Improved Algorithms for Submodular Function Minimization and Submodular Flow

Lisa Fleischer *
Dept. of Ind. Eng. & Oper. Res.
Columbia University
New York, NY 10027, USA
lisa@ieor.columbia.edu

Satoru Iwata †
Grad. School of Eng. Science
Osaka University
Toyonaka, Osaka 560-8531, Japan
iwata@sys.es.osaka-u.ac.jp

Abstract

Very recently, two groups of researchers independently developed the first combinatorial, strongly polynomial-time algorithms for submodular function minimization (Iwata, Fleischer, Fujishige; and Schrijver). In this paper, we improve on these algorithms and show that the ideas generated in the design of these algorithms are helpful in other contexts. This work demonstrates one use of combinatorial algorithms for submodular function minimization.

In particular we accomplish three things. First, we improve the complexity of Schrijver's algorithm by designing a push-relabel algorithm for submodular function minimization (SFM). Second, we exploit the common structure shared between submodular function minimization and maximum submodular flow to design the first algorithm for maximum submodular flow that does not depend on an oracle for SFM. The overall time complexity is the same as for SFM. Finally, we design the first algorithms for minimum cost submodular flow that do not depend on an oracle for SFM, using the framework of submodular function minimization of Iwata, Fleischer, Fujishige. We show that optimal dual solutions can be computed in the same time as SFM, and that optimal primal solutions can thus be obtained with one additional maximum submodular flow computation. We give both weakly and strongly polynomial versions.

*Part of this work done while on leave at the Fields Institute, Toronto, Canada. Partially supported by NSF grants INT-9902663 and EIA-9973858.

†A part of this work is done while on leave at the Fields Institute, Toronto, Canada. Partly supported by Grants-in-Aid for Scientific Research from Ministry of Education, Science, Sports, and Culture of Japan.

STOC 2000 Portland Oregon USA

1. Introduction

A function f defined on all the subsets of a 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, matroid rank functions, and entropy functions. *Submodular function minimization (SFM)* is the problem of finding a subset $X \subseteq V$ with $f(X) \leq f(Y)$ for all $Y \subseteq V$.

Connecting submodular functions with network flows, Edmonds and Giles [4] introduced the submodular flow problem, which includes network flow, matroid intersection, and directed cut covering. Since then, several combinatorial optimization problems have been shown to be special cases of submodular flow. In particular, Frank and Tardos [7] solved the minimum cost rooted vertex connectivity augmentation problem in directed graphs by reducing it to minimum cost submodular flow. A recent paper of Jordán [14] also reduces a simultaneous edge-connectivity augmentation problem in undirected graphs to the submodular intersection problem, which is equivalent to maximum submodular flow.

A number of network flow algorithms have been extended to submodular flow problems. All these algorithms rely on an oracle for finding the minimizer of a given submodular function. The best known time complexity in this framework is $O(n^3h)$ for finding a feasible submodular flow [9] and $O(n^4h \min\{\log U, \log C, n^2 \log n\})$ for solving the minimum cost submodular flow problem [5, 8, 13], where h is the time required for SFM, U is an upper bound on the absolute value of the arc capacities and function values, C is the maximum absolute value of the arc costs, and all input numbers are integers.

The first polynomial time algorithm for SFM was introduced in [11] and uses the ellipsoid method. The ellipsoid method is well-known for its use in establishing the polynomial time equivalence of separation and optimization for problems in combinatorial optimization. While many optimization problems were shown to be polynomially solv-

able using separation implies optimization, the optimization problem for submodular function polyhedra is solvable by the greedy algorithm [3]. SFM is the harder-to-solve separation problem. For almost two decades, optimization implies separation via the ellipsoid method gave the only polynomial time algorithm for SFM. In the interim, researchers achieved combinatorial, strongly polynomial-time algorithms for special cases, including Cunningham's algorithm for testing membership in matroid polyhedra [1], and Queyranne's algorithm for minimizing symmetric submodular functions [15].

Very recently, two groups independently devised combinatorial, strongly polynomial-time algorithms for general submodular function minimization. Both of these algorithms are based on Cunningham's approach [1, 2] to design an augmenting path algorithm for SFM. Let γ be the time to evaluate f on one set. Schrijver [16] describes an algorithm that runs in $O(n^9 + \gamma n^8)$. Iwata, Fleischer, Fujishige [12] describe a strongly polynomial time algorithm that runs in $O(\gamma n^7 \log n)$, and a weakly polynomial time algorithm that runs in $O(\gamma n^5 \log M)$ time, where M is an upperbound on the maximum function value, assuming all function values are integer.

Our Contributions

First,we present a faster strongly polynomial-time algorithm for SFM. The new algorithm exploits a subroutine devised in [16]. We reduce the number of subroutine calls by a factor of n by embedding the subroutine of [16] in a push-relabel framework for submodular intersection developed by Fujishige and Zhang [9]. The resulting algorithm runs in $O(n^8 + \gamma n^7)$ time. If a function evaluation takes at least linear time, then this is the fastest strongly polynomial algorithm for SFM.

Next, we show that this algorithm can be modified to solve a more general problem in the same time bound: We describe the first algorithm for maximum submodular flow that does not require an oracle for SFM. Instead, we modify combinatorial algorithms for SFM to design a more direct, strongly polynomial algorithm for this problem.

Finally, we describe the first algorithms for solving minimum cost submodular flow that do not require an oracle for SFM. We present both weakly and strongly polynomial time algorithms. The design of these combinatorial algorithms builds on ideas in [5, 12]. Our algorithm computes optimal dual node prices in the same time as SFM. Then, an optimal flow can be computed either with one maximum submodular flow, or m iterations of the price finding algorithm.

2. Preliminaries

We 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 $= 0$ otherwise.

Throughout this paper, we assume that $f(\emptyset) = 0$. The *base polyhedron* of f is defined as

$$\mathrm{B}(f) := \left\{ x \;\middle|\; \begin{array}{l} x \in \mathbf{R}^V,\ x(V) = f(V) \\ x(X) \leq f(X), \forall\, X \subset V \end{array} \right\}$$

A vector $x \in \mathrm{B}(f)$ is called a *base*. An *extreme base* is an extreme point of $\mathrm{B}(f)$. A fundamental step in algorithms for SFM and submodular flow is to move from one base x to another x' via an *exchange operation*: $x' := x + \alpha(\chi_v - \chi_w)$. The maximum possible exchange α allowable is called the *exchange capacity*, and is defined as

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

It is not hard to see that this is equivalent to

$$\sigma(x, v, w) := \min\{f(X) - x(X) \mid v \in X \subseteq V \backslash \{w\}\}.$$

For a given base x, we define $A_x := \{(w, v) | \sigma(x, v, w) > 0\}$. We call $(w, v) \in A_x$ an *exchange arc*. Computing exchange capacities in general is as hard as SFM. However, if y is an extreme base, then exchange capacities can be computed with one function evaluation for special vertex pairs (w, v), as follows.

Let $L = (v_1, \cdots, v_n)$ be a *linear ordering* of V. The greedy algorithm [3] *generates* an extreme base y by setting $L(v_h) := \{v_1, \cdots, v_h\}$ and $y(v_h) = f(L(v_h)) - f(L_{h-1})$ for each h. Edmonds showed that every extreme base is generated by the greedy algorithm applied to some linear ordering. Note that a linear ordering L generates base x if and only if $x(L(v_h)) = f(L(v_h))$ for all $h = 1, 2, \ldots, n$. Any set X with $x(X) = f(X)$ is called *x-tight*. Note that a set X is x-tight if and only if there are no arcs in A_x that enter X. The following lemma follows from the greedy algorithm and the definition of exchange capacity.

Lemma 2.1: *Let L be a linear ordering of V in which w immediately follows v and that generates an extreme base $y \in \mathrm{B}(f)$. Let L' be the linear ordering obtained by interchanging w and v. Then the extreme base y' generated by L' satisfies $y' = y + \sigma(y, v, w)(\chi_v - \chi_w)$ with*

$$\sigma(y, v, w) = f(L(v) \backslash \{w\}) - y(L(v) \backslash \{w\}). \qquad (2.1)$$

For an extreme base y, we denote $w \preceq_y v$ if w belongs to every y-tight set containing v. Note that $w \preceq_y v$ implies w precedes v in the linear ordering generating y. Then $\preceq_y$ is a partial order on V. If $w \preceq_y v$, then $\sigma(y, v, w) > 0$. For $s, t \in V$, we call $[s, t]_y = \{v \mid s \preceq_y v \preceq_y t\}$ the *interval* between s and t. If $s \not\preceq_y t$, the interval is empty by definition. The *Hasse diagram* of $\preceq_y$ is a directed acyclic graph on V whose arc set consists of pairs (w, v) of distinct vertices such that $[w, v]_y = \{w, v\}$.

3. Submodular Function Minimization

The following dual characterization of a minimizer of a submodular function follows from a min-max theorem on the vector reduction of a polymatroid due to Edmonds [3]. For $x \in \mathbf{R}^V$ define x^- by $x^-(v) := \min\{0, x(v)\}$ for $v \in V$. Then Edmonds' theorem implies

$$\max\{x^-(V) \mid x \in \mathrm{B}(f)\} = \min\{f(X) \mid X \subseteq V\}. \quad (3.1)$$

This result can also be derived from LP strong duality. This characterization has driven most searches for combinatorial algorithms for SFM.

It is not necessarily true that the base achieving the maximum in 3.1 is an extreme base. Thus, in order to apply Lemma 2.1, Cunningham [1, 2] chose to represent a base $x \in \mathrm{B}(f)$ as a convex combination of extreme bases: $x = \sum_i \lambda_i y_i$, $\lambda_i \geq 0$, $\sum_i \lambda_i = 1$, $y_i \in \mathrm{B}(f)$. This idea is also used in the recent strongly polynomial time algorithms for SFM, and in this current paper.

Roughly speaking, Cunningham uses this in an augmenting path framework which seeks to increase $x^-(V)$ by augmenting from vertices v with $x(v) < 0$ to vertices u with $x(u) > 0$ along paths of arcs in the union of Hasse diagrams of y_i, $i = 1, 2, \ldots, n$. He obtains a pseudopolynomial $O(n^6 M \log(nM))$ time algorithm for SFM.

A major difference between the recent combinatorial, polynomial time algorithm of Schrijver [16] and Cunningham's algorithm [2], is that Schrijver maintains a directed graph whose arc set is given by $\widehat{A}_x = \{(s,t) \mid \exists i \in I,\ s \preceq_{y_i} t\}$, while Cunningham's algorithm uses only the arcs of the Hasse diagrams. Another difference is that Schrijver's algorithm does not perform augmentation along a path. It constructs a layered network to detect a shortest augmenting path and applies an exchange operation only to the last arc of a shortest augmenting path.

Instead of computing the exchange capacity, Schrijver devises the following subroutine that computes an amount of exchange that is sufficient to eliminate the arc from $\widehat{A}_x$.

Reduce-Interval(b, s, t)

Input: An extreme base $b \in \mathrm{B}(f)$ and distinct $s, t \in V$ such that $s \preceq_b t$.

Output: A positive constant μ and a decomposition of $b + \mu(\chi_t - \chi_s)$ as a convex combination of extreme bases $y \in \mathrm{B}(f)$ such that $[s,t]_y \subset [s,t]_b$.

This subroutine works as follows. Let $L = (v_1, \cdots, v_n)$ be a linear extension of $\preceq_b$ such that $\{v_p, \cdots, v_q\} = [s,t]_b$. Namely, $v_p = s$ and $v_q = t$. For each $r = p+1, \ldots, q$, compute the extreme base y_r generated by $L_r = (v_1, \cdots, v_{p-1}, v_r, v_p, \cdots, v_{r-1}, v_{r+1}, \cdots, v_n)$. Determine $\eta_{p+1}, \cdots, \eta_q \geq 0$ such that

$$\chi_t - \chi_s = \sum_{r=p+1}^{q} \eta_r (y_r - b).$$

Put $\eta = \sum_{r=p+1}^{q} \eta_r$ and $\mu = 1/\eta$. Then

$$b + \mu(\chi_t - \chi_s) = \sum_{r=p+1}^{q} \frac{\eta_r}{\eta} y_r$$

holds, and each y_r satisfies $[s,t]_{y_r} \subset [s,t]_b$. Thus this subroutine runs in $O(n^2\gamma)$ time.

The above algorithm of Schrijver[16] minimizes f by calling the subroutine $O(n^6)$ times. We will present another algorithm that calls it $O(n^5)$ times.

A Push-Relabel Algorithm for SFM

We now describe the push-relabel algorithm for SFM. The push-relabel approach was introduced for network flows by Goldberg and Tarjan [10], and is among the most efficient known algorithms for maximum flow. It has been applied to polymatroid intersection, a problem equivalent to maximum submodular flow, by Fujishige and Zhang [9].

The algorithm maintains $x \in \mathrm{B}(f)$ as a convex combination $x = \sum_{i \in I} \lambda_i y_i$ of extreme bases y_i and a directed graph $(V, \widehat{A}_x)$. We start with an extreme base $x \in \mathrm{B}(f)$ obtained by the greedy algorithm [3]. Let $S := \{s \mid s \in V,\ x(s) > 0\}$ and $T := \{t \mid t \in V,\ x(t) < 0\}$. The labeling $d : V \to \mathbf{Z}$ is *valid* if it satisfies $d(t) = 0$ for $t \in T$ and $d(s) \leq d(t) + 1$ for all $(s,t) \in \widehat{A}_x$. The algorithm maintains a valid labeling. Initially, $d(s) = 0$ for $s \in V$, which is clearly valid. Note that a valid distance labeling d serves as a lower bound on the minimum number of arcs from s to T. For a valid labeling d, we define $Q := \{s \mid s \in S,\ d(s) < n\}$.

The algorithm consists of two basic operations. Operation Push(s,t) applies if $s \in Q$, $(s,t) \in \widehat{A}_x$ and $d(s) = d(t) + 1$. Select $k \in I$ with the largest interval $[s,t]_{y_k}$, and apply the subroutine Reduce-Interval(y_k, s, t) to get $\mu > 0$ and a convex decomposition $\sum_{j \in J} \kappa_j y_j$ of $y_k + \mu(\chi_t - \chi_s)$. Update $x := x + \varepsilon(\chi_t - \chi_s)$ with $\varepsilon = \min\{x(s), \lambda_k \mu\}$. Putting $I := I \cup J$, $\lambda_k := \lambda_k - \varepsilon/\mu$, and $\lambda_j := \lambda_k \kappa_j$ for $j \in J$, we obtain a convex combination $x = \sum_{i \in I} \lambda_i y_i$. By a standard linear programming technique, reduce the number of positive coefficients in this expression to at most n in $O(n^3)$ time, and then delete those indices with zero coefficients from I. The operation Push(s,t) iterates this until $x(s) = 0$ or $(s,t) \notin \widehat{A}_x$. If $(s,t) \notin \widehat{A}_x$, we call Push$(s,t)$ *saturating*, and otherwise *nonsaturating*. In each iteration, the maximum size of the intervals $[s,t]_{y_i}$ decreases or the number of extreme bases that attain the maximum decreases. Thus Push(s,t) performs at most $O(n^2)$ iterations.

Operation Relabel(s) applies if $s \in Q$ and $d(s) \leq d(t)$ for every $(s,t) \in \widehat{A}_x$. It updates $d(s) := d(s) + 1$. Clearly, $d(s) \leq n$ holds for $s \in V$ throughout the algorithm.

The algorithm fixes an arbitrary total order $\leq$ on the vertices. The algorithm repeatedly selects a vertex $s \in Q$ with highest $d(s)$ to apply a procedure Scan(s). The procedure Scan(s) repeatedly picks a vertex $t \in V$ in the total order and applies Push(s,t) if possible, until $x(s) = 0$ or it has examined every $t \in V$. If Scan(s) ends with a non-saturating Push(s,t), the next time 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 Scan(s) for each $s \in V$. The algorithm increments $\tau(s)$ if it performs a saturating Push$(s,\tau(s))$ or it finds Push$(s,\tau(s))$ is not applicable. If $\tau(s)$ is the last vertex in $\leq$, then the algorithm performs Relabel(s) and resets $\tau(s)$ to be the first vertex in $\leq$.

The algorithm terminates when either Q or T is empty. If $Q = \emptyset$, let W denote the set of vertices from which there is a directed path to T. Then $x(s) \leq 0$ for $s \in W$ and $x(s) \geq 0$ for $s \in V \backslash W$. This implies $x^-(V) = x(W)$. Since no arc in $\widehat{A}_x$ enters W, we have $y_i(W) = f(W)$ for every $i \in I$, which implies $x(W) = f(W)$. Thus W is a minimizer of f. If $T = \emptyset$, then $f(X) \geq x(X) \geq 0$ holds for every $X \subseteq V$, which means $\emptyset$ is a minimizer of f.

To establish the correctness and complexity of the algorithm, we require the following technical lemma adapted from Schönsleben (1980). This lemma also highlights the additional difficulty of working with $\widehat{A}_x$. Namely, arcs in $\widehat{A}_x$ may appear, disappear, or change capacity when operations are applied to completely disjoint arcs in $\widehat{A}_x$. This extra complication does not arise in traditional network flows.

Lemma 3.1: *Let $z+\mu(\chi_t-\chi_s) = \sum_{j\in J} \kappa_j y_j$ be the convex combination obtained by the subroutine, where $\kappa_j > 0$ for $j \in J$. If $u \not\preceq_z v$ and $u \preceq_{y_j} v$ for some $j \in J$, then $u \preceq_z t$ and $s \preceq_z v$.*

Proof. Since $u \not\preceq_z v$, there must be an $X \subseteq V$ that satisfies $v \in X$, $u \notin X$, and $y(X) = f(X)$. If $u \not\preceq_z t$, then there exists an Y such that $t \in Y$, $u \notin Y$, and $z(Y) = f(Y)$. It follows from $t \in X \cup Y$ and the submodularity of f that $y_j(X \cup Y) = z(X \cup Y) = f(X \cup Y)$ holds for any $j \in J$, which together with $v \in X \cup Y$ and $u \notin X \cup Y$ contradicts to $u \preceq_{y_j} v$. Similarly, if $s \not\preceq_z v$, then there exists a $Z \subseteq V$ such that $v \in Z$, $s \notin Z$, and $z(Z) = f(Z)$. It follows from the submodularity of f and $s \notin X \cap Z$ that $y_j(X \cap Z) = z(X \cap Z) = f(X \cap Z)$ holds for any $j \in J$, which together with $v \in X \cap Z$ and $u \notin X \cap Z$ contradicts to $u \preceq_{y_j} v$. ■

Lemma 3.2: *The operations* Push *and* Relabel *maintain valid d.*

Proof. At start d is valid. The operation Relabel, if applicable, maintains valid d. Suppose d is valid before Reduce-Interval(y_k,s,t) introduces a new arc (u,v) to $\widehat{A}_x$. In this case, $u \not\preceq_{y_k} v$ and $u \preceq_{y_j} v$ and for some $j \in J$. Lemma 3.1 then implies that $u \preceq_{y_k} t$ and $s \preceq_{y_k} v$. That is, (u,t) and (s,v) belonged to $\widehat{A}_x$ before Reduce-Interval(y_k,s,t). Since d is valid before Reduce-Interval(y_k,s,t), we have $d(u) \leq d(t) + 1$ and $d(s) \leq d(v) + 1$. Since Push(s,t) applies, $d(s) = d(t) + 1$, which implies $d(u) \leq d(v) + 1$. Thus the push operation maintains valid labels d,and hence d remains valid throughout the algorithm. ■

Since $d(s) \leq n$ for every $s \in V$, the algorithm performs at most n^2 relabel operations in total. Lemma 3.3 below ensures that Relabel(s) is applicable when the algorithm resets $\tau(s)$ in Scan(s). Therefore the algorithm performs at most n^3 saturating pushes.

Lemma 3.3: *If $v < \tau(u)$ and $(u,v) \in \widehat{A}_x$, then $d(u) \leq d(v)$.*

Proof. Suppose the statement holds before a call to Reduce-Interval(y_k,s,t) introduces a new arc $(u,v) \in \widehat{A}_x$. It follows from Lemma 3.1 that $u \preceq_{y_k} t$ and $s \preceq_{y_k} v$. By the validity of d, we have $d(u) \leq d(t)+1$ and $d(s) \leq d(v)+1$. By the applicability of Push(s,t), we also have $d(s) = d(t)+1$. If $t < \tau(u)$, then $d(u) \leq d(t)$. On the other hand, if $t \geq \tau(u) > v$, then $d(s) \leq d(v)$. In either case, we have $d(u) \leq d(v)$. ■

Lemma 3.4: *Between a non-saturating* Push(s,t) *and the next* Scan(s)*, the algorithm performs* Relabel(u) *for some $u \in V$.*

Proof. As a consequence of Push(s,t), we have $x(s) = 0$. Before applying Scan(s) again, the algorithm must increase $x(s)$ by 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 somewhere before Push(v,s). ■

Lemma 3.4 implies that the number of non-saturating pushes is also at most n^3. The proof is straightforward. Thus the algorithm performs $O(n^2)$ relabel and $O(n^3)$ push operations. Since each push operation calls the subroutine $O(n^2)$ times, the algorithm calls it $O(n^5)$ times in total. Therefore, the push-relabel algorithm runs in $O(n^7\gamma + n^8)$ time.

4. Feasible Submodular Flow

In this section, we give the first combinatorial algorithm for maximum submodular flow that does not call an oracle for SFM. The best known algorithm runs in time $O(n^3h)$ where h is the time required by the SFM oracle [9]. We show how to solve this in the same time as the SFM algorithm in the preceding section, by modifying the polymatroid intersection algorithm of Fujishige and Zhang [9]. Our algorithm replaces each call that their algorithm makes to an SFM oracle with n calls to Reduce-Interval. The resulting algorithm looks very similar to our SFM algorithm in the preceding section, and could easily be interpreted as a modification of that algorithm. We begin by describing how to find a feasible submodular flow.

Let $G = (V, E)$ be a directed graph with lower and upper bounds $l \leq u$ on the flow values on arcs. For a flow φ in G, its boundary $\partial\varphi$ is defined by $\partial\varphi(X) = \varphi(\Delta^+X) - \varphi(\Delta^-X)$, where Δ^+X and Δ^-X are the sets of arcs leaving X and entering X, respectively. In words, $\partial\varphi(X)$ is the net flow leaving X. Let f be a submodular function on V such that $f(\emptyset) = f(V) = 0$. A flow φ is a *submodular flow* if it satisfies:

$$\text{(SF)} \quad \begin{array}{l} \partial\varphi \in \mathrm{B}(f), \\ l \leq \varphi \leq u. \end{array}$$

Theorem 4.1 (Frank [6]): *System* (SF) *is feasible if and only if*

$$l(\Delta^+X) - u(\Delta^-X) \leq f(X) \tag{4.1}$$

holds for every $X \subseteq V$. If in addition l, u, f are integer valued, then there exists an integral solution. ■

Our algorithm maintains a base $x \in \mathrm{B}(f)$ as a convex combination of extreme bases $y_i \in \mathrm{B}(f)$, $x = \sum \lambda_i y_i$; and a flow φ satisfying $l \leq \varphi \leq u$. Initially, x is an extreme base obtained using the greedy algorithm and φ is any flow obeying upper and lower bounds, for instance we can start with $\varphi \equiv l$. When $x = \partial\varphi$, then we have found a feasible solution.

The algorithm also maintains a directed graph $(V, \widehat{A}_x \cup E_\varphi)$, where $E_\varphi = F_\varphi \cup B_\varphi$ is defined by $F_\varphi := \{(v,w) \mid \varphi(v,w) < u(v,w)\}$, $B_\varphi := \{(v,w) \mid \varphi(w,v) > l(w,v)\}$. For an arc $(v,w) \in E_\varphi$, we denote its residual capacity by $r(s,t)$. That is, $r(v,w) := u(v,w) - \varphi(v,w)$ if $(v,w) \in F_\varphi$ and $r(v,w) := \varphi(w,v) - l(w,v)$ if $(v,w) \in B_\varphi$. Let $S := \{v \in V \mid x(v) > \partial\varphi(v)\}$ and let $T := \{v \in V \mid x(v) < \partial\varphi(v)\}$.

As with the previous algorithm, our feasibility algorithm maintains a valid distance labeling d, which satisfies $d(s) \leq d(t) + 1$ for every arc $(s,t) \in \widehat{A}_x \cup E_\varphi$ and $d(t) = 0$ for every $t \in T$. Initially, the algorithm starts with $d(s) = 0$ for $s \in V$. Again, we set $Q := \{s \in S \mid d(s) < n\}$.

As before, our algorithm consists of two types of basic operations, pushes and relabels. However, now pushes fall into two categories: pushes on arcs in E_φ, and the previously defined $\mathsf{Push}(s,t)$. We will differentiate the former type of push by denoting it $\mathsf{FPush}(s,t)$. An $\mathsf{FPush}(s,t)$ applies if $s \in Q$, $(s,t) \in E_\varphi$, and $d(s) = d(t)+1$. It augments $\varphi(s,t)$ by $\varepsilon := \min\{x(s) - \partial\varphi(s), r(s,t)\}$. If $\varepsilon = r(s,t)$, then the push is called *saturating*. Otherwise it is called *nonsaturating*.

In this algorithm, we also modify the applicability of relabel, calling the operation $\mathsf{FRelabel}(s)$. It is now applicable if $s \in Q$ and $d(s) \leq d(t)$ for every $(s,t) \in \widehat{A}_x \cup E_\varphi$. It updates $d(s) := d(s) + 1$.

The algorithm works as before, but we replace $\mathsf{Scan}(s)$ with $\mathsf{FScan}(s)$, where $\mathsf{FScan}(s)$ applies both $\mathsf{Push}(s,t)$ and $\mathsf{FPush}(s,t)$; and $\mathsf{Relabel}(s)$ is replaced with $\mathsf{FRelabel}(s)$.

The algorithm terminates when Q is empty. If S is also empty, then so is T and $x(v) = \partial\varphi(v)$ for all $v \in V$, so that φ is a feasible flow. Otherwise, the set W of vertices reachable from S, does not intersect T. In this case, we have $l(\Delta^+W) - u(\Delta^-W) = \partial\varphi(W) < x(W) = \sum_{i \in I} \lambda_i y_i(W) = f(W)$. Thus W is a certificate of infeasibility via Theorem 4.1.

The correctness and complexity of the algorithm are shown by extending Lemmas 3.2, 3.3, and 3.4 to this algorithm by including operation $\mathsf{FPush}(s,t)$. The arguments for this extra operation are similar and simpler, so are omitted here.

This algorithm may be extended to find a feasible submodular flow maximizing flow on a particular arc (s^*, t^*). Instead of starting with $d(v) = 0$ for all v, we set $d(s^*) = n$ and allow labels to increase to $2n$. We also start with the flow φ with $\varphi(s,v) = u(s,v)$ $\forall\, v \in V$, and with the extreme base obtained using the greedy algorithm with an ordering that puts s^* first. This implies that there will be no initial arcs leaving s^* in $E_\varphi \cup \widehat{A}_x$, and thus the modified initial labeling is valid.

Theorem 4.2: *There is a combinatorial algorithm for computing a maximum submodular flow using $O(n^7)$ oracle calls and $O(n^8)$ arithmetic computations.*

5. Minimum Cost Submodular Flow

In this section we describe the first combinatorial, polynomial time algorithm for minimum cost submodular flow that does not call an oracle for SFM. Our algorithm computes optimal dual node prices in the same time as the fastest combinatorial polynomial time algorithm for SFM [12]. We can then modify our problem to obtain the flow with m additional iterations so that the resulting algorithm runs in time $O(mn^5 \log(nU))$. We also obtain a strongly polynomial time algorithm.

We obtain these results by exploiting the similarity between the two recent papers: a scaling algorithm for submodular flow by Fleischer, Iwata, and McCormick [5] and the combinatorial, polynomial time algorithm for submodular function minimization by Iwata, Fleischer, Fujishige [12], which was inspired by [5].

On Notation: The choice of direction for an exchange arc made in Section 2 is arbitrary, but once fixed has implications for other choices of orientation in the paper, such as which vertices are sources and sinks, how $\partial\varphi$ is defined, and the relation of x-tight sets to A_x. In [16] and [5] this choice was made one way, and in [12] the opposite choice was made. Since our work in this paper builds on all of these algorithms, we could not be consistent with both choices. We chose to be consistent with [16] and [5]. The current section builds on work in [5, 12], however. Thus it may seem that what we are describing below is backwards from what is contained in [12], but this is simply a matter of definitions.

5.1. Optimality Conditions

The *minimum cost submodular flow problem*, often called the *submodular flow problem*, asks for a solution to (SF) that minimizes $c^{\mathrm{T}}\varphi$ for a cost vector $c \in \mathbf{R}^E$. In this section, we review optimality conditions for the submodular flow problem.

As with standard network flows, we can consider a dual problem that defines node prices p for $v \in V$. 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 $H_i = \{v \mid p(v) \geq p_i\}$, the ith *level set* of p. Define $H_0 := \emptyset$ and let $f_p : 2^V \to \mathbf{R}$ be defined by

$$f_p(X) = \sum_{i=1}^{k} \{f((X \cap H_i) \cup H_{i-1}) - f(H_{i-1})\}.$$

The following lemma follows easily from submodularity of f and implies that $\mathrm{B}(f_p) \subseteq \mathrm{B}(f)$.

Lemma 5.1: *The function f_p is submodular and satisfies $f_p \leq f$. In addition, if there exists i such that for set X $H_i \subseteq X \subseteq H_{i+1}$, then $f_p(X) = f(X)$.*

Theorem 5.2: *For a base $x \in \mathrm{B}(f)$, the following are equivalent:*
⟨i⟩ *x is p-maximum.*
⟨ii⟩ *$x \in B(f_p)$.*
⟨iii⟩ *$x(H_i) = f(H_i)$ for every i.*
⟨iv⟩ *$p(w) \geq p(v)$ for every $(w, v) \in A_x$.* ■

For arc $(w, v) = a \in E$, define $\partial^+ a = w$ and $\partial^- a = v$. 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 $a \in E \cup A_x$.

Theorem 5.3: *A submodular flow φ is optimal if and only if there exists $p \in \mathbf{R}^V$ such that:*
⟨a⟩ *For any $a \in E$, $c_p(a) > 0$ implies $\varphi(a) = l(a)$, and $c_p(a) < 0$ implies $\varphi(a) = u(a)$, and*
⟨b⟩ *$\partial\varphi$ is a p-maximum base in $\mathrm{B}(f)$.*
Moreover, if c is integral, then we may restrict the above p to be integral. ■

5.2. A Scaling Algorithm

We begin by discussing the algorithm that obtains optimal node prices p for a submodular function f that takes on integral values. In Section 5.2.8, we show how to obtain the optimal flow with $\leq m$ applications of this algorithm. Instead, we could use the optimal node prices to fix flows on all nonzero reduced cost arcs and then call a maximum submodular flow algorithm, such as the one described in Section 4 to find the rest of the flow. Due to the differences in complexities of the algorithms, it may sometimes be more efficient to do the former.

We keep the general framework of the weakly polynomial submodular flow algorithm described in [5]. This algorithm uses a shortest augmenting path subroutine within a scaling framework. We review this algorithm below, and highlight the changes that are necessary to obtain an algorithm that does not require an oracle for SFM. Our main contribution is a subroutine to find a *least cost* δ-augmenting path that does not require such an oracle. This subroutine is described in Section 5.2.4.

5.2.1. The Scaling Framework

In the δ scaling phase, capacities and submodular constraints are relaxed by δ by adding the arc set of a complete directed graph on V with capacity δ to the initial graph. This arc set is denoted $D = \{(w, v) \mid w \neq v \in V\}$. For all $a \in D$, we set $c(a) = 0$, $l(a) = 0$, and $u(a) = \delta$. Define the submodular function $b : 2^V \to \mathbf{R}$ by $b(X) = |X| \cdot |V - X|$. Equivalently, $\delta b(X)$ is the capacity of the cut X in (V, D). This relaxation can be thought of as either relaxing the condition that $\partial\varphi \in \mathrm{B}(f)$ to $\partial\varphi \in \mathrm{B}(f + \delta b)$, or as relaxing the capacities l and u by δ. However, the arcs in D have no cost, so this is not a pure relaxation of capacities. We treat the arcs of D as having their own separate flow, denoted ψ. 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 either (v, w) or (w, v) has residual capacity δ.

At any given point in the algorithm we will have a flow φ on E, and flow ψ on D, a price vector p, and a base $z \in \mathrm{B}(f_p + \delta b)$. We maintain $z = x - \partial\psi$ as the sum of $x \in \mathrm{B}(f_p)$ and $-\partial\psi \in \mathrm{B}(\delta b)$. Since we are not allowed to compute exchange capacities in general, as with the algorithm in Section 4, we maintain $x = \sum_{i \in I} \lambda_i y_i$ as a convex combination of extreme bases $y_i \in B(f_p)$. Following [12], for each y_i, we maintain a linear ordering L_i that generates y_i. This will allow us to apply Lemma 2.1 when appropriate. The algorithm also maintains ⟨a⟩ of Theorem 5.3 for φ and p, and ⟨b⟩ of Theorem 5.3 for x and p.

We measure progress in the algorithm via the *discrepancy* between z and $\partial\varphi$, which is defined by the *discrepancy function* $\Phi = \sum_v |z(v) - \partial\varphi(v)|$. In a δ-scaling phase, the algorithm repeatedly looks for a path from $S^+(\delta) := \{v | z(v) \geq \partial\varphi(v) + \delta\}$ to $S^-(\delta) := \{v | z(v) \leq \partial\varphi(v) - \delta\}$ of residual capacity $\geq \delta$, and then augments flow on this path, decreasing the discrepancy by 2δ. This is a path consisting of arcs in $E_\varphi(\delta) \cup D_\psi(\delta)$, where $E_\varphi(\delta)$ is the set of arcs in E_φ with capacity at least δ, and $D_\psi(\delta)$ is the set of arcs $a \in D$ with $\psi(a) = 0$. This is a *δ-augmenting path.*

Our algorithm starts with large enough $\delta = U$ as specified in Section 5.2.2. We show in Section 5.2.3 that a scaling phase starts with the discrepancy is at most $4n^2\delta$ and ends with the discrepancy at most $n^2\delta$. In Section 5.2.7 we show that if f, l, u, and c are integer, then p is optimal at the end of the scaling phase with $\delta < 1/n^2$.

5.2.2. Initialization

We start with a flow φ and a price vector p that satisfy condition $\langle a \rangle$ of Theorem 5.3 obtained as follows. First, we check whether there exists a dual feasible solution p and obtain such a solution, using the Bellman–Ford–Moore algorithm in a modified graph, as described in [5]. With dual feasible p, we can construct a flow φ that satisfies condition $\langle a \rangle$ of Theorem 5.3 in $O(m)$ time. We also start with linear ordering L that is ordered according to nonincreasing p-values (see Lemma 5.5 for motivation), and a base $x \in B(f_p)$ obtained by applying the greedy algorithm [3] to L. We do not check for primal infeasibility, since our algorithm will detect this.

We set $U := \max\{\max\{|u(a)| \mid u(a) < +\infty\}, \max\{|l(a)| \mid l(a) > -\infty\}, \max\{f(\{v\}) \mid v \in V\}\}$. Since $\varphi(a) \leq U$ for all $a \in E$, we have $|\partial\varphi(v)| \leq nU$ for all $v \in V$. Since $x(V) = f(V) = 0$, we have that $|x(v)| \leq (n-1)U$. Thus the initial discrepancy between x and φ is at most $2n^2U$.

5.2.3. A Scaling Phase

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 (5.1) for arcs a with residual capacity $\delta \leq r(a) < 2\delta$.

The object in a scaling phase is to decrease the discrepancy by augmenting along δ-augmenting paths restricted to $E_\varphi(\delta) \cup D_\psi(\delta)$, while maintaining reduced cost optimality conditions

$$c_p(a) \geq 0, \ \forall\, a \in E_\varphi(\delta) \cup D_\psi(\delta) \cup A_x \qquad (5.1)$$

implied by Theorem 5.3. To do this, it is necessary to find a least-cost (with respect to reduced costs c_p) δ-augmenting path. The algorithm repeatedly calls the subroutine SubmodDijkstra to find this. This subroutine is an extension of Dijkstra's algorithm and is discussed in the next section. It returns a least-cost path on $E_\varphi(\delta) \cup D_\psi(\delta)$. It may seem that it should consider also arcs in A_x when searching for a least cost δ-augmenting path. Instead, SubmodDijkstra performs a *double-exchange* on selected arcs in its search for a least cost path to avoid the appearance of exchange arcs on this path. A double-exchange on arc a is an exchange operation on a followed by a modification of flow on a, so that z is unchanged at the endpoints of a.

Since $z = x - \partial\psi$ and we augment by exactly δ along a least-cost δ-augmenting path, each augmentation decreases the discrepancy by δ at both endpoints of the augmenting path, maintains the discrepancy of all other nodes. A phase ends when one of $S^+(\delta)$ or $S^-(\delta)$ is empty, or the set of nodes R reachable from $S^+(\delta)$ in $E_\varphi(\delta) \cup D_\psi(\delta) \cup A_x$ is disjoint from $S^-(\delta)$. In the first case, since the net excess is 0, at the end of a phase the total discrepancy is $\Phi \leq 2n\delta$. In the second case, either the total discrepancy is bounded by the residual capacity in $(E_\varphi \cup D_\psi) \cap \Delta^+ R$, which is bounded by $(n^2/2)\delta$, or we have a proof of primal infeasibility.

5.2.4. Finding a Shortest δ Augmenting Path

We describe how to find a least-cost δ-augmenting path in $E_\varphi(\delta) \cup D_\psi(\delta) \cup A_x$ without using an exchange capacity oracle. Our algorithm is an extension of Dijkstra's shortest path algorithm to handle exchange capacities. Dijkstra's algorithm has been used in the Edmonds-Karp capacity scaling algorithm for minimum cost flow to find a least-cost path of capacity at least δ. This can be done by ignoring edges with residual capacity less than δ. This becomes more complicated in submodular flow settings, since there may also be exchange arcs. We avoid using exchange arcs on the least-cost path by a double-exchange operation that trades exchange capacity on an arc in A_x for residual flow capacity on the parallel arc in D. Since both these arcs have zero cost, they also have the same reduced cost, and thus serve equally well on a least cost path.

Performing an exchange operation on (s,t) can increase exchange capacity on other arcs, thus changing the residual exchange capacity graph. This makes it tricky to maintain valid distance labels as required for the correctness of Dijkstra's algorithm. In particular, unlike the case for residual flow arcs, we cannot ignore exchange arcs that have positive residual capacity less than δ, since the capacity of these arcs may change even when exchange operations are performed on completely different arcs. Fortunately, we can characterize when the capacity of exchange arcs can become strictly positive. Lemma 5.4 is a simpler version of Lemma 3.1.

Lemma 5.4: *Let y be a base, and $y' = y + \mu(\chi_t - \chi_s)$ for $\mu \leq \sigma(y,t,s)$. If $(w,v) \notin A_y$ and $(w,v) \in A_{y'}$, then $\{(w,t),(s,v)\} \subseteq A_y$.* ■

In [5], the authors use Lemma 5.4 to develop a version of Dijkstra's that works in the presence of exchange arcs. To solve the SFM problem, this idea was modified to find δ-augmenting paths without an oracle for computing exchange capacities in [12]. Instead, the algorithm in [12] ignores most exchange capacity arcs and only considers pairs satisfying the conditions of Lemma 2.1. However, it does not find a *least-cost* augmenting path. In this section, we extend these ideas to find a least-cost δ-augmenting path while restricted to computing exchange capacities using Lemma 2.1. It is not immediately evident that it is possible to find least-cost paths if some exchange arcs are ignored. We show that this is possible by carefully choosing the linear orderings L_i generating y_i for $i \in I$. The result yields an efficient, combinatorial algorithm for submodular flow.

The subroutine SubmodDijkstra starts with a parameter δ, node prices p, a base $x = \sum_{i \in I} \lambda_i y_i \in B(f_p)$, a flow φ on E, and a flow ψ on $D(\delta)$. The algorithm also maintains the linear orderings L_i that generate y_i, $\forall i \in I$. The subroutine SubmodDijkstra is described in Figure 2. It maintains distance labels d and a set R of permanently labeled vertices that are reachable from $S^+(\delta)$ by δ-augmenting paths in $E_\varphi(\delta) \cup D_\psi(\delta)$. The algorithm returns either a least cost

(w.r.t. c_p) δ-augmenting path in $E_\varphi \cup D_\psi$ or a set R of all nodes reachable from $S^+(\delta)$ in $E_\varphi(\delta) \cup D_\psi(\delta) \cup A_x$ such that $R \cap S^-(\delta) = \emptyset$.

The distance labels $d : V \to \mathbf{Z}$ satisfy $d(w) \leq d(v) + c_p(v,w)$ for all $(v,w) \in E_\varphi(\delta) \cup D_\psi(\delta)$ with $v \in R$, and $d(w) \leq d(v) + c_p(v,w)$ for all $(v,w) \in E_\varphi(\delta) \cup D_\psi(\delta) \cup A_x$ with $v, w \in R$. At start, $R = S^+(\delta)$ and all nodes in R have distance label 0. Immediately, all nodes reachable from R on 0-reduced cost arcs in $E_\varphi(\delta) \cup D_\psi(\delta)$ are added to R with distance label 0.

SubmodDijkstra proceeds by looking for arcs $(w,v) \in A_x$ with $w \in R$, $v \in V \backslash R$ and $c_p(a) = 0$. If such an arc is found, a double-exchange is applied by performing an exchange operation, and then sending flow backwards on the parallel ψ arc to maintain $z = x - \partial\psi$. This works as follows. Double-Exchange(i,w,v) swaps exchange capacity on (w,v) for residual flow capacity on (w,v) by setting $y_k = y_i + \alpha(\chi_v - \chi_w)$ for $\alpha := \min\{\sigma(y_i,v,w), \delta\}$ and reducing $\psi(w,v)$ by α. In words, α is the minimum of δ, which is lower bound on the amount of ψ flow that can be sent backwards on (w,v) (forwards on (v,w)), and the maximum amount of change possible to affect in x by performing an exchange operation on y_i for (w,v). Double-Exchange is *saturating* if $\alpha = \sigma(y_i,v,w)$, and *nonsaturating* otherwise. A saturating Double-Exchange(i,w,v) updates y_i as $y_i := y_i + \sigma(y_i,v,w)(\chi_v - \chi_w)$ and modifies L_i by interchanging v and w. A nonsaturating double-exchange in addition adds a new index k to I with y_k equal to the old y_i, sets $\lambda_i = \alpha/\sigma(y_i,v,w)$ and sets λ_k as the difference between old and new λ_i. In both cases, x moves to $x + \alpha(\chi_v - \chi_w)$. Thus $z = x - \partial\varphi$ is invariant. The operation Double-Exchange is depicted in Figure 1. It is based on the operation Swap introduced in [5]. Swap was modified in [12] to the subroutine given here.

If Double-Exchange creates a residual ψ-arc (w,v), then this arc has zero reduced cost. (All ψ arcs and all exchange capacity arcs have initial cost 0, and hence the corresponding reduced costs are the same. By the applicability of double exchange, this is 0.) In this case, the set $R(v)$ of all vertices in $V \backslash R$ reachable from v on 0-reduced cost paths in $E_\varphi(\delta) \cup D_\psi(\delta)$ may be added to R after updating the labels of these vertices to be equal to $d(w)$.

If no arc $(w,v) \in A_x$ with $w \in R$, $v \in V \backslash R$ and $c_p(a) = 0$ is found, then SubmodDijkstra selects the lowest, finitely-labeled vertex t in $V \backslash R$, adds $R(t)$ to R after updated all the labels in this set to be $d(t)$, and updates the labels of vertices in $V \backslash R$ adjacent to $R(t)$ in $E_\varphi(\delta) \cup D_\psi(\delta)$.

At the end of the subroutine SubmodDijkstra, the set of extreme bases I is reduced to an affinely independent set using a standard linear programming technique.

5.2.5. Implementation

To implement SubmodDijkstra efficiently, we start the subroutine by reordering each linear ordering L_i so that the p-values of vertices are monotone nonincreasing.

```
Double-Exchange(i, w, v):

α ← min{δ, λ_i σ(y_i, v, w)}
If α < λ_i σ(y_i, v, w) then
        k ← a new index
        I ← I ∪ {k}
        λ_k ← λ_i − α/σ(y_i, v, w)
        λ_i ← α/σ(y_i, v, w)
        y_k ← y_i
        L_k ← L_i
y_i ← y_i + σ(y_i, v, w)(χ_v − χ_w)
Update L_i by interchanging v and w.
x ← Σ_{i∈I} λ_i y_i    [ x ← x + α(χ_v − χ_w)]
ψ(w, v) ← ψ(w, v) − α
ψ(v, w) ← ψ(v, w) + α
```

Figure 1: The operation Double-Exchange(i,w,v).

Lemma 5.5: *For an extreme base y_i encountered during the algorithm, there exists a linear ordering generating y_i that is ordered according to nonincreasing p-values.*

Proof. All exchange arcs (v,w) have $c(v,w) = 0$. Thus if $(v,w) \in A_x$ then since the algorithm maintains $c_p(v,w) \geq 0$, we have that $p(v) \geq p(w)$. By construction, $(v,w) \in A_{y_i}$ implies $(v,w) \in A_x$. Suppose L_i has $p(v_k) < p(v_{k+1})$. Then $(v_k, v_{k+1}) \notin A_{y_i}$ and thus $\sigma(y_i, v_k, v_{k+1}) = 0$. Then by Lemma 2.1, we can interchange v_k and v_{k+1} in L_i, and the resulting order still generates y_i. ■

Since p-values don't change between augmentations, this grouping of vertices by level sets in each L_i remains unchanged between augmentations.

A pair of vertices (v,w) is called *level-active* if $v \in R$, $w \in V \backslash R$, both v and w are in $H_l \backslash H_{l-1}$ for some l, and v immediately precedes w in L_i for some $i \in I$. (This is a refinement of the concept of active pair introduced in [12].) Since all exchange arcs have initial cost zero, if a level-active pair has positive exchange capacity, then its reduced cost is 0. The following lemma says that it is sufficient to consider level-active pairs when looking for 0-reduced cost exchange arcs.

Lemma 5.6: *If there are no level-active pairs, then there are no 0-reduced cost arcs in A_x that leave R.*

Proof. Suppose that the level sets of p are $\emptyset = H_0 \subset H_1 \subset \cdots \subset H_r = V$. We prove the lemma by proving the following statement: If there are no level-active pairs, then $(H_l \backslash R) \cup H_{l-1}$ is tight for all $l \in [1,r]$. This statement implies the lemma, since if $(w,v) \in A_x$ has 0-reduced cost, then $\{w,v\} \subseteq (H_l \backslash H_{l-1})$ for some l.

To prove the statement, it suffices to consider a fixed set $J_l = H_l \backslash H_{l-1}$. If there are no level-active pairs, then all elements in $J_l \backslash R$ precede all elements in $J_l \cap R$ in every linear ordering L_i, $i \in I$. This implies that $H_{l-1} \cup J_l \backslash R =$

SubmodDijkstra($\varphi, \psi, \delta, x = \sum_i \lambda_i y_i, p$)

Initialization
  $d(v) \leftarrow +\infty \quad \forall v \in V \backslash S^+(\delta)$
  $d(w) \leftarrow 0 \quad \forall w \in S^+(\delta)$
  $R \leftarrow S^+(\delta)$ [*permanently labeled vertices*]
  for all $(w,v) \in E_\varphi(\delta) \cup D_\psi(\delta)$ with $v \in V \backslash R$
    $d(v) \leftarrow \min\{d(v), d(w) + c_p(w,v)\}$
    for all $i \in I$,
      reorder L_i by nonincreasing p-values.

while $R \cap S^- = \emptyset$ and $\exists$ level-active pairs,
  while $\exists$ level-active pair (w,v) for some $i \in I$,
    Double-Exchange(i,w,v).
    if $\psi(w,v) = 0$,
      For all $z \in R(v)$,
        $d(z) \leftarrow d(w)$,
      $R \leftarrow R \cup R(v)$
      for all $a = (z,t) \in E_\varphi(\delta) \cup D_\psi(\delta)$
        with $z \in R(v)$ and $t \in V \backslash R$,
          $d(t) \leftarrow \min\{d(t), d(z) + c_p(a)\}$
  if $\exists\, v \in V \backslash R$ with $d(v) < +\infty$,
    $v \leftarrow$ node in $V \backslash R$ with smallest label.
    For all $z \in R(v)$,
      $d(z) \leftarrow d(w)$,
    $R \leftarrow R \cup R(v)$
    for all $a = (z,t) \in E_\varphi(\delta) \cup D_\psi(\delta)$
      with $z \in R(v)$ and $t \in V \backslash R$,
        $d(t) \leftarrow \min\{d(t), d(z) + c_p(a)\}$

Reduce I to an affinely independent set.
if $\exists\, v \in R \cap S^-(\delta)$,
  return path P from $S^+(\delta)$ to v on nodes in R.
  Else, **return** R.

Figure 2: Finding a least-cost δ-augmenting path.

$H_l \backslash R \cup H_{l-1}$ is y_i-tight for all $i \in I$. But, if a set X is y_i-tight for all $i \in I$, then it is also x-tight, since in this case $x(X) = \sum_{i \in I} \lambda_i y_i(X) = \sum_{i \in I} \lambda_i f(X) = f(X)$. ■

5.2.6. Correctness and Complexity

Lemma 5.7: Double-Exchange(i,w,v) *maintains valid distance labels, reduced cost optimality conditions, and* $z = x - \partial\psi$.

Proof. Double-Exchange is only applied to arcs with zero reduced cost. Thus any new arc in $D_\psi(\delta)$ has zero reduced cost.

Suppose Double-Exchange(i,w,v) is applied and moves y_i to y_i', and ψ to ψ'. We show that if it creates a new exchange arc (s,t) in $A_{y_i'}$, then there exists a path in $A_{y_i} \cup D_\psi(\delta)$ of the same reduced cost: If a new arc (s,t) appears in $A_{y_i'}$, then Lemma 5.4 implies that arc (s,v) and arc (w,t) existed in A_{y_i}. Since Double-Exchange(i,w,v) applies, $(v,w) \in D_\psi(\delta)$ before the double-exchange. Thus there is a path from z to t of zero initial cost arcs before the double-exchange. Since $(s,t) \in A_{y_i'}$ also has zero initial cost, the reduced cost of these two paths are the same.

Thus, since reduced cost optimality conditions held before the double-exchange, they hold afterward. Similarly, since the reduced cost of any new arc equals the reduced cost of any path between the same endpoints, distance labels remain valid.

Finally, for all vertices v, Double-Exchange alters $x(v)$ and $\partial\varphi(v)$ by the same amount, so z is unchanged. ■

Theorem 5.8: SubmodDijkstra *returns a least-cost δ-augmenting path, or a proof no such path exists, in $O(n^3)$ time, and using at most $O(n^3)$ function evaluations.*

Proof. (Sketch) Correctness: The key is to show that when a vertex is added to R, its label is the shortest path distance using distances c_p from $S^-(\delta)$ on arcs in $E_\varphi(\delta) \cup D_\psi(\delta) \cup A_x$ by induction. This is uses Lemmas 5.6 and 5.7 combined with standard arguments for correctness of Dijkstra's algorithm. Details omitted for lack of space.

Complexity: The initial reordering of L_i takes at most n^2 steps per $i \in I$ for a total of n^3 steps. Each nonsaturating Double-Exchange augments R, so there are at most n nonsaturating double-exchanges. Each saturating Double-Exchange moves an element of $V \backslash R$ closer to the start of L_i. This can happen at most $n-1$ times per element of $V \backslash R$ per $i \in I$, for an upper bound of n^3 saturating Double-Exchanges. Each Double-Exchange uses a constant number of evaluations of f_p plus a constant number of arithmetic operations.

While a single evaluation of f_p could require a linear number of evaluations of f, f_p is only evaluated on sets nested in level sets of p. Thus, by Lemma 5.1, it suffices to evaluate f. Thus the total effort spent finding a δ-augmenting path is $O(n^3)$ arithmetic steps and function evaluations.

Finally, SubmodDijkstra updates I to be affinely independent. SubmodDijkstra starts with an affinely independent set I, and only increases the size of I in nonsaturating double-exchanges. Thus at end $|I| < 2n$. Hence with $O(n^3)$ arithmetic operations, we can reduce I so that $|I| \leq n$. ■

5.2.7. Termination

We now discuss how to terminate the scaling algorithm, provided that l, u, and f are all integer-valued. For a price function p, we define l_p and u_p by setting $l_p(a) := u_p(a) := l(a)$ if $c_p(a) > 0$; $l_p(a) := l(a)$, $u_p(a) := u(a)$ if $c_p(a) = 0$; $l_p(a) := u_p(a) := u(a)$ if $c_p(a) < 0$. At the end of the δ scaling phase, for any $X \subset V$, we have

$$\begin{aligned} \kappa_p(X) &:= l_p(\Delta^+X) - u_p(\Delta^-X) - f_p(X) \\ &\leq \partial\varphi(X) - x(X) \\ &\leq \partial\varphi(X) - z(X) + n^2\delta/2 \end{aligned}$$

Since the discrepancy Φ is at most $n^2\delta/2$, we obtain $\kappa_p(X) \leq n^2\delta$. If $\delta < 1/n^2$, the integrality assumption implies that $\kappa_p(X) \leq 0$. Then it follows from Theorem 4.1 that there exists a submodular flow ξ that satisfies $l_p \leq \xi \leq u_p$ and $\partial\xi \in \mathrm{B}(f_p)$. Hence, Theorem 5.3 implies that ξ is an optimal flow and p is an optimal price function.

Theorem 5.9: *Optimal dual node prices for the minimum cost submodular flow problem for integer f can be found with $O(n^5 \log(nU))$ arithmetic steps and function evaluations.*

Proof. There are $\log(nU)$ scaling phases. After the initial flow adjustments, the initial discrepancy in a δ-phase is at most $4n\delta + 3n^2\delta + 4m\delta$. Thus the total number of augmentations in any δ phase is at most $4n^2 + 2n = \mathrm{O}(n^2)$, since each augmentation decreases the discrepancy by 2δ. Theorem 5.8 then implies the complexity bound. ■

5.2.8. Obtaining the Flow

In order to compute an optimal submodular flow, it suffices to find a feasible submodular flow with l_p, u_p and f_p. This could be done by using the algorithm in Section 4. However, a more efficient algorithm is to apply the price finding algorithm described above to m slightly modified problems: Again, start with l_p, u_p but now modify the reduced cost of one of the zero reduced cost arcs to be (reduced) cost -1. The price finding algorithm finds optimal prices p_1 which reveal the objective function value, and thus the maximum amount of flow on this arc in any feasible flow. Fix this flow, and repeat.

Corollary 5.10: *A minimum cost submodular flow can be found via a combinatorial algorithm in $\mathrm{O}(mn^5 \log(nU))$ time.*

We have just explained how to find a feasible submodular flow, or a flow maximizing the flow on a specified arc, with m calls to a modified submodular function minimization algorithm. András Frank points out that this can be done with m calls to any SFM algorithm: Let $h(X) := u(\Delta^- X) - l(\Delta^+ X)$. Since $u \geq l$, h is submodular. Thus Theorem 4.1 implies that feasibility of a submodular flow problem can be checked by applying an SFM algorithm to $f + h$. Applying this test to the modified problem with $l'(a) = u(a)$ for a fixed arc a reveals the maximum flow possible on a in any feasible flow.

5.3. A Strongly Polynomial Algorithm

To obtain a strongly polynomial algorithm, we embed the subroutine SubmodDijkstra in a variant of the strongly polynomial algorithm in [5]. This uses $\log n$ scaling phases to fix the sign of the reduced cost of one arc. Thus after $n^2 \log n$ scaling phases, all reduced costs are fixed, and an optimal price vector p is deduced. The optimal flow can then be found as described in Section 5.2.8.

Acknowledgements

We are grateful to Lex Schrijver for sharing with us drafts of his paper on a combinatorial algorithm for minimizing submodular functions in strongly polynomial time. We also thank András Frank and Tom McCormick for useful comments.

References

[1] W. H. Cunningham. Testing membership in matroid polyhedra. *J. Combinatorial Theory B*, 36:161–188, 1984.

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

[3] J. Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and their Applications*, pages 69–87. Gordon and Breach, 1970.

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

[5] L. Fleischer, S. Iwata, and S. T. McCormick. A faster capacity scaling algorithm for submodular flow. Technical Report 9947, C.O.R.E. Discussion Paper, Louvain-la-Neuve, Belgium, 1999.

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

[7] A. Frank and É. Tardos. An application of submodular flows. *Linear algebra and its applications*, 114/115:329–348, 1989.

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

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

[10] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.

[11] M. Grotschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[12] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial, strongly polynomial-time algorithm for minimizing submodular functions. In *Proceedings of the 32th Annual ACM Symposium on Theory of Computing*, 2000. This proceedings.

[13] S. Iwata, S. T. McCormick, and M. Shigeno. A fast cost scaling algorithm for submodular flow. To appear.

[14] T. Jordán. Edge-splitting problems with demands. In G. Cornuéjols, R. E. Burkard, and G. J. Woeginger, editors, *Integer Programming and Combinatorial Optimization*, volume 1610 of *LNCS*, pages 273–288, Graz, Austria, June 1999. Springer.

[15] M. Queyranne. Minimizing symmetric submodular functions. *Math. Programming*, 82:3–12, 1998.

[16] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. Preprint. Submitted to JCTB., 1999.

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