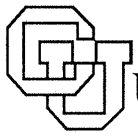
A Framework for Cost-scaling Algorithms for Submodular Flow Problems

Harold N. Gabow

CU-CS-661-93 August 1993



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

A Framework for Cost-scaling Algorithms for Submodular Flow Problems

Harold N. Gabow*
Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309
hal@cs.colorado.edu

August 3, 1993

Abstract. The submodular flow problem includes such problems as ordinary network flow, dijoin, edge-connectivity orientation and others. We present a cost-scaling algorithm for submodular flow problems. The algorithm applies to these problems in general; we also examine its efficiency for the dijoin and edge-connectivity orientation problems. A minimum-cost dijoin is found in time $O(\min\{m^{1/2}, n^{2/3}\}nm\log(nN))$, where n, m and N denote the number of vertices, number of edges and largest magnitude of an integral edge cost. The previous best-known bound is $O(n^2m)$ if fast matrix multiplication is not used. A k-edge-connected orientation is found in time $O(kn^2(\sqrt{kn} + k^2\log(n/k)))$. A minimum-cost k-edge-connected orientation is found in the above time bound for dijoins when $k = O(1)$ (and a more complicated bound for general k). The scaling algorithm uses a transformation that eliminates vertex weights in edge-capacitated graphs. It also incorporates a scheme to limit the growth in the size of intermediate solutions, using a dual minimum-cost network flow problem.

---

* Research supported in part by NSF Grant No. CCR-9215199.

1. Introduction

Submodular flow is among the most general integer linear programs having a polynomial-time algorithm [e.g., S]. A number of polynomial-time submodular flow algorithms have been presented [see Fu, FT, and their references]. This paper gives an efficient cost-scaling algorithm for 0-1 submodular flow. The algorithm is presented for arbitrary 0-1 flow problems. Its efficiency is illustrated for two of the most well-known such problems.

Our algorithm generalizes a large number of cost-scaling algorithms [e.g., GoT, GaT89, GX]. A difficulty is that the size of the intermediate solutions can grow, thereby degrading performance, even if the final solution is small. This does not occur in previous algorithms, e.g., in minimum-cost matching [GaT89] a matching always has $O(n)$ edges. We introduce a "potential compression" step to cut back growth in the size of the solution. Nonetheless the phenomenon of growing solutions prevents us from achieving the best efficiency a naive observer might hope for.

We state our results for specific flow problems. In this paper n, m and N denote the number of vertices, number of edges and largest magnitude of an edge cost for a given graph. When N is used the edge costs are assumed to be integral.

In a digraph, a *dijoin* is a set of edges whose contraction gives a strongly-connected graph. In the *minimum-cost dijoin problem* we are given a digraph with edge costs, and we seek a dijoin of smallest possible cost. This problem contains minimum-cost bipartite matching as a special case. The Lucchesi-Younger Theorem gives a minimax characterization of the solution [GLS]. Polynomial-time algorithms are in [L, K, F81]; the most efficient algorithm is the implementation of Frank's algorithm given in [Ga93a] running in time $O(\min\{n^2m, nM(n)\})$. Here $M(n)$ is the time to multiply $n \times n$ matrices and $M(n) = O(n^{2.38})$ [CW]. Our cost-scaling algorithm achieves time $O(\min\{m^{1/2}, n^{2/3}\}nm\log(nN))$. Under the assumption of similarity $\log N = O(\log n)$ [Ga85] this improves the first bound of [Ga93a]; it is at most a factor $n^{.3}$ more than the second bound, without using fast matrix multiplication. In fact our algorithms do not use sophisticated data structures and should perform well in practice. Note the similarity assumption holds for the minimum-cardinality dijoin problem ($N = 1$) where all the results listed above are the best known.

The feedback arc set problem is NP-complete in general but for planar graphs it amounts to the dijoin problem. Our algorithm solves the planar feedback arc set problem in time $O(n^{5/2}\log(nN))$. Under similarity this improves the bound of $O(n^3)$ of [Ga93a].

An *orientation* of an undirected graph assigns a direction to each edge. The (*k-edge-connected*) *orientation problem* is to orient the edges of a given undirected graph to make it k-edge-connected, if possible. Nash-Williams gives a minimax characterization of graphs that can be so oriented [NW].

In the *minimum-cost k-edge-connected orientation problem* each orientation of an edge has a cost and we wish to minimize the total cost. These problems were solved in polynomial time by Frank [F82]. The most efficient implementation of the minimum-cost orientation algorithm is in [Ga93a], achieving time $O(\min\{kn^2m, kn(km\log(n^2/m) + M(n))\})$. We improve the bound for finding a minimum-cost orientation as follows. For $k = O(1)$ the time to find a minimum-cost orientation is the same as the above bound for dijoin. For general k (by definition $k \leq n$) the time to find a minimum-cost orientation is $O((\min\{(kn)^{2/3}nm,\ nm^{3/2} + m^2\log n\} + k^3n^2\log(n/k))\log(nN))$ (without fast matrix multiplication).

A feasible 0-1 submodular flow can be found by executing one scale of our algorithm. This improves the time for the connectivity orientation problem (i.e., find a k-edge-connected orientation) to $O(kn^2(\sqrt{kn} + k^2\log(n/k)))$. Frank [F84] gives a polynomial-time algorithm for finding a feasible submodular flow (for 0-1 and arbitrary flow problems).

The cost-scaling algorithm incorporates two ideas that may be of independent interest. First, it is necessary to analyze cuts in graphs having edge capacities and vertex weights. We give a transformation to eliminate vertex weights in such graphs. This generalizes Picard's algorithm for finding a maximum-profit closed set in a graph [P, C]. Second, we show that the growth problem mentioned above can be formulated as the dual of a minimum-cost flow problem. This leads to our efficient potential compression algorithm.

We mention some other work related to our scaling algorithm. Cunningham and Frank give a polynomial-time algorithm for arbitrary submodular flows based on cost-scaling [CF]. The algorithm is not oriented toward 0-1 flow or efficiency on the above problems. Efficient submodular flow algorithms for large-capacity problems, using capacity-scaling and other ideas related to this paper, are discussed in [Ga93b].

We present the cost-scaling algorithm in a top-down fashion, using the following organization. Section 2 reviews the submodular flow problem and Frank's algorithm, and gives our main scaling algorithm. Section 3 analyzes this main algorithm to determine its fundamental parameters. Section 4 presents the potential compression algorithm. Section 5 gives the details of a search in the scaling algorithm. It is based on the notion of a "swap," generalizing the matroid intersection algorithm of [GX]. Section 6 gives the transformation to eliminate vertex weights, and applies it to supply the final details of the algorithm. Section 7 summarizes the scaling algorithm and gives the overall timing analysis. Section 8 shows how the feasibility problem can be solved by executing one scale of the algorithm.

Throughout the paper the connectivity-orientation problem is used to illustrate the discussion.

It requires the full generality of our ideas. The dijoin algorithm admits some slight simplifications from the general algorithm (these two problems are discussed at the end of Section 2).

This section closes with notation and definitions. $\mathbf{R}$ is the set of real numbers, $\mathbf{R}_+$ the non-negative reals, $\mathbf{Z}$ the integers. For a set S, $\overline{S}$ denotes its complement when the universe is clear. For a function $f : S \to \mathbf{R}$ and any set $T \subseteq S$, $f(T)$ denotes $\sum\{f(t) \mid t \in T\}$. For example for two functions $d, x : E \to \mathbf{R}$, $dx(E)$ denotes $\sum\{d(e)x(e) \mid e \in E\}$.

In a digraph $G = (V, E)$, an edge uv *enters* a set $S \subseteq V$ if v is in S but u is not; this edge *leaves* $\overline{S}$. $\rho_G(S)$ denotes the number of edges entering S. For a "capacity" function $c : E \to \mathbf{R}_+$, $\rho_c(S)$ denotes the total capacity of edges entering S. We abbreviate ρ_G or ρ_c to ρ if the graph or capacity function is clear. $\delta(S)$ denotes the number of edges leaving S. The same conventions apply to δ as ρ.

In a digraph if A is a set of edges then A^R denotes the set of reversals of these edges. Similarly for G^R and e^R for a graph G and edge e, respectively.

Let $G = (V, E)$ have capacity function c and a function $f : E \to \mathbf{R}_+$ with $f \leq c$. The *residual graph of f in G* has edges $E \cup E^R$ with any $e \in E \cup E^R$ having capacity $c(e) - f(e) + f(e^R)$ (if $e \notin E$ then assume $f(e) = c(e) = 0$). For instance if f is a 0-1 flow then the residual graph of f in G is formed by starting with G and reversing the edges with flow. We form residual graphs from functions f that do not necessarily obey the flow conservation law; also f may be defined on a graph containing G.

Consider a finite universe S. Two subsets X, Y are *intersecting* if $X \cap Y$, $X - Y$ and $Y - X$ are all nonempty. If in addition $X \cup Y \neq S$ then X and Y are *crossing*. $\mathcal{F} \subseteq 2^S$ is an *intersecting* (*crossing*) *family* if $X \cap Y$, $X \cup Y \in \mathcal{F}$ whenever $X, Y \in \mathcal{F}$ and X and Y are intersecting (crossing) [Fu, GLS]. A real-valued function f on subsets of S is *submodular* if for two sets $X, Y \subseteq S$, $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$; f is *modular* if equality always holds. ρ and δ are submodular. f is *submodular on intersecting* (*crossing*) *pairs* if the above inequality holds when X and Y are intersecting (crossing) sets.

2. Scaling submodular flows

This section presents the Main Routine of our submodular flow algorithm, which scales the costs. It begins by reviewing the submodular flow problem. Our algorithm incorporates ideas from scaling algorithms [e.g., GaT89] together with a combination of Frank's submodular flow algorithms [F82] and [F84] (this is crucial for efficiency).

Consider a digraph $G = (V, E)$ with functions d (the *profit function*), ℓ and u (the *bound*

functions) having domain E and range $\mathbf{R}$, $\mathbf{R} \cup \{-\infty\}$ and $\mathbf{R} \cup \{+\infty\}$ respectively. In addition there is a crossing family $\mathcal{F}$ defined on V and a function $b : \mathcal{F} \to \mathbf{R}$ submodular on crossing pairs of $\mathcal{F}$. The (*general*) *submodular flow problem* is to find a flow function $x : E \to \mathbf{R}$ solving this linear program:

$$\begin{aligned} &\text{maximize } dx(E) \text{ subject to} \\ &\quad \rho_x(S) - \delta_x(S) \leq b(S) \qquad \text{for each } S \in \mathcal{F} \\ &\quad \ell(e) \leq x(e) \leq u(e) \qquad \text{for each } e \in E \end{aligned} \tag{1}$$

Recall our notational convention that implies $dx(E)$ denotes $\sum\{d(e)x(e) \mid e \in E\}$. If b, ℓ and u are integer-valued then x can be chosen to be integer-valued [EG].

In the *0-1 submodular flow problem* as defined in [F82], d is nonnegative, b is integral, ℓ is identically 0 and u is identically 1. Thus x can be chosen as 0-1 valued. Frank shows the 0-1 problem is equivalent to the general problem [F82]. An edge $e \in E$ with $x(e) = 1$ is called a *1-edge*; similarly for a *0-edge*. We restrict the discussion to 0-1 submodular flow unless we explicitly refer to general submodular flow.

As in ordinary network flow, it is convenient to work on the residual graph. Let x be a 0-1 valued function and let RG be the residual graph of x (i.e., RG contains e if $x(e) = 0$ and e^R if $x(e) = 1$). It is useful to observe that the constraints of (1) are equivalent to

$$\rho_G(S) - b(S) \leq \rho_{RG}(S) \qquad \text{for each } S \in \mathcal{F}. \tag{2}$$

This follows since $\rho_{RG}(S) = \rho_G(S) - \rho_x(S) + \delta_x(S)$. A *residual edge* is an edge of RG. Extend the profit function d to the residual edges by setting $d(e)$ to its given value if $e \in E$ (i.e., $x(e) = 0$) and to $-d(e^R)$ if $e^R \in E$ (i.e., $x(e^R) = 1$). Thus pushing a unit of flow in a residual edge e increases the profit by $d(e)$.

The connectivity-orientation problem corresponds to the following submodular flow problem. The given undirected graph is first made into a digraph G, by orienting every edge in the direction of larger cost. The profit of an edge equals the decrease in cost if it is reoriented. A flow value $x(e)$ is 1 iff e is to be reoriented. Let RG be the residual graph of x. The flow constraints express the fact that RG is k-edge connected, i.e., $\rho_{RG}(S) \geq k$ for each set $S \neq \emptyset, V$. $\mathcal{F}$ is the family of sets $S \neq \emptyset, V$ and $b = \rho_G - k$. Clearly (2) amounts to the desired constraint, $k \leq \rho_{RG}(S)$.

We make our algorithm more efficient using the notion of a *witness graph* W for a submodular flow, defined as a subgraph of G such that changing the flow on any set of edges not in W gives

a feasible flow. G itself is a witness graph but there is often a sparser witness. For example in connectivity orientation, a witness graph consists of 2 complete k-intersections for the residual graph RG [E, Ga91]. (It is not necessary for the reader to recall the notion of k-intersections. The k-intersections have $O(kn)$ edges total, and so fulfill the role of being a sparse subgraph.) In the dijoin problem a witness graph is a spanning tree of the 0-edges. For a general 0-1 flow problem a witness graph corresponds to a subgraph of RG that satisfies (2). When the witness graph W is fixed, a *witness edge* belongs to W while a *nonwitness edge* does not.

Frank [F82] introduces the following important notions. Any (arbitrary) function $p : V \to \mathbf{R}$ is called a *potential function.* A set $S \in \mathcal{F}$ is *c-tight* if its inequality in (1) (or equivalently (2)) holds with equality. For example in connectivity orientation a c-tight set S has $\rho_{RG}(S) = k$. For any $v \in V$ the set $P(v) \subseteq V$ is defined as the intersection of V and all c-tight sets containing v. A *jumping edge* is a directed edge uv where $u \in V$ and $v \in P(u)$. A jumping edge need not be in E.

The scaling algorithm assumes that the profit function d takes on nonnegative integral values. It works in a number of "scales", each scale finding an approximate optimum flow, the last such flow being optimum. We define approximate optimum by relaxing Frank's optimality conditions as follows (see [GoT, GaT89]). For a feasible 0-1 flow x, let RG be its residual graph. A *1-optimum flow* consists of a feasible 0-1 flow x, plus a potential function p, such that

$$d(uv) \leq p(v) - p(u) + 1 \qquad \text{for each residual edge } uv, \tag{3a}$$

$$p(u) \leq p(v) \qquad \text{for each jumping edge } uv. \tag{3b}$$

Dropping the $+1$ term in (3a) (and keeping (3b) unchanged) gives conditions equivalent to those presented in [F82] that guarantee a flow is optimum. An edge uv of G is *satisfied* if its corresponding residual edge satisfies Frank's optimality condition; otherwise uv is *violated.* The $+1$ term allows the scaling algorithm to achieve increased efficiency (by linking cost to path length, see [GaT89]).

We also use a second notion of approximate optimum. A *μ-optimum flow* [GaT88] consists of a feasible 0-1 flow x plus a potential function p such that (3b) holds and (3a) is replaced by the following condition. Define the *violation* of a residual edge uv to be $v(uv) = \max\{d(uv) - p(v) + p(u), 0\}$. (Thus (3$a$) amounts to $v(uv) \leq 1$.) The last condition for a μ-optimum flow is that the total violation $\sum\{v(e) \mid e \in RG\}$ is $O(\mu)$.

As in [F84], the scaling algorithm enlarges G to a new graph SFG, by adding new vertices s and t and certain new edges vs, tv. The new graph allows the final flow of one scale to be used as the starting point for finding the flow of the next scale. This involves pushing flow on the edges to s and from t. The edges incident to s and t have no given profit value, and s and t have no

potential. We say that a flow x on SFG that satisfies (1), plus a potential function p, is 1-optimum if it satisfies (3), with (3a) restricted to residual edges of G (no jumping edges are incident to s or t). Each scale maintains a 1-optimum flow on SFG. The goal of a scale is to void the edges incident to s and t.

The reader should realize that the family $\mathcal{F}$ need not be crossing after we enlarge the ground set from V to $V \cup \{s,t\}$. To circumvent this difficulty Frank shows that the given flow problem (1) is equivalent to an *induced problem.* This is a submodular flow problem whose family $\overline{\mathcal{F}}$ is intersecting and whose constraint function $\overline{b}$ is submodular on intersecting pairs of $\overline{\mathcal{F}}$; $\mathcal{F} \subseteq \overline{\mathcal{F}}$ and the original flow problem defines the same polyhedron as the induced problem [F82, F84]. The flow problem on SFG is defined by using the induced problem, considering $\overline{\mathcal{F}}$ to be an intersecting family on the ground set $V \cup \{s,t\}$. (This approach is needed even for connectivity orientation. We cannot take the family of sets in the original LP (1) to be intersecting, because of difficulties with V as a constraint set.) A set $S \in \overline{\mathcal{F}}$ is *i-tight* if its inequality in the induced problem holds with equality. This notion is used in Sections 4–5. A c-tight set is clearly i-tight, but not vice versa. ("c-tight" and "i-tight" refer to the crossing and intersecting problems, respectively. [F82] uses "strict" instead of "tight.")

Our scaling algorithm can be used in one of two modes, depending on whether or not "potential compression" is used. Compression speeds up the algorithm on some flow problems. But sometimes the simpler algorithm without compression achieves the desired efficiency. Furthermore in our current implementation compression can increase the space requirement. Thus we describe the algorithm both with and without compression. The compression operation keeps the flow closely related to a witness graph.

Now we present the the *Main Routine* of the scaling algorithm. It is given a 0-1 submodular flow problem having a nonnegative integral profit function. In addition it is given a feasible flow x. The Main Routine returns with x a maximum profit flow.

The Main Routine works by scaling the profit function. It starts by computing a new profit $\overline{d}(e)$ for each edge e, equal to $m+1$ times the given profit. Consider each value $\overline{d}(e)$ to be a binary number $b_1 b_2 \ldots b_k$ of $k = \lfloor \log(m+1)N \rfloor + 1$ bits. The routine maintains a variable $d(e)$ for each edge e equal to its profit in the current scale. The routine initializes each $d(e)$ to 0, each potential value $p(v)$ to 0, and x to the given feasible flow. The algorithm with compression changes every nonwitness edge of x into a 1-edge. Then the Main Routine repeats the following three steps, for index S going from 1 to k:

Double Step. For each $v \in V$ set $p(v) \leftarrow 2p(v)$. For each $e \in E$ set $d(e) \leftarrow 2d(e)+$ (bit b_S of $\overline{d}(e)$). Initialize graph SFG to graph G with isolated vertices s, t. For each edge $uv \in E$ that is now violated (for x and the new profit function d) do the following:

if $x(uv) = 0$ then add edges tu and vs to SFG, set $x(tu) = x(uv) = x(vs) = 1$;

if $x(uv) = 1$ then add edges tv and us to SFG, set $x(tv) = x(us) = 1$, $x(uv) = 0$.

Void Step. Call the Voiding Procedure with the 1-optimum flow x, p on SFG (every edge of G is satisfied in this flow). It returns a 1-optimum flow x, p on G.

Compress Step. (Executed for the algorithm with compression.) Compress the potential function (using the procedure of Section 4). This makes the flow μ-optimum, with every nonwitness edge a satisfied 1-edge. ∎

To motivate the Compress Step observe that any submodular flow problem has an optimum flow where every nonwitness edge is a 1-edge. Compression achieves a similar property.

We now show that the Main Routine is correct in the following sense. Corollary 3.1 proves that a 1-optimum or μ-optimum flow for the profit function $\overline{d}$ is actually optimum for the given profit function. Section 5 shows the Voiding Procedure operates as described, and Section 4 does the same for the compression procedure. Assuming these results, we need only show that at the start of each Void Step x, p is a 1-optimum flow on SFG with every edge of G satisfied.

To do this observe that the initialization of the Main Routine makes x, p a 1-optimum flow with every edge satisfied. We claim, inductively, that the Double Step gives a 1-optimum flow on SFG with every edge of G satisfied. In proof note that the changes to x have no effect on the left-hand side of inequalities (1). Thus the new flow is feasible, the sets $P(v)$ do not change and (3b) is preserved.

Each iteration of the Main Routine is called a *scale.* There are $O(\log(nN))$ scales. Each scale ends with a 1-optimum flow (in the algorithm without compression) or a μ-optimum flow (in the algorithm with compression) for the current cost function d. We note one more property related to efficiency: In the algorithm with compression, SFG has at most μ edges incident to s. In proof observe that any satisfied 1-edge scales up to a satisfied 1-edge (since a satisfied residual edge uv has $2d(uv) \leq 2p(v) - 2p(u)$, and if vu is a 1-edge the scaled-up residual cost of uv is at most the left-hand side). This implies that in the Double Step immediately after scaling up, only witness edges of x are violated. Thus $\leq \mu$ edges incident to s are added.

Now we briefly give the ideas of the Voiding Procedure, to introduce terminology and state some important properties. The procedure works like other mincost flow algorithms, by repeatedly

changing the potential function and augmenting the flow. An *augmenting path* is a directed path from s to t, consisting of residual edges of SFG and jumping edges, each of which satisfies its constraint of (3) with equality (edges incident to s or t have no constraint). (An augmenting path actually satisfies properties in addition to these. The complete definition is given in Section 5.) Given an augmenting path P from s to t, we *augment* the flow by changing the 0-edges of P to 1-edges and the 1-edges to 0-edges. Analogous to [F82] an augmenting path without shortcuts gives a new 1-optimum flow. Our algorithm uses a different criteria to get valid augmenting paths, the notion of topological numbers introduced in [GX]. But we also require that an augmenting path does not contain two consecutive jumping edges.

The Voiding Procedure finds augmenting paths "in batches," like the matching algorithm of Hopcroft and Karp [HK] and other cost-scaling algorithms. The algorithm is organized as a number of *searches*; each search is actually a depth-first search that finds a maximal sequence of augmenting paths and augments the flow along each one. The procedure stops when all edges incident to s and t are void.

The time-consuming part of our algorithm is computing the jumping edges. They must be recomputed after every augment since they depend on the flow. [Ga93a] provides an efficient oracle for jumping edges. It first constructs a representation of all c-tight sets. It can then be repeatedly called to return, for any given vertex v, all the jumping edges directed from v.

The scaling algorithm uses the oracle as follows. After every augment the oracle computes the representation of c-tight sets for the new flow. In the course of a search the oracle is called once for every vertex, plus once for every jumping edge on an augmenting path. This regime is the basis of the efficiency of the algorithm.

The time for the algorithm is estimated as follows. There are $O(\log(nN))$ scales. Define these parameters:

α = the number of augmenting paths in a scale;
λ = the total length of all augmenting paths in a scale;
σ = the number of searches in a scale;
μ = the greatest number of edges in a witness graph.

Note that we have already shown that in the algorithm with compression, $\alpha \leq \mu$. We shall see that the scaling algorithm without compression achieves $\alpha = O(m)$, $\lambda = O(m \log m)$, $\sigma = O(\sqrt{m})$. The scaling algorithm with compression is oriented toward problems that have a small witness graph. For example $\mu \leq 2kn$ for connectivity orientation and $\mu \leq n$ for dijoins. The algorithm

with compression achieves $\alpha = O(\mu)$, $\lambda = O(\mu\sqrt{n})$, $\sigma = O(\mu^{2/3})$ (the previous bounds on λ and σ still hold). In one scale the oracle representation is constructed α times. The oracle is called a total of $O(n\sigma + \lambda)$ times.

We close this section with remarks on the dijoin and orientation problems. As mentioned in Section 1, minimum-cost perfect bipartite matching is a special case of the minimum-cost dijoin problem. In proof consider a bipartite graph G with node sets X and Y, edges E and real-valued edge costs of magnitude at most N. As usual n is the total number of vertices. Construct digraph G' by increasing all edge costs by $nN + 1$, adding a set of edges of cost n^2N from X to Y to make G' Hamiltonian, and directing all edges from X to Y. A minimum-cost dijoin on G' is a minimum-cost perfect matching on G (if one exists). (Note that contracting a perfect matching on G' makes the Hamiltonian cycle into a connected Eulerian digraph, hence strongly connected.)

The dijoin problem is a special case of the orientation problem. In proof a dijoin problem on a digraph G with edge costs c is equivalent to the orientation problem where $k = 1$, the undirected graph contains two copies of uv for each directed edge uv of G, and the cost of orientation uv (vu) is 0 ($c(uv)$).

Finally note that the Main Routine of the scaling algorithm must be called with a feasible flow. For the dijoin problem we must provide a dijoin of the given digraph G. This is simple since any spanning tree of G (with edge directions ignored) is a dijoin (contracting a spanning tree makes G strongly-connected). For connectivity orientation we must provide a k-edge-connected orientation of the given undirected graph. This is done using the procedure of Section 8.

3. μ-optimum flows

This section presents the properties of 1-optimum and μ-optimum flows. It determines the values of the basic parameters λ and σ.

For uniformity when the algorithm does not use compression we use G as the trivial witness graph and take $\mu = m$. Thus whether or not compression is used, the flow obtained at the end of a scale is μ-optimum, there are $O(\mu)$ 0-edges and $\alpha \leq \mu$. We note that the arguments presented here simplify in the special case of 1-optimum flows. Also recall that parameter μ bounds the size of a witness graph for a submodular flow on the given graph G. In general μ does not bound the size of a witness graph for a flow on SFG. Indeed we do not know good bounds for witness graphs of such flows.

We observe that [F82] assumes the given profit function d is nonnegative. The only need for this assumption is to ensure that any feasible flow can be used to initialize the algorithm. Our

analysis can introduce negative values of d. But we can still rely on the properties of flows proved in [F82].

In this section we commit some slight abuses of notation: We often identify a flow x with its set of 1-edges, e.g., for flows x and x', $x \oplus x'$ denotes symmetric difference of the 1-edges of the flows. Also the symbol c is repeatedly used to denote a constant. Different occurrences of c can denote different constants.

We first bound the accuracy of a μ-optimum flow.

Lemma 3.1. Let x be a μ-optimum flow for a possibly negative profit function d, and let v be the violation function. Then any feasible flow x' has $dx'(E) \leq dx(E) + v(x \oplus x')$.

Proof. Define 0- and 1-edges using the flow x. Let v_0 (v_1) be the total violation of all 0-edges (1-edges). Let u_0 (u_1) be the total violation of all 0-edges (1-edges) in x'. Define a profit function $d^* : E \to \mathbf{Z}$ by setting $d^*(e) = d(e) - v(e)$ if e is a 0-edge, $d^*(e) = d(e) + v(e)$ if e is a 1-edge. Thus any residual edge e is satisfied with respect to d^* and p, so x is an optimum flow for d^* [F82]. This implies $dx'(E) - u_0 + u_1 = d^*x'(E) \leq d^*x(E) = dx(E) + v_1$. Rearranging gives $dx'(E) \leq dx(E) + v_1 - u_1 + u_0$ which amounts to the desired inequality. ∎

The lemma justifies the number of scales in the Main Routine.

Corollary 3.1. Let x be μ-optimum for the profit function Dd, where D is an integer $> m$. Then x is an optimum flow for d.

Proof. The violation of E is $\leq \mu \leq m$. Thus $Ddx'(E) \leq Ddx(E) + m < D(dx(E) + 1)$. Cancelling, $dx'(E) < dx(E) + 1$. Now integrality shows $dx'(E) \leq dx(E)$. ∎

The rest of the section concentrates on the Voiding Procedure. It assumes some basic properties of this procedure, which we now state. The properties are similar to other scaling algorithms [e.g., GaT89]; they are are proved at the end of Section 5.

We use the following notation. Fix any time during the Voiding Procedure. Let x, p be the current flow and potential, with d the profit function. Let x_0, p_0 be the 1-optimum flow at the start of the Voiding Procedure. Let x_-, p_- be the μ-optimum flow at the end of the previous scale, with d_- the profit function. (During the first scale, x_-, p_- is the flow constructed in the initialization of the Main Routine.) The Voiding Procedure maintains a 1-optimum flow, so x, p is a 1-optimum flow. The Double Step of the Main Routine shows any vertex v has $p_0(v) = 2p_-(v)$.

Define the multiset S of vertices currently joined to s by a 1-edge. Thus $|S|$ is the number of augmenting paths that remain to be found in the Voiding Procedure. Similarly define the multiset T as the vertices currently having a 1-edge from t. Clearly $|S| = |T|$.

The Voiding Procedure searches for augmenting paths from all edges sv, $v \in S$, simultaneously. It repeatedly increases potentials $p(v)$ for certain vertices v, always increasing the vertices v that remain in S by the same amount. The potential of a vertex in T is not changed. At any time define

Δ = the total increase in $p(v)$ since the start of the Voiding Procedure, for any v remaining in S.

We need one more property of the Voiding Procedure: Each search increases Δ and decreases $|S|$.

Lemma 3.2. At any time during the Voiding Procedure $|S|\Delta \leq c\mu + |x_- - x|$.

Proof. The argument is similar to [GoT] and other scaling algorithms [e.g., GaT89] comparing the flows x, p and x_-, p_-. Recall that x_- is a flow on the given graph $G = (V, E)$ while x is a flow on the enlarged graph SFG with added edges $\{vs \mid v \in S\} \cup \{tv \mid v \in T\}$. We can consider x_- to be a flow on SFG. To make the desired comparison we extend the profit and potential functions to SFG, as follows. Let SFE denote the set of edges of SFG. Let p^*, d^* be the potential and profit functions p, d or p_-, d_-. Extend the functions to SFG by the relations

$$p^*(s) = p^*(t) = 0, \quad d^*(vs) = -p^*(v) \text{ for } v \in S, \quad d^*(tv) = p^*(v) \text{ for } v \in T.$$

Note that this introduces possibly negative values for the profit function. Also the violation of any edge vs or tv is 0.

We prove two inequalities:

$$dx_-(E) \leq dx(E) - p(S) + p(T) + c\mu + |x_- - x|, \tag{4a}$$

$$dx(E) - p_0(S) + p_0(T) \leq dx_-(E) + c\mu. \tag{4b}$$

First observe that inequalities (4) imply the lemma: If $v \in S$ then $p(v) = p_0(v) + \Delta$, and if $v \in T$ then $p(v) = p_0(v)$. Thus combining the inequalities of (4) gives $|S|\Delta \leq c\mu + |x_- - x|$ as desired.

To prove (4a), note that x, p (as extended above) is a 1-optimum flow on SFG. Let v be the violation function for x. Lemma 3.1 implies $dx_-(SFE) \leq dx(SFE) + v(x_- \oplus x)$. Equivalently

$$dx_-(E) \leq dx(E) - p(S) + p(T) + v(x_- \oplus x).$$

Since x is 1-optimum and x_- has $O(\mu)$ 0-edges,

$$v(x_- \oplus x) = v(x_- - x) + v(x - x_-) \leq |x_- - x| + c\mu.$$

Combining the last two inequalities gives $(4a)$.

Next we prove $(4b)$. x_-, p_- (as extended above) is a μ-optimum flow on SFG. Let v_- be the violation function for x_-. Lemma 3.1 implies $d_-x(SFE) \leq d_-x_-(SFE) + v_-(x_- \oplus x)$. Thus

$$d_-x(E) - p_-(S) + p_-(T) \leq d_-x_-(E) + v_-(x_- \oplus x).$$

Since d is nonnegative and it results from scaling up d_-,

$$dx(E) - dx_-(E) \leq 2d_-x(E) - 2d_-x_-(E) + |x - x_-|.$$

Combining the last two inequalities and using the relations $p_0(v) = 2p_-(v)$ and $v_-(x_- \oplus x), |x - x_-| \leq c\mu$ gives $(4b)$. ∎

Next we estimate λ. The argument extends that of [GaT89]. It is convenient to work with a related quantity: Define κ_j to be the number of residual edges in augmenting paths in the first j augmenting paths. Each jumping edge except the last in an augmenting path is followed by a residual edge. Thus the first j augmenting paths have length $\leq 2\kappa_j + j$, and $\lambda \leq 2\kappa_\alpha + \alpha$.

For an integer j, let the jth augmenting path of a scale be found when $\Delta = \Delta_j$.

Lemma 3.3. For any j, $\kappa_j \leq c\mu + \sum_{i=1}^{j} \Delta_i$.

Proof. Define the "profit-length" $dl(e)$ of a residual edge e to be $d(e) - 1$ ($d(e)$ is the residual profit). We estimate the profit-length of the first j augmenting paths.

Assume x, p is the flow immediately after the jth augmenting path is found. Recall the sets S and T are defined for x, i.e., after the jth augment. Define S_0 and T_0 similarly for the flow x_0. (Thus $|S_0|$ is the number of augmenting paths found in the scale.)

Consider an augmenting path P from a to b, where $a \in S_0, b \in T_0$. Each residual edge $uv \in P$ has $dl(uv) = d(uv) - 1 = p(v) - p(u)$. Each jumping edge $uv \in P$ has $p(v) = p(u)$. Thus $dl(P) = p(b) - p(a)$. If P is found when the dual adjustment is Δ then $dl(P) = p_0(b) - p_0(a) - \Delta$.

The total profit-length of the first j augmenting paths equals both $dx(E) - dx_0(E) - \kappa_j$ and $p_0(T_0 - T) - p_0(S_0 - S) - \sum_{i=1}^{j} \Delta_i$. Thus

$$\kappa_j = dx(E) - dx_0(E) + p_0(S_0 - S) - p_0(T_0 - T) + \sum_{i=1}^{j} \Delta_i.$$

The Double Step of the Main Routine derives flow x_0 from x_- by saturating 0-edges with $d(uv) \geq p(v) - p(u) + 1$ and voiding 1-edges with $d(uv) \leq p(v) - p(u) - 1$. Thus $dx_0(E) \geq dx_-(E) + p_0(S_0) - p_0(T_0)$. This implies

$$\kappa_j \leq dx(E) - dx_-(E) - p_0(S) + p_0(T) + \sum_{i=1}^{j} \Delta_i.$$

Combining this with (4b) gives $\kappa_j \le c\mu + \sum_{i=1}^{j} \Delta_i$ as desired. ∎

Corollary 3.2. At any time during the Voiding Procedure $|S|\Delta \le cm$. $\alpha \le m$. $\sigma \le c\sqrt{m}$. $\lambda = O(m \log m)$.

Proof. The first part follows from Lemma 3.2 since $|x_- - x| \le m$. The bound on α follows from the Double Step.

To bound σ, the first part implies that either $|S|$ or Δ is $\le \sqrt{cm}$. As mentioned, each search increases Δ and decreases $|S|$. This gives the desired bound for σ.

To bound λ, Lemma 3.3 implies that $\lambda \le c\mu + 2\sum_{i=1}^{\alpha} \Delta_i$. The first part shows $\Delta_i \le cm/(\alpha + 1 - i)$, so $\sum_{i=1}^{\alpha} \Delta_i = O(m \log m)$. This gives the desired bound for λ. ∎

The rest of this section derives bounds like the corollary using parameter μ rather than m. We first claim

$$|S|\Delta_{j+1} \le c\mu + \sum_{i=1}^{j} \Delta_i. \tag{5}$$

In proof, applying Lemma 3.2 immediately before the $(j+1)$-st augment shows $|S|\Delta_{j+1} \le c\mu + |x_- - x|$. Write $|x_- - x| \le |x_- - x_0| + |x_0 - x|$. We have $|x_- - x_0| \le \alpha \le \mu$. Also $|x_0 - x| \le \kappa_j$ since an edge of $x_0 - x$ occurs as a 1-edge in an augmenting path. Combining these inequalities implies $|S|\Delta_{j+1} \le c\mu + \kappa_j$. Now Lemma 3.3 implies (5).

Define these quantities for any integer $k \ge 0$:

a_k = the number of augmenting paths found when $\Delta = k$;

$s_k = \mu - \sum_0^k a_i$;

$\ell_k = c\mu + \sum_0^k i a_i$.

s_k is the value $|S|$ after all augments for $\Delta = k$ are done. (The scale starts with $|S| = \alpha \le \mu$.) The sequence ends when $s_k = 0$. (5) applied when j is the largest index having $\Delta_j \le k$ becomes $s_k(k+1) \le \ell_k$. Rearranging gives

$$\mu(k + 1 - c) \le \sum_{i=0}^{k} (k + 1 + i) a_i. \tag{6}$$

Now consider all sequences a_k that satisfy (6), allowing a_k to take on nonnegative real values. Fix a particular sequence a_k (and its associated values s_k and ℓ_k) by the relations $a_k = 0$ for $k < c$, $a_k = s_{k-1}/(2k+1)$ for $k \ge c$. Summing the relations $(2i+1)a_i = s_{i-1}$ for $c \le i \le k$ shows that the fixed values a_k satisfy (6) with equality for any $k \ge c$.

We claim that among nonnegative sequences a'_k satisfying (6), the fixed sequence a_k maximizes both s_k and $\ell_k + ks_k$ for every index k. To prove this consider any a'_k satisfying (6). Let h be the smallest index with $a'_h \neq a_h$. It is easy to see that $a'_h > a_h$. Decrease a'_h by $(a'_h - a_h)$ and increase a'_{h+1} by the same amount. This gives another sequence satisfying (6), since the right-hand side of (6) increases for $k > h$. The value s'_k increases for $k = h$ and is unchanged otherwise; the value $\ell'_k + ks'_k$ increases for $k > h$ and is unchanged otherwise. Since the new value a'_h equals a_h, repeating this procedure proves the claim.

We show that for $k \geq c - 1$,

$$s_k = \mu \prod_{i=c}^{k} (1 - \frac{1}{2i+1}) \text{ and } \ell_k = (k+1)s_k.$$

The first relation follows since $s_{c-1} = \mu$ and for $k \geq c$, $s_k = s_{k-1} - a_k = s_{k-1}(1 - 1/(2k+1))$. The second relation follows from the inequality preceding (6).

Next we show for $k \geq c - 1$,

$$s_k \leq \frac{c\mu}{\sqrt{k}} \text{ and } \ell_k \leq c\mu\sqrt{k}.$$

In proof note that $\prod_{i=2c+1}^{2k+1}(1 - 1/i) = 2c/(2k+1)$. The product is $\geq \prod_{i=c}^{k-1}(1 - 1/(2i+1))^2$, so $\prod_{i=c}^{k-1}(1 - 1/(2i+1)) \leq c/\sqrt{k}$. This implies the bound on s_k. The bound on ℓ_k follows immediately.

Corollary 3.3. $\alpha \leq \mu$. $\sigma = O(\mu^{2/3})$. $\lambda = O(\min\{\mu\sqrt{n}, \mu^{2/3}(nm)^{1/3}\})$.

Proof. The bound on α was proved in Section 2.

To bound σ observe that for $k = \mu^{2/3}$, $s_k \leq c\mu^{2/3}$. This implies that the scaling algorithm always has either $\Delta \leq \mu^{2/3}$ or $|S| \leq c\mu^{2/3}$. As mentioned above, each search increases Δ and decreases $|S|$. This gives the desired bound.

To bound λ first take j as the largest index with $\Delta_j \leq n$. Then $\kappa_j \leq \ell_n + ns_n$. For larger values of j the total augmenting path length is at most ns_n. Now $\ell_n + ns_n \leq c\mu\sqrt{n}$ implies the first desired bound, $\lambda = O(\mu\sqrt{n})$.

For the second bound on λ set $k = (nm/\mu)^{2/3}$ and take j as the largest index with $\Delta_j \leq k$. Then $\kappa_j \leq \ell_k + ks_k = c\mu\sqrt{k} = c\mu^{2/3}(nm)^{1/3}$. Corollary 3.2 shows that for larger values of j the total augmenting path length is at most $cnm/k = c\mu^{2/3}(nm)^{1/3}$. This gives the desired bound. ∎

4. Potential compression

This section describes the postprocessing that is done at the end of a scale for some submodular flow problems. It calculates a better potential function to ensure a good starting point for the next scale. Specifically potential compression converts the last 1-optimum flow found in a scale into a μ-optimum flow where every nonwitness edge is a satisfied 1-edge.

Throughout this section we fix a submodular flow on G, and its witness graph W. A *witness residual edge* is a residual edge (for the submodular flow) whose corresponding edge of G belongs to W (e.g., a residual edge e with e^R a 1-edge in W). Similarly for *nonwitness residual edge.*

We achieve the goal of potential compression in two steps: The first (and main) step modifies the potential function, achieving properties stated in Lemma 4.2. The second step makes every nonwitness edge a 1-edge and achieves the desired goal.

All results of this section except Lemma 4.8 hold for an arbitrary submodular flow problem. Lemma 4.8 shows our postprocessing algorithm achieves the desired goal for the connectivity orientation problem. Our results might need to be elaborated to achieve the goal for a different submodular flow problem (although we know of no such problem, and furthermore connectivity orientation requires the full generality of our results).

A natural approach to get a μ-optimum flow is to complement the flow in nonwitness edges that are violated. This achieves total violation $O(\mu)$ and gives a feasible flow. However complementing is not valid: Complementing can destroy a c-tight set, thus enlarging a $P(v)$ set, thus breaching condition (3*b*). Our more involved approach models the problem as the dual of a minimum-cost circulation problem.

We begin with a crucial property.

Lemma 4.1. A nonwitness residual edge e has e^R a jumping edge.

Proof. The witness edges ensure that (2) holds. Thus if a nonwitness residual edge uv enters a set S, (2) holds with strict inequality (for that S). This implies $u \in P(v)$. ∎

In the dijoin problem, any edge e of G is a jumping edge (but not conversely). This slightly simplifies the compression algorithm. No such simplification occurs for connectivity orientation.

The first consequence of the lemma is that a nonwitness 0-edge uv has $d(uv) \in \{0,1\}$ and $p(v) \in \{p(u), p(u)-1\}$. In proof the lemma and (3*b*) imply $p(u) \geq p(v)$. Furthermore $0 \leq d(uv) \leq p(v) - p(u) + 1$ implies $p(v) \geq p(u) - 1$ and $d(uv) \leq 1$.

We solve the potential compression problem by adding a quantity $q(v)$ to each potential $p(v)$ (so the new potential function is $p + q$). The function q is found as the optimal price function for a minimum-cost circulation problem.

Recall the following definitions of the minimum-cost circulation problem and its dual. For a digraph $G = (V, E)$ with functions c and u having domain E and range $\mathbf{R}$ and $\mathbf{R} \cup \{+\infty\}$ respectively, the *minimum-cost circulation problem* is to find a flow function $f : E \to \mathbf{R}$ solving this LP:

$$\begin{aligned} &\text{minimize } cf(E) \text{ subject to} \\ &\qquad \delta_f(v) - \rho_f(v) = 0 \qquad \text{for each } v \in V \\ &\qquad 0 \le f(e) \le u(e) \qquad \text{for each } e \in E \end{aligned}$$

An *uncapacitated* edge e has infinite capacity, $u(e) = \infty$. The dual LP involves dual variables $q : V \to \mathbf{R}$ (the *price function*) and $U : E \to \mathbf{R}$. It is this LP:

$$\begin{aligned} &\text{maximize } -uU(E) \text{ subject to} \\ &\qquad q(u) - q(v) - U(uv) \le c(uv) \qquad \text{for each } uv \in E \\ &\qquad U(e) \ge 0 \qquad \text{for each } e \in E \end{aligned}$$

Now we describe the graph CG for the circulation problem. CG has vertices V; its edges are the jumping edges and the residual edges, with the following costs and capacities:

uv a jumping edge: $c(uv) = p(v) - p(u)$; $u(uv) = \infty$
uv a nonwitness residual edge: $c(uv) = \max\{p(v) - p(u), -1\}$; $u(uv) = \infty$
uv a witness residual edge: $c(uv) = p(v) - p(u) - d(uv)$; $u(uv) = 1$

A jumping edge has nonnegative cost in CG. A nonwitness edge costs -1 if $p(u) > p(v)$ and 0 otherwise, since Lemma 4.1 and $(3b)$ imply $p(u) \ge p(v)$. A witness edge costs -1 if it is violated and has nonnegative cost otherwise, by $(3a)$. (The term $d(uv)$ in the definition of the cost refers to the residual profit of uv.) We shall use the phrases "nonwitness edge of CG" and "uncapacitated edge of CG." The former means a nonwitness residual edge, and the latter means a nonwitness edge or a jumping edge (the reader will not be confused by the fact that a jumping edge is not a witness edge).

Lemma 4.2. Consider a given 1-optimum submodular flow with potential function changed to $p+q$, where q is a feasible price function for the circulation problem on CG. Then $(3b)$ holds, all nonwitness 1-edges are satisfied and all nonwitness 0-edges uv have $p(u)+q(u) = p(v)+q(v)$. The total violation of all witness edges and the dual objective value have nonpositive sum.

Proof. In the dual problem the constraint for a jumping edge uv is $q(u) - q(v) \leq p(v) - p(u)$. This amounts to $(3b)$.

The constraint for a nonwitness edge uv is $q(u) - q(v) \leq \max\{p(v) - p(u), -1\}$. The right-hand side is ≤ 0 in general, and $= -1$ if uv corresponds to a violated 1-edge (for the latter, equality in $(3a)$ implies $0 \geq d(uv) = p(v) - p(u) + 1$). Now combining with $(3a)$ gives $d(uv) \leq p(v) + q(v) - (p(u) + q(u))$ if uv is originally satisfied or if it corresponds to a violated 1-edge.

If uv is a nonwitness 0-edge then as noted after Lemma 4.1, $p(v) \in \{p(u), p(u) - 1\}$. For both values the above dual constraint becomes $q(u) - q(v) \leq p(v) - p(u)$. Since the jumping edge constraint for vu is $q(v) - q(u) \leq p(u) - p(v)$, equality holds as desired.

Finally the constraint for a witness edge uv is $q(u) - q(v) - U(uv) \leq p(v) - p(u) - d(uv)$. Rearranging, $d(uv) - (p(v) + q(v)) + (p(u) + q(u)) \leq U(uv)$. This and nonnegativity of U imply violation $v(uv) \leq U(uv)$. ∎

We shall see that the lemma implies all nonwitness 0-edges become satisfied 1-edges in the second step of the postprocessing. So the lemma implies that a feasible price function with dual objective $\geq -2\mu$ solves our potential compression problem. Thus we can achieve our goal by showing that a minimum-cost circulation on CG exists and costs $\geq -2\mu$. Most of the rest of this section is devoted to this.

First observe that a minimum-cost circulation exists: Any uncapacitated edge uv of CG has $c(uv) \geq p(v) - p(u)$. Thus any uncapacitated cycle has nonnegative cost.

Choose a minimum-cost circulation f so it minimizes the total flow through nonwitness edges $f\{e \mid e \text{ a nonwitness edge}\}$; furthermore subject to that constraint, it minimizes the number of edges with positive flow.

We first show the number of edges with positive flow is limited.

Lemma 4.3. Let G be an arbitrary directed network where a minimum-cost flow/circulation exists. Then there is a minimum-cost flow such that no undirected cycle of uncapacitated edges has positive flow in all its edges.

Proof. An uncapacitated edge with positive flow has residual capacity in both directions. Thus an undirected cycle of such edges gives two oppositely oriented directed cycles in the residual graph. Each cycle must have residual cost 0. Pushing more flow along one of the cycles voids an edge. ∎

In our context the proof shows that no undirected cycle of uncapacitated edges has positive flow in all its edges.

Now we give the plan for the rest of the proof. Consider the orientation problem. We will prove that any nonwitness edge has $\leq k$ units of flow. Since the lemma shows $\leq n-1$ such edges have flow, the uncapacitated edges cost $\geq -kn$. The witness edges cost $\geq -\mu$ (since they have capacity 1 and cost ≥ -1). This shows the minimum cost is $\geq -kn - \mu \geq -(3/2)\mu$ as desired.

Define digraph R, a subgraph of the residual graph for circulation f: R contains all jumping edges, plus the reverse of any uncapacitated edge with positive flow.

Lemma 4.4. A directed cycle in R consists of jumping edges or their reversals.

Proof. The residual cost of an edge uv in R is at most $p(v) - p(u)$. (If uv is a reversed flow edge then the definition of CG shows $c(vu) \geq p(u) - p(v)$, so $-c(vu) \leq p(v) - p(u)$.) Thus a directed cycle C in R has nonpositive residual cost, so it must cost 0. If C contains the reverse of a nonwitness edge then pushing flow along the C violates the choice of f. ∎

We recall from [F82] that $S \subseteq V$ is i-tight if it is the intersection of V with zero or more c-tight sets. In fact [F82, Lemma 12] shows S is i-tight set iff $\overline{S}$ is the union of zero or more disjoint sets H, each of whose complements $\overline{H}$ is c-tight. Call H a *hole of* S. (In connectivity orientation, the holes of an i-tight set are disjoint sets H having $\delta_{RG}(H) = k$.) The union and intersection of two intersecting i-tight sets is i-tight; similarly the union and intersection of two crossing c-tight sets is c-tight [F82, Lemma 2].

For any vertex x, let X be the set of all vertices reachable from x in R.

Lemma 4.5. X is i-tight.

Proof. Obviously $P(x) \subseteq X$. Inductively assume some set $X' \subseteq X$ is i-tight, and let uv be an edge of R leaving X'. It suffices to show that $X' \cup P(v)$ is i-tight. Any edge uv of R has $v \in P(u)$ or $u \in P(v)$. For our edge $u \in P(v)$. Thus X' and $P(v)$ are intersecting i-tight sets, and their union is i-tight. ∎

Lemma 4.6. Let Z be an i-tight set. No jumping edge or nonwitness edge joins two distinct holes of Z.

Proof. Let H be a hole of Z. $H \cup Z$ is i-tight. Thus any $u \in H$ has $P(u) \subseteq H \cup Z$ (by the definition of $P(u)$). This means no jumping edge goes from H to another hole. A nonwitness edge joins the ends of a jumping edge (Lemma 4.1). ∎

Recall that a circulation can be partitioned into simple cycles of flow; call this the *flow decomposition.* For vertices x, y, define X as above, and let Y be the hole of X containing y (if it exists).

Lemma 4.7. Let xy be a nonwitness edge in a cycle C of the flow decomposition. Then Y exists and some witness edge of C leaves Y.

Proof. To prove the hole Y exists it suffices to show $y \notin X$. If on the contrary $y \in X$, then the reverse flow edge yx completes a cycle in R, contradicting Lemma 4.4.

Let uv be the first edge on C after y that leaves Y. It suffices to show that uv is not an uncapacitated edge. Assume it is uncapacitated, i.e., a jumping edge or a nonwitness edge. Lemma 4.6 shows $v \in X$. But since vu is in R we get $u \in X$, a contradiction. ∎

Now we complete the analysis of potential compression for connectivity orientation.

Lemma 4.8. For a connectivity orientation problem, a minimum-cost circulation on CG costs $\geq -2\mu$.

Proof. The preceding discussion shows we need only prove that a nonwitness edge xy has $f(xy) \leq k$. Suppose the contrary. Lemma 4.7 shows $> k$ witness edges leave y's hole Y. This contradicts the fact that $\overline{Y}$ is c-tight, i.e., $\rho_{RG}(\overline{Y}) = k$. ∎

We turn to the second step of potential compression. It changes all nonwitness 0-edges into 1-edges. To verify this change is correct, first note it gives a valid submodular flow. Second a new 1-edge uv is satisfied, since its residual edge has $d(uv) \leq 0 = p(v) + q(v) - (p(u) + q(u))$ (by Lemma 4.2). Finally we must check (3*b*) (recall from the start of this section that in general this change does not preverve (3*b*)). The proof is by induction, changing one 0-edge uv at a time, using the following result.

Lemma 4.9. Consider an arbitrary submodular flow, with uv a 0-edge, $p(u) = p(v)$ and $u \in P(v)$. Making uv a 1-edge preserves $(3b)$.

Proof. Making uv a 1-edge has the same effect on the submodular flow constraints (1) as adding edges sv and ut and pushing 1 unit of flow on each. As defined in Section 5, the latter operation is called executing the swap v, u, and the hypothesis shows v, u is an eligible swap. Now Lemma 5.2 shows $(3b)$ is preserved. ∎

We close the section with an efficient implementation of the potential compression algorithm. For any $v \in V$ define $u(v) = k \max\{\rho_{RG}(v), \delta_{RG}(v)\}$. There is a minimum-cost circulation on CG such that the flow through any vertex v is at most $u(v)$. In proof, we can always replace a unit of flow in two jumping edges vw and wx by a unit of flow in the jumping edge vx, without changing the cost. (The jumping edges are transitive.) Doing this as many times as possible leaves a circulation where each v has every jumping edge directed from v void, or every jumping edge directed to v void. The desired bound on the flow through v follows since we can assume any nonjumping edge of CG has $\leq k$ units of flow.

We reformulate the circulation problem as a degree-constrained subgraph problem (equivalently, a capacitated transportation problem) on a bipartite multigraph BG, as follows. For each $v \in V$, BG has vertices v^-, v^+, each with degree-constraint $u(v) + 1$. BG has these edges:

v^-v^+ for each $v \in V$, multiplicity $u(v) + 2$, cost 0;
v^+w^- for each jumping edge vw of CG, multiplicity $u(v) + 1$, cost $c(vw)$;
v^+w^- for each nonwitness edge vw of CG, multiplicity $k + 1$, cost $c(vw)$;
v^+w^- for each witness edge vw of CG, multiplicity 1, cost $c(vw)$.

A degree-constrained subgraph of BG is a subgraph where the degree of any vertex equals its degree constraint. A minimum-cost circulation on CG corresponds to a minimum-cost degree-constrained subgraph D of BG. Recall that if D is an optimum degree-constrained subgraph and y is an optimum price function, then

$$y(v) + y(w) \leq c(vw) \qquad \text{for each edge } vw \notin D,$$
$$y(v) + y(w) \geq c(vw) \qquad \text{for each edge } vw \in D.$$

In the optimum subgraph D corresponding to the flow described above, for each $v \in V$ there is a copy of edge v^-v^+ in D and another copy not in D. Thus $y(v^-) = -y(v^+)$. Now it is easy to

check that the price function $q(v) = y(v^+)$ is an optimum price function on CG (e.g., it satisfies the complementary slackness conditions).

Theorem 4.1. A 1-optimum flow on G can be converted into a μ-optimum flow with every non-witness edge a satisfied 1-edge, in time $O(\sqrt{km}\, n^2 \log n)$.

Proof. We solve the degree-constrained subgraph problem using the cost-scaling algorithm of [GaT89]. On a graph with n vertices, m edges, total degree-constraints U, and integral edge costs of magnitude at most N, the algorithm finds a minimum-cost subgraph in time $O((\sqrt{U}m + U \log U) \log (nN))$ [GaT89, Theorem 3.2]; it finds an optimum price function y in additional time $O(m)$ [GaT89, Corollary 2.2].

We make one change to reduce the factor $\log (nN)$ in the time bound to $\log n$. The given algorithm uses $\log (nN)$ scales. It suffices to reduce this number to $\log n$. The purpose of the first $\log N$ scales is to get a 1-optimum solution to the degree-constrained subgraph problem using the given cost function. Set D to the subgraph with containing $u(v) + 1$ copies of edge v^-v^+ for each $v \in V$; set $y(v^-) = y(v^+) = 0$ for each $v \in V$. Since all given costs are ≥ -1, this is a 1-optimum solution for the given costs. Thus the algorithm need only use $\log n$ scales.

BG has $O(n)$ vertices, $O(n^2)$ edges, and total degree-constraints $U = O(km)$. We conclude that the potential compression problem is solved in time $O((\sqrt{km}\, n^2 + km \log n) \log n)$. Since $m \geq kn$, $\sqrt{km}\, n^2 \geq kn^{2.5} \geq km \log n$. ∎

Note that the algorithm uses $O(n^2)$ space, because of jumping edges.

5. Augmenting

This section presents the two algorithms that implement a search, combining them in the Voiding Procedure. Unlike [F82] we organize the notion of augmenting paths around the concept of a swap, generalizing [GX]. Section 5.1 discusses the properties of swaps and augmenting paths. Section 5.2 gives the Augmenting Procedure, which finds augmenting paths and augments the flow. Section 5.3 gives gives the Potential-changing Procedure, which adjusts potentials to create an augmenting path. Finally Section 5.4 combines these two procedures to get the Voiding Procedure.

5.1. Swaps and augmenting paths

Fix a submodular flow on SFG. If uv is a jumping edge , i.e., $v \in P(u)$, then we call u, v a *swap* and we *execute swap* u, v by adding edges su and vt to SFG and pushing 1 unit of flow on each edge. Executing a swap gives a feasible flow. In proof observe that if S is a set whose constraint (1) fails after the swap, then before the swap S is a c-tight set containing u but not v. But $v \in P(u)$ implies there is no such set.

In the following discussion suppose we execute a swap u, v. Define the sets $P(\cdot)$ before executing the swap and sets $P'(\cdot)$ after.

Lemma 5.1. $P'(v) = P(u)$. $P'(u) \subseteq P(u)$. For any vertex w, $v \notin P(w)$ implies $P'(w) = P(w)$ and $v \in P(w)$ implies $P(w) \oplus P(u) \subseteq P'(w) \subseteq P(w) \cup P(u)$.

Proof. To prove the first two relations note that $P(u)$ is i-tight after the swap, and no subset of V containing v is i-tight unless it contains u.

Suppose $v \notin P(w)$. Thus $u \notin P(w)$ and $P(w)$ is i-tight after the swap. Furthermore no subset of $P(w)$ becomes i-tight. Thus $P'(w) = P(w)$.

For the rest of the argument suppose $v \in P(w)$. Thus $P(w) \cup P(u)$ is i-tight before the swap. It remains i-tight after the swap so $P'(w) \subseteq P(w) \cup P(u)$, giving one of the two desired inclusions.

To prove the second inclusion consider the set $P'(w) \cap \{u, v\}$. This set is nonempty since otherwise $P'(w)$ was i-tight before the swap, so $P(w) \subseteq P'(w)$, contradicting the hypothesis on $P(w)$. The set does not equal $\{v\}$ since this implies $P'(w)$ is not i-tight after the swap. If the set equals $\{u, v\}$ then $P'(w)$ was i-tight before the swap, and it contains both w and u. Thus $P(w) \cup P(u) \subseteq P'(w)$. This implies $P'(w) = P(w) \cup P(u)$, giving the second inclusion.

The remaining possibility is that $P'(w) \cap \{u, v\} = \{u\}$. The second inclusion is equivalent to the two inclusions $P(w) \subseteq P'(w) \cup P(u)$ and $P(u) \subseteq P'(w) \cup P(w)$. We prove them as follows.

Suppose z is a vertex such that $P'(w)$ and $P(z)$ both contain z and v is in $P(z)$ (note $v \notin P'(w)$ by assumption). We prove $P'(w) \cup P(z)$ is i-tight before the swap. By hypothesis $P'(w)$ and $P(z)$ are intersecting sets or they nest. Let $\overline{\mathcal{F}}$ be the intersecting family of the induced submodular flow problem, with $\overline{b}$ its submodular function. Since the function $\rho_x - \delta_x$ is modular, the "slack" function $sl = \overline{b} - \rho_x + \delta_x$ is submodular on intersecting pairs of $\overline{\mathcal{F}}$. Evaluating sl before the swap is executed, $sl(P'(w)) + sl(P(z)) \geq sl(P'(w) \cap P(z)) + sl(P'(w) \cup P(z))$. We show this inequality amounts to $1 \geq 1$: Since $P(z)$ is i-tight, $sl(P(z)) = 0$. Since $P'(w)$ contains u but not v and becomes i-tight after the swap, $sl(P'(w)) = 1$. Since x is feasible, $sl(P'(w) \cup P(z)) \geq 0$. Since $P'(w) \cap P(z)$ is a

proper subset of $P(z)$, $sl(P'(w) \cap P(z)) \geq 1$. This shows the submodular inequality amounts to $1 \geq 1$, so all inequalities hold with equality. In particular $sl(P'(w) \cup P(z)) = 0$ as desired.

Applying the previous paragraph to $z = u$ shows $P'(w) \cup P(u)$ is i-tight. Since this set contains w, $P(w) \subseteq P'(w) \cup P(u)$, the first desired inclusion. Similarly applying the previous paragraph to $z = w$ shows $P'(w) \cup P(w)$ is i-tight. Since this set contains u, $P(u) \subseteq P'(w) \cup P(w)$ as desired. ∎

Two more notions are needed to define augmenting paths. Fix a 1-optimum flow on SFG. An edge is *eligible* if it is an edge of G with equality in (3), or a residual edge incident to s or t. A *topological numbering* is a function $\tau : V \to \mathbf{Z}_+$ such that any eligible edge uv of G has $\tau(u) \geq \tau(v)$, with strict inequality for uv a residual edge. Note that τ is undefined on vertices s and t. An eligible jumping edge uv is *strongly eligible* if $\tau(u) = \tau(v)$.

Suppose we have a flow and potential function satisfying (3b), with τ a topological numbering. The first part of this result is used in the proof of Lemma 4.9.

Lemma 5.2. Let uv be a jumping edge. If uv is eligible then executing swap u, v keeps (3b) true for all jumping edges. If uv is strongly eligible then executing swap u, v keeps τ a topological numbering.

Proof. For both assertions we consider a new jumping edge xy. Since $P(x)$ changes Lemma 5.1 shows $v \in P(x)$ and $y \in P(u)$.

The first part follows since $p(y) \geq p(u) = p(v) \geq p(x)$.

For the second part, if xy is eligible then $p(x) = p(y)$, equality holds throughout the previous inequality and so uy and xv are eligible jumping edges (or self-loops). Thus $\tau(y) \leq \tau(u) = \tau(v) \leq \tau(x)$. ∎

An *augmenting path* P is a directed path from s to t consisting of eligible edges with each jumping edge strongly eligible; in addition no two jumping edges are consecutive. (The last requirement is for simplicity: The algorithm finds paths with this property, but Lemma 5.3 holds even with consecutive jumping edges.) Recall that we *augment* the flow by changing the 0-edges of P to 1-edges and the 1-edges to 0-edges.

Let P be an augmenting path for a 1-optimum flow with a topological numbering.

Lemma 5.3. Augmenting P gives a 1-optimum flow with a topological numbering.

Proof. First suppose P has no jumping edges. The augment reverses the edges of P in RG. Thus ρ_{RG} is unchanged and the new flow satisfies (2). Similarly the c-tight sets do not change, so the jumping edges do not change and (3b) still holds. The reverse of an edge on P satisfies (3a) with strict inequality. Thus (3a) continues to hold and no new residual edge is eligible. This also implies that τ remains valid.

In the general case let $P = sa \ldots uvw \ldots bt$ where uv is the first jumping edge of P. Let us execute the augment by adding edges su, sv, ut, vt to SFG and augmenting three paths: first $sa \ldots ut$, then $suvt$ and finally $svw \ldots bt$. Augmenting these three paths has the same effect as augmenting P (flow in the newly added edges cancels out). The previous paragraph shows the first augment gives a flow as desired, with the same jumping edges as before. Augmenting the path $suvt$ amounts to executing swap u, v. Lemma 5.2 shows the flow remains as desired.

We can prove the last augment is valid by induction, if we can show that any jumping edge xy of the third path remains valid after the swap u, v. Since uv is a jumping edge vw is residual and $\tau(v) > \tau(w) \geq \tau(x)$. Now the definition of τ implies that either $v \notin P(x)$ or $p(v) > p(x)$. If $v \notin P(x)$ then Lemma 5.1 shows $P(x)$ is not changed by the swap. Suppose $v \in P(x)$. Then $p(u) = p(v) > p(x) = p(y)$. Thus $y \in P(x) - P(u)$ and Lemma 5.1 shows $y \in P'(x)$. ∎

5.2. The Augmenting Procedure

The Augmenting Procedure finds augmenting paths, using depth-first search. For each $v \in S$ it either voids edge vs or increases $p(v)$.

The *Augmenting Procedure* uses these data structures: Path P, which is grown to an augmenting path, is a list of vertices managed as a stack. T denotes the largest topological number. Each vertex u has an associated set of vertices $J(u)$.

The Augmenting Procedure examines each residual edge sa ($a \in S$). For each such a it initializes path P to s, a and all sets $J(u)$ to $\emptyset$. Then it executes the following steps, until either the Dead-end Step or the Complete Step stops. (In the first case no augmenting path containing a exists; in the second case the flow gets augmented.) The Augmenting Procedure continues by examining the next edge sa (if no augmenting path containing a exists, edges parallel to sa are skipped). The Augmenting Procedure halts when all residual edges from s have been examined.

Scan Step. Let wu be the last edge in P. Select an edge uv according to (i) below if possible else according to (ii):

(i) Let uv be an eligible residual edge.

(ii) If wu is a residual edge, let uv be an eligible jumping edge with $v \notin J(u)$ and $\tau(v)$ as large as possible.

If edge uv exists then add v to the end of P, and go to the Complete Step if $v = t$ (P is an augmenting path) else go to the Scan Step. If no edge uv exists then go to the Dead-end Step.

Dead-end Step. (wu is the last edge of P.) Delete u from P. If wu is a jumping edge then add u to $J(w)$ and go to the Scan Step. Otherwise (wu is a residual edge) set $T \leftarrow T + 1$, and for $x \in \{u\} \cup J(u)$, set $p(x) \leftarrow p(x) + 1$ and $\tau(x) \leftarrow T$; then stop if $P = s$ (no augmenting path containing a exists) else go to the Scan Step.

Complete Step. For each jumping edge uv of P, for $x \in \{u\} \cup J(u)$, set $\tau(x) \leftarrow \tau(v)$. Augment the flow along P. Stop. ∎

To prove the procedure correct we start with five facts (a)–(e). These facts assume that the procedure maintains a 1-optimum flow with a topological numbering. In an execution of the Augmenting Procedure a vertex x *dies* (and becomes *dead*) when its potential $p(x)$ is increased in the Dead-end Step.

(a) Path P never contains two consecutive jumping edges. This follows from rules (i)–(ii).

(b) At any time the sets $\{u\} \cup J(u)$, $u \in P$, are disjoint. In proof, if uv ranges over all jumping edges in P then the intervals $[\tau(v), \tau(u)]$ are disjoint (by (a) and the definition of τ). Rule (ii) shows $\tau(\{u\} \cup J(u)) \subseteq [\tau(v), \tau(u)]$. (b) follows.

(c) An eligible edge xy with y dead has x dead. To prove this statement suppose it holds at the start of a search. When a vertex dies no eligible edge is directed to it. Thus no dead vertex is added to P during this search. The Complete Step does not create an eligible jumping edge directed to a dead vertex. (The proof of Lemma 5.2 shows a swap uv that creates an eligible jumping edge xy has uy eligible. In the algorithm u is not dead, so y is not dead.) Thus the statement holds at the end of the search.

(d) A vertex x dies at most once. To prove this note that when $x \in \{u\} \cup J(u)$ dies it is not in any other such set (by (b)). Furthermore a dead vertex is never added to P, by (c).

(e) If x is in a set $J(u)$ and has not died, then no eligible residual edge is directed from x. This was true when x was added to $J(u)$ in the Dead-end Step, by the Scan Step. It remains true until $p(x)$ increases, i.e., x dies.

Now we prove the procedure is correct.

Lemma 5.4. The Augmenting Procedure maintains a 1-optimum flow with a topological numbering. It halts with any residual edge sa leading to a dead vertex a. It augments at least one path if on entry there is an st-path of eligible edges.

Proof. We prove the first part of the lemma by induction on the number of steps executed. There is nothing to prove for a Scan Step.

Consider a Dead-end Step. We can assume wu is a residual edge. Fact (e) shows no eligible residual edge is directed from any vertex of $J(u)$. The same holds for u. This implies no eligible jumping or residual edge leaves $\{u\} \cup J(u)$. Thus increasing $p(x)$, $x \in \{u\} \cup J(u)$ maintains (3) on edges directed from x. It maintains (3) on edges directed to x, and in fact no eligible edge now enters $\{u\} \cup J(u)$. So assigning the topological number T is valid. We conclude the Dead-end Step works as desired.

Consider a Complete Step. After τ is changed every jumping edge in P is strongly eligible. Thus P is a valid augmenting path if the new τ is a topological numbering.

The Complete Step does not increase any value $\tau(x)$ (see (b)). Thus we need only check τ on an eligible edge xy with $x \in \{u\} \cup J(u)$. Fact (e) shows that for $x \in J(u)$, xy is not residual, i.e., xy is an eligible jumping edge; this holds for $x = u$ too. Thus uy is an eligible jumping edge (or a self-loop). Rule (ii) implies that after τ is changed $\tau(y) \leq \tau(v) = \tau(x)$. So τ is valid.

Now Lemma 5.3 shows that augmenting gives a 1-optimum flow and a topological numbering.

For the second part of the lemma note that the Augmenting Procedure eventually halts, since a vertex added to P dies unless the flow gets augmented. The outer loop of the procedure ensures that when it halts only dead vertices are adjacent to s.

For the last part of the lemma, let $sa \ldots bt$ be a path of eligible edges on entry to the procedure. We claim that the flow gets augmented before a dies. If uv, $v \neq t$, is an eligible edge on entry to the procedure and at some point u is dead, then v is also dead (else uv would violate (3)). Thus if a is dead, so is b. But the first time b gets added to P the flow gets augmented, because of edge bt. ∎

Now we estimate the time for the Augmenting Procedure. Recall from Section 2 that the

oracle is used to compute jumping edges. The representation for the oracle is constructed after every augment. Section 6 gives the details of constructing the representation efficiently. The following analysis concentrates on calls to the oracle. Recall that calling the oracle for a given vertex v returns all jumping edges directed from v.

When the Augmenting Procedure adds a residual edge uv to P, the oracle is called for all jumping edges vx directed from v. Those edges vx that are eligible are placed in a list sorted by decreasing $\tau(x)$. Rule (ii) finds the next eligible jumping edge by scanning this list.

The following result accounts for all time spent by the Augmenting Procedure except the time to construct the oracle representation (this is discussed in Lemmas 6.2–3). Assume the Augmenting Procedure is always called with $T \leq n$ (this will be verified in the Voiding Procedure in Section 5.4). Furthermore assume an oracle call uses time $\Omega(n)$ (the time bound is $O(m)$ for connectivity orientation).

Lemma 5.5. In one scale the Augmenting Procedure uses time $O(\sigma m + \alpha n)$ plus the time for $O(\sigma n + \lambda)$ oracle calls.

Proof. To bound the number of oracle calls, suppose the oracle is called when a residual edge uv is added to P. Eventually either vertex v dies or an augmenting path containing edge uv is found. The former occurs at most σn times, since a vertex dies only once per search. The latter occurs $\leq \lambda$ times.

Next we show that the jumping edges vx directed from v can be sorted by $\tau(x)$ in time $O(n)$ (and so this time can be charged to the oracle call). The value T never exceeds $2n$ (since a vertex dies when T is incremented). Thus we can use a bucket sort with $2n$ buckets.

A residual edge uv that is scanned in rule (i) eventually becomes ineligible (either v dies or an augment reverses the edge and makes it ineligible). The time for rule (ii) amounts to $O(1)$ per jump stored in a list. Thus it is easy to see that the remaining time is $O(\sigma m + \alpha n)$. ∎

5.3. The Potential-changing Procedure

The *Potential-changing Procedure* is called with a 1-optimum flow and topological numbering. It changes potentials so there is an st-path of eligible edges, keeping the flow 1-optimum with a topological numbering.

The main data structure is a search tree $\mathcal{T}$ rooted at s. $\mathcal{T}$ consists of eligible edges directed away from s; any vertex except t can be in $\mathcal{T}$. On entry, and on exit, T equals the largest topological number. The Grow Step assigns new topological numbers in nondecreasing order. During the

procedure T is the value of the last topological number $\tau(v)$ that was assigned. Another variable T_0 records original values $\tau(v)$.

The procedure initializes $\mathcal{T}$ to the root s, sets T_0 to ∞ and increases T by n. Then it executes the following steps until the Finish Step returns.

Grow Step. (Note that if uv is a residual edge or an eligible edge, the condition uv leaves $\mathcal{T} \cup \{t\}$ is equivalent to uv leaves $\mathcal{T}$ and $v \neq t$.) Let uv be an eligible edge leaving $\mathcal{T} \cup \{t\}$, with largest value $\tau(v)$. Add uv to $\mathcal{T}$. If $\tau(v) < T_0$ then set $T_0 \leftarrow \tau(v)$, $T \leftarrow T - 1$. Set $\tau(v) \leftarrow T$. If an eligible edge leaves $\mathcal{T} \cup \{t\}$ then go to the Grow Step. If no residual or jumping edge leaves $\mathcal{T} \cup \{t\}$ then go to the Finish Step.

Change Step. Compute

$$\delta = \min\{p(v) - p(u) + 1 - d(uv) \mid uv \text{ a residual edge leaving } \mathcal{T} \cup \{t\}\}$$
$$\cup \{p(v) - p(u) \mid uv \text{ a jumping edge leaving } \mathcal{T} \cup \{t\}\}.$$

For each $v \in \mathcal{T} - \{s\}$ increase $p(v)$ by δ. Set $T_0 \leftarrow \infty$ and go to the Grow Step.

Finish Step. For each $v \in \mathcal{T} - \{s\}$ increase $p(v)$ by 1. Make all topological numbers between 1 and n by changing all occurrences of the ith largest value $\tau(v)$ to i. Set T to the largest topological number. Return. ∎

Note that the first execution of the Grow Step adds all vertices adjacent to s into $\mathcal{T}$ (since any edge sa is eligible). Although the Finish Step changes topological numbers, any two values $\tau(u)$ and $\tau(v)$ compare the same before and after the step ("compare the same" means any relation $\tau(u) = \tau(v)$, $\tau(u) < \tau(v)$, $\tau(u) > \tau(v)$ is unchanged). Now we prove the procedure is correct.

Lemma 5.6. The Potential-changing Procedure returns a 1-optimum flow with a topological numbering, having an st-path of eligible edges.

Proof. The argument has three parts. The first is to show that on exit the desired st-path of eligible edges exists, or equivalently, some eligible edge joins a vertex of $\mathcal{T}$ to t. The latter follows easily by noting the procedure (ignoring topological numbers) amounts to a search of Frank's algorithm [F82]. For completeness we give an argument based on first principles (also implicit in [F82]).

Let the submodular flow LP (1) refer to the linear program for the induced family. Use $\mathcal{F}$ and b to denote the set family and submodular function, respectively, of the induced problem. A set $S \in \mathcal{F}$ is i-tight iff equality holds in (1).

Any $S \in \mathcal{F}$ partitioned into disjoint sets S_i, $i = 1, \ldots, k$ has $\sum_i b(S_i) \geq -\delta_G(S)$. In proof, recall we assume that the given submodular flow problem is feasible. If y denotes a feasible flow (on G) then for each i, $b(S_i) \geq \rho_y(S_i) - \delta_y(S_i)$. Adding these inequalities and cancelling gives $\sum_i b(S_i) \geq \rho_y(S) - \delta_y(S) \geq -\delta_G(S)$.

Let S be the set of all vertices in $\mathcal{T} - \{s\}$ when the procedure returns. As usual x denotes the current submodular flow (on SFG). Since no jumping edge leaves S, S is the union of i-tight sets $P(v)$, $v \in S$. Thus S is the disjoint union of i-tight sets S_i. Similar to above this implies $\sum_i b(S_i) = \rho_x(S) - \delta_x(S)$. Since no residual edge leaves $S \cup \{s,t\}$ and some edges to s have flow, $\delta_x(S) > \delta_G(S)$. Thus $\rho_x(S) > \sum_i b(S_i) + \delta_G(S)$. Now the previous paragraph implies $\rho_x(S) > 0$. Since no residual edge leaves $S \cup \{s,t\}$ we conclude some residual edge goes from S to t, as desired. This completes the first part of the proof.

The second part, that the final flow is 1-optimum, is simple. Note that increasing the potentials by 1 in the Finish Step keeps the flow 1-optimum. This follows since no eligible edge leaves $\mathcal{T} \cup \{t\}$.

The last part of the proof is to show that the procedure ends with τ a valid topological numbering. Let τ_0 be the numbering on entry to the procedure. Call all vertices that enter $\mathcal{T}$ during the same execution of the Grow Step (i.e., without an intervening execution of the Change Step) a *growth*. Let uv be an edge that is eligible at the end of the procedure. We check that τ is valid on uv.

If neither u nor v enter $\mathcal{T}$ then uv is eligible before the procedure. Thus the validity of τ_0 implies that of τ. So assume at least one of u, v enters $\mathcal{T}$. Consider three cases. Suppose u enters $\mathcal{T}$ in a growth before v. (Thus v enters $\mathcal{T}$ after the growth for u or v never enters $\mathcal{T}$.) Since T is nonincreasing, it is easy to see this implies $\tau(u) > \tau(v)$. Thus τ is valid.

Suppose u and v enter $\mathcal{T}$ during the same growth. The validity of τ_0 implies that T_0 is nonincreasing during a growth. Thus $\tau(u)$ and $\tau(v)$ compare the same as $\tau_0(u)$ and $\tau_0(v)$. The supposition implies that edge uv is eligible even when the procedure starts. Thus the validity of τ_0 on uv implies that of τ.

Finally suppose that v enters $\mathcal{T}$ in a growth before u. Then $p(v)$ is increased by a positive quantity (by some $\delta > 0$ in the Change Step or by 1 in the Finish Step) before u enters $\mathcal{T}$. This makes edge uv ineligible. ∎

Now we give an efficient implementation of this procedure. Each vertex $v \notin \mathcal{T} \cup \{t\}$ maintains a value $\delta(v)$ equal to the minimum of the terms in the definition of δ corresponding to edges directed to v. When a vertex u enters $\mathcal{T}$, each residual edge uv with $v \notin \mathcal{T} \cup \{t\}$ is used to update $\delta(v)$. In

addition the oracle is called to find all jumping edges uv, and these edges are also used to update values $\delta(v)$.

As in Lemma 5.5 assume an oracle call uses time $\Omega(n)$.

Lemma 5.7. The Potential-changing Procedure uses time $O(m)$ plus the time for n oracle calls.

Proof. It is easy to see that one iteration uses time $O(n)$ to compute δ, update p and find the next edge to add to $\mathcal{T}$. The time spent scanning residual edges is $O(m)$, and the time for jumping edges amounts to n oracle calls. In the Finish Step the topological numbers are sorted using a bucket sort in time $O(n)$. ∎

5.4. The Voiding Procedure

Now we combine the two procedures for a search into the Voiding Procedure. It works by repeatedly finding an augmenting path and augmenting the flow, until all edges directed to s are void.

Recall from Section 2 the *Voiding Procedure* is called with a 1-optimum flow x, p on SFG, with no residual edge of G eligible. It must return a 1-optimum flow on SFG. The Voiding Procedure initializes the topological numbering τ to the zero function and T (the largest topological number) to 0. Then it repeats the following steps until the Done Step returns the desired flow.

Done Step. If no residual edge from s remains then return the 1-optimum flow x, p on G.

Adjust Step. Execute the Potential-changing Procedure. It adjusts potentials so there is an st-path of eligible edges, keeping the flow 1-optimum with topological numbering τ.

Augment Step. Execute the Augmenting Procedure. It repeatedly augments the flow, and returns a 1-optimum flow with topological numbering τ, and the potential of any vertex a still on a residual edge sa increased by 1. ∎

An iteration of the Voiding Procedure is called a *search* in previous sections. Lemmas 5.4 and 5.6 show the Augmenting and Potential-changing Procedures work as described. Lemma 5.4 implies that each search does at least one augment. Thus the Voiding Procedure eventually returns the desired 1-optimum flow on G.

Now we verify the properties of the Voiding Procedure assumed in Section 3. Each search increases the potential of a vertex in S by the same amount: The increase is 1 in the Augmenting Procedure (Lemma 5.4) and some nonnegative amount in the Potential-changing Procedure (as

noted after the procedure, each vertex of S enters $\mathcal{T}$ in the first execution of the Grow Step). No potential of a vertex $v \in T$ is changed: In the Augmenting Procedure v can die only after all residual edges vt are deleted. In the Potential-changing Procedure v doesn't enter $\mathcal{T}$ by definition. Finally each search increases Δ and decreases $|S|$, by Lemma 5.4.

6. Vertex-weighted cuts

This section discusses cuts of networks with edge capacities and vertex weights. Section 6.1 shows how to eliminate vertex weights when the total weight is 0. Section 6.2 applies this principle to maintaining the jumping edge oracle in the scaling algorithm. This completes the statement of the scaling algorithm.

6.1. The transformation

Consider a digraph G with nonnegative real-valued edge capacities. In addition each vertex v has a real-valued weight $w(v)$ (positive, negative or zero). The *(vertex-)weighted capacity* of a set $T \subseteq V$ is $\rho(T) + w(T)$ (in this section ρ with no subscript takes capacities into account). A *minimum weighted cut* is a set of vertices $T \neq \emptyset, V$ having minimum weighted capacity. In this section λ denotes this minimum weighted capacity.

The well-known algorithm of Picard [P,C] finds a minimum weighted cut when all capacities are infinite and more importantly, T can be any set including $\emptyset$ or V. This makes the problem easier. However this condition fails in the applications to submodular flow. We first sketch Picard's approach and indicate how it fails for our problem.

Let P (N) be the set of all vertices with positive (negative) weight. Construct FG (the "flow graph") by starting with the given graph G, adding a vertex s with edges sv, $v \in P$, of capacity $w(v)$, and a vertex t with edges vt, $v \in N$, of capacity $-w(v)$. An s,t-cut in FG is a set of vertices $T \cup \{t\}$, with (ordinary) capacity $\rho(T) + w(P \cap T) - w(N - T) = \rho(T) + w(T) - w(N)$.

In FG any $T \neq \emptyset, V$ has capacity $\geq \lambda - w(N)$; the capacity of $\emptyset$ is $-w(N)$ and that of V is $w(T) - w(N) = w(P)$. Picard's problem is solved by finding a minimum s,t-cut on FG. When $\lambda > 0$ (e.g., in submodular flow problems) this cut provides no useful information.

Our solution is as follows: Let f be a maximum flow from s to t in FG. Let EG (the "equivalent edge-capacitated graph") be the residual graph of f in G.

Theorem 6.1. Let G be a digraph with nonnegative edge capacities, arbitrary vertex weights, $w(V) = 0$ and $\lambda \geq 0$. For any $T \subseteq V$ the weighted capacity of T in G equals its ordinary capacity in EG.

Proof. In FG an s,t-mincut has value $w(P) = -w(N)$. Thus f saturates every edge incident to s or t. Let g be the flow f restricted to graph G. Flow conservation for f implies the net flow out of T is 0, i.e., $\delta_g(T) = \rho_g(T) + w(T)$. Let c be the capacity function on G. Then $\rho_{EG}(T) = \rho_{c-g}(T) + \delta_g(T) = \rho_c(T) + w(T)$ as desired. ∎

6.2. Maintaining the oracle

This section discusses how to efficiently maintain the oracle for jumping edges in our scaling algorithm.

Assume the given submodular function b is defined on all sets $S \neq \emptyset, V$ and satisfies $b(S) \leq \rho(S)$. This assumption is not unreasonable since the right-hand side is a trivial upper bound on the flow into S. (For instance in the orientation problem the assumption holds since it is the inequality $\rho(S) - k \leq \rho(S)$.)

Suppose we have an oracle that computes the jumping edges for any feasible flow on G. We can apply the oracle to compute the jumping edges for the current flow in the scaling algorithm even though this flow is on SFG. We proceed as follows.

For a given flow x on SFG define the weight of a vertex v as $x(\{vs\}) - x(\{tv\})$. Feasibility of x amounts to the condition that for any $S \neq \emptyset, V$, $\rho_x(S) - \delta_x(S) - w(S) \leq b(S)$. Let RG be the residual graph of x on graph G. The previous inequality is equivalent to $\rho(S) - b(S) \leq \rho_{RG}(S) + w(S)$. By assumption the left-hand side is nonnegative, so Theorem 6.1 shows there is an equivalent edge-capacitated graph EG. We use the oracle on EG.

Lemma 6.1. Consider a 0-1 submodular flow problem with $\rho(S) \geq b(S)$ for all $S \neq \emptyset, V$. An oracle that computes jumping edges for any feasible flow on G can be used to compute the jumping edges in the scaling algorithm by applying it to EG.

Proof. The jumping edges for EG are derived from the sets S where equality holds in the inequality $\rho(S) - b(S) \leq \rho_{EG}(S)$. These are the same sets that determine the jumping edges for x. ∎

As an example, in the orientation problem we can compute the jumping edges by applying the oracle for minimum cuts of [Ga93a] on graph EG. In general the hypothesis of the lemma can be enforced by increasing the multiplicity of each edge appropriately (as in [F84] assume an upper bound on b is known). In this case we assume the oracle can compute jumping edges for feasible flows on such enlarged graphs.

The rest of this section elaborates on how the lemma is applied in the scaling algorithm. The approach is presented for an arbitrary submodular flow problem, although it is designed to be efficient for the orientation problem. We break the task into two procedures, maintaining the equivalent graph EG and maintaining a graph OG for the jumping edge oracle (OG, defined below, is derived from EG). These procedures are executed after every augment (since an augment changes EG).

The equivalent graph EG is easily derived from a maximum flow in FG. Constructing this flow from scratch can be too time-consuming (this is the case in the orientation problem). Our approach is to maintain a maximum flow on FG. Note how the flow graph FG changes in an augment: FG is derived from the current submodular flow x, which defines the residual graph RG and the vertex weights (see the discussion preceding Lemma 6.1). Suppose the scaling algorithm augments along an augmenting path $AP = sv \ldots wt$ (in SFG). Let J be the set of all jumping edges of AP. The edges of $AP - J$ receive the opposite orientation in RG and FG. The weight of vertex v decreases by 1, so its capacity as a source in FG decreases by 1. Similarly for the sink w in FG.

Suppose we are given the flow graph FG for the current submodular flow, along with a maximum flow on FG. The flow maintenance algorithm works in two steps. The first step constructs the desired maximum flow on the new graph FG with edges J^R added. The second step voids the edges J^R. The details are as follows.

The first step traverses the augmenting path AP from v to w, maintaining the invariant that the current vertex x has a deficit of 1 unit of flow (i.e., 1 extra unit leaves x). Initially there is a deficit of 1 unit at v. In general if AP traverses an edge $xy \notin J$ then FG currently contains the edge xy. Update FG by replacing xy by yx; also push 1 unit of flow along yx (this may cancel a unit of flow in the deleted edge xy). If AP traverses $xy \in J$ then add yx to FG and pushing 1 unit of flow along it. Eventually we reach w and stop. The result is a maximum flow f on $FG + J^R$; here FG is the flow graph for the new submodular flow.

The second step of the flow maintenance algorithm cancels the flow in J^R as follows. Let R be the residual graph for $FG + J^R$ and flow f, with the edges J deleted (each jumping edge is an edge of the residual graph). Let S be the set of all heads of edges of J and T the set of all tails (where an edge is directed from tail to head). Let f^* be a maximum flow on FG. It is easy to check that $f^* - f$ is a flow from S to T in R of value $|S|$ (here each vertex of S has capacity 1 as a source and similarly for T).

The algorithm finds a maximum flow g from S to T in R. Then $f + g$ is the desired maximum

flow on FG.

For efficiency find the maximum flow g using the Ford-Fulkerson algorithm. Recall the parameter λ (Section 2).

Lemma 6.2. The equivalent graph EG can be maintained in total time $O(m\lambda)$. ∎

The second step of our procedure is to construct the representation of EG for the jumping edge oracle. We give an efficient representation construction algorithm. (This algorithm is needed in the orientation problem only for certain values of k. It is not needed for the dijoin problem or for most cases in the dense graph bound.)

For the orientation problem the oracle of [Ga93a] is based on the witness subgraph, i.e., a subgraph consisting of 2 complete k-intersections, partitioned into $2k$ spanning trees. In general assume that the oracle representation is based on a graph OG (the "oracle graph") constructed from EG. Assume further that the oracle graph has these properties: The residual graph of any feasible flow has a corresponding oracle graph; an oracle graph is the residual graph of a feasible flow; a graph is an oracle graph if it satisfies certain properties that do not depend on the rest of the residual graph. (These properties are obvious for the orientation problem. In the orientation problem OG must be maintained, along with its partition.)

The first step of the flow maintenance algorithm does not change EG (or OG). In proof let xy be an edge of $AP - J$. The augment places edge yx in RG. In FG yx gets 1 unit of flow. If xy originally had no flow then xy remains in EG. If xy originally had 1 unit of flow then yx remains in EG.

The algorithm for the second step of the maintenance algorithm uses the following operation. Consider a digraph $G = (V, E)$ and a (directed) cycle C consisting of edges that may or may not be in E. To *add-and-flip* C *into* G means to replace E by $E - C \cup C^R$. If G is the residual graph of a feasible flow then adding-and-flipping a cycle preserves this property. This follows since adding-and-flipping can be viewed as adding edges $C - E$ and then reversing C, and both these operations preserve (2).

Let OG be the oracle graph before the second step of the flow maintenance algorithm. The second step finds a maximum flow g from S to T in R. The new graph EG is derived from the previous one by reversing the edges of g. To update OG, make g a circulation by pushing 1 unit of flow along each edge of J. Add-and-flip (the cycles of) this circulation into OG, obtaining a new graph OG'. OG' is the residual graph of a feasible flow (and it contains the edges J^R). Find the oracle graph for OG'. (Do this from scratch, rather than updating the oracle graph.)

It remains to remove any edges of J^R from the new oracle graph. We give the details of this procedure in the context of the orientation problem. To remove an edge $j \in J^R$, delete it from OG, the partitioned oracle graph; find an augmenting path for the partitioned oracle graph, searching in the rest of EG.

The result is the desired oracle graph OG; furthermore OG is partitioned into spanning trees. The final step is to use the partitioned graph OG to build the jumping edge representation of [Ga93a].

Lemma 6.3. For the orientation problem, the representation of EG for the jumping edge oracle can be maintained in total time $O(\alpha k^2 n \log(n/k) + \lambda(m + kn \log(n/k)))$, assuming $\alpha \log n = O(\lambda)$.

Proof. A complete k-intersection on a graph of m edges is constructed in time $O(km \log(n^2/m))$ [Ga91]. If AP contains j jumping edges the number of edges in OG' is $O((k+j)n)$ (since the flow g has at most n edges for each jumping edge of AP). One augmenting path is found in time $O(m)$. These remarks imply the terms of the time bound. Finally note that the jumping edge representation can be constructed from the partitioned oracle graph OG in time $O(m \log(n^2/m))$ ([Ga93a, Theorem 8.3]). This contributes total time $O(\alpha m \log(n^2/m))$, which is $O(\lambda m)$ by assumption. ∎

7. The overall algorithm

This section reviews the entire scaling algorithm. Then it gives the final timing analysis for connectivity orientation.

The Main Routine (Section 2) scales the profits. It calls the Voiding Procedure (Section 5) to find a 1-optimum flow on G. This procedure consists of the Augmenting and Potential-changing Procedures (Section 5). Both of these make use of the oracle for jumping edges, which is maintained by the procedure of Section 6.2. The Main Routine also executes the potential compression procedure of Section 4 in the algorithm with compression.

Now we estimate the time for the entire algorithm. The bound for connectivity orientation is somewhat involved so we first note two special cases. If $m \geq (kn)^{4/3}$ then the time to find a minimum cost k-edge-connected orientation is

$$O((kn)^{2/3} nm \log(nN)).$$

If $k \geq n^{2/3}$ the time is

$$O(k^3 n^2 \log(n/k) \log(nN)).$$

This last bound seems to be the best achievable using our approach, since it equals the time to compute the oracle representation kn times on a graph of $O(kn)$ edges.

Theorem 7.1. A minimum-cost dijoin can be found in time

$$O(\min\{n^{2/3}, \sqrt{m}\}nm \log(nN)).$$

A minimum-cost k-edge-connected orientation can be found in time

$$O((\min\{(kn)^{2/3}nm,\ nm^{3/2} + m^2 \log n\} + k^3 n^2 \log(n/k)) \log(nN)).$$

Proof. First note that the orientation bound implies the dijoin bound. In fact for $k = O(1)$ the orientation bound simplifies to the dijoin bound. To see this note that if the second term of the minimization is smaller then $(kn)^{2/3}nm \geq nm^{3/2}$ so $(kn)^{2/3} \geq \sqrt{m}$. For $k = O(1)$ this implies $m = O(n^{4/3})$. In this case the second term simplifies to $nm^{3/2}$.

Next note that for connectivity-orientation the Main Routine is called with a k-edge-connected orientation. This orientation is found using the procedure of Section 8. The time for this, given in Theorem 8.2, is easily seen to be dominated by our desired bound.

To prove the connectivity-orientation bound we estimate the time for one scale. We first summarize the contributions to the time. Corollary 3.2 shows the parameter values for the algorithm (with or without compression) are

$$\alpha = O(m),\ \sigma = O(\sqrt{m}),\ \lambda = O(m \log m).$$

Corollary 3.3 shows the parameter values for the algorithm with compression are

$$\alpha = O(kn),\ \sigma = O((kn)^{2/3}),\ \lambda = O(nk^{2/3}m^{1/3}).$$

An oracle call for connectivity orientation uses time $O(m)$ [Ga93a]. Thus Lemmas 5.5 and 5.7 (with the trivial observation $\alpha n \leq \lambda m$) show the Augmenting and Potential-changing Procedures use time

$$O(\sigma nm + \lambda m).$$

Lemmas 6.2–3 show the time to construct all oracle representations is

$$O(\alpha k^2 n \log(n/k) + \lambda(m + kn \log(n/k))).$$

Note that the hypothesis of Lemma 6.3 holds, i.e., the parameter values above satisfy $\alpha \log n = O(\lambda)$ (for the second set of values, $m \geq kn$ implies $\lambda = nk^{2/3}m^{1/3} \geq kn^{4/3} > kn \log n = \alpha \log n$). Theorem 4.1 shows potential compression uses time

$$O(\sqrt{km}\, n^2 \log n).$$

In what follows, for convenience we use some inequalities that hold for sufficiently large n, e.g., $n^{1/3} \geq \log n$.

We first show that when compression is used the time is $O((kn)^{2/3}nm + k^3n^2\log(n/k))$. Using the parameter values for the algorithm with compression this bound corresponds to $O(\sigma nm + \alpha k^2 n\log(n/k))$. It suffices to check that the term σnm dominates the remaining terms given above for the time. We compute the ratio between σnm and each of these terms, making use of the relation $m \geq kn$:

$$\frac{\sigma nm}{\lambda m} = \left(\frac{n^2}{m}\right)^{1/3},$$

$$\frac{\sigma nm}{\lambda nk\log(n/k)} = \frac{m^{2/3}}{n^{1/3}k\log(n/k)} \geq \frac{(n/k)^{1/3}}{\log(n/k)},$$

$$\frac{\sigma nm}{\sqrt{km}n^2\log n} = \frac{k^{2/3}\sqrt{m}}{\sqrt{k}n^{1/3}\log n} \geq \frac{k^{2/3}\sqrt{n}}{n^{1/3}\log n}.$$

Since the last term on each line is at least 1, all ratios are at least 1, and the numerator σnm dominates the denominator as desired.

Next we show the time bound $O(nm^{3/2} + m^2\log n + k^3n^2\log(n/k))$. Together with the first bound this implies the time bound of the theorem. To achieve this bound we use compression if $m < k^4\log^2(n/k)$.

The analysis uses the first set of parameter values; in addition let $\alpha' = kn$. The desired bound corresponds to $O(\sigma nm + \lambda m + \alpha' k^2 n\log(n/k))$. We first claim the last term in the time for oracle representations, $\lambda nk\log(n/k)$, is dominated by our time bound. In proof note these ratios:

$$\frac{\sigma nm}{\lambda nk\log(n/k)} = \frac{\sqrt{m}}{k\log m\log(n/k)},$$

$$\frac{\alpha' k^2 n\log(n/k)}{\lambda nk\log(n/k)} = \frac{k^2 n}{m\log m}.$$

If $m \geq k^2\log^4 n$ then the first ratio is at least one; if $m < k^2\log^4 n$ then $m\log m < k^2\log^5 n < k^2 n$ so the second ratio is at least one.

Next we justify the term $\alpha' k^2 n\log(n/k)$ in the desired bound. If compression is used then this term corresponds to the first term in the time for oracle representations. If compression is not used then $\alpha k^2 n\log(n/k)$ is the first term in the time for oracle representations, but it is dominated by our time bound since $m \geq k^4\log^2(n/k)$ implies

$$\frac{\sigma nm}{\alpha k^2 n\log(n/k)} = \frac{\sqrt{m}}{k^2\log(n/k)} \geq 1.$$

Last we show that the time for compression is dominated by our time bound. If compression is used then $k^{5/2}\log^{5/4}(n/k) > m^{5/8}$, so

$$\frac{\alpha' k^2 n \log(n/k)}{\sqrt{kmn^2}\log n} = \frac{k^{5/2}\log(n/k)}{\sqrt{m}\log n} > \frac{m^{1/8}}{\log^{1/4}(n/k)\log n}\ .$$

Thus the ratio is at least 1 as desired. ∎

We conclude by proving the claim before the theorem, that the time is $O((kn)^{2/3}nm\log(nN))$ if $m \geq (kn)^{4/3}$. The inequality implies that the minimum in the time bound equals the first term. Furthermore the last term of the time bound, $k^3n^2\log(n/k)$, is dominated by the first term, since the inequality implies

$$\frac{(kn)^{2/3}nm}{k^3n^2\log(n/k)} = \frac{m}{(kn)^{1/3}k^2\log(n/k)} \geq \frac{n/k}{\log(n/k)}\ .$$

8. Feasibility

This section presents efficient algorithms for finding a feasible submodular flow. First it summarizes Frank's two-step feasibility algorithm [F84]. Section 8.1 discusses the second step, reducing it to ordinary network flow for many general submodular flow problems. Section 8.2 implements the first step efficiently using our scaling algorithm. It also gives our feasibility result for connectivity orientation.

Frank's algorithm finds a feasible solution to a general submodular flow problem (1) as follows [F84]. Choose any function $x : E \to \mathbf{R}$ satisfying $\ell \leq x \leq u$.

Step 1. Find a vertex weight function $w : V \to \mathbf{R}$ such that $w(V) = 0$ and any $S \in \mathcal{F}$ satisfies $\rho_x(S) - \delta_x(S) - w(S) \leq b(S)$.

Step 2. Reduce w to the zero function, preserving the last inequality and the bounds on x. ∎

The result is a feasible flow x. If b and the initial x are integral then w and the final x are integral.

8.1. Finding a flow

Step 2 is a submodular flow problem in general. However in many cases it simplifies to an ordinary network flow problem, as we now show.

We discuss feasibility for the general submodular flow problem (1) in this special case: Assume the given submodular function b is defined on all sets $S \subseteq V$ except possibly $\emptyset, V$ and satisfies

$b(S) \leq \rho_u(S) - \delta_\ell(S)$. As in Section 6.2 this assumption is not unreasonable since the right-hand side is a trivial upper bound on the flow into S.

Suppose we are given an initial x and a vertex weight function w as in Step 1. To find a feasible submodular flow, define the function $y = x - \ell$. Let RG be the residual graph of y on graph G with capacity function $u - \ell$. The inequality of Step 1 is equivalent to $\rho_u(S) - \delta_\ell(S) - b(S) \leq \rho_{u-\ell-y}(S) + \delta_y(S) + w(S)$. By assumption this holds for all $S \neq \emptyset, V$ and the left-hand side is nonnegative. The right-hand side is the weighted capacity of S in RG. Thus Theorem 6.1 applies. Let f be the maximum flow on FG. Form function g by starting with $y + f$ and cancelling flow in any edge and its reverse.

Theorem 8.1. Consider a submodular flow problem with $\rho_u(S) - \delta_\ell(S) \geq b(S)$ for all $S \neq \emptyset, V$. The function $\ell + g$ is a feasible submodular flow constructed in one maximum flow computation from x and w.

Proof. Let c be the capacity function in RG. The definition of the equivalent graph EG shows $\rho_u(S) - \delta_\ell(S) - b(S) \leq \rho_{c-f}(S) + \delta_f(S)$, or by rearranging, $\rho_u(S) - \rho_{c-f}(S) - \delta_{\ell+f}(S) \leq b(S)$. Since $\rho_{c-f}(S) = \rho_{u-\ell-y}(S) + \delta_y(S) - \rho_f(S)$ the rearranged inequality simplifies to $\rho_{x+f}(S) - \delta_{x+f}(S) \leq b(S)$. We can replace $x + f$ by $\ell + g$ in this last inequality, giving the first desired inequality.

We must check that $\ell + g$ obeys the upper bounds (it clearly obeys the lower bounds). For any e, $f(e) \leq (u - \ell - y)(e) + y(e^R)$. If $f(e) + y(e) \geq y(e^R)$ cancellation implies the desired upper bound. If $f(e) + y(e) < y(e^R)$ cancellation voids e, again implying the desired upper bound. ∎

8.2. Finding vertex weights

Frank implements Step 1 using his discrete separation algorithm [F84]. We present an efficient implementation of this approach based on our scaling algorithm. The discussion is for 0-1 flow.

We begin with a version of Frank's construction, specialized for the feasibility problem and our scaling algorithm. We choose the initial function x as the zero function. Thus the goal is to find a vertex weight function $w : V \to \mathbf{Z}$ such that $w(V) = 0$ and any $S \in \mathcal{F}$ satisfies $w(S) \leq b(S)$.

Call such a vertex weight function *permissible* if every $v \in V$ has $-\delta_G(v) \leq w(v) \leq \rho_G(v)$. A feasible 0-1 problem has a permissible weight function. Specifically if x is feasible define $w(v) = \rho_x(v) - \delta_x(v)$. Obviously $w(v)$ is in the desired range. Furthermore any $S \in \mathcal{F}$ has $w(S) = \rho_x(S) - \delta_x(S) \leq b(S)$.

We will find a permissible weight function by solving a submodular flow problem. Start by choosing an arbitrary vertex r. Define two intersecting families on the ground set $V - r$,

$\mathcal{F}^+ = \{S \mid S \in \mathcal{F}, r \notin S\}$ and $\mathcal{F}^- = \{\overline{S} \mid S \in \mathcal{F}, r \in S\}$. Define two functions that are submodular on intersecting pairs of these families, by $b^+(S) = b(S)$ for $S \in \mathcal{F}^+$ and $b^-(S) = b(\overline{S})$ for $S \in \mathcal{F}^-$. (Throughout this section $\overline{S}$ denotes $V - S$ for $S \subseteq V$.)

Now we define a 0-1 submodular flow problem on a graph BG. Let $V^- = \{v^- \mid v \in V\}$ and similarly for V^+. If $S \subseteq V$ the notation S^- denotes the corresponding subset of V^-, and similarly for S^+. BG has vertex set $V^- \cup V^+$, plus $\rho_G(v)$ copies of edge v^-v^+ and $\delta_G(v)$ copies of v^+v^-. Define an intersecting family $\mathcal{F}'$ on the vertices of BG by using the family $\mathcal{F}^+$ on V^+ and $\mathcal{F}^-$ on V^-. Define a function b' submodular on intersecting pairs of this family, by using b^+ on $\mathcal{F}^+$ and b^- on $\mathcal{F}^-$.

A permissible weight function gives a feasible flow on BG. In proof make $\max\{w(v), 0\}$ edges v^-v^+ and $\max\{-w(v), 0\}$ edges v^+v^- into 1-edges, and make all other edges into 0-edges. Inequality (1) for a subset of V^- follows using $w(\overline{S}) = -w(S)$. Conversely a feasible flow x on BG gives a permissible weight function, by setting $w(v) = \rho_x(v^+) - \delta_x(v^+)$ for $v \neq r$ and $w(r) = -w(V - r)$.

Thus our problem is to find a feasible flow on BG. The *Vertex-weight Procedure* does this by first enlarging BG to the graph SFG by adding vertices s and t, plus $2m$ copies of edge tr^-, plus for each $v \in V$, $\rho_G(v) + \delta_G(v)$ copies of v^+s. Define a flow on SFG by making all edges from V^+ to V^- 0-edges and all other edges 1-edges. Make this a 1-optimum flow on SFG with every edge of G satisfied, by defining the profit function d to be identically 0 and all potentials to be 0. Then find the desired flow on BG by executing the Voiding Procedure.

To prove the Vertex-weight Procedure correct we need only show that the initially constructed flow is feasible. Assume that the original submodular flow problem is feasible (infeasible problems are discussed after Lemma 8.1). Thus any $S \in \mathcal{F}$ has $-\delta_G(S) \leq b(S)$. Thus any $S^+ \in \mathcal{F}^+$ has $-\sum\{\delta_G(v) \mid v \in S\} \leq -\delta_G(S) \leq b(S) = b'(S^+)$. Similarly any $S^- \in \mathcal{F}^-$ has $-\sum\{\rho_G(v) \mid v \in S\} \leq -\rho_G(S) = -\delta_G(\overline{S}) \leq b(\overline{S}) = b'(S^-)$.

We prove an analog of Corollary 3.2 for the efficiency of this procedure. Note graphs BG and SFG have $O(m)$ edges.

Lemma 8.1. The Vertex-weight Procedure finds a permissible vertex weight function for a feasible flow problem. It achieves $\alpha \leq m$, $\sigma \leq c\sqrt{m}$, $\lambda = O(m \log m)$.

Proof. The analysis is similar to the analysis of the Voiding Procedure in Section 3. The notation x, p, d is the same as in Section 3; x_- denotes any feasible flow on BG (instead of the flow at the end of the previous scale) and p_-, x_0, p_0 are not used.

The analog of Lemma 3.2 states that $|S|\Delta \leq cm$. In proof, Lemma 3.1 implies $dx_-(E) \leq dx(E) - p(S) + p(T) + v(x_- \oplus x)$. (The proof is the same as the proof of inequality $(4a)$ in Section 3.) Since d is zero this simplifies to $|S|\Delta = p(S) \leq v(x_- \oplus x) \leq cm$.

The analog of Lemma 3.3 is $\kappa_j = \sum_{i=1}^{j} \Delta_i$. The proof is a simplification of Section 3: The profit-length of a residual edge is -1, so the profit-length of an augmenting path is the negative of the number of residual edges. As in Section 3 an augmenting path P found when the dual adjustment is Δ has profit-length $-\Delta$.

Now the desired parameter values follow as in Corollary 3.2. ∎

We briefly mention infeasible problems. (For connectivity orientation this general discussion is irrelevant: Infeasibility is detected before we execute our procedure, as indicated at the end of this section.) The initialization of our procedure requires that each $S \in \mathcal{F}$ has $-\delta_G(S) \leq b(S)$. This condition is necessary but not sufficient for feasibility. We assume it can be checked by an oracle. If the condition holds but the problem is infeasible, the Voiding Procedure detects infeasibility and can output a witness (see Lemma 5.6, or for more detail [F84]).

The Voiding Procedure is exactly as specified in Section 5. It remains only to modify the procedure of Section 6.2 for the jumping edge oracle. As in Lemma 6.1 assume $\rho_G(S) \geq b(S)$ for all $S \neq \emptyset, V$. Let x be the current flow on SFG.

We maintain two oracles, one for vertices in V^+ and the other for V^-. The oracle for V^+ gives jumping edges with both ends in V^+. This is sufficient, since a vertex $v \in V^+$ has a jumping edge to a vertex of V^- iff v is in no c-tight set other than $V^- \cup V^+$ iff v has a jumping edge to every vertex of V^- and V^+. Similar remarks hold for V^-.

We first prove an analog of Lemma 6.1, the jumping edges can be computed from an "equivalent graph". Consider V^+. The jumping edges directed from vertices in V^+ are derived from the sets $S^+ \in \mathcal{F}^+$ where equality holds in the inequality $\sum\{\rho_x(v) - \delta_x(v) \mid v \in S^+\} \leq b'(S^+)$ (this inequality is equivalent to (1) for SFG). For convenience identify V^+ with V. Define a weight function on V by $w(v) = \delta_x(v) - \rho_x(v)$ for $v \neq r$ and $w(r) = -w(V - r)$. The above inequalities are equivalent to $\rho_G(S) - b(S) \leq \rho_G(S) + w(S)$ for any set S not containing r. For each vertex v add $\delta_{SFG}(v) + \rho_G(v) + 1$ copies of edge vr. We will show that any set S containing r has

$$\rho(S) + w(S) > \rho_G(S) - b(S).$$

We have assumed that the right-hand side is nonnegative, so this ensures the hypotheses of Theorem 6.1 hold. Thus there is an equivalent edge-capacitated graph EG for V^+.

To prove the above inequality, $\rho(S) + w(S) > \rho_G(S) + \sum\{\delta_{SFG}(v) + \rho_G(v) \mid v \notin S\} + w(S) =$

$(\rho_G(S)+\sum\{\rho_G(v) \mid v \notin S\})+(\sum\{\delta_{SFG}(v) \mid v \notin S\}+w(S))$. The second term is nonnegative since any $v \in V$ with positive weight contributes $\geq w(v)$ to it (if $v \notin S$ then v contributes $\delta_{SFG}(v) \geq w(v)$). For the first term note that feasibility implies $\sum\{\rho_G(v) \mid v \notin S\} \geq \delta_G(S) \geq -b(S)$, so the first term is at least $\rho_G(S) - b(S)$ as desired.

We treat V^- similarly as follows. Define a weight function by $w(v) = \rho_x(v) - \delta_x(v)$ for $v \neq r$ and $w(r) = -w(V - r)$. Inequalities (1) amount to $\rho_G(S) - b(S) \leq \rho_G(S) + w(S)$ for any set S containing r. For each vertex v add $\delta_{SFG}(v) + \delta_G(v) + 1$ copies of edge rv. As above it suffices to show that any set S not containing r has

$$\rho(S) + w(S) > \rho_G(S) - b(S).$$

In proof, $\rho(S) + w(S) > \rho_G(S) + \sum\{\delta_{SFG}(v) + \delta_G(v) \mid v \in S\} + w(S) = (\rho_G(S) + \sum\{\delta_G(v) \mid v \in S\}) + (\sum\{\delta_{SFG}(v) \mid v \in S\} + w(S))$. The second term is nonnegative since any $v \in S$ has $\delta_{SFG}(v) + w(v) \geq 0$. Feasibility implies $\sum\{\delta_G(v) \mid v \in S\} \geq \delta_G(S) \geq -b(S)$ so the first term is at least $\rho_G(S) - b(S)$ as desired.

We maintain the equivalent graphs EG similar to Section 6 by maintaining the flow graph FG and a maximum flow on it, as follows. Let AP be an augmenting path in SFG. Observe that AP contains at most one jumping edge from one of the sets V^+, V^- to the other. Specifically if such an edge exists we can jump to r^- and complete the augmenting path.

Consider V^+. Suppose AP contains a jumping edge uv with $u, v \in V^+$. The augment decreases $w(u) = \delta_x(u) - \rho_x(u)$ by 1; similarly $w(v)$ increases by 1. We update FG (adding an artificial edge) as follows. First adjust the capacity and flow on edges incident to s or t in FG (e.g., if $w(u)$ is positive before the augment, decrease the capacity and flow for su by 1). Then add edge vu to FG and push 1 unit of flow on it. This gives the desired flow on FG since 1 more unit of flow must enter u (regardless of the sign of $w(u)$) and 1 more unit must leave v.

A vertex $u \in V^+$ can occur in AP in two other ways. AP can begin with residual edges su, uv for $v \in V^-$. The occurrence in the augment has no effect on $w(u)$ or FG. AP can end with residual edge vu ($v \in V^- \cup \{s\}$) and jumping edge ur^-. This decreases $w(u)$ by 1 and increases $w(r^+)$ by 1. We process this similar to the previous case.

Now proceed as in Section 6.2: find a maximum flow on the residual graph of the current flow, with jumping edges deleted; add this flow to the current flow to get the desired flow on FG.

V^- is handled similarly. Suppose AP contains a jumping edge uv with $u, v \in V^-$. The augment increases $w(u) = \rho_x(u) - \delta_x(u)$ by 1; similarly $w(v)$ decreases by 1 (even if $v = r^-$). As above adjust the capacity and flow on edges incident to s or t in FG, add edge uv and push 1 unit of flow on it. After all jumping edges are processed, find a maximum flow on the residual graph of

the current flow, with jumping edges deleted; add this flow to the current flow to get the desired flow on FG.

The last two displayed inequalities show that in V^+ (V^-) no set (not) containing r is c-tight. So the jumping edges in EG are the desired jumping edges for V^+ (V^-). We apply the oracle to compute these jumping edges. (Unlike Lemma 6.1 EG need not be the residual graph of a flow.)

After the equivalent graph EG is updated we compute the new oracle graph OG. Although the procedure of Section 6 can be used, for connectivity orientation it is just as efficient to simply construct OG from EG starting from scratch.

This completes the description of the feasibility algorithm. Now we apply it to connectivity orientation. We increase the efficiency using a special property of this problem: The witness graph of a feasible flow can be found efficiently. Specifically Nash-Williams' theorem [NW] states that an undirected graph has a k-edge-connected orientation iff it is $2k$-edge-connected (as an undirected graph). Given an undirected graph G, we begin by finding a subgraph of $\leq 4kn$ edges that is $2k$-edge connected, using the procedure of [Ga91]. (This procedure detects an infeasible problem, i.e., connectivity less than $2k$.) We execute our feasibility algorithm on this subgraph.

Theorem 8.2. A k-edge-connected orientation of an undirected graph can be found, if one exists, in time $O(kn^2(\sqrt{kn} + k^2 \log(n/k)))$.

Proof. A $2k$-edge-connected subgraph is found in time $O(m + k^2 n \log(n/k))$ [Ga91]. The time for the oracle is $O(\alpha k^2 n \log(n/k) + \lambda kn)$. This follows from Lemmas 6.2–3 and the fact that FG contains $O(kn)$ edges (see the definition above). Now the estimate of Theorem 7.1 shows the time for the feasibility algorithm satisfies the bound of the theorem. The last step is to find the maximum network flow of Theorem 8.1 corresponding to the vertex weight function. Since the total of all positive weights is $O(kn)$, the Ford-Fulkerson algorithm finds the desired flow in time $O((kn)^2)$. ∎

References

[C] V. Chvátal, *Linear Programming*, W.H. Freeman and Co., New York, 1983.

[CF] W.H. Cunningham and A. Frank, "A primal-dual algorithm for submodular flows", *Math. of Op. Res., 10,* 2, 1985, pp. 251–262.

[CW] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *J. Symbolic Comp. 9*, 1990, pp. 251–280.

[E] J. Edmonds, "Edge-disjoint branchings", in *Combinatorial Algorithms*, R. Rustin, Ed., Algorithmics Press, New York, 1972, pp. 91–96.

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

[F81] A. Frank, "How to make a digraph strongly connected", *Combinatorica 1*, 2, 1981, pp. 145–153.

[F82] A. Frank, "An algorithm for submodular functions on graphs", in *Bonn Workshop on Combinatorial Optimization*, A Bachem, M. Grötschel and B. Korte, Eds., Ann. Discrete Math., 16, North-Holland, Amsterdam, 1982, pp. 97–120.

[F84] A. Frank, "Finding feasible vectors of Edmonds-Giles polyhedra", *J. Comb. Th., B, 36*, 1984, pp. 221–239.

[FT] A. Frank and É. Tardos, "Generalized polymatroids and submodular flows", *Math. Programming, 42*, 1988, pp. 489–563.

[Fu] S. Fujishige, *Submodular Functions and Optimization*, North-Holland, New York, 1990.

[Ga85] H.N. Gabow, "Scaling algorithms for network problems", *J. Comp. and System Sci., 31,* 2, 1985, pp. 148–168.

[Ga91] H.N. Gabow, "A matroid approach to finding edge connectivity and packing arborescences", *Proc. 23rd Annual ACM Symp. on Theory of Comp.,* 1991, pp. 112–122.

[Ga93a] H.N. Gabow, "A representation for crossing set families with applications to submodular flow problems", *Proc. 4th Annual ACM-SIAM Symp. on Disc. Algorithms,* 1993, pp. 202–211; "Centroids, representations and submodular flows", submitted for publication.

[Ga93b] H.N. Gabow, "Efficient algorithms for large submodular flow problems", in preparation.

[GaT88] H.N. Gabow and R.E. Tarjan, "Almost-optimum speed-ups of algorithms for bipartite matching and related problems", *Proc. 20th Annual ACM Symp. on Theory of Comp.,*

1988, pp. 514–527.

[GaT89] H.N. Gabow and R.E. Tarjan, "Faster scaling algorithms for network problems", *SIAM J. Comput., 18,* 5, 1989, pp. 1013–1036.

[GoT] A.V. Goldberg and R.E. Tarjan, "Finding minimum-cost circulations by successive approximation", *Math. of Op. Res., 15,* 3, 1990, pp. 430–466.

[GLS] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, New York, 1988.

[GX] H.N. Gabow and Y. Xu, "Efficient algorithms for independent assignment on graphic and linear matroids", *Proc. 30th Annual Symp. on Found. of Comp. Sci.,* 1989, pp. 106–111.

[HK] J. Hopcroft and R. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", *SIAM J. Comput., 2,* 4, 1973, pp. 225–231.

[K] A.V. Karzanov, "On the minimal number of arcs of a digraph meeting all its directed cutsets", abstract, *Graph Theory Newsletters, 8*, March 1979.

[L] C.L. Lucchesi, "A minimax equality for directed graphs", Ph. D. Dissertation, University of Waterloo, Waterloo, Ontario, 1976.

[NW] C. St. J. A. Nash-Williams, "Well-balanced orientations of finite graphs and unobtrusive odd-vertex-pairings", in *Recent Progress in Combinatorics*, W.T. Tutte, Ed., Academic Press, New York, 1969, pp. 133–149.

[P] J.-C. Picard, "Maximal closure of a graph and applications to combinatorial problems", *Management Sci. 22*, 11, 1976, pp. 1268–1272.

[S] A. Schrijver, "Min-max results in combinatorial optimization", in *Mathematical Programming, the State of the Art: Bonn, 1982*, A. Bachem, M. Grötschel and B. Korte, Eds., Springer-Verlag, Berlin, 1983, pp. 439–500.