The Complexity of Multiway Cuts
(Extended Abstract)

E. Dahlhaus[1], D. S. Johnson[2], C. H. Papadimitriou[3]
P. D. Seymour[4], and M. Yannakakis[2]

Abstract. In the Multiway Cut problem we are given an edge-weighted graph and a subset of the vertices called terminals, and asked for a minimum weight set of edges that separates each terminal from all the others. When the number k of terminals is two, this is simply the min-cut, max-flow problem, and can be solved in polynomial time. We show that the problem becomes NP-hard as soon as $k = 3$, but can be solved in polynomial time for planar graphs for any fixed k. The planar problem is NP-hard, however, if k is not fixed. We also describe a simple approximation algorithm for arbitrary graphs that is guaranteed to come within a factor of $2-2/k$ of the optimal cut weight.

1. Introduction

The *Multiway Cut* problem can be defined as follows: Given a graph $G = (V,E)$, a set $S = \{s_1,s_2,...,s_k\}$ of k specified vertices or *terminals*, and a positive weight $w(e)$ for each edge $e \in E$, find a minimum weight set of edges $E' \subseteq E$ such that the removal of E' from E disconnects each terminal from all the others.

When $k = 2$ this problem reduces to the famous "min-cut/max-flow" problem, a problem of central significance in the field of combinatorial optimization due to its many applications and the fact that it can be solved in polynomial time (e.g., see [6,14,15,17]). The "k-way cut" problem for $k > 2$ has been a subject of discussion in the combinatorics community for years (closely-related variants were proposed as early as 1969 by T. C. Hu [14,p.150]). A variety of applications have been suggested, most having to do with the minimization of communication costs in parallel computing systems. In [20], Stone points out how the problem of assigning program modules to processors can be formulated in this framework. Other applications involve partitioning files among the nodes of a network, assigning users to base computers in a multicomputer environment, and partitioning the elements of a circuit into the subcircuits that will go on different chips. It is known that such problems can become NP-hard even for $k = 2$ if there is a constraint imposed on the *size* of the components into which the graph is cut [8,9]. In this paper we ask whether the problem might be tractable without such a constraint (as it is for $k = 2$).

Our first results concern the planar case. The restriction to planar graphs, besides its basic graph-theoretic significance, has potential relevance in the circuit partitioning application.

Theorem 1. *(a) For $k = 3$, the planar Multiway Cut problem can be solved in time $O(n^3 \log n)$.*

(b) For any fixed $k \geq 3$, the planar Multiway Cut problem is solvable in polynomial time.

The algorithms of Part (b) are, unfortunately, exponential in k. (Specifically, they are $O((4k)^k n^{2k-1} \log n)$.) That such exponential behavior is likely to be unavoidable follows from the next result.

Theorem 2. *If k is not fixed, the Multiway Cut problem for planar graphs is NP-hard even if all edge weights are equal to 1.*

For the Multiway Cut problem in arbitrary graphs, NP-hardness sets in much earlier.

Theorem 3. *The Multiway Cut problem for arbitrary graphs is NP-hard for all fixed $k \geq 3$ and all edge weights equal to 1.*

This theorem is proved using a "gadget" that has interesting properties on its own (as a counterexample to a conjecture about the possible submodularity of 3-Way Cut). The theorem's negative consequences are partially miti-

[1] University of Bonn. Current Address: Department of Computer Science, University of Sidney, New South Wales, Australia
[2] AT&T Bell Laboratories, Murray Hill, NJ 07974
[3] Department of Computer Science and Engineering, University of California at San Diego
[4] Bellcore, Morristown, NJ 07960

24th ANNUAL ACM STOC - 5/92/VICTORIA, B.C., CANADA
© 1992 ACM 0-89791-512-7/92/0004/0241...$1.50

gated by technical lemmas that may yield substantial reductions in the sizes of instances encountered in practice.

Finally, we have the following two approximation results, one positive and one negative.

Theorem 4. *There is a polynomial-time approximation algorithm for the Multiway Cut problem that for arbitrary graphs and arbitrary k is guaranteed to find cuts that are within $2(k-1)/k$ of optimal.*

Theorem 5. *For any fixed $k \geq 3$, the k-Way Cut problem is MAX SNP-hard (and hence unlikely to have a polynomial time approximation scheme).*

The results presented here can be contrasted to those of [11,13,19], which concern what might be called the Multiway *Split* problem. In this problem we are given G, k, and w as above (but not S), and are asked merely for a minimum weight set of edges E' whose removal separates the graph into at least k nonempty connected components. It is reported in [11] that although this problem is NP-hard for arbitrary k, it is solvable in polynomial time for each fixed $k > 2$, even for arbitrary graphs. The running time is $O(n^{k^2/2-k+11/2})$. For $k = 3$ and unweighted planar graphs, a faster $O(n^2)$ algorithm is presented in [13]. Thus the Multiway Split problem is significantly easier than the problem we study here, although we note that for fixed $k \geq 6$, our planar Multiway Cut algorithm will provide a better method for solving the planar Multiway Split problem than will the general Multiway Split algorithms of [11]: Simply run our algorithm for all possible sets S of k vertices and take the least-weight solution found. A factor proportional to n^k is added to our running time, but the resulting time bound is still $O(n^{3k-1}\log n)$. Reference [19] concerns approximation results for the Multiway Split problem, showing that the bounds we obtain in Theorem 4 for Multiway Cut can be obtained for Multiway Split without having to apply our result to all possible sets of k vertices.

To avoid bibliographic confusion, we should mention that, with the exception of Theorem 5, the results in the current paper were first announced in 1983 in an unpublished but widely circulated extended abstract of the same title [3]. The abstract has since been widely cited, both in the above-mentioned work on Multiway Split, and in follow-up work on the Multiway Cut problem itself: In [1], Chopra and Rao observe, as we failed to do in our original abstract, that for trees and 2-trees, the general k-Way Cut problem can be solved in linear time by a straightforward dynamic programming algorithm. (This can be generalized to graphs of bounded tree-width for any fixed bound, by standard techniques.) The facets of the Multiway Cut polyhedron are studied in [1,2]. An interesting generalization of the Multiway Cut problem, about which we shall have more to say in our concluding section, is studied in [4,5].

The paper is organized as follows. In Section 2 we cover the positive results for the planar case (Theorem 1a and 1b). Section 3 elaborates on the negative result for the planar case (Theorem 2). Section 4 covers our results for general graphs (Theorems 3, 4, and 5 and associated technical lemmas). A concluding Section 5 discusses additional variants and generalizations of Multiway Cut to which our techniques can apply, and points out some of the remaining open problems in the area. Because of space limitations, we have omitted most of the proofs in this extended abstract. A full version of this paper containing all the proofs is available from the authors.

2. Algorithms for The Planar Case

Our main result for planar graphs (Theorem 1) says that for all fixed k, the Multiway Cut problem is solvable in polynomial time. This is in contrast to Theorem 3, which says that for arbitrary graphs, the problem is NP-hard for any fixed $k \geq 3$. The key advantage we gain from planarity lies in the existence of a planar dual to our given graph G. We will assume without loss of generality that our graph $G = (V,E)$ is connected and that we have fixed an embedding of it on the plane. We will use a superscript D to denote a dual object. Thus G^D is the dual graph of G. If F is a subset of the edges of G, F^D is the corresponding set of edges of G^D. (Note: F^D is *not* the dual of the subgraph (V,F) of G.)

We start with Theorem 1a and the case of $k = 3$, and then show how our proof techniques can be generalized to cover the case of general fixed k (Theorem 1b).

2.1. Planar 3-Way Cuts

Figure 1 shows a graph G with a 3-way cut C, together with the duals G^D and C^D of each. The thicker edges in the figure are those of C and C_D, respectively. Note that the edges of C^D partition the geometric embedding of G^D into three regions. (In the case of Figure 1, two of these regions are single faces of G^D, but the other is the union of several faces.) Let us say that a vertex of G is *in* a given region if the face of G^D to which the vertex corresponds is part of that region. Then observe that in the figure, each of the terminals of G is in a separate region of C^D. This is clearly a general property: C is a 3-way cut of a graph G if and only if the terminals s_1, s_2, s_3 are in different regions of C^D. Thus if D is an optimal 3-way cut, C^D has exactly three regions, each one containing a distinct terminal. Furthermore, removing any edge from C^D must merge two regions, as otherwise the corresponding edge of C is not needed in the cut.

For a general instance of the 3-way cut problem, there are two topologically distinct possibilities for an optimal cut C^D.

Cut Type I. C^D consists of two edge-disjoint cycles. See Figure 2a,b. Note that the cycles may have one vertex in common and/or one cycle may lie inside the other, as in Figure 2b. They cannot have more than one vertex in com-

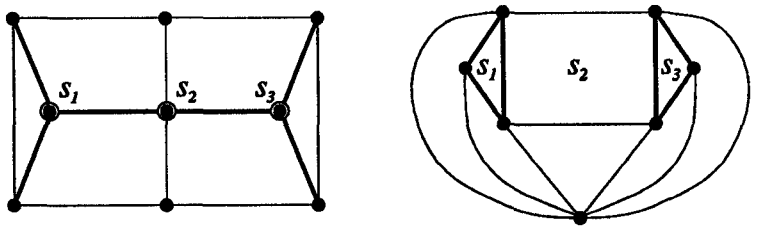


FIGURE 1. A planar 3-way cut (a) and its dual (b).

mon, however, as this would imply that C^D had more than three regions.

Cut Type II. Each pair of regions of C^D shares an edge. See Figure 2c.

For Type I cuts, the cut C consists of two edge-disjoint cuts (corresponding to the two cycles of C^D), each isolating one of the three terminals from the other two. The notion of such an "isolating cut" will also be of use in treating the case of Type II cuts. We define it formally (and generalize it to arbitrary k) as follows.

Definition. *For a given terminal s_i, an* isolating cut *for s_i is any set of edges that cuts all paths between s_i and all the other terminals.*

Note that a minimum weight isolating cut for s_i can be constructed by merging all the terminals other than s_i into a special vertex s_0, and then finding a minimum $s_i - s_0$ cut in the resulting graph by a standard 2-terminal minimum cut algorithm.

We shall now describe how to find an optimal 3-way cut. We provide procedures that work for each type of cut. Each procedure either returns the best cut of the corresponding type, or else reports (correctly) that any optimal cut is of the other type.

Our procedure for Type I cuts is straightforward. We simply compute the three minimum weight isolating cuts for s_1, s_2, and s_3 respectively. Note that a minimum weight Type I cut must have weight at least as large as the sum of the weights of the two smallest of these three isolating cuts. If the two smallest are edge-disjoint, then their union is optimal among all 3-way cuts of Type I. If the two smallest are not edge-disjoint, then their union is a 3-way cut that has strictly smaller weight (since all edge weights are by assumption positive). Consequently, the best 3-way cut is not of Type I.

Our procedure for Type II cuts is significantly more complicated. Suppose we have an optimal 3-way cut that is of Type II. Look again at Figure 2c. The cycle that bounds each region corresponds to an isolating cut for the terminal contained in that region, but these isolating cuts are not necessarily optimal, as they overlap. Consider the two vertices that are of degree 3 in C^D and are labeled a and b in the figure. The following lemma allows us to fix one of the three paths connecting a and b in G^D.

Lemma 2.1. *Suppose that the dual of an optimal 3-way cut C is of Type II and a and b are the two vertices of degree 3 in C^D. Let P be any shortest path from a to b in G^D. Then there is an optimal 3-way cut C_0 that is of Type II, has a and b as its two vertices of degree 3 in C_0^D, and such that P is one of the three paths that join a to b in C_0^D.*

In light of Lemma 2.1, our procedure for Type II cuts can work by repeatedly calling a subroutine, once for each potential pair a,b of degree-3 vertices in C_D. The subroutine either constructs a minimum weight 3-way cut C which is of Type II and has a and b as the two degree-3 vertices in C^D, or reports (correctly) that no minimum weight 3-way cut has that form. The subroutine proceeds as follows:

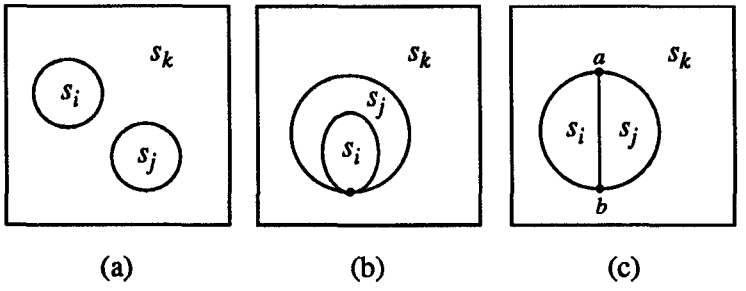


FIGURE 2. Types of 3-way cuts: Type I (a) and (b), Type II (c).

First, construct a shortest path P between a and b in G_D. By Lemma 2.1 we may assume that P is contained in C^D. Delete the edges in G corresponding to the edges of P, obtaining a new graph H. In the embedding of this new graph induced by our original embedding of G, all the regions of G corresponding to vertices on P in G^D are merged into a single region. This corresponds in H^D to coalescing all the vertices along the path P from a to b into a single vertex v_P. This coalescence turns C^D from a Type II cut into a Type I cut like the one in Figure 2b in which the two edge-disjoint cycles share a common vertex, in this case v_P. (The two cycles need not however be nested as they are in the figure; they can have disjoint interiors.)

We can now apply our previously described procedure for Type I cuts to H, obtaining a Type 1 cut C_H, or a report that the best 3-way cut for H is *not* of Type I. In the latter case, an optimal 3-way cut for G could not have been of Type II with the pair a,b as its degree-3 vertices, and we report this fact. In the former case, the cut C_H will, when augmented with the edges of G corresponding to the edges of P in G^D, be a 3-way cut for G. It cannot be an optimal cut, however, unless C_H^D has the desired form of two edge-disjoint cycles with v_P as a single common vertex. Otherwise the edges corresponding to P can be deleted and a valid (and lighter) 3-way cut for G will remain. Thus if C_H^D does not have the desired form, we once again report that no optimal 3-way cut for G is of Type II with a,b as its two degree-3 vertices.

Our overall algorithm for finding an optimal 3-way cut can thus proceed as follows:

Procedure 3-Way

1. Perform the Type I procedure on G.
   If a valid Type I cut is found, put it on the list of potential optima.
2. Construct the dual graph G^D and perform an all-pairs shortest path computation for G^D.
   For each pair a,b of vertices in G^D, do the following:
   - 2.1. Let P be the shortest path in G_D between a and b as constructed in step 2, and let H be the graph obtained from G by deleting the edges corresponding to edges of P.
   - 2.2. Perform the Type I procedure on H.
   - 2.3 Let v_P be the coalesced vertex in H^D corresponding to the path P. If a valid Type I cut C_H for H is found and has a dual consisting of two edge-disjoint cycles having v_P as their unique common vertex, do the following:
     - 2.3.1 Let C_G be the 3-way cut for G consisting of C_H together with the edges of G corresponding to the edges of P in G^D.
     - 2.3.2 Add C_G to the list of potential optima.
3. Output the lightest 3-way cut on the list of potential optima.

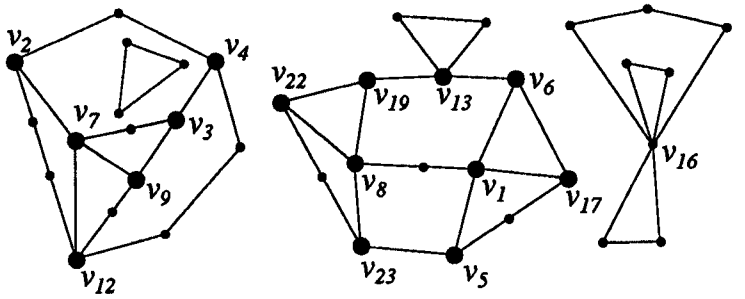
Theorem 1a. *Given a planar graph G with specified terminals s_1, s_2, and s_3, Procedure 3-Way outputs an optimal 3-way cut, and can be implemented to run in time $O(n^3 \log n)$, where n is the number of vertices in G.*

2.2. Planar Multiway Cuts

In this section we turn to the case of k-way cuts where $k > 3$. The algorithm we present will work for all $k \geq 3$ and will have a running time that, although exponential in k, is polynomial whenever k is fixed. It can be viewed as a (major) generalization of the algorithm of the previous section for the $k = 3$ case. For our discussion here, it will be convenient to assume that no two subsets of edges has the same total weight. (We can make sure that the assumption is satisfied in various ways. For instance, if Δ is the weight of the lightest edge and the edges are ordered $e_1, e_2, \ldots, e_m$, we could use the revised edge weights $w'(e_i) = w(e_i) + \Delta/2^i$.) The key consequence of the assumption is that optimal cuts, shortest paths, etc. are unique, so that we can refer to *the* optimal cut etc. A less desirable consequence is that the cost of doing additions and comparisons of edge weights may go up by a factor of n, given the large number of bits needed to represent them, but given that our main goal here is to show that running times are $O(n^{ck})$ for some c, a factor of n will not make a significant difference.

In the $k = 3$ case, we observed that the dual C^D of the optimal cut was a subgraph of G^D that partitioned the embedding of G^D into three regions, each containing a distinct terminal. We then reduced the problem to the computation of 2-way cuts and shortest paths by first guessing (i.e., trying all possibilities for) some information about C^D. In particular, we guessed the *topology* (whether the cut consisted of two edge-disjoint cycles or not) and (in the latter case) the identity of the two degree-3 nodes a and b in C^D.

For general $k \geq 3$, we follow the same approach. Assume as before that we have previously decided on some fixed embedding of G. Suppose C is the optimal k-way cut, and once again let C^D be the dual of C viewed as a subgraph of a predetermined planar embedding of G^D. Then C^D must partition the embedding of G^D into precisely k regions, each containing a distinct terminal. Our notion of a *topology* for C^D is derived as follows: Consider the connected components of C^D, and call such a component *complex* if it contains more than one vertex that has degree three or more in C^D. Let $C_1^D, C_2^D, \ldots, C_q^D$ be an enumeration of the complex components of C^D, and for each i, $1 \leq i \leq q$, let N_i be the set of vertices with degree three or more in C_i^D. (Note that we must have $q \leq k-1$, since every connected component of C^D must enclose at least one terminal, and the infinite region of C^D contains one ter-



$$N_1 = \{v_2, v_3, v_4, v_7, v_9, v_{12}\} \quad N_2 = \{v_1, v_5, v_6, v_8, v_{13}, v_{17}, v_{19}, v_{22}, v_{23}\}$$

FIGURE 3. The dual C^D of an optimal cut C and its topology $\{N_1, N_2\}$.

minal.) Let $N = \cup_{i=1}^{q} N_i$. The *topology* of C^D is simply the (unordered) partition of N given by the sets $N_1, N_2, \ldots, N_q$. See Figure 3 for an example of a C_D and its topology, consisting of two sets N_1 and N_2. In the figure the vertices of degree three or greater are highlighted. Note that the degree-6 vertex v_{16} does not participate in the topology because its connected component contains only one vertex of degree three or greater.

Lemma 2.2. *If G is a connected planar graph with n vertices, let C be an optimal k-way cut for G, and let C^D be the planar dual of C, viewed as a subgraph of G^D. Then the number of distinct possibilities for the topology of C^D is $O((2n)^{2k-4})$.*

Our algorithm for the general k-way cut problem will work by considering in turn each of the possibilities for the topology of the optimal cut. For each we will invoke a subroutine analogous to those used in our 3-way cut algorithm. The subroutine will either output the shortest cut that has the given topology, or else report (correctly) that the optimal cut cannot have the given topology. In order to specify the subroutine, we will need to know some structural results about the topology of the optimum cut.

So suppose we are given a topology $N_1, N_2, \ldots, N_q$. If this is the optimal topology, then C^D contains connected components $C_1^D, C_2^D, \ldots, C_{q'}^D$, $q' \geq q$, where for $1 \leq i \leq q$, N_i is the set of vertices with degree three or more in C_i^D, and for $q < i \leq q'$, C_i^D contains at most one vertex with degree three or more. The analogue of Lemma 2.1 for this general k case is that for each C_i^D, $i \leq q$, we can efficiently identify a subtree T_i^D of C_i^D that spans all the vertices of N_i. This was the case in Lemma 2.1, where for $N_1 = \{a,b\}$ (Figure 2c), we identified a path between the two degree-3 vertices a and b by doing a shortest path computation. For the general case, we shall need both shortest paths and minimum spanning trees.

For a given topology, the trees T_i^D, $i \leq q$, are constructed as follows. We treat the sets N_i, $i \leq q$, in turn. (Order is not important.) Given N_i, we construct an auxiliary weighted complete graph H_i with N_i as its vertex set and with the weight of the edge linking u and v being the length of the shortest path in G^D between u and v. Compute the minimum spanning tree $T[H_i]$ of H_i, and let T_i^D be the subgraph of G^D formed by replacing each edge in $T[H_i]$ by the corresponding shortest path in G^D. We shall call T_i^D the *minimum spanning tree* of N_i (although note that it may not even be a tree if C^D does not have the given topology). Figure 4 portrays the C^D of Figure 3 with the trees T_i^D (chosen according to the correct topology) highlighted. (For future reference, the figure also indicates which terminal is contained in which region of C^D.) The next lemma establishes the key properties satisfied by T_i^D when C^D has the given topology.

Lemma 2.3. *Let C be the optimal k-way cut and C^D be its planar dual, viewed as a subgraph of G^D. Let C_i^D,*

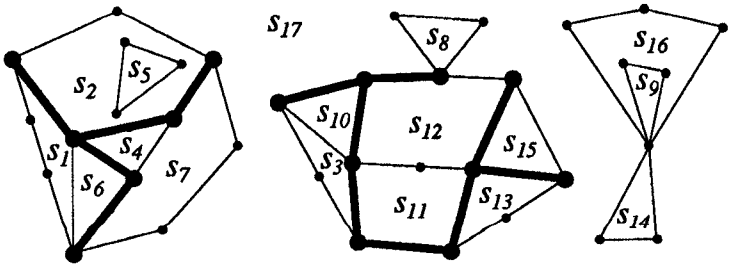


FIGURE 4. The dual C^D with the trees T_i^D highlighted and the locations of terminals indicated.

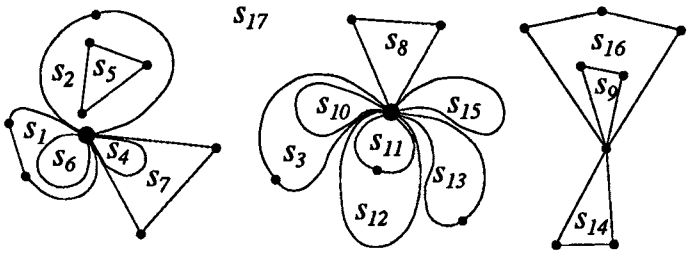


FIGURE 5. The graph $C[0]^D$ obtained by coalescing the spanning trees T_i^D in C^D.

$1 \le i \le q$, be the complex connected components of C^D, and let N_i be the set of vertices with degree three or greater in C_i^D, $1 \le i \le q$. Then (a) each C_i^D, $1 \le i \le q$, contains the minimum spanning tree T_i^D for N_i, and (b) no two of the paths in T_i^D corresponding to edges of $T[H_i]$ intersect except at a common endpoint (and hence T_i^D is indeed a tree).

Lemma 2.3 treats the connected components C_i^D of C^D in isolation. Let us now look at how they interact. First note that the trees T_i^D are all vertex disjoint, since each is a subgraph of a different connected component of C^D. Thus they constitute a subforest of C^D. Let T^D represent this subforest, the union of all the edges in the T_i^D, and let T be the subset of edges in G whose planar dual is T^D. By Lemma 2.3, the optimal cut C for G will consist of T plus some additional edges, assuming we chose the correct topology for C. To find those additional edges, we delete T from G to obtain a new graph that we shall call $G[0]$. Assuming we have the correct topology, the optimal cut for $G[0]$ will be $C[0] = C - T$.

Let $G[0]^D$ be the embedded dual of $G[0]$ obtained by coalescing all the edges of each T_i^D in G^D to a single point. Note that $G[0]^D$ remains connected, and has lost none of the regions from G^D since only trees were coalesced. Assuming that we have the optimal topology, $C[0]^D$ will be a collection of $k-1$ simple cycles in $G[0]^D$, each of which is a biconnected component of $C[0]^D$. That is, a connected component of $C[0]^D$ may consist of more than one cycle, but all such cycles must share a single common vertex. (In particular, the coalesced vertices corresponding to the T_i^D will be such common vertices, although there may be others that were already present in C^D.) See Figure 5.

Note that in the embedding of $G[0]^D$, some of the cycles of $C[0]^D$ will contain others. An innermost cycle will constitute the complete boundary of a region in $C[0]^D$, whereas other regions may have boundaries made up of the edges of several cycles. We shall show that if one treats the cycles in an appropriate "inside out" order, each of these cycles can be viewed as a minimum isolating cut in an appropriately constructed graph. Let us identify each region with the terminal s_i it contains, and construct a partial order $\le$ on the terminals as follows: $s_i \le s_j$ if and only if the outermost cycle bounding s_i's region is contained in the outermost cycle bounding s_j's region. (By convention, we assume that "the outermost bounding cycle" for the infinite region contains all other cycles, so that the terminal contained in the infinite region is > all other terminals.) Note that by this definition, the outermost bounding cycle for s_i must separate s_i from all terminals $s_j > s_i$.

Let π be an ordering of the terminals that is consistent with this partial order, i.e., if $s_i \le s_j$, then $\pi(i) \le \pi(j)$. For instance, for the $C[0]^D$ depicted in Figure 5, such an ordering would be $s_6, s_1, s_4, s_7, s_5, s_2, s_{10}, s_3, s_{11}, s_{12}, s_8, s_{15}, s_{13}, s_9, s_{16}, s_{14}, s_{17}$. We define a sequence of graphs and isolating cuts as follows. For $1 \le i \le k-1$, let A_i^D be the set of edges in $G[0]^D$ making up the outer boundary cycle for the region containing terminal $s_{\pi(i)}$. Let A_i be the corresponding set of edges in $G[0]$. Note that $C[0] = \cup_{i=1}^{k-1} A_i$. Now let $G[i]$ be the graph obtained from $G[0]$ by deleting the edges of $\cup_{j=1}^{i} A_j$.

Lemma 2.4. *For $1 \le i \le k-1$, A_i is a minimum isolating cut for terminal $s_{\pi(i)}$ in graph $G[i-1]$.*

Given Lemmas 2.3 and 2.4, the following procedure will handle any given topology $N_1, N_2, \ldots, N_q$ appropriately, either outputting the minimum weight cut with that topology or reporting correctly that the optimal cut cannot have that topology. Assume that as a preprocessing step we have constructed our standard embedding of the dual graph G^D and computed all shortest paths between vertices of G^D.

Procedure CheckTopology $(N_1, N_2, \ldots, N_q)$

1. For $1 \le i \le q$, do the following.
   - 1.1 Construct the auxiliary graph H_i, the minimum spanning tree $T[H_i]$, and the subtree T_i^D.
   - 1.2. If any two paths in T_i^D corresponding to edges of $T[H_i]$ share a vertex other than a common endpoint, reject the topology.
2. If any two subtrees T_i^D share a common vertex, reject the topology.
   Otherwise, let T^D be the union of the edge sets T_i^D, let $G[0]$ be the graph obtained from G by deleting all edges in the dual set T.

3. Let $W^* = \infty$ (our initial estimate of the optimal cut weight).
For all possible permutations $s_{\pi(1)}, s_{\pi(2)}, \ldots, s_{\pi(k)}$ of the terminals, do the following.

   3.1. Set $C = T$.

   3.2. For $1 \le i \le k-1$ do the following:

      3.2.1. Find a minimum isolating cut A_i for $s_{\pi(i)}$ in $G[i-1]$.

      3.2.2. Set $C = C \cup A_i$, and let $G[i]$ be the graph obtained by deleting the edges of A_i from $G[i-1]$.

   3.3. If $w(C) < W^*$, set $W^* = w(C)$ and $C^* = C$ (the current best cut with this topology).

4. Output C^*.

It should be clear from the above discussion that this subroutine has the desired properties. Its running time is dominated by that for the minimum isolating cut computations occurring at Step 3.2.1. As discussed, each such computation can performed using a standard 2-terminal minimum cut algorithm in time $O(n^2 \log n)$, assuming a machine model in which additions, subtractions, and comparisons take constant time. Given our proposed method for imposing the restriction that all sets of edges have unique weights, however, such a model is inappropriate. Even given the standard assumption that the original instance has edge weights that fit into a single computer word, our method for insuring the subset weight uniqueness restriction gives rise to weights whose binary representations involve $\Theta(n)$ bits. With such large numbers, the time bound for the isolating cut computations grows to $O(n^3 \log n)$. We perform $k-1$ such computations for each of the $k!$ permutations of the terminals, yielding total of $O(k^k)$ for each topology.

The overall algorithm then consists of performing Procedure CheckTopology for each possible topology, of which there are $O((2n)^{2k-4})$ by Lemma 2.2, and outputting the best cut found for any non-rejected topology. Thus we have our claimed result for general fixed k, stated here in slightly more precise form than given in the introduction:

Theorem 1b. *Given a planar graph G with n vertices and k specified terminals, a minimum k-way cut can be constructed in time $O((4k)^k n^{2k-1} \log n)$.*

Note that if we specialize this result to the previously considered case of $k = 3$, the time bound is $O(n^5 \log n)$, substantially larger than the $O(n^3 \log n)$ of Theorem 1a. This is a result of two factors: (1) A factor of n because we can't in general find the needed isolating cuts using planar 2-terminal cut algorithms, as we could when $k = 3$, and so have to use general 2-terminal cut algorithms. (2) A factor of n because we needed to operate with $\Theta(n)$-bit weights in order to insure that every subset of edges had a unique weight.

It may well be that more efficient algorithms can be derived by careful algorithmic design and analysis, but the bounds we have presented adequately fulfill our goal of showing that the k-way cut problem can be solved in polynomial time for fixed k and planar graphs. Moreover, as the NP-completeness result of the next section implies, it is likely that any algorithms for the general problem will have exponential (or at least super-polynomial) running times.
/.NH Complexity of Planar Multiway Cut

Our proof of NP-completeness for the general planar multiway cut problem is via a transformation from PLANAR 3-SATISFIABILITY [16], using an elaborate component design construction (details in the full paper). The problem remains NP-complete even if all edge weights are equal to 1 and no vertex degree exceeds 11 (a degree bound that should be reducible to 6 by a more-complicated variant on our proof). If one allows allows edge weights to range over the set {1,2,3,4,5} rather than requiring them all to be equal, the degree bound can be reduced to 3. The problem is of course trivial if the maximum degree is 2.

3. Multiway Cut for Fixed k and Arbitrary Graphs

This section covers our results about the multiway cut problem on arbitrary graphs. We begin with our NP-completeness result for fixed $k \ge 3$ and then discuss ways of coping with the implied complexity of the problem. Note that it suffices to prove the problem NP-complete for $k = 3$, since the problem for higher values of k can trivially be derived from that for $k = 3$. We discovered the key "gadget" needed our NP-completeness proof while pursuing what at first seemed like a promising algorithmic approach to the 3-Way Cut problem, based on results on submodular set functions by Grötschel, Lovász, and Schrijver [12].

3.1. Cuts as Submodular Set Functions

In order to understand the results of [12], we first need some definitions. Let U be a finite set. A function f defined on the subsets of U is *submodular* if for any two subsets X and Y of U, $f(X) + f(Y) \ge f(X \cap Y) + f(X \cup Y)$. Grötschel, Lovász, and Schrijver [12] show that if a submodular set function f can be computed in polynomial time, then f can also be *minimized*, i.e., set Y with $f(Y) = \min\{f(X) : X \subseteq U\}$ can be found, in polynomial time. (The algorithm involves an appropriate application of the ellipsoid method.)

A paradigmatic example of a submodular set function involves the usual (2-way) minimum cut problem. In this case, U is the set of nonterminal vertices $V - \{s_1, s_2\}$, and $f(X)$ is the total cost of the edges which have precisely one endpoint in the set $X \cup \{s_1\}$. The submodularity of this function is easy to verify, as is the fact that

$\min\{f(X): X \subseteq U\}$ is the weight of a minimum 2-way cut. We of course already know how to minimize this function f in polynomial time without resorting to the ellipsoid method, but the formulation is suggestive. Could it be possible that 3-way cuts might also be computable as minima of a submodular set function?

It is easy to define a set function for 3-way cuts that is analogous to the one above for the 2-way case. Let $U = V - \{s_1, s_2, s_3\}$ be the set of nonterminal vertices. For any 3-way cut E', let us say that a set $X \subseteq U$ is *associated with* s_1 under E' if (1) X contains neither s_2 nor s_3 and (2) every edge of G with precisely one endpoint in $X \cup \{s_1\}$ belongs to E'. Note that if a 3-way cut E' disconnects some vertices from all three terminals, then more than one set can be associated with s_1 under E'. (Under a minimum 3-way cut, however, the set associated with s_1 is uniquely determined.) For any subset X of U, let $f(X)$ be the minimum cost of a 3-way cut under which X is associated with s_1. It is easy to see that the minimum value for this function over all $X \subseteq U$ equals the minimum weight for *any* 3-way cut. Moreover, we can use a polynomial-time algorithm for the 2-way cut problem to evaluate f in polynomial time: Given a subset X of U, find a minimum weight cut separating s_2 from s_3 in the graph obtained by deleting s_1 and all the vertices in X from G. Add to the weight of this cut the weight of all edges with precisely one endpoint in $X \cup \{s_1\}$. Therefore, if f were submodular (for all graphs), we could solve the 3-way cut problem in polynomial time.

Unfortunately, this is not the case. Consider the 9-vertex graph C depicted in Figure 6. Note that in addition to the three terminals s_1, s_2, s_3, the graph contains two specified vertices x and y. The 12 edges incident on the terminals have weight 4, as indicated in the figure. The other 6 edges, unlabeled in the figure, have weight 1. Let c^* be the cost of an optimal 3-way cut for C. For each i,j, $1 \le i,j \le 3$, let an *i,j-cut* be a 3-way cut that leaves vertex x connected to s_i and vertex y connected to s_j, and let

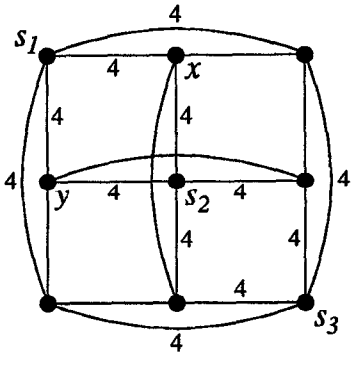


FIGURE 6. Graph C: Submodularity counterexample and NP-completeness gadget.

$c(i,j)$ be the cost of a minimum i,j-cut. The sets X and Y that cause f to violate submodularity are defined as follows:

Let X be the set of vertices connected to s_1 in an optimal 1,2 cut. (Note that by definition of i,j-cut, x is in X and y is not.) Let Y be the set of vertices connected to s_1 in an optimal 2,1 cut. (Note that y is in Y and x is not.) By definition, we have $f(X) = c(1,2)$ and $f(Y) = c(2,1)$. We also must have $f(X \cup Y) \ge c(1,1)$ and $f(X \cap Y) \ge \min\{c(2,3), c(3,2), c(2,2), c(3,3)\}$. Thus if f were submodular, we would need to have $c(1,2) + c(2,1) \ge c(1,1) + \min\{c(2,3), c(3,2), c(2,2), c(3,3)\}$. In light of the following lemma, however, this claim is false.

Lemma 4.1. *For the graph C of Figure 6, the following properties hold:*

(a) $c(1,2) = c(2,1) = c^*$,

(b) $c(i,j) \ge c^* + 1$ *for all other pairs i,j, and*

(c) $c(1,1) = c(2,2) = c^* + 1$.

3.2. The NP-Completeness of 3-Way Cut

As mentioned above, the graph C does more than simply rule out a prominsing algorithmic approach. It is the key gadget in our NP-completeness for 3-WAY CUT, a local replacement transformation from the SIMPLE MAX CUT problem [8,9]. One replaces each edge of the original graph with a copy of C, identifying the vertices x and y with the edge's endpoints. Details are in the full paper.

The graph constructed in our proof does not have bounded vertex degrees. This is unavoidable, so long as we assume all edge weights are equal. If k is fixed, all edge weights are equal, and there is a bound d on vertex degree, then Multiway Cut can be solved in polynomial time! Observe that in this case the weight of a cut is simply the number of edges it contains, and an optimal cut can contain no more than kd edges (since the cut that simply breaks all the edges incident on each terminal is no bigger than this). But since k and d are fixed, kd is a constant independent of n. Consequently, we can use exhaustive search and still take time that is polynomially bounded in n.

If we remove the restriction to equal-weight edges, however, the fixed-k problem becomes NP-complete even if all vertex degrees are three or less.

3.3. Reducing the Instance Size

The results of Sections 4.1 and 4.2 effectively dash any hope of finding optimal k-way cuts, $k \ge 3$ efficiently by means of 2-way cut (i.e. max flow) algorithms. Such algorithms may still be useful, however, as we shall see in this and the next section. Recall that an *isolating cut* for a terminal s_i is a set of edges that separates s_i from the other two terminals, and that minimum weight isolating cuts can be found in $O(nm\log(n^2/m))$ time by performing max flow computations in a modified graph. The following Lemma implies that the computation of minimum weight isolating cuts can be used to reduce the number of vertices

in an instance. Suppose $G = (V,E)$ is a connected graph with specified terminals $s_1, s_2, \ldots, s_k$.

Lemma 4.2. *Suppose $i \in \{1,2,\ldots,k\}$, $k \geq 3$. Let E_i be a minimum weight isolating cut for terminal s_i, and let V_i be the set of vertices that remain connected to s_i when the edges of E_i are removed from G. Then there exists an optimal k-way cut for G that leaves all the vertices of V_i connected to s_i.*

Using Lemma 4.2, we can reduce the number of vertices in our instance by $|V_i - 1|$. Simply construct a new graph in which all the vertices of V_i are merged into the terminal s_i. An optimal k-way cut for this shrunken graph will induce an optimal cut for the original one.

In order to get the maximum effect from applying Lemma 4.2 in this way, we should start with the minimum isolating cut for s_i such that V_i is as big as possible. Let us say that a set V_i is an *isolation set* for a terminal s_i if it is the set of vertices left connected to s_i by some minimum weight isolating cut. Let us call it an *optimum isolation set* for s_i if it has maximum cardinality over all isolation sets for s_i. From observations made in [6, pp.10-13], it can be seen that the optimum isolation set for a given terminal is unique and contains all other isolation sets for that terminal. It can be found by performing one maximum-flow computation followed by some linear-time post-processing. A corollary of the following Lemma is that k optimum isolating set computations suffice to shrink G as far as it can go.

Lemma 4.3. *Let V_i be the optimum isolation set for s_i, $1 \leq i \leq k$, and let $\overline{G}$ be the graph obtained from G by merging all the vertices of V_1 into s_1. Then in $\overline{G}$ the optimum isolation set for s_1 is $\{s_1\}$, and the optimum isolation set for s_i is $V_i - V_1$, $2 \leq i \leq k$.*

Lemma 4.3 indicates both the efficiency with which we can apply Lemma 4.2 to reduce the instance size, and the bounds on how much shrinkage can be obtained. In particular, k optimum separation set computations suffice to yield all the shrinkage one can expect: Note that the proof of Lemma 4.3 would apply just as well if we renamed the terminals in any order. So let $G_0 = G$, and inductively obtain G_i from G_{i-1} by performing an optimum isolation set computation for s_i and merging all the vertices in the set obtained into the terminal s_i. Lemma 4.2 says that an optimal k-way cut in G_i induces one in G_{i-1} (and, by induction, in G), and Lemma 4.3 says that the optimum isolation set for s_i in G_i is $\{s_i\}$ (and by induction, the optimum isolation set for s_h, $1 \leq h \leq i$, is $\{s_h\}$). Thus in G_k, the optimum isolation set for each terminal consists of the terminal itself, but a minimum weight k-way cut still induces one in the original graph G. Thus G_k is a maximally shrunken graph that can still induce an optimal k-way cut.

3.4. Near-Optimal Multiway Cuts

If one is willing to settle for cuts that are only *near*-optimal, one can exploit a bit further our ability to construct optimum isolating cuts. Consider the following straightforward heuristic.

Isolation Heuristic

1. For $1 \leq i \leq k$ construct a minimum weight isolating cut $\hat{E}_i$ for terminal s_i.
2. Determine h such that $\hat{E}_h$ has maximum weight among all the $\hat{E}_i$'s.
3. Let $\hat{E}$ be the union of all cuts $\hat{E}_i$ except $\hat{E}_h$.
4. Return $\hat{E}$.

Note that the Isolation Heuristic clearly outputs a k-way cut. Moreover, it can be implemented to run in $O(knm\log(n^2/m))$ time by using the max flow algorithm of [10] to compute each of the k required isolating cuts. This is the heuristic to which we referred in Theorem 4 of the Introduction. A more precise statement of that theorem can now be given.

Theorem 4. *The Isolation Heuristic constructs a k-way cut whose weight is guaranteed to be no more than $2(k-1)/k$ times the optimal weight.*

Proof. We first consider the upper bound. Let $\overline{E}$ be an optimal k-way, and let $\overline{w} = w(\overline{E})$. For $1 \leq i \leq k$, let $\overline{V}_i$ be the set of vertices left connected to s_i by $\overline{E}$, and let $\overline{E}_i$ be the set of edges in $\overline{E}$ with one endpoint in $\overline{V}_i$. Note first that for each i, the set $\overline{E}_i$ is an isolating cut for s_i. Hence $w(\overline{E}_i) \geq w(\hat{E}_i)$. Thus

$$w(\hat{E}) \leq \frac{k-1}{k}\sum_{i=1}^{k} w(\hat{E}_i) \leq \frac{k-1}{k}\sum_{i=1}^{k} w(\overline{E}_i)$$

On the other hand, each edge in $\overline{E}$ is in exactly two different sets $\overline{E}_i$, and so $\Sigma_{i=1}^{k} w(\overline{E}_i) = 2\overline{w}$. Thus $w(\hat{E}) \leq \overline{w}(2(k-1)/k)$, as claimed. (Note that it is easy to show that this bound cannot be improved.) □

For $k = 3$, the ratio guaranteed by Theorem 4 is 4/3 and fairly close to 1. An interesting question is whether there are any polynomial time heuristics that provide better guarantees. In particular, is it conceivable that we could guarantee ratios arbitrarily close to 1? Formally, is there a *polynomial time approximation scheme* for 3-way cut, i.e., a sequence of polynomial time algorithms A_t, where A_t is guaranteed to find a 3-way cut of weight at most $1 + 1/t$ times the optimal weight? (The algorithms need not be polynomial in t, only in the instance size.)

There is significant evidence that the answer to this question is no. In particular, our proof of Theorem 3 implies that the 3-Way Cut problem is *MAX SNP-hard* [18]. As a consequence, it is no more likely to have a polynomial time approximation scheme than optimization versions of a variety of other problems for which no such

schemes are known, such as MAX 3-SAT, the problem of finding a truth assignment for a (not-necessarily satisfiable) instance of 3-SAT that maximizes the number of satisfied clauses. See the full paper for a more-detailed explanation of these matters.

4. Related Results and Open Problems

Our NP-completeness results in Sections 3 and 4 can be adapted to several related problems of interest. In 1969, T. C. Hu [14] raised the question of the complexity of the following problem. Suppose we are given a list of vertex pairs (u_i, v_i), $1 \le i \le k$, and are asked to find a minimum $C(u_1,u_2,...,u_k,v_1,v_2,...,v_k)$ cut, i.e., a minimum weight set of edges separating each pair of vertices u_i, v_i, $1 \le i \le k$. This is just 2-way Cut when $k = 1$. The problem is also polynomial time solvable when $k = 2$, by using two applications of a 2-way cut algorithm [21]. Our result for 3-Way Cut implies that it is NP-hard for arbitrary graphs when $k = 3$, even if all edge weights are equal: merely let the three pairs be (s_1,s_2), (s_2,s_3), and (s_3,s_1). (If one wants all the u_i and v_i to be distinct, the problem remains NP-hard, as can be proved via a simple modification to the input graph.)

Note that for any fixed k the Isolation Heuristic of Section 4.4 can be used in the design of a polynomial-time approximation algorithm for Hu's problem. For each partition P of the vertices $u_1,u_2,...,u_k,v_1,v_2,...,v_k$ into sets $S_1,\ldots,S_{|P|}$ such that no pair (u_i,v_i) is in the same set, let G_P be the graph obtained by merging all the vertices in set S_j into a single terminal vertex s_j, $1 \le j \le |P|$. Run the Isolation Heuristic on each such graph G_P, and output the best cut found. Since the optimal $C(u_1,u_2,...,u_k,v_1,v_2,...,v_k)$ cut must induce one of the partitions P, and since no partition contains more than $2k$ sets, the weight of the cut we output is at most $2(2k-1)/2k = (2k-1)/k$ times optimal by Theorem 4. The running time is $O(2^{2k}knm\log(n^2/m))$, which is polynomial for fixed k. The question remains open, however, as to whether there is a polynomial-time approximation algorithm that works for arbitrary k and provides a similar guarantee.

More recently, Erdös and Székely in [4,5] proposed the following generalization of Multiway Cut. Suppose you are given a graph $G = (V,E)$ weighted edges, and a partial k-coloring of the vertices, i.e., a subset $V' \subseteq V$ and a function $f:V' \to \{1,2,...,k\}$. Can f be extended to a total function such that the total weight of edges that have different colored endpoints is minimized? The k-Way Cut problem is the special case where $|V'| = k$ and f is 1-1, i.e., each color is initially assigned to precisely one vertex. It is easy to see that for general graphs, this problem is in fact equivalent to Multiway cut: simply merge all the vertices with the same color, call the resulting merged vertices "terminals," and find the minimum weight k-way cut for the resulting graph. For special classes of graphs, however, the "Colored Multiway Cut" problem can be more general. (The above merging trick need not for instance preserve planarity or acyclicity.) Nevertheless, the dynamic programming algorithm mentioned in the Introduction for Multiway Cut on trees extends in a natural way to the Colored Multiway Cut problem, yielding an $O(nk)$ algorithm, as Erdös and Székely observe. This in turn implies that if G is such that deleting all the terminals renders it acyclic, then Multiway Cut can itself still be solved in $O(nk)$ time. (Simply split each terminal s_i into $degree(s_i)$ separate vertices, one for each edge incident on s_i, assign color i to all the derived vertices, and apply the abovementioned algorithm for Colored Multiway Cut on trees to the resulting graph [5].)

An obvious question is whether our algorithms for planar graphs also extend to Colored Multiway Cut problem. The answer is no. Colored Multiway Cut is clearly polynomial-time solvable if $k = 2$, even for general graphs. For any fixed $k \ge 3$, however, it remains NP-complete even for planar graphs and all weights equal to 1. The $k = 4$ case follows directly from our proof of Theorem 2. For $k = 3$, the result can be proved by a transformation from PLANAR 3-COLORABILITY [8,9], using a local replacement argument in which each edge is replaced by a partially colored structure designed to make it expensive for the endpoints of the original edge to get the same color. We leave the details to the enterprising reader. (Note that this last result provides us with an alternate proof of the NP-completeness of 3-WAY CUT for general graphs: once again simply merge all vertices with the same color. The fact that such an operation may destroy planarity is in this case irrelevant.)

Returning to the original Multiway Cut problem, in our opinion the most interesting open problem is whether one can improve upon the approximation results of Theorem 4. Although polynomial-time algorithms with worst-case ratios arbitrarily close to 1 are unlikely in light of Theorem 5, can we do better than the $2(k-1)/k$ guarantee we proved for the Isolation Heuristic? Noga Alon [private communication, 1991] has observed that for the special cases of $k = 4$ and $k = 8$ improvements can be obtained using a variant of our approach. For $k = 4$, the Isolation Heuristic provides a guarantee of 3/2. An improved guarantee of 4/3 can be obtained as follows: For each partition of the terminals into sets S_1,S_2 of size two, use max flow techniques to compute the minimum cut that separates the terminals in S_1 from those in S_2. Output the union of the two best such cuts. The reader can readily verify that this union is a 4-way cut whose weight is at most 4/3 optimal. Note that this approach requires only three max flow computations versus the four needed by the Isolation Heuristic, so it is faster as well. (Cunningham reports in [2] that F. Zhang has independently obtained this $k = 4$ improvement.)

For $k = 8$, the guarantee of our theorem can be improved from 7/4 to 12/7. Here one computes minimum

2-way cuts based on partitions of the set of terminals into sets of size four. It can be shown that the average weight of these cuts is at most 4/7 times the weight of an optimal 8-way cut, and that there exists a set of three of these cuts whose union is an 8-way cut and whose total weight is no more than average. This yields the claimed bound. Moreover, the running time is once again an improvement on the Isolation Heuristic, which in this case would require eight max flow computations. This is because we can show that it suffices to restrict attention to just seven of the 35 possible partitions of the eight terminals into sets of size four (the seven being derived from the rows of a Hadamard matrix).

Unfortunately, the above approach does not yield improvements over the Isolation Heuristic for any values of k other than 4 and 8. Is there some general technique that will improve on the Isolation Heuristic for arbitrarily large values of k? What about simply beating our bound for the case of $k = 3$?

Turning to our optimization algorithms for the planar case, the obvious question is whether the running times can be improved, although for the case of general k the improvement would have to be substantial to be interesting. For instance, although we would expect any algorithm to be exponential in k, the exponent containing k might not have to be attached to n. Could there be an algorithm whose running time was $c^k n^\alpha$, where α was independent of k?

REFERENCES

1. S. CHOPRA AND M. R. RAO, "On the multiway cut polyhedron," *Networks* **21** (1991), 51-89.
2. W. H. CUNNINGHAM, "The optimal multiterminal cut problem," *DIMACS Series in Disc. Math. and Theor. Comput. Sci.* **5** (1991), 105-120.
3. E. DAHLHAUS, D. S. JOHNSON, C. H. PAPADIMITRIOU, P. D. SEYMOUR, AND M. YANNAKAKIS, "The complexity of multiway cuts," extended abstract (1983).
4. P. L. ERDÖS AND L. A. SZEKELY, "Evolutionary trees: An integer multicommodity max-flow min-cut theorem," *Adv. in Appl. Math.*, to appear.
5. P. L. ERDÖS AND L. A. SZEKELY, "On weighted multiway cuts," unpublished manuscript (1991).
6. L. R. FORD, JR, AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.
7. G. N. FREDERICKSON, "Fast algorithms for shortest paths in planar graphs, with applications," *SIAM J. Comput.* **16** (1987), 1004-1022.
8. M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
9. M. R. GAREY, D. S. JOHNSON, AND L. STOCKMEYER, "Some simplified NP-complete graph problems," *Theor. Comput. Sci.* **2** (1976), 237-267.
10. A. V. GOLDBERG AND R. E. TARJAN, "A new approach to the maximum-flow problem," *J. Assoc. Comput. Mach.* **35** (1988), 921-940.
11. O. GOLDSCHMIDT AND D. S. HOCHBAUM, "Polynomial algorithm for the k-cut problem," *in* "Proceedings 29th Ann. Symp. on Foundations of Computer Science," IEEE Computer Society, Los Angeles, Calif., 1988, 444-451.
12. M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica* **1** (1981), 169-198.
13. D. S. HOCHBAUM AND D. B. SHMOYS, "An $O(|V|^2)$ algorithm for the planar 3-cut problem," *SIAM J. Algebraic and Discrete Methods* **6** (1985), 707-712.
14. T. C. HU, *Integer Programming and Network Flows*, Addison-Wesley Publishing Co., Reading, MA, 1969.
15. E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
16. D. LICHTENSTEIN, "Planar formulae and their uses," *SIAM J. Comput.* **11** (1982), 329-343.
17. C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.
18. C. H. PAPADIMITRIOU AND M. YANNAKAKIS, "Optimization, approximation, and complexity classes," *J. Comput. System Sci.* **43** (1991), 425-440.
19. H. SARAN AND V. V. VAZIRANI, "Finding k-cuts within twice the optimal," *in* "Proceedings 32nd Ann. Symp. on Foundations of Computer Science," IEEE Computer Society, Los Angeles, Calif., 1991, 743-751.
20. H. S. STONE, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. Software Engineering* **SE-3** (1977), 85-93.
21. M. YANNAKAKIS, P. C. KANELLAKIS, S. C. COSMADAKIS, AND C. H. PAPADIMITRIOU, "Cutting and partitioning a graph after a fixed pattern," *in* "Automata, Languages, and Programming," Lecture Notes in Computer Science, Vol. 154, Springer, Berlin, 1983, 712-722.