SIAM J. COMPUT.
Vol. 24, No. 1, pp. 101–108, February 1995

006

FINDING k CUTS WITHIN TWICE THE OPTIMAL*

HUZUR SARAN† AND VIJAY V. VAZIRANI†

Abstract. Two simple approximation algorithms for the minimum k-cut problem are presented. Each algorithm finds a k cut having weight within a factor of $(2 - 2/k)$ of the optimal. One algorithm is particularly efficient—it requires a total of only $n - 1$ maximum flow computations for finding a set of near-optimal k cuts, one for each value of k between 2 and n.

Key words. graph partitioning, minimum cuts, approximation algorithms

AMS subject classifications. 68Q20, 68Q25

1. Introduction. The minimum k-cut problem is as follows: Given an undirected graph $G = (V, E)$ with nonnegative edge weights and a positive integer k, find a set $S \subseteq E$ of minimum weight whose removal leaves k connected components. This problem is of considerable practical significance, especially in the area of VLSI design. Solving this problem exactly is NP hard [GH], but no efficient approximation algorithms were known for it.

In this paper we give two simple algorithms for finding k cuts. We prove a performance guarantee of $(2 - 2/k)$ for each algorithm; however, neither algorithm dominates the other in all instances. One of our algorithms is particularly fast; it requires only $n - 1$ max flow computations, using the classic result of [GoHu]. In fact, within the same running time, this algorithm finds near-optimal k cuts for each k, $2 \leq k \leq n$. We also give a family of graphs that show that the bound of $(2 - 2/k)$ is tight for each algorithm. The problem of achieving a factor of $(2 - \epsilon)$, for some fixed $\epsilon > 0$, independent of k, seems to be difficult, and may possibly be intractable.

The minimum k-cut problem and its variants have been extensively studied in the past. For fixed k, polynomial time algorithms for solving the problem exactly have been discovered by [DJPSY] for planar graphs and by [GH] for arbitrary graphs. These algorithms have running times of $O(n^{ck})$, for some constant c, and $O(n^{k^2})$, respectively. More efficient algorithms for the case of planar graphs and $k = 3$ are given in [He] and [HS]. Polynomial time algorithms have also been developed for several variants [Ch], [Cu], and [Gu].

In [DJPSY], the complexity of multiway cuts is studied: Given an edge-weighted undirected graph with k specified vertices, find a minimum weight k cut that separates these vertices. They show that this problem is NP hard even when k is fixed, for $k \geq 3$. They also give an approximation algorithm that finds a solution within a factor of $(2 - 2/k)$ of the optimal, using k max-flow computations. Using their algorithm as a subroutine one can get a $(2 - 2/k)$ approximation algorithm for our problem; however this will require $\binom{n}{k}$ calls and is therefore not polynomial time in case k is not fixed.

We shall first present the more efficient algorithm, which we call EFFICIENT. The other algorithm is called SPLIT. We will actually establish the $(2-2/k)$ performance guarantee for a slight variant of EFFICIENT. The weight of the k cut found by this variant dominates the k cuts found by both EFFICIENT and SPLIT. Finally, we report on some preliminary results obtained by applying these techniques to the balanced graph partitioning problem.

2. Algorithm EFFICIENT. Let $G = (V, E)$ be a connected undirected graph and let $wt : E \rightarrow \mathbf{Z}^+$ be an assignment of weights to the edges of G. We will extend the function wt to subsets of E in the obvious manner. A partition $(V', V - V')$ of V specifies a *cut*; the cut consists of all edges, S, that have one endpoint in each partition. We will denote a cut by the

*Received by the editors August 7, 1992; accepted for publication (in revised form) May 28, 1993.
†Department of Computer Science and Engineering, Indian Institute of Technology, New Delhi-110016, India.

set of its edges, S, and will define its *weight* to be $wt(S)$. For any set $S \subseteq E$, we will denote the number of connected components of the graph $G' = (V, E - S)$ by comps(S).

We will first find a k cut for a specified k. Our algorithm is based on the following greedy strategy: It keeps picking cuts until their union is a k cut and in each iteration it picks the lightest cut; in the original graph, that creates additional components.

Algorithm EFFICIENT:

1. For each edge e, pick a minimum weight cut, say, s_e, that separates the endpoints of e.
2. Sort these cuts by increasing weight, obtaining the list $s_1, \ldots, s_m$.
3. Greedily pick cuts from this list until their union is a k cut; cut s_i is picked only if it is not contained in $s_1 \cup \cdots \cup s_{i-1}$.

(Note: in case the algorithm ends with a cut B such that comps$(B) > k$, we can easily remove edges from B to get a cut B' such that comps$(B') = k$.)

Notice that since edge e is in s_e, $s_1 \cup \cdots \cup s_m = E$, and so this must be an n cut. Therefore, the algorithm will certainly succeed in finding a k cut.

Let $b_1, \ldots, b_l$ be the successive cuts picked by the algorithm. In the next lemma we show that with each cut picked we increase the number of components created, and hence the total number of cuts picked is at most $k - 1$.

LEMMA 2.1. *For each i, $1 \leq i < l$,* comps$(b_1 \cup \cdots \cup b_{i+1}) >$ comps$(b_1 \cup \cdots \cup b_i)$.

Proof. Because of the manner in which EFFICIENT picks cuts, $b_{i+1} \not\subseteq (b_1 \cup \cdots \cup b_i)$. Let (u, v) be an edge in $b_{i+1} - (b_1 \cup \cdots \cup b_i)$. Clearly, u and v are in the same component in the graph obtained by removing the edges of $b_1 \cup \cdots \cup b_i$ from G, and they are in different components in the graph obtained by removing $b_1 \cup \cdots \cup b_{i+1}$ from G. Hence, the latter graph has more components. □

COROLLARY 2.2. *The number of cuts picked, l, is at most $k - 1$.*

Notice that at each step we are indeed picking the lightest cut that creates additional components: among all edges that lie within connected components, we choose the edge whose endpoints can be disconnected with the lightest of the cuts from the original graph.

The complexity of our algorithm is dominated by the time taken to find cuts $s_1, \ldots, s_m$. This can clearly be done with m max flow computations. A more efficient implementation is obtained by using Gomory–Hu cuts. Gomory and Hu [GoHu] show that there is a set of $n - 1$ cuts in G such that for each pair of vertices, $u, v \in V$, the set contains a minimum weight cut separating u and v; moreover, they show how to find such a set using only $n - 1$ max flow computations. The cuts $s_1, \ldots, s_m$ can clearly be obtained from such a set of $n - 1$ cuts.

Incorporating all this, we get a particularly simple description of our algorithm. First, notice that step 3 is equivalent to

3′. Find the minimum i such that comps$(s_1 \cup \ldots \cup s_i) \geq k$. Output the k cut $s_1 \cup \ldots \cup s_i$.

Next, observe that when implemented with Gomory–Hu cuts, Algorithm EFFICIENT essentially boils down to the following:

1. Find a set of Gomory-Hu cuts in G.
2. Sort these cuts by increasing weight, obtaining the list $g_1, \ldots, g_{n-1}$.
3. Find the minimum i such that comps$(g_1 \cup \ldots \cup g_i) \geq k$.

The algorithm extends in a straightforward manner to obtaining near-optimal k cuts for each k, $2 \leq k \leq n$, with the same set of Gomory–Hu cuts.

3. Performance guarantee for EFFICIENT. In this section we shall prove the next theorem.

THEOREM 3.1. *Algorithm EFFICIENT finds a k cut having weight within a factor of* $(2 - 2/k)$ *of the optimal.*

Let $b_1, \ldots, b_l$ be the successive cuts found by Algorithm EFFICIENT; $B = b_1 \cup \cdots \cup b_l$. Central to our proof is a special property of these cuts with respect to an enumeration of all cuts in G. Let $c_1, c_2, \ldots$ be such an enumeration, sorted by increasing weight, such that $b_1, \ldots, b_l$ appear in this order in the enumeration. Such an enumeration will be said to be *consistent* with the cuts $b_1, \ldots, b_l$. For any cut c in G, index(c) is its index in this enumeration.

The union property. Let $s_1, \ldots, s_p$ be a set of cuts in G, sorted by increasing weight. Pick any enumeration of all cuts in G, $c_1, c_2, \ldots$ that is consistent with $s_1, \ldots, s_p$. We will say that $s_1, \ldots, s_p$ satisfy the union property if, intuitively, w.r.t. any such enumeration, the union of the cuts in any initial segment of $c_1, c_2, \ldots$ is equal to the union of all cuts s_j contained in this initial segment. More formally, pick any consistent enumeration of all cuts in G, let i be any index, $1 \leq i \leq$ index(s_p), and let s_q be the last cut in the sorted order having index at most i. Then, $c_1 \cup \cdots \cup c_i = s_1 \cup \cdots \cup s_q$.

LEMMA 3.2. *The cuts* $b_1, \ldots, b_l$ *satisfy the union property.*

Proof. Suppose the cuts do not satisfy the union property. Then there is an enumeration of all cuts in G, $c_1, c_2, \ldots,$ that is consistent with $b_1, \ldots, b_l$, and yet there is an index i, $i \leq$ index(b_l), such that

$$b_1 \cup \cdots \cup b_q \neq c_1 \cup \cdots \cup c_i,$$

where b_q is the last cut on our list having index at most i. The smallest such index will be referred to as the *point of discrepancy.* Let us fix an enumeration for which the point of discrepancy is maximum. Clearly, the point of discrepancy will have index less than index(b_l).

Let b_{q+1} be the next cut picked by our algorithm, and let index$(b_{q+1}) = j$. Because of the manner in which we fixed the enumeration, $wt(c_j) > wt(c_i)$ (otherwise we could interchange c_i and c_j, and obtain an enumeration in which discrepancy occurs at a larger index).

Clearly, $b_q \neq c_i$ and $c_1 \cup \cdots \cup c_{i-1} = b_1 \cup \cdots \cup b_q$. Let $e \in c_i - (c_1 \cup \cdots \cup c_{i-1})$. Then, $e \notin b_1 \cup \cdots \cup b_q$. Clearly, $wt(s_e) \leq wt(c_i)$, where s_e is a minimum weight cut that disconnects the endpoints of e. Now, our algorithm must pick edge e with a cut of weight at most $wt(s_e)$, and hence at most $wt(c_i)$. Since $wt(b_{q+1}) > wt(c_i)$, this means that $e \in b_1 \cup \cdots \cup b_q$, leading to a contradiction. □

Remark. Since $b_1, \ldots, b_l$ are drawn from $s_1, \ldots, s_m$, the latter cuts also satisfy the union property. For similar reasons, $g_1, \ldots, g_{n-1}$ satisfy the union property as well.

Let A be a minimum weight k cut in G. The second key idea in our proof is to view A as the union of k cuts as follows: Let $V_1, \ldots, V_k$ be the connected components of $G' = (V, E - A)$. Let a_i be the cut that separates V_i from $V - V_i$ for $1 \leq i \leq k$. Then $A = \bigcup_{i=1}^{k} a_i$. Notice that $\sum_{i=1}^{k} wt(a_i) = 2wt(A)$ (because each edge of A is counted twice in the sum). Assume without loss of generality that $wt(a_1) \leq wt(a_2) \leq \cdots \leq wt(a_k)$.

The $(2 - 2/k)$ bound is established by showing that the sum of weights of our cuts, $b_1, \ldots, b_l$ is at most the sum of weights of $a_1, \ldots, a_{k-1}$, i.e.,

$$\begin{aligned} wt(B) \leq \sum_{i=1}^{l} wt(b_i) &\leq \sum_{i=1}^{k-1} wt(a_i) \\ &\leq 2(1 - 1/k)wt(A), \end{aligned}$$

since a_k is the heaviest cut of A.

Actually, it will be simpler to prove a stronger statement. We will consider a slight variant of EFFICIENT that picks cuts with multiplicity; cut b_i will be picked t times if its inclusion created t additional components, i.e., if $(\text{comps}(b_1 \cup \ldots \cup b_i) - \text{comps}(b_1 \cup \ldots \cup b_{i-1})) = t$.

So, now we have exactly $k-1$ cuts; let us call them $b_1, \ldots, b_{k-1}$, to avoid introducing excessive notation. As before, $b_1, \ldots, b_{k-1}$ are sorted by increasing weight, and moreover we shall assume that cuts with multiplicity occur consecutively. We will show that

$$\sum_{i=1}^{k-1} wt(b_i) \le \sum_{i=1}^{k-1} wt(a_i) \ldots (i).$$

For the rest of the proof, we will study how the cuts a_i's and b_j's are distributed in $c_1, c_2, \ldots$, an enumeration of all cuts in G w.r.t. which $b_1 \ldots b_{k-1}$ satisfy the union property. Denote by α_i the number of all cuts a_j that have index $\le i$, i.e., $\alpha_i = |\{a_j |\ \text{index}(a_j) \le i\}|$. We will show that for each index i, $1 \le i \le \text{index}(a_{k-1})$, the number of cuts b_j having index $\le i$ is at least α_i (of course, cuts b_j are counted with multiplicity). If so, there will be a 1–1 map from $\{b_1, \ldots b_{k-1}\}$ onto $\{a_1, \ldots, a_{k-1}\}$ such that if b_i is mapped onto a_j, then index $(b_i) \le \text{index}(a_j)$. This will prove (i).

Two nice properties of the cuts a_i's and b_j's will help prove the assertion of the previous paragraph. Denote by A_i the union of all cuts a_j that have index $\le i$, i.e., $A_i = \bigcup_{a_j:\text{index}(a_j)\le i} a_j$. Similarly, let $B_i = \bigcup_{b_j:\text{index}(b_j)\le i} b_j$.

The next lemma shows that for each index i, the cuts b_j are making at least as much progress as the cuts a_j's, where progress is measured by the number of components created.

LEMMA 3.3. *For each index i*, $\text{comps}(A_i) \le \text{comps}(B_i)$.

Proof. The lemma is clearly true for $i > \text{index}(b_{k-1})$. For $i \le \text{index}(b_{k-1})$, $B_i = c_1 \cup \cdots \cup c_i$, since the cuts $b_1, \ldots, b_{k-1}$ satisfy the union property. Therefore, $A_i \subseteq B_i$, and the lemma follows. □

It is easy to construct an example showing that each cut a_j may not necessarily create additional components. Yet, at each index i, the number of components created by the cuts a_j is at least $\alpha_i + 1$. This is established in the next lemma.

LEMMA 3.4. *For each i, $1 \le i \le \text{index}(a_{k-1})$*, $\text{comps}(A_i) \ge \alpha_i + 1$.

Proof. Corresponding to each cut a_j having index $\le i$, the partition V_j will be a single connected component all by itself in the graph $G_i = (V, E - A_i)$. Let us charge a_j to this component of G_i. Since index $(a_k) > i$, the component of G_i containing V_k will not get charged. The lemma follows. □

At this point we have all the ingredients to finish the proof. Consider an index i, $1 \le i \le \text{index}(a_{k-1})$. By Lemma 3.4, $\text{comps}(A_i) \ge \alpha_i + 1$. This together with Lemma 3.3 gives $\text{comps}(B_i) \ge \alpha_i + 1$. For each additional component created by us, we have included a cut b_j (by including cuts with appropriate multiplicity), and thus it follows that the number of cuts b_j having index $\le i$ is at least α_i. The theorem follows.

Remark. The proof given above shows that any set of cuts satisfying the union property will give a near-optimal k cut. This explains why the Gomory–Hu cuts work.

Clearly, the proof given above holds *simultaneously* for each value of k between 2 and n. Hence we get a set of near-optimal k cuts, $2 \le k \le n$. Notice that at the extremes, i.e., for $k = 2$ and $k = n$, we get optimal cuts.

THEOREM 3.5. *Algorithm EFFICIENT finds a set of k cuts, one for each value of k, $2 \le k \le n$; each cut is within a factor of $(2 - 2/k)$ of the optimal k cut. The algorithm requires a total of $n - 1$ max flow computations.*

Using the current best known max flow algorithm [GT], [KRT], our algorithm has a running time of $O(mn^2 + n^{3+\epsilon})$, for any fixed $\epsilon > 0$.

4. Algorithm SPLIT. Perhaps the first heuristic that comes to mind for finding a k cut is the following.

Algorithm SPLIT: Start with the given graph. In each iteration, pick the lightest cut in the current graph that splits a component, and remove the edges of this cut. Stop when the current graph has k connected components.

Notice that SPLIT, like EFFICIENT, is also a greedy algorithm. Whereas EFFICIENT picks a lightest cut in the original graph that creates additional components, SPLIT picks a lightest cut in the current graph.

SPLIT needs to find a minimum weight cut in each new component formed—this can be done using $n-1$ max flow computations in a graph on n vertices. Hence SPLIT requires $O(kn)$ max flow computations to find a k cut.

We shall establish a $(2-2/k)$ performance guarantee for SPLIT as well. However, first let us point out that neither algorithm dominates the other. Consider the following graph on eight vertices, $\{a, b, c, d, e, f, g, h\}$. The edges and their weights are as follows.

$wt(a,b) = 3$,
$wt(a,c) = 3$,
$wt(b,d) = 7$,
$wt(c,e) = 7$,
$wt(d,e) = 10$,
$wt(d,f) = 4$,
$wt(f,g) = 5$,
$wt(g,h) = 5$,
$wt(e,h) = 4$.

Now, for $k = 3$, the cuts found by SPLIT and EFFICIENT have weights 13 and 14, respectively, but for $k = 4$, the weights are 20 and 19, respectively.

THEOREM 4.1. *The k cut found by Algorithm SPLIT has weight within a factor of* $(2-2/k)$ *of the optimal.*

Proof. In Theorem 3.1 we showed that a slight variant of EFFICIENT, that picks cuts with appropriate multiplicity, has a performance bound of $(2-2/k)$. We will now prove that the weight of the k cut found by SPLIT is dominated by the weight of the k cut found by this variant.

Let $b_1, \ldots, b_{k-1}$ be the cuts picked by the variant. Notice that since SPLIT picks a minimum weight cut in a component, it splits it into exactly two components. Therefore, SPLIT picks exactly $k-1$ cuts, say, $d_1, \ldots, d_{k-1}$.

By induction on i, we will show that $wt(d_i) \leq wt(b_i)$ for $1 \leq i \leq k-1$. The assertion is clearly true for $i = 1$. To show the induction step, first notice that comps$(d_1 \cup \cdots \cup d_i) = i+1$ and comps$(b_1 \cup \cdots \cup b_{i+1}) \geq i+2$. Therefore, there is a cut b_j, $1 \leq j \leq i+1$, that is not contained in $(d_1 \cup \cdots \cup d_i)$. By the proof of Lemma 2.1, this cut will create additional components, and is available to SPLIT. Hence, SPLIT will pick a cut of weight at most $wt(b_j)$. Since $wt(b_j) \leq wt(b_{i+1})$, the assertion follows. □

5. Lower bound. We will show that the bounds established in Theorem 3.1 and Theorem 4.1 are tight in the following sense.

THEOREM 5.1. *For any* ϵ, $0 < \epsilon \leq 1$, *there is an infinite family of minimum k-cut instances* (G, wt, k) *such that the weight of the k cut found by each algorithm, EFFICIENT and SPLIT, lies in the range*

$$[(1-\epsilon)(2-2/k)W,\ (2-2/k)W],$$

where W is the weight of an optimal k cut for the instance. Moreover, k is unbounded in this family.

Proof. The kth instance, in which we want to find a k cut, consists of a graph on $2k-1$ vertices, $V = \{u_1, \ldots, u_{k-1}, v_1, \ldots, v_k\}$. The only edges are (with weights specified)

$$\begin{aligned} wt(u_i, u_{i+1}) &= \beta \quad \text{for } 1 \le i \le k-2, \\ wt(u_{k-1}, v_1) &= \beta, \\ wt(v_i, v_{i+1}) &= \alpha \quad \text{for } 1 \le i \le k-1, \\ wt(v_1, v_k) &= \alpha, \end{aligned}$$

where α is a positive integer, and $\beta = 2\alpha(1-\epsilon)$.

The minimum k cut, A, picks all edges of weight α, and so $wt(A) = k\alpha$. Each algorithm, EFFICIENT and SPLIT, picks the k cut, B, consisting of all edges of weight β. So $wt(B) = (k-1)\beta = 2(k-1)\alpha(1-\epsilon)$. Hence

$$wt(B)/wt(A) = (2 - 2/k)(1-\epsilon).$$

This proves the theorem. □

6. Balanced graph partitioning. Given an edge-weighted graph, $G = (V, E)$, $wt : E \to \mathbf{Z}^+$, with an even number of vertices, the balanced graph partitioning problem asks for a partition of V into two sets, V_1 and V_2, each containing half the vertices, such that the weight of the cut separating V_1 and V_2 is minimized. This problem is NP hard [GJ]. It models realistic situations such as circuit partitioning and is frequently used in practice. (See [KL] for the well-known "swap" heuristic for this problem.) We give the first approximation algorithm for this problem; it achieves a performance ratio of $n/2$. The algorithm extends in a straightforward manner to the problem of partitioning the graph into k equal size pieces, for any fixed k. The performance ratio achieved for this problem is $(\frac{k-1}{k})n$. In the past, [LR] and [LMPSTT] have used multicommodity flow for obtaining a $\frac{1}{3}, \frac{2}{3}$ graph partition algorithm that is within an $O(\log n)$ factor of the best balanced partition (in a $\frac{1}{3}, \frac{2}{3}$ graph partition each side of the cut has between $\frac{1}{3}$ and $\frac{2}{3}$ fraction of the vertices).

Our algorithm uses the fact that there is a pseudopolynomial time algorithm for determining whether n given numbers $a_1, \ldots, a_n$ can be partitioned into two sets each of which adds up to $W/2$, where $\sum_{i=1}^{n} a_i = W$. This algorithm is based on a straightforward dynamic programming approach, and has a running time of $O(nW)$ [GJ]; if the answer is "yes," it finds a valid partition as well.

Balanced graph partitioning algorithm:

1. Find a set of Gomory-Hu cuts in G.
2. Sort these cuts by increasing weight, obtaining $g_1, \ldots, g_{n-1}$.
3. Find the minimum i such that the connected components of $G' = (V, E - (g_1 \cup \cdots \cup g_i))$ can be partitioned into two sets containing $\frac{n}{2}$ vertices each.

The complexity of our algorithm is dominated by step 1, since the total time required by step 3 is $O(n^3)$. We will use the following property of the partitioning problem to establish the bound of $n/2$.

LEMMA 6.1. *Let n be an even integer, and let $a_1, \ldots a_{(n/2)+1}$ be positive integers such that*

$$\sum_{i=1}^{\frac{n}{2}+1} a_i = n.$$

Then the answer to the partitioning problem is "yes."

Proof. Without loss of generality assume that the a_i's are sorted in decreasing order. Notice that in order to maintain a sum of n, if $a_1 = 2$ then $a_2 = \cdots = a_{(n/2)-1} = 2$ and $a_{n/2} = a_{(n/2)+1} = 1$. In this case, the a_i's can clearly be partitioned. In general, let k be the largest index such that $a_k > 1$, i.e., the last $\frac{n}{2} + 1 - k$ numbers are 1's. It is easy to see that the sizes of a_i's exceeding 2 determine the number of 1's in the following manner:

$$\sum_{i=1}^{k}(a_i - 2) = \left(\frac{n}{2} + 1 - k\right) - 2. \tag{1}$$

Now, associate $a_i - 2$ distinct 1's with each a_i that exceeds 2. By (1) this is feasible, and moreover exactly two 1's will be left over unassociated. For each a_i exceeding 2, we will include a_i in one partition and its associated 1's in the other partition. This effectively gives the former partition a weight of 2 more than the latter. Hence, once again we are essentially left with partitioning numbers of the form $2, 2, \ldots, 2, 1, 1$. □

As before, let $c_1, c_2, \ldots$ be an enumeration of all cuts in G, ordered by increasing weight, and let $B = g_1 \cup \cdots \cup g_i$ be the set of edges picked by our algorithm. Let c_k be the first cut in the enumeration that yields an optimal balanced partitioning of G.

LEMMA 6.2. $wt(B) \leq \frac{n}{2} wt(c_k)$.

Proof. As in Algorithm EFFICIENT, among the cuts $g_1, \ldots, g_i$, pick g_j iff it is not contained in $(g_1 \cup \cdots \cup g_{j-1})$. Let $b_1, \ldots, b_l$ be the cuts picked in this manner. Clearly, $B = b_1 \cup \cdots \cup b_l$. The proof is based on Lemma 3.2, i.e., the fact that the cuts $b_1, \ldots, b_l$ satisfy the union property. This and the fact that the components of $G' = (V,\ E - C)$ can certainly be partitioned into equal sized sets, for any set C containing c_k, imply that index$(b_l) \leq k$. Now by Lemma 2.1, $l \leq n - 1$, thereby giving a factor of $(n - 1)$. To achieve a better factor, notice that Lemma 6.1 implies that $l \leq n/2$. Therefore $wt(B) \leq \sum_{i=1}^{l} wt(b_i) \leq \frac{n}{2} wt(c_k)$. □

LEMMA 6.3. *The bound of* $\frac{n}{2}$ *is tight for our algorithm.*

Proof. As in Theorem 5.1, for any ϵ, $0 < \epsilon < 1$, we will give an infinite family of instances on which the cut found by our algorithm lies in the range $[(1 - \epsilon)\frac{n}{2} W,\ \frac{n}{2} W]$, where W is the weight of the optimal cut. The kth instance consists of a graph on $2k$ vertices $\{u, v, u_1 \ldots u_{k-1}, v_1, \ldots, v_{k-1}\}$, and edges (with weights) $wt(u_i, u) = \alpha, 1 \leq i \leq k - 1$, $wt(v_i, v) = \alpha$, $1 \leq i \leq (k - 1)$, $wt(u, v) = \beta$, where $\alpha = (1 - \epsilon)\beta$. It is easy to see that our algorithm finds a cut of weight $k\alpha$, and the optimal cut has weight β. □

THEOREM 6.4. *The algorithm given above finds a balanced partitioning of an edge-weighted graph, using* $n - 1$ *max-flow computations. The weight of the cut found is within a factor of* $n/2$ *of the optimal. Moreover, the bound of* $n/2$ *is tight for our algorithm.*

7. Discussion and open problems. It will be interesting to see how Algorithms SPLIT and EFFICIENT compare in practice, even though neither algorithm dominates the other in worst-case performance. Our guess is that SPLIT will typically give lighter k cuts.

Notice that the graphs given in Theorem 5.1 help establish the $(2 - 2/k)$ lower bound for the k cut found for each k, $2 \leq k \leq (n + 1)/2$, where n is the number of vertices in the graph (n is odd), but not for higher values of k. Certainly, the bound of $(2 - 2/k)$ is not tight for $k = n$, since we get the optimal cut. Is there a better analysis of our algorithms for $k > n/2$?

Is there a better approximation algorithm for the minimum k-cut problem? We believe that the problem of achieving a factor of $(2 - \varepsilon)$, for some $\varepsilon > 0$, independent of k, is intractable. The minimum k-cut problem is MAX-SNP complete [Pa], and hence by the recent result of [ALMSS], achieving a factor of $1 + \varepsilon$, is NP complete for some $\varepsilon > 0$.

Our investigation of the balanced graph partitioning problems appears to be quite preliminary, and it should be possible to improve the bound. Interesting special cases are (a) the graph is planar, and (b) all edge weights are 1. Since the graphs used in Lemma 6.3 are planar, the lower bound for our algorithm holds for case (a). It is easy to see that if ties are resolved arbitrarily in our algorithm, Lemma 6.3 holds for case (b) as well; however, this seems to be a small hurdle. It will also be useful to consider the version of the balanced graph partitioning problem in which vertices have weights; in this case the pseudopolynomial algorithm for partition will not be useful.

Finally, some of these methods may be useful for obtaining approximation algorithms for other NP-hard graph partitioning problems (see [GH] and [GJ]).

Acknowledgments. We wish to thank Fan Chung, Samir Khuller, Laszlo Lovasz, Milena Mihail, Christos Papadimitriou, Umesh Vazirani, and Mihalis Yannakakis for valuable discussions and comments.

REFERENCES

[ALMSS] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and intractability of approximation problems*, Proc. 33rd Annual Symp. on the Foundations of Computer Science, 1992, pp. 14–23.

[Ch] V. Chvátal, *Tough graphs and Hamiltonian circuits*, Discrete Math., 5 (1973), pp. 215–228.

[Cu] W. H. Cunningham, *Optimal attack and reinforcement of a network*, J. Assoc. Comput. Mach., 32 (1985), pp. 549–561.

[DJPSY] E. Dalhaus, D. S. Johnson, C. H. Papadimitriou, P. Seymour, and M. Yannakakis, *The complexity of the multiway cuts*, Proc. 24th Annual ACM Symposium on the Theory of Computing, 1992, pp. 241–251.

[GoHu] R. Gomory and T. C. Hu, *Multi-terminal network flows*, J. SIAM, 9 (1961), pp. 551–570.

[GH] O. Goldschmidt and D. S. Hochbaum, *Polynomial algorithm for the k-cut problem*, Proc. 29th Annual Symp. on the Foundations of Computer Science, 1988, pp. 444–451.

[GJ] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, CA, 1979.

[GT] A. V. Goldberg and R. E. Tarjan, *A new approach to the maximum flow problem*, Proc. of the 18th Annual ACM Symp. on Theory of Computing, 1987, pp. 136–146.

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

[He] X. He, *On the planar 3-cut problem*, J. Algorithms, 12 (1991), pp. 23–37.

[HS] D. Hochbaum and D. Shmoys, *An $O(V^2)$ algorithm for the planar 3-cut problem*, SIAM J. Discrete Meth., 6:4:707–712, 1985.

[KL] B. W. Kernighan and S. Lin, *An efficient heuristic for partitioning graphs*, Bell Systems Technical Journal, 40 (1970), pp. 291–308.

[KRT] V. King, S. Rao, and R. Tarjan, *A faster deterministic maximum flow algorithm*, Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms, 1992, pp. 157–164.

[LMPSTT] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas, *Fast approximation algorithms for multicommodity flow problems*, Proc. 23rd Annual ACM Symp. on the Theory of Computing, 1991, pp. 101–111.

[LR] T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, Proc. 29th Annual Symp. on the Foundations of Computer Science, 1988, pp. 422–431.

[Pa] C. Papadimitriou, private communication, 1991.