n^2\delta' + \partial_I\xi'(T^\circ) - y'(T^\circ) + y'(T^\circ) - f_{p^\circ}(T^\circ) \leq 4n^4\delta' + 2n\delta' + 3n^6\delta' \leq 5n^6\delta'$, contradicting (4).

In case (a1), we must have that $l_{p^\circ}(a) = u(a)$, implying that $c_{p^\circ}(a) < 0$. Thus $\varphi'(a) < u(a) - 3n^2\delta'$, and from Corollary 6.2 $\langle$a1$\rangle$ we can conclude that $c_{p^*}(a) \geq 0$. Since we had $c_{p^\circ}(a) < 0$, this is a new sign restriction on c_p. Symmetric arguments show that in case (a2), $c_{p^*}(a) \leq 0$ is a new sign restriction on c_p.

Finally, in case (b), from Corollary 6.2 $\langle$b$\rangle$ we have $p^*(v) \leq p^*(w)$ for all optimal p^*. Since L° is a level set of p°, we had $p^\circ(v) > p^\circ(w)$, so this is a new sign restriction on p. ∎

Theorem 6.5 *The algorithm described in this section computes a minimum cost submodular flow in* $\mathrm{O}(n^6 h \log n)$ *time.*

Proof. Each block imposes at least one new sign restriction on $c_{p^*}(a)$ for some $a \in I$. At most $2m'$ such sign restrictions can be imposed before p^* is completely determined, so after at most $2m' = \mathrm{O}(n^2)$ blocks we know the optimal p^*. Each block requires $\mathrm{O}(\log n)$ scaling phases, each of which takes $\mathrm{O}(n^4 h)$ time. The time to compute a most positive cut, initial ξ°, and final submodular flow given p^* is $\mathrm{O}(n^3 h)$, which is not a bottleneck. ∎

The strongly polynomial algorithm can be modified to work for crossing submodular functions: In the definition of the cut value $\kappa_p(X)$, the last term $f_p(X)$ should be replaced by $\widetilde{f}_p(X)$, where $\widetilde{f}$ is the fully submodular function with $\mathrm{B}(f) = \mathrm{B}(\widetilde{f})$. An implementation of Fujishige and Zhang's algorithm [24] finds a most positive cut for the new definition of κ_p. The algorithm does not need the function values of $\widetilde{f}$, which is used only for its analysis.

Acknowledgments

We would like to thank the IPCO VII Organizing Committee for helping to arrange the opportunity for us to work together in Graz, Maiko Shigeno for many valuable conversations and discussions on related topics, and a referee for prompting us to consider the crossing submodular function case.

References

[1] G. Birkhoff. *Lattice Theory.* Amer. Math. Soc., 1967.

[2] W. H. Cunningham and A. Frank. A primal-dual algorithm for submodular flows. *Math. Oper. Res.*, 10 (1985), 251–262, .

[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, 1970, 69–87.

[4] J. Edmonds and R. Giles. A min-max relation for submodular functions on graphs. *Ann. Discrete Math.*, 1 (1977), 185–204, .

[5] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19 (1972), 248–264, .

[6] T. R. Ervolina and S. T. McCormick. Two strongly polynomial cut canceling algorithms for minimum cost network flow. *Discrete Appl. Math.*, 46 (1993) 133–165.

[7] L. Fleischer and S. Iwata. Improved algorithms for submodular function minimization and submodular flow. *Proceedings of the 32th Annual ACM Symposium on Theory of Computing*, 107–116.

[8] A. Frank. Finding feasible vectors of Edmonds–Giles polyhedra. *J. Combin. Theory*, 36 (1984), 221–239, .

[9] A. Frank and É. Tardos. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica*, 7 (1987), 49–65.

[10] A. Frank and É. Tardos. Generalized polymatroids and submodular flows. *Math. Programming*, 42 (1988), 489–563.

[11] S. Fujishige. Algorithms for solving the independent-flow problems, *J. Oper. Res. Soc. Japan*, 21 (1978), 189–204.

[12] S. Fujishige. Structures of polyhedra determined by submodular functions on crossing families. *Math. Programming*, 29 (1984), 125–141.

[13] S. Fujishige. A capacity-rounding algorithm for the minimum cost circulation problem — A dual framework of the Tardos algorithm. *Math. Programming*, 35 (1986), 298–308.

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

[15] S. Fujishige and S. Iwata: Algorithms for submodular flows, *IEICE Trans. Inform. Syst.*, E83-D (2000), 322–329.

[16] S. Fujishige, H. Röck, and U. Zimmermann. A strongly polynomial algorithm for minimum cost submodular flow problems. *Math. Oper. Res.*, 14 (1989), 60–69.

[17] S. Fujishige and N. Tomizawa. A note on submodular functions on distributive lattices. *J. Oper. Res. Soc. Japan*, 26 (1983), 309–318.

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

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

[20] S. Iwata. A fully combinatorial algorithm for submodular function minimization. METR 00-09, University of Tokyo, October 2000. To appear in *J. Combin. Theory Ser. B.*

[21] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial, strongly polynomial-time algorithm for minimizing submodular functions. *Proceedings of the 32th Annual ACM Symposium on Theory of Computing* (2000), 97–106. To appear in *J. ACM.*

[22] S. Iwata, S. T. McCormick, and M. Shigeno. A faster algorithm for minimum cost submodular flows. *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms* (1998), 167–174. To appear in *Combinatorica.*

[23] 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.

[24] S. Iwata, S. T. McCormick, and M. Shigeno. A fast cost scaling algorithm for submodular flow. *Inform. Process. Lett.*, 74 (2000), 123–128.

[25] E. L. Lawler and C. U. Martel. Computing maximal polymatroidal network flows. *Math. Oper. Res.*, 7 (1982), 334–347.

[26] C. L. Lucchesi and D. H. Younger. A minimax relation for directed graphs. *J. London Math. Soc.*, 17 (1978), 369–374.

[27] S. T. McCormick and T. R. Ervolina. Computing maximum mean cuts. *Discrete Appl. Math.*, 52 (1994), 53–70.

[28] T. Naitoh and S. Fujishige. A note on the Frank–Tardos bi-truncation algorithm for crossing-submodular functions. *Math. Programming*, 53 (1992), 361–363.

[29] T. Radzik. Newton's method for fractional combinatorial optimization. *Proceedings of the 33rd IEEE Annual Symposium on Foundations of Computer Science* (1992), 659–669; Parametric flows, weighted means of cuts, and fractional combinatorial optimization. *Complexity in Numerical Optimization*, P. Pardalos, ed., World Scientific, 1993, 351–386.

[30] P. Schönsleben. *Ganzzahlige Polymatroid-Intersektions Algorithmen.* Dissertation, ETH Zürich, 1980.

[31] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Combin. Theory Ser. B* 80 (2000), no. 2, 346-355.

[32] M. Shigeno, S. Iwata, and S.T. McCormick. Relaxed most negative cycle and most positive cut canceling algorithms for minimum cost flow. *Math. Oper. Res.*, 25 (2000), 76–104.

[33] É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5 (1985), 247–256.

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

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