Applications of a Poset Representation to Edge Connectivity and Graph Rigidity

Harold N. Gabow*

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

hal@cs.colorado.edu

Abstract.

This paper investigates a poset representation for a family of sets defined by a "labelling algorithm". It gives poset representations for the family of minimum cuts of a graph and shows how to compute them quickly. For undirected graphs a representation was known but we find it faster. For directed graphs we know of no previous compact representation. The representations are the starting point for algorithms that increase the edge connectivity of a graph, from λ to a given target $\tau = \lambda + \delta$, adding the fewest edges possible. For undirected graphs the time bound is essentially the best-known bound to test τ-edge-connectivity; for directed graphs the time bound is roughly a factor δ more. We also construct poset representations for the family of rigid subgraphs of a graph, when graphs model structures constructed from rigid bars. The link between these problems is that they all deal with graphic matroids.

1. Introduction.

Algorithms for optimization problems like network flow, matroid intersection and many others are often based on a labelling procedure. The labelling procedure can often be adapted to carry out sensitivity analysis, or even give a representation of all solutions, often as a poset. This paper investigates the poset representation for problems related to graphic matroids. We obtain representations for the family of minimum cuts of a graph. We use these representations as the starting point of efficient algorithms to increase the edge connectivity of a graph. We also construct representations for the family of rigid subgraphs of a graph, when graphs model structures constructed from rigid bars.

* Research supported in part by NSF Grant No. CCR-8815636.

To elaborate, consider a family $\mathcal{F}$ of sets such that each element is in a unique minimal set. Thus each element e is in a set $M(e)$ of $\mathcal{F}$ such that any set of $\mathcal{F}$ containing e contains $M(e)$. An obvious example of $\mathcal{F}$ is a family closed under intersection. In general any set of $\mathcal{F}$ is a union of sets $M(e)$, and the sets $M(e)$ can be represented as follows. For any element e let $[e]$ be the set of all elements f with $M(e) = M(f)$. Define a partial order P whose elements are the sets $[e]$, with $[f]$ a successor of $[e]$ if $M(f) \subseteq M(e)$. Thus $M(e)$ consists of all elements in successors of $[e]$. P is called the *poset representation of* $\mathcal{F}$. Often it is unnecessary for P to have all transitive edges. To emphasize that all transitive edges are explicitly included we write P^c; P^r denotes the transitively reducted poset; P denotes any relation whose transitive closure is P^c.

The poset representation is precisely the minimal difference poset in [GI] used to represent all stable marriages; it is the representation of Picard and Queyranne for all minimum cuts of a flow network [PQ]; it is the Dulmage-Mendelsohn representation for maximum bipartite matchings [DM]. It is closely related to the representation of Nakamura for all rigid subgraphs of a bar-and-joint structure [N] and to various notions of the principal partition of a matroid or polymatroid [NakI]. Section 3 suggests that the source of this representation may be broader – graph and network labelling algorithms often induce such posets.

Consider first edge connectivity. Fix a vertex a. An *a-set* is a nonempty set of vertices not containing a. An *a-cut* is the set of edges directed from $V - S$ to S for an a-set S. An *a-mincut* is an a-cut containing the smallest possible number of edges (a is fixed). In this case the set S is an *a-minsink*.

The *edge connectivity* of a directed graph G, denoted $\lambda(G)$ or λ, is the smallest cardinality of an a-cut, when a ranges over all vertices. Such a set of λ edges is a *mincut*. For any a, an a-mincut in G or in G with all edges reversed is a mincut. The edge connectivity

of an undirected graph is the same as that of the corresponding directed graph having each edge oriented in both directions. We often refer to a *spanning tree* of a directed graph – for this, ignore edge directions.

Edmonds established the relation between edge connectivity and graphic matroids: Fix a vertex a and integer k. A set of $k(n-1)$ edges T is a *complete intersection* (for a and k) if it contains precisely k edges directed to each vertex except a, and it can be partitioned into k spanning trees. (T is a maximum cardinality matroid intersection of two matroids, one of which is derived from the graphic matroid of G.) *Edmonds' Theorem* states that every a-cut has at least k edges if and only if G has a complete intersection for a and k [E69, E72]. This relation is the basis of an algorithm that finds an a-mincut in time $O(\lambda m \log(n^2/m))$, the best-known bound [G91a].

We investigate the poset representation for a-minsinks. We show that in an undirected graph the poset explicitly gives *all* the mincuts. Another representation is known, the cactus representation of Dinits, Karzanov and Lomosonov [DKL]. Our poset is either a tree or more generally a graph we call an m-tree, which allows limited sharing of children. The m-tree is easily converted to the cactus representation, although our applications can use either. Our algorithm finds the m-tree representation in time $O(m+\lambda^2 n \log(n/\lambda))$. This improves the algorithm of [KT] that constructs the cactus representation in time $O(\lambda n^2)$ (note $\lambda \leq n$).

The poset representation also gives a representation of the a-minsinks of a directed graph. We know of no previous compact representation nor any striking properties of the poset. We construct this poset in time $O(\lambda m \log(n^2/m) + n^2)$. However various properties of the poset can be determined in time $O(\lambda m \log(n^2/m))$ and this suffices for our applications.

The representation of all a-minsinks provides the starting point for our algorithms to increase edge connectivity. In the *edge connectivity augmentation problem* we are given a graph G with edge connectivity λ. The task is to increase the edge connectivity by some positive quantity δ to a target connectivity $\tau = \lambda + \delta$, adding as few edges as possible. The solution graph is allowed to have parallel edges.

We present efficient algorithms for both directed and undirected graphs. The high-level approach of both algorithms is that of Naor, Gusfield and Martel [NGM] for undirected graphs: repeatedly increase the connectivity by one, using a good representation of all mincuts. Toward this end we solve the $\delta = 1$ augmentation problem for directed graphs in time $O(\lambda m \log(n^2/m))$. This problem is easier for undirected graphs: [NGM] shows it can be done by a depth-first traversal of the representation of all mincuts, so our poset representation solves the undirected problem in time $O(m + \lambda^2 n \log(n/\lambda))$. These two bounds for the $\delta = 1$ augmentation problem are precisely the best-known bounds to check λ-edge-connectivity [G91a].

Naor et. al. solve the general augmentation problem for undirected graphs in time $O(\delta^2 nm + n^{5/3}m)$ [NGM]. We improve this to $O(m + \tau^2 n \log n)$, close to the best-known bound to test τ-edge-connectivity. Frank solves the augmentation problem for directed multigraphs in time $O(n^5)$ [F] (this allows edges with arbitrarily large multiplicities). Our algorithm for directed graphs uses time $O(\delta\tau(m+\delta n)\log^2 n)$ and space $O(m+\delta\tau n)$ space. For example when $\delta = O(1)$ this is within a logarithmic factor of the best-known bound to test τ-edge-connectivity. A more space-efficient implementation uses time $O(\delta^2\tau(m+\delta n)\log^2 n)$ and space $O(m+\delta n)$. Both [F] and [NGM] use network flow as the main technical tool; we use the edge connectivity algorithm of [G91a].

Our poset representation algorithm is stated as a combination of depth-first search plus the labelling algorithm for the problem at hand. Our specific implementation is for labelling algorithms related to graphic matroids like the minsink poset above. We obtain efficient poset representation algorithms for other problems related to graphic matroids: The poset representation is important in understanding the rigidity properties of bar-and-body frameworks and bar-and-joint frameworks. For both we construct P^c and P^r for an n vertex graph in time $O(n^2)$. One of these four bounds was known.

The rest of this abstract is organized as follows. Section 2 gives an algorithm to increment the connectivity of a directed graph by one. It assumes certain characteristics of the poset representation of all minsinks can be found efficiently. Section 3 gives an algorithm that finds the nodes of the poset representation, in general and for the minsink poset. This completes the algorithm of Section 2. Section 4 discusses combinatoric properties of posets related to graphic matroids, in particular the minsink poset for undirected graphs. Section 5 presents algorithms to construct the complete poset, including the minsink posets and posets for graph rigidity. Section 6 solves the edge connectivity augmentation problem for undirected graphs. It also highlights the more complicated augmentation algorithm for directed graphs. Sections 3–5 and the directed graph algorithm are highly abbreviated due to space limitations; [G91b] contains a

complete development. The rest of this section gives notation and definitions.

Consider sets S and T in a universe U. If e is an element then $S+e$ denotes $S \cup \{e\}$ and $S-e$ denotes $S-\{e\}$. We use both set containment $S \subseteq T$ and proper set containment $S \subset T$. Sets S and T *nest* if $S \subseteq T$ or $T \subseteq S$. The sets *meet* if $S \cap T$ is nonempty. The sets *cross* if each set $S-T$, $T-S$, $S \cap T$ and $U-S-T$ is nonempty. A *subpartition* of a set S is a collection of disjoint subsets of S.

For any graph G, $V(G)$ and $E(G)$ denote the vertex set and edge set of G, respectively. V and E denote these sets for the input graph of a problem, and $n = |V|$, $m = |E|$. For vertices v and w, the notation vw denotes an undirected edge joining v and w, or a directed edge from v to w; it will be clear from context which is meant.

Consider a directed graph. For $S \subseteq V$, $\rho(S)$ denotes the set of all edges directed from $V-S$ to S. The function δ is analogous to ρ for edges directed from a set; thus $\delta(S) = \rho(V-S)$. For any sets of vertices X and Y, $|\rho(X)|+|\rho(Y)| \geq |\rho(X \cap Y)|+|\rho(X \cup Y)|$; a similar inequality holds if each ρ is changed to δ. These two inequalities are referred to as *submodularity*.

Consider an undirected graph. For $S \subseteq V$, $d(S)$ denotes the number of edges joining $V-S$ and S. In addition to the submodular identity d satisfies the following inequality: For any sets of vertices X and Y, $d(X)+d(Y) \geq d(X-Y)+d(Y-X)$.

We do not use results from matroid theory although there is a close relationship. For instance a set of edges that can be partitioned into k forests is precisely an independent set in $\mathcal{G}^k$, the matroid sum of k copies of the graphic matroid of G. This notion appears in Edmonds' Theorem and is used throughout this paper.

2. Incrementing directed edge connectivity.

In this section we are given a directed graph G with edge connectivity λ. We wish to increase the connectivity by one, adding as few edges as possible. We present an algorithm that combines ideas of [F] and [NGM].

An *a-minsource* is an a-set S that has $|\delta(S)|$ minimum. A *minsource* is a set of vertices S that has $|\delta(S)| = \lambda$. Frank's solution to the general edge connectivity augmentation problem specializes to the following result for $\delta = 1$: The minimum number of edges needed to increase the edge connectivity by one is precisely the maximum number of pairwise-disjoint sets that are all minsources or all minsinks. We begin with several lemmas that provide an alternate proof of this result (essentially Lemma 2.4). In addition the lemmas form the basis of our algorithm.

Define a *cover* to consist of two sets $\mathcal{X}$ and $\mathcal{Y}$ where $\mathcal{X}$ meets every minsource and $\mathcal{Y}$ meets every minsink. The first lemma is the basis of an algorithm that uses a cover to increase the edge connectivity by one. It needs one more concept. Given a cover, for each vertex $y \in \mathcal{Y}$ define $Y(y)$ as the maximal minsink satisfying $Y(y) \cap \mathcal{Y} = \{y\}$; if a unique such set does not exist then $Y(y) = \emptyset$. If $|\mathcal{Y}| = 1$ we may have $Y(y) = \emptyset$ because there are many such sets; we will only use $Y(y)$ when $|\mathcal{Y}| > 1$. In that case, if some minsink S satisfies $S \cap \mathcal{Y} = \{y\}$ then set $Y(y)$ is well-defined. This follows because the family of minsinks S satisfying $S \cap \mathcal{Y} = \{y\}$ is closed under union. (This is proved using submodularity and the assumption $|\mathcal{Y}| > 1$.) Similarly define $X(x)$ for $x \in \mathcal{X}$.

Lemma 2.1. Let $\mathcal{X}$, $\mathcal{Y}$ be a cover.

(*i*) Adding all edges xy for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ gives a graph with connectivity $\lambda+1$.

(*ii*) Let $|\mathcal{X}|$, $|\mathcal{Y}| > 1$. Adding an edge xy with $x \in \mathcal{X}-Y(y)$ and $y \in \mathcal{Y}-X(x)$ gives a graph with cover $\mathcal{X}-x$, $\mathcal{Y}-y$.

Proof. (*i*) It suffices to show that for each minsink S of G, $\rho(S)$ contains an added edge. Clearly S contains some $y \in \mathcal{Y}$. Since $V-S$ is a minsource, it contains some $x \in \mathcal{X}$. Hence edge $xy \in \rho(S)$.

(*ii*) Let S be a minsink of G with $S \cap \mathcal{Y} = \{y\}$. As noted above $S \subseteq Y(y)$. Thus $x \notin S$ and S is not a minsink after xy is added. (This argument is vacuous if $Y(y) = \emptyset$.) ∎

The lemma motivates the following *basic procedure*. Given a cover $\mathcal{X}$, $\mathcal{Y}$, it increases the connectivity by one by adding $\max\{|\mathcal{X}|, |\mathcal{Y}|\}$ edges: While both $|\mathcal{X}|$ and $|\mathcal{Y}|$ are larger than 1, apply Lemma 2.1(*ii*). When $|\mathcal{X}|$ or $|\mathcal{Y}|$ equals 1, apply Lemma 2.1(*i*).

We show the basic procedure is correct by proving that vertices x, y of Lemma 2.1(*ii*) always exist. The main reason is the following property.

Lemma 2.2. Let $\mathcal{X}$, $\mathcal{Y}$ be a cover with $|\mathcal{X}|, |\mathcal{Y}| > 1$. For any vertex $y \in \mathcal{Y}$, $y \in X(x)$ implies $X(x)$ and $Y(y)$ nest. Similarly for any $x \in \mathcal{X}$.

Proof. Suppose $y \in X(x)$. We show that if $X(x)$ and $Y(y)$ do not nest then

$$|\rho(Y(y)-X(x))| = \lambda. \qquad (1)$$

But this means $Y(y)-X(x)$ is a minsink disjoint from $\mathcal{Y}$, a contradiction.

First suppose that $X(x)$ and $Y(y)$ cross. Thus $X(x)$ and $V-Y(y)$ are two minsources that cross. Submodularity implies their union is a minsource. This is equivalent to (1).

Next suppose that $X(x) \cup Y(y) = V$. Then (1) follows since $X(x)$ is a minsource. ∎

Now we show the basic procedure is correct. It suffices to show that the vertices x and y of Lemma 2.1(ii) always exist. Choose any $x \in \mathcal{X}$. Since $X(x)$ is a minsource, its complement is a minsink, whence there is some $y \in \mathcal{Y} - X(x)$. (The conclusion is true even if $X(x) = \emptyset$.) If $x \notin Y(y)$ then x, y is the desired pair. Otherwise $x \in Y(y)$. There is some $x' \in \mathcal{X} - Y(y)$. Clearly $y \notin X(x')$ since otherwise Lemma 2.2 shows $x \in Y(y) \subseteq X(x')$, contradicting the definition of $X(x')$. Thus the basic procedure is correct.

We will find the smallest number of edges to add by finding a smallest cover. Using terminology similar to [NGM], an *in-extreme set* is a set S that is a minimal minsink, i.e., $|\rho(S)| = \lambda$ but no proper subset satisfies this condition. Similarly for *out-extreme set*; *extreme set* refers to either. (In Section 6 these terms have a more general meaning.) Clearly in the definition of cover we can replace minsinks by in-extreme sets and minsources by out-extreme sets.

Lemma 2.3. Either (i) all in-extreme sets are pairwise disjoint, or (ii) any two in-extreme sets meet and cover V. Similarly for out-extreme sets.

Proof. First observe that if S and T are in-extreme sets that meet then $S \cup T = V$. For S and T do not nest, by definition of in-extreme. They do not cross, since submodularity would imply that $S \cap T$ is in-extreme, a contradiction.

Next observe that if an in-extreme set S meets another in-extreme set T then it meets every in-extreme set. For if U is in-extreme and disjoint from S then $U \subset T$, a contradiction. ∎

Let a_y be a vertex in some in-extreme set. Form the set $\mathcal{Y}$ by including a_y plus an arbitrarily chosen vertex from each in-extreme a_y-set. (Note that Lemma 2.3 shows for any vertex a, the a-sets that are in-extreme are pairwise disjoint. Possibly there are none.) Similarly form $\mathcal{X}$, with vertex a_x, for out-extreme sets. Clearly $\mathcal{X}$, $\mathcal{Y}$ is a cover; call it an *extreme-set cover*.

Lemma 2.4. For any extreme set cover $\mathcal{X}$, $\mathcal{Y}$, $\max\{|\mathcal{X}|, |\mathcal{Y}|\}$ is precisely the minimum number of edges needed to increase the connectivity by one.

Proof. By the basic procedure and symmetry it suffices to show that $|\mathcal{Y}|$ edges are needed. This is obvious if Lemma 2.3(i) holds. Suppose Lemma 2.3(ii) holds. This implies $|\mathcal{Y}| \leq 2$. Clearly we can assume $|\mathcal{Y}| = 2$. Let S and T be two in-extreme sets corresponding to $\mathcal{Y}$. Sets $S - T$ and $T - S$ are disjoint minsources. Thus at least two edges are needed to increase the connectivity. ∎

We have shown that the basic procedure, executed on an extreme-set cover, solves our problem. We turn to making this algorithm efficient. We wish to avoid calculating the sets $X(x)$, $Y(y)$. We do this by maintaining a partition of each set $\mathcal{X} - a_x$ and $\mathcal{Y} - a_y$. We now define the partition of $\mathcal{X} - a_x$; that of $\mathcal{Y} - a_y$ is analogous. Say that vertex x *elicits* vertex y if any a_y-minsink containing x contains y. Each vertex $y \in \mathcal{Y} - a_y$ has an associated set $E(y)$. The sets $E(y)$ form a partition of $\mathcal{X} - a_x$ such that any $x \in E(y)$ elicits y.

The key property of the partition is that $Y(y) \subseteq E(y)$. This follows since $Y(y)$ is an a_y-minsink. Hence if $x \in Y(y) \cap E(y')$ then $y' \in Y(y)$. Since $Y(y) \cap \mathcal{Y} = \{y\}$ this implies $y' = y$ as desired. Note that in this argument all sets refer to the current graph.

This gives a simple rule for choosing vertices x and y as required by Lemma 2.1(ii): Choose x and y so that $x \notin E(y)$ and $y \notin E(x)$. Clearly this implies $x \notin Y(y)$ and $y \notin X(x)$ as desired.

The main tool to implement the algorithm is the poset representation P of all a-minsinks. It is easy to see that the in-extreme sets not containing a are precisely the sinks of P. Now we present the entire algorithm.

Initialize Step. Find an in-extreme set and choose a vertex a_y in it. Let P_y be the poset for the a_y-minsinks. Form $\mathcal{Y}$ as a_y plus a vertex from each sink of P_y. Similarly define a_x, P_x and $\mathcal{X}$.

Partition $\mathcal{X} - a_x$: Place each $x \in \mathcal{X} - a_x$ in a set $E(y)$ where $y \in \mathcal{Y}$ and $[y]$ is a successor of $[x]$. (If x is not in any a_y-minsink then place x in an arbitrary set.) Similarly partition $\mathcal{Y} - a_y$.

Add_edges Step. Repeatedly execute the case below that applies until the Low Case halts.

High Case $\min\{|\mathcal{X}|, |\mathcal{Y}|\} \geq 4$. Choose vertices x and y so that $x \in E(y')$ for some $y' \neq y$ and $y \in E(x')$ for some $x' \neq x$. Add edge xy to the graph. Remove x from $\mathcal{X}$ and $E(y')$; remove y from $\mathcal{Y}$ and $E(x')$. Merge $E(y)$ into $E(y')$ and $E(x)$ into $E(x')$.

Middle Case $\min\{|\mathcal{X}|, |\mathcal{Y}|\} \in [2, 3]$. Find all sets $X(x), x \in \mathcal{X}$ and $Y(y), y \in \mathcal{Y}$. Choose vertices x and y so that $x \in \mathcal{X} - Y(y)$ and $y \in \mathcal{Y} - X(x)$. Add edge xy to the graph. Remove x from $\mathcal{X}$ and y from $\mathcal{Y}$.

Low Case $\min\{|\mathcal{X}|, |\mathcal{Y}|\} = 1$. Add all possible edges from a vertex of $\mathcal{X}$ to a vertex of $\mathcal{Y}$. Halt. ∎

Correctness follows from the preceding discussion plus the fact that the High Case updates the partition correctly. To prove this fact consider any $z \in E(x)$. Let S be an a_x-minsource containing z. Since z elicits x, $x \in S$. If S is a minsource after xy is added then $y \in S$. Since y elicits x', $x' \in S$ as desired.

The efficiency depends on further details of the implementation. We first discuss the High Case. The partition sets $E(x)$ and $E(y)$ are maintained using an algorithm for disjoint set merging. Observe that the desired vertices x and y can be found by considering 3 arbitrary vertices of $\mathcal{X} - a_x$ and 3 arbitrary vertices of $\mathcal{Y} - a_y$. These vertices form 9 pairs x, y, of which 6 can be disqualified by relations $x \in E(y)$. Hence the desired pair can be found among the chosen vertices. Thus it is easy to see that the total time for the High Case is the time for $O(n)$ *unions* and $O(n)$ *finds*, which is $O(n\alpha(n,n))$. (Another approach uses total time $O(n)$.)

The remaining details of the algorithm are implemented using the posets P_x and P_y. We do not construct these orders explicitly. Instead we rely on three operations on each poset representation P:

(*i*) Find the subpartition of $V(G)$ into the nodes of P.

(*ii*) For each node of P, find a successor that is a sink.

(*iii*) Find all nodes of P having only one successor that is a sink.

Note that operation (*ii*) identifies all the sinks. Operations (*i*) and (*ii*) suffice to implement the Initialize Step and (*iii*) suffices to implement the Middle Case.

Section 3 shows that each of the operations (*i*) – (*iii*) can be done in $O(m)$ time, given a complete intersection of λ trees. Such an intersection is found in time $O(\lambda m \log(n^2/m))$ using the round robin connectivity algorithm of [G91a].

Theorem 2.1. The directed edge connectivity augmentation problem for $\delta = 1$ can be solved in time $O(\lambda m \log(n^2/m))$ and space $O(m)$. ∎

3. The node-finding algorithm.

This section sketches an algorithm that finds the nodes $[e]$ of a poset representation. The node-finding algorithm is stated in general and a specific implementation is given for the minsink poset. Extensions determine other properties of the poset. Complete details are in [G91b].

We begin by characterizing the minsinks. Consider a directed graph G. Let k be the number of edges in an a-mincut. Let T be the complete intersection given by Edmonds' Theorem (Section 1). A set of vertices S is *cut by* T if $\rho(S) \subseteq T$ and *k-spanned by* T if each tree of T contains a spanning tree of S. (Note that a single vertex is k-spanned by T.) The following result is a simple extension of [G91a].

Lemma 3.1. The a-minsinks are precisely the a-sets that are cut and k-spanned by T. ∎

Let $\mathcal{F}$ denote the family of a-sets that are cut and k-spanned by T. It is easy to see that $\mathcal{F}$ is closed under intersection. Thus as discussed in Section 1 there is a poset representation of the a-minsinks.

Let us review and extend the notation of Section 1. For a vertex v, $M(v)$ denotes the smallest set of $\mathcal{F}$ that contains v. Extend this so for an edge e, $M(e)$ denotes the smallest set of $\mathcal{F}$ that contains both ends of e. We shall construct the sets $M(e)$. The sets of $\mathcal{F}$ containing at least two vertices are precisely the sets that can be written as $\bigcup\{M(e) \mid e \in R\}$ where the set of edges R forms a subtree of T_1. The set $M(v)$ for vertices v are easily derived from $M(e)$.

To construct the poset representation we give a general algorithm, illustrating it for the minsink poset. As in Section 1 consider a family of sets over a universe E, such that each $e \in E$ is in a unique minimal set $M(e)$. The sets $M(e)$ can be specified by a function $\mathcal{L} : E \to 2^E$ that gives the relations

$$M(e) = \{e\} \cup \bigcup\{M(f) \mid f \in \mathcal{L}(e)\}. \qquad (1)$$

A trivial choice of $\mathcal{L}$ is given by $\mathcal{L}(e) = M(e)$. We interpret (1) as defining the family $M(e)$, i.e., sets $M(e)$ are the smallest sets satisfying (1), and they are unique.

For many combinatorial problems the function $\mathcal{L}$ is given by an efficient labelling algorithm. For instance to represent all minimum cuts of a flow network, we use the standard Ford-Fulkerson labelling algorithm for maximum value network flow [PQ].

Given $\mathcal{L}$, define a directed graph L having vertex set E, and edges ef for $f \in \mathcal{L}(e)$. Thus $M(e)$ consists of all the successors of e in L. Let P denote the graph L with each strongly connected component contracted to a node. It is easy to see that the nodes of P are precisely the sets $[e]$ and P is the poset representation of the given family of sets.

To construct P we adapt Tarjan's strongly connected components algorithm [T] (unlike other algorithms Tarjan's uses adjacency lists in only one direction, which corresponds to the information given by $\mathcal{L}$). For the problems involving graphic matroids treated in this paper, the labelling functions are adapted from the labelling algorithm for the graphic matroid partitioning algorithm [R]. An efficient labelling algorithm is the *cyclic scanning rule* [GW]: Let the complete intersection T consist of trees T_i, $i = 0, \ldots, k-1$. For $e \in T_i$, let $\mathcal{L}(e)$ be the fundamental cycle of e in T_{i+1}, where addition is modulo k. Using this function $\mathcal{L}$ in (1) defines sets $M(e)$ as the smallest set containing both ends of e that is k-spanned by T. It is easy to extend this function to give the smallest set that is both cut and k-spanned

by T, i.e., the desired set $M(e)$. (The resulting $\mathcal{L}$ does not correspond to any matroid structure. So the poset representation goes beyond network and matroid algorithms.)

To compute $\mathcal{L}$ we use set merging data structures on the trees T_i, to avoid re-exploring tree edges and avoid generating edges not needed by the strong components algorithm. We use the static tree set merging algorithm of [GT] which performs n *union* and m *find* operations in time $O(m+n)$. Our basic algorithm computes the nodes $[e]$ of P. It is given a complete intersection T for a and λ and trees T_i.

Theorem 3.1. For a directed graph, let $\mathcal{F}$ be the family of a-minsinks. Given a complete intersection, the nodes of P can be found in $O(m)$ time. ∎

Similar set merging techniques can be used to implement operations (i) – (iii) of Section 2 and these operations:

(iv) For a given set $S \subseteq V(P)$, find all nodes of P having a proper successor in S.

(v) Find the maximal a-minsinks.

(iv) is used to find P^r in Section 4; (v) is used in the general directed augmentation algorithm; (i) – (iii) are used in several other algorithms.

Corollary 3.1. For a directed graph, let $\mathcal{F}$ be the family of a-minsinks. Given a complete intersection, each of the operations (i) – (v) can be done in $O(m)$ time. ∎

4. Poset properties.

This section summarizes combinatorial properties of poset representations for set families related to graphic matroids. Complete statements and proofs are in [G91b].

For directed graphs even with $m = O(n)$, P^r can require $\Omega(n^2)$ edges and there can be $\Omega(2^n)$ minsinks (there are simple examples of each). Although the obvious bound on $V(P)$ is $m+n$ the following holds.

Theorem 4.1. For a directed graph, let $\mathcal{F}$ be the family of a-minsinks. P has at most $2n$ nodes. ∎

We also show that P can be represented in one tree T_i of a complete intersection. Similar results hold for the family of edge sets k-spanned by an undirected multigraph consisting of k edge-disjoint spanning trees. This is useful for rigidity.

We turn to undirected graphs. The main result is illustrated in Figure 1 and rests on this notion: An *m-tree* rooted at R (where R is a sequence of nodes) is a directed graph defined by the following recursive rules. We use this notation: for x a node and S a set of nodes, xS denotes the set of directed edges $\{xs \mid s \in S\}$.

(i) A single node r is an m-tree rooted at r.

(ii) Let T_i be an m-tree rooted at R_i for $i = 1,\ldots,h$, $h \geq 1$. Let r be a new node. Then the set of edges $\bigcup_{i=1}^{h}(rR_i \cup T_i)$ is an m-tree rooted at r.

(iii) Let T_i be an m-tree rooted at R_i for $i = 0,\ldots,h$, $h \geq 2$. Let r_i, $i = 1,\ldots,h$ be new nodes. Then the set of edges $\bigcup_{i=1}^{h}(r_i(R_{i-1} \cup R_i) \cup T_i)$ is an m-tree rooted at r_i, $i = 1,\ldots,h$.

Theorem 4.2. For an undirected graph, let $\mathcal{F}$ be the family of a-minsinks. P^r is an m-tree; it is a tree if λ is odd. ∎

The m-tree represents all mincuts, explicitly (there are $O(n^2)$ minsinks). Figure 1 shows a 2-edge-connected graph G and the m-tree of a-minsinks. Each node $[e]$ of the m-tree is labelled with the edges and vertices in $[e]$.

Recall that for any integer k, the vertices of an undirected graph G can be partitioned into k-edge-connected components: two vertices are in the same k-edge-connected component if and only if they cannot be separated by deleting fewer than k edges. The partition of $V(G)$ into nodes of P^r gives the $(\lambda+1)$-edge-connected components of G.

5. Full poset constructions.

This section summarizes algorithms to construct the complete poset representation for problems involving graphic matroids. Complete statements and proofs are in [G91b].

We construct the tree or m-tree for the mincuts of an undirected graph. We adapt the algorithm of Theorem 3.1, using another set merging data structure for postvisited nodes.

Theorem 5.1. For an undirected graph, the m-tree representing all a-minsinks can be found in time $O(m+\lambda^2 n\log(n/\lambda))$ and space $O(m)$. ∎

The m-tree can be converted to the cactus representation of [DKL] in linear time.

The representation mentioned after Theorem 4.1 is the basis of the directed graph algorithm.

Theorem 5.2. For a directed graph, let $\mathcal{F}$ be the family of a-minsinks. P can be found in time $O(\lambda m\log(n^2/m)+n^2)$. P^r and P^c can be found in time $O(nm)$. The space is $O(n^2)$. ∎

A *bar-and-body framework* is a collection of rigid bars attached to rigid bodies by universal joints. It can be represented by a multigraph G. Intuitively G is *generically rigid* in $\mathbf{R}^d$ if it is rigid (i.e., it admits no instantaneous internal motions) whenever it models a bar-and-body framework in $\mathbf{R}^d$ (a technical condition is that the coordinates of all endpoints of all edges are algebraically independent [R]). White and Whiteley [WW] investigate frameworks in $\mathbf{R}^d$ and

on more general surfaces parameterized by k. Multigraph G is *k-isostatic* if it is generically rigid for such a surface but deleting any edge destroys this property. They show G is k-isostatic if and only if its edges can be partitioned into k spanning trees. The k-isostatic subgraphs of a k-isostatic graph correspond to the k-spanned edge sets (discussed after Theorem 4.1). The sets $M(e)$ correspond to the irreducible factors of the polynomial describing the rigidity of G and are used to describe certain stresses and motions of the frame [WW, pp. 18–23].

Theorem 5.3. Consider a k-isostatic graph, given as k edge-disjoint spanning trees. Let $\mathcal{F}$ be the family of k-isostatic subgraphs. P can be found in time $O(n^2)$. P^c and P^r can be found in time $O(kn^2)$. The space is $O(n^2)$. ∎

A *bar-and-joint framework* is a collection of rigid bars connected with universal joints. The known theory is for frameworks in $\mathbf{R}^2$ [R]. Nakamura shows that a poset P_2 derived from the poset for k-spanned edge sets represents all rigid subgraphs of a minimally rigid graph [N].

Theorem 5.4. P_2^c and P_2^r can be found in time $O(n^2)$. The space is $O(n^2)$. ∎

Imai constructs P_2^c in the same time using network flow techniques [I].

6. Augmenting edge connectivity.

This section solves the edge connectivity augmentation problem for undirected graphs. It concludes by briefly highlighting the algorithm for directed graphs.

We use two notational conventions. For a function $f : A \to \mathbf{R}$ and a set $S \subseteq A$, define $f(S) = \sum\{f(s) \mid s \in S\}$. The value $d(S)$ refers to the original graph G even after we have added edges, unless explicitly noted otherwise.

Our algorithm, like all others we know of, is based on the following duality relation first proved in [WN]: The minimum number of edges needed to increase the edge connectivity of an undirected graph to τ is precisely the maximum, over all subpartitions $\mathcal{P}$ of V, of $\lceil \frac{1}{2} \sum\{\tau - d(S) \mid S \in \mathcal{P}\} \rceil$. Our development provides another proof. It is easy to see that the number of edges is at least as large as the subpartition quantity.

We begin with the case $\delta = 1$. [NGM] shows this problem can be solved in linear time given the cactus representation of G. Their algorithm works unchanged for the m-tree representation.

Lemma 6.1. The undirected edge connectivity augmentation problem for $\delta = 1$ can be solved in time $O(m + \lambda^2 n \log(n/\lambda))$ and space $O(m)$. ∎

We need the following properties of the $\delta = 1$ algorithm. When executed on a given graph G, each new edge added by the algorithm joins two minimal minsinks of G. (A minimal minsink is a minsink not properly containing any minsink. It is an example of an "extreme set" defined below.) Each end x of a new edge can be chosen as an arbitrary vertex in the minsink K. At most one minimal minsink is incident to exactly two new edges; all others are incident to exactly one new edge.

Now we discuss the general problem. We use the basic notion of [NGM]: An *extreme set* is a set of vertices X such that any nonempty proper subset has larger degree, i.e., $\emptyset \subset Y \subset X$ implies $d(Y) > d(X)$. A *weak extreme set* is the analogous notion using weak inequality, i.e., $\emptyset \subset Y \subset X$ implies $d(Y) \geq d(X)$. Note that V is an extreme set.

A fundamental fact due to [NGM] is that the extreme sets nest. We prove the following slightly stronger version of this fact.

Lemma 6.2. If an extreme set meets a weak extreme set then the sets nest.

Proof. Recall that for any vertex sets X and Y, $d(X) + d(Y) \geq d(X - Y) + d(Y - X)$. Let X be extreme and Y weak extreme, and suppose the sets meet but do not nest. Then $d(X - Y) > d(X)$ and $d(Y - X) \geq d(Y)$, a contradiction. ∎

The tree corresponding to the nesting of extreme sets is the basis of the algorithm of [NGM]. We define a smaller tree T that we can construct efficiently.

Each node of T is labelled with a vertex x of G; x occurs as at most one label. For convenience we denote a node of T by its label x. Associated with each node x is an extreme set $M(x)$ (M stands for maximal) and a nonnegative integer $\overline{d}(x)$.

We start by defining T and M. The root of T is labelled with an arbitrarily chosen vertex r, and $M(r) = V$. Consider any node x of T. Each child of x corresponds to a maximal extreme set N of G contained in $M(x) - x$ and having $d(N) < \tau$. The label for this child is an arbitrary vertex $y \in N$, and $M(y) = N$.

Next we define the function $\overline{d}$. It has domain V; let $\overline{d}(x) = 0$ if x is not a node of T. If x is a node of T, assume $\overline{d}$ is defined for all descendants of x. Let P be the extreme set (of G) with $x \in P \subseteq M(x)$, $P \neq V$, that minimizes $d(P) + \overline{d}(P - x)$. Then $\overline{d}(x) = \max\{\tau - d(P) - \overline{d}(P - x),\ 0\}$. We occasionally write $P(x)$ to denote the minimizing set P.

T has two important properties. First it gives the subpartition of V of the fundamental duality relation, as follows. Define a subpartition by choosing the max-

imal sets $P(x)$ that have $\overline{d}(x)$ positive. These sets are disjoint (by the nesting of extreme sets). They contain all nodes with positive $\overline{d}$. Each such set $P(x)$ satisfies $d(P(x))+\overline{d}(P(x)) = \tau$. This implies the total number of edges needed to make G τ-edge-connected is at least $\lceil \overline{d}(V)/2 \rceil$. Our algorithm adds precisely this number of edges, so this is the minimum.

The second property of T is that any extreme set $K \subset V$ has $d(K) + \overline{d}(K) \geq \tau$. In proof, there is a (unique) node x of T such that $x \in K \subseteq M(x)$. Hence the definition of $\overline{d}$ implies $d(K) + \overline{d}(K) \geq \tau$.

Our connectivity augmentation algorithm has the same high-level structure as [NGM]. It repeatedly increments the connectivity by one; δ such incrementations achieve the desired connectivity τ. Each connectivity incrementation is done using the algorithm of Lemma 6.1 with one added rule: When choosing an endpoint x of a new edge, in an extreme set (minimal minsink) K, if possible choose $x \in K$ so it is on fewer than $\overline{d}(x)$ new edges.

Lemma 6.3. The algorithm constructs a τ-edge-connected graph, adding the fewest possible number of edges.

Proof. Let G^c denote the graph constructed by the algorithm that achieves connectivity c, for $c = \lambda, \ldots, \tau$. Thus G^λ is the original graph G and G^τ is the final graph. Recall that if K is an extreme set of G^c of degree c (i.e., a minimal minsink) then $G^{c+1} - G^c$ contains either one or two edges incident to K.

We first show that for any $c > \lambda$, any extreme set of G^c is extreme in G^{c-1}. If not, choose the smallest c where this fails. Suppose that adding edge xy creates a new extreme set S of G^c. Clearly both x and y are in S. At least one of x and y is in an extreme set of G^{c-1} that has degree c in G^c. Let this be vertex x in set X. Note that X is weak extreme in G^c. Hence Lemma 6.2 shows that S and X nest. Since $y \notin X$ this implies $X \subseteq S$. In G^c, $d(S) \geq c = d(X)$, a contradiction.

An extreme set of degree c in G^c is a maximal extreme set of G^c. Thus no edge added to G^c has both ends in the same extreme set of G^c. Thus if K is extreme in G^c, no edge of $G^c - G$ has both ends in K.

Now consider the connectivity incrementation algorithm when it adds edges to G^c to obtain G^{c+1}. Let K be an extreme set of G^c of degree c. We show that for each new edge added incident to K, K contains a vertex y on fewer than $\overline{d}(y)$ new edges, except possibly for one end of the last edge added by the algorithm.

Before any edge is added to K, K has degree c. Recall the basic property $d(K) + \overline{d}(K) \geq \tau$. Since no edge of $G^c - G$ has both ends in K, fewer than $\overline{d}(K)$ edges with an end in K have been added. Thus K contains a vertex y on less than $\overline{d}(y)$ new edges if $c \leq \tau - 2$, even if two edges incident to K are added. Similarly if $c = \tau - 1$ the desired vertex y exists for the first edge added to K. Thus the desired y does not exist only if $c = \tau - 1$ and two edges incident to K are added, i.e., one end of the last new edge.

We conclude that the total increase in degree of vertices (due to new edges) is at most $\overline{d}(V) + 1$. Thus the number of edges added is at most $\lfloor (\overline{d}(V) + 1)/2 \rfloor = \lceil \overline{d}(V)/2 \rceil$.

The final graph is τ-edge-connected. We have already seen that $\lceil \overline{d}(V)/2 \rceil$ new edges are needed, and the algorithm adds at most this number. The lemma follows. (This also proves the basic duality relation.) ∎

It remains to show how to compute T and the associated functions. Start by setting each value $\overline{d}(x)$ to 0. Then execute the procedure $t(r, V)$. Here $t(x, X)$ is a recursive procedure, called with X an extreme set containing vertex x. Procedure t constructs the subtree of T rooted at node x with $M(x) = X$, plus all function values for nodes in this subtree. It works in two phases. Phase one finds the entire subtree and all function values except $\overline{d}(x)$. Phase two finds $\overline{d}(x)$. The main data structure for t is a multigraph H that gets repeatedly modified. The vertices of H consist of the vertices in X plus a new vertex a, that represents $V - X$ if $X \neq V$. When we contract a set of vertices Y in H, $[Y]$ denotes the new vertex; any parallel edges resulting from contraction are included in the new multigraph H.

Procedure $t(x, X)$ starts by creating a new node of T labelled x and setting $M(x) \leftarrow X$. Phase one initializes multigraph H by starting with G and doing the following: If $X \neq V$ then contract $V - X$ into a single vertex a; if $X = V$ then add a new vertex a; in both cases add τ edges joining a and x. Then repeat the following steps until phase one is complete:

If $\lambda(H) \geq \tau$ then stop (phase one is complete). Otherwise find all extreme a-sets Y of graph H that have degree $\lambda(H)$. Do the following for each such Y. Arbitrarily choose a vertex $y \in Y$ and call $t(y, Y)$. This returns a subtree rooted at a node labelled y; make node y a child of x in T. In H contract Y to a single vertex $[Y]$ and add $\min\{\overline{d}(Y), \delta\}$ new edges joining $[Y]$ with a.

Phase two starts by modifying (the current) H as follows. Delete the τ edges joining a and x that phase one added (keep any other edges joining a and x). Furthermore if $X = V$ then for Y_0 an extreme set of smallest degree found in phase one, increase the number of edges joining $[Y_0]$ and a to τ. Finally set $\overline{d}(x) \leftarrow \max\{\tau - \lambda(H), 0\}$. ∎

The algorithm is proved correct by induction. The following inductive assertion is useful: At any time during phase one, consider the degree (in H) of any a-set S of H. This degree is the same as in G unless S contains x or some contracted vertex $[Y]$, in which case it is at least τ.

The analysis also shows an important property for the efficiency of t: In phase one $\lambda(H)$ increases every iteration.

We turn to efficiency. Procedure t is implemented as follows. Compute $\lambda(H)$ using the round robin connectivity algorithm [G91a]. In phase one, find the extreme a-sets Y of H with degree $\lambda(H)$ by constructing the m-tree representation of H; the sets Y are the leaves of the m-tree representation that do not contain a.

The time for the entire algorithm is dominated by the time for all connectivity computations (in phases one and two). Recall that the round robin connectivity algorithm computes $\lambda(H)$ by finding a complete intersection for vertex a and integer k, where k takes on successive values $1,\ldots,\lambda(H)$. We speed this up by making each phase one connectivity computation start with a previously computed intersection, as follows.

Consider a call $t(y,Y)$ made from invocation $t(x,X)$. Suppose the current graph H of $t(x,X)$ has $\lambda(H)=c$. Thus the connectivity algorithm has found a complete intersection T for c on H. The subgraph induced by Y is the same in H as in G (clearly Y does not contain any contracted vertex). Thus this subgraph corresponds precisely to the graph H constructed initially in $t(y,Y)$. Since Y is an a-minsink, Lemma 3.1 shows T contains c subtrees that span Y, say Y_i, $i=1,\ldots,c$, and T also contains each of the c directed edges v_iw_i, $i=1,\ldots,c$ in $\rho(Y)$. (T may also contain reversed edges w_iv_i; furthermore a tree of T may contain Y_i but no directed edge v_jw_j; this causes no harm.) To get a complete intersection for c on graph H of $t(y,Y)$, let the ith spanning tree consist of Y_i plus the edge aw_i corresponding to v_iw_i. The invocation $t(y,Y)$ uses this intersection to start its first connectivity computation.

Lemma 6.4. Procedure $t(r,V)$ uses time $O(\tau(m+\tau n)\log n)$ and space $O(m+\delta n)$.

Proof. We estimate the time for all connectivity computations. Call a graph processed by an execution of round robin (in any invocation $t(x,X)$) an "r-graph". Each r-graph R has an associated index c, where round robin was run to compute a complete intersection of c trees on R. (Thus round robin started with $c-1$ trees and computed a cth tree.) We show below that all r-graphs corresponding to a fixed index c contain $O(n)$ vertices and $O(m+\tau n)$ edges. Recall that executing round robin on a multigraph of size $\overline{n},\overline{m}$ uses time $O(\overline{m}\log\overline{n})$. Thus round robin uses total time $O(\tau(m+\tau n)\log n)$ on all r-graphs. This also bounds the time for computing m-tree representations. This proves the time bound of the lemma.

We first bound the number of vertices in r-graphs. For any index c, any vertex v of G is in at most one r-graph for c in each of phases one and two. To show this choose node x of T so that v is in $M(x)$ but not in $M(y)$ for any child y of x. Let $t(x,X)$ denote the invocation of t for node x. In phase one $\lambda(H)$ increases every iteration (as indicated above). Thus it is easy to see that before the invocation $t(x,X)$, v is in precisely one phase-one r-graph for each $c\in[1,\lambda(X)]$ (this occurs in invocations for proper ancestors of x in T). In $t(x,X)$, v is in at most one phase-one r-graph for each value $c\in(\lambda(X),\tau]$. Also in $t(x,X)$, v is in at most one phase-two r-graph for each $c\in[1,\tau]$. After $t(x,X)$, v is never in an r-graph since it gets contracted.

Since there are at most n contracted vertices $[Y]$ it is easy to show the desired bound of $O(n)$ vertices. The bound on edges is similar. ∎

When the edges are added by the $\delta=1$ algorithm, the m-tree representations are constructed in the same time as Lemma 6.4.

The algorithm can be improved if $m>\tau n$: Any graph G has a subgraph S of at most τn edges, such that for any set of new edges N, $G+N$ is τ-edge-connected if and only if $S+N$ is. S can be found in $O(m)$ time [NagI]. This gives the final result.

Theorem 6.1. The edge-connectivity augmentation problem for undirected graphs can be solved in time $O(m+\tau^2 n\log n)$ and space $O(m+\delta n)$. ∎

The connectivity augmentation algorithm for directed graphs is similar to the undirected algorithm but more complicated. For instance the in-extreme sets (defined analogous to the undirected case) do not nest. However the union of two in-extreme sets that cross is in-extreme. This leads to a tree similar to T to compute the degrees of vertices in the new graph. Edges are added using a modification of the $\delta=1$ algorithm of Section 2, although the connectivity itself need not increase. Some edges are added incorrectly and the algorithm and uses binary search to backtrack. The final result is stated in Section 1.

Acknowledgments. Many thanks to András Frank, Dan Gusfield, Dalit Naor and Éva Tardos for their kind help.

References.

[DKL] E.A. Dinits, A.V. Karzanov and M.V. Lomonosov, "On the structure of a family of minimal weighted cuts in a graph", in *Studies in Discrete Optimization*, A.A. Fridman (Ed.), Nauka Publ., Moscow, 1976, pp. 290–306.

[DM] A.L. Dulmage and N.S. Mendelsohn, "A structure theory of bipartite graphs of finite exterior dimension", *Trans. Roy. Soc. Canada*, Section III, 53, 1959, pp. 1–13.

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

[E72] J. Edmonds, "Edge-disjoint branchings", in *Combinatorial Algorithms*, R. Rustin, Ed., Algorithmics Press, New York, 1972, pp. 91–96.

[F] A. Frank, "Augmenting graphs to meet edge-connectivity requirements", *Proc. 31st Annual Symp. on Found. of Comp. Sci.*, 1990, pp. 708–718.

[G91a] H.N. Gabow, "A matroid approach to finding edge connectivity and packing arborescences", *Proc. 23rd Annual ACM Symp. on Theory of Comp.*, 1991, pp. 112–122.

[G91b] H.N. Gabow, "Applications of a poset representation to edge connectivity and graph rigidity", Dept. of Comp. Sci., Univ. of Colorado, Tech. Rept., 1991.

[GI] D. Gusfield and R.W. Irving, *The Stable Marriage Problem: Structure and Algorithms*, MIT Press, Cambridge, MA, 1989.

[GT] H.N. Gabow and R.E. Tarjan, "A linear-time algorithm for a special case of disjoint set union", *J. Comp. and System Sci., 30*, 2, 1985, pp. 209–221.

[GW] H.N. Gabow and H.H. Westermann, "Forests, frames and games: Algorithms for matroid sums and applications", *Proc. 20th Annual ACM Symp. on Theory of Comp.*, 1988, pp. 407–421.

[I] H. Imai, "Network-flow algorithms for lower-truncated transversal polymatroids", *J. Op. Res. Soc. of Japan, 26*, 3, 1983, pp. 186–210.

[KT] A.V. Karzanov and E.A. Timofeev, "Efficient algorithm for finding all minimal edge cuts of a nonoriented graph", *Kibernetika*, 2, 1986, pp. 8–12; translated in *Cybernetics*, 1986, pp. 156–162.

[N] M. Nakamura, "On the representation of the rigid sub-systems of a plane link system", *J. Op. Res. Soc. of Japan, 29*, 4, 1986, pp. 305–318.

[NGM] D. Naor, D. Gusfield and C. Martel, "A fast algorithm for optimally increasing the edge-connectivity", *Proc. 31st Annual Symp. on Found. of Comp. Sci.*, 1990, pp. 698–707.

[NagI] H. Nagamochi and T. Ibaraki, "Linear time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph", *Algorithmica*, to appear.

[NakI] M. Nakamura and M. Iri, "Fine structures of matroid intersections and their applications", *Proc. Int. Symp. Circuits and Systems*, 1979, pp. 996–999.

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

[R] A. Recski, *Matroid Theory and its Applications in Electric Network Theory and in Statics*, Springer-Verlag, New York, 1989.

[T] R.E. Tarjan, "Depth-first search and linear graph algorithms", *SIAM J. Comput., 1*, 2, 1972, pp. 146–160.

[WN] T. Watanabe and A. Nakamura, "Edge-connectivity augmentation problems", *J. Comp. and System Sci., 35*, 1, 1987, pp. 96–144.

[WW] N. White and W. Whiteley, "The algebraic geometry of motions of bar-and-body frameworks", *SIAM J. Alg. Disc. Meth., 8*, 1, 1987, pp. 1–32.

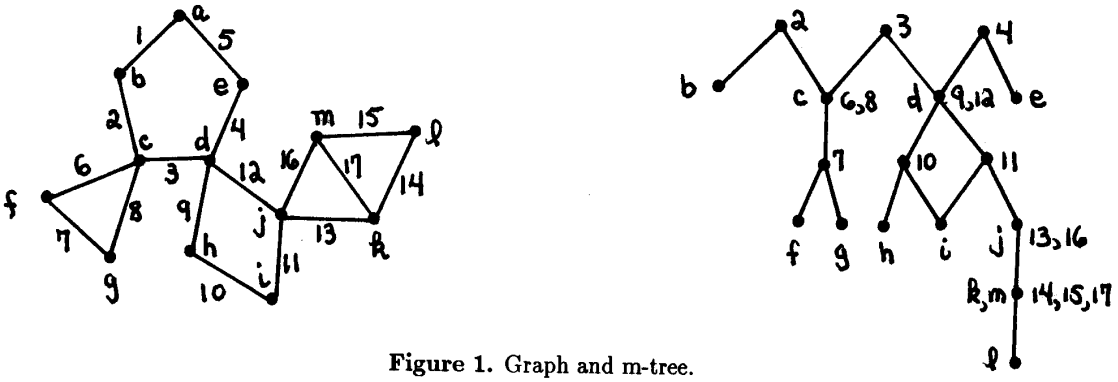

Figure 1. Graph and m-tree.