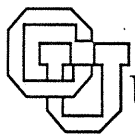
ALGORITHMS FOR GRAPHIC POLYMATROIDS AND PARAMETRIC S-SETS

Harold N. Gabow

CU-CS-736-94



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Algorithms for Graphic Polymatroids and Parametric s-Sets

Harold N. Gabow

CU-CS-736-94 1994

Department of Computer Science
University of Colorado at Boulder
Campus Box 430
Boulder, Colorado 80309-0430 USA

Algorithms for Graphic Polymatroids and Parametric s-Sets

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

August 10, 1994

Abstract. We present efficient algorithms for covering, finding a base and packing in graphic polymatroids. The integral covering number is the arboricity of an undirected graph; computing it is suggested as an open problem by Gallo et. al. [GGT]. We compute the arboricity in time $O(nm \log(n^2/m))$, the same bound as the other parametric flow algorithms of [GGT] (n and m denote the number of vertices and edges of the given graph respectively). Finding a minimum-cost base solves problems like optimal reinforcement of a network. We find a base in time $O(n^2 m \log(n^2/m))$, improving the previous bound of m maximum flow computations. The fractional packing number is known as the strength of a network. We compute it in time $O(n^2 m \log(n^2/m))$ and space $O(m)$, improving the best previous result by a factor n in space. Our algorithms are based on a new characterization of the vectors in a graphic polymatroid, and also an extension of parametric flow techniques to a problem concerning global minimum cuts, parametric augmentation for s-sets.

---

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

1. Introduction

Graph problems such as packing spanning trees, computing the strength, and covering by forests are most fruitfully studied using graphic matroids and polymatroids. Previous approaches to such problems are based on characterizations of the graphic polymatroid using network flow, equivalently, s,t-cuts. Recent algorithms for parametric network flow have proved useful for this approach. We give a new characterization of graphic polymatroids in terms of s-cuts (synonymously, s-sets) rather than s,t-cuts. We propose a new parametric framework, called parametric augmentation for s-sets, to handle the problems arising from our characterization. As a result we obtain more efficient algorithms for problems about graphic polymatroids, specifically, covering, finding a base and packing.

We now state our results precisely and compare them to previous work. In the time and space bounds throughout this paper, n and m denote the number of vertices and edges of the given graph, respectively. In this section T_{NF} denotes the time to find a maximum flow on a network of n vertices, m edges and arbitrary capacities. Phillips and Westbrook extend a long chain of network flow algorithms and show that for any fixed $\epsilon > 0$, $T_{NF} = O(\min\{nm \log_{m/n} n + n^2 \log^{2+\epsilon} n, nm \log n\})$ [PhW].

Our covering result is to compute the arboricity of an undirected graph with integral edge capacities. The arboricity is the smallest number of forests that contain all the edges (the capacity of an edge is the number of times it must be covered). Computing the arboricity for a graph with arbitrary edge capacities is suggested as an open problem by Gallo et. al. [GGT]. We compute the arboricity in time $O(nm \log(n^2/m))$, the same time as the other parametric flow algorithms of [GGT]. For graphs with integral capacities that are $O(1)$ our algorithm uses time $O(m^{3/2} \log(n^2/m))$. This improves the bound of [GW], which is $O(\min\{m^{5/3} \log^{2/3} n, nm \log n\})$ for $m = \Omega(n \log n)$ and slightly more otherwise. The algorithm of [GW] computes the corresponding forests that cover the graph, in the case of $O(1)$ capacities. Our algorithm for $O(1)$ capacities does this also. Earlier algorithms for arboricity include [PQ82a] for $O(1)$ capacities and [PaW84] for general capacities.

Our second result is to find a minimum-cost base of a graphic polymatroid in time $O(n^2 m \log(n^2/m))$. This improves the algorithm of [C85a] which uses time $O(mT_{NF})$. A minimum-cost base algorithm solves other problems in the same time, specifically the problem of optimal reinforcement of a network [C85a] and optimal augmentation for covering (defined in Section 6).

Another application of this algorithm is our packing result, which is to compute the packing number of an undirected graph with edge capacities. The integral packing number is the greatest

number of spanning trees in the graph (an edge can occur in as many trees as its capacity). The fractional packing number is called the strength of a graph, a measure of its vulnerability [C85a, Gu83]. We compute these packing numbers in $O(n^2 m \log(n^2/m))$ time and $O(m)$ space. This is the same time bound as the recent algorithm of Cheng and Cunningham and it improves their space bound of $O(nm)$. Previous algorithms for strength include Cunningham's original algorithm [C85a] using $O(nmT_{NF})$ time and $O(m)$ space, and Gusfield's algorithm using $O(nm^2 \log(n^2/m))$ time and $O(m^2)$ space [Gu91].

Note that for large capacities our algorithms do not find the trees for covering or packing. Our approach indicates a way to find these trees as a set of arborescences. This is investigated further in [GM]. However the algorithms are slower than those of this paper (e.g., the strong polynomial time bound is a factor n more than our packing bound). The algorithms of this paper do find the set of edges composing the trees.

We turn to the methodology used for these results. At the highest level our algorithms are implementations of Newton's method for fractional optimization. This method is used for graphic matroids and polymatroids in [C85a, PQ82a, PaW84]. We show that for arbitrary polymatroids the packing and covering problems can be solved efficiently by Newton's method, assuming an oracle to find a base. We also propose the prefix algorithm, an algorithm for finding a polymatroid base which is well-suited for implementing Newton's method for packing.

We propose a new characterization of vectors in graphic polymatroids. Previous work [C85b, PQ82b, PaW83] tests a vector for membership in a graphic polymatroid (equivalently, the forest polytope) by solving n network flow problems. The algorithm of [CC] is based on a characterization of Barahona [B] that tests a vector for membership in the dominant of the spanning tree polytope, also in n network flow computations. Our characterization tests a vector for membership in a graphic polymatroid by solving one network flow problem (i.e., a minimum s,t-cut problem) and one minimum t-cut problem.

This characterization leads to different parametric problems in analyzing graphic polymatroids. We formulate a parametric problem concerning the global minimum cut, which we call parametric augmentation for s-sets. We extend the global minimum cut algorithms of Hao and Orlin [HO] and Gabow [Ga91a] to solve this problem efficiently on graphs with both large and small capacities. We believe parametric augmentation will find other applications in future work.

The paper is organized as follows. Section 2 discusses Newton's method for covering and packing, and formulates the prefix algorithm. Section 3 shows how to transform membership questions for graphic polymatroids into s,t-cut and s-cut problems. Section 4 discusses the parametric aug-

mentation problem for s-sets. Section 5 gives our algorithms for arboricity. Section 6 implements the prefix algorithm on graphic polymatroids. Section 7 applies the prefix algorithm for packing graphic polymatroids. The rest of this section gives notation, definitions and some background on polymatroids.

$\mathbf{R}$ denotes the set of real numbers, $\mathbf{R}_+$ the set of nonnegative real numbers. For integers i, j, $[i..j]$ denotes the set of integers k, $i \leq k \leq j$.

Consider a universe V containing elements s, t and subsets S, T. $\overline{S}$ denotes the complement $V - S$. We use both set containment $S \subseteq T$ and proper set containment $S \subset T$. We often denote singleton sets by omitting set braces, so $\{s\}$ becomes s. An *$s\bar{t}$-set* contains s but not t; a *$\bar{t}$-set* is a nonempty set not containing t. If f is a function $f : S \to \mathbf{R}$ or a vector $f \in \mathbf{R}^S$ then for any set $T \subseteq S$, $f(T)$ denotes $\sum\{f(t) \mid t \in T\}$.

In a graph with vertices v and w, the notation vw denotes an undirected edge joining v and w or a directed edge from v to w; it will be clear from context which is meant. For a digraph G, G^R denotes the reverse digraph, i.e., all edges of G are reversed.

Let W be a set of vertices in graph G. If G is undirected then $\gamma(W)$ denotes the set of all edges with both ends in W. If G is directed then $\delta(W)$ and $\rho(W)$ denote the set of edges uv with W a $u\bar{v}$-set and a $v\bar{u}$-set respectively. If the graph G is not clear we write it as a subscript, e.g., $\delta_G(W)$. If G has a capacity function $c : E \to \mathbf{R}_+$ then the *out-degree* (*in-degree*) of W is $c(\delta(W))$ $(c(\rho(W)))$.

Consider the set of vectors $\mathbf{R}^E$. For vectors b and c, $b \leq c$ means $b_i \leq c_i$ for each $i \in E$. A *polymatroid function* is a function $f : 2^E \to \mathbf{R}_+$ such that $f(\emptyset) = 0$, and f is nondecreasing and submodular, i.e., for any subsets A, B of E, $A \subseteq B$ implies $f(A) \leq f(B)$, and $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. Such an f determines the *polymatroid* $P(f) = \{b \in \mathbf{R}_+^E \mid b(F) \leq f(F) \text{ for all } F \subseteq E\}$. The *graphic polymatroid function* r has $r(F)$ equal to the number of edges in any maximal forest of F, for any $F \subseteq E$. For any $k \in \mathbf{R}_+$, $P(kr) = kP(r)$ is a *graphic polymatroid.*

Fix a polymatroid $P = P(f)$. A set $F \subseteq E$ is *closed* in P if $F \subset F'$ implies $f(F) < f(F')$. For any $b \in P$, a set $F \subseteq E$ is *tight* if $b(F) = f(F)$. For any such b the union of tight sets is tight (by submodularity). Thus any $b \in P$ has a unique maximal tight set T. Clearly T is closed.

For any vector $c \in \mathbf{R}_+^E$, a *base* (or *P-base*) of c is any maximal vector $b \in P$ with $b \leq c$. Any such b has

$$b(E) = \min\{c(\overline{F}) + f(F) \mid F \subseteq E\}. \tag{1}$$

The minimizing set F^* in (1) can be taken as the maximal tight set of b. In fact (1) implies that F^* is uniquely determined by c.

The *polymatroid greedy algorithm* finds a minimum-cost P-base b of a given vector c, as follows. It initializes b to the zero vector. Then it examines each component i ($1 \leq i \leq |E|$) in order of nondecreasing cost. For each i it increases b_i as much as possible keeping $b \in P$ and $b \leq c$.

The polymatroid covering and packing problems are defined in Section 2. For more background on polymatroids see [C85a, W].

2. Newton's method for covering and packing in polymatroids

This section discusses the application of Newton's method to covering and packing problems in polymatroids. It proposes the prefix algorithm as an efficient way to find a base in Newton's method.

Consider a polymatroid $P = P(f)$ and an arbitrary vector $c \in \mathbf{R}_+^E$. The (*fractional*) *P-packing number of c* is the largest $k \in \mathbf{R}_+$ such that a $P(kf)$-base b of c has $b(E) = kf(E)$. The *integral P-packing number* is the floor of this value. Next suppose that $c(e) = 0$ whenever $f(e) = 0$. The (*fractional*) *P-covering number of c* is the smallest $k \in \mathbf{R}_+$ such that $c \in P(kf)$. The *integral P-covering number* is the ceiling of this value.

Let f be integer-valued. Baum and Trotter [BT] (and independently Giles [Gi]) show that for any integer k, any integral vector in $P(kf)$ is the sum of k integral vectors in $P(f)$. This implies the following characterizations of the packing and covering numbers for integral f, $P = P(f)$ and an integral vector c. The integral P-packing number of c is the greatest number of integral P-bases summing to at most c. (Note that an integral vector has an integral $P(g)$-base for an integral function g.) The fractional P-packing number of c is the greatest weight k such that some collection of P-bases, each with a positive rational weight, has total weight k and weighted sum at most c. The integral P-covering number of c is the smallest number of integral vectors in P with sum c. The fractional P-covering number of c is the smallest weight k such that some collection of integral vectors in P, each with a positive rational weight, has total weight k and weighted sum c.

A number of previous algorithms for covering and packing in graphic polymatroids [C85a, PQ82a, PaW84] have been based on Newton's method. We now summarize this approach, at the same time generalizing it to arbitrary polymatroids.

We first describe Newton's method for fractional optimization, also known as Dinkelbach's method [D]. It computes $\min\{\alpha(x)/\beta(x) \mid x \in \mathcal{X}\}$. Here $\mathcal{X}$ is a set equipped with functions $\alpha, \beta : \mathcal{X} \to \mathbf{R}$ with $\beta(x) > 0$ for all $x \in \mathcal{X}$. Equivalently we wish to find the largest value k making $\alpha(x) \geq k\beta(x)$ for all $x \in \mathcal{X}$. In this version it makes sense to relax the assumptions and allow $\beta(x) = 0$ if it implies $\alpha(x) \geq 0$.

The algorithm maintains a value k that is an upper bound on the desired value. k is initialized to any upper bound. Then the following loop is executed.

while $\min\{\alpha(x) - k\beta(x) \mid x \in \mathcal{X}\} < 0$ **do begin**
let x^* be a minimizer of $\{\alpha(x) - k\beta(x) \mid x \in \mathcal{X}\}$;
$k \leftarrow \alpha(x^*)/\beta(x^*)$; **end**;

We always have $\beta(x^*) \neq 0$, so the assignment to k is well-defined and maintains k as an upper bound on the desired value. Thus if the algorithm halts, k equals the desired minimum value (and x^* is a minimizer). Note that each assignment to k reduces its value. Each iteration of the loop is called a *Newton iteration.*

We now show that for our polymatroid problems Newton's method does a small number of iterations and each iteration amounts to a simple polymatroid problem.

Theorem 2.1. Let f be an integral polymatroid function. Newton's method computes the (integral or fractional) $P(f)$-covering or packing number of a vector c in at most $f(E) + 1$ iterations. Each iteration finds a base of c and its maximal tight set in a polymatroid $P(kf)$.

Proof. First consider the problem of finding the packing number of a vector c for an arbitrary polymatroid P. Formula (1) of Section 1 implies the packing number is the largest k such that any set $X \subseteq E$ has $k(f(E) - f(X)) \leq c(\overline{X})$. Finding k is the second optimization problem for Newton's method. Since $c(\overline{X}) \geq 0$ Newton's method is applicable.

Each Newton iteration computes $\min\{c(\overline{X}) + kf(X) \mid X \subseteq E\} - kf(E)$ and a minimizer. By (1) the minimizer X is the maximal tight set of a $P(kf)$-base of c, as claimed. Note that initialization is simple, e.g., choosing $X = \emptyset$ gives $c(E)/f(E)$ as a valid upper bound.

If f is integer-valued then there are at most $f(E)+1$ iterations. This follows since each iteration before the last increases $f(X)$, a fact which is well-known (even for totally arbitrary functions f; see [e.g., S]). For completeness we prove the following slightly stronger fact: The minimizers T for each Newton iteration before the last form a nested family, with $f(T)$ strictly increasing.

First we show a general fact about polymatroids. Consider a polymatroid function f and a vector $c \in \mathbf{R}_+^E$. For positive real values $k < k'$ let T be the maximal tight set of a $P(kf)$-base of c, and similarly for T'. Then $T' \subseteq T$. In proof let y' be a $P(k'f)$-base of c. Let $z = (k/k')y'$. Then $z \in P(kf)$ with T' tight, and $z \leq c$. Thus z can be enlarged to a $P(kf)$-base of c with tight set including T'. Hence $T' \subseteq T$.

This fact implies that the minimizers T for each Newton iteration before the last form a nested family. Furthermore $f(T)$ is strictly increasing. This follows since each T is closed, and two consecutive iterations with minimizers T, T' have $T' \neq T$ if neither iteration is the last. Thus if f is integer-valued, Newton's method does at most $f(E) + 1$ iterations, as desired.

Now consider the problem of finding the covering number of a vector c for an arbitrary polymatroid P. The definition implies the covering number is the smallest k where c is a base of c in $P(kf)$. By formula (1) this means that any set $X \subseteq E$ has $kf(X) \geq c(X)$. Thus we seek the largest value $-k$ such that $(-k)f(X) \leq -c(X)$ for all $X \subseteq E$. This is the second optimization problem for Newton's method. By assumption for covering $f(X) = 0$ implies $c(X) = 0$ so Newton's method is applicable.

Each Newton iteration computes $\min\{-c(X) - kf(X) \mid X \subseteq E\}$ and a minimizer. This is the same as computing $\min\{c(\overline{X}) - kf(X) \mid X \subseteq E\} - c(E)$ and a minimizer. If k is negative the desired minimizer X is the maximal tight set of a $P((-k)f)$-base of c. We initialize k to a convenient upper bound such as $-c(E)/f(E)$. Since the initial k is negative all values k are negative. Thus each Newton iteration finds a polymatroid base as claimed.

The efficiency analysis is essentially the same as for packing. There are at most $f(E) + 1$ iterations, and the minimizers form a nested family with f decreasing. ∎

We now give a refined version of Newton's method for packing in polymatroids. Section 7 implements this method for graphic polymatroids. (Our algorithm for covering in graphic polymatroids uses ideas in Theorem 2.1 but goes beyond it.)

Each Newton iteration can be implemented by using the greedy algorithm for finding a polymatroid base. The greedy algorithm finds a base in $|E|$ iterations, so Theorem 2.1 implies $O(|E|f(E))$ greedy-algorithm iterations. We improve this bound by using a variant of the greedy algorithm called the prefix algorithm. We first describe the prefix algorithm and then show how it is used in Newton's method for packing.

Fix a polymatroid P with a cost vector $a \in \mathbf{R}^E$. We seek a minimum-cost base b of a given vector $c \in \mathbf{R}^E_+$, i.e., b is a P-base of c having the smallest possible inner product ab.

Consider a vector $b \in \mathbf{R}^E$. For a subset $F \subseteq E$, b_F denotes the restriction of b to F, i.e., components not in F are set to 0. Suppose E is indexed from 1 to m. For $i, j \in E$, $b_{i..j}$ abbreviates $b_{[i..j]}$. A *prefix of* b is b itself or a vector d where for some index i, $1 \leq i \leq m$, $b_{1..i-1} \leq d < b_{1..i}$. In the latter case d is an *i-prefix* (for the former case, b is an m-prefix of b). A *longest prefix of* b satisfying some condition is a lexically maximum prefix d of b that satisfies the condition

(equivalently $d(E)$ is maximum).

The *prefix algorithm* works as follows. Index the elements of E from 1 to m so cost is non-decreasing. The algorithm maintains an index $i \leq m$, a vector $b \in \mathbf{R}_+^E$ and a set $T \subseteq E$. Each iteration increases i and makes b the vector found by the first i iterations of the greedy algorithm. T is a tight set for b in P.

$i \leftarrow 0$; $b \leftarrow 0$; $T \leftarrow \emptyset$;
while $i < m$ **do begin**
  $b \leftarrow$ (the longest prefix of $b + c_{[i+1..m]-T}$ that is in P);
    let b be an i-prefix;
  $T \leftarrow$ (the maximal subset of E that is tight for b in P); **end**;

Correctness follows from the easily-checked invariant for i and b given above.

The prefix algorithm does at most as many iterations as the greedy algorithm, but for graphic polymatroids and certain others it does fewer. Specifically we show that if $P = P(kf)$ for an integer-valued function f and positive real value k then there are at most $1 + f(E)$ iterations. This follows since each iteration except the last enlarges T. (An iteration not the last ends with $b_i < c_i$, $i \notin T$ before the iteration and $i \in T$ after the iteration.) Since each T is maximal tight $kf(T)$ is strictly increasing, which is equivalent to $f(T)$ strictly increasing.

The crucial part of implementing the prefix algorithm efficiently is the first line of the loop, finding the longest prefix b. For graphic polymatroids we use the steppingstone approach, proposed in [GW] for matroids: We use a polymatroid SP that contains P and is computationally simpler. We find b as follows:

$d \leftarrow$ (the longest prefix of $b + c_{[i+1..m]-T}$ that is in SP);
$b \leftarrow$ (the longest prefix of d that is in P);

This modification finds the same vector, say b^*, as found by the original algorithm: $P \subseteq SP$ implies b^* is a prefix of d.

Now we give the packing algorithm based on the prefix algorithm. The idea is to carry over the base b and tight set T from one Newton iteration to the next. The details are as follows.

Make b and T global variables, so they retain their values from the previous iteration. The first Newton iteration chooses an arbitrary cost function for the edges and runs the prefix algorithm unchanged. For an iteration after the first, let k_- and k be the previous and current values of k, respectively. Let $t = |T|$. Revise the cost function so any element of T is cheaper than any element

of $E - T$. Reindex E according to this cost function, so the elements of T are indexed from 1 to t. Change the initialization of the prefix algorithm to the following:

$$i \leftarrow t; \quad b \leftarrow \tfrac{k}{k_-}\, b_{1..t} \ ;$$

These statements make T tight for b, i.e., $b \in P(kf)$ and $b(T) = kf(T)$. The rest of the algorithm is unchanged.

Corollary 2.1. Let f be an integral polymatroid function. Newton's method for $P(f)$-packing using the prefix algorithm finds a total of at most $2f(E) + 1$ prefixes.

Proof. Theorem 2.1 shows Newton's method for packing has at most $1 + f(E)$ iterations. Each iteration of the prefix algorithm except the last in its Newton iteration enlarges T. Since T is closed in $P(f)$ this occurs at most $f(E)$ times. This gives at most $1 + 2f(E)$ iterations of the prefix algorithm. ∎

A potential drawback of this packing algorithm compared to using the straightforward greedy algorithm is numerical accuracy: Repeated execution of the assignment $b \leftarrow \frac{k}{k_-}\, b_{1..t}$ can create rational numbers with large numerators and denominators. These large numbers can be avoided by doing a polymatroid contraction operation for the set T, assuming there is a good implementation of contraction. Such is the case for graphic polymatroids. We give the details of the graphic case in Section 7.

3. A characterization for graphic polymatroids

This section shows how to transform problems about graphic polymatroids to s,t-cut and t-cut problems. Our approach combines two known constructions. We begin by reviewing the constructions.

We start with an undirected graph $G = (V, E)$ having functions $c_E : E \to \mathbf{R}_+$ and $c_V : V \to \mathbf{R}_+$. We shall introduce several graphs, most notably E^* and EG. All these graphs have a capacity function on the edges. In this section c is used to denote all these capacity functions. The meaning of c will always be clear, e.g., $c(\delta_{EG}(S))$ denotes the capacity of all edges of EG that leave S. Throughout this paper γ always denotes γ_G.

The first construction is a well-known bipartite version of an undirected graph (see [PQ82a] and its references). We define a digraph $G^*(c_E, c_V)$; we abbreviate this to G^* when possible. G^* has vertex set $\{s,t\} \cup V \cup E$. The fact that $v \in V$ and $e \in E$ occur in G^* as well as in G will not

cause confusion. The edge set of G^* is $\{se \mid e \in E\} \cup \{ev, ew \mid e = vw \in E\} \cup \{vt \mid v \in V\}$. A capacity function c on G^* is defined by the identities $c(se) = c_E(e)$, $c(ev) = c(ew) = \infty$, $c(vt) = c_V(v)$.

The second construction is the "equivalent graph" of [Ga94]. It eliminates the source vertex of a flow network. This paper only constructs equivalent graphs for graphs G^*. We give a two step construction of the equivalent graph for G^*, where the first step is from [Ga94] and the second step reduces the size.

For the first step let f be a maximum flow from s to t on G^*. Assume that f saturates all edges of $\delta(s)$. Let EQ be the residual graph of f with vertex s deleted. So in EQ the capacity function $c(e)$ gives the residual capacity of e with respect to flow f. Any $\bar{t}$-set X in EQ has $c(\delta_{EQ}(X)) = c(\delta_{G^*}(s + X)) - c_E(E)$. This can be easily verified or see [Ga94].

The second step starts with EQ. For each $e = xy \in E$ delete e and replace it by edges xy and yx of capacities $f(ex)$ and $f(ey)$ respectively. Then delete all edges of $\delta(t)$. The result is the *equivalent graph* EG. EG has vertex set $t \cup V$ and $O(m)$ edges. For any set $W \subseteq V$, $c(\delta_{EQ}(W \cup \gamma(W))) = c(\delta_{EG}(W))$.

Let us summarize the relationship between G, G^* and EG. Let W be a (possibly empty) set of vertices in G. It gives an $s\bar{t}$-set in G^*, $s \cup W \cup \gamma(W)$, having out-degree equal to

$$c_E(E) + c_V(W) - c_E(\gamma(W)).$$

Assume that a maximum flow in G^* has value $c_E(E)$, i.e., it saturates all edges of $\delta(s)$. Then the set W has out-degree in EG equal to

$$c_V(W) - c_E(\gamma(W)).$$

We extend this relationship to the following.

Lemma 3.1. Let $G = (V, E)$ be an undirected graph, $c_E : E \to \mathbf{R}_+$ and $c_V : V \to \mathbf{R}_+$. Let G^* denote $G^*(c_E, c_V)$, with EG its equivalent graph. Then

$$\min\{c_V(W) - c_E(\gamma(W)) \mid \emptyset \neq W \subseteq V\} = \min\{c(\delta_{G^*}(S)) - c_E(E) \mid S \neq \{s\} \text{ an } s\bar{t}\text{-set of } G^*\}.$$

If this common minimum is nonnegative then it equals

$$\min\{c(\delta_{EG}(S)) \mid S \text{ a } \bar{t}\text{-set of } EG\}.$$

Proof. Consider the first part. The relationship stated above implies that any quantity in the left-hand set is in the right-hand set. (A nonempty set $W \subseteq V$ gives a set $s \cup W \cup \gamma(W)$ that is distinct from $\{s\}$.) Furthermore these terms achieve the minimum of the right-hand set, since any $s\bar{t}$-set S has out-degree at least the out-degree of $s \cup \gamma(S \cap V) \cup (S \cap V)$. We can assume $S \cap V \neq \emptyset$ (otherwise S equals $\{s\}$ or has infinite out-degree).

For the second part of the lemma, its hypothesis implies that a maximum flow in G^* has value $c_E(E)$. So the second part follows from the properties noted for EQ and EG. ∎

We use this result to characterize the graphic polymatroid $P = P(kr)$. We also characterize the related polymatroid $SP = P(kf)$ where for any set $F \subseteq E$, $f(F)$ is the number of distinct vertices on edges F. Note that $P \subseteq SP$ (since $r \leq f$). The following result uses the easily-verified fact that $b \in P$ iff every nonempty set of vertices $W \subseteq V$ has $b(\gamma(W)) \leq k(|W| - 1)$.

Theorem 3.1. Let b be a vector in $\mathbf{R}_+^E$ and let G^* denote $G^*(b,k)$. $b \in SP$ iff a maximum flow on G^* has value $b(E)$. $b \in P$ iff $b \in SP$ and in the equivalent graph EG for G^*, any $\bar{t}$-set has out-degree at least k. ∎

The equivalent graph provides other information about graphic polymatroids as well. First we characterize the maximal tight set of a vector b in P. A tight set of b has the form $\bigcup\{\gamma(U) \mid U \in \mathcal{U}\}$, where $\mathcal{U}$ is a family of disjoint sets of vertices U having $b(\gamma(U)) = k(|U| - 1)$ and $|U| > 1$. For the maximal tight set, $\mathcal{U}$ is the family of maximal nonsingleton $\bar{t}$-sets of out-degree k in EG. (This follows from the relationship preceding Lemma 3.1.)

Next we show that the representation of b as a collection of forests can be found in EG. We begin by reviewing a theorem of Edmonds (also used in Section 4).

Fix a digraph G and a vertex s. For a nonnegative integer k, a *complete k-intersection* consists of k (undirected) spanning trees collectively containing k edges directed to each vertex $\neq s$. (Unlike [Ga91a] in this paper we assume that a complete k-intersection is always given with its partition into k spanning trees.) Edmonds showed that G has a complete k-intersection iff any $\bar{s}$-set U has $|\rho(U)| \geq k$ [E69].

Take G, b, G^* and EG as in Theorem 3.1 with $b \in P$. Suppose first that k and b are integral. Theorem 3.1 implies that EG^R has a complete k-intersection T. Ignoring the directions of edges in T, we claim that every edge e of G occurs in precisely $b(e)$ trees of T. This claim implies that $T - t$ gives the desired partition of b into k forests.

We prove the claim in two steps. First observe that for each edge $e = xy$ of G, the total capacity of xy and yx in EG is $b(e)$. This follows since using notation introduced above, in EG xy and yx have capacity $f(ex)$ and $f(ey)$ respectively and $b(e) = f(ex) + f(ey)$.

To prove the claim it suffices to show that T consists of all the edges of EG^R (by the observation just made). T contains $k|V|$ edges, since EG^R has $|V| + 1$ vertices. So it suffices to show the edges of EG have total capacity $k|V|$. The total capacity of EG equals the total out-degree of all vertices

of V in EG. Each of these vertices has out-degree k in G^*, and this out-degree is preserved when we pass to the equivalent graph.

Now consider the case where k and b take on arbitrary rational values. Reduction to the integral case shows the following. EG^R has a collection of spanning trees T_i, each with a nonnegative rational weight w_i, such that $\sum w_i = k$ and ignoring edge directions the total weight of all trees containing a given edge e of G is precisely $b(e)$. Thus $T - t$ gives a partition of b into forests of total weight k.

We remark that the algorithm of [Ga91a] finds a complete k-intersection efficiently for small k (see Section 4). This allows our algorithm for covering graphic polymatroids with small capacities to find a tree decomposition (Section 5). The other algorithms of this paper do not produce a tree decomposition because they involve large k. [GM] presents algorithms for finding a complete k-intersection for large k but these algorithms are slower than the other algorithms of this paper.

We close this section with some motivation. Our graphic packing algorithm amounts to implementing the prefix algorithm. Recall the main task is given a vector d, find the longest prefix of d that is in P. The prefixes of d will give a parameterized family of equivalent graphs EG_λ. As Theorem 3.1 suggests, our task will be to find the smallest λ where each $\overline{t}$-set has out-degree at least k. The next section defines a parameterized problem that generalizes this task. The generalization is also the key to our covering algorithm.

4. Parametric augmentation

This section defines our parametric problem on digraphs. It presents two efficient solutions, one oriented toward large capacities and the other small.

The *parametric augmentation problem* (*for s-sets*) is defined by a digraph G with distinguished vertex s and parameterized capacity function $c_\lambda : E \to \mathbf{R}_+$ where λ is a parameter in $\mathbf{R}_+$. Also given is a target function $\tau : \mathbf{R}_+ \to \mathbf{R}_+$. The problem is to find $\overline{\lambda} = \min\{\lambda \mid c_\lambda(\rho(U)) \geq \tau(\lambda)$ for any $\overline{s}$-set $U\}$. $\overline{\lambda}$ is infinite if the set is empty.

We need some additional assumptions to make the problem well-defined and tractable. To make the problem well-defined we assume that any $\overline{s}$-set U has a value λ_U (possibly infinite) such that $c_\lambda(\rho(U)) \geq \tau(\lambda)$ iff $\lambda \geq \lambda_U$. To make the problem tractable we assume that given U we can compute λ_U in time $O(m)$. Also given λ and an edge e we can compute $c_\lambda(e)$ in $O(1)$ time. Finally and most importantly we assume monotonicity properties for the capacities: Assume $c_\lambda(e)$ is nondecreasing for each $e \in \delta(s)$ and nonincreasing for all other e; furthermore for each vertex v, $c_\lambda(\rho(v))$ is nondecreasing.

We adapt the Hao-Orlin mincut algorithm to solve the parametric augmentation problem. We

first review the Hao-Orlin algorithm. Its input is a digraph with capacity function $c : E \to \mathbf{R}_+$ and a vertex s. It returns a value M equal to the smallest in-degree $c(\rho(W))$ of an $\bar{s}$-set W. The algorithm works by repeatedly finding the smallest value of an S,t-cut. Here S is a set of vertices, that is initially $\{s\}$; t is a vertex not in S; over the course of the algorithm t takes on the value of every vertex $\neq s$; after the smallest S,t-cut is found t gets added to S and the next value of t is chosen. The algorithm at a high level is as follows:

HO_initialize; /* sets $S = \{s\}$ and chooses a sink t */
while $S \neq V$ **do begin**
HO_preflow_push; /* makes (D, W) a minimum S,t cut */
$M \leftarrow \min\{M, c(\rho(W))\}$;
HO_select_new_sink; /* adds t to S and chooses a new t */ **end**;

The routines HO_initialize, HO_preflow_push, HO_select_new_sink are described in detail in [HO]. HO_preflow_push is a modification of the Goldberg-Tarjan maxflow algorithm [GT]; HO_select_new_sink adds the old sink t to S and chooses a new one.

The idea of our parametric augmentation algorithm is to execute the Hao-Orlin algorithm, increasing the parameter λ whenever a mincut smaller than the current target $\tau(\lambda)$ is discovered. The algorithm is as follows. Assume that whenever the algorithm changes the value of λ that automatically changes each edge capacity function to c_λ. The function $f : E \to \mathbf{R}_+$ is the current flow function for the HO algorithm (f is actually a preflow).

$\lambda \leftarrow 0$; HO_initialize;
while $S \neq V$ **do begin**
HO_preflow_push;
while $c_\lambda(\rho(W)) < \tau(\lambda)$ /* (D,W) is a minimum S,t cut */ **do begin**
$\lambda \leftarrow \lambda_W$; /* increase λ */
for each edge $e \in \delta(s)$ **do** $f(e) \leftarrow c_\lambda(e)$; /* increase flow */
for each edge $e \notin \delta(s)$ **do** $f(e) \leftarrow \min\{f(e), c_\lambda(e)\}$; /* decrease flow */
HO_preflow_push; **end**;
HO_select_new_sink; **end**;

We make two observations to show this algorithm is correct. First we must show that the invariants maintained by the HO routines continue to hold. The HO routines maintain f as a preflow with "W-valid" distance labels, and a "dormancy property." It is easy to see that these

invariants are maintained in the "increase flow" line, because it is similar to pushes from s done by the HO routines. We need only check the invariants in the "decrease flow" line. The decrease in flow does not create a new residual edge. So we need only check that f is maintained f as a preflow, i.e., each vertex has nonnegative flow excess. It suffices to show that when λ changes from λ to a larger value λ', the change in the excess at v is nonnegative. For each edge $e \notin \delta(s)$ let $\Delta(e)$ be the amount that our algorithm decreases the flow $f(e)$. Then the excess at v increases by $(c_{\lambda'} - c_\lambda)(sv) - \Delta(\rho(v) - \{sv\}) + \Delta(\delta(v)) \geq (c_{\lambda'} - c_\lambda)(sv) - (c_\lambda - c_{\lambda'})(\rho(v) - \{sv\}) = (c_{\lambda'} - c_\lambda)(\rho(v))$. The last quantity is nonnegative by assumption for the augmentation problem.

The second observation is that the algorithm halts with the desired value $\overline{\lambda}$. The final λ is either 0 or λ_W for some set W, so it is $\leq \overline{\lambda}$. Furthermore the algorithm has verified that any $\overline{s}$-set U has $c_\lambda(\rho(U)) \geq \tau(\lambda)$, i.e., $\lambda_U \leq \lambda$. To see this let t' be the first vertex of U chosen as a sink. Suppose the last iteration with sink t' ends with set W' and parameter value λ' satisfying $c_{\lambda'}(\rho(W')) \geq \tau(\lambda')$. The definition of W' shows $c_{\lambda'}(\rho(U)) \geq c_{\lambda'}(\rho(W))$. This implies $\lambda_U \leq \lambda' \leq \lambda$ as desired.

Theorem 4.1. The parametric augmentation problem for s-sets can be solved in time $O(nm \log(n^2/m))$ and space $O(m)$.

Proof. We need only analyze the efficiency of the algorithm. The main observation is that the parameter λ is increased at most $2n$ times. In proof consider a fixed sink t. Each increase in λ after the first for sink t is caused by a new cut (D, W). Examination of the HO algorithm shows this means the dormant node set D is enlarged. [HO] proves that D is enlarged, without changing the sink, at most n times in the entire algorithm. The proof also holds for our modified algorithm. This gives the desired bound of $2n$ iterations.

To compute the time for the algorithm note that computing new λ values uses $O(nm)$ time total, extra pushes use time $O(n^2)$ and flow decreases use time $O(nm)$. We implement the HO algorithm using dynamic trees, exactly as in [HO]. We conclude that our modified algorithm does not change the time bound. ∎

Our second augmentation algorithm is oriented towards digraphs where the target $\tau(\overline{\lambda})$ is small. We strengthen two assumptions of the problem. Assume that for a family $\mathcal{U}$ of disjoint sets of vertices U, we can compute all of the values λ_U, $U \in \mathcal{U}$ in total time $O(m)$ (actually time $O(m \log(n^2/m))$ is sufficient). Also assume that edges not in $\delta(s)$ have constant capacity, but as

before $c_\lambda(e)$ is nondecreasing for each $e \in \delta(s)$. (This is the case of augmentation that is needed in Section 5.) We call this version of the problem as *the augmentation problem with parametric* $\delta(s)$.

Since we are concerned with small connectivities we assume that λ and all functions c_λ are integral-valued. Thus each c_λ specifies a multigraph. In resource estimates we assume the parameter m counts the edges of a graph, with each edge counted according to its capacity.

We first briefly review the round robin algorithm of [Ga91a] that finds an $\overline{s}$-set of minimum in-degree. Fix a digraph G and a vertex s. Recall the notion of complete k-intersection from Section 3. The round robin algorithm takes as its input a complete $(k-1)$-intersection; its output is a complete k-intersection, if such exists. In this discussion we assume that when a complete k-intersection exists, round robin outputs such an intersection and the value k; when a complete k-intersection does not exist, round robin outputs the given $(k-1)$-intersection and the value $k-1$. Thus to find the smallest in-degree of an $\overline{s}$-set we repeatedly run round robin until T is a complete k-intersection for k maximal. The final k is the desired smallest in-degree.

We state the augmentation algorithm using the following convention for edge capacities: The algorithm maintains a variable $c(e)$ for the capacity of an edge e. When the algorithm changes the parameter λ, it also changes each $c(e)$ by the appropriate amount (we do not explicitly mention changes to $c(e)$).

$T \leftarrow \emptyset$; $\lambda \leftarrow 0$; /* set capacities $c(e)$ appropriately */
run round robin to make T a complete k-intersection, for maximal $k \leq \tau(\lambda)$;
while $k < \tau(\lambda)$ **do begin**
  $\lambda \leftarrow \max\{\lambda_U \mid U$ an $\overline{s}$-set and $c(\rho(U)) = k\}$; /* increase λ, and capacities $c(e)$ */
  run round robin to enlarge T to a complete k-intersection, for maximal $k \leq \tau(\lambda)$;
  end;

Correctness of this algorithm follows from the fact that k is guaranteed to increase every iteration.

To estimate efficiency we must elaborate on two issues. The first is how the "increase λ" line calculates the new value of λ. First note that this is trivial for the special case of parametric augmentation used in Section 5: In this case changing λ to $\lambda + 1$ increases the capacity of sv for every $v \neq s$. Thus the smallest in-degree of an $\overline{s}$-set increases. This allows us to use $\lambda \leftarrow \lambda + 1$ for the "increase λ" line.

The general case is treated as follows. [Ga91b] presents an algorithm that given T, finds the family $\mathcal{U}$ of all minimal $\overline{s}$-sets of in-degree k in time $O(m)$. It is easy to see that in the "increase

λ" line the sets U can be restricted to the family $\mathcal{U}$. The sets of $\mathcal{U}$ are disjoint, so by assumption each value λ_U, $U \in \mathcal{U}$ can be computed in time $O(m)$.

The second issue is to ensure the efficiency of round robin. Let m denote the number of edges in the graph excluding $\delta(s)$. Round robin enlarges a complete $(k-1)$-intersection to a complete k-intersection in time $O((m+kn)\log n)$; if any edge not in $\delta(s)$ has capacity $O(1)$ the time bound decreases to $O(m\log(n^2/m)+kn)$. The space is $O(m+kn)$. In the augmentation algorithm the term kn can dominate these estimates. We now modify the augmentation algorithm, reducing n to a value n' so that $kn' = O(m)$ always holds.

We make two changes. First we modify the initialization of λ to $\lambda \leftarrow \max\{\lambda_v \mid v \in V - s\}$. Second whenever round robin increases k to a power of two we execute the following *clean-up step*:

> **for** each vertex $v \neq s$ with $c(sv) > c(\delta(v))$ **do begin**
>
> **for** each edge $vw \in \delta(v)$ **do** increase $c(sw)$ by $c(vw)$;
>
> delete v from the graph; **end**;
>
> $T \leftarrow \emptyset$; run round robin to make T a complete k-intersection;

These modifications do not change the desired value $\overline{\lambda}$. In proof let v be a vertex that gets deleted. Any $v\overline{s}$-set U has larger in-degree than $U - v$, both for the current λ and any larger value. For $U = \{v\}$ the initialization of λ ensures $\lambda \geq \lambda_U$. Hence deleting v and adjusting capacities $c(sw)$ as in the clean-up step does not change $\overline{\lambda}$.

Now we show that for n' the number of vertices still in the graph, $kn' = O(m)$ as claimed. It suffices to show that at any time any vertex v remaining in the graph is on at least $k/4$ edges of the original graph G (counting each edge according to its initial capacity). In proof consider a vertex v that is not deleted in a clean-up step. At the time of this step $c(sv) \leq c(\delta(v))$. Also $c(\rho(v)) \geq k$ since there is a complete k-intersection. Thus $c(\delta(v))$ or $c(\rho(v) - sv)$ is at least $k/2$, and until the next clean-up step it is at least $k/4$.

In the following theorem recall that m counts each edge of the graph according to its initial capacity.

Theorem 4.2. The augmentation problem with parametric $\delta(s)$ can be solved in time $O(\tau(\overline{\lambda})m\log n)$ and space $O(m)$. The time is $O(\tau(\overline{\lambda})m\log(n^2/m))$ if any edge not in $\delta(s)$ has capacity $O(1)$.

Proof. Since $kn' = O(m)$ each execution of round robin uses time $O(m\log n)$. The total time for round robin excluding clean-up steps is $O(\tau(\overline{\lambda})m\log n)$. The time for a clean-up step at the value

k is $O(km \log n)$. Summing over all powers of two $k \leq \tau(\overline{\lambda})$ gives total time $O(\tau(\overline{\lambda})m \log n)$. The logarithmic term in these estimates decreases to $\log(n^2/m)$ if any edge not in $\delta(s)$ has capacity $O(1)$. ∎

We can construct a complete $\tau(\overline{\lambda})$-intersection T for the original graph G with capacity function $c_{\overline{\lambda}}$ in time proportional to the size of T, $O(\tau(\overline{\lambda})n)$. To do this start with the final intersection T of the algorithm, and add the vertices v that were deleted in clean-up steps in reverse order of their deletion. To add v first do the following for each tree T_i of T that contains an edge sw where $vw \in \delta(v)$ and $c(vw) > 0$: For all such w in T_i, replace sw by vw in T_i and decrease $c(vw)$ by 1; then add sv to T_i and decrease $c(sv)$ by 1. Second for every other tree T_i in T, add v to T_i by adding an edge from a vertex of T_i to v and decreasing its capacity.

To show this construction is correct observe that in the first step edge sv starts out with positive capacity. This follows from the deletion criterion of the clean-up step. In the second step the edge to T_i exists, by the clean-up step that deleted v.

The covering algorithm of Section 5 requires only a subset of the above intersection T, specifically $T - \delta(s)$. This (partitioned) subset consists of $\tau(\overline{\lambda})$ forests of edges in the original graph. It can be constructed in time $O(m)$ using the same algorithm.

5. Covering graphic polymatroids

This section presents algorithms to compute the arboricity efficiently, for both large-capacity and small-capacity graphs.

Fix an undirected graph G and a vector $c \in \mathbf{R}_+^E$. G determines the graphic polymatroid $P = P(r)$. The P-covering number is called the *arboricity*, denoted Γ. More precisely *integral (fractional) arboricity* refers to the integral (fractional) P-covering number. Γ denotes integral or fractional arboricity, as determined by context. If c is integral, a collection of forests of G *covers* the edges of G if it contains each edge e precisely $c(e)$ times. The integral arboricity is the smallest number of forests that covers the edges of G. The fractional arboricity has a similar interpretation. Nash-Williams gave the formula for integral arboricity, $\Gamma = \max\{\lceil c(\gamma(W)/(|W| - 1)\rceil \mid W \subseteq V, |W| > 1\}$ [NW]. (This formula is implicit in Section 3.)

Graph G also determines the polymatroid $SP = P(f)$ defined in Section 3. The *density* d of c is the SP-covering number of c. *Integral* and *fractional density* have the obvious meaning, and d denotes the variant determined by context. We use the density as a steppingstone to the arboricity. Clearly $\Gamma \geq d$, for integral and fractional variants.

Theorem 3.1 shows that d is the smallest k where a maximum flow on $G^*(c,k)$ has value $c(E)$. For any k let EG_k denote the equivalent graph of $G^*(c,k)$. Theorem 3.1 shows that Γ is the smallest $k \geq d$ where in EG_k, any $\bar{t}$-set has out-degree at least k. Note that for any $k \geq d$, EG_k is the same as EG_d except that the capacity of each edge xt increases by $k-d$ (the maximum flow on $G^*(c,k)$ equals $c(E)$ for any $k \geq d$).

Our algorithm to compute the arboricity of a graph G with capacity function $c \in \mathbf{R}_+^E$ works in two steps. The first step computes the density d.

The second step finds Γ. Γ is the smallest $k \geq d$ such that in the reverse digraph EG_k^R, any $\bar{t}$-set has in-degree at least k. We find k by solving the parametric augmentation problem for t-sets specified as follows. The parameter λ equals $k-d$. For $e \in \delta(t)$, $c_\lambda(e) = c_d(e) + \lambda$. The target function is $\tau(\lambda) = d + \lambda$. A $\bar{t}$-set T has λ_T equal to d if $c_d(\rho(T)) \geq d$ and $\frac{d - c_d(\rho(T))}{|T|-1}$ otherwise. (Note that if $|T| = 1$ then $c_d(\rho(T)) = d$ from the definition of equivalent graph.)

Theorem 5.1. The integral and fractional arboricity for a graph with capacity function can be found in time $O(nm \log(n^2/m))$ and space $O(m)$.

Proof. We find the density using the algorithm of [GGT] in time $O(nm \log(n^2/m))$. The assumptions of the parametric augmentation algorithm hold, so Theorem 4.1 shows it is solved in the same time bound. ∎

The same algorithm runs faster on graphs with small capacities. We concentrate on the case of integral capacities that are $O(1)$. Note that in this case $\Gamma = O(\sqrt{m})$. This is easy to check using Nash-Williams' formula.

Theorem 5.2. If all capacities are integral and $O(1)$ then the integer Γ, along with Γ forests that cover the edges of G, can be found in time $O(m^{3/2} \log(n^2/m))$ and space $O(m)$.

Proof. When all capacities are $O(1)$ the density can be found by binary search in time $O(\min\{\sqrt{m}, n^{2/3}\} m \log n)$. ([GW] gives slightly better bounds.) The augmentation problem satisfies the assumptions for augmentation with parametric $\delta(s)$. Hence Theorem 4.2 applies. As noted after the theorem we can find the Γ forests that cover the edges in additional time $O(m)$. ∎

6. Finding a graphic polymatroid base

This section implements the prefix algorithm given in Section 2 to find a minimum-cost base of a graphic polymatroid $P = P(kr)$.

Recall that the prefix algorithm begins an iteration with a vector $b \in P$ where $b_j = 0$ for $j > i$. Let u denote the vector $b + c_{[i+1..m]-T}$. The three steps of an iteration are as follows. The first step sets d to the longest prefix of u that is in SP. We will find d using parametric network flow. The second step sets b to the longest prefix of d that is in P. We will find b using parametric augmentation. The third step finds the maximal tight set for b in P. We will find T using the Hao-Orlin algorithm. Now we describe these steps in detail.

The following notation concerning the graph G^* of Section 3 is useful. For any edge $e \in E$, e^* denotes edge se of G^*. For any flow f on G^*, f_{E^*} denotes the vector of flow values $(f(1^*), \ldots, f(m^*))$.

The first step finds d using the characterization of SP of Theorem 3.1, as follows. Construct the parametric network $G^*(u_{1..\lambda}, k)$. Here λ is an integral parameter ranging over $[1..m]$. Starting at $\lambda = 0$, repeatedly increase λ by 1 until a maximum flow f in the parametric network does not saturate edge λ^*. The desired prefix d is f_{E^*}, where f is the final flow.

This method can compute up to m maximum flows. We reduce this to $O(n)$ maximum flows by doing two parametric flow computations: First starting at $\lambda = 0$, repeatedly increase λ by n to find $\lambda = hn$, the largest multiple of n where a maximum flow saturates all edges of $\delta(s)$. Then starting at $\lambda = hn$, repeatedly increase λ by 1 until as above, a maximum flow does not saturate edge λ^*.

The second step finds b using the characterization of P in Theorem 3.1 and parametric augmentation. Let f denote the final maximum flow from the first step. Let a real-valued parameter λ range from 0 to $d(E)$. Each value of λ will specify a flow f^λ, obtained by retracting the "most recent" flow of f. The equivalent graph of this flow will give a parameterized digraph EG_λ. Now we give the details, first describing f^λ and then describing EG_λ.

To construct f^λ, recall that flow f has value $d(E)$. Flow f^λ has value $d(E) - \lambda$, and $f^\lambda_{E^*}$ is the unique prefix of vector f_{E^*} having this value. Now we specify f^λ on the remaining edges of $G^*(d,k)$ (recall f is a flow on this graph). Consider an edge j^*, and suppose that going from f to f^λ the flow in j^* decreases by Δ. Let j be edge xy. Reduce the flow in edges jx and xt by Δ_x, and the flow in jy and yt by Δ_y, where $0 \le \Delta_x \le f(jx)$, $0 \le \Delta_y \le f(jy)$ and $\Delta_x + \Delta_y = \Delta$.

EG_λ is the equivalent graph of flow f^λ in graph $G^*(f^\lambda_{E^*}, k)$. Let $\overline{\lambda}$ be the smallest parameter value where each $\bar{t}$-set of EG_λ has out-degree at least k. The desired prefix b equals $f^{\overline{\lambda}}_{E^*}$.

It will help to describe EG_λ by specifying how the equivalent graph changes when, as described above, the flow in edge j^* decreases by Δ. Use the same notation as above. Then in the equivalent graph, the capacity of edge xy decreases by Δ_x and the capacity of xt increases by Δ_x. Similar changes occur for yx.

Now we specify the parametric augmentation problem that finds the desired value $\overline{\lambda}$. We work on the reverse digraph EG_λ^R, with $\tau(\lambda) = k$. The above description of EG_λ implies the monotonicity properites for parametric augmentation: $c_\lambda(e)$ is nondecreasing for each $e \in \delta(t)$, nonincreasing for $e \notin \delta(t)$ and $c_\lambda(\rho(v))$ does not change for any vertex v. A value λ_T is computed by increasing λ and retracting flow until the in-degree of T becomes k. It is easy to check the other assumptions of the parametric augmentation problem.

The third step of an iteration of the prefix algorithm updates T. The remark after Theorem 3.1 shows that T corresponds to the family of maximal nonsingleton $\bar{t}$-sets of in-degree k in $EG_{\overline{\lambda}}^R$, for the value $\overline{\lambda}$ found in the second step. This family of sets is easily found by executing the Hao-Orlin algorithm.

Theorem 6.1. A minimum-cost base of a graphic matroid or polymatroid $P(kr)$ can be found in time $O(n^2 m \log(n^2/m))$ and space $O(m)$.

Proof. We need only analyze the time. Section 2 shows that the prefix algorithm has $\leq f(E) + 1 = n$ iterations. Thus it suffices to show that each of the three steps of an iteration uses time $O(nm \log(n^2/m))$. The first step uses the parametric network flow algorithm of [AOST]. As already noted $O(n)$ maximum flows are computed. Hence the time is $O(nm \log(n^2/m))$. The second step solves a parametric augmentation problem, and hence has the same time bound by Theorem 4.1. The third step uses the Hao-Orlin algorithm, and hence has the same time bound. ∎

Cunningham [C85a] proposes the *reinforcement problem*: Given is a graph with capacity functions $c, u \in \mathbf{R}_+^E$ and cost function $a \in \mathbf{R}^E$. The problem is to increase c to $c + z$, where $0 \leq z \leq u$, so the resulting graph has packing number at least k, and the cost az is as small as possible. Cunningham shows the reinforcement problem reduces to two executions of the greedy algorithm.

We refer to this problem as the *optimal augmentation problem for packing.* We can also define a *optimal augmentation problem for covering*: A graph with the same functions is given. The problem is to increase c to $c + z$, where $0 \leq z \leq u$, so the resulting graph has covering number at most k, and az is as large as possible. This problem also reduces to two executions of the greedy algorithm.

Theorem 6.2. The optimal augmentation problems for packing and covering can be solved in time $O(n^2 m \log(n^2/m))$ and space $O(m)$. ∎

7. Packing graphic polymatroids

This section applies the prefix algorithm to get an efficient algorithm for packing graphic polymatroids.

Fix an undirected graph G, a vector $c \in \mathbf{R}_+^E$ and the graphic polymatroid $P = P(r)$ determined by G. The integral P-packing number is the greatest number of spanning forests of G (spanning trees if G is connected) containing each edge e at most $c(e)$ times. The fractional P-packing number equals the strength of a connected graph G. To see this recall that strength is defined as $\min\{c(F)/(r(E) - r(\overline{F})) \mid F \subseteq E,\ r(E) > r(\overline{F})\}$ [C85a]. (The motivation is that the number of connected components of graph $G - F$ is $1 + r(E) - r(\overline{F})$, i.e., deleting F creates $r(E) - r(\overline{F})$ new components, so strength is the smallest effort per new component when we destroy a set of edges.) This expression is Edmonds' formula for the packing number [E65] (alternatively see the proof of Theorem 2.1).

We use Newton's method for packing in polymatroids, with the implementation of the prefix algorithm given in Section 6. (We initialize k to any upper bound on the packing number, e.g., $c(E)/(n-1)$.)

Theorem 7.1. The integral and fractional packing numbers for a graphic polymatroid can be found in time $O(n^2 m \log(n^2/m))$ and space $O(m)$.

Proof. Corollary 2.1 shows Newton's method amounts to $O(f(E)) = O(n)$ iterations of the prefix algorithm. Section 6 shows each iteration uses time $O(nm \log(n^2/m))$. ∎

As mentioned in Section 2 a potential drawback of this packing algorithm is numerical accuracy: the values of b get repeatedly multiplied and are used as capacities in flow networks G^*. We fix this as follows.

Each time the prefix algorithm is executed the initialization makes T tight for b in $P = P(kr)$. The prefix algorithm finds a vector b' such that for this value of b, $b + b'$ is a P-base of c. Let $T = \bigcup\{\gamma(U) \mid U \in \mathcal{U}\}$, where $\mathcal{U}$ is a family of disjoint sets of vertices U having $b(\gamma(U)) = k(|U| - 1)$ and $|U| > 1$. Form the graph G' by starting with G and contracting each set $U \in \mathcal{U}$. Let P' be the graphic polymatroid $P(kr')$ for r' the graphic polymatroid function of G'. It is an easy exercise to check that for any nonnegative vector b', $b + b' \in P$ iff $b' \in P'$. Thus we can find the desired b' by

running the prefix algorithm on P'. This algorithm is initialized as in the original prefix algorithm (e.g., b is 0).

The asymptotic time bound of Theorem 7.1 holds for this modified algorithm. The only numbers involved in the modified algorithm derive from the current value of k, a quotient of given numbers. Thus large numbers are avoided.

Acknowledgments

Thanks to Bill Cunningham for his help.

References

[AOST] R.K. Ahuja, J.B. Orlin, C. Stein and R.E. Tarjan, "Improved algorithms for bipartite network flow," *SIAM J. Comput.*, to appear.

[B] F. Barahona, "Separating from the dominant of the spanning tree polytope," *Op. Res. Letters, 12*, 1992, pp. 201–203.

[BT] S. Baum and L.E. Trotter, Jr., "Integer rounding for polymatroid and branching optimization problems," *SIAM J. Alg. Disc. Meth., 2,* 4, 1981, pp. 416–425.

[C85a] W.H. Cunningham, "Optimal attack and reinforcement of a network," *J. ACM, 32,* 3, 1985, pp. 549–561.

[C85b] W.H. Cunningham, "Minimum cuts, modular functions and matroid polyhedra," *Networks, 15,* 1985, pp. 205–215.

[CC] E. Cheng and W.H. Cunningham, "A faster algorithm for computing the strength of a network," *Inf. Proc. Letters, 49,* 1994, pp. 209–212.

[D] W. Dinkelbach, "On nonlinear fractional programming," *Management Sci., 13*, 1967, pp. 492–498.

[E65] J. Edmonds, "Lehman's switching game and a theorem of Tutte and Nash-Williams," *J. Res. National Bureau of Standards 69B*, 1965, pp. 73–77.

[E69] J. Edmonds, "Submodular functions, matroids, and certain polyhedra," *Calgary International Conf. on Combinatorial Structures and their Applications*, Gordon and Breach, New York, 1969, pp. 69–87.

[Ga91a] 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; *J. CSS*, to appear.

[Ga91b] H.N. Gabow, "Applications of a poset representation to edge connectivity and graph rigidity," *Proc. 32nd Annual Symp. on Found. of Comp. Sci.*, 1991, pp. 812–821; also Tech. Rept. CU-CS-545-91, Dept. of Comp. Sci., Univ. of Col. at Boulder, Boulder, CO, 1991.

[Ga94] H.N. Gabow, "Efficient splitting off algorithms for graphs," *Proc. 26th Annual ACM Symp. on Theory of Comp.*, 1994, pp. 696–705.

[Gi] F.R. Giles, "Submodular functions, graphs and integer polyhedra," Ph. D. Dissertation, Dept. of Combinatorics and Optimization, Univ. of Waterloo, Waterloo, Ontario, 1975.

[Gu83] D. Gusfield, "Connectivity and edge-disjoint spanning trees," *Inf. Proc. Letters, 16,* 2, 1983, pp. 87–89.

[Gu91] D. Gusfield, "Computing the strength of a graph," *SIAM J. Comput., 20,* 4, 1991, pp. 639–654.

[GGT] G. Gallo, M.D. Grigoriadis and R.E. Tarjan, "A fast parametric maximum flow algorithm and applications," *SIAM J. Comput., 18,* 1, 1989, pp. 30–55.

[GM] H.N. Gabow and K.S. Manu, "Algorithms for packing spanning trees and arborescences in multigraphs," in preparation.

[GT] A.V. Goldberg and R.E. Tarjan, "A new approach to the maximum flow-problem," *J. ACM, 35,* 4, 1988, pp. 921–940.

[GW] H.N. Gabow and H.H. Westermann, "Forests, frames and games: Algorithms for matroid sums and applications," *Algorithmica, 7,* 1992, pp. 465–497.

[H] F. Harary, *Graph Theory*, Addison-Wesley, Reading, Mass., 1969.

[HO] J. Hao and J.B. Orlin, "A faster algorithm for finding the minimum cut in a graph," *Proc. 3rd Annual ACM-SIAM Symp. on Disc. Algorithms,* 1992, pp. 165–174.

[NW] C. St. J. A. Nash-Williams, "Decomposition of finite graphs into forests," *J. London Math. Soc., 39,* 1964, p.12.

[PQ82a] Picard, J.-C. and M. Queyranne, "A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory," *Networks, 12,* 1982, pp. 141–159.

[PQ82b] Picard, J.-C. and M. Queyranne, "Selected applications of minimum cuts in networks," *INFOR 20,* 1982, pp. 394–422.

[PaW83] M.W. Padberg and L.A. Wolsey, "Trees and cuts," *Ann. Discrete Math., 17,* 1983, pp. 511–517.

[PaW84] M.W. Padberg and L.A. Wolsey, "Fractional covers for forests and matchings," *Math. Programming, 29,* 1984, pp. 1–14.

[PhW] S. Phillips and J. Westbrook, "Online load balancing and network flow," *Proc. 25th Annual ACM Symp. on Theory of Comp.,* 1993, pp. 402–411.

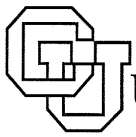
[S] S. Schaible, "Fractional programming II. On Dinkelbach's algorithm," *Management Sci., 22,* 1976, pp. 868–873.

[W] D.J.A. Welsh, *Matroid Theory,* Academic Press, New York, 1976.

ALGORITHMS FOR GRAPHIC POLYMATROIDS AND PARAMETRIC S-SETS

Harold N. Gabow

CU-CS-736-94



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE