Algorithmica (1992) 7: 465–497

Algorithmica

Forests, Frames, and Games: Algorithms for Matroid Sums and Applications[1]

Harold N. Gabow[2] and Herbert H. Westermann[3]

Abstract. This paper presents improved algorithms for matroid-partitioning problems, such as finding a maximum cardinality set of edges of a graph that can be partitioned into k forests, and finding as many disjoint spanning trees as possible. The notion of a clump in a matroid sum is introduced, and efficient algorithms for clumps are presented. Applications of these algorithms are given to problems arising in the study of the structural rigidity of graphs, the Shannon switching game, and others.

Key Words. Matroid, Matroid sum, Matroid partitioning, Covering, Arboricity, Packing, Bar-and-joint framework, Bar-and-body framework, Rigidity, Shannon switching game.

1. Introduction. Matroid theory provides a unifying framework for many diverse areas of combinatorics. The matroid sum has been called one of the most successful notions in matroid theory [A, p. 294]. In large part this is due to the theorems of Edmonds on covering and packing, and Edmonds' matroid-partitioning algorithm [E1], [E2].

This paper investigates matroid-partitioning algorithms, both on the general matroid level and for specific matroids. Our general matroid algorithm incorporates several ideas that should make it more efficient than Edmonds' algorithm (and other partitioning algorithms) on any specific matroid. We support this claim by applying the algorithm to a number of matroids, the most important being graphic matroids. Our main results for specific matroids are summarized in Table 1. (The expressions in Table 1 are asymptotic, e.g., m represents $O(m)$.) Before explaining the table we introduce some notation and terminology.

All of the specific matroids in this paper are derived from undirected graphs. The given graph is denoted $G = (V, E)$. We use several parameters for the size of G. As usual n and m denote the number of vertices and edges, respectively. In problems involving the parameter k (which indicates the size of the desired solution) it is convenient to define

$$n' = \min\{n, 2m/k\}, \qquad m' = m + n' \log n'.$$

---

[1] This is a revised and expanded version of a paper appearing in the *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, 1988. This research was supported in part by National Science Foundation Grants MCS-8302648 and DCR-851191.

[2] Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309, USA.

[3] IBM Laboratories, Department 3228, Schönaidner Strasse 220, 7030 Böblingen, Federal Republic of Germany.

Received October 4, 1989; revised April 8, 1990. Communicated by Greg N. Frederickson.

Table 1. Main results (notation: $n' = \min\{n, 2m/k\}$, $m' = m + n' \log n'$).

Problem	Previous bounds	New bounds
k-forests	$k^{3/2}n\sqrt{m \log n}$, if $m \leq n \log n$ $k^{3/2}m\sqrt{n}$, otherwise [GS] $(kn)^2$ [I], [RT]	$k^{3/2}n'\sqrt{m'}$ $k(n')^2 \log k$
weighted k-forest	$O((kn)^2 + m \log m)$ [RT]	$O(k(n')^2 \log k + m \log m)$
$cov(\mathcal{G})$	m^2 [I], [RT]	$m(mm' \log n)^{1/3}$, if $m \leq n^{3/2} \log n$ $mn \log n$, otherwise
$pack(\mathcal{G})$	m^2 [RT]	$m\sqrt{(mm'/n) \log(m/n)}$ $mn \log(m/n)$
Top clump		$n\sqrt{m'}$
Degrees of freedom	mn [I]	n^2
Shannon switching game		$n\sqrt{m'}$, preprocessing time n, query time

(If a problem is not stated in terms of k yet our time bound involves n' or m', there is no harm in assuming $n' = n$, but the text gives a better choice.)

A *matroid* consists of a finite set S and a family of subsets of S, called *independent sets*. The independent sets are closed under taking subsets, and they satisfy an enlargement axiom: For any independent sets I and J with $|I| < |J|$, I can be enlarged to a bigger independent set by adding some element of $J - I$. A *base* of a matroid is a maximum cardinality independent set. For example, the *graphic matroid* of a graph G, denoted $\mathcal{G}(G)$ (or $\mathcal{G}$ when the graph is obvious), has elements the edges of G, independent sets the forests of G, and bases the spanning trees if G is connected else the spanning forests. For matroids M_i, $i = 1, \ldots, k$, on the same set S, the *matroid sum* $\bigvee_{i=1}^{k} M_i$ is a matroid M on S, whose independent sets are all sets $I \subseteq S$ that can be partitioned into disjoint subsets I_i, $i = 1, \ldots, k$, with I_i independent in M_i. For such I and corresponding I_i, we say $I = \bigcup_{i=1}^{k} I_i$ is a *partitioned independent set* of M. (An independent set can have more than one valid partition.) A *partitioned base* of M is defined similarly. An important special case is when all summands are identical—the *k-fold sum* of a matroid N, denoted N^k, is $\bigvee_{i=1}^{k} N$. As an example, an independent set in $\mathcal{G}(G) \vee \mathcal{G}(G)$ is a subgraph that can be partitioned into two forests. Figure 1(a) shows a graph used as an example throughout this paper; Figure 1(b) gives a partitioned base of the two-fold graphic sum, where the two forests consist of the solid edges and the dotted edges, respectively. The *matroid-parititioning problem* is to find a partitioned base B of $\bigvee_{i=1}^{k} M_i$ for given matroids M_i.

The first entry in Table 1 is for the *k-forest problem*: Given a graph G and an integer k, find k edge-disjoint forests collectively containing as many edges as possible. (Equivalently this is the matroid-partitioning problem of the k-fold sum of a graphic matroid.) This problem has many applications: In the analysis of electrical networks, the solution for $k = 2$ (called *extremal trees* [KK]) is central to hybrid analysis (it gives the minimum fundamental equations of the network [OIW], [IF]). The k-forest problem also arises in the study of rigidity of structures.

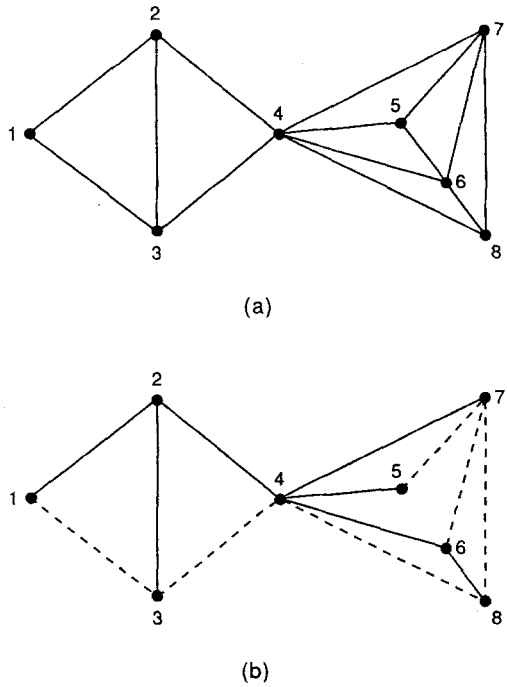

Fig. 1. (a) Example graph. (b) Partitioned base of $\mathscr{G} \vee \mathscr{G}$.

For $k = 2$ it determines if a graph is rigid as a bar-and-joint framework on the surface of a cylinder in three-dimensional space $\mathbf{R}^3$ [Wh]. For $k = d(d+1)/2$ it determines if a graph is rigid as a bar-and-body framework [Tay] and also as a body-and-hinge framework [Wh] in $\mathbf{R}^d$. For aribitrary k it determines rigidity of bar-and-joint frameworks on the flat torus in $\mathbf{R}^k$ [Wh] and also rigidity of k-frames [WW].

We give two k-forest algorithms. The first, giving the first line of Table 1, applies our general matroid algorithm. It improves the previous algorithms over all ranges of parameters. The second k-forest algorithm has better dependence on k than the first. (A third algorithm gives a minor improvement.) To explain the second algorithm we introduce another matroid on graphs.

The *bicircular matroid* of a graph G, denoted $\mathscr{B}(G)$ ($\mathscr{B}$, when the graph is understood), was studied in [M]. It is related to the graphic matroid but seems to have simpler structure (see, e.g., [GT2]). A set is independent in the bicircular matroid if it is a *pseudoforest*, i.e., each connected component has at most one cycle. The *k-pseudoforest problem* is the analog of the k-forest problem. This problem arises in the theory of structural rigidity; e.g., for $k = 2$ it determines the rigidity of bar-and-joint frameworks on the surface of a regular tetrahedron [Wh]. The *mixed-forest problem* is to find a maximum cardinality set of edges that can be partitioned into f forests and p pseudoforests, along with its partition, i.e., a partitioned base of $\mathscr{G}^f \vee \mathscr{B}^p$. This problem determines the rigidity of bar-and-joint

frameworks on the surface of a three-dimensional cone ($f = p = 1$); rigidity on higher-dimensional surfaces has been conjectured to be determined by the problem with higher values of f and p [Wh].

Our k-pseudoforest algorithm is another implementation of the general algorithm. It is faster than the k-forest algorithm because of the simpler structure of the bicircular matroid. Some techniques from network flow theory [ET] are used in the analysis. Our algorithm for the mixed-forest problem combines the two algorithms in a way that simplifies the computation on the more difficult graphic matroids. (The time bounds for these algorithms are given in Theorems 3.2 and 3.5. We know of no previous work.)

Our second k-forest algorithm is based on an interesting fact: k pseudoforests can always be converted into $k + 1$ forests. This observation is due to Picard and Queyranne [PQ] who phrased it in terms of the arboricity of a graph (defined below). Our second algorithm uses this observation recursively in a divide-and-conquer scheme. We call this the *stepping-stone approach.* The stepping-stone approach extends to weighted problems. For example it gives the table entry for the *weighted k-forest problem,* i.e., finding a maximum weight set of k edge-disjoint forests.

We turn to covering and packing problems. The *covering problem* for a matroid M is to find $cov(M)$, the smallest number of independent sets that contain all elements, together with the independent sets. For example $cov(\mathcal{G})$ is the *arboricity* of the graph, the fewest number of forests that contain all edges. The *packing problem* is to find $pack(M)$, the greatest number of disjoint bases contained in S, along with the bases themselves. The packing problem on a graphic matroid gives spanning trees that can be used for fault tolerant communication (up to $pack(\mathcal{G}) - 1$ edge failures can be tolerated without disrupting communication [G], [IR]).

We present algorithms for covering and packing on a general matroid M. The algorithms use a *balanced binary search.* This speeds up ordinary binary search, as follows: each probe gathers slightly less than complete information; after the search, postprocessing closes the knowledge gap. This technique is used in [GT3] for bottleneck matching. The bounds for graphic matroids improve previous work, as indicated in Table 1. For bicircular matroids, the "pseudoarboricity" was investigated in [PQ] and [GGT] (they do not use matroids). Our bound, given in Theorem 4.1, improves these results. We know of no previous work on packing bicircular matroids.

We introduce the notion of *clump* in a matroid sum. This term was invented by Roskind and Tarjan [RT], who defined a clump in a graph algorithmically. We generalize their notion and use it to analyze our algorithms. We also define the *top clump.* This leads to a family of invariants of matroids (*a fortiori,* invariants of a graph). It generalizes two other matroid invariants, the principal partition of [KK] and its generalization, the r-minors of [BW2]. We introduce another useful notion, the *preclump* (essentially a maximal set not containing a clump).

The next problem is to find a top clump. Here are three applications:

(i) Lovász and Yemini [LY] give an algorithm to determine if a graph is generic independent (minimally rigid) in the plane. It involves solving n

2-forest problems. Using the top clump, our time bound is the time to solve one 2-forest problem.

(ii) White and Whiteley [WW] define k-irreducible graphs, related to rigidity of bar-and-body frameworks. We can recognize a k-irreducible graph, essentially by finding a top clump. The time is the time for one k-forest problem. (There does not seem to be an obvious algorithm for this problem; [WW] does not give an algorithm.)

(iii) The Shannon switching game can be solved by finding a top clump and doing simple postprocessing, as indicated below.

We present a related algorithm to determine the number of degrees of freedom of a bar-and-joint framework in the plane; in fact the algorithm can be used to give a complete description of the possible motions of such a framework. The algorithm amounts to finding a maximum cardinality generic-independent set, or, equivalently, finding a base of the two-dimensional rigidity matroid. (The algorithm presented here is for the rigidity matroid, but it generalizes to matroids defined by similar submodular functions.)

The Shannon switching game (defined in Section 7) has been greatly studied (see, e.g., [Law]). We improve previous solutions in two ways. First, our solution to the 2-forest problem gives the best-known bound for the simple form of the game investigated by most researchers (e.g., [RT], [KK], and [BW1]). Second, we give a more general analysis of the switching game. We assume a fixed graph and allow the goal vertices to vary. We show that the top clump and preclumps determine the winner of a switching game; furthermore, they give the trees and cotrees necessary to execute a winning strategy. This leads to various algorithms for finding the winner of the game (classification and tree queries, defined in Section 7). For instance, we can find the winner of each of the $\binom{n}{2}$ possible switching games on a given graph, in time $O(n^2 \log n)$. This result is based on an algorithm that finds a decomposition of a graph into maximal preclumps. (This algorithm also extends to more general submodular functions and their matroids.)

In summary, the main contributions of our work are using pseudoforest algorithms as a stepping stone to tree algorithms; showing the applicability of balanced binary search; using top clumps and preclumps to solve rigidity problems and the Shannon switching game. The stepping-stone approach and the notions of top clump and preclump generalize to other matroids, although this is not discussed here.

The remainder of this paper is organized as follows. Section 2 presents our general matroid-partitioning algorithm. Section 3 describes our algorithms for the k-forest problem and related problems. Section 4 gives algorithms for covering and packing problems. Section 5 discusses the top clump, giving a general algorithm to compute it. Section 6 solves rigidity problems for bar-and-joint and bar-and-body problems. Section 7 presents our analysis of the Shannon switching game. The rest of this section gives some notation. Readers more interested in applications than algorithmic techniques can skim Sections 3 and 4.

If S is a set and e is an element, $S + e$ denotes $S \cup \{e\}$, $S - e$ denotes $S - \{e\}$. We use the following notation for matroid concepts; detailed definitions can be

found in [A], [Law], and [Wel]. Let M be a matroid on a set S, and let A be a set of elements of S. The *rank* of A in M is denoted $\rho(A, M)$. Its *span* is $sp(A, M)$. (Recall the definition $sp(A, M) = A \cup \{e | e \cup C \text{ is a circuit for some } C \subseteq A\}$.) If A is independent and spans an element e, the *fundamental circuit* of e in A is denoted $C(e, A, N)$. In all these notations we drop the last argument M when it is clear from context. The *restriction* of M to A is denoted $M|A$; its elements are A. The *contraction* of M by A is denoted M/A; its elements are $S - A$, and $T \subseteq S - A$ is independent in M/A if $T \cup A'$ is independent in M, for A' a maximal independent subset of A. The *dual matroid* of M is denoted M^*; its independent sets are all $A \subseteq S$ such that $sp(S - A, M) = S$. A *partition matroid* M is given by a partition of S into r sets $S_1, \ldots, S_r$ and nonnegative integers $d_1, \ldots, d_r$; $A \subseteq S$ is independent if $|A \cap S_q| \leq d_q$ for $q = 1, \ldots, r$.

2. Clumps and Augmenting Paths. This section sketches our algorithm for the matroid-partitioning problem on general matroids. The section starts by introducing clumps, which help analyze the algorithm. Throughout this section M denotes an arbitrary matroid sum $M = \bigvee_{i=1}^{k} M_i$ on a set S, unless stated otherwise.

Any independent set L of M obviously satisfies $|L| \leq \sum_{i=1}^{k} \rho(L, M_i)$. A set L is a *clump* of M if it is independent in M and the previous inequality holds with equality. (Figure 1(b) has two clumps—the subgraphs induced by vertices 4, 6, 7, 8 and 4, ..., 8.) Equivalently, a set of elements L is a clump of M if it can be partitioned into sets L_i, $i = 1, \ldots, k$, with L_i independent in M_i and $L \subseteq sp(L_i, M_i)$ for $i = 1, \ldots, k$. A useful property of clumps is that $L + e$ is dependent in M for any element $e \in \bigcap_{i=1}^{k} sp(L, M_i) - L$.

The main algorithmic tool for matroid partitioning is the augmenting path, introduced by Edmonds [E1] for matroid sums. Consider a partitioned independent set $I = \bigcup_{i=1}^{k} I_i$ of M. An *augmenting path* for I is a sequence of elements e_j, $j = 0, \ldots, s$, with $e_j \in I$ precisely when $j > 0$, that enables e_0 to be added to I. Specifically, inserting e_0 into some partition set I_q creates a circuit, which is broken by removing e_1 from I_q; inserting e_1 into a different partition set creates a circuit which is broken by removing e_2; this pattern continues until e_s is inserted into a partition set and does not create a circuit. The process described above is called *augmenting I along $e_0, \ldots, e_s$*. The formal definition of an augmenting path is as follows: $e_0, \ldots, e_s$ is an *augmenting path* if and only if $e_j \in I$ precisely when $j > 0$; for all $j < s$, choosing q so that $e_{j+1} \in I_q$, $I_q + e_j$ is dependent in M_q and $e_{j+1} \in C(e_j, I_q, M_q)$; for some index r, $e_s \notin I_r$ and $I_r + e_s$ is independent in M_r; lastly, augmenting I along $e_0, \ldots, e_s$ gives a new independent set. (Strictly speaking we should include the above index r in the specification of the augmenting path but for convenience we shall not do so.) The crucial last condition is not implied by the others. However, it is implied if the path has no shortcuts, i.e., there are no indices j, k with $j + 1 < k$ and $e_k \in C(e_j, I_q, M_q)$ (here q is the index with $e_k \in I_q$; we give another proof of this well-known fact in Lemma 2.1 below). A prefix of an augmenting path is a *partial augmenting path*. If I is not a base, then an augmenting path exists, and can be used to enlarge the independent set as above [E1].

We use three different algorithms to find augmenting paths in a matroid sum M. In the first two algorithms all knowledge of M is supplied by two subroutines that solve the *static-base circuit problem.* This problem was introduced in [GS] for matroid intersection. A partitioned independent set $\bigcup_{i=1}^{k} I_i$ is given. The problem is to process a sequence of intermixed operations called *start* and $c(e, i)$. The operation $c(e, i)$ outputs "independent" if $I_i + e$ is independent in M_i; otherwise it outputs all elements in $C(e, I_i, M_i) - e$ that have not been output (by any c operation) since the last *start* operation.

The first two algorithms find one augmenting path. They are given a partitioned independent set $I = \bigcup_{i=1}^{k} I_i$ and an alement $e_0 \notin I$. They either find an augmenting path that starts with e_0 or discover that none exists. The latter is called an *unsuccessful search.*

The first algorithm is *breadth-first scanning.* It starts at e_0 and grows partial augmenting paths breadth-first. This is accomplished with two data structures: Any element that has been placed on a partial path gets labeled, and there is a queue Q of labeled elements to be scanned. Initially e_0 is labeled and in Q. The algorithm repeatedly removes the first element e from Q and scans it as follows: If $e \in I_j$ it executes $c(e, i)$ for every index $i \neq j$ (for e_0 take $j = 0$); it labels all elements that are output and adds them to Q. The algorithm stops when an augmenting path is found (i.e., some $c(e, i)$ returns "independent") or when Q becomes empty. The second alternative is an unsuccessful search. (In Figure 1(b) if $e_0 = 56$, then scanning e_0 labels four edges, eventually all edges in the graph induced by vertices $4, \ldots, 8$ get labeled and the search is unsuccessful.)

The last detail of *breadth-first scanning* concerns the initialization. *Breadth-first scanning* begins by doing a *start* operation, *unless* the previous call to *breadth-first scanning* was an unsuccessful search. In this case no *start* is done.

LEMMA 2.1. Breadth-first scanning *is correct.*

PROOF. Correctness amounts to two properties: If an augmenting path is found, augmenting I along that path gives a new partitioned independent set $I + e_0$. If no augmenting path is found, then $I + e_0$ is dependent in M.

To prove the first assertion it is obvious that the algorithm finds paths that satisfy all conditions of the definition of augmenting path except possibly the last; in addition the paths have no shortcuts. (The last property follows since the search is breadth-first. It holds even if there have been unsuccessful searches since the last *start* operation.) Call a path with these properties an "a-path." An easy induction shows that augmenting along an a-path $e_0, \ldots, e_s$ gives a partitioned independent set: This is obvious if $s = 0$. Suppose $s > 0$. Then $e_0, \ldots, e_{s-1}$ is an a-path for $I - e_s$ (because the original path has no shortcuts). Now complete the augment by inserting e_s into the appropriate set I_r (only the last insertion of an augmenting path changes a span $sp(I_i, M_i)$).

To prove the second assertion, for $j = 1, \ldots, k$, define $L_j \subseteq I_j$ to contain all elements that were output by operations $c(\cdot, j)$ since the last *start* operation. The rule for scanning an element implies that $L_i \subseteq sp(L_j, M_j)$ for $1 \leq i, j \leq k$. (This holds even if there have been unsuccessful searches since the last *start* operation.)

Thus the set of labeled elements $L = \bigcup_{i=1}^{k} L_i$ is a clump of M. If the unsuccessful search started with e_0, clearly, $e_0 \in \bigcap_{i=1}^{k} sp(L, M_i) - L$, whence $L + e_0$ is dependent in M. □

Omitting *start* operations after unsuccessful searches makes the algorithm efficient—in fact we say "unsuccessful searches are free." To make this claim precise we first state two assumptions that hold for all applications of this paper. First, the time bound for a successful search allows time for every element to be labeled. (This assumption can fail only in circumstances where we can guarantee that many elements will not be labeled, which is not usually the case.) Second, the input has size $\Omega(mk)$. This assumption holds because we use *breadth-first scanning* only when the k given matroids on S are distinct.

Now we show why unsuccessful searches are free. Consider δ consecutive unsuccessful searches that are followed by a successful search. (If the last δ searches of the algorithm are unsuccessful, invent a fictitious successful search to follow them.) The worst-case time bound for these $\delta + 1$ searches is $O(\delta k)$ plus the worst-case time bound for one successful search. (The term $O(\delta k)$ accounts for the fact that search from e_0 starts with $k - 1$ calls $c(e_0, j)$. The time bound for a successful search accounts for the time to label all elements, in all $\delta + 1$ searches, by our first assumption.) The terms $O(\delta k)$, summed over all unsuccessful searches, amount to $O(mk)$ (there is at most one unsuccessful search per element). This is no more than the size of the input, by our second assumption. Thus we have shown that the unsuccessful searches are not significant in the asymptotic time bound for the algorithm. In other words, unsuccessful searches are free. (This holds even under more relaxed assumptions than those above. For instance the overhead of $O(mk)$ can often be decreased to $O(m)$ if, as with graphic matroids, we can test if an element is in the span of a clump in $O(1)$ time. Since this is not needed for this paper we omit details.)

The labels used by *breadth-first scanning* indicate how to carry out the augmentation of the independent set. Specifically, the label of an element f output by the operation $c(e, i)$ is the pair (e, i). To augment along the path $e_j, j = 0, \ldots, s$, if e_{j+1} has label (e_j, i), then e_j is moved into partition set I_i (replacing e_{j+1}). Note that the routine calling *breadth-first scanning* performs the augment operation.

Breadth-first scanning gets less efficient as k, the number of matroids in the sum, gets larger (since it takes $(k - 1)$ c operations to scan an element e). Our second algorithm to find an augmenting path avoids this inefficiency. It is *cyclic scanning*, introduced in [RT] for graphic matroids. *Cyclic scanning* is applicable to k-fold sums but not general matroid sums. *Cyclic scanning* is similar to *breadth-first scanning* except it calls c fewer times. Specifically to scan a labeled element $e \in I_j$, it only calls $c(e, j + 1)$. (When discussing *cyclic scanning* we make the convention that $k + 1 = 1$. Hence when $e \in I_k$ we call $c(e, 1)$. Also to scan the initial element $e_0 \notin I$ call $c(e, 1)$.) *Cyclic scanning*, like *breadth-first scanning*, omits the *start* operation if the previous search was unsuccessful.

LEMMA 2.2. Cyclic scanning *on a k-fold sum is correct.*

PROOF. The argument is similar to *breadth-first scanning* (Lemma 2.1). To show that any augmenting path P found by *cyclic spanning* is valid, note that P has no

"cyclic shortcuts," i.e., indices j, k with $j + 1 < k$ and $e_k \in C(e_j, I_{j+1})$. (We omit the third argument from C, since all component matroids of a k-fold sum are identical.) The rest of the argument is unchanged.

Now we analyze unsuccessful searches. Define $L_j \subseteq I_j$ as in Lemma 2.1. The cyclic scanning rule implies that $L_i \subseteq sp(L_{i+1})$. This implies $L_i \subseteq sp(L_j)$ for all $1 \leq i, j \leq k$. (This conclusion holds because all matroids in the sum are identical. It fails in a general matroid sum.) The rest of the argument is the same as Lemma 2.1. □

As in *breadth-first scanning*, unsuccessful searches are free. (Here the total overhead for unsuccessful searches is $O(m)$, again the same bound as the input.) Also as in *breadth-first scanning* augments are performed by the calling routine.

The third augmenting-path algorithm *batch* finds more than one augmenting path. This algorithm is the matroid-intersection algorithm of [GS], adapted for matroid sums. It uses the approach of Dinic [D] for network flow: At any point in the algorithm let l denote the length of a shortest augmenting path. The algorithm finds batches of disjoint augmenting paths, where each path in a batch has length l, and the batch is maximal. Unlike network flow, the paths in a batch must be augmented in the correct order. The *batch* algorithm augments the independent set for each path it finds (this contrasts with the first two algorithms, which do not augment). The *batch* routine works using the static-base circuit problem, and also the *dynamic-base circuit problem* [GS]. The inner workings of *batch* are unimportant here. (A lengthy discussion would carry us too far afield. For details see [GS] and [Wes].) We need only two properties, analogous to the cardinality matching algorithm of Hopcroft and Karp [HK]: First after a call to *batch*, the length of a shortest path l increases. Second, let Δ denote the number of paths that remain to be found. Equivalently the current independent set has $\rho(S, M) - \Delta$ elements. Then

$$\Delta l \leq \rho(S, M). \tag{1}$$

We now state our algorithm to find a base of a matroid sum. It starts by calling *batch* some number of times l (from now on l has this meaning). After the lth call at most $\rho(S, M)/l$ paths remain to be found, by the above two properties of *batch*. These paths are found one at a time, using *breadth-first scanning* if the component matroids are different and *cyclic scanning* if they are the same. The running time depends on l; we choose l to balance the time for the two parts of the algorithm.

3. Matroid-Partitioning Algorithms. This section presents our algorithms for the k-forest, k-pseudoforest, and mixed-forest problems. It concludes with our algorithm for the weighted k-forest problem.

For all problems in this section we do the following preprocessing and postprocessing: Delete all vertices from the graph that have degree at most k (for the mixed-forest problem $k = f + p$); solve the problem for this smaller graph; add the edges incident to each deleted vertex to the solution, at most one per forest or pseudoforest. Clearly, the preprocessing and postprocessing uses linear time. The smaller graph has at most $n' = \min\{n, 2m/k\}$ vertices. Hence assume that in each of our algorithms, the number of vertices is n'.

The following result is due to [RT].

LEMMA 3.1. Cyclic scanning *uses* $O(kn')$ *time and space on matroid* $\mathscr{G}^k$.

PROOF. The operation $c(e, i)$ of the static-base circuit problem is performed as follows. Let the current partitioned independent set be $\bigcup_{i=1}^{k} I_i$, where each I_i is a forest. At the start of the search, root every tree in each I_i. Let $e = uv$. If u and v are not in the same tree of I_i, answer "independent." Otherwise output the edges of $C(e, I_i)$ that have not been output yet. Cycle $C(e, I_i)$ consists of the paths from u and v to their nearest common ancestor.

The following data structure is used to implement the algorithm. Consider a forest I_i. To decide if two vertices are in the same tree assign a tree number to each vertex. To find the paths to the nearest common ancestor assign a preorder number and parent pointer to each vertex. (Recall that an ancestor of a vertex has a lower preorder number.) To avoid outputting an edge more than once do set merging on the vertices (the ends of an edge that has been output are in the same set). In addition, each vertex has a pointer from its information in I_i to that for I_{i+1}.

Rooting the trees, assigning tree numbers, preorder numbers, and parent pointers uses $O(kn')$ time. Since the trees are initially given, the linear-time, static-tree set-merging algorithm of [GT1] can be used. Thus *start* (which initializes the data structure for the static-base circuit routines of all k copies of $\mathscr{G}$) uses $O(kn')$ time and space, and $c(e, i)$ uses $O(1)$ (amortized) time per call and edge output. □

THEOREM 3.1. *The k-forest problem can be solved in the first of the two time bounds of Table* 1 *and* $O(m)$ *space.*

PROOF. Use the algorithm of Section 2 to solve the matroid-partitioning problem for $\mathscr{G}^k$: Execute *batch* l times, obtaining a partial solution I. Then for each edge e not in I do *cyclic scanning*, and add e to I if the search is successful.

To estimate the resources note that *batch* is executed once in $O(km')$ time and $O(m)$ space. (The data structure is a simple application of dynamic trees; see [GS] and [Wes].) By Lemma 3.1 *cyclic scanning* uses $O(kn')$ time and space. These observations give the desired space bound. They also imply that the total time is $O(km'l + (kn'/l)kn')$. (The second term is the total time for *cyclic scanning*, since unsuccessful searches are free and by inequality (1).) Now choose $l = n'\sqrt{k/m'}$. □

LEMMA 3.2. $\mathscr{B}^k$ *is a matroid sum of partition matroids.*

PROOF (see also [M] and [A]). For each vertex v define M_v to be a partition matroid with two sets in the partition. One set contains all edges incident to v; an independent set can contain up to k of its edges. The other set contains all edges not incident to v; an independent set cannot contain any of its edges. We show that $\mathscr{B}^k = \bigvee_{v \in V} M_v$.

First assume $k = 1$. A set I is independent in $\mathscr{B}$ if and only if each of its connected components contains at most one cycle. The latter holds if and only if the edges of I can be directed so each vertex v has at most one outgoing edge. This implies the desired equality, since we can associate v's outgoing edge with M_v. This argument easily generalizes to arbitrary k. □

Using the lemma it is simple to show that *breadth-first scanning* can be implemented in time and space $O(kn')$. (The main observation is that if the independent set I has partition set I_v in M_v and e is an edge incident to v, then $|I_v| < k$ implies $I_v + e$ is independent and $|I_v| = k$ implies $C(e, I_v, M_v) = I_v$.) We wish to show the same bound for one execution of *batch*. For this and other properties of $\mathscr{B}^k$ it is convenient to switch to a graph model. The lemma shows that an independent set of $\mathscr{B}^k$ can be viewed as a degree-constrained subgraph [Law] as follows. Let $G = (V, E)$ be the given graph. Let B be a bipartite graph with vertices $V \cup E$ and edges ve, where v is an end of e in G. For vertex x of B set $u(x)$ equal to k if $x \in V$ and 1 if $x \in E$. A degree-constrained subgraph of B is a subgraph where each x has degree at most $u(x)$. The proof of the lemma shows that a set is independent in $\mathscr{B}^k$ if and only if it is a degree-constrained subgraph of B. Furthermore, it is simple to construct the partitioned independent set for a given degree-constrained subgraph. So we can work with degree-constrained subgraphs of B.

Now we sketch how *batch* can be implemented in $O(kn')$ time and space. Let D be the current degree-constrained subgraph. The idea is to do $O(1)$ work for each edge of D. Define

$$F = \{x | x \text{ is a vertex of } B \text{ on fewer than } u(x) \text{ edges of } D\}.$$

We search for augmenting paths by starting from the vertices of $F \cap V$. (*batch* searches for shortest augmenting paths, so it starts the search from all possible vertices on one side of the bipartite graph B. Clearly, starting from the vertices of $F \cap E$ uses too much time.) We always stop a search as soon as the first edge completing an augmenting path is found. Careful programming achieves this rule. The rule allows all work to be charged to the edges of D, giving the desired time bound. (This procedure can be interpreted in terms of the matroid representation of Lemma 3.2—it amounts to finding augmenting paths in reverse order. This approach is discussed further in Section 5.)

THEOREM 3.2. *The k-pseudoforest problem can be solved in time*

$$O(m + \min\{(kn')^{3/2}, k(n')^{5/3}\}).$$

The space bound is $O(m)$.

PROOF. Use the algorithm of Section 2 to solve the matroid-partitioning problem for $\mathscr{B}^k$: Execute *batch* l times, obtaining a partial solution I. Then, for each edge e not in I, do *breadth-first scanning*, and add e to I if the search is successful.

To analyze the time, recall that *batch* and *breadth-first scanning* each use $O(kn')$ time and space. Let Δ denote the number of edges I is short of a base. Since unsuccessful searches of *breadth-first scanning* are free the total time is $O(kn'(l + \Delta))$. Δ can be bound using inequality (1) or, as we will show,

$$\Delta \leq (2n'/l)^2. \tag{2}$$

In either case choose $l = \Delta$. This gives the desired time bound.

Inequality (2) is derived using the ideas of [ET]. We sketch the argument. Consider the current degree-constrained subgraph D of bipartite graph B. Let n_i denote the number of vertices of V whose shortest alternating path from some vertex of $F \cap V$ contains i edges of D. Since D has Δ edge-disjoint augmenting paths, it is easy to see that $n_i n_{i+i} \geq \Delta$. This implies $(l/2)\sqrt{\Delta} \leq n'$, which gives (2). □

Our second k-forest algorithm is based on an observation that follows from [PQ]: A set of edges that can be partitioned into k pseudoforests can be partitioned into $k + 1$ forests. This can be proved as in [PQ] using Nash-Williams' formula for the arboricity of a graph [N]; alternatively it is easily derived using clumps [Wes]. We call this the *stepping-stone observation*, for reasons apparent from the algorithm:

Pseudoforest Step. Solve the k-pseudoforest problem.

Convert Step. Convert the k pseudoforests into $k + 1$ forests. Initialize F to contain all but one of these $k + 1$ forests.

Complete Step. Insert as many edges of $G - F$ into F as possible, keeping F a set of k forests.

Correctness follows from the stepping-stone observation. Now we give further implementation details.

The Convert Step is done by the following divide-and-conquer scheme: The case $k = 1$ is trivial, so suppose $k > 1$. Divide the k pseudoforests into halves (containing $\lceil k/2 \rceil$ and $\lfloor k/2 \rfloor$ pseudoforests). Recursively convert both halves into forests. This gives $k + 2$ forests total. Pick any one of these forests, and insert its edges into the remaining $k + 1$ forests. Do this using *cyclic scanning*.

The Complete Step is done using *cyclic scanning* as well.

To analyze the time, first observe that the timing function T for the Convert Step obeys the recurrence,

$$T(k) = T(\lfloor k/2 \rfloor) + T(\lceil k/2 \rceil) + k(n')^2;$$
$$T(1) = n'.$$

This follows from Lemma 3.1 and the fact that a forest has at most $n' - 1$ edges. Hence the Convert Step uses $O(k(n')^2 \log k)$ time.

The Pseudoforest Step uses time $O(m + (kn')^{3/2})$ by Theorem 3.2. Clearly, the largest k-pseudoforest has at least as many edges as the largest k-forest. Thus the Complete Step adds at most $n' - 1$ edges to F. Since unsuccessful searches are free and by Lemma 3.1, the Complete Step uses time $O(k(n')^2)$. The time for the Convert Step clearly dominates.

THEOREM 3.3. *The k-forest problem can be solved in the second of the two bounds of Table* 1 *and* $O(m)$ *space.*

Our third k-forest algorithm also uses the stepping-stone observation. First solve the k-pseudoforest problem. Then solve the k-forest problem for the graph whose edges are the k pseudoforests just found. Finally add as many edges of G as possible to the k-forest just found.

Using Theorem 3.1, the time to find the first k-forest is $O(k^{3/2}n'\sqrt{kn' + n' \log n})$. The rest of the analysis follows the previous algorithm.

THEOREM 3.4. *The k-forest problem can be solved in time*

$$O((kn')^{3/2} + k^{3/2}n'\sqrt{kn' + n' \log n} + k(n')^2)$$

and $O(m)$ *space.*

For k, n, and m in a small region, this bound becomes $O(k(n')^2)$. This is superior to the other two bounds in this region.

We turn to the mixed-forest problem. We are given a graph with integers f and p, and we seek a maximum cardinality set of edges partitioned into f forests and p pseudoforests. Recall that we preprocess the graph so its has only n' vertices.

THEOREM 3.5. *The mixed-forest problem can be solved in* $O((f + p)n'\sqrt{fm'})$ *time and* $O(m)$ *space.*

PROOF. The task is to solve the matroid partitioning problem for $M = \mathscr{G}^f \vee \mathscr{B}^p$. The overall approach is to use the algorithm of Section 2 on M, treating it as the sum of $f + p$ matroids: First execute *batch* l times, obtaining a partial solution I. Then enlarge I to the desired solution by finding augmenting paths one at a time.

The *batch* routine for M is implemented as in Theorems 3.1 and 3.2. To find single augmenting paths we treat M as the sum of two matroids $\mathscr{G}^f$ and $\mathscr{B}^p$, and use *breadth-first scanning*. To implement *breadth-first scanning* we must solve the static-base circuit problem in each matroid. In $\mathscr{G}^f$ we find fundamental circuits using *cyclic scanning*. In $\mathscr{B}^p$ we find fundamental circuits using *breadth-first scanning*.

Now we estimate the time for this algorithm. The time to find a single augmenting path is $O((f + p)n')$. (Lemma 3.1 shows *cyclic scanning* on $\mathscr{G}^f$ uses time and space $O(fn')$; the discussion after Lemma 3.2 shows *breadth-first scanning* on $\mathscr{B}^p$ uses time and space $O(pn')$.) By the proofs of Theorems 3.1 and 3.2 algorithm *batch* runs in $O(fm')$ time. (The time on all graphic matroids is $O(fm')$. The time

on all bicircular matroids is clearly linear in the total number of edges, $O(m)$.) Choosing l to balance the times gives the desired bound. □

This result is best for small values of f and p, for example. the rigidity applications mentioned in Section 1. We can use the stepping-stone approach to get more efficient algorithms for large values. Details are in [Wes].

The stepping-stone approach can be used to solve weighted problems. To illustrate, consider the *weighted k-forest problem*: this problem is to find a maximum weight base of $\mathscr{G}^k$. Equivalently we are given an integer $k > 1$ and a graph G, where each edge e has a real-valued weight $w(e)$. The weight of a set of edges is the sum of the weights of all its edges. The problem is to find k edge-disjoint forests with the largest total weight possible. We start with a relation between maximum weight bases of graphic and bicircular sums.

LEMMA 3.3. *A maximum weight base of $\mathscr{B}^{k-1}$ can be enlarged to a maximum weight base of $\mathscr{G}^k$.*

PROOF. Let B be a maximum weight base of $\mathscr{B}^{k-1}$. Choose T as a maximum weight base of $\mathscr{G}^k$ containing as many edges of B as possible. The argument is by contradiction: Assume there is an element $e \in B - T$. Let B' consist of the elements of $B - e$ having weight at least $w(e)$. We will show that $e \in sp(B', \mathscr{B}^{k-1})$, contradicting the independence of B.

Consider the fundamental circuit $C = C(e, T, \mathscr{G}^k)$. Observe that $C - e \subseteq sp(B', \mathscr{B}^{k-1})$. To see this consider any element $f \in C - e$. The choice of T implies that either $w(f) > w(e)$, or $w(f) = w(e)$ and $f \in B$. In either case, $f \in sp(B', \mathscr{B}^{k-1})$ (for the first case, observe that the maximality of B implies B' spans all elements of weight larger than $w(e)$).

Because of the observation, to complete the argument we need only find a set $D \subseteq C - e$ with $e \in sp(D, \mathscr{B}^{k-1})$. The definition of circuit implies that $C - e$ consists of k trees, each spanning the same set of vertices W. Clearly, $|W| > k$, since the trees contain $k(|W| - 1)$ edges. So we can dismantle one tree, placing one of its edges in each of the remaining $k - 1$ trees, to get a $(k - 1)$ pseudoforest D spanning W. Clearly, D spans e, as desired. □

The algorithm to find a maximum weight base of $\mathscr{G}^k$ works as follows. Find a maximum weight base B of $\mathscr{B}^{k-1}$. Convert B to a k-forest (use the Convert Step of Theorem 3.3). Then run the greedy algorithm to enlarge B to a base of $\mathscr{G}^k$. (The greedy algorithm attempts to add each edge of $G - B$ to B, processing edges in nonincreasing order of weight.)

Now we analyze the time. The main observation is that at most n edges need be added to B to get the desired base. This follows since a base of $\mathscr{B}^{k-1}$ has at most $n - k$ fewer edges than a base of $\mathscr{G}^k$.

The Convert Step takes time $O(k(n')^2 \log k)$. The greedy algorithm takes time $O(m \log m)$ to sort the edges, plus $O(k(n')^2)$ time to enlarge B to a base. Now we show that the time for the Convert Step dominates. We find B using the algorithm of [GT4] for finding a maximum weight, maximum cardinality degree-constrained

subgraph of a bipartite graph. When executed on a bipartite graph with bipartition V_0, V_1, where each vertex of V_1 has degree $O(1)$, and the weights are integers of magnitude at most N, this algorithm runs in time $O(n^{2/3}\mu \log(nN))$; here n is the number of vertices and μ is the number of edges in the desired subgraph. Since $\mathscr{B}^{k-1}$ is a matroid, we can replace the given weights by integers between 1 and m, so $N = m$. Hence the time to find B is $O(k(n')^{5/3} \log n)$.

THEOREM 3.6. *The weighted k-forest problem can be solved in the bound of Table 1 and $O(m)$ space.*

4. Covering and Packing. This section gives algorithms for the covering and packing problems for an arbitrary matroid M. Applying the algorithms to graphic and bicircular matroids gives improved time bounds. The algorithms use the *balanced binary search* technique of [GT3].

A balanced binary search works in two steps. The Search Step finds a "good" value close to the desired optimum. The Postprocessing Step finds the exact optimum.

We start with an algorithm for the covering problem on an arbitrary matroid. To define "good," let Δ be a parameter. (The value of Δ is determined by the specific matroid, using efficiency considerations.) The covering problem seeks the smallest value k such that S is independent in M^k. Call the value k *good* if there is a set I that is independent in M^k and has $|I| \geq |S| - \Delta$. The Search Step finds a good value k, with corresponding set I, such that $k \leq cov(M)$; the Postprocessing Step adds elements to I, possibly increasing k, until all elements are in I and $cov(M)$ has been found. The details are as follows.

The *Search Step* is a binary search organized in probes. It maintains an interval $[l, h]$ that contains the desired value k. The value h is always a good value; l is always a value less than $cov(M)$. (Clearly, when $h = l + 1$ the desired value $k = h$ has been found.) To probe the midpoint k of the current interval, execute *batch* $|S|/\Delta$ times. If the final independent set I found by *batch* has at least $|S| - \Delta$ elements, then k is good, with corresponding set I. Hence assign h the value k. In the remaining case, $|I| < |S| - \Delta$. Inequality (1) implies that a base of M^k has fewer than $|S|$ elements, i.e., $k < cov(M)$. Hence assign l the value k.

The *Postprocessing Step* must add at most Δ elements to the solution I. For each element e it either finds an augmenting path for $I + e$, or, if unsuccessful, it increases k by one ($cov(G)$ is at least this large) and adds e into the new copy of M.

The correctness of this algorithm follows from the above discussion. On a specific matroid, the value of Δ is determined by balancing the times for the binary search and postprocessing.

The most well-known covering problem is to find the arboricity of a graph. Let Γ denote the arboricity $cov(\mathscr{G})$ and p the pseudoarboricity $cov(\mathscr{B})$. The time for a probe in the Search Step depends heavily on the probe value k. For arboricity and pseudoarboricity, good bounds are available. The following bounds are derived using clumps, and give slight improvements of [CN].

LEMMA 4.1. $\Gamma \leq (5 + \sqrt{8m - 7})/4$; *a similar bound holds for* p.

PROOF. For $\Gamma = 1$ the lemma is true by inspection. Let $\Gamma > 1$. The proof uses the fact that $M = \mathscr{G}^{\Gamma-1}$ has at least one circuit C. Thus $C - e$ is a (connected) clump in M (see Lemma 5.1). If r is the number of vertices in C, then $|C| = (\Gamma - 1)(r - 1) + 1$. This implies $(\Gamma - 1)(r - 1) \leq r(r - 1)/2$, whence $r \geq 2\Gamma - 2$. Thus $m \geq |C| \geq (\Gamma - 1)(2\Gamma - 3) + 1$. The desired bound for m follows. □

THEOREM 4.1. *The pseudoarboricity* p *can be found in* $O(m \min\{\sqrt{m \log n}, (n \log n)^{2/3}\})$ *time and* $O(m)$ *space.*

PROOF. By the proof of Theorem 3.2 the Search Step can be done in time $O((m/\Delta)m \log n)$. By Lemma 3.2 the Postprocessing Step can be done in time $O(\Delta m)$. Choosing $\Delta = \sqrt{m \log n}$ gives the first time bound.

For the second bound do each probe in the Search Step by running *batch* $2n/\sqrt{\Delta}$ times. Inequality (2) shows this is correct. Hence the Search Step runs in $O((n/\sqrt{\Delta})m \log n)$ time; the Postprocessing Step is $O(\Delta m)$. Choosing $\Delta = (n \log n)^{2/3}$ gives the desired time bound. □

This improves the best previous bound for pseudoarboricity, $O(nm \log(n^2/m))$ [GGT] (their algorithm actually solves the more general problem of finding a maximum density subgraph).

To find the arboricity the stepping-stone observation implies that $p \leq \Gamma \leq p + 1$ [PQ]. Hence first compute the pseudoarboricity p; then test if the arboricity is p; otherwise it is $p + 1$.

We implement this algorithm as follows. Let $p_0 = (m \log m)^{2/3}/(m')^{1/3}$. If $p \leq p_0$ use Theorem 3.1 to test the arboricity. In this case the arboricity is found in time $O(m\sqrt{pm'})$ (substitute m/p for n'). If $p > p_0$ use Theorem 3.3 to test the arboricity. In this case the arboricity is found in time $O((m^2/p) \log p)$ time (substitute m/p for n'). In both cases the time is at most the first of the two time bounds in Table 1. (Note that the algorithm can easily calculate p_0 to within a factor of two, which is adequate for the time bound. Note also that Theorem 4.1 shows the time to find the pseudoarboricity is a lower order of magnitude.)

For dense graphs the first bound in Table 1 is an overestimate. That is, any graph has $p \geq m/n$, so if $m/n \geq p_0$, a better estimate for the time of Theorem 3.3 is $O(mn \log n)$.

THEOREM 4.2. *The arboricity can be found in the time bounds of Table* 1 *and* $O(m)$ *space.*

Now we give an algorithm for the packing problem on an arbitrary matroid M. It uses balanced binary search. To define "good," let Δ be a parameter to be determined later; let ρ be the rank of M (i.e., the size of a base). Call the value k *good* if there is a set I independent in M^k with $|I| \geq k\rho - \Delta$.

The Search Step does a binary search to find a good k (with corresponding I)

such that $k \geq pack(M)$. In the search interval $[l, h]$, h is always a value bigger than $pack(M)$; l is always a good value. A probe for k executes *batch* $k\rho/\Delta$ times. If the final independent set I found by *batch* has at least $k\rho - \Delta$ elements, k becomes the new value l; otherwise, by (1) we can make k the new value h.

The Postprocessing Step repeats the following until the "deficiency" $k\rho - |I|$ is zero. Repeatedly try to enlarge I by an element e, by searching for an augmenting path. If no such element exists and $|I| < k\rho$, we know $pack(G) < k$. Discard the smallest of the k sets that I is partitioned into, and decrease k by one.

For the correctness simply observe that, after the Search Step, $k \geq pack(M)$.

The time is estimated as follows. After the Search Step the deficiency $k\rho - |I|$ is at most Δ. If I is enlarged by an element e, the deficiency decreases by one. If the smallest of the k sets is discarded, the deficiency decreases at least by one (the smallest set has at most $\rho - 1$ elements). Thus the Postprocessing Step is repeated at most Δ times. Note that since unsuccessful searches are free, the time for postprocessing is the same as the time for Δ searches. Choose Δ to balance both steps of the algorithm.

THEOREM 4.3. *Packing bicircular matroids can be done in time*

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

and space $O(m)$.

PROOF. The Postprocessing Step is done using *breadth-first scanning* on $\bigvee_{v \in V} M_v$ (see Lemma 3.2).

By the proof of Theorem 3.2 and since $pack(\mathscr{B}) \leq m/n$, the Search Step runs in time $O((m/\Delta)m \log(m/n))$. By Lemma 3.2 the search for an augmenting path for a given element e is $O(m)$. Since unsuccessful searches are free, each iteration of the Postprocessing Step runs in $O(m)$ time. So postprocessing is $O(m\Delta)$. Choosing $\Delta = \sqrt{m \log(m/n)}$ gives the first time bound.

Alternatively do each probe in the Search Step by executing *batch* $2n/\sqrt{\Delta}$ times. Inequality (2) shows this is correct. Thus the Search Step runs in $O((n/\sqrt{\Delta})m \log(m/n))$ time. Choosing $\Delta = (n \log(m/n))^{2/3}$ gives the second time bound. □

Our first graphic packing algorithm is the above matroid algorithm.

THEOREM 4.4. *Graphic packing can be done in the first time bound stated in Table* 1 *and* $O(m)$ *space.*

PROOF. Use Theorem 3.1 for the Search Step. Use *cyclic scanning* for the Postprocessing Step.

By the proof of Theorem 3.1 the Search Step runs in time

$$O((m/\Delta)(m/n)m' \log(m/n)).$$

Using Lemma 3.1 and the same reasoning as in Theorem 4.3 each iteration of the

Postprocessing Step runs in $O(m)$ time. Choose Δ to balance the two terms for the time. □

The relation between graphic and bicircular matroids for packing is weaker than covering; we only have $pack(\mathscr{G}) \leq pack(\mathscr{B}) + 1$. To see this assume G contains k spanning trees. The kth spanning tree can be used to turn the remaining $k-1$ spanning trees into bases for $\mathscr{B}$, since $k - 1 \leq n - 1$. The same reasoning applies to graphs that are not connected.

Our second graphic packing algorithm first does bicircular packing, and then converts the solution to $k = pack(B) + 1$ forests. Then it does the Postprocessing Step of the general packing algorithm, using *cyclic scanning.*

THEOREM 4.5. *Graphic packing can be done in the second time bound stated in Table* 1 *and* $O(m)$ *space.*

PROOF. The correctness of the algorithm follows from the stepping-stone observation together with $pack(\mathscr{G}) \leq pack(\mathscr{B}) + 1$.

By Theorem 4.3 bicircular packing can be done in time $O(m\sqrt{m \log(m/n)})$. Using the Convert Step of Theorem 3.3, the conversion to forests can be done in time $O(nm \log(m/n))$ (substitute n for n' and m/n for k). By Lemma 3.1 and the same reasoning as in Theorem 4.3 each iteration of the Postprocessing Step runs in $O(m)$ time.

Let $k = pack(\mathscr{B}) + 1$. If $pack(\mathscr{B}) = 1$ the problem is trivial so assume the opposite. Thus a base of $\mathscr{B}$ contains n edges (even if G is not connected). Since the solution for $\mathscr{B}$ has $(k-1)n$ edges, the deficiency after the Convert Step is at most $k(n-1) - (k-1)n = n - k$. Thus the Postprocessing Step runs in time $O(nm)$. The Convert Step dominates. □

5. Top Clumps. This section proves a number of facts about clumps, the top clump, and dual matroids. These facts shed light on the structure of matroid sums and are used in Sections 6 and 7.

Throughout this section unless stated otherwise M denotes a matroid sum $M = \bigvee_{i=1}^{k} M_i$ on S. Recall the basic definition: L is a clump of M if and only if it is independent in M and $|L| = \sum_{i=1}^{k} \rho(L, M_i)$. An equivalent condition is that L can be partitioned into sets L_i independent in M_i with $L \subseteq sp(L_i, M_i)$, $i = 1, \ldots, k$; equivalently, L is independent and *any* partition into sets L_i independent in M_i has $L \subseteq sp(L_i, M_i)$, $i = 1, \ldots, k$.

Clumps make a matroid sum interesting. If M has no clumps, then it has no circuits, i.e., every set is independent. This is shown in the following lemma; it is illustrated by Figure 1, where the subgraph induced by vertices $4, \ldots, 8$ is a circuit.

LEMMA 5.1. *If element e is in a circuit C of M, then $C - e$ is a clump.*

PROOF. Try to add element e to the independent set $C - e$, using *breadth-first scanning.* Every element gets labeled but the search is unsuccessful. As shown in Lemma 2.1 the labeled elements $C - e$ form a clump. □

The next property is not needed in the remainder of the paper. We include it because, as we shall see, it gives an alternate, high-level justification of *breadth-first* and *cyclic scanning*.

THEOREM 5.1. *Let L be a clump of M. Then $M/L = \bigvee_{i=1}^{k} M_i/L$.*

PROOF. We show that a set $A \subseteq S - L$ is independent in M/L if and only if it is independent in $\bigvee_{i=1}^{k} M_i/L$. If A is independent in M/L, then $A \cup L$ is independent in M and thus can be partitioned into sets $A_i \cup L_i$, $i = 1, \ldots, k$, where each $A_i \subseteq A$, $L_i \subseteq L$, and $A_i \cup L_i$ is independent in M_i. Since L is a clump, L_i is a maximal independent subset of L in M_i. Hence A is independent in $\bigvee_{i=1}^{k} M_i/L$.

The converse statement—independence in $\bigvee_{i=1}^{k} M_i/L$ implies independence in M/L—holds for any independent set L, not necessarily a clump. □

This fact gives an alternate justification for omitting *start* operations after an unsuccessful search in *breadth-first* or *cyclic scanning*: Suppose an unsuccessful search labels elements L. The next search for an augmenting path can be conducted in the matroid M/L, since L is already in the independent set and will not leave it. As noted in Lemmas 2.1 and 2.2, L is a clump. Thus $M/L = \bigvee_{i=1}^{k} M_i/L$. Thus each matroid M_i can be replaced by M_i/L. Equivalently, the elements of L need never be output by a c operation again. This is just what omitting the *start* operation accomplishes.

We turn to the main notion of this section. For an arbitrary set A, a *top clump of A* is a clump of M that is maximal in A. A *top clump of M* is a top clump of S. In the subgraph of Figure 1(b) the subgraph induced by vertices $4, \ldots, 8$ is the unique top clump.

Suppose A is independent in M. Then the clumps contained in A are closed under union. This follows since if clumps K and L have $K \cup L$ independent, say as a partitioned independent set $\bigcup_{i=1}^{k} K_i \cup L_i$, then

$$K \cup L \subseteq sp(K_i, M_i) \cup sp(L_i, M_i) \subseteq sp(K_i \cup L_i, M_i).$$

Hence an independent set A has a unique top clump. (The clumps of an independent set are also closed under intersection. This and other properties are discussed in [Wes].) Theorem 5.3 below shows that a top clump of a set determines its span. First note a related fact, a clump L has $sp(L, M) = \bigcap_{i=1}^{k} sp(L, M_i)$.

THEOREM 5.2. *An arbitrary subset A of S, with a top clump L, has $sp(A, M) = A \cup sp(L, M)$.*

PROOF. Let B be a maximum cardinality independent subset of A that includes L. Clearly, $sp(A, M) = sp(B, M)$. The elements of $sp(B, M) - B$ are actually spanned by L (Lemma 5.1). This implies the theorem. □

The theorem implies that the top clump L of any base B of M is a top clump of M (since any element not in B is in $sp(L, M)$).

Recall that in any matroid, a *bridge* is an element that is in every base; a *circuit element* is an element that is in some circuit. It is easy to see that the circuit elements are precisely the nonbridges; also if B is any base, the circuit elements are precisely the elements in some fundamental circuit with respect to B. Any top clump L consists of bridges and circuit elements, where the circuit elements of L span all circuit elements of M.

If we are given a partitioned base B of M and execute *breadth-first* or *cyclic scanning* on the elements not in B, then every search is unsuccessful, the labeled elements are precisely the circuit elements and the unlabeled elements are the bridges. Furthermore, the preprocessing of Section 3 is valid—vertices of degree at most k can be deleted, since all their incident edges are bridges. Hence, given a partitioned base, we can find the bridges of a k-fold graphic sum in time $O(m)$. Note that this finds the circuit elements in the top clump of B (i.e., they are all the labeled elements of B). However, this algorithm does not find the bridge elements of the top clump. We present an algorithm to find the top clump below.

First we give some terminology. It is useful to enlarge the matroid M with a copy of an element. More precisely fix an element $e \in S$ and let $\bar{e}$ be a new element not in S, parallel to e in each M_i. Let $\bar{M}_i$ denote the corresponding matroid on $S + \bar{e}$. The matroid sum $\bar{M} = \bigvee_{i=1}^{k} \bar{M}_i$ is called *the matroid M with e duplicated (by $\bar{e}$)*. If a set A contains e, then certainly $e \in sp(A, \bar{M})$, but we may not have $\bar{e} \in sp(A, \bar{M})$ (recall the definition of span, reiterated in Section 1).

Define

$$T = \{e \mid \text{duplicating } e \text{ does not increase } \rho(M)\};$$

more exactly, the matroid M with e duplicated has the same rank as M. This set relates all top clumps:

THEOREM 5.3. *Any top clump L of M has $sp(L, M) = T$.*

PROOF. Consider any element $e \in S$ and form the matroid M with e duplicated by $\bar{e}$. Note that e is spanned by L if and only if $\bar{e}$ is spanned by L. (This is trivial if $e \notin L$; if $e \in L$ it follows from the definition of clump.)

Obviously $e \in sp(L, M)$ implies $e \in T$. Conversely, if $e \in T$ then, letting B denote a base of M containing L, the fundamental circuit $C(\bar{e}, B, \bar{M})$ exists. Lemma 5.1 implies that $\bar{e}$ is spanned by L. Thus e is spanned by L. □

As a corollary note that if L_1, L_2 are two top clumps of M, then $sp(L_1, M_i) = sp(L_2, M_i)$ for $i = 1, \ldots, k$. This follows since $L_1 \subseteq T = \bigcap_{i=1}^{k} sp(L_2, M_i)$.

The theorem gives the following algorithm to find a top clump. Suppose that we have a partitioned base $B = \bigcup_{i=1}^{k} I_i$ of M.

Test Step. For each element $e \in B$, check if $B + \bar{e}$ is independent in $\bar{M}$. The top clump consists of the elements that check out dependent.

The algorithm is correct since B contains a top clump L, and $L = B \cap sp(L, M) = B \cap T$. Clearly the Test Step tests membership in T.

We give two implementations of the Test Step. The first uses *breadth-first* or *cyclic scanning* on M. For each $e \in B$ search for an augmenting path for $\bar{e}$ using the static-base circuit routines (see Section 2). The search for an augmenting path may label all elements of B. Thus in the worst case, the static-base circuit routines output B, $|B|$ times.

The second implementation avoids outputting elements multiple times. It accomplishes this by finding augmenting paths in reverse. To do this we introduce a matroid that will be studied in the remainder of this section—recalling that $M = \bigvee_{i=1}^{k} M_i$ define

$$M' = \bigvee_{i=1}^{k} M_i^*.$$

We now show that augmenting paths in M are essentially the reverse of such paths in M'.

Consider a partitioned independent set $I = \bigcup_{i=1}^{k} I_i$ of M with a shortcut-free partial augmenting path $e_0, \ldots, e_s$. Define $J_i = sp(I_i, M_i) - I_i$, so J_i is independent in M_i^*. Define independent set $J = \bigcup_{i=1}^{k} J_i$ of M'. Then the reverse path $e_s, \ldots, e_0$ is a shortcut-free partial augmenting path for J in M'. (Strictly speaking $J = \bigcup_{i=1}^{k} J_i$ is not a partitioned independent set, since the sets J_i are not necessarily disjoint. We could correct this by introducing multiple copies of elements that are in more than one J_i.) The observation can be used to find augmenting paths by working with matroid duals, when this is more efficient.

The second implementation of the Test Step uses this observation as follows. Recall $B = \bigcup_{i=1}^{k} I_i$; let A be the set of all elements of $e \in B$ such that, for some i, $e \notin I_i$ and $I_i + e$ is independent in M_i. Let J be the above independent set of M' corresponding to B. Do *breadth-first scanning* on M' with independent set J, starting by labeling not one element as usual, but rather all elements of A. It is easy to see that the top clump of M consists of all elements that do not get labeled in the search.

Now consider the special case of a k-fold sum matroid $M = N^k$. We can find augmenting paths using *cyclic scanning on* M. Thus we can use *cyclic scanning* on M'. For increased efficiency note that in M_i the only elements that get used in augmenting paths are in $I_{i-1} \cup I_i$. Thus we can restrict the *ith* component matroid of M' to these elements, i.e., use *cyclic scanning* on matroid

$$M'' = \bigvee_{i=1}^{k} N^*|(I_{i-1} \cup I_i),$$

with partitioned independent set $\bigcup_{i=1}^{k} I_{i-1} \cap sp(I_i)$.

THEOREM 5.4. *Given $M = \mathscr{G}^k$ with a partitioned base B, a top clump can be found in time $O(kn \log n)$ and space $O(m)$.*

PROOF. Use the *cyclic scanning* implementation of the Test Step. In M'' the ith matroid has $|I_{i-1} \cup I_i| = O(n)$ elements. The static-base circuit routine for $\mathscr{G}^*$ uses

time $O(m + n \log n)$ for a graph with n vertices and m edges [GS]. So each of the k matroids composing M'' uses $O(n \log n)$ time, giving the time bound. □

The rest of this section proves some useful facts about the matroid sum $M = \bigvee_{i=1}^{k} M_i$ on S and its corresponding sum M'. We first relate the bases of M and M'.

Start with the case $k = 2$. B_1, B_2 is an *extremal pair* of bases if B_i is a base of M_i, $i = 1, 2$, and their intersection is as small as possible. The bases of M and the extremal pairs correspond as follows: An extremal pair B_1, B_2 gives a partitioned independent set $B_1 \cup (B_2 - B_1)$. A partitioned base $B = I_1 \cup I_2$ gives a pair of bases B_1, B_2 if we enlarge I_i to B_i by adding elements of I_{3-i}. (This can be done since B is a base.) This correspondence actually takes an extremal pair to a partitioned base and vice versa. To see this let n_i be the rank of M_i, let b be the size of a base of M, and let c be the cardinality of the intersection for an extremal pair. The constructions imply the relation $b + c = n_1 + n_2$, which gives the desired conclusion.

Next observe that an extremal pair B_1, B_2 of M corresponds to an extremal pair $S - B_1$, $S - B_2$ of M'. This follows since if the first pair has intersection cardinality c, then de Morgan's law shows the second pair has intersection cardinality $m - (n_1 + n_2 - c)$.

We have validated the following construction.

THEOREM 5.5. *Let $B = I_1 \cup I_2$ be a partitioned base for $M = M_1 \vee M_2$. Enlarge I_i to a base B_i of M_i by adding elements of I_{3-i}. Then $B' = (S - B_1) \cup (B_1 - B_2)$ is a partitioned base for $M' = M_1^* \vee M_2^*$.*

The construction generalizes to $k > 2$ as follows. Bases B_i, $i = 1, \ldots, k$, are *extremal* for M if B_i is a base of M_i and their intersection $\bigcap_{i=1}^{k} B_i$ is as small as possible. Let $\hat{S}$ denote the set S with each element e duplicated $k - 2$ times, i.e., there are a total of $k - 1$ copies of e. Let $\hat{M}$ denote the corresponding matroid sum over $\hat{S}$. The bases of $\hat{M}$ and the extremal bases of M correspond, as above. The new relation is that if n_i is the rank of M_i, b is the size of a base of $\hat{M}$, and c is the cardinality of the intersection for an extremal set, then $b + c = \sum_{i=1}^{k} n_i$.

The extremal sets of M and the bases of M' correspond in the obvious way. The relation is that if c is the cardinality of the intersection for an extremal set and d is the size of a base of M', then $m = c + d$.

Now we relate several sets. Take M, M' as above (k arbitrary). Let C be the set of all circuit elements in M. Choose any top clump L of M and define T as above. Clearly, $C \subseteq T$. Similarly, for M' define circuit elements C', top clump L', and set T'; $C' \subseteq T'$. We first analyze the case $k = 2$.

THEOREM 5.6. *For $k = 2$, set T and C' are complements (in S); similarly, T' and C are complements.*

PROOF. By duality it suffices to show the first relation. We show that an element $e \notin T$ if and only if $e \in C'$. Consider a base B of M. Clearly, $e \notin T$ if and only if

$B + \bar{e}$ is independent in the matroid $\bar{M}$ with e duplicated. Equivalently, B can be partitioned into sets I_1, I_2 with $e \in I_1$ and $I_2 + e$ independent. Theorem 5.5 shows this amounts to M' having a base not containing e. This is equivalent to $e \in C'$. □

The above results are what is needed for the applications in the next two sections. We conclude by generalizing Theorem 5.6 to arbitrary k.

THEOREM 5.7. *For $k \geq 2$, sets T and C' are disjoint, as are T' and C.*

PROOF. We prove the first relation; the second follows by duality. Consider any element $e \in T$. Recall the correspondence between bases of $\hat{M}$, extremal bases of M, and bases of M'. Clearly, any base of $\hat{M}$ contains at most one copy of e. Thus e is not in the intersection set of any extremal bases. This is equivalent to e in every base of M'. Thus $e \notin C'$. □

The last two theorems generalize previous work. Bruno and Weinberg [BW2] define the (*augmented*) *k-minor* of a matroid N as the minimal (maximal) set $X \subseteq S$ maximizing $|X| - k\rho(X, N)$. Clumps generalize these notions in the following sense. If $M = N^k$, then C (T) is the (augmented) k-minor of N [Wes]. For the case $k = 2$, Kishi and Kajitani [KK] introduced the *principal partition* (D_2, G_2, H_2) of a graph G. Bruno and Weinberg generalized the principal partition to an arbitrary matroid N, showing S partitions into the 2-minor of N, the 2-minor of N^*, and the rest of S; further, G_2 is the 2-minor of matroid $\mathcal{G}$, H_2 is the 2-minor of matroid $\mathcal{G}^*$, and D_2 is the remaining elements. Theorem 5.6 generalizes this to an arbitrary matroid sum with two component matroids, implying that S partitions into C, C' and the remaining elements; further, in the special case $M = N \vee N$, C is the 2-minor of N and C' is the 2-minor of N^*.

6. Frameworks and Rigidity. This section discusses rigidity problems that arise in the study of frameworks modeled by graphs. A number of these problems, listed in Section 1, are equivalent to problems treated in Section 3. Here we discuss some less-obvious applications. Specifically for two-dimensional bar-and-joint frameworks, efficient algorithms are given to test a framework for redundant bars and to compute the number of degrees of freedom. An efficient algorithm is given to test for k-irreducibility, a notion relating to stresses in a bar-and-body framework of arbitrary dimension. The algorithms of this section are based on top clumps and a related notion, preclumps.

We briefly summarize bar-and-joint frameworks and related concepts; a complete discussion is in [LY], see also [R]. A *bar-and-joint framework* consists of rigid bars connected with universal joints, in some dimension space $\mathbf{R}^d$. It is in *generic position* if the coordinates of the joints (points) are algebraically independent. A framework corresponds to a graph $G = (V, E)$ (vertices correspond to the joints, edges correspond to the bars). If a framework is in generic position in the plane, its rigidity properties are completely determined by its graph alone.

Laman [Lam] and Lovász and Yemini [LY] describe the graph properties that correspond to various rigidity properties. We present algorithms for the two most important of these.

The first property, generic independence, is the absence of redundant bars (for a planar framework). Put another way, a set of edges A is *generic independent* if each edge of A takes away one degree of freedom of motion (a more precise definition is in [LY]; also see below). Lovász and Yemini show that A is generic independent if and only if $|A'| \leq 2|V(A')| - 3$ for every nonempty set $A' \subseteq A$. (This generalizes Laman's theorem [Lam], which is stated below.) As an example, the subgraph obtained from Figure 1(b) by deleting edge 78 is generic independent; this can be verified by observing that its only clump is $\varnothing$ and then using the following result.

A set of elements is *clumpless* for a matroid sum M if it does not contain a nonempty clump of M. A set A is generic independent if and only if it is clumpless for $\mathscr{G} \vee \mathscr{G}$. To see this, note that a set A is independent in $\mathscr{G} \vee \mathscr{G}$ if and only if $|A'| \leq 2|V(A')| - 2$ for every nonempty $A' \subseteq A$. (This follows easily from Lemma 5.1.) A connected clump of $\mathscr{G} \vee \mathscr{G}$ is an independent set A with $|A| = 2|V(A)| - 2$. Since a set contains a clump if and only if it contains a connected clump, the desired condition follows.

This gives a way to test if a graph is generic independent: check $m \leq 2n - 3$; if so check it is independent in $\mathscr{G} \vee \mathscr{G}$; if so use the partitioned base found in this check to test if the top clump is empty. Implementing this with the procedures of Theorems 3.1 and 5.4 gives the following.

THEOREM 6.1. *A graph can be tested for generic independence in time* $O(n\sqrt{n \log n} + m)$ *and space* $O(m)$.

The second rigidity property is the number of degrees of freedom. Intuitively a framework has d degrees of freedom if it has d independent nontrivial motions. (Any planar framework has three independent trivial motions—translation in the x and y directions and rotation. These are not counted as degrees of freedom.) Let $f(A)$ denote the number of degrees of freedom of a set of edges A; $f(E)$ is the number of degrees of freedom of the framework G. For any set of edges A of a planar framework, $f(A) = 2n - 3 - |B|$, where B is a maximum cardinality generic-independent subset of A [LY]. (Hence for a generic-independent set A, $f(A) = 2n - 3 - |A|$, justifying the intuitive definition above.) Thus computing the number of degrees of freedom of a planar framework is equivalent to finding a maximum cardinality generic-independent set.

The generic-independent sets form a matroid, the *rigidity matroid* $\mathscr{R}(G)$ ($\mathscr{R}$, when G is understood) [LY]. Thus our problem amounts to finding a base of $\mathscr{R}$. As in Section 3, the algorithm builds up a base by considering edges one by one, adding those that preserve generic independence and rejecting the others. To make this efficient we need another concept to facilitate rejecting edges. In what follows $sp(A)$ denotes the span of A in the graphic matroid $\mathscr{G}$ (not $\mathscr{R}$).

A set of edges P is a *preclump* if it is generic independent and has $|P| = 2|V(P)| - 3$. (Some preclumps in Figure 1(b) are the subgraph induced by vertices

1, ..., 4, and the subgraphs induced by 4, ..., 8 or 1, ..., 8 if we delete edge 78 from both.) This corresponds to the main notion in Laman's theorem [Lam] which amounts to the fact that P is a preclump if and only if P is minimally rigid, i.e., when placed in generic position P is rigid but removing any bar makes it nonrigid.

Clearly, P is a preclump if and only if it is clumpless in $\mathscr{G} \vee \mathscr{G}$ and $|P| = 2|V(P)| - 3$. (As a trivial example, any single edge forms a preclump.) A preclump has a property similar to clumps: if $e \in sp(P) - P$, then $P + e$ is not generic independent. Also a preclump is connected—as an independent set of $\mathscr{G} \vee \mathscr{G}$, P partitions into a spanning tree of $V(P)$ and a spanning forest of $V(P)$ with two trees.

Let P_1, P_2 be preclumps contained in a generic independent set. In general their union is not a preclump. This is so even for two preclumps with a common vertex. (Thus a given vertex can be in many preclumps.) However, if P_1 and P_2 have two or more common vertices, then $P_1 \cup P_2$ is a preclump. To prove this we need only check that $|P_1 \cup P_2| = 2|V(P_1 \cup P_2)| - 3$. Construct a new edge e joining two common vertices (if this edge already exists, e is a new edge parallel to it). In the matroid sum $\mathscr{G} \vee \mathscr{G}$, $P_1 \cup P_2 + e$ is independent and, for $i = 1, 2$, $P_i + e$ is a clump (since $e \in sp(P_i)$). Thus the union of the clumps, $P_1 \cup P_2 + e$, is a connected clump. Hence $P_1 \cup P_2$ has the desired number of edges. In what follows we use only a special case of this principle: if P_1 and P_2 have a common edge, then $P_1 \cup P_2$ is a preclump.

Now we give our algorithm to find a maximum cardinality generic-independent set B. Let M be the matroid $\mathscr{G} \vee \mathscr{G}$. The algorithm maintains a partition of the edges of B into preclumps. Initially both B and its edge partition are empty. The algorithm executes the following steps for each edge e in turn.

Preclump Test Step. If some preclump (of the current edge partition of B) contains both vertices of e, reject e and continue by processing the next edge.

Independence Test Step. Let $\bar{M}$ denote matroid M with e duplicated by $\bar{e}$. If $B + e + \bar{e}$ is independent in $\bar{M}$, then add e to B, make e a singleton preclump of B, and continue by processing the next edge.

New Preclump Step. Reject e. Merge all preclumps containing an edge of $C(\bar{e}, B + e, \bar{M}) - \{e, \bar{e}\}$ into one new set of the edge partition. Continue by processing the next edge.

Now we prove that the algorithm is correct. This amounts to showing that B is always generic independent, an edge e is rejected only if $B + e$ is not independent, and the edge partition is maintained according to its definition.

Consider the Preclump Test Step. Clearly, $e \in sp(P')$ for some preclump $P' \subseteq B$. Thus $P' + e$ is not generic independent and e should be rejected.

Consider the Independence Test Step. If $B + e + \bar{e}$ is independent in M, then clearly $B + e$ does not have a clump containing e. Thus $B + e$ is clumpless (since B is clumpless). Hence adding e to B preserves independence.

Finally consider the New Preclump Step. In this step $B + e + \bar{e}$ is dependent in $\bar{M}$. By Lemma 5.1, $C(\bar{e}, B + e, \bar{M}) - \bar{e}$ is a clump in $\bar{M}$. Hence e should be rejected, and $C(\bar{e}, B + e, \bar{M}) - \{e, \bar{e}\}$ is a preclump. This implies the merging in

this step is correct, since the union of preclumps of B sharing an edge is a preclump. We conclude the entire algorithm is correct.

The algorithm is implemented as follows. It maintains a partition of B into two forests F_i, $i = 1, 2$. The Independence Test and updating this partition are done using *cyclic scanning.*

The other main data structure is for the preclumps. Recall that the preclumps partition the edges of B (but not the vertices). The partition is maintained using an algorithm for disjoint set merging [Tar]. Each edge of B is in a unique set, its preclump. In addition each preclump P stores its "root," defined as follows. For each forest F_i, $i = 1, 2$, root each tree of F_i at an arbitrary vertex. Recall that P is split by the F_i into two forests spanning $V(P)$—a spanning tree of $V(P)$ in F_j ($j = 1$ or 2) and a forest of two trees in F_{3-j}. The root of P is the root of the spanning tree.

The Preclump Test for an edge $e = uv$ is implemented by performing the following test, for $i = 1, 2$. Define e_u as the edge from u to its parent in F_i (e_u is undefined if u is the root of a tree in F_i). Define e_v similarly. Then u and v are in a common preclump if e_u and e_v are in the same set, or the set containing e_u has root v, or the set containing e_v has root u. If neither test for $i = 1, 2$ indicates a common preclump, then u and v are not in a common preclump.

Correctness of this implementation follows from the fact that if a preclump P contains both u and v, then the test is passed for the index i where F_i contains the spanning tree of P.

Now we show that the time is $O(n^2)$. Note that an efficient set-merging algorithm performs $O(m)$ *find* operations and $O(n)$ *union* operations in time $O(m\alpha(m, n))$ which is within this bound [Tar].

An execution of the Preclump Test Step uses $O(1)$ time plus the time for $O(1)$ *finds* (these operations compute set names for the test). This gives total time $O(m)$ plus $O(m)$ *finds.*

Next observe that each time the Independence Test Step is executed, either B is enlarged or at least two preclumps are merged (in the New Preclump Step). Thus the Independence Test Step and New Preclump Step are executed $O(n)$ times.

The Independence Test Step is done with *cyclic scanning* which uses $O(n)$ time. If e is added to B, the partition of B into forests is updated, along with the preorder numberings and names of preclump sets, all in $O(n)$ time. If e is rejected, recall that *cyclic scanning* marks the fundamental circuit $C(\bar{e}, B + e, \bar{M})$. The New Preclump Step merges preclumps by *union* operations, maintaining appropriate set names. This gives $O(n)$ *unions* total.

THEOREM 6.2. *A maximum cardinality generic-independent set can be found in time $O(n^2)$ and space $O(m)$.*

Observe that the above algorithm also finds the bridges of the rigidity matroid $\mathscr{R}$: Call a preclump of the algorithm's edge partition *trivial* if it consists of a single edge, else *nontrivial.* Then the bridges of $\mathscr{R}$ are precisely the edges in trivial preclumps. This follows from two observations: In the New Preclump Step, $C(\bar{e}, B + e, \bar{M}) - \bar{e} = C(e, B, \mathscr{R})$, so the nontrivial preclumps consist of only edges

in fundamental circuits (of $\mathscr{R}$). In the Preclump Test Step if e is rejected, its fundamental circuit is already contained in a nontrivial preclump of the algorithm. Thus the nontrivial preclumps are precisely the edges in fundamental circuits.

As an application, [S] calls a graph edge birigid if it has maximum rank $2n - 3$ in $\mathscr{R}$ and no bridges. Thus our algorithm can test if a graph is edge birigid.

Another application is to find a maximum weight base of $\mathscr{R}$. Clearly, it suffices to use the above algorithm, considering the edges of G in order of nonincreasing cost. Thus a maximum weight base of $\mathscr{R}$ can be found in time $O(n^2 + m \log m)$.

A *bar-and-body framework* consists of rigid bars attached to rigid bodies with universal joints. Let $G = (V, E)$ be a multigraph representing a bar-and-body framework in generic position. White and Whiteley [WW] call a bar-and-body framework *k-irreducible* if and only if $|E| = k(|V| - 1)$ and $|E'| < k(|V'| - 1)$ for every nonempty proper subgraph (V', E'). The intuitive meaning of this property is that any stress put on a k-irreducible structure (in n-space, where $k = n(n + 1)/2$) propagates to all bars in the structure. There seems to be no obvious algorithm to test k-irreducibility (White and Whiteley do not give one).

Clearly, a bar-and-body framework is k-irreducible if and only if $\mathscr{G}^k$ has a unique nonempty clump, E. This can be tested as follows.

k-forest Step. Solve the k-forest problem for G. If G does not partition into k spanning trees it is not k-irreducible.

Top Clump Step. Remove any edge e from G and compute the top clump of $G - e$. If the top clump is nonempty, G is not k-irreducible.

Circuit Step. Create a new edge $\bar{e}$ parallel to e. Compute the fundamental circuit $C = C(\bar{e}, E, \mathscr{G}^k)$. If $C - \bar{e} = E$, then G is k-irreducible else it is not.

Correctness of this algorithm is proved as follows. Suppose it reaches the Circuit Step. So E is a clump in $\mathscr{G}^k$ and $E - e$ is clumpless. Thus any clump contains e. The smallest clump containing e is $C - \bar{e}$. This justifies the Circuit Step.

The time is estimated as follows. The k-forest Step runs in time $O(k^{3/2}n\sqrt{m'})$ by Theorem 3.1. The Top Clump Step uses time $O(kn \log n)$ by Theorem 5.4. The Circuit Step, using *cyclic scanning*, runs in time $O(kn)$.

THEOREM 6.3. *A graph can be tested for k-irreducibility in time* $O(k^{3/2}n\sqrt{m'})$ *and space* $O(m)$.

7. The Shannon Switching Game. This section analyzes the structure of the Shannon switching game in terms of top clumps and preclumps. This leads to efficient algorithms for two types of queries.

A *Shannon switching game* is specified by a triplet G, u, v where $G = (V, E)$ is an undirected graph with $u, v \in V$. Two players *join* and *cut* are to take turns claiming an edge of G. Initially each edge is unclaimed. Each edge can be claimed just once. Join wins if his edges contain a path from u to v. Cut wins if he prevents join from doing so. Any Shannon switching game is either a *join game* (join can win

against all possible strategies of cut), a *cut game* (cut can win against all possible strategies of join), or a *neutral game* (whoever plays first can win). In all three cases it is easy to execute a winning strategy if we are given two appropriate trees or cotrees [Wel]. (The reader may enjoy showing that, using dynamic trees, the winning strategy can be executed in time $O(\log n)$ per move for join and $O(\log^2 n)$ for cut. Our original solution achieving this time for cut used topology trees [F]; Bob Tarjan pointed out the elegant solution using dynamic trees.)

We introduce the problem of *Shannon switching game queries*: Fix the graph G. A *classification query* consists of two vertices u, v. The answer classifies the game G, u, v as join, cut, or neutral. A *tree query* outputs the trees or cotrees that enable the winning strategy to be played.

Previous work analyzes the Shannon switching game by concentrating on a single query; changing u or v necessitates rerunning the algorithm. Bruno and Weinberg [B], [BW1] allow changing u and v and give an efficient solution, but uv is restricted to being an edge of G and it cannot be claimed—it is a "nonplayable" edge. Our algorithm works for any pair u, v. Furthermore, it takes advantage of a fixed graph—after the first query the algorithm is significantly faster.

We analyze the game using two cases: first, $uv \in E$ but is nonplayable; then the "natural" case of no nonplayable edges (i.e., either $uv \in E$ and is playable or $uv \notin E$). The nonplayable case may be considered an artifact of our analysis (although it is the case that is considered by other researchers).

Consider the nonplayable case. Throughout the discussion of this case fix uv as a nonplayable edge of E. Let $M = \mathcal{G} \vee \mathcal{G}$ and $M' = \mathcal{G}^* \vee \mathcal{G}^*$. Let C be the union of all circuits in M and let C' be the union of all circuits in M'.

The analysis of the Shannon switching game as developed collectively in [Le], [E2], and [BW1] (see also [Law]) solves the nonplayable case. We restate the results in terms of clumps. The condition for G, u, v to be a join game is that there is a clump L in M such that $uv \in sp(L, M) - L$; equivalently, $uv \in C$ (recall Lemma 5.1). Cut and neutral games are similar: The condition for G, u, v to be a cut game is that there is a clump L' in M' such that $uv \in sp(L', M') - L'$; equivalently, $uv \in C'$. The condition for G, u, v to be a neutral game is that G, u, v is not a join game and there is a clump L in M containing uv; equivalently, G, u, v is not a cut game and there is a clump L' in M' containing uv. (If it is not immediately clear why the three alternatives are mutually exclusive, recall Theorem 5.6.) In each case a winning strategy can be executed if we have the above clump L (L') as a partitioned independent set.

LEMMA 7.1. *Let G be a fixed graph. Assume uv is always an edge of G but is not playable. Then a sequence of classification (tree) queries G, u, v can be answered in $O(n\sqrt{m'})$ preprocessing time, $O(m)$ space, and $O(1)$ ($O(n)$) query time.*

PROOF. Consider classification queries. Clearly, once the sets C and C' are known, a query can be answered in $O(1)$ time (and $O(m)$ space). The sets are found as follows. Compute a base B of M using Theorem 3.1. The edges labeled in unsuccessful searches of *cyclic scanning* form the set C (see Lemma 5.1). Next

compute the top clump L of B. Finally compute C' using the relation $C' = E - sp(L, M) = E - (L \cup C)$, which follows from Theorem 5.6.

This preprocessing uses time $O(n\sqrt{m'})$. In proof, by Theorem 3.1 B and C are found in time $O(n\sqrt{m'})$. By Theorem 5.4, L is found in time $O(n \log n)$.

Next consider tree queries. The preprocessing consists of finding base B and circuit elements C for M, as above. In doing this some additional information is recorded: each edge $e \in B \cap C$ is labeled with a "cocircuit edge" f that can replace it in B, i.e., $B - e + f$ is a base. (In constructing B, edge e was labeled in an unsuccessful search from some edge f, since $e \in C$. Hence the preprocessing can record the desired label f for e.)

Now consider a tree query u, v. Suppose $uv \in C$ so the game is join. Let e be the cocircuit edge of uv if $uv \in B$, otherwise $e = uv$. Attempt to add uv to B using *cyclic scanning*. The search is unsuccessful and marks a clump L of B. If $uv \notin B$, L spans uv, so output L. If $uv \in B$, $L - uv + e$ is a clump that spans uv, so output it. (Note that in both cases the clump is returned as a partitioned independent set.)

Now suppose $uv \notin C$. Thus the game is cut or neutral; also clearly $uv \in B$. Attempt to add a copy $\overline{uv}$ to B using *cyclic scanning*. If the search fails it marks a clump L containing uv. The game is neutral; output L to answer the query for the join strategy of the neutral game. If the search succeeds we will show below that the game is cut. For both cut and neutral games we must output an appropriate clump of M'. We show how to do this in time $O(m)$. Then we reduce the time to the desired bound $O(n)$.

The search gives a partitioned base $I_1 \cup I_2$ of M (ignore the copy $\overline{uv}$). Use it to find a base B' for M', following Theorem 5.5. Note that the extremal pair of Theorem 5.5 is found in time $O(n)$, from which B' is constructed in time $O(m)$. Then find the top clump L' of B' using the relation

$$L' = B' - C. \tag{3}$$

This holds since $L' = B' \cap T'$; now apply Theorem 5.6. Using (3), L' is found in time $O(m)$. Note that L' is found as a partitioned independent set. The algorithm outputs L'. To see this is correct note that if the original search succeeded, then $uv \notin B'$. Thus $uv \in C'$, the game is cut, $uv \in sp(L', M')$ and L' can be output. If the original search failed, then the game is neutral. Then since uv is in some clump of M' but not C', it is a bridge and in every top clump of M'. In particular L' can be output.

Now we reduce the time to $O(n)$. L' has $O(n)$ elements, since (3) shows each element of L' is a bridge of M, and M has at most $2n - 2$ bridges. We need only avoid explicitly constructing the base B' (which can have $\Theta(m)$ elements). Note from (3) and Theorem 5.5 that $L' = (S - B_1 - C) \cup (B_1 - B_2 - C)$. The first set can be constructed by examining $S - C$ rather than all of C. This allows L' to be easily found in time $O(n)$. □

Now we reduce the "natural" case for queries to the nonplayable case. Consider a sequence of games G, u, v where G is fixed and uv is arbitrary (and playable if

it is in G). This game is equivalent to the game $G + uv$, u, v with uv nonplayable. (If uv is an edge of G, the graph $G + uv$ corresponds to adding a second copy of uv. In the following discussion uv refers to this second copy.) We shall discuss tree queries for this case (classification queries use just a subpart of the algorithm described). In the rest of this section let

$$M_{uv} = \mathscr{G}_{uv} \vee \mathscr{G}_{uv}, \qquad M'_{uv} = \mathscr{G}^*_{uv} \vee \mathscr{G}^*_{uv},$$

where $\mathscr{G}_{uv}$ is the graphic matroid of $G + uv$.

The preprocessing consists of finding the base B and circuit elements C for matroid M.

A classification query for uv is processed as follows. Attempt to add uv to B using *cyclic scanning*. Observe that the search fails if and only if the game is join. (If the search succeeds the game is not join since $B + uv$ is a base of M_{uv}, and clearly uv is a bridge of M_{uv}, not a circuit element.) If the search fails output the clump of M_{uv} that was marked by *cyclic scanning*. Otherwise attempt to add a second copy $\overline{uv}$ to $B + uv$ using *cyclic scanning*. Observe that this second search fails if and only if the game is neutral. (If the search succeeds the game is cut because $B + uv$ is a base of M_{uv} and uv is not in its top clump, i.e., $uv \in C'$ by Theorem 5.6. Note also that if uv is originally in G this search must fail, i.e., the game cannot be cut.) If the second search fails output the clump of M_{uv} that was marked by *cyclic scanning*. This answers the classification query for a neutral game for join.

Now for cut games and for the cut strategy for neutral games, we follow the lemma to output an appropriate clump of M'_{uv}: Use the partitioned independent set (from after the second search) to find a base B' of M'_{uv}. (Note that B' contains the nonplayable edge uv if and only if the game is neutral. Also note that if uv was originally in G, the game is neutral and B' also contains the original edge uv.) Mark the top clump L' of B' in $O(n)$ time following the procedure of the lemma. (To do this observe that the set of circuit elements in M_{uv} is C, the set computed in preprocessing. The reason is that uv is not a circuit element in M_{uv}, since the game is not join.) Note that $uv \in L'$ if and only if the game is neutral. Hence L' can be returned.

THEOREM 7.1. *Let G be a fixed graph, and let u, v range over arbitrary pairs of vertices. Then a sequence of classification or tree queries G, u, v can be answered in $O(n\sqrt{m'})$ preprocessing time, $O(m)$ space, and $O(n)$ query time.*

Now we give an algorithm with faster query time. We shall see that part of the structure of the Shannon switching game is determined by the rigidity matroid $\mathscr{R}$, more specifically by the partition of a base A of $\mathscr{R}$ into maximal preclumps. Note that the partition of a base A of $\mathscr{R}$ into maximal preclumps is well defined, since the union of preclumps of A that share an edge is a preclump.

The following algorithm achieves $O(1)$ time per classification query. Assume the query u, v is "natural," i.e., no nonplayable edges (otherwise Lemma 7.1 applies).

Preprocessing Step. Find a maximum cardinality generic-independent set A for G. Find all maximal preclumps P in A. Enlarge A to a base B for M. Find the top clump L of B in M.

Query Step. If $uv \in sp(L, \mathcal{G})$ the game is join. Otherwise if $uv \in sp(P, \mathcal{G})$ for some maximal preclump P of A the game is neutral. Otherwise the game is cut.

The correctness of the algorithm is proved as follows. If $uv \in sp(L, \mathcal{G})$, then uv is in a circuit of M_{uv}. Hence $G + uv$, u, v, where uv is not playable, is a join game, and the algorithm is correct. If $uv \notin sp(L, \mathcal{G})$ but $uv \in sp(P, \mathcal{G})$ for a maximal preclump P of A, then $B + uv$ is independent in M_{uv} and $P + uv$ is a clump contained in $B + uv$. Thus uv is in the top clump of $B + uv$. Hence $G + uv$, u, v, where uv is not playable, is a neutral game, and the algorithm is correct. If $uv \notin sp(Q, \mathcal{G})$ for $Q = L$ or any maximal preclump of A, then uv is not in the span of any top clump in M_{uv}. Hence $G + uv$, u, v, where uv is not playable, is a cut game. Correctness follows.

Consider next the implementation of the Preprocessing Step, starting with finding all maximal preclumps in A. For an edge $e \in A$, let $\bar{e}$ be a new copy of e. Let L be the top clump of $A + \bar{e}$. Then the maximal preclump containing e is $L - \bar{e}$. (This follows from the definition of top clump and the fact that $\{e, \bar{e}\}$ forms a clump. As an example, if the top clump is $\{e, \bar{e}\}$, then $\{e\}$ is a maximal preclump.) Hence to find the maximal preclump containing e, run *cyclic scanning* to add $\bar{e}$ to A; then use Theorem 5.4 to find the top clump of $A + \bar{e}$. The time for n top clump computations is $O(n^2 \log n)$. This dominates, giving total time $O(n^2 \log n)$ to find the maximal preclumps.

The rest of the Preprocessing Step can be done in time $O(n^2)$ as follows. By Theorem 6.2, A can be computed in time $O(n^2)$. Similarly, B and L are found in time $O(n^2)$. The last part of this step is to preprocess the top clump and preclumps for the Query Step. For the top clump this amounts to marking its connected components. This uses time $O(n)$. For the preclumps we use the data structure based on the paritition of edges by preclumps, described for the algorithm of Theorem 6.2. This uses time $O(n^2)$. Note that there is no need to use the set-merging algorithm in this data structure, since the preclumps do not change. Thus the *find* operation in the Query Step uses $O(1)$ time.

THEOREM 7.2. *Let G be a fixed graph and let u, v be arbitrary. Then a sequence of classification queries G, u, v can be answered in $O(n^2 \log n)$ preprocessing time, $O(m)$ space, and $O(1)$ time per query.*

References

[A] M. Aigner, *Combinatorial Theory*, Springer-Verlag, New York, 1979.

[B] J. Bruno, Matroids, Graphs and Resistance Networks, Ph.D. Dissertation, Dept. Electrical Engineering, City College of New York, New York, 1967.

[BW1] J. Bruno and L. Weinberg, A constructive graph-theoretic solution of the Shannon switching game, *IEEE Trans. Circuit Theory*, **17**(1), 1970, 74–81.

[BW2] J. Bruno and L. Weinberg, The principal minors of a matroid, *Linear Algebra Appl.*, **4**, 1971, 17–54.

[CN] N. Chiba and T. Nishizeki, Arboricity and subgraph listing algorithms, *SIAM J. Comput.*, **14**(1), 1985, 210–223.

[D] E. A. Dinic, Algorithm for solution of a problem of maximum flow in a network with power estimation, *Soviet Math. Dokl.*, **11**(5), 1970, 1277–1280.

[E1] J. Edmonds, Minimum partition of a matroid into independent subsets, *J. Res. Nat. Bur. Standards*, **69B**, 1965, 67–72.

[E2] J. Edmonds, Lehman's switching game and a theorem of Tutte and Nash-Williams, *J. Res. Nat. Bur. Standards*, **69B**, 1965, 73–77.

[ET] S. Even and R. E. Tarjan, Network flow and testing graph connectivity, *SIAM J. Comput.*, **4**(4), 1975, 507–518.

[F] G. N. Frederickson, Data structures for on-line updating of minimum spanning trees, with applications, *SIAM J. Comput.*, **14**(4), 1985, 781–798.

[GS] H. N. Gabow and M. Stallmann, Efficient algorithms for graphic matroid intersection and parity, *Automata, Languages and Programming: 12th Colloquium*, Lecture Notes in Computer Science, Vol. 194, W. Brauer, ed., Springer-Verlag, Berlin, 1985, pp. 210–220.

[GT1] H. N. Gabow and R. E. Tarjan, A linear-time algorithm for a special case of disjoint set union, *J. Comput. System Sci.*, **30**(2), 1985, 209–221.

[GT2] H. N. Gabow and R. E. Tarjan, A linear-time algorithm for finding a minimum spanning pseudoforest, *Inform. Process. Lett.*, **27**(5), 1988, 259–263.

[GT3] H. N. Gabow and R. E. Tarjan, Algorithms for two bottleneck optimization problems, *J. Algorithms*, **9**(3), 1988, 411–417.

[GT4] H. N. Gabow and R. E. Tarjan, Almost-optimum speed-ups of algorithms for bipartite matching and related problems, *Proc. 20th Annual ACM Symp. on Theory of Computing*, 1988, pp. 514–527.

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

[G] D. Gusfield, Connectivity and edge-disjoint spanning trees, *Inform. Process. Lett.*, **16**, 1983, 87–89.

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

[I] H. Imai, Network-flow algorithms for lower-truncated transversal polymatroids, *J. Oper. Res. Soc. Japan*, **26**(3), 1983, 186–210.

[IF] M. Iri and S. Fujishige, Use of matroid theory in operations research, circuits and systems theory, *Internat. J. Systems Sci.*, **12**(1), 1981, 27–54.

[IR] A. Itai and M. Rodeh, The multi-tree approach to reliability in distributed networks, *Inform. and Control*, **79**, 1988. 43–59.

[KK] G. Kishi and Y. Kajitani, Maximally distant trees and principal partition of a linear graph, *IEEE Trans. Circuit Theory*, **16**(3), 1969, 323–330.

[Lam] G. Laman, On graphs and rigidity of plane skeletal structures, *J. Engrg. Math.*, **4**(4), 1970, 331–340.

[Law] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rhinehart, and Winston, New York, 1976.

[Le] A. Lehman, A solution to the Shannon switching game, *J. Soc. Indust. Appl. Math.*, **12**, 1964, 687–725.

[LY] L. Lovász and Y. Yemini, On generic rigidity in the plane, *SIAM J. Algebraic Discrete Methods*, **3**, 1982, 91–98.

[M] L. R. Matthews, Bicircular matroids, *Quart. J. Math. Oxford*, **28**(2), 1977, 213–228.

[N] C. St. J. A. Nash-Williams, Edge-disjoint spanning trees of finite graphs, *J. London Math. Soc.*, **36**, 1961, 445–450.

[OIW] T. Ohtsuki, Y. Ishizaki, and H. Watanabe, Topological degrees of freedom and mixed analysis of electrical networks, *IEEE Trans. Circuit Theory*, **17**(4), 1970, 491–499.

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

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

[RT] J. Roskind and R. E. Tarjan, A note on finding minimum-cost edge-disjoint spanning trees, *Math. Oper. Res.*, **10**(4), 1985, 701–708.

[S] B. Servatuis, Birigidity in the plane, *SIAM J. Discrete Math.*, **2**(4), 1989, 582–589.

[ST] D. D. Sleator and R. E. Tarjan, A data structure for dynamic trees, *J. Comput. System Sci.*, **26**, 1983, 362–391.

[Tar] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM Monograph, Philadelphia, PA, 1983.

[Tay] T.-S. Tay, Rigidity of multi-graphs. I. Linking rigid bodies in n-space, *J. Combin. Theory Ser. B*, **36**, 1984, 95–112.

[Tu] W. T. Tutte, On the problem of decomposing a graph into connected factors, *J. London Math. Soc.*, **36**, 1961, 221–230.

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

[Wes] H. H. Westermann, Efficient Algorithms for Matroid Sums, Ph.D. Dissertation, Dept. Computer Science, University of Colorado at Boulder, Boulder, CO, 1987.

[WW] N. White and W. Whiteley, The algebraic geometry of motions of bar-and-body frameworks, *SIAM J. Algebraic Discrete Methods*, **8**(1), 1987, 1–32.

[Wh] W. Whiteley, The union of matroids and the rigidity of frameworks, *SIAM J. Discrete Math.*, **1**(2), 1988, 237–255.