Efficient Algorithms for Generalized Cut Trees[+]

Dan Gusfield[*]
Dalit Naor[*]

Abstract

The Gomory-Hu cut tree is a compact and efficiently computed representation of selected minimum *edge* cuts in a weighted *undirected* graph G=(V,E) with n nodes. It represents $\binom{n}{2}$ minimum cuts, one for each pair of nodes in G, and can be constructed with only $n-1$ flow computations.

In this paper we generalize the types of cut trees that can be efficiently constructed. We solve the open problem, posed by T.C. Hu [H], of constructing with $n-1$ flows a *cut-tree* for minimum *node weighted* cuts in an undirected graph. We then show how to build cut-trees that compactly represent the minimum edge cuts in *directed* graphs, partially solving the open problem of constructing cut-trees for weighted edge cuts in directed graphs. We also briefly discuss a generalization of the above cut trees to efficiently represent *all* minimum cuts between any pair of nodes.

* Computer Science Division, University of California, Davis, CA. 95616.

+ Research partially supported by a grant CCR-8803704 from the National Science Foundation.

1 Introduction and Statement of Results

For an undirected edge-capacitated graph G with n nodes, Gomory and Hu [GH] showed that the flow values (and thus the minimum edge cut values) between all of the $n(n-1)/2$ pairs of nodes can be computed using only $n-1$ flow computations. The $\binom{n}{2}$ flow values can be represented by a weighted tree T on n nodes, where for any pair of nodes (x, y), if e is the minimum weight edge on the path from x to y in T, then the value of the minimum (x, y) cut in G is exactly the weight of e. Such a tree is called an *equivalent flow* tree.

Gomory and Hu further showed that there exists an equivalent flow tree, where for every pair of nodes (x, y), if e is as above, then the two components of $T-e$ define a minimum cut between x and y in G. Such a tree is called a *cut tree*[GH]. The cut tree compactly represents all $\binom{n}{2}$ minimum cut values as well as a set of n-1 cuts with the property that for any pair of nodes (i, j), at least one cut in this set is a minimum (i, j) cut, and it can be retrieved quickly given (i, j). In Figure 1, an undirected graph G and its cut tree T are shown.

Producing a cut tree (by Gomory-Hu method [GH] or Gusfield's method [G]) or an equivalent flow

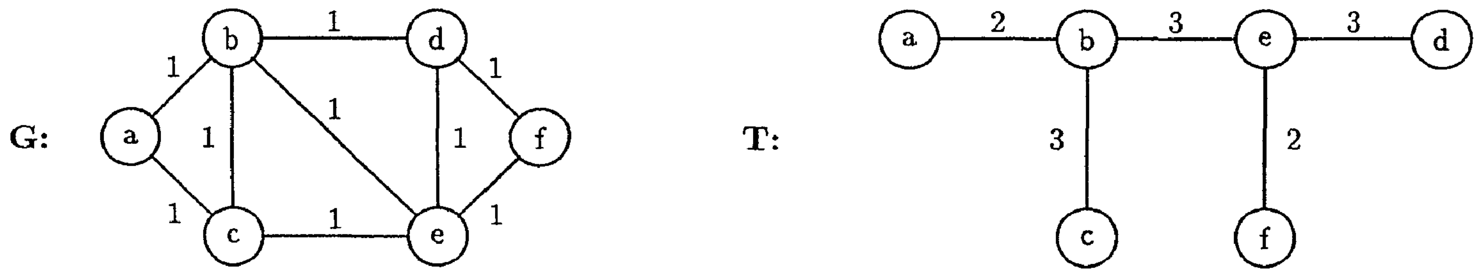


Figure 1: *A graph G and its cut tree T.*

tree (by Gusfield's method [G] or Granot-Hassin's method [GrH]) requires only $n-1$ flow computations, a large improvement over the obvious approach for computing the $\binom{n}{2}$ minimum cuts.

There are additional cut types that are not handled by the Gomory-Hu method, for example weighted *node* cuts or *directed* cuts. It is natural to ask whether a representation, analogous to the GH (Gomory Hu) tree, exists for other types of cuts. T.C. Hu [H] posed the question of the existence and efficient construction of a cut tree representing minimum *node* weighted cuts, and left open the question of building cut trees for directed graphs. Granot and Hassin [GrH] partially answered Hu's first question, constructing with $n-1$ flows an *equivalent flow* tree but not a *cut tree* for node weighted graphs.

The cut tree problem for directed graphs seems somehow different as there can be as many as $(n+2)(n-1)/2$ distinct minimum cut values in a directed graph with n nodes [FrFr]. Hence, a single small cut tree like the GH tree can not possibly represent all these values and cuts. However, we will see that these cuts can be represented in some compact and useful structure.

1.1 Main Results

Before we discuss the main results in this paper, note that a GH *cut tree* has three desirable features that we would like to achieve in other cut trees:

- The cut tree takes only $O(n)$ space, and yet implicitly represents one minimum cut for any pair of nodes.
- For any pair of nodes, one minimum cut separating the nodes can be extracted in $O(n)$ time from the tree. (Also, for any pair of nodes, the max flow *value* between the nodes can be retrieved in $O(1)$ time after an $O(n\alpha(n))$ time preprocessing of the values in the tree, by using the *least common ancestor* algorithm on weighted trees of [HT] or [SV]. A related method to efficiently retrieve values once the GH tree is known can be found in [T] and [Cha].)
- The cut tree can be constructed n times faster than by finding each of the $\binom{n}{2}$ cuts separately.

In this paper we solve the problem posed by Hu [H] and show how to construct a cut-tree for minimum node weighted cuts with only $O(n)$ flows; the tree that we construct is of size $O(n)$, and given any pair of nodes (i,j), the minimum weight node cut between i and j and its value can be found in $O(n)$ time. Hence all desirable features of a cut tree are obtained in this case.

In the case of directed graphs, we construct a family of n trees $T_1,\ldots,T_n$, where T_s is a cut tree representing the minimum cuts for all ordered pairs $\langle s,j\rangle$, i.e. all ordered pairs whose first node is s. These trees therefore represent one directed minimum cut for any ordered pair of nodes, in space which is an order of n smaller than the obvious representation. Further, for any ordered pair $\langle i,j\rangle$ a minimum cut separating i from j can be extracted from T_i in $O(n)$ time. The construction of each T_i requires $n-1$ flows, so these trees have only two of the desirable features of a GH cut tree. However, it is known that for a *directed* graph of n nodes, there can be as many as $(n+2)(n-1)/2$ distinct minimum cut values in the set of minimum cut values taken over all the $n(n-1)$ ordered pairs of nodes [FrFr]. Hence there seems to be little hope to be able to build cut trees for directed graphs in worst case time much better than we have achieved.

For all these cut trees, as well as for the Gomory-Hu tree, we describe an extension that represents *all* minimum cuts for any pair of nodes (and not only one cut per pair).

Throughout the paper we denote by $f(s,t)$ the maximum flow (or the minimum cut) value between s and t in an *edge*-capacitated graph. If a cut $(S,\bar{S})$ is said to be a min (s,t)-cut, then $s \in S$ and $t \in \bar{S}$. For a *node*-weighted graph, f_{st} denotes the minimum weighted node cut value (or the maximum flow) separating s from t.

Our first two results involve an additional type of cut tree on a directed graph, called the β *cut tree*, studied by Schnorr [Sch]. Section 2 describes the β cut tree and Schnorr's construction. Section 3 describes cut trees for node weighted graphs, and Section 4 describes cut trees for directed edge-capacitated graphs. In both sections, we first define cut trees in these types of graphs, and then show how to construct them efficiently. Finally, Section 5 discusses extensions of cut trees to solve the all-minimum-cuts problem.

2 The β Cut Tree

The generalization of cut trees to different types of graphs involve simulating and speeding up a cut tree construction algorithm due to C.P.Schnorr [Sch]. This section describes Schnorr's method.

For a *directed* graph G with n nodes, let $\beta(i,j)$ be the *minimum* between the max flow from i to j and the max flow from j to i, and let the β-cut between i and j be a directed cut separating i from j whose value is $\beta(i,j)$. Schnorr [Sch] showed that the β cuts and values between all of the $n(n-1)/2$ possible pairs of nodes can be represented by a directed cut tree T_β that is very similar to the Gomory-Hu cut tree. For every pair of nodes (x,y), if e is the minimum weighted edge on the (undirected) path from x to y in T_β, then the two components of $T_\beta - e$ define a β-cut between x and y (the direction of the cut is determined by the direction of e). Figure 2 shows a directed graph G and its β cut tree T_β.

However, whereas the GH cut tree can be computed with only $n-1$ flow computations, Schnorr's computation of a β cut tree requires $O(n \log n)$ flow computations . Schnorr shows that these $O(n \log n)$ flow computations can be implemented with some tricks to take $O(n^4)$ amortized time using particular flow algorithms. Note that an algorithm that requires only $O(n)$ max flow computations is always at least equal or superior (in its running time) to an $O(n^4)$ time algorithm that requires $O(n \log n)$ max flow computations; moreover, it lends itself to any further improvements in max flow techniques.

We now give a high level description of Schnorr's algorithm for constructing a β cut tree for any directed graph G with n nodes. We first need the following definition:

Definition : For a given subset of nodes $U \subseteq V$, we say that $(A,\overline{A})$ is a *minimum U cut* if it the minimum weighted cut taken over all cuts in G that split the nodes of U.

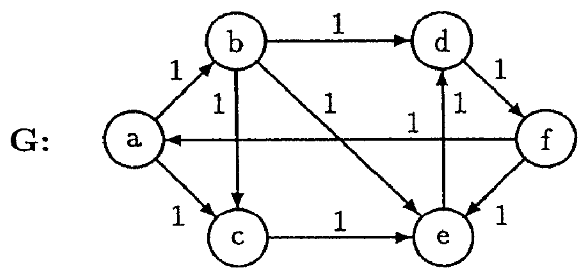
Note that a min U-cut is a cut *in G* which splits U.

The β cut tree algorithm [Sch] :

Input: A directed graph G

Output: A β cut tree $\mathcal{N}_n$ of G.

- Let $\mathcal{N}_0$ be a tree represented by a single supernode containing $\{V\}$.
- Given $\mathcal{N}_k$, construct $\mathcal{N}_{k+1}$ as follows:
  Let U be any supernode in $\mathcal{N}_k$. Contract each



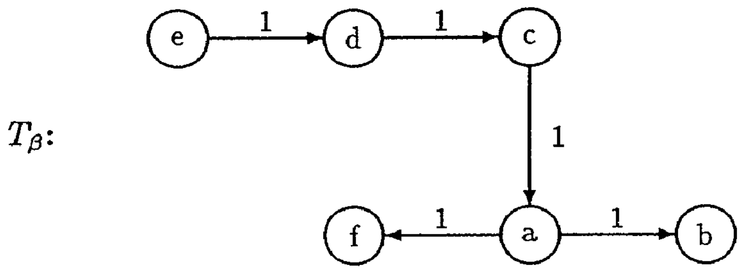


Figure 2: *A directed graph G and its β cut tree T_β. T_β was obtained by selecting the following sequence of min U-cuts (directed from left to right):(cde, abf), $(abcde, f)$, $(acdef, b)$, $(e, abcdf)$ and $(de, abcf)$.*

component of $\mathcal{N}_k - U$ in G and compute a minimum U-cut in the contracted graph. Suppose that $(X_{k+1}, \overline{X_{k+1}})$ is a minimum U-cut, directed from X_{k+1} to $\overline{X_{k+1}}$, and that its weight is α_{k+1}. Reconstruct $\mathcal{N}_{k+1}$ by splitting U into two nodes and adding an edge

$$(U \cap X_{k+1}) \longrightarrow (U \cap \overline{X_{k+1}})$$

of weight α_{k+1}. Attach the components of $\mathcal{N}_k - U$ either to $(U \cap X_{k+1})$ or to $(U \cap \overline{X_{k+1}})$, depending on the side of the cut each component falls in, while preserving the previous weights of edges.

Example: The β cut tree T_β for the graph G shown in Figure 2 was obtained by selecting the following sequence of min U-cuts (the cuts are directed from left to right):(cde, abf), $(abcde, f)$, $(acdef, b)$, $(e, abcdf)$ and $(de, abcf)$.

Note that the method computes only $n-1$ minimum U-cuts. The key to Schnorr's original method is how to efficiently compute these $n-1$ minimum U-cut. Using a clever idea, Schnorr showed how to do this efficiently so that the entire algorithm requires only $O(n \log n)$ max flow computations, rather than the obvious $n(n-1)$ flows. Since it is not relevant for our purposes, we do not describe it here.

Remark- Schnorr's original β tree construction algorithm can be sped up by using and adapting a new method, recently proposed by Cheng and Hu ([ChHu],[GN2]). By using this method, the general β tree construction algorithm can be implemented with only $O(n)$ maximum flows, as opposed to $O(n \log n)$ maximum flows that the original method requires. Although this method could have been used in Theorems 3.2 and 4.2, the solutions presented here are simpler and use fewer flow computations.

3 Cut Trees of Node Weighted (Undirected) Graphs

Let $G = (V, E)$ be an undirected, node-weighted graph, where w_i is the weight at node i. The maximum flow, or the minimum node cut, between a pair of nodes (i, j) is defined as

$$f_{ij} = C_{ij} = \min\{w_i, w_j, \min_{S \subseteq V}\{w_S\}\}$$

where S is any set of nodes whose removal disconnects i from j and $w_S = \sum_{k \in S} w_k$ (S is called *an (i,j) separating set*). This definition is the natural one when the nodes are weighted, but it differs from the definition of a node cut for unweighted nodes, because the unweighted node cut between node i and j is not permitted to contain either i or j; allowing either of those to be in the cut trivializes the problem in the unweighted case, but not in our more general weighted case.

The problem of finding all possible minimum cut *values* between pairs of nodes in an undirected node-weighted graph has been solved by Granot and Hassin [GrH]. They gave a method that computes these values with only $n-1$ maximum flow computations and represents them in an equivalent flow tree, which is not a cut tree in the sense defined here. The question of a cut tree was not addressed there.

In this section we show how to construct a *node cut tree* T_n for such graphs with only $O(n)$ maximum flow computations. Since a node cut is a *subset* of the nodes (as opposed to an edge cut which is a partition of the nodes into two sets), we have to define precisely what we mean by a "node cut tree", i.e. how can a set of $n-1$ node cuts be represented by a tree T_n, and we have to state what the mechanism is for retrieving a cut from this tree.

A **node cut tree** T_n of a graph G is a directed tree whose vertices are $\{u, u' | u \in V\}$, such that for every pair of nodes (i,j) $(i, j \in V)$, the weight of the minimum edge e on the undirected path from i to j' in T_n is f_{ij}. Let $(X, \overline{X})$, directed from X to $\overline{X}$, be the (i, j') cut in T_n defined by the removal of edge e from the tree (the direction of the cut is defined by the direction of e in T_n; w.l.o.g, if e is directed from the subtree that contains i to the subtree that contains j', then X is the side of the cut that contains i); then the set of all nodes u such that $u \in X$, $u' \in \overline{X}$ is a minimum (i,j)-separating set in G (that is, a set of nodes whose removal disconnects i from j in G).

In Figure 3 we show a node weighted (undirected) graph G and its node cut tree T_n. Consider, for example, the pair (b,d). A minimum (b,d) separating set in G can be obtained by deleting the edge $\langle b, d'\rangle$ from T_n. This deletion identifies the nodes $\{a, c\}$ as a minimum (b,d)-separating set in G since both a, a' and c, c' are partitioned by it.

Typically, a node weighted (undirected) graph G is transformed into an edge-weighted (directed) graph $\vec{G}$ as follows: each node u of weight w_u in G splits into two nodes u and u' in $\vec{G}$ which are connected by a directed edge $\langle u, u'\rangle$ of weight w_u; in addition, each edge (u,v) in G is transformed in $\vec{G}$ into two directed edges $\langle u', v\rangle$ and $\langle v', u\rangle$, both with an infinite weight, i.e. $w_{u'v} = w_{v'u} = \infty$.

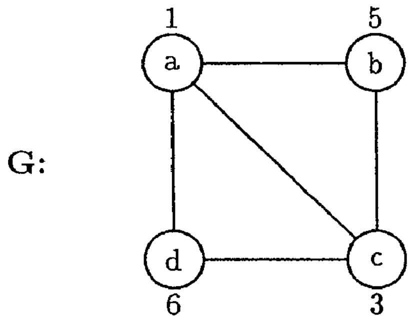
Note that if $\vec{f}(s,t)$ denotes the maximum flow from s into t in $\vec{G}$ then $\vec{f}(i,j') = \vec{f}(j,i') = f_{ij}$; also, as we will show in Lemma 3.2, if $(X, \overline{X})$ is a minimum (i,j')-cut in $\vec{G}$ with $i \in X$, $j' \in \overline{X}$, then the set $S = \{u | u \in X, u' \in \overline{X}\}$ is a minimum weighted (i,j) separating set in G. Hence, an edge cut tree for $\vec{G}$ suits the definition of a node cut tree for G.

Since $\vec{G}$ is a directed graph, the Gomory-Hu method cannot be applied in this case. However, in Theorem 3.1 below we show that such a tree exists and in Theorem 3.2 we show that it can be constructed efficiently. We first state the following Lemmas (Lemmas 3.2 and 3.3 are straightforward and left without proofs):

Lemma 3.1 *Let $(u^\star, v^\star)$ be any pair of nodes in $\vec{G}$ such that $u^\star \in \{u, u'\}$, $v^\star \in \{v, v'\}$ and $u \neq v$. Then, $\beta(u^\star, v^\star) \equiv \min\{\vec{f}(u^\star, v^\star), \vec{f}(v^\star, u^\star)\} = f_{uv}$.*
Also, $\beta(u, u') = \min\{w_u, w_{N(u)}\}$ where $w_{N(u)}$ is the sum of weights of u's neighbors in G.

Proof: For any pair of nodes (u,v) in G, $u \neq v$, let $M_{uv} = \min_{S \subseteq V \setminus \{u,v\}} \{w_S\}$ where S is any subset of nodes (excluding u and v) whose removal disconnects u form v in G, and $w_S = \sum_{i \in S} w_i$ (M_{uv} is the minimum weighted node cut between u and v, when we exclude the possibility of removing either u or v).

$\vec{f}(u', v) = \vec{f}(v', u) = M_{uv}$, and clearly $\vec{f}(u, u') = w_u$, $\vec{f}(v, v') = w_v$; therefore $\vec{f}(u, v) = \min\{w_u, M_{uv}\}$, $\vec{f}(u', v') = \min\{M_{uv}, w_v\}$. Hence, for $u \neq v$ the lemma follows since $\beta(u^\star, v^\star) = \min\{\vec{f}(u^\star, v^\star), \vec{f}(v^\star, u^\star)\} = \min\{w_u, w_v, M_{uv}\} = f_{uv}$.
Note that $\vec{f}(u, u') = w_u$ as $\langle u, u'\rangle$ is the only edge out of u and into u'. Also, if $v_1, \ldots v_k$ are the neighbors of u in G, then there are k disjoint paths from u' to u, all of the type $u' \to v_i \to v_i' \to u$, and each can flow w_{v_i} units, yielding the total of $w_{N(u)}$. Moreover, the flow out of u' can not exceed $w_{N(u)}$ since the total capacity out of the neigh-



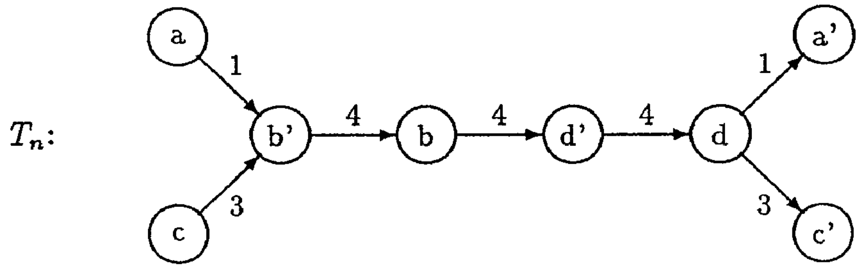


Figure 3: *A node weighted (undirected) graph G and its node cut tree T_n. The sequence of cuts that were selected for constructing T_n is (cuts are directed from left to right): $(a, a'bb'cc'dd')$, $(abb'cc'dd', a')$, $(c, aa'bb'c'dd')$, $(aa'bb'cdd', c')$, $(acbb', a'c'dd')$, $(acb', a'c'bdd')$ and $(abb'cd', a'c'd)$.*

bors of u' is $w_{N(u)}$, which certainly bounds the flow out of u'. Hence $\vec{f}(u', u) = w_{N(u)}$ implying that $\beta(u, u') = \min\{w_u, w_{N(u)}\}$. □

Lemma 3.2 *Let $(Y, \overline{Y})$ be a cut in $\vec{G}$ with finite weight, and define $S_Y = \{u | u \in Y, u' \in \overline{Y}\}$. Then: (1) The weight of $(Y, \overline{Y})$ equals $\sum_{k \in S_Y} w_k$, and (2) For any $s^\star \in Y$, $t^\star \in \overline{Y}$, where $s^\star \in \{s, s'\}$ and $t^\star \in \{t, t'\}$, S_Y is an (s,t) separating set in G.*

Lemma 3.3 *Let $(X, \overline{X})$ be a β cut between s and t' in $\vec{G}$ directed from X to $\overline{X}$. The subset of nodes $S_X = \{u | u \in X, u' \in \overline{X}\}$ is a minimum weighted (s,t) separating set in G.*

Theorem 3.1 *For any undirected, node weighted graph G there exists a node cut tree T_n.*

Proof: It follows from Lemma 3.1 that $\beta(i, j') = f_{ij}$ for any pair (i,j) in G, and from Lemma 3.3 that a β cut separating i from j' is a minimum weight (i,j) node cut in G. Hence, a β cut tree of $\vec{G}$ is a node cut tree of G, and the existence of T_n follows, as a β cut tree exists for any directed graph [Sch]. □

How efficiently can T_n be constructed? Clearly T_n can be constructed (by directly applying Schnorr's method) with $O(n \log n)$ max flow computations on $\vec{G}$. However, the algorithm of Granot and Hassin [GrH] builds the *equivalent flow* tree of G with only n-1 flow computations. We now show how to use it so that the β cut tree algorithm of Schnorr can be simulated with only $2n - 1$ additional max flow computations, yielding a method that builds the cut tree with $2n - 3 = O(n)$ max flows.

The idea for devising a faster method is to use Lemma 3.1 which says that the β value between any two nodes (primed or unprimed) in $\vec{G}$ is really the flow value between the corresponding unprimed nodes in G; also, the β value between u and u' in $\vec{G}$ is the minimum between the weight of u in G and the sum of the weights of u's neighbors in G and thus can be easily computed from the input graph with no flow computation. Hence, all β values in $\vec{G}$ are known once the max flow values in G are known. Now, recall that the time dominant step in Schnorr's β cut tree algorithm is the one that finds a min U-cut for a given subset U (the minimum cut in G that splits U). But once all β values are *known*, this step requires only a single max flow computation since we know in advance which pair among the nodes of U gives a minimum U-cut.

Using this idea, we can construct a node cut tree as follows: First, find an equivalent flow tree for G by applying the Granot-Hassin algorithm. Then construct the graph $\vec{G}$ and for each pair of nodes $(u^\star, v^\star)$ in the $\vec{G}$, $u^\star \in \{u, u'\}$, $v^\star \in \{v, v'\}$, compute $\beta(u^\star, v^\star)$ using flow values obtained from the equivalent flow tree (by the formulas given in Lemma 3.1). Finally, simulate Schnorr's β cut tree algorithm : whenever a min U cut is to be found, use the pre-computed β values to find the pair of nodes from U

whose β value is the smallest over all pairs, and compute a single minimum cut in $\vec{G}$ between this pair of nodes. Hence we get

Theorem 3.2 *The node cut tree T_n of an undirected node-weighted graph G can be computed with at most 3n-2 flow computations.*

Proof: The first step requires $n-1$ maximum flows, and the simulation requires $2n-1$ max flows since the number of nodes in $\vec{G}$ is $2n$ and Schnorr's method computes $2n-1$ minimum U-cut computations. □

Example: Consider the graph G depicted in Figure 3. To construct its node cut tree, we first apply the transformation (not shown here) and then calculate the β values in the transformed graph (via the Granot-Hassin algorithm and Lemma 3.1) and get:

- $\beta(a,x) = \beta(a',x) = 1$ for $x \in \{a',b,b',c,c',d,d'\}$,
- $\beta(c,x) = \beta(c',x) = 3$ for $x \in \{b,b',c',d,d'\}$,
- $\beta(b,d) = \beta(b',d) = \beta(b,d') = \beta(b',d') = 4$.

Given these values, the tree is constructed by the following sequence of cuts (directed from left to right): $(a, a'bb'cc'dd')$, $(abb'cc'dd', a')$, $(c, aa'bb'c'dd')$, $(aa'bb'cdd', c')$, $(acbb', a'c'dd')$, $(acb', a'c'bdd')$ and $(abb'cd', a'c'd)$. The final tree T_n is shown in Figure 3.

Remarks - Note that although the node case was reduced to a directed graph, the result obtained is stronger than that obtained in the case of a general directed graph discussed in the next section. That is, in the node weighted case n flows suffice, whereas in the general directed graph case $O(n^2)$ flows are needed. We also note that the case where both nodes and edges are weighted can be handled by the above techniques.

4 Cut Trees of Directed Graphs

Let $G = (V,E)$ be an edge capacitated *directed* graph where for any $\langle i,j\rangle \in E$, w_{ij} is the capacity on edge $\langle i,j\rangle$. $f(s,t)$ is the maximum flow value from s to t, and $C(s,t) = f(s,t)$ is the minimum cut separating s from t (i.e. the cut is directed *from s to t*; note that in the context of directed graphs the direction is significant). The Gomory-Hu method cannot be applied to directed graphs; in fact, it can be shown that a directed graph can have $(n-1)(n+2)/2$ different flow values ([FrFr]), whereas in an undirected graph there can be at most $n-1$ distinct flow values. Hence, it seems that the determination of all possible flow values requires $O(n^2)$ flow computations, and that a set of minimum cuts that contains at least one cut for each pair of nodes must consist of $O(n^2)$ cuts. But is there a collection of cuts which is nicely structured (i.e. representable by trees) that contains at least one minimum cut for each pair of nodes? Such a nicely structured set of cuts would more compactly represent the desired minimum cuts than if each cut were represented separately, while still allowing $O(n)$ time to retrieve a minimum cut for each pair of nodes.

We show that a collection of $n(n-1)$ cuts with the above property exists. This collection of cuts can be represented by a family of n trees $T_1,\ldots,T_n$, one for each node. For a given node s, T_s is a directed weighted cut tree rooted at s that contains all information concerning *flow values* and *minimum cuts* directed from s to any other node in the graph. That is, for any node t, if e is the minimum weight edge on the path from s to t in T_s, then the maximum flow from s to t in G is the weight of e, and the two components of $T_s - e$ form a minimum cut that separates s from t in G.

The Family of Trees

Definition: For an arbitrary vertex $s \in V$, define the edge-weighted graph $G^s = (V, E^s)$ as follows:

$E^s = E \cup \{(u,s) | u \in V, u \neq s\}$, where $w_{us} = \infty$. That is, in addition to the original set E, all nodes but s are connected to s by an edge, directed into s, with a very large capacity. Let $f^s(s,j)$ denote the value of the maximum flow from s to j in G^s, and let $C^s(s,j)$ denote the minimum cut separating s from j in G^s. The following lemma is straightforward:

Lemma 4.1 *A cut C is a minimum (directed) cut separating s from j in G^s iff it is a minimum cut that separates s from j in G. Also, $\beta^s(s,j) \equiv \min\{f^s(s,j), f^s(j,s)\} = f(s,j)$. Hence, any (s,j) β-cut in G^s is a minimum (s,j)-cut in G.*

Theorem 4.1 *For any node s in a directed graph G, there exists a cut tree T_s that represents minimum cuts for all oredered pairs (s,j).*

Proof: Consider the β cut tree of G^s. It represents, for any node j, one (s,j) β-cut in G^s and the value $\beta^s(s,j)$. But, from Lemma 4.1, an (s,j) β-cut in G^s is a minimum (s,j) cut in G, and $\beta^s(s,j) = f(s,j)$. Hence, the β cut tree of G^s is the desired cut tree T_s. □

Example: Consider the graph from Figure 2. If we let $s = b$, then the β cut tree T_b of G^b is the tree shown in Figure 4(i).

Theorem 4.1 implies that we can apply Schnorr's algorithm to the transformed graph G^s to obtain T_s. However, Schnorr's method solves $O(n \log n)$ maximum flow problems although the tree represents only n-1 relevant minimum cuts. We next show that, in this special case, Schnorr's algorithm can be sped up and that T_s can be constructed with only $n-1$ flow computations. As opposed to Schnorr's method, in our algorithm, instead of picking *any* supernode U in $\mathcal{N}_k$ for further partitioning, we pick at each stage the unique supernode that contains s. The other supernodes are never partitioned further. Therefore, the final tree T_s may contain supernodes which do not contain s but which contain more than one node (the cut tree shown in Figure 4(ii) for $s = b$ is such a tree). Our improvement of Schnorr's algorithm for this special case is based on the following lemma (given without a proof):

Lemma 4.2 *Let $U \subseteq V$ with $s \in U$, and let $(A, \overline{A})$ be a minimum U cut in G^s. Then there is a node $j \in U$ such that $(A, \overline{A})$ is an (s,j)-minimum cut in G.*

Lemma 4.2 suggests that if $s \in U$, then any minimum U cut in G^s is an (s,j) minimum cut in G for some $j \in U$. Thus, to find a minimum U cut in G^s we would simply have to find the u that minimizes $f(s,u)$ over all u's in U, and for this u any minimum (s,u) cut in G will be an appropriate choice.

Our algorithm uses a representation, due to J.C. Picard and M. Queyranne [PQ], that represents all minimum (s,t) cuts for a given pair of nodes (s,t). We outline their method and its main features here:

At a cost of one flow computation from s to t plus an additional $O(m)$ time one can represent *all* minimum (s,t) cuts in a DAG (Directed Acyclic Graph) of size $O(n)$. We denote it by $DAG_{s,t}$. Each node ("supernode") in $DAG_{s,t}$ corresponds to a set of nodes in the original graph, and these supernodes partition the original node set of G. The supernode containing t is of in-degree 0, the supernode containing s is of out-degree 0.
There is a 1-1 correspondence between the set of all closed sets in the DAG and the set of all minimum (s,t) edge cuts in G, where a closed set C is a set of nodes such that if $i \in C$ then $successors(i) \subseteq C$. A cut $(S, \bar{S})$ separating s from t is a minimum (s,t)-cut if and only if S is a closed set in the DAG containing s and not t.

Given a maximum flow f from s to t on the graph $G = (V, E)$, the DAG is constructed from the augmentation graph $H = (V, E')$ defined by f. The DAG is obtained from H by collapsing s and $successors(s)$ into a single supernode, collapsing t and $predecessors(t)$ into a single supernode, collapsing each of the remaining strongly connected components into a supernode, while maintaining the relations between supernodes to be as in H. Given such a representation, it is possible to select, in $O(n)$ time, one (s,t) cut. In particular, it is easy to select one (s,t) cut which does not split certain subsets of nodes, if such a cut exists. In this algorithm we use the DAGs to be able to select cuts with this property.

Algorithm for Constructing T_s :

Input : A directed graph G,
Output : A cut tree T_s.

- Start by computing the flows $f(s,v_1), f(s,v_2), \ldots, f(s,v_n)$ and their corresponding DAGs $DAG_{s,v_1}, \ldots, DAG_{s,v_n}$ which represent all (s,v_i) minimum cuts in G.
- Let $f(s,v_1) = \alpha_1 = \min_{v \in V}\{f(s,v)\}$. From DAG_{s,v_1}, find one (s,v_1)-minimum cut $(A,\overline{A})$; Let the tree $\mathcal{N}_1$ be represented by an edge $A \rightarrow \bar{A}$ of weight α_1.
- Given $\mathcal{N}_k$, construct $\mathcal{N}_{k+1}$ as follows: Let U be the supernode in $\mathcal{N}_k$ that contains s. If $U = \{s\}$ then $T_s \leftarrow \mathcal{N}_k$. Otherwise, suppose that $Y_1, \ldots, Y_l$ are the components created by the removal of the l edges that are attached to U in $\mathcal{N}_k$.
  1. Find $\alpha_{k+1} = f(s,v_{k+1}) = \min_{u \in U}\{f(s,u)\}$ by lookup from the values already computed.
  2. For each $j = 1,\ldots,l$, contract the nodes of Y_j in $DAG_{s,v_{k+1}}$ into a single node.
  3. Find a minimum (s,v_{k+1}) cut in the contracted $DAG_{s,v_{k+1}}$. Let this cut be $(X_{k+1}, \overline{X_{k+1}})$, directed from X_{k+1} to $\overline{X_{k+1}}$. Due to the contraction, the cut $(X_{k+1}, \overline{X_{k+1}})$ is an (s,v_{k+1}) minimum cut which does not cross any Y_j, i.e. any Y_j lies either in X_{k+1} or in $\overline{X_{k+1}}$.
  4. Update the tree by splitting U into two nodes $(X_{k+1} \cap U)$ and $(\overline{X_{k+1}} \cap U)$ and adding an edge $(X_{k+1} \cap U) \longrightarrow (\overline{X_{k+1}} \cap U)$ with weight α_{k+1}, and reconnect the components $Y_1, \ldots, Y_l$ according to the following rules: if Y_j lies in X_{k+1} then connect it by $Y_j \longleftarrow (X_{k+1} \cap U)$, and if Y_j lies in $\overline{X_{k+1}}$ then connect it by $Y_j \longleftarrow (\overline{X_{k+1}} \cap U)$; maintain the previous weights on these edges.

We state, without a proof, that

Theorem 4.2 *Using the above algorithm, T_s can be computed by solving $(n-1)$ maximum flow problems.*

Example: In Figure 4 we show two cut trees for G^b, where G is the graph depicted in Figure 2. Either one of the trees can be obtained by our algorithm, depending on the cuts that are selected at each step. The tree in 4(i) was obtained by the following sequence of cuts (directed from left to right): $(bcdef, a)$, $(abcde, f)$, $(bdef, ac)$, $(abcef, d)$, $(bd, acef)$. The tree in 4(ii) was obtained by the sequence: $(bcde, af)$, $(abdef, c)$, $(abcef, d)$, (bcd, aef). Note that it contains a supernode with two nodes.

The directed analogue for the Gomory-Hu tree is, therefore, a family of n partial cut trees $T_1, \ldots, T_n$, one for each node, where T_i corresponds to minimum cuts and flows originated at i. Given a pair of nodes $\langle i,j \rangle$, $f(i,j)$ and one minimum cut separating i from j can be obtained from T_i in the usual way.

5 Extending Cut Trees for the All Minimum Cut Problem

Cut trees represent a single minimum cut for each pair of nodes and can be constructed with n-1 flow computations, whereas for any fixed pair of nodes (s,t), a single max flow computation with an additional strongly connected components computation suffice to obtain a compact representation of the set of all min (s,t)-cuts, the Picard-Queyranne DAG. In this section we show an extension of cut trees that

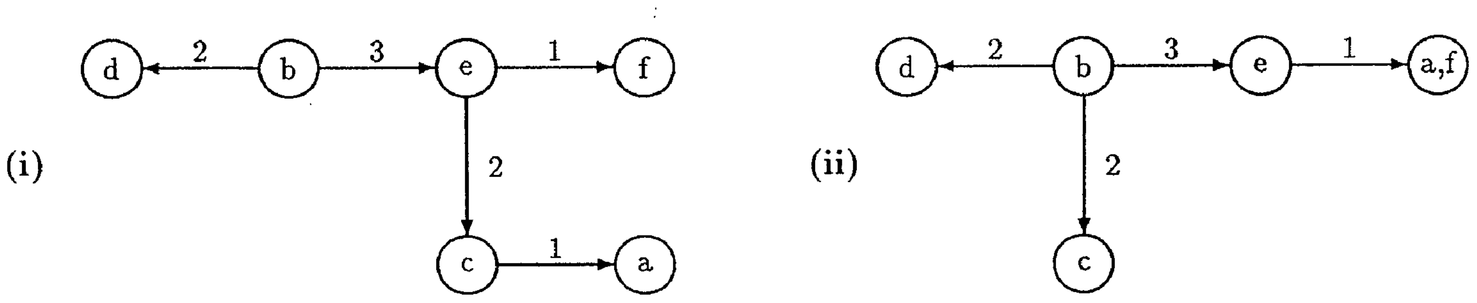


Figure 4: *Two possible cut trees for the directed graph G^b, where G is the graph shown in Figure 2. The tree (i) is a complete β cut tree of the graph G^b, and the tree (ii) contains all the relevant flow values and cuts directed from b.*

can be used to obtain a representation of *all* minimum cuts for *any* pair of nodes. We outline the method; its full description is given in [GN1].

We show how an *Extended Cut Tree T* can be used to obtain all $\binom{n}{2}$ DAGs, one for each pair of nodes, by using only $n-1$ flow computations (that are needed for the cut tree construction) plus $O(m)$ work per DAG. This is a large improvement over the obvious approach which requires one flow computation to construct each DAG. We show that the $\binom{n}{2}$ DAGs can be constructed from a sequence of $n-1$ flow computations $f(s,t)$, where each (s,t) corresponds to an edge in the cut tree T. We consider only undirected graphs and their Gomory-Hu cut trees; however, the same technique is applicable for the other types of graphs discussed earlier and their corresponding cut trees.

Consider an undirected weighted graph $G = (V, E)$. Let $C_{s,t}$ be the set of all minimum (s,t) edge-cuts. Also define C_{MIN} as the set of all cuts in G which are minimum (a,b)-cuts for *some* pair of nodes (a,b), i.e. $C_{MIN} = \cup_{(a,b)\in V\times V} \{C_{a,b}\}$. Define the *extended cut tree T* as a cut tree of G, associated with $n-1$ DAGs, one for each edge; that is, suppose that in addition to the cut tree we are given the DAG representation of $C_{s,t}$, $DAG_{s,t}$, for each tree edge $(s,t) \in T$. We first show that for any (a,b) not an edge in T, any cut in $C_{a,b}$ is in $C_{s,t}$ for some tree edge $(s,t) \in T$; we give a method to generate any cut in $C_{a,b}$ exactly once only from $C_{s,t}$'s for tree edges $(s,t) \in T$. Then we claim that for any (a,b) not an edge in T, $DAG_{a,b}$ that represents $C_{a,b}$ can be efficiently constructed from the extended cut tree T. The details of this construction are given in [GN1].

Definition : Let $DAG_{s,t}(X,Y)$, $X,Y \subseteq V$, denote the DAG resulting from $DAG_{s,t}$ by contracting the set X with all its successors, and contracting the set Y with all its predecessors.

As $s \in successors(u)$ and $t \in predecessors(u)$ $\forall u$, the supernode of out-degree 0 in $DAG_{s,t}(X,Y)$ contains $\{s \cup X\}$, and the supernode of in-degree 0 in $DAG_{s,t}(X,Y)$ contains $\{t \cup Y\}$. It is not difficult to see that any closed set in $DAG_{s,t}(X,Y)$ is a closed set in $DAG_{s,t}$; in fact, any closed set in $DAG_{s,t}(X,Y)$ which contains s but not t, is a closed set in $DAG_{s,t}$ which contains $\{s \cup X\}$ but not $\{t \cup Y\}$. Therefore, $DAG_{s,t}(X,Y)$ represents all min (s,t)-cuts in which the set X appears on the s side and the set Y appears on the t side of the cut.

Theorem 5.1 *For a fixed pair of nodes (a,b), let $(x_1,y_1), \ldots, (x_k,y_k)$ be the k minimum-weight tree edges along the path between a and b in T. Let Z_i ={the nodes between y_i and x_{i+1} along the path, including y_i}, and define $y_0 = a$, $x_{k+1} = b$ (see Figure 5). Then,*

1. *A cut $(A, \bar{A})$ is a min (a,b)-cut if and only if it appears (as a closed set) in at least one $DAG_{x_i,y_i}(a,b)$ for some $1 \le i \le k$.*
2. *Let $A_i = \{y_0, y_1, \ldots, y_{i-1}\}$; then, any mini-*

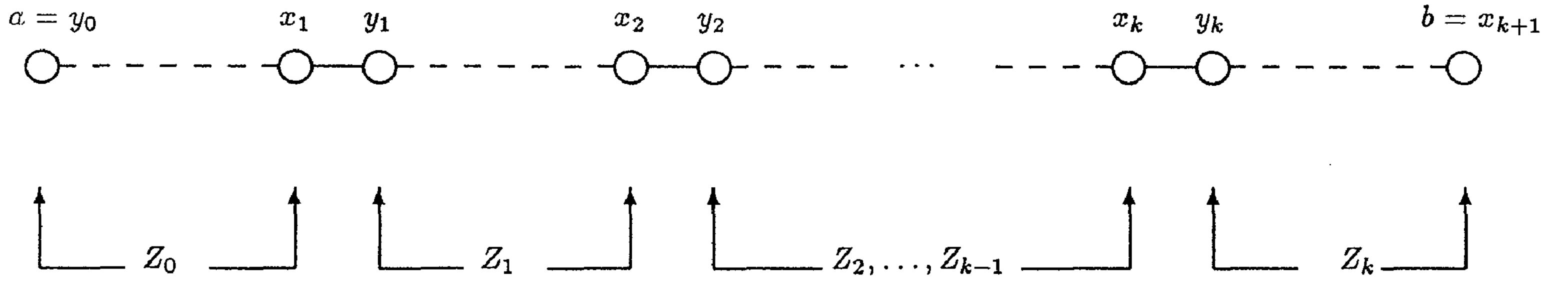


Figure 5: *The Path between a and b in an Equivalent Flow Tree.* $(x_1, y_1), \ldots (x_k, y_k)$ *are the k Minimum-Weight Edges along the Path.*

mum (a,b)-cut appears (as a closed set) in exactly one $DAG_{x_i,y_i}(A_i, b)$ *for some* $1 \le i \le k$.

Proof : (1) The **only if** part follows directly from the definition of $DAG_{x_i,y_i}(a,b)$, because any closed set in $DAG_{x_i,y_i}(a,b)$ is a minimum (x_i,y_i)-cut which separates a from b, and since $\forall i \;\; f(a,b) = f(x_i,y_i)$, it is a minimum (a,b)-cut.

For the **if** part, note that any min (a,b)-cut $(A, \bar{A})$ must contain the entire set Z_i in one or the other side of the cut; Z_i's nodes never split since the minimum cut value between nodes within Z_i is greater than the value of the cut $(A, \bar{A})$. As $a \in Z_0$ and $a \in A$, $Z_0 \subseteq A$; similarly, $Z_k \subseteq \bar{A}$. Let Z_j be the set with smallest index which lies in $\bar{A}$ (such j exists since $Z_k \subseteq \bar{A}$). As $Z_{j-1} \subseteq A$ and $Z_j \subseteq \bar{A}$, $(A, \bar{A})$ is a minimum (x_j, y_j)-cut and a minimum (a,b)-cut, and therefore it appears as a closed set containing $\{x_j, a\}$ and not $\{y_j, b\}$ in $DAG_{x_j,y_j}(a,b)$. Hence, any min (a,b)-cut is a closed set in $DAG_{x_i,y_i}(a,b)$ for some $1 \le i \le k$.

(2) $DAG_{x_i,y_i}(A_i, b)$ contains all (x_i, y_i) min-cuts in which $\{a, y_1, \ldots, y_{i-1}\}$, and hence $Z_0, \ldots, Z_{i-1}$, appear on the a side and $\{y_i, b\}$, and hence Z_i, Z_k, appear on the b side of the cut. That is, $DAG_{x_i,y_i}(A_i, b)$ contains all cuts of value $f(a,b)$ which separate $\{Z_0, \ldots, Z_{i-1}\}$ from $\{Z_i, Z_k\}$. From the argument of (1), any minimum (a,b)-cut separates $\{Z_0, \ldots, Z_{j-1}\}$ from Z_j for a *unique* j, and thus this cut appears in exactly one of these DAGs. □

Theorem 5.1 implies the following (a proof is given in [GN1]):

Corollary 5.1 *All cuts in* C_{MIN} *can be generated in* $O(nm + n \cdot |C_{MIN}|)$ *time as follows :*

For any distinct weight w *in the tree, let* $D_{(s_1,t_1)}, \ldots, D_{(s_h,t_h)}$ *be the DAGs which correspond to all edges in* T *with the weight* w.

For i=1 to h, contract s_1 *with* t_1 *, ..., *s_{j-1} *with* t_{j-1} *in* DAG_{s_j,t_j} *(this is the usual type of contraction) and generate the closed sets in the resulting DAG.*

Theorem 5.1 showed that $C_{a,b}$ can be enumerated using the extended cut tree T. As the size of $C_{a,b}$ can be exponential in n, it is often of more use to produce only the $DAG_{a,b}$ which compactly represents the min (a,b)-cuts. We know that the DAGs associated with the edges of minimum weight along the (a,b) path in T contain the needed information - they contain all (a,b)-cuts. But there may be an exponential number of minimum-(a,b) cuts, so how can we use those DAGs to efficiently create the (a,b)

DAG? In [GN1] it is shown that this can be done efficiently, that is

Theorem 5.2 *Given an extended cut tree T, $DAG_{a,b}$, for* all $\binom{n}{2}$ *pairs (a,b), can be found in $O(n^2m)$ time, which implies that the amortized cost for constructing a single DAG is $O(m)$.*

References

[Cha] B. Chazelle, *Computing on a Free Tree via Complexity-Preserving Mappings,* Algorithmica (1987) Vol. 2, 337-361.

[ChHu] C.K. Cheng, T.C. Hu, *Maximum Concurrent Flow and Minimum Ratio Cut,* Tech. Report CS88-141 UC San Diego, Dec. 1988.

[FF] L. R. Ford, D. R. Fulkerson, *Flows in Networks,* Princeton University Press, Princeton N.J. 1962.

[FrFr] H. Frank, I.T. Frisch, *Communication, Transmission and Transportation Networks,* Addison-Wesley, 1971.

[GH] R. E. Gomory, T. C. Hu, *Multi-Terminal Network Flows,* Siam J. Appl. Math., Vol. 9 (1961), 551-560.

[GrH] F. Granot, R. Hassin, *Multi-Terminal Maximum Flows in Node-Capacitated Networks,* Discrete Applied Math., Vol. 13 (1986), 157-163.

[G] D. Gusfield, *Very Simple Methods for All Pairs Network Flow Analysis,* Siam J. Computing, to appear.

[GN1] D. Gusfield, D. Naor, *Extracting Maximal Information about Sets of Minimum Cuts,* Tech. Report CSE-88-14, UC Davis.

[GN2] D. Gusfield, D. Naor, *Efficient Algorithms for Generalized Cut Trees,* Tech. Report CSE-89-5, UC Davis.

[H] T. C. Hu, *Integer Programming and Network Flows,* Addison-Wesley, 1969.

[HT] D. Harel, R.E. Tarjan, *Fast Algorithms for Finding Nearest Common Ancestors,* Siam J. Computing, vol. 13 (1984), 338-355.

[PQ] J. C. Picard, M. Queyranne, *On the Structure of All Minimum Cuts in a Network and Applications,* Mathematical Programming Study 13 (1980), 8-16.

[SV] B. Schieber, U. Vishkin, *On Finding Lowest Common Ancestors: Simplification and Parallelization,* Siam J. Computing, Vol. 17, 6, (1988) 1253-1262.

[Sch] C. P. Schnorr, *Bottlenecks and Edge Connectivity in Unsymmetrical Networks,* Siam J. Computing, 1979, 265-274.

[T] R. E. Tarjan, *Applications of Path Compression on Balanced Trees,* J. ACM, Vol. 26, 4 (1979), 690-715.