Perfect Arborescence Packing in Preflow Minicut Graphs

Harold N. Gabow

CU-CS-790-95

University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR(S) AND DO NOT NECESSARILY REFLECT THE VIEWS OF THE AGENCIES NAMED IN THE ACKNOWLEDGMENTS SECTION.

Perfect Arborescence Packing in Preflow Mincut Graphs

Harold N. Gabow*

Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309

hal@cs.colorado.edu

Abstract. In a digraph with distinguished vertex a, for any vertex $v \neq a$ let $\lambda(v)$ equal the value of a maximum flow from a to v. A perfect packing of a-arborescences contains each vertex in $\lambda(v)$ arborescences and contains some fixed vertex in every arborescence. Determining if an arbitrary graph has a perfect packing is NP-complete. We present the most general known condition that guarantees the existence of a perfect packing: each vertex $v \neq a$ is separated from a by a set that has in-degree $\lambda(v)$ and out-degree no greater. This result is based on other useful properties of such graphs, e.g., they always have a pair of edges that can be "split off" preserving λ values. We show a perfect packing can be found in $O(nm^2)$ time, where n (m) is the number of vertices (edges). If the graph has a capacity function the time is the same as computing $O(n^2)$ maximum network flows. We also show a preflow mincut graph has a fractional perfect packing using only $m + n - 2$ distinct arborescences.

1 Introduction

In a digraph $G = (V, E)$ with distinguished vertex a, consider a collection of edge-disjoint a-arborescences (all notation and terminology is defined precisely at the end of this section). Obviously any vertex $v \neq a$ occurs in at most $\lambda(v)$ arborescences, where $\lambda(v)$ denotes the value of a maximum flow from a to v. In a perfect packing of a-arborescences, each $v \neq a$ occurs in $\lambda(v)$ arborescences, and some v occurs in every arborescence. When does G have a perfect packing?

Lovász showed even a graph with at most two edges directed to each vertex need not have a perfect packing [L76]. Edmonds proved that a perfect packing exists if all values of $\lambda(v)$ are equal [Ed]. (Edmonds' theorem is usually stated in an equivalent form: an arbitrary digraph having $\lambda(v) \geq k$ for all $v \neq a$ has k edge-disjoint spanning arborescences.) Recently Bang-Jensen, Frank and Jackson [BFJ] generalized Edmonds' theorem as follows. G has a perfect packing if every vertex v not achieving the maximum value of $\lambda(v)$ is "preflow," i.e., its in-degree is at least its out-degree, in symbols $\rho(v) \geq \delta(v)$. (The equivalent statement in [BFJ] is that for any digraph, if every vertex $v \neq a$ has $\lambda(v) \geq k$ or is preflow, then there are k edge-disjoint a-arborescences with each v in $\min\{k, \lambda(v)\}$ of them.) Bang-Jensen et.al. note some interesting special cases: They call G a "preflow digraph" if every $v \neq a$ has $\rho(v) \geq \delta(v)$. Examples of preflow digraphs are Eulerian digraphs and the digraph of any undirected graph (i.e., each edge occurs in both directions). Their theorem shows any preflow digraph has a perfect packing.

In this paper we first show that determining if an arbitrary graph has a perfect packing is NP-complete. This holds even for packing just two arborescences. The main contribution of this paper is to generalize the perfect packing result of [BFJ]. Call G a "preflow mincut graph" if every $v \neq a$

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

is separated from a by a set having in-degree exactly $\lambda(v)$ and out-degree no greater. These graphs have several other characterizations. For instance G is preflow mincut as long as every nonpreflow vertex v has the separating set just described. (This implies the graphs of the theorem of [BFJ] are preflow mincut.) A basic reason why preflow mincut graphs are useful is that the family of separating sets can be transformed to a laminar family. We show any preflow mincut graph has a perfect packing. (To get the result of Edmonds or Bang-Jensen et.al. make G preflow mincut by adding a new vertex a_0 with k edges from a_0 to a, for k the value of the theorem. Note the original set V separates any v from a_0 and has in-degree k and out-degree 0.)

We prove the packing theorem like [M] and [BFJ], by showing that a preflow mincut graph has a pair of edges that can be split off (splitting off is defined in Section 4). The splitting off technique was introduced by Mader [M] and has been shown a useful tool in connectivity augmentation [F92], graph orientation [F93] and others. We anticipate further applications of preflow mincut graphs for this reason.

We give several algorithms for perfect packing based on splitting off. Consider a preflow mincut graph G. Here and throughout the paper n and m denote the number of vertices and edges of the given graph, respectively. First suppose G is uncapacitated, i.e., parallel edges are allowed but each copy contributes to m. A perfect packing can be found in $O(nm^2)$ time and $O(nm)$ space, or alternatively $O(n^2 \min\{m^{1/2}, n\}m)$ time and $O(m)$ space. Next suppose G has an integral capacity function. Here and throughout the paper T_{MF} denotes the time to find a maximum flow on a network of n vertices, m edges and arbitrary capacities. King, Rao and Tarjan showed $T_{MF} = O(nm \log_b n)$ for $b = m/(n \log n) + 2$ [KRT]. Throughout this paper we assume (for notational convenience) $T_{MF} = \Omega(nm)$. A perfect packing can be found in $O(n^2 T_{MF})$ time and $O(n^3)$ space.

Our packing for capacitated graphs uses $O(n^2)$ distinct arborescences (each distinct arborescence has an associated integral multiplicity in the packing). To reduce this number we investigate a second approach to packing. Instead of splitting off, we grow each arborescence of the packing directly. Furthermore we allow the packing to be fractional, i.e., the multiplicity of an arborescence in the packing can be real-valued rather than just integral. We show a preflow mincut graph has a perfect fractional packing containing at most $m + n - 2$ distinct arborescences. This fractional packing can be found in time $O(n^3 T_{MF})$ and $O(nm)$ space. The space reduces to $O(m)$ if each arborescence can be output when it is found.

Our two approaches to packing use the same basic technique. To do the splitting or to grow the arborescence, in both the proofs and algorithms we maintain a laminar family of sets. We guess a split or arborescence that is "consistent" with this family. Then we check if the guess is valid or not. If valid, the split can be executed or the arborescence can be added to the packing. If invalid, we can enlarge the laminar family. Since a laminar family has $O(n)$ sets we do not guess wrong too many times.

In related work, Gabow and Manu present efficient packing algorithms for Edmonds' theorem [GM]. Here the problem is to find a packing of spanning a-arborescences with total multiplicity $\min\{\lambda(v)\}$. They find a fractional (integral) packing with at most m ($m +$

$n - 2)$ distinct arborescences. The time is $O(n^3 m \log(n^2/m))$ for fractional packing and $O(\min\{n, \log(nN)\} n^2 m \log(n^2/m))$ for integral packing (here the integer N is the largest capacity of an edge). We follow the approach of [GM] in using a laminar family.

To implement our packing algorithms efficiently we also develop a variant of binary search. The problem it solves – finding the minimum of n integers, each given by an oracle – seems natural, but we are unaware of a previous solution. Our algorithm combines n binary searches into one. This algorithm solves some other problems efficiently, e.g., computing the binding number of a graph [C].

The paper is organized as follows. Section 2 shows the NP-completeness of perfect packing. Section 3 defines preflow mincut graphs and gives their basic properties. Section 4 proves our splitting off theorem. Section 5 proves our perfect packing theorem and gives the packing algorithms based on splitting. Section 6 completes the implementation of our packing algorithms by presenting the binary search technique. The next four sections give our fractional packing algorithm. Sections 7 and 8 give the basic ideas. Section 9 gives our fractional packing algorithm. Section 10 completes the algorithm by giving a method to compute the capacity of an arborescence. The rest of this section gives notation and definitions.

Consider a universe V containing elements u, w and subsets U, W. $\overline{U}$ denotes the complement $V - U$. We use both set containment $U \subseteq W$ and proper set containment $U \subset W$. We often denote singleton sets by omitting set braces, e.g., $U \cup u$. A *$u\overline{w}$-set* contains u but not w; a u-set contains u; a *$\overline{w}$-set* is a nonempty set not containing w. (Section 4 uses an extension of this notation: A $U\overline{W}$-set is a subset of V that contains U and is disjoint from W.) Sets U and W are *intersecting* if $U \cap W$, $U - W$ and $W - U$ are all nonempty. A family of subsets of V is *laminar* if every two sets in the family are either disjoint or one contains the other, equivalently no two sets are intersecting. If $\mathcal{F}$ is a family of sets then $\cup\mathcal{F}$ denotes $\bigcup\{U : U \in \mathcal{F}\}$.

Consider a digraph $G = (V, E)$. If A is a set of edges $V(A)$ denotes the set of all vertices on these edges. A digraph can have parallel edges. In this case each copy of an edge is counted separately in m. A *capacity function* assigns a positive real value $c(e)$ to each edge. If an edge ever gets capacity zero we delete it from E. If a graph does not have an explicit capacity function we assume each edge has capacity one (if there are parallel edges, each copy of an edge adds an additional unit of capacity).

An edge uv *enters* any $v\overline{u}$-set and *leaves* any $u\overline{v}$-set. For a set of vertices W, the *in-degree* $\rho_G(W)$ (*out-degree* $\delta_G(W)$) equals the total capacity of all edges entering W (leaving W). Sections 7–10 use this convention: Suppose A is a set of edges in a capacitated graph. We assume there is no capacity function on A, so $\rho_A(W)$ denotes the number of edges entering W. To prevent confusion, in Sections 7–10 the set of edges is always named A when this convention is used.

We use two other degree functions: For $U, W \subseteq V$, $d_G(U, W)$ equals the total capacity of all edges having one end in $U - W$ and the other in $W - U$; $\overline{d}_G(U, W)$ equals the total capacity of all edges having one end in $U \cap W$ and the other in $\overline{U \cup W}$. Both notations ignore the directions of edges. In all these notations we drop the subscript G if it is clear from context.

A set of edges A in a digraph is an *arborescence* if any vertex v has $\rho_A(v) \le 1$, and ignoring edge directions A is a tree. A is an *a-arborescence* if a is the vertex of $V(A)$ having $\rho_A(v) = 0$ (a is unique unless $A = \emptyset$). A *spans* the set of vertices $V(A)$.

Consider a digraph $G = (V, E)$ with a distinguished vertex a and a real-valued capacity function c. Vertex a is called the *root* and any other vertex is a *nonroot*. For any nonroot v, $\lambda_G(v)$ (more succinctly $\lambda(v)$) equals the value of a maximum flow from a to v. For any set $S \subseteq V - a$ define $\Lambda(S) = \max\{\lambda(v) : v \in S\}$; also $\Lambda(G) = \Lambda(V - a)$. A *fractional packing (of a-arborescences)* is a family $\mathcal{A}$ of a-arborescences A, each with an associated positive real multiplicity $\alpha(A)$ such that for any edge e, $\sum\{\alpha(A) : e \in A \in \mathcal{A}\} \le c(e)$. An *integral packing* has all multiplicities $\alpha(A)$ integral. $\mathcal{A}$ is a *perfect packing (of a-arborescences)* if it is a packing where every nonroot v has $\sum\{\alpha(A) : v \in V(A) \in \mathcal{A}\} = \lambda(v)$ and $\sum\{\alpha(A) : A \in \mathcal{A}\} = \Lambda(G)$ (i.e., some fixed nonroot is in every arborescence).

2 NP-completeness of Perfect Packing

This section shows the general perfect packing problem is NP-complete. This holds even when every edge of G is in the packing, i.e., every vertex v has $\rho(v) = \lambda(v)$.

Theorem 2.1. *Determining if a digraph G with root a has a perfect integral packing is NP-complete. This holds even if $\Lambda(G) = 2$ and every nonroot v has $\rho(v) = \lambda(v)$.*

Proof. It is clear that the problem is in NP. We begin by showing perfect packing on graphs with $\Lambda(G) = 2$ is NP-hard. We reduce the satisfiability problem, as follows.

Consider a conjunctive normal form expression E consisting of clauses $C_j, j = 1, \ldots, m$ over variables $x_i, i = 1, \ldots, n$. We construct a digraph G that has a perfect packing if and only if E is satisfiable. The vertices of G are the root a, T, F, y_0; $x_i, \overline{x}_i, y_i$ for $i = 1, \ldots, n$ and C_j for $j = 1, \ldots, m$. The first group of edges of G uses the x-vertices to form two edge-disjoint paths through the y-vertices: edges aT, Ty_0, aF, Fy_0; $y_{i-1}x_i, x_iy_i, y_{i-1}\overline{x}_i, \overline{x}_iy_i$ for $i = 1, \ldots, n$. The second group of edges represents the clauses: Each clause vertex C_j has edges FC_j and zC_j for each literal z in the clause. All edges of G have capacity one.

Since $\delta(a) = 2$ each vertex has flow value at most 2. Each vertex y_i has $\lambda(y_i) = 2$ (using paths $aTy_0x_1y_1 \ldots x_iy_i$ and the "complementary" path $aFy_0\overline{x}_1y_1 \ldots \overline{x}_iy_i$). Each vertex x_i and $\overline{x}_i$ clearly has $\lambda(x_i) = \lambda(\overline{x}_i) = 1$. Each vertex C_j has $\lambda(C_j) = 2$ (using paths aFC_j and $aTy_0x_1y_1 \ldots x_{i-1}y_{i-1}zC_j$ for literal $z = x_i$ or $\overline{x}_i$ in C_j).

Suppose a truth assignment satisfies E. We construct a perfect packing of two arborescences. The first arborescence consists of the path from a to y_n that contains F and each false literal, plus edge FC_j for every clause C_j. The second arborescence consists of the path from a to y_n that contains T and each true literal, plus an edge zC_j for some true literal z in C_j.

Conversely suppose there is a perfect packing. Since $\lambda(y_i) = 2$ each arborescence contains y_i. This implies that one arborescence A contains a path from a to y_n that contains T, each y_i, and one vertex of each pair $x_i, \overline{x}_i$ (the other arborescence contains the "complementary" path). Make each literal in arborescence A true. Since A contains each clause C_j, it contains an edge zC_j for a literal z of C_j. Since z is true clause C_j is satisfied. This completes the reduction.

It remains to show that we can further restrict the problem so that each vertex has $\rho(v) = \lambda(v)$. This condition holds for all vertices of G except the clause vertices C_j. Consider a typical clause $C = C_j$ with edges z_iC, $i = 0, \ldots, k$ (so clause C contains k literals). Replace vertex C by new vertices $v_i, i = 0, \ldots, k$ and edges z_iv_i and v_iv_{i+1} (in this discussion we use arithmetic modulo $k+1$ in subscripts, e.g., $v_{k+1} \equiv v_0$). In the new graph G', $\rho(v_i) = 2$ and $\lambda(v_i) = 2$ (route a unit of flow from a to z_i to v_i and from a to z_{i-1} to v_{i-1} to v_i).

A perfect packing in G' implies expression E is satisfiable (the arborescence containing T contains an edge z_iv_i for each clause). Let us prove that conversely if E is satisfiable then G' has a perfect packing. Begin as above, constructing one arborescence A_T to contain a path from a to y_n using the true literals and the other arborescence A_F using the false literals. Consider a clause C of E. Each arborescence contains one or more vertices z_i adjacent to vertex C (in A_T, z_i is any true literal of C; in A_F, z_i is F or any false literal of C). For each arborescence $A = A_T, A_F$, for each edge z_iv_i of G' add edge z_iv_i to A if z_i is in A else add $v_{i-1}v_i$. This gives an arborescence since each vertex has in-degree at most one and there are no cycles. It is easy to see this is a perfect packing. $\square$

3 Preflow Mincut Graphs

This section gives some basic properties of preflow mincut graphs. In particular the preflow mincuts can be chosen to form a laminar family.

Throughout this paper we consider a digraph G with root vertex a; "cut" denotes a set of nonroot vertices (sometimes called an "a-cut"). An *x-mincut* (synonymously *mincut for* x) is an $x\overline{a}$-set X with $\rho(X) = \lambda(x)$. A *mincut* is a mincut for some vertex x. $X \subseteq V$ is a *preflow set* if $\rho(X) \ge \delta(X)$; if $X = \{x\}$, x is a *preflow vertex*. G is a *preflow mincut graph* if every vertex except a has a preflow mincut.

We start with a tool for uncrossing mincuts. Consider two intersecting mincuts U and W. The sets are *U-intersecting* if U is a mincut for a vertex of $U \cap W$; *central intersecting* means U-intersecting or W-intersecting. The sets are *polar intersecting* if U is a mincut for a vertex of $U - W$ and W is a mincut for a vertex of $W - U$. Any two intersecting mincuts are central or polar intersecting (or both).

Lemma 3.1. *Let U and W be intersecting mincuts in an arbitrary graph.*

(i) If U and W are central intersecting then $U \cap W$ and $U \cup W$ are mincuts and $d(U, W) = 0$.

(ii) If U and W are polar intersecting and $U \cap W$ is a preflow set then $U - W$ and $W - U$ are mincuts, $\overline{d}(U,W) = 0$ and $\rho(U \cap W) = \delta(U \cap W)$.

Before proving this result note a useful principle: A mincut U has in-degree $\Lambda(U)$. This makes it easy to determine the in-degree of the sets proved to be mincuts in the lemma: In part (i) if the mincuts are W-intersecting then (since $\Lambda(W) \le \Lambda(U)$) $\rho(U \cup W) = \Lambda(U)$ and $\rho(U \cap W) = \Lambda(W)$. In part (ii), $\rho(U - W) = \Lambda(U)$ and similarly for $W - U$.

Proof. (i) Let the mincuts be W-intersecting. Consider the identity $\rho(U) + \rho(W) = \rho(U \cap W) + \rho(U \cup W) + d(U,W)$ (which holds for arbitrary sets U, W in any digraph). Since U is a mincut, $\rho(U \cup W) \ge \rho(U)$. Since the sets are W-intersecting, $\rho(U \cap W) \ge \rho(W)$. Combining these relations shows that equality holds in both inequalities and $d(U,W) = 0$.

(ii) Consider the identity $\rho(U)+\rho(W) = \rho(U-W)+\rho(W-U)+\overline{d}(U,W)+\rho(U\cap W)-\delta(U\cap W)$ (it holds for arbitrary sets U, W in any digraph [BFJ]). The hypotheses imply $\rho(U - W) \ge \rho(U)$, $\rho(W - U) \ge \rho(W)$ and $\rho(U \cap W) \ge \delta(U \cap W)$. Combining these relations shows that equality holds in these three inequalities and furthermore $\overline{d}(U,W) = 0$. □

We extend this to a tool for uncrossing preflow mincuts.

Corollary 3.2. *Let U and W be intersecting mincuts in an arbitrary graph.*

(i) If U and W are central intersecting and both are preflow sets then $U \cap W$ or $U \cup W$ is a preflow mincut.

(ii) If U and W are polar intersecting and both U and $U \cap W$ are preflow sets then $U - W$ is a preflow mincut.

Proof. Recall that $\Delta(U) \equiv \rho(U) - \delta(U)$ is a modular function, i.e., the identity $\Delta(U) + \Delta(W) = \Delta(U \cap W) + \Delta(U \cup W)$ holds for any two sets U and W.

(i) Since U and W are preflow sets, the left-hand side of the above modular identity is nonnegative. This implies at least one of the sets $U \cap W$, $U \cup W$ is preflow.

(ii) By modularity $\Delta(U) = \Delta(U - W) + \Delta(U \cap W)$. Since U is preflow the left-hand side is nonnegative. Lemma 3.1(ii) shows $\Delta(U \cap W) = 0$. Thus $U - W$ is preflow. □

Now we take the major step towards characterizing preflow mincut graphs. For any set of nonroot vertices N, a *covering family for N* is a laminar family of preflow mincuts, one for each vertex of N (vertices can share the same preflow mincut). A *covering family* is a covering family for $V - a$.

Lemma 3.3. *Let N be a set of nonroots containing every nonroot that is nonpreflow. If G has a preflow mincut for every vertex of N then G has a covering family for N.*

Proof. Consider the following algorithm. We will show it grows a laminar family $\mathcal{L}$ to eventually become the desired covering family. Recall the notation $\cup\mathcal{L}$ defined in Section 1.

Initialize $\mathcal{F}$ to a minimal family containing a preflow mincut for each vertex of N. Initialize $\mathcal{L}$ to $\emptyset$.

```
**while** $N \not\subseteq \cup\mathcal{L}$ **do begin**
    $U \leftarrow$ a minimal set of $\mathcal{F}$ having minimum in-degree $\rho(U)$; $U_0 \leftarrow U$;
    **while** some $W \in \mathcal{F}$ is intersecting with $U$ **do**
        **if** $N \cap U \cap W \subseteq \cup\mathcal{L}$ **then** $U \leftarrow U - W$
        **else if** $U \cap W$ is a preflow set **then** $U \leftarrow U \cap W$
        **else** replace $W$ by $U \cup W$ in $\mathcal{F}$;
    add $U$ to $\mathcal{L}$;
    **if** every vertex of $U_0 \cap N - \cup\mathcal{L}$ has a mincut in $\mathcal{F} - U_0$ **then** delete $U_0$ from $\mathcal{F}$; **end**;
```

We prove the algorithm finds the desired family $\mathcal{L}$ by establishing two invariants.

The first invariant has two conditions: first, $\mathcal{L}$ is laminar; second, any $L \in \mathcal{L}$ and $X \in \mathcal{F} \cup \{U\}$ are either disjoint or $L \subseteq X$. We begin by showing that the second condition is preserved in the loop that modifies $\mathcal{F}$ and U (this loop does not change $\mathcal{L}$ so obviously the first condition is preserved): The second condition implies for any $L \in \mathcal{L}$ and $X, Y \in \mathcal{F} \cup \{U\}$, L is either disjoint from $X \cup Y$ or contained in $X - Y$ or $Y - X$ or $X \cap Y$. Each set added to $\mathcal{F}$ and each new set U has one of these four forms. Thus the second condition is preserved.

Next we show the invariant is preserved when we add U to $\mathcal{L}$. The second condition implies $\mathcal{L}$ remains laminar. So we need only check the second condition still holds. When U is chosen its minimality ensures no set of $\mathcal{F}$ is contained in U. We now show that the loop ensures U is either disjoint from or contained in any set of $\mathcal{F}$. Each iteration of the loop makes U disjoint from or contained in one more set of $\mathcal{F}$. These relations are preserved in later iterations since whenever U is changed it is replaced by a subset. Thus the loop halts (after at most $|N|$ iterations) with U disjoint from or contained in any set of $\mathcal{F}$. Now we have established that the first invariant always holds.

The second invariant consists of conditions for $\mathcal{L}$, $\mathcal{F}$ and U: $\mathcal{L}$ is a covering family for $N \cap (\cup\mathcal{L})$. (This condition, once established, implies the algorithm works as desired.) $\mathcal{F} \cup \{U\}$ consists of preflow mincuts, at least one for each vertex of $N - \cup\mathcal{L}$, and $\mathcal{F}$ is a minimal such family each time we choose a new set U. U contains a vertex of $N - \cup\mathcal{L}$ and is a preflow mincut for all such vertices. Now we prove these conditions are maintained.

Consider the initially chosen set U. It contains a vertex of $N - \cup\mathcal{L}$ since $\mathcal{F}$ is minimal. Similarly U is preflow by the definition of $\mathcal{F}$. U is also a mincut for any vertex in $U \cap N - \cup\mathcal{L}$. (Any $u \in N - \cup\mathcal{L}$ has $\lambda(u) \geq \rho(U)$ by the choice of U. Any $u \in U$ has $\lambda(u) \leq \rho(U)$.)

Now we check that each time the loop modifies a set the invariants for $\mathcal{F}$ and U are preserved. This implies that when U is added to $\mathcal{L}$, the invariant on $\mathcal{L}$ is preserved.

Suppose in the initial test, $N \cap U \cap W \subseteq \cup\mathcal{L}$. We claim U and W are polar intersecting and satisfy the hypotheses of Corollary 3.2(ii). The test along with the definition of U shows $U - W$ contains a vertex of $N - \cup\mathcal{L}$ and U is a preflow mincut for all such vertices. W is a mincut for some vertex in $W - U$ (since $\mathcal{F}$ is minimal). The last hypothesis is that $U \cap W$ is a preflow set. The test implies that each vertex of $N \cap U \cap W$ is contained in a set of $\mathcal{L}$, which is contained in $U \cap W$ (by the first invariant). Since $\mathcal{L}$ is laminar $U \cap W$ can be written as the disjoint union of preflow sets (in $\mathcal{L}$) and preflow vertices (not in $\cup\mathcal{L}$). Thus $U \cap W$ is preflow. Corollary 3.2(ii) now shows that $U - W$ is a preflow mincut. The remaining invariants on U and $\mathcal{F}$ are clear.

Next suppose the initial test fails. Since U is a mincut for any vertex in $U \cap N - \cup\mathcal{L}$, U and W are U-intersecting. Corollary 3.2(i) shows $U \cap W$ or $U \cup W$ is a preflow mincut. It is easy to see the algorithm processes this preflow set to preserve the invariants.

The last line of the algorithm makes $\mathcal{F}$ minimal (clearly the only set that can possibly be removed from $\mathcal{F}$ after U is added to $\mathcal{L}$ is U_0). □

To characterize the preflow mincut graphs we need one more observation. Consider a graph where every nonpreflow vertex has a preflow mincut. Take any nonroot u (preflow or not). We claim the maximum u-mincut U is a preflow set. In proof let $\mathcal{F}$ be a covering family for the nonpreflow vertices (by the lemma). Each nonpreflow vertex $x \in U$ has a preflow mincut $X \in \mathcal{F}$. $X \subseteq U$ by Lemma 3.1(i). Thus U is the disjoint union of preflow sets (in $\mathcal{F}$) and preflow vertices, so U is a preflow set. Now we have the following characterization.

Theorem 3.4. *G is a preflow mincut graph if and only if every nonroot (every nonpreflow nonroot) has a preflow mincut if and only if for every nonroot (every nonpreflow nonroot) the maximal mincut is preflow. Any preflow mincut graph has a covering family.* □

Examples show that in general neither the minimal preflow mincuts nor the maximal preflow mincuts form a laminar family.

The theorem allows us to check if a given digraph is preflow mincut, and if so construct a covering family, in time $O(nT_{MF})$. First we find a maximal mincut for each nonroot. Then we execute uncrossing procedure of Lemma 3.3. The time for the two steps is $O(nT_{MF})$ and $O(n^2m)$ respectively, which implies the time bound. (If all edges have capacity one a maximum flow can be found in time $O(nm)$ [Ev] so the time is $O(n^2m)$.)

We give one more property of preflow mincut graphs that is useful for uncrossing. We first extend the notion of central intersecting. Suppose U is a mincut and $\mathcal{W}$ is a family of disjoint mincuts, each one intersecting with U. U is *central intersecting with the family* $\mathcal{W}$ if each $W \in \mathcal{W}$ is W-intersecting with U. Let $Z = U \bigcup \cup\mathcal{W}$. It is easy to see that Z is a mincut. However we will encounter families that do not quite satisfy this definition. U is *near-central intersecting with* $\mathcal{W}$ if each $W \in \mathcal{W}$ is central intersecting with U and in fact with at most one exception, is W-intersecting with U. Thus $\mathcal{W}$ consists of a central intersecting family and perhaps one other set that is U-intersecting. If this U-intersecting set exists we denote it as W_u, where U is a u-mincut and $u \in W_u$. If there is no exception then W_u is $\emptyset$. Let $Z = U \bigcup \cup \mathcal{W}$.

Lemma 3.5. *In an arbitrary graph, suppose U is near-central intersecting with $\mathcal{W}$. Then Z is a mincut. Any two sets $W, W' \in \mathcal{W}$ have $d(W - U, W' - U) = 0$.*

Note that $\rho(Z) = \max\{\rho(U), \rho(W_u)\}$. If U is central intersecting with $\mathcal{W}$ then $\rho(Z) = \rho(U)$.

Proof. Every set $U \cup W$, $W \in \mathcal{W}$ is a mincut since U and W are central intersecting. Furthermore $U \cup W$ is a mincut for a vertex of U if $W \neq W_u$. Consider any $W \in \mathcal{W} - W_u$ and a set X disjoint from W having $U \cup X$ a mincut. Lemma 3.1(i) applies to $U \cup X$ and $U \cup W$ (these sets are central intersecting by the choice of W). This shows $U \cup X \cup W$ is a mincut, with $d(U \cup W, U \cup X) = d(W - U, X - U) = 0$. Applying this observation repeatedly (choosing W_u as the first set X) proves the two relations of the lemma. □

Examples show that Z is not necessarily a mincut if we allow two exceptions in the notion of near-central intersecting.

Now let $\mathcal{F}$ be a laminar family. For any set R define $\mathcal{F}(R)$ to be the family of maximal sets in $\{F :$ for some $r \in R$, F is the smallest r-set of $\mathcal{F}\}$. By definition $\mathcal{F}(R)$ is a family of disjoint sets. In a preflow mincut graph let $\mathcal{P}$ be the laminar family formed by a covering family for the nonpreflow nonroots and the singleton sets $\{v\}$, v a preflow nonroot. For any $\overline{a}$-set R, the set $S = \cup\mathcal{P}(R)$ is a preflow set containing R with $\Lambda(S) = \Lambda(R)$.

Lemma 3.6. *In a preflow mincut graph with U a u-mincut and R any subset of U, $U \bigcup \cup\mathcal{P}(R)$ is a u-mincut.*

Proof. Observe that $U \bigcup \cup\mathcal{P}(R)$ is formed from U and a central intersecting family of mincuts. □

The lemma allows us to get versions of the uncrossing Lemma 3.1(ii) that drop the restriction that $U \cap W$ is preflow. To do this we apply the lemma to both U and W with $R = U \cap W$. Lemmas 4.1 and 8.1 are examples.

Preflow mincut graphs have a closure property that is useful for perfect packing. Lovász showed any digraph G with root vertex a has a subgraph H such that every nonroot v satisfies $\lambda_G(v) = \lambda_H(v) = \rho_H(v)$ [L73]. If G is preflow mincut, so is any such H. (In proof, if U is a preflow mincut for u in G then $\rho_H(U) = \rho_G(U) \geq \delta_G(U) \geq \delta_H(U)$.) Thus to find a perfect packing of G we can work on the smaller graph H. Furthermore a perfect packing of H uses every edge of H, so H is the smallest graph we can work with. Lovász showed H can be constructed in time $O(nT_{MF})$ [L73]. The graphs for the perfect packing theorem of [BFJ] do not have this closure property.

4 Splitting Off

This section proves our splitting theorem for preflow mincut graphs. A pair of edges rs, st is *splittable* if replacing these two edges by the single edge rt does not change any value $\lambda(v), v \neq s, a$. We will show that in a preflow mincut graph for any preflow vertex $s \neq a$ and any edge st, there is an edge rs splittable with st.

Throughout this section G is a preflow mincut graph. Furthermore without loss of generality assume $\rho(a) = 0$. This assumption is just for notational convenience – it implies any preflow vertex is a nonroot.

We begin with an uncrossing lemma for splitting. Fix an arbitrary preflow vertex s. U is a *proper mincut (w.r.t. s)* if it is a u-mincut for some $u \neq s$. (We shall see that proper mincuts are the key concept for splitting off at s.) As before let $\mathcal{P}$ be the laminar family formed by a covering family for the nonpreflow nonroots and the singletons $\{v\}$, v a preflow vertex.

Lemma 4.1. *Let U and W be intersecting proper mincuts. Then at least one of these possibilities holds:*

(i) $U \cup W$ and $U \cap W$ are proper mincuts with $d(U, W) = 0$.

(ii) Let $Z = \cup\mathcal{P}(U \cap W)$. $U - Z$ and $W - Z$ are proper mincuts. For any set $X \in \mathcal{P}(U \cap W)$, $\overline{d}(X, X \cup U \cap W \cup U - Z \cup W - Z) = 0$.

Before proving this result consider part (*ii*). First note that Z does not contain U or W ($\emptyset$ is not a mincut). Next consider the equation of part (*ii*). In this and all other set expressions of this paper assume $\cup$ has less precedence than $-$ or $\cap$. The equation means that any edge with one end in X has the other end in the second argument of $\overline{d}$, $X \cup U \cap W \cup U - Z \cup W - Z$. Although the equation $\overline{d}(U, W) = 0$ of Lemma 3.1(*ii*) need not hold, we get the following relaxation: Any edge with one end in both sets U, W and the other end in neither set has both ends in the same set of $\mathcal{P}(U \cap W)$.

Proof. The first case is when $U \cap W$ contains a vertex distinct from s for which U or W is a mincut. It is easy to see that (*i*) follows from Lemma 3.1(*i*).

Suppose the first case does not apply. Let U be a u-mincut and W a w-mincut, $u, w \neq s$. Thus $U \cup Z$ is a u-mincut and $W \cup Z$ is a w-mincut (Lemma 3.6). These two sets intersect in Z (by definition $U \cap W \subseteq Z$) which is a preflow set (by definition). These two sets are polar intersecting ($u, w \notin Z$ since $\Lambda(Z - s) = \Lambda(U \cap W - s) < \min\{\lambda(u), \lambda(w)\}$, by the definitions of Z and the second case).

We have shown that Lemma 3.1(*ii*) applies to sets $U \cup Z$ and $W \cup Z$. Clearly $U - Z$ and $W - Z$ are proper mincuts.

For the last assertion of part (*ii*) consider a set $X \in \mathcal{P}(U \cap W)$. Take any edge xy or yx with $x \in X$. Lemma 3.1(*ii*) shows $\overline{d}(U \cup Z, W \cup Z) = 0$, so $y \in U \cup W \cup Z$. Thus it suffices to show that

$y \in Z$ implies $y \in X \cup U \cap W$. Assume for the sake of contradiction that $y \in Z - (U \cap W) - X$. Thus some mincut $Y \in \mathcal{P}(U \cap W)$ contains y. Also $y \notin X$ and $x \notin Y$ (recall $\mathcal{P}(U \cap W)$ is a family of disjoint sets). We can assume without loss of generality that $y \notin U$ (since $y \notin U \cap W$). Y and U are central intersecting mincuts. Now edge xy contradicts the relation $d(U, Y) = 0$ of Lemma 3.1(i) if $x \in U$ or $d(X - U, Y - U) = 0$ of Lemma 3.5 if $x \notin U$. □

Now we will prove the splitting theorem. Fix a preflow vertex s and an edge st. Consider any edge rs. Define $T = \{r, t\}$. An $\overline{a}$-set U is *consistent* as long as U is not a $T\overline{s}$-set or an $s\overline{T}$-set (the notation $T\overline{s}$-set is defined in Section 1). If we split off rs, st (i.e., replace them by rt) then for any $\overline{a}$-set U the quantities $\rho(U)$ and $\delta(U)$ change only if U is inconsistent, in which case both quantities decrease by one. Thus rs, st is splittable if and only if every proper mincut is consistent.

We use some additional terminology related to consistency whose meaning is clear. For instance we say the pair rs, st makes U consistent, or since st is fixed in our discussion, rs makes U consistent. A family of sets is consistent if each of its sets is.

To show the desired edge rs exists we will use a laminar family $\mathcal{F}$. Fix $\mathcal{F}$ as a laminar family of proper mincuts that contains a preflow mincut for each nonpreflow nonroot. ($\mathcal{F}$ exists by Theorem 3.4). For notational convenience we assume that $V - a$ is a proper mincut and $V - a \in \mathcal{F}$. (If $V - a$ is not a proper mincut then add a new vertex a_0 to G with $\Lambda(G)$ edges from a_0 to a. In the new graph $V - a_0$ is an a-mincut that is also proper.) The proof is in two main steps. The first step shows there is an edge rs that makes $\mathcal{F}$ consistent. The second step shows that if this edge rs is not splittable with st then $\mathcal{F}$ can be enlarged.

We begin by showing there is an edge rs that makes a given mincut consistent.

Lemma 4.2. *For any proper mincut U there is an edge rs that makes U consistent.*

Proof. If U contains 0 or 2 of the vertices s, t then any edge rs makes U consistent. An edge rs exists since s is preflow.

Suppose U is a $t\overline{s}$-set. Since U is a mincut, $\rho(U \cup s) \geq \rho(U)$. Since edge st exists, the inequality shows there is an edge rs with $r \notin U$. This edge makes U consistent.

Suppose U is an $s\overline{t}$-set. Since U is a mincut for a vertex distinct from s, $\rho(U - s) \geq \rho(U)$. Let H be the subgraph consisting of the edges directed from s to $U - s$ or from $\overline{U}$ to s. The previous inequality means that $\delta_H(s) \geq \rho_H(s)$. Edge st shows $\delta(s) > \delta_H(s)$. Thus since s is preflow, $\rho(s) > \rho_H(s)$. This inequality means there is an edge rs with $r \in U$. This edge makes U consistent. □

Now we accomplish the first step of the proof: we show there is an edge rs that makes $\mathcal{F}$ consistent. Let S be the smallest set of $\mathcal{F}$ properly containing s. If $t \notin S$ then using the lemma, choose edge rs to make S consistent, i.e., $r \in S$. It is easy to see that rs makes $\mathcal{F}$ consistent. If t belongs to S but no set of $\mathcal{F}$ properly contained in S then any edge rs makes $\mathcal{F}$ consistent. In the remaining case let W be the largest set of $\mathcal{F}$ that contains t and is properly contained in S. Choose rs to make W consistent, i.e., $r \notin W$. This makes $\mathcal{F}$ consistent.

For the second step of the proof, assume the pair rs, st makes $\mathcal{F}$ consistent but is unsplittable. We will show $\mathcal{F}$ can be enlarged. Let U be a minimal inconsistent proper mincut contained a set of $\mathcal{F}$ of smallest possible size. Let $\mathcal{W}$ be the family of maximal sets of $\mathcal{F}$ that are intersecting with U. Add $X = U \bigcup \cup\mathcal{W}$ to $\mathcal{F}$. We call this operation *uncrossing* $\mathcal{F} \cup \{U\}$. Our goal is to show that this uncrossing operation preserves the definition of $\mathcal{F}$ and actually enlarges $\mathcal{F}$. It is easy to see that $\mathcal{F} \cup \{X\}$ is laminar. Thus we must show X is a proper mincut not in $\mathcal{F}$. We do this by showing X is an inconsistent proper mincut.

We begin with an observation about Lemma 4.1(ii): Suppose the above set U is intersecting with another proper mincut W and Lemma 4.1(ii) applies to these sets. If a set of $\mathcal{P}(U \cap W)$ contains s it is $\{s\}$. To prove this suppose for the sake of contradiction that $X \in \mathcal{P}(U \cap W)$ is an x-mincut containing s, with $x \in U \cap W - s$. X does not contain U (since in Lemma 4.1(ii) Z does not contain U). $X \cap U$ is a proper mincut. (To prove this examine the two cases, $X \subseteq U$ and X (central) intersecting with U.) If U is an $s\overline{T}$-set then so is $X \cap U$, contradicting the minimality of U. If U is a $T\overline{s}$-set, then U is intersecting with X, so $d(X, U) = 0$. Thus $T \subseteq X \cap U$. Now $X \cap U$ is a $T\overline{s}$-set, again contradicting the minimality of U.

Now we analyze a set $W \in \mathcal{W}$.

Lemma 4.3. *Let U be intersecting with a set $W \in \mathcal{F}$.*

(i) $U \cup W$ is a proper mincut.

(ii) $U \cup W$ and U are both $T\overline{s}$-sets or both $s\overline{T}$-sets.

Proof. Lemma 4.1 applies to U and W. We begin by showing that Lemma 4.1(ii) leads to a contradiction.

Fix e as edge rs or st. We claim no end of e is in $U \cap Z$. Suppose the contrary. No set of $\mathcal{P}(U \cap W)$ contains both ends of e (by the above observation). Also U contains precisely one end of e. Now Lemma 4.1(ii) shows the other end of e is in $W - Z$. $W - Z$ is a proper mincut. If U is an $s\overline{T}$-set ($T\overline{s}$-set) then $W - Z$ is a $T\overline{s}$-set ($s\overline{T}$-set). Now $W - Z$ violates the choice of U (it is contained in a smaller set of $\mathcal{F}$ than U is). This contradiction proves the claim.

We have shown that $r, s, t \notin U \cap Z$. Thus $U - Z$ is a proper mincut and an $s\overline{T}$-set ($T\overline{s}$-set) if U is. This violates the minimality of U.

We conclude that Lemma 4.1(i) applies to U and W. Thus $U \cup W$ is a proper mincut, giving part (i) of the lemma. To show part (ii) note that $U \cap W$ is a proper mincut. If U is a $T\overline{s}$-set the minimality of U (with the proper mincut $U \cap W$) shows at least one vertex of T is in $U - W$, say vertex t. Now $d(U, W) = 0$ shows $s \notin W$. If U is an $s\overline{T}$-set a similar argument shows T is disjoint from W. $\square$

Now consider an uncrossing operation that adds $X = U \bigcup \cup \mathcal{W}$ to $\mathcal{F}$. The lemma implies that X and U are both $T\overline{s}$-sets or both $s\overline{T}$-sets. Hence X is inconsistent. To complete the second step of the proof we need only show that X is a proper mincut.

Let $\mathcal{W}$ consist of sets $W_i, i = 1, \ldots, k$, $k \geq 0$. Write $U_i = U \cup \bigcup_{j=1}^{i} W_j$ for $i \geq 0$. We prove U_i is a proper mincut by induction. The case $i = 0$ is trivial and the lemma gives the case $i = 1$. For the inductive step assume $i > 1$. U_{i-1} and $U \cup W_i$ are intersecting proper mincuts. We need only show that Lemma 4.1(ii) does not apply to these two sets. Lemma 4.1(ii) implies that an edge with one end in Z has the other end in $U \cup W \cup Z$. We will show our sets contradict this relation.

Towards this end note that the intersection of our sets is $U_{i-1} \cap (U \cup W_i) = U$. Thus $Z = \cup \mathcal{P}(U)$. Assume that the family $\mathcal{P}$ of Lemma 4.1(ii) is constructed from sets of $\mathcal{F}$ (i.e., $\mathcal{F}$ contains a covering family for the nonpreflow nonroots, so use these to form $\mathcal{P}$). We claim $Z \subseteq X$. In proof consider any set $Y \in \mathcal{P}(U)$. Either $Y \subseteq U$, Y is intersecting with U, or $U \subseteq Y$. The second alternative implies Y is contained in some set of $\mathcal{W}$. Thus it suffices to show the third alternative never occurs. Suppose $U \subseteq Y$. This implies $W_i \subseteq Y$ by the laminarity of $\mathcal{F}$. Thus $U \cup W_i \subseteq Y \subseteq Z$, contradicting Lemma 4.1(ii). This proves the claim.

The claim implies $U_{i-1} \cup (U \cup W_i) \cup Z \subseteq X$. Edges rs and st both join U and $\overline{X}$, which implies they join Z and the complement of $U_{i-1} \cup (U \cup W_i) \cup Z$. This is the desired contradiction of Lemma 4.1(ii). This completes the second step of the proof.

Theorem 4.4. *Let G be a preflow mincut graph. Take any preflow vertex s and any edge st. There is an edge rs forming a splittable pair rs, st.*

Proof. Initialize $\mathcal{F}$ to a covering family for the nonpreflow nonroots. Repeatedly choose an edge rs that makes $\mathcal{F}$ consistent and if rs, st is unsplittable then enlarge $\mathcal{F}$. This can be done by the above discussion. Since $|\mathcal{F}|$ is finite we eventually choose an edge rs that gives a splittable pair. □

Our splitting theorem, like others, can be iterated. Specifically we can reduce a preflow mincut graph to the graph consisting of just the root vertex a by the following process: Repeatedly choose a preflow vertex s; split off splittable pairs of edges incident to s until $\delta(s) = 0$; delete s. The sequence of splits produced by this process is a *split sequence.* When discussing a split sequence and its execution G always denotes the current graph (so G starts as the given graph and gets reduced to the trivial graph).

To see this split sequence actually exists first observe that a preflow vertex always exists ($\Delta(V - a) > 0$, for Δ defined in Corollary 3.2). Splitting off a splittable pair of edges does not change the in-degree or out-degree of any proper mincut. Thus G remains preflow mincut, and we can split off edges incident to s until no edge leaves s. When $\delta(s) = 0$ no edge entering s enters a proper mincut. Thus deleting s preserves λ values. Furthermore it can only decrease the out-degree of a proper mincut. Thus G remains preflow mincut. We conclude that the process can be repeated until G consists of only vertex a.

A *nondecreasing split sequence* is a split sequence where s is always chosen to have the (globally) minimum value $\lambda(s)$. We now show that a preflow mincut graph always has such a sequence. It suffices to show that the minimum value $\lambda(v)$ is achieved by a preflow vertex. In proof if a nonpreflow vertex v achieves the minimum consider a preflow mincut U for v. U contains a preflow vertex u since $\Delta(U) \geq 0$. Furthermore $\lambda(u) = \lambda(v)$ (since $\lambda(u) \geq \lambda(v) = \Lambda(U)$). Thus u is the desired vertex.

Bang-Jensen et.al. [BFJ] showed nondecreasing split sequences are particularly useful. They are the basis of the arborescence packing algorithm of the next section. We close this section by bounding the length of such a sequence. This bound is the basis for the efficient splitting off algorithms of next section. Let us first review how we construct a nondecreasing split sequence.

Let G be a preflow mincut graph (as usual parallel edges are allowed). Follow the procedure for constructing a split sequence, with these additions concerning $\mathcal{F}$. Initially $\mathcal{F}$ is a covering family for the nonpreflow nonroots. When choosing a new vertex s for splitting off, s is a preflow vertex with the minimum value $\lambda(s)$. At this point delete $\{s\}$ from $\mathcal{F}$ if it belongs to $\mathcal{F}$. In splitting off s, choose each pair rs, st to be an arbitrary pair that makes $\mathcal{F}$ consistent. If the pair is splittable, split it off. If the pair is unsplittable then enlarge $\mathcal{F}$ by the process described above.

Let us check that this procedure maintains $\mathcal{F}$ as a laminar family of proper mincuts containing a preflow mincut for each nonpreflow nonroot. First observe that when we choose a new vertex s, any mincut in G is proper except $\{s\}$. For suppose U is an s-mincut distinct from $\{s\}$. Any vertex $u \in U - s$ has $\lambda(u) \geq \lambda(s) = \Lambda(U)$. Thus U is a u-mincut.

This observation shows that $\mathcal{F}$ remains a family of proper mincuts after a new vertex s is chosen. We have already verified that $\mathcal{F}$ continues to satisfy its definition when we do a split, when we enlarge $\mathcal{F}$, and when we delete vertex s (note that deleting s cannot create a nonpreflow vertex). The sequence of pairs (both splittable and unsplittable) produced by this process is a *nondecreasing split supersequence*. Deleting the unsplittable pairs gives a nondecreasing split sequence.

Theorem 4.5. *In a preflow mincut graph, a nondecreasing split supersequence has $O(n)$ unsplittable pairs.*

Proof. When s is chosen for splitting $|\mathcal{F}|$ can decrease by 1. When s is deleted it is possible that $|\mathcal{F}|$ decreases: if $U, U - s \in \mathcal{F}$ then these two sets become identical. However note that such a U must be the smallest set of $\mathcal{F}$ properly containing s. Hence when s is deleted $|\mathcal{F}|$ decreases by ≤ 1. In summary each vertex s decreases $|\mathcal{F}|$ by ≤ 2, so over the whole sequence $|\mathcal{F}|$ decreases a total of $O(n)$.

Each unsplittable pair enlarges $|\mathcal{F}|$ by 1. But $|\mathcal{F}|$ is $O(n)$. This implies there are $O(n)$ unsplittable pairs. □

The theorem also facilitates splitting off in capacitated graphs, which we now discuss. Let G be a preflow mincut graph with an integral edge capacity function c. Define the *capacity* of a pair of edges rs, st as the largest integral value γ such that reducing $c(rs)$ and $c(st)$ both by γ and increasing $c(rt)$ by γ preserves all values $\lambda(v)$, $v \neq s$. In effect we split off γ pairs rs, st. Clearly $0 \leq \gamma \leq \min\{c(rs), c(st)\}$, e.g., $\gamma = 0$ if the edges are unsplittable. If $\gamma = \min\{c(rs), c(st)\}$ the split is *voiding.*

In a capacitated graph define a *split sequence* the same as in a graph except that when splitting off edges rs, st we do the split according to its capacity, as described above. We make the same convention in a *nondecreasing split supersequence.* Furthermore we treat nonvoiding splits as follows. Suppose rs, st is a nonvoiding split. After splitting this pair off to its capacity rs, st is an unsplittable pair. Thus we enlarge $\mathcal{F}$ for it. (The original pair rs, st may have capacity 0, in which case we simply enlarge $\mathcal{F}$ for it.) Theorem 4.5 shows that in a capacitated graph, a nondecreasing split supersequence has $O(n)$ nonvoiding splits.

Another consequence of the theorem concerns split sequences that are constructed without regard to a laminar family: In a capacitated graph any nondecreasing split sequence where each split has positive capacity has $O(n)$ nonvoiding splits. This follows since our procedure to construct a nondecreasing split supersequence can construct any such sequence.

Now observe that in a split sequence for a capacitated graph, each graph formed contains $O(m)$ distinct edges. This follows since a voiding split preserves the number of distinct edges, while a nonvoiding split increases the number by one. (If the edge rt introduced by the split already exists in G then these estimates decrease by 1, e.g., a voiding split decreases the number of edges by 1. However in general we cannot count on rt already existing.) Thus the theorem shows each graph formed actually has $m + O(n)$ edges.

In summary Theorem 4.5 has a two-fold payoff for computing a split sequence in a capacitated graph. First it indicates we can find a split sequence efficiently by finding a split supersequence. Second, intermediate computations in computing the split supersequence (such as network flow computations) are done on a graph of $O(m)$ edges. The details of this payoff are in the next section.

5 Integral Arborescence Packing

This section gives our integral packing theorem and efficient packing algorithms. As in Section 4 we assume G is a preflow mincut graph with $\rho(a) = 0$.

Section 4 showed a nondecreasing split sequence exists in a preflow mincut graph. Bang-Jensen et.al. [BFJ] give a construction that proves any digraph with a nondecreasing split sequence has a perfect packing. Since the construction is the basis of our packing algorithm we give their construction in detail here. It is convenient to state the construction recursively.

Given a preflow mincut graph G, we wish to compute a perfect packing $\mathcal{A}$ of a-arborescences. If G has only two vertices a, x then each arborescence of $\mathcal{A}$ consists of a copy of edge ax. If G has more than two vertices proceed as follows.

(Splitting Step.) Let s be a preflow vertex with minimum value $\lambda(s)$. Split off splittable pairs of edges incident to s until $\delta(s) = 0$. Delete s.

(Recursive Step.) Recursively construct a perfect packing $\mathcal{A}$ on the remaining graph.

(Unsplitting Step.) Do the following for each arborescence $A \in \mathcal{A}$. Let $r_it_i, i = 1, \ldots, k$ be the edges in A resulting from splits at s (i.e., each r_it_i was introduced by a split of edges r_is, st_i). Assume $k > 0$. Number the edges so r_1 does not properly descend from any $r_i, i > 1$. Replace A by the arborescence $A - \{r_it_i : 1 \le i \le k\} \cup \{r_1s\} \cup \{st_i : 1 \le i \le k\}$.

(Grow Step.) Repeat the following until s occurs in $\lambda(s)$ arborescences of $\mathcal{A}$. Choose an edge rs of G that is not in an arborescence of $\mathcal{A}$. Add rs to an arborescence of $\mathcal{A}$ containing r but not s.

Let us prove this construction is correct. The Unsplitting Step modifies $\mathcal{A}$ to a valid packing on G, i.e., every edge is an edge of G, and each $A \in \mathcal{A}$ is a valid arborescence. Every nonroot $r \ne s$ occurs in exactly $\lambda(r)$ arborescences, and there are $\Lambda(G)$ arborescences total. Next consider the Grow Step. Suppose s occurs in less than $\lambda(s)$ arborescences of $\mathcal{A}$. Some edge rs is not in any arborescence since $\rho(s) \ge \lambda(s)$. Some arborescence contains r but not s, since r occurs in $\lambda(r) \ge \lambda(s)$ arborescences. This implies that the Grow Step halts with s in $\lambda(s)$ arborescences, i.e., the packing is perfect.

Theorem 5.1. *A preflow mincut graph has a perfect packing of a-arborescences.* □

The rest of this section is organized as follows. We give a basic packing algorithm, and then implement it in two ways. Then we extend the second implementation to pack capacitated graphs.

The *basic packing algorithm* follows the above construction. The two implementations differ in how they find the split sequence. We now describe the basic algorithm, and estimate its time exclusive of finding the split sequence. The discussion is for uncapacitated graphs; the modifications for capacitated graphs are given below.

We begin with the Splitting Step of the basic algorithm. It constructs a nondecreasing split supersequence (this gives the nondecreasing split sequence used in the construction). Initialize $\mathcal{F}$ to a covering family for the nonpreflow nonroots. Repeatedly choose a preflow vertex s that has the minimum value $\lambda(s)$. Delete $\{s\}$ from $\mathcal{F}$ if it belongs to $\mathcal{F}$. Then split off pairs of edges incident to s to make $\delta(s) = 0$, as follows. (As usual G denotes the current graph.)

Repeatedly choose a pair of edges rs, st that makes $\mathcal{F}$ consistent. Let SG be the graph $G - \{rs, st\} \cup \{rt\}$. Check whether or not the pair of edges is splittable by checking if, for each nonroot $x \ne s$, $\lambda_{SG}(x) = \lambda_G(x)$. If so execute the split, i.e., replace G by SG. If not enlarge $\mathcal{F}$, as described in the next paragraph. When $\delta(s) = 0$ delete s.

The procedure to enlarge $\mathcal{F}$ is as follows. Recall from Section 4 that the set U for uncrossing is defined as a minimal inconsistent proper mincut contained a set of $\mathcal{F}$ of smallest possible size. Note that the inconsistent proper mincuts are precisely the sets that are x-mincuts in SG for vertices $x \ne s, a$ having $\lambda_{SG}(x) = \lambda_G(x) - 1$. For each such x find its unique minimal inconsistent mincut X, by examining the residual graph of a maximum flow from a to x in SG. Then find the smallest set of $\mathcal{F}$ that contains X. It is easy to find U from these sets X. Finally uncross $\mathcal{F} \cup \{U\}$.

This completes the description of Splitting Step. Further details of how we check a pair of edges is splittable are given in the implementations of the basic algorithm. We now show that the time for the Splitting Step, excluding these checks and the time to initialize $\mathcal{F}$, is $O(n^2 m)$.

The pair of edges rs, st is found using the procedure described after Lemma 4.2, in time $O(n)$. Theorem 4.5 shows there are $O(m)$ such pairs in the entire sequence. This gives total time $O(nm)$. Now we discuss enlarging $\mathcal{F}$. For each vertex x we find its minimal inconsistent mincut in time $O(m)$, assuming a maximum flow from a to x is given (as is the case in our implementations). It is easy to see that we find U in time total time $O(nm)$. We uncross $\mathcal{F} \cup \{U\}$ in time $O(n)$. Thus each enlarging step uses time $O(nm)$. Theorem 4.5 implies the total time for enlarging steps is $O(n^2 m)$.

The Unsplitting Step and Grow Step of the basic algorithm build the packing $\mathcal{A}$ by following the construction exactly. We now estimate the time. The notation $|\mathcal{A}|$ denotes the number of arborescences in $\mathcal{A}$. Let $\|\mathcal{A}\|$ denote the total number of edges of all arborescences in $\mathcal{A}$. (Thus $\|\mathcal{A}\| \leq (n-1)|\mathcal{A}|$, with equality when $\mathcal{A}$ consists of spanning arborescences. For an uncapacitated graph $\|\mathcal{A}\| \leq m$.) We will show that the total time for the Unsplitting and Grow Steps is $O(\|\mathcal{A}\| n)$, for $\mathcal{A}$ the final packing. To do this it suffices to show the time for one execution of the Unsplitting and Grow Steps is $O(n + \|\mathcal{A}\|)$, where $\mathcal{A}$ denotes the current packing.

For the Unsplitting Step, we identify the edges in the packing that are the result of splits at s using a radix sort in time $O(n + \|\mathcal{A}\|)$. Then we modify each arborescence for these edges as described in the construction, in time $O(n + \|\mathcal{A}\|)$. The Grow Step is done by scanning each arborescence, in the same time bound.

Now we present two implementations of the basic algorithm. As already noted the only detail to be specified is how to compute the split supersequence in the Splitting Step. We begin with the *augmenting algorithm.* This implementation is good for uncapacitated graphs when storage is not a problem.

The augmenting algorithm computes the split supersequence by maintaining, over the whole computation, a maximum flow from a to v for each nonroot v. Initially we find a maximum flow for each nonroot. We process each pair rs, st in the split supersequence as follows.

For each nonroot v we start with a maximum flow from a to v in G. First convert it to a flow on SG. (To do this repeat this procedure for edge e equal to rs and st: If e carries flow, find a path P carrying flow that passes through e and is either a cycle or goes from a to v. Remove one unit of flow from P.) The resulting flow on SG has value at least $\lambda_G(v) - 2$. Try to augment the flow to value $\lambda_G(v)$ by searching for up to two augmenting paths.

If every nonroot has a flow of value $\lambda_G(v)$ the pair is splittable. In the opposite case find the set U for uncrossing as described in the basic algorithm. This procedure uses the maximum flows on SG of value $\lambda_G(v) - 1$. After uncrossing, backtrack to make all flows maximum on G rather than SG.

This completes the augmenting algorithm. The following result applies to uncapacitated graphs. (Recall that parallel edges are allowed but counted as distinct in the parameter m.)

Theorem 5.2. *A perfect packing in a preflow mincut graph can be found in time $O(nm^2)$ and space $O(nm)$.*

Proof. Consider the Splitting Step in the augmenting algorithm. Finding n maximum flows initially uses time $O(n^2m)$. (In a network where each edge has capacity one, the time to find a maximum flow is $O(nm)$ [Ev].) This also accounts for the time to initialize $\mathcal{F}$ (recall the remark after Theorem 3.4). Since the split supersequence has length $O(m)$, we search for $O(nm)$ augmenting paths over the entire Splitting Step. This uses time $O(nm^2)$. This dominates the time for the rest of the Splitting Step (recall the analysis of the basic algorithm).

The time for the Unsplitting and Grow Steps is $O(\|\mathcal{A}\|n)$ as shown for the basic algorithm. Since $\|\mathcal{A}\| \leq m$ this is $O(nm)$. The space is linear except for storing n flows, which is $O(nm)$. $\square$

The second implementation of the basic algorithm is the *searching algorithm.* It uses linear space but is usually slower than the augmenting algorithm. It also generalizes to capacitated graphs.

The searching algorithm computes the split supersequence by maintaining the current graph G and the laminar family $\mathcal{F}$, as in the basic algorithm, plus a prefix P of the desired split supersequence. At any point we have executed the splittable pairs of P on G and enlarged $\mathcal{F}$ for the unspittable pairs. The algorithm repeatedly extends P by new pairs, modifying G and $\mathcal{F}$, until P is a complete split supersequence.

To extend P we construct a sequence of new pairs Q as follows. Let s be the vertex in the last split rs, st of P; if this vertex has been deleted from G then let s be a preflow vertex with minimum value $\lambda(s)$. Initialize graph SG to G. Pair each edge st leaving s with an edge rs that makes $\mathcal{F}$ consistent. Add each such pair rs, st to Q and in SG, replace rs, st by rt.

We repeat this procedure as many times as possible. The procedure terminates when $\delta_{SG}(s) = 0$ or when an edge rs making $\mathcal{F}$ consistent does not exist. (This can occur if Q already contains a split $r's, st'$ that is actually unsplittable.)

After Q is constructed we search it to find the longest prefix P' that is splittable, i.e., the graph SG corresponding to P' has the same λ values as G, but if $P' \neq Q$ the next graph SG does not. Add the pairs of P' to P and execute those splits on G. If $P' = Q$ then delete s from G. If $P' \neq Q$ then, for the split rs, st immediately following P' in Q, add rs, st to P and enlarge $\mathcal{F}$ for it. This completes the description of how we extend P.

The main task in this procedure is finding the longest splittable prefix P'. To do this, index the pairs of Q from 1 to q. We seek μ, the index of the last pair in P'. For $1 \leq i \leq q$ let $SG(i)$ be the graph SG after executing the splits for the first i pairs of Q. Then μ is the largest index such that for all $v \neq s$, $\lambda_{SG(i)}(v) = \lambda_G(v)$. Note that $q = O(n)$. Thus using binary search we can compute μ in $O(n \log n)$ max flow computations on graphs $SG(i)$. Section 6 shows that $O(n)$ max flow computations on these graphs suffice. The next two theorems assume this result.

The following result again applies to uncapacitated graphs.

Theorem 5.3. *A perfect packing in a preflow mincut graph can be found in time $O(n^2 \min\{m^{1/2}, n\}m)$ and space $O(m)$.*

Proof. Consider the computation of the split supersequence P. First observe there are $O(n)$ iterations, i.e., $O(n)$ sequences Q are computed. This follows since each sequence Q either deletes s or adds an unsplittable pair to P. The latter occurs $O(n)$ times by Theorem 4.5.

Each iteration computes a sequence Q in time $O(qn) = O(n^2)$. It also does $O(n)$ max flow computations on graphs $SG(i)$, as noted above. Each such graph has at most m edges (each split decreases the number of edges by one). A maximum flow computation uses time $O(\min\{m^{1/2}, n\}m)$ [Ev] (if the given graph has no parallel edges this equals $O(m^{3/2})$). This shows the time to compute the entire split supersequence is $O(n^2 \min\{m^{1/2}, n\}m)$. This dominates the time for the rest of the basic algorithm. $\square$

Our last packing algorithm is for capacitated graphs. Let the given preflow mincut graph G have a capacity function c. We use the searching algorithm, with details concerning the capacity of splits added. We now describe these details, beginning with the Splitting Step. We also verify that this step uses $O(n^2 T_{MF})$ time.

We construct the pairs of Q as follows. Repeatedly choose an edge st leaving s, and find an edge rs that makes $\mathcal{F}$ consistent. Add the pair rs, st to Q. Set $\gamma = \min\{c(rs), c(st)\}$. Decrease $c(rs)$ and $c(st)$ by γ, deleting any edge from SG whose capacity becomes 0. Increase $c(rt)$ by γ.

We find the longest splittable prefix P' in two steps. The first step proceeds as before to find the index μ of the last pair of Q that gives a valid voiding split. The second step applies if $\mu < q$. Let rs, st be the pair immediately following P'. We compute the capacity γ of split rs, st.

To compute the capacity recall that when splitting off rs, st the only sets whose in-degree decreases are $s\overline{T}$-sets and $T\overline{s}$-sets ($T = \{r, t\}$). Thus $\gamma = \min\{\rho(U) - \lambda_G(u) : u, s \in U \subseteq V - \{a, r, t\} \text{ or } u, r, t \in U \subseteq V - \{a, s\}\}$. We compute each term of this set using two max flow computations, in the first contracting a, r and t into the source and contracting u and s into the sink, and similarly for the second. Thus we compute the capacity γ in $O(n)$ max flow computations.

The rest of the procedure is similar to the uncapacitated case: Add the pairs of P' to P and execute those voiding splits on G. If $P' = Q$ then delete s from G. If $P' \neq Q$ then, for the split rs, st immediately following P', execeute that split to its capacity, add rs, st to P and enlarge $\mathcal{F}$ for it.

Now we show that the entire split supersequence is computed in time $O(n^2 T_{MF})$. As in the uncapacitated case there are $O(n)$ iterations. The time to construct Q is the same as the uncapacitated case. Each iteration does $O(n)$ max flow computations. Any graph for a max flow computation has $O(m)$ edges. This follows since such a graph is either some $SG(i)$ (used in the search procedure of Section 6) or a contraction of it (used to compute the capacity γ). A graph $SG(i)$ has $O(m)$ edges since there are $O(n)$ nonvoiding splits (see the discussion at the end of Section 4) and any pair added to Q has a multiplicity γ that makes it voiding. This implies the desired bound $O(n^2 T_{MF})$.

In the rest of the Splitting Step the procedure to enlarge $\mathcal{F}$ is unchanged. Clearly the rest of the Splitting Step runs within $O(n^2 T_{MF})$ time (this includes the time to initialize $\mathcal{F}$, $O(nT_{MF})$).

It remains to discuss the Unsplitting and Grow Steps. We state these steps completely for capacitated graphs and then analyze them. Recall that a packing $\mathcal{A}$ in a capacitated graph is a set of arborescences A, each with an associated multiplicity $\alpha(A)$.

Consider the Unsplitting Step. It undoes the splits at a fixed vertex s. We maintain a value $\gamma(rt)$ for each edge rt resulting from a split rs, st at s. Initialize $\gamma(rt)$ to the capacity of the split rs, st in our split sequence. (Note that the total capacity of rt in $\mathcal{A}$ can be greater than $\gamma(rt)$ – there can be copies of rt that are not the result of a split at s.)

Do the following for each arborescence $A \in \mathcal{A}$. Let $r_i t_i, i = 1, \ldots, k$ be the edges in A with positive values $\gamma(r_i t_i)$. Assume $k > 0$. Number the edges so r_1 does not properly descend from any $r_i, i > 1$. Let B be the arborescence $A - \{r_i t_i : 1 \leq i \leq k\} \cup \{r_1 s\} \cup \{st_i : 1 \leq i \leq k\}$. Let $\beta = \min\{\alpha(A), \gamma(r_i t_i) : 1 \leq i \leq k\}$. Add B to $\mathcal{A}$ with multiplicity β. Decrease $\alpha(A)$ by β, removing it from $\mathcal{A}$ if this makes $\alpha(A) = 0$. Decrease $\gamma(r_i t_i)$ by β, for $i = 1, \ldots, k$. (Note that B will not be processed again in this Unsplitting Step. A can be processed again if $\alpha(A)$ and some $\gamma(r_i t_i)$ are still both positive.)

Consider the Grow Step. It adds edges directed to the fixed vertex s to $\mathcal{A}$. We maintain a value $\gamma(rs)$ for each edge rs directed to s. Initialize $\gamma(rs)$ to $c(rs)$ decreased by the total multiplicity of arborescences of containing rs.

Repeat the following until the total multiplicity of arborescences containing s equals $\lambda(s)$. For each arborescence $A \in \mathcal{A}$ not containing s but containing a vertex r with $\gamma(rs) > 0$, let B be the arborescence $A \cup \{rs\}$ (choose r arbitrarily if there is more than one such vertex). Let $\beta = \min\{\alpha(A), \gamma(rs)\}$. Add B to $\mathcal{A}$ with multiplicity β. Decrease $\alpha(A)$ by β, removing it from $\mathcal{A}$ if this makes $\alpha(A) = 0$. Decrease $\gamma(rs)$ by β.

Now we give the time for these two steps. The notation $|\mathcal{A}|$ denotes the number of distinct arborescences in $\mathcal{A}$; $\|\mathcal{A}\|$ denotes the total number of edges of all distinct arborescences in $\mathcal{A}$. The total time for all Unsplitting and Grow Steps is $O(\|\mathcal{A}\| n)$, where $\mathcal{A}$ denotes the final packing. This follows by the same argument as the uncapacitated case. Since $\|\mathcal{A}\| \leq |\mathcal{A}| n$ the time for these steps is $O(|\mathcal{A}| n^2)$. Define Σ as the number of splits in the split sequence.

Lemma 5.4. $|\mathcal{A}| = O(m + \Sigma)$.

Proof. The proof is a charging scheme. The algorithm begins with one arborescence in $\mathcal{A}$ (created in the base case of the recursion). Each time a new arborescence in $\mathcal{A}$ is created (in the Unsplitting or Grow Steps) we will charge one unit to an edge e in the current graph. The total charge to any e will be at most two. There are $O(m + \Sigma)$ edges e in all graphs (e is either an edge in the original graph or is the result of a split). This implies the lemma.

Consider the Unsplitting Step. Suppose an iteration increases $|\mathcal{A}|$, i.e., it adds B to $\mathcal{A}$ without removing A. Then $\beta = \gamma(r_i t_i)$ for some index i. Charge the new arborescence to edge $r_i t_i$. It gets charged only once since the iteration sets $\gamma(r_it_i)$ to 0. (Clearly r_it_i does not get charged in another Unsplitting Step, since r_it_i is the result of a split at s.)

Next consider the Grow Step. Suppose an iteration adds B to $\mathcal{A}$ without removing A. Then $\beta = \gamma(rs)$. Charge the new arborescence to rs. It gets charged only once since the iteration sets $\gamma(rs)$ to 0. (Clearly rs does not get charged in another Grow Step since it is directed to s.) □

Now observe that $\Sigma = O(n^2)$. In proof, there are fewer than $2n$ voiding splits at any vertex v (since fewer than n edges enter (leave) v). This gives a total of $O(n^2)$ voiding splits. Theorem 4.5 shows there are an additional $O(n)$ nonvoiding splits.

We conclude that the time for the Unsplitting and Grow Steps is $O(n^2(m + \Sigma)) = O(n^4) = O(n^2T_{MF})$. This completes the time analysis of the algorithm.

Theorem 5.5. *A perfect packing in a capacitated preflow mincut graph can be found in time $O(n^2T_{MF})$ and space $O(n^3)$.* □

Note that the packing of this theorem contains $|\mathcal{A}| = O(n^2)$ distinct arborescences. In Sections 7–10 we show how to construct a fractional packing that has $O(m)$ arborescences.

6 Binary Search and the Oracle Minimum Problem

This section gives the binary search method used to implement our packing algorithm. We solve the following *oracle minimum problem*: Consider a set of n elements V. Each $v \in V$ has an associated value $M(v)$, an integer in $[0, N)$. The values $M(v)$ are unknown but we have an oracle that answers the question "Is $M(v) \geq i$?" for a given v and i. The problem is to compute $\min\{M(v) : v \in V\}$. We accomplish this using $O(n + \log N)$ oracle calls.

A natural example of the oracle minimum problem is finding the minimax value satisfying a predicate. More precisely we have a predicate $P(v, i)$ such that for each v, $P(v, i)$ holds exactly when i is at most some (unknown) value. We seek $\min_{v \in V} \max\{i : P(v, i)\}$. The oracle evaluates $P(v, i)$.

In the searching algorithm of Section 5, finding the longest splittable prefix P' in Q is an instance of the oracle minimum problem. The predicate $P(v, i)$ asserts that $\lambda_{SG(i)}(v) = \lambda_G(v)$ (the notation here follows Section 5). Thus as claimed in Section 5, the first unsplittable pair in NP can be found in time $O((n + \log n)T_{MF}) = O(nT_{MF})$.

The idea of the algorithm is to do one binary search for the minimum. Candidates in V for the minimum conduct the search in a round robin fashion, dropping out when they are shown not to be the minimum.

In detail, we use the following data structure. Each element v has associated integers ℓ_v and h_v, always satisfying $\ell_v \leq M(v) < h_v$. L is a list of elements known to contain the minimum. L is a managed as an input-restricted deque, i.e., all accesses are to the first or last element. The variable

a always refers to the first element of L and z always refers to the last. The primitives push and pop insert and delete the first element of L, respectively (thus they change a). The primitive dequeue deletes the last element of L (changing z).

```
$L \leftarrow V$; **for** $v \in V$ **do begin** $\ell_v \leftarrow 0$; $h_v \leftarrow N$; **end**;
**while** $\ell_z + 1 < h_z$ **do begin**
    **while** $M(z) < \ell_a$ **do** pop;    /* changes $a$ */
    **if** $M(z) < h_a$ **then begin**
        $\ell_z \leftarrow \ell_a$; $h_z \leftarrow h_a$; $p \leftarrow \lfloor(\ell_z + h_z)/2\rfloor$;
        **if** $M(z) \geq p$ **then** $\ell_z \leftarrow p$ **else** $h_z \leftarrow p$;
        push $z$;    /* changes $a$ */ **end**;
    dequeue;    /* changes $z$ */ **end**;
**return** $\ell_z$ as the minimum value;
```

Lemma 6.1. *The oracle minimum problem can be solved in $O(n + \log N)$ oracle calls.*

It is clear that the time bound is dominated by the time for the oracle calls. Also it is easy to see that this result is optimum.

Proof. It is straightforward to check the algorithm maintains these three invariants: The value of each element v lies in its interval $[\ell_v, h_v)$. Some element in L achieves the minimum. The intervals of elements in L are nested as in a binary search (except for extra intervals $[0, N)$ at the end of L).

The last two invariants imply that the value of any element in L lies in the interval $[\ell_z, h_z)$. Thus the minimum is in this interval. Hence if the algorithm halts ℓ_z is the desired minimum.

To complete the proof we show the algorithm halts and in fact the number of oracle calls is $O(n + \log N)$. Associate each oracle call with the first pop, push or dequeue that follows it. This associates $O(1)$ oracle calls with each deque operation (the worst case is three calls associated with a push). Since the number of pops plus the number of dequeues does not exceed n plus the number of pushes, it suffices to show the number of pushes is $O(n + \log N)$.

Each element v is pushed onto L for the last time at most once. So there are n such "last" pushes. Any other "nonlast" push eventually results in v becoming the last element z in L. At that point its search interval $[\ell_z, h_z)$ becomes an interval in the binary search for the minimum. So there are a total of at most $\log N$ "nonlast" pushes. □

We briefly mention some other applications of the binary search algorithm. The first two give only minor improvements to previous algorithms but they are straightforward applications of binary search.

The binding number of a graph is associated with its vertex connectivity. (As an example of its usefulness, a binding number $\geq 3/2$ guarantees a Hamiltonian circuit in a graph with > 2 vertices.) Cunningham [C] reduces computing the binding number to a predicate minimax problem, where each predicate is evaluated by a max flow computation. He finds the binding number in time $O(nT_{PMF})$, where T_{PMF} is the time to solve a parametric network flow problem in the framework of Gallo et.al. [GGT]. Our binary search algorithm finds the binding number in time $O(nT_{MF})$. (For calculating the binding number the oracle minimum problem has $N \leq n^3$.) The current bounds for T_{MF} are slightly better than T_{PMF} (by a sublogarithmic factor).

The strength of a network is computed in time $O(nT_{GMC})$ in [G]. Here T_{GMC} is the time to find a global mincut. The best-known bound for T_{GMC} [HO] is the same as the bound for T_{PMF}. Our algorithm gives time $O((n + \log N)T_{MF})$ which is a sublogarithmic factor better when $N = O(2^n)$. Another application is computing maximum throughput for the multicommodity flow problem of [BFJ, Corollary 2.2]. Here we know no previous techniques that apply except binary search. Our algorithm is a large improvement for large capacities N.

7 Consistent Arborescences for Fractional Packing

This section presents the basic ideas concerning arborescences for fractional packing in preflow mincut graphs. This includes the notions of capacity and consistency of an arborescence.

We first characterize the arborescences in a perfect packing. Consider a nonempty arborescence A. Define

$$\mu(A) = \min\{c(e) : e \in A\}.$$

For any α, $0 \leq \alpha \leq \mu(A)$, let $G - \alpha A$ denote the graph G with the given capacity function c decreased by α on the edges of A. As mentioned in Section 1, in $G - \alpha A$ we delete any edge that has zero capacity. As an example when all capacities are one, $G - A$ is the graph with the edges A removed.

Our strategy for packing is to transfer an arborescence A from the graph to the packing and recursively pack the remaining graph. The *capacity of* A is the largest value $\alpha \leq \mu(A)$ such that the desired packing can be formed from αA and a perfect packing in $G - \alpha A$. In this packing each vertex $u \neq a$ has in-degree $\lambda_{G-\alpha A}(u)$. So the capacity is the largest $\alpha \leq \mu(A)$ such that each $u \neq a$ satisfies

$$\alpha \rho_A(u) + \lambda_{G-\alpha A}(u) \geq \lambda_G(u).$$

(This inequality will actually hold with equality, since the left-hand side is obviously at most the right.) Since $\alpha = 0$ satisfies this condition the capacity of A exists although it may be 0. Note that the inequality for each u is equivalent to each $u\bar{a}$-set U having

$$\rho_{G-\alpha A}(U) \geq \lambda_G(u) - \alpha \rho_A(u).$$

We now characterize the arborescences of positive capacity. Consider an arbitrary graph G. An a-arborescence A is *consistent with a mincut* U if $\rho_A(U) = \rho_A(u)$ whenever U is a u-mincut. Note any a-arborescence has $\rho_A(U) \geq \rho_A(u)$. Thus consistency means A enters U at most once, and if it enters U then it spans every vertex for which U is a mincut.

Lemma 7.1. *The capacity of an a-arboresence A is positive if and only if $\mu(A) > 0$ and A is consistent with every mincut.*

Proof. For the only if direction suppose the capacity α of A is positive. Thus any u-mincut U has $\lambda_G(u) - \alpha\rho_A(u) \le \rho_{G-\alpha A}(U) = \rho_G(U) - \alpha\rho_A(U) = \lambda_G(u) - \alpha\rho_A(U)$. Simplifying with $\alpha > 0$ shows $\rho_A(u) \ge \rho_A(U)$. Thus $\rho_A(U) = \rho_A(u)$ and A is consistent with U.

For the if direction consider any $u\bar{a}$-set U. If U is a u-mincut then $\rho_A(U) = \rho_A(u)$ so a calculation similar to the above shows that for any α, $\rho_{G-\alpha A}(U) = \lambda_G(u) - \alpha\rho_A(u)$. Thus U does not limit the capacity. If U is not a u-mincut, i.e., $\rho_G(U) > \lambda_G(u)$, then for sufficiently small positive α, $\rho_{G-\alpha A}(U) = \rho_G(U) - \alpha\rho_A(U) > \lambda_G(u) - \alpha\rho_A(u)$. Thus the capacity of A is positive. □

It is convenient to allow broader usage of the term consistency: If $\mathcal{F}$ is a family of mincuts then A is *consistent with* $\mathcal{F}$ if it is consistent with each mincut of $\mathcal{F}$. Next suppose A is fixed. Then a mincut U is *consistent with vertex* u if U is a u-mincut and $\rho_A(U) = \rho_A(u)$. Conversely U is inconsistent with u if U is a u-mincut and $\rho_A(U) \ne \rho_A(u)$, i.e., $\rho_A(U) > \rho_A(u)$; such a U is an *inconsistent mincut.* We use this consequence of the lemma: An arborescence with capacity 0 has an inconsistent mincut.

Our packing algorithm will maintain a covering family $\mathcal{F}$ for the (changing) graph. This makes it convenient to extend the notion of covering family. In the rest of this paper, a *covering family* is a laminar family of mincuts that contains a preflow mincut for each vertex. Thus we allow a covering family to contain extra mincuts, which can even be nonpreflow sets.

We shall assume that $V - a$ is a preflow mincut in $\mathcal{F}$ (if not, add a new vertex a_0 with edge a_0a having capacity $A(G)$, and pack a_0-arborescences). Note that $A, B \in \mathcal{F}$ with $A \subseteq B$ implies $\rho(A) \le \rho(B)$ (this holds for any laminar family of mincuts). Call a set of vertices $A \in \mathcal{F} - \{V - a\}$ *loose* if any $B \in \mathcal{F}$ properly containing A has $\rho(B) > \rho(A)$. (The reason we do not call $V - a$ loose will be apparent after Lemma 7.3.)

Lemma 7.2. *Fix a covering family $\mathcal{F}$. Any loose set is preflow. The loose sets and $V - a$ form a covering family.*

Proof. Any vertex has a mincut in $\mathcal{F}$ that is $V - a$ or is loose. Thus we need only show any loose set A is preflow. If $u \in A$ then the preflow u-mincut in $\mathcal{F}$ is contained in A (by looseness). Thus A is the disjoint union of preflow sets, and so is itself preflow. □

Our packing algorithm will grow an a-arborescence A that is consistent with our covering family $\mathcal{F}$. We now show that a nonempty such A exists.

Consider a preflow mincut graph G with a covering family $\mathcal{F}$. Fix any set $X \in \mathcal{F}$. Define G_X as the subgraph of G induced by X, with every maximal set of $\mathcal{F}$ that is properly contained in X contracted. For a vertex $v \in X$ let $[v]$ denote the vertex of G_X that contains v (possibly $[v] = v$). In G_X define a vertex $[b]$ and a subset of "required" vertices R as follows: Choose any $[b]$ such that an edge of G enters both X and b. Let the set R consist of all vertices $[v]$ such that X is a v-mincut or an edge of G leaves both X and v.

Lemma 7.3. *G_X has a $[b]$-arborescence spanning R.*

Proof. It suffices to show that for every $\overline{[b]}$-set S in G_X containing a vertex $[v] \in R$, some edge of G goes from $X - S$ to S.

The first case is when we can choose v so that X is a v-mincut. Suppose the desired edge does not exist. Let e be an edge entering both X and $b \notin S$. Then

$$\rho_G(S) + c(e) \leq \rho_G(X) = \lambda_G(v) \leq \rho_G(S).$$

This is a contradiction since any edge capacity is positive, so the desired edge exists.

In the remaining case, each $[v] \in S$ is a loose set (interpret $[v]$ as a set of vertices). Hence each $[v]$ is a preflow set (Lemma 7.2). Thus S is a preflow set, i.e., $\rho_G(S) \geq \delta_G(S)$.

Choose any v with $[v] \in R \cap S$. Thus some edge vw leaves X. Suppose no edge of G_X enters S. Then

$$\rho_G(X - S) \leq \rho_G(X) - \rho_G(S) + \delta_G(S) - c(vw) < \rho_G(X).$$

But $X - S$ contains a vertex $[x]$ with X an x-mincut. This contradiction shows the desired edge entering S exists. $\square$

Now we show that G contains a nonempty arborescence A consistent with $\mathcal{F}$. We give an algorithm to construct A. Initialize A to contain an arbitrary edge leaving a. Then repeat the following step as many times as possible: Choose a maximal previously unchosen set $X \in \mathcal{F}$ such that an edge of A enters a vertex $b \in X$. Enlarge A by adding a minimal $[b]$-arborescence of G_X that spans all vertices $[v]$ such that X is a v-mincut or an edge of A leaves both X and v.

Lemma 7.3 shows the desired $[b]$-arborescence exists. A simple induction (using the definition of R) shows that A is an arborescence. The definition of each arborescence of G_X ensures A is consistent with $\mathcal{F}$. Thus the algorithm finds the desired arborescence.

It is easy to see another property that is important for our algorithm: A leaves any loose set that it enters.

For completeness we note that A is a minimal nonempty a-arborescence consistent with $\mathcal{F}$, and in fact any such arborescence can be constructed by our algorithm. To prove the first property (the minimality of A) let B be an a-arborescence properly contained in A. Consider a maximal set $X \in \mathcal{F}$ for which B does not contain all edges of A in G_X. B enters X but does not contain every vertex for which X is a mincut. Thus B is inconsistent with X. The second property (our algorithm can construct any minimal arborescence) follows by a simple induction. Note we have now shown that any minimal nonempty a-arborescence consistent with $\mathcal{F}$ leaves any loose set that it enters.

8 Enlarging $\mathcal{F}$ in Fractional Packing

This section validates the basic strategy for our fractional packing algorithm. The idea is that an a-arboresence that is inconsistent with some mincut but consistent with $\mathcal{F}$ has an inconsistent mincut that can be used to enlarge $\mathcal{F}$. This section shows precisely how to do this.

We start with a tool for applying our uncrossing Lemma 3.1.

Lemma 8.1. *Let $\mathcal{F}$ be a covering family. Let U and W be intersecting mincuts with $W \in \mathcal{F}$, that are not W-intersecting. Then there is a mincut T, $U \subseteq T \subseteq U \cup W$, satisfying the same conditions as U, with $\rho(T) = \rho(U)$ and $T \cap W$ a preflow set.*

Proof. Let $\mathcal{S}$ be the family of all preflow mincuts $S \in \mathcal{F}$ that are maximal subject to the constraint that they are contained in W and are S-intersecting with U. Form $T = U \bigcup \cup \mathcal{S}$.

U is central intersecting with $\mathcal{S}$, so T is a mincut with $\rho(T) = \rho(U)$. Since U and W are not W-intersecting, W is not a mincut for any vertex of $T \cap W$, i.e., T and W are not W-intersecting. To show $T \cap W$ is a preflow set observe that any vertex $s \in U \cap W$ is in a preflow mincut $S \in \mathcal{F}$. S is contained in W (since $\lambda(s) < \rho(W)$). Hence S is contained in $U \cap W$ or in a set of $\mathcal{S}$. Thus $T \cap W$ is preflow since it is the disjoint union of preflow mincuts of $\mathcal{F}$. □

Now we give two inequalities concerning an a-arborescence A and a mincut U. These lemmas are the main tools for enlarging $\mathcal{F}$. Our first inequality holds for any graph.

Lemma 8.2. *Let A be an arbitrary a-arborescence. Suppose U is a mincut not properly containing an inconsistent mincut. Let U be near-central intersecting with a family $\mathcal{S}$. Then for $T = U \bigcup \cup \mathcal{S}$,*

$$\rho_A(T) \geq \rho_A(U).$$

Strict inequality holds if A enters some $S \in \mathcal{S}$ but not $S \cap U$.

Proof. We first note two relations. Any set $S \in \mathcal{S}$ has

$$\rho_A(S \cap U) \leq 1. \tag{1}$$

This follows since Lemma 3.1(i) shows $S \cap U$ is a mincut. Hence the first hypothesis on U implies $S \cap U$ is consistent. This implies (1).

The second relation is that any sets $S, S' \in \mathcal{S}$ have

$$d_G(S, U) = d_G(S - U, S' - U) = 0. \tag{2}$$

This follows from Lemmas 3.1(i) and 3.5.

To prove the first inequality of the lemma it suffices to show that every edge $e \in A$ entering U corresponds uniquely to an edge of A entering T. If e enters both U and T it corresponds to itself. Suppose e enters U but not T. Thus for some $S \in \mathcal{S}$, e leaves $S - U$. Now (2) shows e goes from $S - U$ to $S \cap U$. Thus (1) shows e is unique for the set S. Some edge f of arborescence A enters S. (1) shows f goes from $\overline{S}$ to $S - U$. Now (2) shows f goes from $\overline{T}$ to $S - U$, i.e., f enters T. Thus we can let e correspond to f. It is clear that e is the only edge corresponding to f.

To prove the second (strict) inequality of the lemma take $S \in \mathcal{S}$ with A entering S but not $S \cap U$. Let edge $f \in A$ enter S. The hypothesis shows f enters $S - U$, and so (2) implies f enters T. No edge $e \in A$ entering U corresponds to f, since $\rho_A(S \cap U) = 0$. This implies the second inequality. □

The second lemma holds for a preflow mincut graph.

Lemma 8.3. *Let A be a minimal a-arborescence consistent with a covering family $\mathcal{F}$. Let U and W be intersecting mincuts with $W \in \mathcal{F}$, where neither set properly contains an inconsistent mincut. Suppose U and W are polar intersecting but not W-intersecting. Then $U - W$ is a mincut with*

$$\rho_A(U - W) \geq \rho_A(U).$$

Proof. Let $\mathcal{L}$ be the subfamily of $\mathcal{F}$ containing only its loose sets, W and $V - a$. $\mathcal{L}$ is a covering family (Lemma 7.2). Apply Lemma 8.1 to U and W to get the mincut T. The lemma makes $T \cap W$ a preflow set. Furthermore $T \cap W$ is the disjoint union of loose sets (see the last sentence of the lemma's proof).

Next we show that T and W are polar intersecting and we uncross them. T is a mincut for a vertex of $T - W = U - W$ (since U and W are polar intersecting). W is a mincut for a vertex of $W - T$ (by Lemma 8.1). Now the hypothesis of Lemma 3.1(*ii*) is satisfied for W and T. Thus $T - W$ and $W - T$ are mincuts. Since $T - W = U - W$ this gives the first assertion of the lemma.

We will prove that $\rho_A(T - W) \geq \rho_A(T)$. Observe that U and T are as in the hypothesis of Lemma 8.2. Thus $\rho_A(T) \geq \rho_A(U)$. These two inequalities plus $T - W = U - W$ imply the lemma's inequality, $\rho_A(U - W) \geq \rho_A(U)$.

For our desired inequality it suffices to show that any edge $vw \in A$ that enters T corresponds uniquely to an edge of A that enters $T - W$. If $w \in T - W$ then vw corresponds to itself. Suppose $w \in T \cap W$. The relation $\overline{d}_G(T, W) = 0$ (Lemma 3.1(*ii*)) implies $v \in W - T$. Choose an arbitrary path P in A from w to a leaf. The minimality of A implies A leaves any loose set that it enters (recall the discussion after Lemma 7.3). As noted above, $T \cap W$ is the disjoint union of loose sets. Since vw enters $T \cap W$, A leaves $T \cap W$. It is easy to see this implies P leaves $T \cap W$. Let f be the first edge of P leaving $T \cap W$ and let vw correspond to f.

We wish to show f enters $T - W$. The hypothesis of the lemma implies $W - T$ is consistent. Thus $\rho_A(W - T) \leq 1$. A enters $W - T$ on an edge preceding v. Thus f does not enter $W - T$. Hence f leaves W. Now $\overline{d}_G(T, W) = 0$ shows f enters $T - W$, as desired.

Edge vw is the only edge corresponding to f since all edges between them in A are in $T \cap W$. Hence we have the desired correspondence. □

For the rest of this section fix A as a minimal nonempty a-arborescence that is consistent with $\mathcal{F}$ but is inconsistent with some mincut. Let U be a minimal inconsistent mincut contained in a set of $\mathcal{F}$ of smallest possible size. Let U be inconsistent with the fixed vertex u. Let $\mathcal{W}$ be the family of maximal sets of $\mathcal{F}$ that are intersecting with U. Set $Z = U \bigcup \cup \mathcal{W}$. The rest of this section proves that $\mathcal{F} \cup \{Z\}$ is a covering family, larger than $\mathcal{F}$. This will validate our basic strategy for packing – when our algorithm discovers an arborescence of capacity 0 it will enlarge $\mathcal{F}$ by adding the set Z.

It is easy to see that $\mathcal{F} \cup \{Z\}$ is laminar. (Let X be the unique minimal set in $\mathcal{F}$ containing U. Any set of $\mathcal{F}$ properly contained in X is either disjoint from Z or contained in some $W \in \mathcal{W}$ or contained in U.) We will show that Z is a mincut (after Lemma 8.4) and $Z \notin \mathcal{F}$ (Lemma 8.5). These two facts complete the proof of our desired result.

Define W_u as the (unique) set of $\mathcal{W}$ containing u if such exists, else $\emptyset$. (The proof of Lemma 8.4 will show this notation is consistent with that used in Lemma 3.5.)

Lemma 8.4. *U is near-central intersecting with $\mathcal{W}$.*

Proof. It suffices to show that each set $W \in \mathcal{W} - W_u$ is W-intersecting with U. Suppose this fails for some $W \neq W_u$. We will derive a contradiction.

The hypotheses of Lemma 8.3 are satisfied: U does not properly contain an inconsistent mincut by its minimality. W does not contain an inconsistent mincut since U is contained in a set $X \in \mathcal{F}$ of smallest possible size, and $W \subset X$. U and W are polar intersecting since $u \in U - W$ ($W \neq W_u$). Thus Lemma 8.3 shows $U - W$ is a u-mincut and $\rho_A(U - W) \geq \rho_A(U)$. Since $\rho_A(U) > \rho_A(u)$ we deduce $\rho_A(U - W) > \rho_A(u)$, i.e., $U - W$ is inconsistent. Thus U properly contains the inconsistent mincut $U - W$, contradicting the minimality of U. □

Now Lemma 3.5 shows Z is a mincut, as desired.

Lemma 8.5. *$Z \notin \mathcal{F}$.*

Proof. Suppose on the contrary that $Z \in \mathcal{F}$. We will deduce conflicting relations $\rho_A(Z) = \rho_A(U) < \rho_A(Z)$.

Lemma 8.2 applies to the collection $\mathcal{W}$ (by Lemma 8.4). Thus $\rho_A(Z) \geq \rho_A(U) > \rho_A(u)$. Since A is consistent with $\mathcal{F}$ our supposition implies $1 \geq \rho_A(Z)$. We conclude $\rho_A(Z) = \rho_A(U) = 1$ and $\rho_A(u) = 0$. This also gives the first relation of our contradiction.

Now we deduce the second relation $\rho_A(Z) > \rho_A(U)$. We do this by showing W_u satisfies the hypotheses of the second part of Lemma 8.2, A enters W_u but not $W_u \cap U$.

A enters any vertex for which Z is a mincut. This follows from $\rho_A(Z) = 1$ and consistency. Since $\rho_A(u) = 0$, Z is not a mincut for u. Thus by Lemma 3.5, $\rho_G(Z) = \max\{\rho_G(U), \rho_G(W_u)\} = \rho_G(W_u)$ and Z is a mincut for some vertex of W_u. Hence A enters W_u, the first desired hypothesis.

$W_u \cap U$ is a u-mincut by Lemma 3.1(i). The minimality of U implies $W_u \cap U$ is consistent. Hence $\rho_A(W_u \cap U) = \rho_A(u) = 0$, the second desired hypothesis. □

9 Fractional Packing

This section presents our fractional packing algorithm.

As noted in Section 7 we assume $V - a$ is a preflow mincut for all vertices having the maximum flow value $\lambda(x)$. Consider the *greedy method* for fractional packing: Take any nonempty a-arborescence A having positive capacity α. Place A in the packing with multiplicity α, and replace G by $G - \alpha A$. If some edge still leaves a then find the rest of the packing recursively.

Our packing algorithm is an implementation of the greedy method (we add a constraint to choose A so it preserves our laminar family). Before giving our algorithm we prove some properties of any implementation of the greedy method.

If the greedy method halts it has found a perfect packing. In proof, we claim that at any point in executing the greedy method the following invariant holds. Let G be the original graph, G' the current graph and $\mathcal{A}$ be the current packing. Then for each vertex u, $\rho_{\mathcal{A}}(u) + \lambda_{G'}(u) \geq \lambda_G(u)$. This invariant follows by a simple induction using the original definition of capacity of an arborescence. When the greedy method halts each u has $\lambda_{G'}(u) = 0$. Thus the invariant implies u has the desired multiplicity in the final packing $\mathcal{A}$. The total multiplicity of $\mathcal{A}$ is at most the maximum flow value k in G, since $V - a$ is a mincut of value k. Thus the final packing is perfect.

Next observe that if a set U becomes a u-mincut, it remains a u-mincut throughout the rest of the greedy method's execution. In proof suppose $\rho_G(U) = \lambda_G(u)$. The definition of capacity and Lemma 7.1 show $\lambda_{G-\alpha A}(u) \geq \lambda_G(u) - \alpha\rho_A(u) = \rho_G(U) - \alpha\rho_A(U) = \rho_{G-\alpha A}(U)$. Comparing the first and last terms shows equality holds, i.e., U is a u-mincut in $G - \alpha A$ as desired.

Our algorithm uses a covering family $\mathcal{F}$ to find the above arborescence A. Let $\mathcal{F}_0$ be the covering family given by Theorem 3.4 for the (preflow mincut) input graph G. Without loss of generality $V - a \in \mathcal{F}_0$. We initialize $\mathcal{F}$ to $\mathcal{F}_0$. We only change $\mathcal{F}$ by the uncrossing operation defined in Section 4; for convenience we restate the definition. Let U be an $\overline{a}$-set. To *uncross* $\mathcal{F} \cup \{U\}$ means to execute the following procedure:

```
$\mathcal{W} \leftarrow$ the family of maximal sets of $\mathcal{F}$ that are intersecting with $U$;
$Z \leftarrow U \bigcup \cup\mathcal{W}$;
add $Z$ to $\mathcal{F}$;
```

Now we present our packing algorithm. G is initially the given graph and gets modified by the packing algorithm.

```
**procedure** *fractional_pack*; **begin**
$\mathcal{F} \leftarrow \mathcal{F}_0$; $\mathcal{A} \leftarrow \emptyset$;
**while** $\delta_G(a) > 0$ **do begin**
    $A \leftarrow$ a minimal nonempty $a$-arborescence of $G$ consistent with $\mathcal{F}$;
    $\alpha \leftarrow$ the capacity of $A$;
    **if** $\alpha > 0$ **then** add $A$, with multiplicity $\alpha$, to $\mathcal{A}$;
```

```
$G \leftarrow G - \alpha A$;
**if** $\mu_G(A) > 0$ and $A$ is consistent with $\mathcal{F}$ in $G$ **then begin**
    $U \leftarrow$ a minimal inconsistent mincut contained in a set of $\mathcal{F}$ of smallest possible size;
    uncross $\mathcal{F} \cup \{U\}$; **end**; **end**; **end**;
```

A exists by the discussion after Lemma 7.3. U exists by Lemma 7.1 (A has capacity 0 after αA is deleted). Thus the algorithm is well-defined. Clearly *fractional_pack* is an implementation of the greedy method. Note that even though A is consistent by construction, the consistency check (in the second **if** statement) can fail: deleting αA can make a set $U \in \mathcal{F}$ a mincut for new vertices, possibly inconsistent with them.

Now we will show that *fractional_pack* maintains $\mathcal{F}$ as a covering family. We must check this when αA is deleted and when the uncrossing operation is done.

Consider the deletion of αA. As noted for any implementation of the greedy method, each set of $\mathcal{F}$ remains a mincut. To show each vertex has a preflow mincut in $\mathcal{F}$ it suffices to show that each loose set remains preflow when αA is deleted (Lemma 7.2).

We first remark that during the execution of *fractional_pack*, a set can become loose only by entering $\mathcal{F}$. More precisely, a nonloose set in $\mathcal{F}$ never becomes loose. In proof suppose a set $C \in \mathcal{F}$ is nonloose. Thus $C \subset B \in \mathcal{F}$ with $\rho(B) = \rho(C)$. Thus C and B are mincuts for some vertex $c \in C$, and they remain so. Thus C never becomes loose.

Now we show that after *fractional_pack* deletes αA, any loose set L is a preflow set. We can assume L is loose before deletion. The minimality of A implies it leaves L if it enters L i.e., $\rho_A(L) > 0$ implies $\delta_A(L) > 0$ (see the discussion after Lemma 7.3). Since A was consistent when it was constructed $\rho_A(L) \leq 1$. Thus $\rho_A(L) \leq \delta_A(L)$. Also L is a preflow set before deletion. The last two facts show L remains a preflow set after deletion.

Now we show that an uncrossing operation keeps $\mathcal{F}$ a covering family; furthermore the operation enlarges $\mathcal{F}$. Consider the algorithm right before an uncrossing operation. As just proved $\mathcal{F}$ is a covering family. At this time A is still an arborescence in G since $\mu_G(A) > 0$. A is consistent with $\mathcal{F}$ by the consistency check of *fractional_pack*. A is minimal (see the discussion after Lemma 7.3). Now recall that Section 8 defines U and Z as in *fractional_pack*. Thus as shown in Section 8 the uncrossing operation, which adds Z to $\mathcal{F}$., keeps $\mathcal{F}$ a covering family and enlarges it.

In the rest of the analysis it is convenient to classify the iterations of *fractional_pack* according to the outcome of the last **if** statement. A *voiding iteration* makes $\mu_G(A) = 0$ (in the new graph G). An *inconsistent iteration* is not voiding and makes A inconsistent with $\mathcal{F}$ (in G). An *uncrossing iteration* is neither voiding nor inconsistent, so it does an uncrossing operation.

Theorem 9.1. *Procedure* fractional_pack *finds a perfect fractional packing $\mathcal{A}$, containing at most $m + n - 2$ distinct arborescences.*

Proof. We begin by showing *fractional_pack* halts. This implies it finds a perfect packing by our discussion of the greedy method.

Obviously there are only a finite number of voiding iterations. In an inconsistent iteration, A enters X but not x for some $x \in X \in \mathcal{F}$ and the iteration makes X an x-mincut. The latter occurs only a finite number of times. An uncrossing iteration enlarges $\mathcal{F}$ as shown above. This also occurs a finite number of times. We conclude that there are a finite number of iterations and *fractional_pack* halts.

To complete the proof we need only refine our estimates to show there are at most $m+n-2$ iterations. We do this by showing there are a total of at most m voiding or inconsistent iterations and $n-2$ uncrossing iterations. Of course m and n refer to the given graph. Our algorithm adds vertex a_0 and edge a_0a so it works with a graph that initially contains $m+1$ edges and $n+1$ vertices. In the rest of this argument we refer to the root vertex as a_0 (rather than a as before).

First consider uncrossing iterations. Each uncrossing operation adds a nonsingleton set to $\mathcal{F}$. This follows since U is not a singleton set (a singleton mincut is consistent) so Z is not a singleton. $\mathcal{F}$ is a laminar family on V, a set of n elements (we have included a). Thus $\mathcal{F}$ has at most $n-1$ nonsingleton sets. Initially $\mathcal{F}$ contains at least one nonsingleton set V (previously referred to as $V-a$). This implies at most $n-2$ uncrossing iterations.

Now consider voiding and inconsistent iterations. Without loss of generality each vertex $x \neq a_0$ has $\lambda(x) > 0$ in the initial graph. Say that the arborescence $A \in \mathcal{A}$ *completes* vertex $x \neq a_0$ if A is the last arborescence added to $\mathcal{A}$ containing x. Each $x \neq a_0$ is the head of precisely one edge in the arborescence that completes it. (This arborescence exists since initially $\lambda(x) > 0$ and we have shown that *fractional_pack* finds a perfect packing.) The initial graph contains $(m+1)-n$ other edges, each of which may give a voiding iteration. Thus it suffices to show the number of arborescences completing a vertex plus the number of inconsistent iterations is at most $n-1$.

To do this we define an equivalence relation on V. Two equivalent vertices are guaranteed to be completed by the same arborescence. Initially we create $n-1$ equivalence classes – if vertex $x \neq a$ achieves the maximum flow value $\lambda(x)$, make x and a equivalent (both are completed by the last arborescence) and make every other vertex its own equivalence class. Each inconsistent iteration will reduce the number of equivalence classes by 1. It is easy to see this implies the desired inequality.

An inconsistent iteration makes some $X \in \mathcal{F}$ a mincut for a new vertex x (see above). Since $\mathcal{F}$ is covering the iteration starts with X a mincut for some x_0. We make x and x_0 equivalent.

Now we show this relation has the two desired properties. First, x and x_0 are completed by the same arborescence. In proof let A be the arborescence that completes x_0. A is constructed after the inconsistent iteration under consideration (the inconsistent iteration does not void the edge of A entering X so it does not complete x_0). Hence by consistency A enters both x and x_0 and completes both (X is a mincut for both).

The second property is that x and x_0 were previously inequivalent. This follows since the inconsistent iteration makes $\rho(X) = \lambda(x_0) = \lambda(x)$ for the first time. □

We turn to the time bound for *fractional_pack.* Computing the capacity is the most time-consuming part of the algorithm. In the best case the algorithm of Section 10 computes the capacity in time $O(nT_{MF})$. Now we show that the time for the rest of an iteration is $O(nT_{MF})$. (As usual we assume $T_{MF} = \Omega(nm)$.)

The arborescence A is found in time $O(m)$ using the recursive procedure given in Section 7. The uncrossing operation is easy to implement in time $O(n)$. It remains to discuss the consistency check and the computation of U.

We start with the former, checking that A is consistent with $\mathcal{F}$. A vertex u is inconsistent with a mincut of $\mathcal{F}$ if and only if $\rho_A(u) = 0$ and the maximal u-mincut $U \in \mathcal{F}$ has $\rho_A(U) = 1$. This implies we can test all vertices u for inconsistency in total time $O(n)$ (do a depth-first search of the tree representing $\mathcal{F}$; the time bound assumes we know all in-degrees $\rho_G(U), U \in \mathcal{F}$).

Next we show how to find the set U of *fractional_pack.* U is defined as a minimal inconsistent mincut contained in a set of $\mathcal{F}$ of smallest possible size. Fix a vertex u. We begin by characterizing the minimal inconsistent mincuts for u. For each vertex x let U_x be the minimal u-mincut containing x (if it exists).

Lemma 9.2. *Any minimal inconsistent mincut for u is a set U_x.*

Proof. Let U be a minimal inconsistent mincut for u. We consider two cases for u depending on $\rho_A(u)$.

First suppose $\rho_A(u) = 0$. Thus $\rho_A(U) > 0$, i.e., U contains a vertex $x \in V(A)$. Since $\rho_A(U_x) > 0$, U_x is inconsistent, so $U = U_x$.

The remaining case for u is $\rho_A(u) = 1$. Thus $\rho_A(U) \geq 2$. Let P be the path in A from a to u. There are two possibilities for U: either $\rho_P(U) \geq 2$ or $\rho_{A-P}(U) \geq 1$.

In the first case an edge xy of P leaves U. Thus U_x is inconsistent, so $U = U_x$. In the second case an edge xy of $A - P$ enters U. Hence it enters U_y so U_y is inconsistent, i.e., $U = U_y$. □

The lemma implies we can find U in the desired time bound, as follows. Consider a vertex u. Construct the representation of all u-mincuts of Picard and Queyranne [PQ] in time $O(T_{MF})$. For each x, find U_x in time $O(m)$. Test if U_x is consistent in time $O(n)$. Find the minimal inconsistent sets U_x in time $O(nm)$. For each such set find the smallest set of $\mathcal{F}$ containing it in time $O(n)$.

Now we show that assuming the results of the next section, we achieve our time bound for fractional packing.

Corollary 9.3. *Procedure* fractional_pack *runs in time $O(n^3T_{MF})$.*

Proof. First consider the voiding iterations of *fractional_pack.* Lemma 10.1 shows the capacity is found in time $O(nT_{MF})$. Since there are $\leq m \leq n^2$ such iterations, the total time for these iterations is within the desired time bound.

Next consider the iterations that are not voiding. Lemma 10.1 shows the capacity is found in time $O(n^2 T_{MF})$. There are $O(n)$ such iterations (the proof of Theorem 9.1 shows there are at most $n-1$ inconsistent iterations and $n-2$ uncrossing iterations). This gives the desired time bound. $\square$

10 Arborescence Capacity Computation

This section shows how to compute the capacity $\alpha(A)$ of an a-arborescence A, using Newton's method for finding a root. It is similar to the capacity computation in [GM].

Recall that the capacity is the largest value $\alpha \leq \mu(A)$ such that for each $u \neq a$ and each $u\bar{a}$-set U,

$$\rho_{G-\alpha A}(U) = \rho_G(U) - \alpha\rho_A(U) \geq \lambda_G(u) - \alpha\rho_A(u). \tag{3}$$

We treat each vertex u separately, finding the largest α for each u. The capacity is then the smallest of these values.

Fix vertex u. We claim the desired inequality for U holds if and only if $\alpha \leq \alpha_U$, where we define

$$\alpha_U = \frac{\rho_G(U) - \lambda_G(u)}{\rho_A(U) - \rho_A(u)}.$$

By definition α_U equals ∞ if the denominator is 0. To prove this recall $\rho_A(U) \geq \rho_A(u)$ (A is an a-arborescence). If $\rho_A(U) = \rho_A(u)$ then (3) holds for any α, since $\rho_G(U) \geq \lambda_G(u)$. If $\rho_A(U) > \rho_A(u)$ then (3) is equivalent to $\alpha \leq \alpha_U$.

This characterization of α motivates the following algorithm to compute the capacity for the fixed vertex u.

```
$\alpha \leftarrow \mu(A)$;
**while** $\lambda_{G-\alpha A}(u) < \lambda_G(u) - \alpha\rho_A(u)$ **do begin**
    $U \leftarrow$ a $u\bar{a}$-set of minimum in-degree in $G - \alpha A$;
    $\alpha \leftarrow \alpha_U$; **end**;
```

To verify this algorithm is correct it suffices to show that it halts. We show that in fact there are at most $|A| + 1$ iterations.

First observe that the sequence of values of α is strictly decreasing. In proof, recall that U has in-degree $\geq \lambda_G(u) - \alpha\rho_A(u)$ exactly when $\alpha \leq \alpha_U$. Thus each iteration changes α from a value larger than α_U to α_U.

Next we claim that if an iteration computes the set U, then any $u\bar{a}$-set X with $\rho_A(X) \geq \rho_A(U)$ has $\alpha_X \geq \alpha_U$. Observe that the claim implies the desired bound: Obviously the claim shows the sequence $\rho_A(U)$ is strictly decreasing. Since $0 \leq \rho_A(U) \leq |A|$ the desired bound of at most $|A| + 1$ iterations follows.

To prove the claim let α equal its value when the iteration begins and U is computed. The definition of U implies $\rho_G(X) - \alpha\rho_A(X) \geq \rho_G(U) - \alpha\rho_A(U)$. Using $\alpha - \alpha_U > 0$ and the assumption

$\rho_A(X) \geq \rho_A(U)$ gives $\rho_G(X) - \alpha_U \rho_A(X) \geq \rho_G(U) - \alpha_U \rho_A(U)$. The right-hand side is at least $\lambda_G(u) - \alpha_U \rho_A(u)$ by (3) (with $\alpha = \alpha_U$). This implies the claim $\alpha_X \geq \alpha_U$.

Lemma 10.1. *The capacity $\alpha(A)$ of an a-arborescence A can be computed in time $O(n^2 T_{MF})$. If $\alpha(A) = \mu(A)$ the time is $O(nT_{MF})$.*

Proof. We focus on the time for a fixed u. The total time is a factor n larger.

The test of the loop amounts to one max flow computation. This computation also finds U. This gives time $O(T_{MF})$ for one iteration. In general there are $\leq |A| + 1 \leq n$ iterations. If $\alpha(A) = \mu(A)$ there is only one iteration. □

References

[BFJ] J. Bang-Jensen, A. Frank, B. Jackson: Preserving and increasing local edge-connectivity in mixed graphs. SIAM J. Disc. Math. **8**, 2, 1995, pp. 155–178

[C] W.H. Cunningham: Computing the binding number of a graph. Disc. Appl. Math. **27**, 1990, pp. 283–285

[Ed] J. Edmonds: Edge-disjoint branchings. In: Combinatorial Algorithms, R. Rustin, Ed., Algorithmics Press, NY, 1972, pp. 91–96

[Ev] S. Even: Graph Algorithms, Computer Science Press, Potomac, MD, 1979

[F92] A. Frank: Augmenting graphs to meet edge-connectivity requirements. SIAM J. Disc. Math. **5**, 1, 1992, pp. 25–53

[F93] A. Frank: Applications of submodular functions. In: Surveys in Combinatorics, London Math. Soc. Lecture Notes Series 187, K. Walker, Ed., Cambridge Univ. Press, NY, 1993, pp. 85–136

[G] H.N. Gabow: Algorithms for graphic polymatroids and parametric s-sets. Proc. 6th Annual ACM-SIAM Symp. on Disc. Algorithms, 1995, pp. 88–97

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

[GM] H.N. Gabow, K.S. Manu: Packing algorithms for arborescences (and spanning trees) in capacitated graphs. Proc. Fourth MPS Conf. on Integer Programming and Combinatorial Optimization, 1995, pp. 388–402

[HO] J. Hao, J.B. Orlin: A faster algorithm for finding the minimum cut in a directed graph. J. Algorithms **17**, 3, 1994, pp. 424–446

[KRT] V. King, S. Rao, R. Tarjan: A faster deterministic maximum flow algorithm. J. Algorithms **17**, 3, 1994, pp. 447–474

[L73] L. Lovász: Connectivity in digraphs. J. Comb. Th. (B) **15**, 1973, pp. 174–177

[L76] L. Lovász: On some connectivity properties of Eulerian graphs. Acta Math. Akad. Sci. Hungar. **28**, 1976, pp. 129–138

[M] W. Mader: On n-edge-connected digraphs. Annals Discr. Math. **17**, 1983, pp. 439–441

[PQ] J.-C. Picard, M. Queyranne: On the structure of all minimum cuts in a network and applications. Math. Prog. Study **13**, 1980, pp. 8–16