Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 186 (2008) 77–90
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

Approximating k-cuts using network strength as a Lagrangean relaxation

R. Ravi [a], Amitabh Sinha [b,*]

[a] *Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
[b] *Ross School of Business, University of Michigan, Ann Arbor, MI 48109, USA*

Received 16 December 2005; accepted 15 January 2007
Available online 25 February 2007

Abstract

Given an undirected, edge-weighted connected graph, the k-cut problem is to partition the vertex set into k non-empty connected components so as to minimize the total weight of edges whose end points are in different components.

We present a combinatorial polynomial-time 2-approximation algorithm for the k-cut problem. We use a Lagrangean relaxation (also suggested by Barahona [F. Barahona, On the k-cut problem, Operations Research Letters 26 (2000) 99–105]) to reduce the problem to the *attack* problem, for which a polynomial time algorithm was provided by Cunningham [W. Cunningham, Optimal attack and reinforcement of a network, Journal of the ACM 32(3) (1985) 549–561]. We prove several structural results of the relaxation, and use these results to develop an approximation algorithm.

We provide analytical comparisons of our algorithm and lower bound with two others: Saran and Vazirani [H. Saran, V. Vazirani, Finding k-cuts within twice the optimal, SIAM Journal of Computing 24(1) (1995) 101–108] and Naor and Rabani [J. Naor, Y. Rabani, Tree packing and approximating k-cuts. In: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 26–27]. We also provide computational results comparing the performance of our algorithm on random graphs with respect to the lower bound provided by the attack problem as well as an alternate 2-approximation algorithm provided by Saran and Vazirani [Cited above].

Keywords: Combinatorial optimization; Graph theory; Approximation algorithm; Lagrangean relaxation

1. Introduction

Among the most fundamental optimization problems in graph theory are *cut* problems, where the objective is usually a minimum-weight set of edges which cuts the graph into two or more pieces under certain constraints. In its simplest form, the minimum cut problem considers an undirected, edge-weighted graph, and searches for the minimum-weight set of edges whose removal partitions the graph into at least two components. This problem was among the first graph optimization problems for which polynomial time algorithms

* Corresponding author. Tel.: +1 734 615 4182.
E-mail addresses: ravi@cmu.edu (R. Ravi), amitabh@umich.edu (A. Sinha).

0377-2217/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2007.01.040

were obtained, and various associated theoretical developments such as the famous *Max-flow min-cut theorem* have contributed significantly to the theory and practice of graph optimization problems.

Consequently, several generalizations of the basic minimum cut problem have also been considered. Among the simplest generalizations one can think of is to make the number of connected components required a part of the input. Specifically, given a graph *and a positive integer* k, one asks for the minimum-weight set of edges whose deletion creates at least k connected components. This is known as the k-cut problem, and is the object of study in this paper.

1.1. Problem definition

Given an undirected, connected graph $G = (V, E)$ with non-negative edge weights s, and a positive integer $k < |V|$, a k-cut is a set of edges $A \subseteq E$ whose removal partitions G into k or more connected components. The k-cut problem is to find a minimum-weight k-cut in G, where the weight of a k-cut is the sum of the weights of the edges in the cut.

1.2. Related work

While the k-cut problem generalizes the minimum cut problem, it was shown to be NP-hard by Goldschmidt and Hochbaum [6]. For fixed k, they also gave an exact algorithm with running time $O(|V|^{k^2})$.

Saran and Vazirani [10] gave a polynomial time $(2 - \frac{2}{k})$ approximation algorithm which finds minimum cuts until the graph has been partitioned into k connected components. Naor and Rabani [8] also provided a 2-approximation algorithm, by rounding an integer linear program formulation of the problem. We compare our work with both of these works in Sections 5 and 6. Recently, Chekuri et al. [3] extended the algorithm in [8] to handle Steiner k-cuts, where each component must contain at least one of a specified set of terminal vertices. Karger and Stein [7] gave a randomized algorithm which runs in expected time $O(|V|^{2(k-1)})$ and finds an exact solution.

The Lagrangean approach of this paper was also suggested by Barahona [2], though this work is independent of his. Barahona considered the value of the optimal k-cut as a function of k, and proved that the Lagrangean relaxation provides a lower bound as well as the convex envelope of the k-cut function. However, he was unable to bound the difference between the solution cost and the lower bound. Our work was done independently of [2], and one of the contributions of this paper is a tight bound on the ratio of the solution cost to the lower bound, thus answering an implicit open question in [2]. In doing so, we also develop various structural properties of the Lagrangean relaxation which were not considered in [2].

1.3. This paper: results and organization

Our primary contribution is an alternate 2-approximation to the k-cut problem. We use a Lagrangean relaxation of the k-cut problem to reduce it to the attack problem (defined shortly). Cunningham [4] gave a polynomial time algorithm to solve the attack problem optimally, which we use to solve the Lagrangean relaxation. We study the relaxation in detail and derive its structural properties in Section 3, after some preliminaries in Section 2.

Our approximation algorithm is developed and described in Section 4. The structure of the solution of the attack problem leads to a fast algorithm for solving the Lagrangean relaxation, which we use in our algorithm for the k-cut problem. We then derive our approximation ratio.

We discuss the merits and demerits of our algorithm in Section 5. We first show examples demonstrating that our algorithm is theoretically incomparable with that of Saran and Vazirani, since there exist examples where each algorithm achieves its worst-case bound while the other finds the optimal solution. We then show that our lower bound always has weight at least as much as the lower bound provided by Naor and Rabani [8]. We also show the limitations of our approach using an example which bounds from below the approximation ratio achievable by algorithms using the Lagrangean relaxation as a subroutine.

Finally, we provide computational results in Section 6 by studying the performance of our algorithm on random graphs (of both the Bernoulli and preferential attachment types). We compare the cost of the solution

provided by our algorithm to the lower bound as well as to the cost of the solution provided by the algorithm of Saran and Vazirani [10]. Briefly, our results indicate that: (i) the lower bound is usually fairly good, (ii) the lower bound worsens with graph density, (iii) the two algorithms often produce the same cuts, and (iv) the two algorithms are incomparable in that there is no empirical evidence to claim that one outperforms the other.

2. Preliminaries

2.1. Notation

The input consists of an undirected, connected graph $G = (V, E)$, a positive integer k, and a weight function on the edges given by $s : E \to \mathbb{R}^+$. Let $A \subseteq E$ be a subset of the edges of G. If $G' = (V, E \setminus A)$ has l connected components, we say that A is an l-cut, and denote it as $\kappa(A) = l$. The weight of A is defined as $s(A) = \sum_{e \in A} s_e$. Therefore, the k-cut problem can be written as the following:

$$\min_{A \subseteq E : \kappa(A) \geqslant k} s(A).$$

If G is a connected graph with all edge weights strictly positive, then a minimum-weight k-cut will never have more than k components. However, if G is initially disconnected, or has some zero-length edges, then it is possible that an optimal solution to the k-cut problem has more than k components. For the purpose of our algorithm, zero-length edges are allowed. Moreover, if the G is initially disconnected, it can be transformed into a connected graph by adding zero-length edges. For this reason, we continue to define the k-cut problem as requiring a minimum-weight set of edges which results in at least k components, as opposed to exactly k components.

Note that the definition of κ does not require G to be connected. Indeed, even if G is a graph with several distinct components not connected to each other, we may define $\kappa(A)$ for any edge-set A to be the *total* number of connected components in G after the removal of the edges in A.

2.2. Strength and attack of a graph

Define the strength of an edge set A to be $\sigma(A) = \frac{s(A)}{\kappa(A)-1}$, with $\sigma(A) = \infty$ if $\kappa(A) = 1$. The strength of G is $\sigma(G) = \min_{A \subseteq E} \sigma(A)$. By definition, $0 \leqslant \sigma(G) \leqslant \frac{s(E)}{|V|-1}$. Cunningham [4] provided a polynomial time algorithm to compute the strength of a graph by reducing it to the attack problem, described next.

As a function of a positive real parameter b, the attack value of an edge set $A \subseteq E$ is given by $g_A(b) = s(A) - b(\kappa(A) - 1)$. The attack value of the graph is given by $g(b) = \min_{A \subseteq E} g_A(b)$. The attack problem is to compute $g(b)$, and Cunningham showed that we need no more than $|V|$ computations of $g(b)$ to compute $\sigma(G)$. We define a new function $g'(b) = g(b) + b(k-1)$ which we use in our algorithm and analysis.

3. Lagrangean relaxation

The algorithm is motivated by the following Lagrangean relaxation of the k-cut problem, which was also suggested by Barahona [2]:

$$\min_{A \subseteq E : \kappa(A) \geqslant k} s(A) \geqslant \max_{b \geqslant 0} \min_{A \subseteq E} s(A) + b(k - \kappa(A)) \tag{1}$$

$$= \max_{b \geqslant 0} \min_{A \subseteq E} g_A(b) + b(k-1) \tag{2}$$

$$= \max_{b \geqslant 0} g'(b). \tag{3}$$

Note that while there is no way to express $\kappa(A) \geqslant k$ in terms of linear constraints, the bound provided above by the Lagrangean relaxation continues to hold. Our broad strategy for an approximation algorithm is to solve the Lagrangean relaxation optimally, and use the solution of the Lagrangean relaxation to develop a 2-approximate solution.

3.1. Structural properties of the relaxation

In this section, we study the combinatorial structure of solutions to the attack problem, which is the subproblem resulting from the Lagrangean relaxation of the k-cut problem. The results derived here are then used to obtain a fast algorithm to solve the Lagrangean relaxation, and to obtain the approximate solution to k-cut.

The next two lemmas are implicit in Cunningham's work [4], but we derive them here for completeness.

Lemma 3.1 (implicit in Cunningham [4]). *If $b < b'$, and A and A' are edge sets minimizing $g(b)$ at b and b', respectively (that is, $g(b) = g_A(b)$ and $g(b') = g_{A'}(b')$), then $\kappa(A) \leqslant \kappa(A')$.*

Proof. We prove the lemma by contradiction; suppose $\kappa(A) > \kappa(A')$. Therefore, we have $(b' - b)\kappa(A') < (b' - b)\kappa(A)$, and since A' is an optimal solution for $g(b')$, we have:

$$\begin{aligned} s(A') - b'(\kappa(A') - 1) \leqslant s(A) - b'(\kappa(A) - 1) &\Rightarrow s(A') - b'(\kappa(A') - 1) + (b' - b)\kappa(A') \\ &< s(A) - b'(\kappa(A) - 1) + (b' - b)\kappa(A) \Rightarrow s(A') - b(\kappa(A') - 1) < s(A) - b(\kappa(A) - 1) \end{aligned}$$

which contradicts the optimality of A for $g(b)$. □

A *breakpoint* of a piecewise linear function is a point at which the function is non-linear. At a breakpoint b_0, we have the following: there are two edge sets A and B such that for $b < b_0$, we have $g_A(b) < g_B(b)$, while for $b > b_0$ we have $g_A(b) > g_B(b)$. Also, $g_A(b_0) = g_B(b_0)$. In this case, we say the breakpoint b_0 is *induced by the edge sets A and B*.

Lemma 3.2 (implicit in Cunningham [4]). *The functions $g(b)$ and $g'(b)$ are continuous, concave, piecewise linear, and have no more than $|V| - 1$ breakpoints. Moreover, $g(b)$ is non-increasing in b.*

Proof. Since $g(b)$ is a lower envelope of a finite set of linear functions $g_A(b)$, it is continuous, concave and piecewise linear. Moreover, since each $g_A(b)$ is a non-increasing function of b, so is $g(b)$.

Suppose b_0 is a breakpoint of $g(b)$, induced by two edge sets A and B, such that for $b < b_0$ we have $g_A(b) < g_B(b)$. Hence for $b > b_0$, we must have $g_A(b) > g_B(b)$. But this can only happen if $\kappa(B) > \kappa(A)$. Using Lemma 3.1 and the fact that $\kappa(A)$ has to be an integer between 1 and $|V| - 1$ for every edge set A, we can have at most $|V| - 1$ breakpoints in $g(b)$.

Since $g'(b) = g(b) + b(k - 1)$, all of the above continue to hold except for the fact that $g'(b)$ need not be non-increasing in b. □

We now make a technical assumption which will help in the analysis.

A graph G is called *non-degenerate* if the following holds:

- If b is not a breakpoint of $g(b)$, then there is a unique edge set A such that $g_A(b) = g(b)$.
- If b is a breakpoint of $g(b)$, then there are exactly two edge sets A and B such that $g_A(b) = g_B(b) = g(b)$.

Suppose we perturb each edge weight by a small random number, so that for any two distinct edge sets A and B, the functions $g_A(b)$ and $g_B(b)$ are different. Such a perturbation will also achieve the second condition in our definition of non-degeneracy, and will not change the problem or the objective function significantly. Hence non-degeneracy is a valid assumption, and for the rest of this paper, we work under the assumption that our graph is non-degenerate.

An alternate way to achieve non-degeneracy is as is done for the simplex method for linear programming. Pick a small $\epsilon > 0$, and list all edges in an (arbitrary) lexicographic ordering. To the ith edge in this ordering, add ϵ^i to its weight. Since edges can be compared according to the lexicographic ordering, we avoid the computational overhead needed to actually maintain high-precision edge weights. Clearly, this forces the graph to be non-degenerate without changing the structure of the graph if ϵ is small enough. Therefore, this is a practical way to enforce non-degeneracy without causing any loss in the quality of the solution or the run-time of the algorithm.

A function $h : 2^E \to \mathbb{R}$ is called *supermodular* if for every $A, B \subseteq E$ we have $h(A \cap B) + h(A \cup B) \geqslant h(A) + h(B)$. Conversely, h is called *submodular* if for every $A, B \subseteq E$ we have $h(A \cap B) + h(A \cup B) \leqslant h(A) + h(B)$. A well known submodular function is the *graphic matroid rank function f*, where $f(A)$ is defined to be the maximum cardinality of a forest in A.

Lemma 3.3. *The function $\kappa : 2^E \to [n]$ is supermodular.*

Proof. By definition of κ, we have $\kappa(A) + f(\overline{A}) = |V|$, where $\overline{A} = E \setminus A$. (Note that G is connected.) Supermodularity of κ now follows immediately from submodularity of f, and can also be verified as follows:

$$\begin{aligned} f(\overline{A} \cap \overline{B}) + f(\overline{A} \cup \overline{B}) \leqslant f(\overline{A}) + f(\overline{B}) &\Rightarrow f(\overline{A \cup B}) + f(\overline{A \cap B}) \leqslant f(\overline{A}) + f(\overline{B}) \\ &\Rightarrow |V| - f(\overline{A \cup B}) + |V| - f(\overline{A \cap B}) \geqslant |V| - f(\overline{A}) + |V| - f(\overline{B}) \\ &\Rightarrow \kappa(A \cup B) + \kappa(A \cap B) \geqslant \kappa(A) + \kappa(B). \qquad \square \end{aligned}$$

Let $s(A, k)$ be the weight of a minimum-weight k-cut on A, where k is now provided as part of the input. Similarly, let $g'(b, k)$ be defined by extending the definition of $g'(b)$ to incorporate k as an argument. It follows from (1)–(3) that $s(A, k) \geqslant g'(b, k)$ for all k. Barahona [2] proved that $\max_{b \geqslant 0} g'(b, k)$ is the convex envelope of $s(A, k)$, using an analysis similar to using Lemmas 3.1–3.3. He concluded with a few computational results, leaving open the question of developing an algorithm which actually utilizes the lower bound. He also did not provide any theoretical guarantees on the quality of the lower bound. We provide both of these, using further structural results which are developed below.

Theorem 3.4. *If b_0 is a breakpoint of $g'(b)$ induced by edge sets A and B where $\kappa(A) > \kappa(B)$, then $B \subset A$.*

Proof. Observe the following set of inequalities:

$$\begin{aligned} g_A(b_0) + g_B(b_0) &\leqslant g_{A \cup B}(b_0) + g_{A \cap B}(b_0) = s(A \cup B) - b_0(\kappa(A \cup B) - 1) + s(A \cap B) - b_0(\kappa(A \cap B) - 1) \\ &\leqslant s(A) + s(B) - b_0(\kappa(A) - 1) - b_0(\kappa(B) - 1) = g_A(b_0) + g_B(b_0). \end{aligned}$$

The first inequality follows from the optimality of A and B for the breakpoint b_0, while the second inequality follows from supermodularity of κ. Therefore, equality holds through the chain of inequalities above, and we have

$$g_A(b_0) + g_B(b_0) = g_{A \cup B}(b_0) + g_{A \cap B}(b_0).$$

But optimality of A and B at b_0 and the definition of non-degeneracy imply that the unique minimizers of $g(b)$ at $b = b_0$ are A and B. Therefore, the pair $(A \cup B, A \cap B)$ is identical to the pair (A, B). Since $\kappa(A) > \kappa(B)$, we must have $A \cup B = A$ and $A \cap B = B$. That is, $B \subset A$. $\square$

Corollary 3.5. *If b_0 is a breakpoint of $g'(b)$ induced by edge sets A and B such that $\kappa(A) > \kappa(B)$, then $A \setminus B$ is contained in some connected component of $G' = (V, E \setminus B)$.*

Proof. Suppose no one connected component of G' contains $A \setminus B$. Then $A \setminus B$ is contained in at least two components. Pick any one such component, and let X be the set of edges of $A \setminus B$ contained in that component. Let $Y = (A \setminus B) \setminus X$, so that we have $B \subset B \cup X \subset A$ as well as $B \subset B \cup Y \subset A$.

Using the optimality of A and B, we have the following:

$$\begin{aligned} g_A(b_0) + g_B(b_0) &\leqslant g_{B \cup X}(b_0) + g_{B \cup Y}(b_0) = s(B \cup X) - b_0(\kappa(B \cup X) - 1) + s(B \cup Y) - b_0(\kappa(B \cup Y) - 1) \\ &= s(A) + s(B) - b_0(\kappa(A) - 1) - b_0(\kappa(B) - 1) = g_A(b_0) + g_B(b_0). \end{aligned}$$

The penultimate equality above ($\kappa(B \cup X) + \kappa(B \cup Y) = \kappa(A) + \kappa(B)$) follows from the fact that all the edges in X are in a single connected component of G', while none of the edges in Y are in this component. Hence the set of new components created in G' by the deletion of the edges in X is distinct from the set of new components created by the deletion of the edges in Y. In other words, the total number of new components created by the deletion of the edges in $A \setminus B$ ($= X \cup Y$) is precisely the number of new components created by X added to the number of new components created by Y.

Therefore, equality holds in the above chain of inequalities. If X and Y are both non-empty, then this violates non-degeneracy, which is a contradiction. Therefore, either X or Y must be empty, so that all edges of $A \setminus B$ are contained in a single connected component of G'. □

Theorem 3.6. *Let b_0 be a breakpoint of $g'(b)$, induced by edge set A. The next breakpoint is induced by the edge set which is the solution to the strength problem on the smallest strength component of $G' = (V, E \setminus A)$.*

Proof. We prove the theorem by contradiction. Let X be the edge set of the next breakpoint (by Corollary 3.5, this is contained in some connected component G_X of G'), and Y be the edge set which is the solution to the strength problem on the smallest strength component G_Y of G'. For contradiction, we assume that $X \neq Y$. Let $\kappa_X(X)$ be the number of components created by the deletion of the edges in X in the subgraph G_X, and $\kappa_Y(Y)$ be similarly defined.

First, since b_0 is a breakpoint, we have $g_A(b_0) = g_{A\cup X}(b_0)$. This implies that $s(A) - b_0(\kappa(A) - 1) = s(A \cup X) - b_0(\kappa(A \cup X) - 1)$. Since $s(A \cup X) = s(A) + s(X)$ and $\kappa(A \cup X) = \kappa(A) + \kappa_X(X) - 1$, we have $s(X) - b_0(\kappa_X(X) - 1) = 0$. That is, $b_0 = \frac{s(X)}{\kappa_X(X)-1}$ is the strength of G_X.

By non-degeneracy, we must have the following at the breakpoint b_0:

$$\begin{aligned} g_A(b_0) = g_{A\cup X}(b_0) < g_{A\cup Y}(b_0) &\Rightarrow s(A \cup X) - b_0(\kappa(A \cup X) - 1) < s(A \cup Y) - b_0(\kappa(A \cup Y) - 1) \\ &\Rightarrow s(A) + s(X) - b_0(\kappa(A) - 1 + \kappa_X(X) - 1) < s(A) + s(Y) - b_0(\kappa(A) - 1 + \kappa_Y(Y) - 1) \\ &\Rightarrow s(X) - b_0(\kappa_X(X) - 1) < s(Y) - b_0(\kappa_Y(Y) - 1). \end{aligned} \tag{4}$$

Also observe that since Y is the component with minimum strength, we have:

$$\alpha = \frac{s(Y)}{\kappa_Y(Y) - 1} \leqslant \frac{s(X)}{\kappa_X(X) - 1} = b_0,$$

where we observed above that the strength of G_X is exactly b_0. Since $0 < \alpha \leqslant b_0$, this gives $s(Y) - b_0(\kappa_Y(Y) - 1) \leqslant s(X) - b_0(\kappa_X(X) - 1)$. This contradicts Eq. (4). Therefore, the sets X and Y cannot be distinct, and the lemma stands proved. □

4. Algorithm

The results in the preceding section enable us to solve the Lagrangean relaxation easily. Using Lemma 3.2, all we need to do is identify the breakpoints of $g'(b)$, which we know are no more than $|V| - 1$ in number. Theorem 3.6 implies that the next breakpoint can be identified by solving the strength problem on each remaining connected component, and selecting the minimum-strength component among these. Since we already have Cunningham's polynomial time algorithm for computing the strength of a graph, it follows that the Lagrangean relaxation can be solved exactly in polynomial time; in fact, we can compute the function $g'(b)$ exactly in polynomial time.

If the solution to the Lagrangean relaxation yields a k-cut, then we have found an optimal solution and are done. If not, we look at the last set of edges selected for deletion, and select a subset of them which yields a k-cut. The details of this selection, as well as the complete algorithm, are described in Fig. 1. Here, A_{i-1} is the set of edges found by the algorithm such that $\kappa(A_{i-1}) < k$, and C_i is the minimum-strength component of $G(V, E \setminus A_{i-1})$. The strength minimizer of C_i creates more components than needed, and hence we select enough edges from the strength minimizer of C_i so that adding them to A_{i-1} results in a k-cut.

4.1. Analysis

Let A^* be an optimal k-cut. We use the solution to the Lagrangean relaxation to derive a simple lower bound for $s(A^*)$. Define $\underline{s}(G, k)$ as follows, where A_{i-1} and A_i are as defined in Algorithm LkC and $\kappa(A_{i-1}) < k < \kappa(A_i)$.

Algorithm LkC

1. Initialize $A_0 = \emptyset, B_0 = \emptyset, i = 0$.
2. While $\kappa(A_i) < k$, do:
3. $\quad C_{i+1}$:= minimum-strength connected component of $G_i = (V, E \setminus A_i)$.
4. $\quad B_{i+1}$:= edges deleted in strength minimizer of C_i.
5. $\quad A_{i+1} := A_i \cup B_i$.
6. $\quad i := i + 1$.
7. If $\kappa(A_i) = k$, output A_i and halt.
8. For each connected component S_j in $C_i \setminus B_i$, define shore$(S_j) := \{e = (u, v) \in B_i : u \in S_j, v \notin S_j\}$.
9. Let B be the union of $k - \kappa(A_{i-1})$ cheapest (w.r.t. s) shores.
10. Output $A_{i-1} \cup B$ and halt.

Fig. 1. Algorithm description.

$$\underline{s}(G,k) = \left(\frac{\kappa(A_i) - k}{\kappa(A_i) - \kappa(A_{i-1})}\right) s(A_{i-1}) + \left(\frac{k - \kappa(A_{i-1})}{\kappa(A_i) - \kappa(A_{i-1})}\right) s(A_i). \tag{5}$$

Lemma 4.1. *Let A_{i-1} and A_i be as defined in Algorithm* LkC, *where $\kappa(A_{i-1}) < k < \kappa(A_i)$. Then $s(A^*) \geqslant \underline{s}(G,k)$.*

Proof. Let $b = b_i$ be the point at which $g'(b)$ is maximized. Since the breakpoints of the function $g'(b)$ are found by successively finding minimum-strength components of the current graph (Theorem 3.6), each edge set B_{i+1} in Algorithm LkC is exactly the set of edges of the corresponding minimum-strength solution. Therefore, this breakpoint is induced by two edge sets A_{i-1} and A_i as defined in Algorithm LkC. We now have $s(A_i) - b_i(\kappa(A_i) - 1) + b_i(k-1) = s(A_{i-1}) - b_i(\kappa(A_i) - 1) + b_i(k-1)$. This yields the following expression:

$$b_i = \frac{s(A_i) - s(A_{i-1})}{\kappa(A_i) - \kappa(A_{i-1})}.$$

This expression can then be substituted in the inequality $s(A^*) \geqslant s(A_i) + b_i(k - \kappa(A_i))$ to yield the lower bound stated in the lemma. □

Theorem 4.2. *Algorithm* LkC *returns a solution which costs no more than $2s(A^*)$.*

Proof. If $\kappa(A_i) = k$, then A_i is an optimal solution for the k-cut problem instance, since otherwise we would have $g_{A^*}(b_i) < g_{A_i}(b_i)$, contradicting the optimality of A_i for $g'(b)$. We now bound the cost of the solution if $\kappa(A_{i-1}) < k < \kappa(A_i)$.

Let $\mathscr{S}_i$ be the set of connected components in $C_i \setminus B_i$. Then since each edge in B_i belongs to exactly two shores, we have:

$$\sum_{S_j \in \mathscr{S}_i} s(\text{shore}(S_j)) = 2s(B_i).$$

There are exactly $\kappa(A_i) - \kappa(A_{i-1})$ connected components in $\mathscr{S}_i$. Hence the cost of the solution output by the algorithm is no more than:

$$\begin{aligned} s(A_{i-1}) + 2\left(\frac{k - \kappa(A_{i-1})}{\kappa(A_i) - \kappa(A_{i-1})}\right) s(B_i) &= \left(1 - \frac{2(k - \kappa(A_{i-1}))}{\kappa(A_i) - \kappa(A_{i-1})}\right) s(A_{i-1}) + 2\left(\frac{k - \kappa(A_{i-1})}{\kappa(A_i) - \kappa(A_{i-1})}\right) s(A_i) \\ &\leqslant 2\left(\frac{\kappa(A_i) - k}{\kappa(A_i) - \kappa(A_{i-1})}\right) s(A_{i-1}) + 2\left(\frac{k - \kappa(A_{i-1})}{\kappa(A_i) - \kappa(A_{i-1})}\right) s(A_i) \leqslant 2\underline{s}(G,k) \\ &\leqslant 2s(A^*), \end{aligned}$$

where the last inequality follows from Lemma 4.1. □

5. Discussion and comparison with other algorithms

5.1. Comparison with the SV algorithm

Saran and Vazirani [10] proposed a $(2-\frac{2}{k})$-approximation algorithm for the k-cut problem, which proceeds by finding minimum cuts (of the traditional $s-t$ type) recursively until the graph is partitioned into k parts. We use SV to refer to their algorithm for this and the subsequent section, where we provide comparisons between our algorithm and SV. We show here that for each algorithm, there exists a family of instances where the performance ratio of the algorithm approaches its worst-case bound while the other algorithm finds an optimal solution, whose value matches the lower bound.

The family is generated by the graph $G_1(r,c)$ displayed in Fig. 2. The graphs are parametrized by r and c, and consist of a clique of r vertices of unit weight each, with a single edge of weight c connecting each vertex in the clique to a distinct vertex outside the clique. Therefore, the graphs have $2r$ vertices.

Proposition 5.1. *Consider the k-cut problem on the family of graphs $G_1(r, r-1-\epsilon)$ as $\epsilon \to 0$ and $r \to \infty$, with $k = r$. Then, the cost of the solution obtained by the SV algorithm is twice as much as the cost of the solution obtained by the* LkC *algorithm. Furthermore, the* LkC *solution cost is equal to the Lagrangean lower bound.*

Proof. In $G_1(r, r-1-\epsilon)$, the minimum cuts are obtained by deleting the "arms" of weight $r-1-\epsilon$. Therefore, when $k=r$, the SV algorithm finds the cut obtained by deleting $r-1$ of the arms, with a total cost of $s^{\mathrm{SV}} = (r-1)(r-1-\epsilon)$.

On the other hand, the strength is minimized by deleting all the unit-cost edges in the clique, with strength $r/2$. This can be observed by the fact that deleting any of the r-length edges increases the cost by r but the number of components by only 1, so will only worsen the strength. Hence the LkC algorithm deletes all the clique edges and returns the r arms as the connected components, resulting in a total cost of $s^{\mathrm{LkC}} = \frac{r(r-1)}{2}$. This is identical to the Lagrangean lower bound.

As $\epsilon \to 0$ and $r \to \infty$, the ratio $\frac{s^{\mathrm{SV}}}{s^{\mathrm{LkC}}} \to 2$. □

Proposition 5.2. *Consider the k-cut problem on the family of graphs $G_1(r, \frac{r}{2}+\epsilon)$ as $\epsilon \to 0$ and $r \to \infty$, with $k = 2$. Then, the cost of the solution obtained by the* LkC *algorithm is twice as much as the cost of the solution obtained by the SV algorithm. Furthermore, the SV solution cost is equal to the Lagrangean lower bound.*

Proof. In $G_1(r, \frac{r}{2}+\epsilon)$, the minimum cuts are unchanged from Proposition 5.1. They are obtained by deleting the arms. Since $k=2$, the optimal solution is a minimum cut, which is found by the SV algorithm and has cost $s^{\mathrm{SV}} = \frac{r}{2}+\epsilon$.

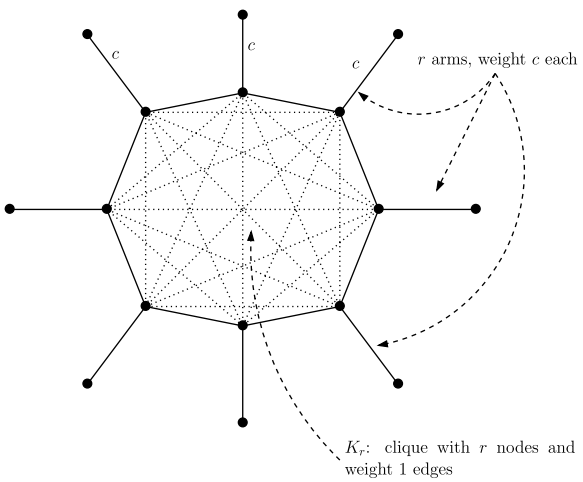

Fig. 2. $G_1(r,c)$: Graph demonstrating incomparability of LkC and SV algorithms in Propositions 5.1 and 5.2.

The minimum-strength component is also unchanged, and remains the clique. The optimal strength is $\frac{r}{2}$, which is close to s^{SV}. On the other hand, the LkC algorithm must pick one "shore" from the clique, and hence obtains a solution of cost $s^{\mathrm{LkC}} = r - 1$.

As $\epsilon \to 0$ and $r \to \infty$, we have $\frac{s^{\mathrm{LkC}}}{s^{\mathrm{SV}}} \to 2$. We also find that the ratio of the cost of the LkC solution to the lower bound approaches 2. □

Propositions 5.1 and 5.2 indicate that neither are the two algorithms identical nor can one dominate the other in any deterministic sense. This does not indicate any conclusion about their empirical performance. We explore that question in our computational experiments, provided in Section 6.

5.2. Comparison with the integer programming formulation

Naor and Rabani [8] introduced the following integer programming formulation for the k-cut problem. Assume that the input graph G is connected, by adding zero-weight edges if necessary. Let $\mathscr{T}$ be the set of all spanning trees of G. The formulation is based on the fact that a set of edges A forms a k-cut of G if and only if A includes at least $k-1$ edges of every spanning tree in $\mathscr{T}$. In the formulation below, the binary variables d_e are indicators, taking the value 1 to indicate the presence of the corresponding edges in A.

$$\min \quad \sum_{e \in E} s_e d_e \tag{6}$$

$$\sum_{e \in T} d_e \geqslant k-1 \quad \forall T \in \mathscr{T}, \tag{7}$$

$$d_e \in \{0,1\} \quad \forall e \in E. \tag{8}$$

We state the main result of Naor and Rabani without proof; the reader is referred to [8] for details. The linear relaxation of (6)–(8) is obtained by replacing (8) with the non-negativity constraint $d_e \geqslant 0\ \forall e \in E$, and the integrality gap of the IP formulation is defined as the worst-case ratio between the optimal solution of the IP to the optimal solution of its linear relaxation.

Theorem 5.3 (Naor and Rabani [8]). *The integrality gap of the formulation (6)–(8) is bounded above by 2. Furthermore, an integral solution can be obtained from the optimal fractional solution in polynomial time, thus providing a 2-approximation algorithm.*

The optimal fractional solution provides an alternative lower bound to the k-cut problem. We now compare this lower bound with our Lagrangean lower bound. A lower bound with a higher value would be more desirable, since the k-cut problem is a minimization problem.

Theorem 5.4. *Given a graph G and given k, let z^* denote the optimal solution to the linear relaxation of (6)–(8), and let $\underline{s}(G,k)$ be our Lagrangean lower bound defined in (5). Then $z^* \leqslant \underline{s}(G,k)$.*

Proof. Consider the behavior of Algorithm LkC on the given instance. If for some i we have $\kappa(A_i) = k$, then by Theorem 4.2 our algorithm has found the optimal integral solution, and this theorem is trivially proved.

Now suppose A_i and A_{i+1} are the edge sets at breakpoints b_i and b_{i+1} respectively, where $\kappa(A_i) < k < \kappa(A_{i+1})$. Define $k_i = \kappa(A_i)$ and $k_{i+1} = \kappa(A_{i+1})$. For every edge e and every i, define $d_e^i = 1$ if and only if $e \in A_i$, and 0 otherwise. Define $d_e = (\frac{k_{i+1}-k}{k_{i+1}-k_i}) d_e^i + (\frac{k-k_i}{k_{i+1}-k_i}) d_e^{i+1}$. We show that $\{d_e\}_{e \in E}$ satisfies the constraints (7).

Consider any spanning tree T of G. Since A_i is a k_i-cut, we have $\sum_{e \in T} d_e^i \geqslant k_i - 1$. Similarly, we have $\sum_{e \in T} d_e^{i+1} \geqslant k_{i+1} - 1$. Hence:

$$\sum_{e \in T} d_e = \sum\nolimits_{e \in T} \left[\left(\frac{k_{i+1}-k}{k_{i+1}-k_i} \right) d_e^i + \left(\frac{k-k_i}{k_{i+1}-k_i} \right) d_e^{i+1} \right] = \frac{k_{i+1}-k}{k_{i+1}-k_i} \sum_{e \in T} d_e^i + \frac{k-k_i}{k_{i+1}-k_i} \sum_{e \in T} d_e^{i+1}$$

$$\geqslant \frac{k_{i+1}-k}{k_{i+1}-k_i}(k_i - 1) + \frac{k-k_i}{k_{i+1}-k_i}(k_{i+1} - 1) = k - 1.$$

Hence $\{d_e\}_{e\in E}$ is a feasible solution to the linear relaxation of (6)–(8), of cost $\underline{s}(G,k) = \sum_{e\in E} s_e d_e$. The optimal fractional solution must have cost no greater than this feasible solution. Therefore, $z^* \leqslant \underline{s}(G,k)$. □

Consequently, for every instance of the k-cut problem, our Lagrangean relaxation is at least as good as the LP lower bound (which is the only other lower bound known in the literature).

5.3. Comparison of computational complexity

Cunningham's algorithm to compute the optimal attack of a network has running time $O(|V|^3|E|)$. Algorithm LkC calls this subroutine at most $k \leqslant |V|$ times, so the overall running time of our algorithm is $O(|V|^4|E|)$.

In contrast, the algorithm of Saran and Vazirani has running time $O(|V|^2|E|)$. Naor and Rabani do not provide implementation or running time details. A naïve implementation must use the ellipsoid method to solve the linear programming relaxation, with a minimum spanning tree constituting the separation oracle at each iteration of the ellipsoid method. This leads to a prohibitively high running time in practice for the Naor–Rabani algorithm.

5.4. Limitations of our approach

One limitation of our algorithm is that it only uses edges which minimize the attack values of the graph for different values of the parameter b. In particular, we have proved that the edge sets of successive breakpoints in Algorithm LkC are supersets of the preceding edge sets. If $k_i < k < k_{i+1}$ and A is the edge set of an optimal k-cut, one might ask the following question: Is it true that $A_i \subset A \subset A_{i+1}$? An affirmative answer would offer hope that there might be a better way to select the edges from those generated by our algorithm. However, we answer this question in the negative.

Proposition 5.5. *Any algorithm for k-cut which only uses edges belonging to minimum attack components cannot achieve an approximation factor better than* 3/2.

Proof. Consider an instance of the k-cut problem with $k = 3$, on the graph $G_2(r)$ displayed in Fig. 3. The graph consists of a cycle C_r of r vertices and unit-weight edges, along with three other edges e_a, e_b and e_c with weights $1+\epsilon$, $1+\frac{2}{r}+\epsilon$, and $1+\frac{2}{r}+\epsilon$ respectively, where $\epsilon \to 0$ and $r \to \infty$.

The optimal solution is found by deleting edges e_b and e_c and has cost $2(1+\frac{2}{r}+\epsilon)$.

Now consider the behavior of Algorithm LkC. The first breakpoint occurs at $b_1 = 1+\epsilon$, with $A_1 = \{e_a\}$ and $k_1 = 2$. The second breakpoint occurs at $b_2 = 1+\frac{1+\epsilon}{r}$, with $A_2 = \{e_a\} \cup C_r$ and $k_2 = r+1$. Hence our algorithm ends up selecting the edge e_a along with two edges from the cycle C_r, at a total cost of $3+\epsilon$. As $\epsilon \to 0$ and $r \to \infty$, the ratio of the cost of the solution obtained by Algorithm LkC and the optimal solution approaches 3/2.

We also observe that the edges of the optimal solution, $\{e_b, e_c\}$, do not belong to A_1 or A_2, completing the proposition. □

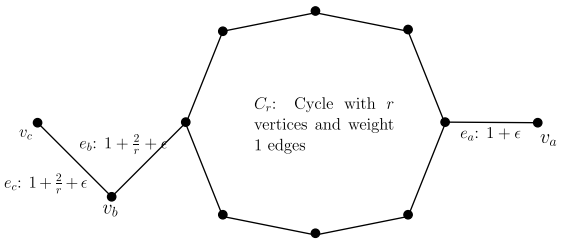

Fig. 3. $G_2(r)$: Graph demonstrating limitations of using edges which minimize the attack values, as in Proposition 5.5.

6. Computational results

The algorithm was implemented in preliminary and experimental form using the Java programming language. In addition to our algorithm, we also implemented the *Efficient* algorithm of Saran and Vazirani [10], referred to in the sequel as SV, in order to provide a comparison of the two algorithms.[1] The implementation is available on the web [11] for replication, experimentation and further extensions.

Our experiments were conducted on random graphs generated by two distinct models:

1. *Bernoulli random graphs*: These graphs are characterized by the number of nodes n, and an edge probability p. Each of the $\binom{n}{2}$ pairs of vertices in the graph are connected by an edge with probability p, independently of all other vertex-pairs. If an edge exists, its weight is an integer chosen uniformly at random between 1 and 100, again independently of all other edges. The study of these graphs was spurred by Erdös and Renyi [5], and they have been the most popular model of random graphs.
2. *Preferential attachment (PA) random graphs*: These graphs are characterized by the number of nodes n, and an average degree $2d$. They are created by the following evolutionary process: nodes are added one at a time, and each time a node is added, it is linked to d of the pre-existing nodes chosen at random. (If d is greater than the number of pre-existing nodes, then the new node is connected by a single edge to each of the pre-existing nodes.) As in the Bernoulli graphs, we assign each edge a weight which is a uniformly chosen integer between 1 and 100, independently of all other edges. Barabasi and Albert [1] proposed this graph model to explain various structural properties of real-world graphs like the web, social networks, foreign trade networks, etc.

A representative output of our algorithm is shown in Fig. 4. This particular instance was a 30-node graph with average degree 12 (that is, during creation of the graph, each node connected to 6 preceding nodes). As can be seen, both algorithms perform extremely well for small values of k, often finding the optimal solution. For larger values of k, both algorithms find near-optimal solutions, though not necessarily the same ones. In all our experiments, there seemed to be no obvious pattern of either algorithm dominating the other. The worst-case examples for the algorithms shown in Section 5, while indicating that in theory the performance of the two algorithms could widely diverge, seem unlikely to occur in random instances.

The observation that the algorithms perform better for lower values of k was also found in all our trials, though the quality of the approximation worsened as the graphs became denser. All our trials resulted in graphs which look similar to Fig. 4, and there seemed to be no distinction between the two types of graphs either in terms of one algorithm out-performing the other.

Further computational results are provided in Fig. 5. Once again, we ran our experiments for pre-selected values of the graph parameters and k, and report the results. While it may seem that the LkC algorithm finds a better solution more often than the Saran–Vazirani algorithm in the figure, this seems to be an accident of the results we chose to report rather than any observable dominance in our complete set of experiments.

One might also observe that for low values of k in PA random graphs which are not dense, both algorithms find the optimal solution. Indeed, this should not be a surprise since by the nature of these graphs, they have several vertices with low degree which are obvious candidates for participating in the k-cuts as singleton components.

Finally, we observe that for graphs with low density, the quality of the lower bound seems to be pretty good, since both algorithms find solutions which are usually not very far from optimal. The performance degrades as the graphs get denser. This empirical observation, combined with the discussion in Section 5 which shows examples where both algorithms achieve their worst case, seems to indicate that the Lagrangean lower bound performs poorly in dense graphs.

The running times corresponding to the graphs tested in Fig. 5 are shown in Fig. 6. The times shown are in seconds, averaged over 5 runs of each type of graph. Recall that once the function $g'(b)$ has been computed,

[1] At the time of writing this paper, there is no known implementation of the algorithm of Naor and Rabani [8]; hence the computational experiments do not compare with their algorithm.

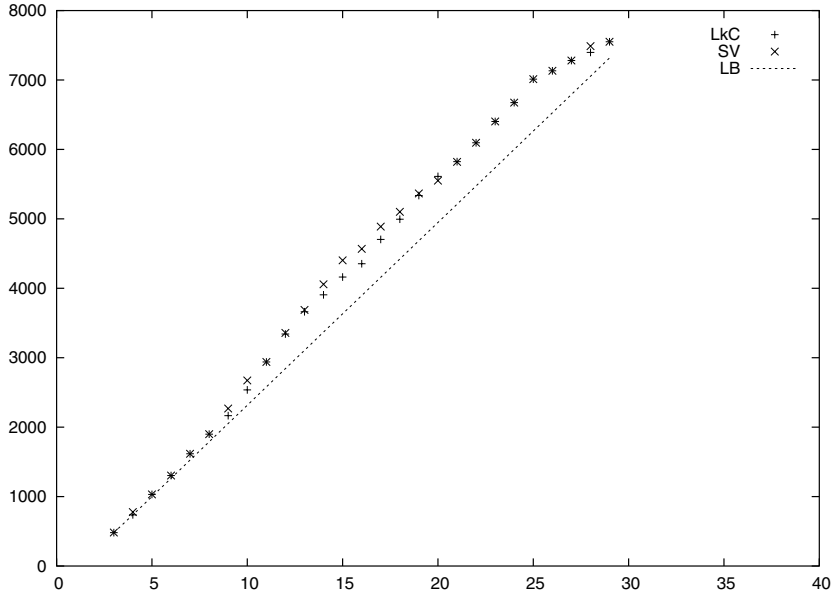

Fig. 4. Output for a PA random graph with $n = 30$ vertices and average degree $2d = 12$. LB refers to the lower bound, LkC to the solution found by Algorithm LkC, and SV to the solution found by the Saran–Vazirani algorithm. The value of k is measured on the x-axis, while the value of the solutions and lower bound are on the y-axis.

Bernoulli graphs					PA random graphs				
Parameters	k	LB	LkC	SV	Parameters	k	LB	LkC	SV
$n = 20$	3	101.500	1.103	1.103	$n = 20$	3	82.000	1.000	1.000
$p = 0.2$	5	250.333	1.054*	1.202	$d = 2$	5	203.000	1.000*	1.064
	10	660.000	1.000*	1.109		10	614.000	1.000*	1.079
$n = 30$	3	201.000	1.000*	1.049	$n = 30$	3	221.000	1.000	1.000
$p = 0.2$	6	506.000	1.000*	1.031	$d = 3$	6	618.000	1.000*	1.067
	15	1676.272	1.220	1.143*		15	1936.764	1.020*	1.116
$n = 50$	3	342.000	1.055	1.055	$n = 50$	3	254.000	1.000	1.000
$p = 0.2$	10	1826.341	1.012*	1.026	$d = 5$	10	1690.000	1.000*	1.015
	25	5491.463	1.120	1.120		25	5233.000	1.019*	1.075
$n = 30$	3	5.000	1.000	1.000	$n = 30$	3	481.000	1.000*	1.004
$p = 0.1$	6	72.000	1.000	1.000	$d = 6$	6	1263.384	1.030	1.030
	15	525.000	1.000	1.000		15	3632.115	1.145*	1.212
$n = 30$	3	605.428	1.025	1.025	$n = 30$	3	1138.344	1.154	1.154
$p = 0.5$	6	1689.714	1.222*	1.230	$d = 15$	6	2845.862	1.208	1.208
	15	4942.571	1.218	1.218		15	7968.413	1.339	1.339
$n = 30$	3	1339.862	1.473	1.473	$n = 30$	3	1426.965	1.559	1.559
$p = 0.9$	6	3349.655	1.510	1.510	$d = 27$	6	3567.413	1.554	1.554
	15	9379.034	1.374	1.374		15	9988.758	1.411	1.411

Fig. 5. Typical computational results. LB indicates the value of the lower bound, LkC the ratio between the LkC solution and the lower bound, and SV the ratio between the Saran–Vazirani algorithm and the lower bound. An asterisk indicates that one algorithm was strictly better than the other for that input.

the LkC algorithm is able to find k-cut solutions for any value of k. Therefore, the results shown here correspond to finding k-cut solutions for every value of k between 2 and $n - 1$, as also shown in the instance in Fig. 4. We also note that the running times shown here are only for the purpose of comparison between the two algorithms. The implementation was coded in Java, and the experiments were run on an Intel Pentium II machine running Linux. A more efficient implementation, in a programming language such as C/C++, is likely to result in running times which are one or more orders of magnitude better than the running times shown here. The only visible trend in the running times seem to be that for sparse graphs the Saran–Vazirani

Graph parameters	LkC, seconds	SV, seconds	LkC/SV
Bernoulli, $n = 20,\ p = 0.2$	3.06	0.20	15.66
Bernoulli, $n = 30,\ p = 0.2$	30.60	3.22	9.49
Bernoulli, $n = 50,\ p = 0.2$	518.95	47.81	10.85
Bernoulli, $n = 30,\ p = 0.1$	11.12	1.61	6.92
Bernoulli, $n = 30,\ p = 0.5$	14.22	10.18	1.40
Bernoulli, $n = 30,\ p = 0.9$	68.67	47.64	1.44
PA, $n = 20,\ d = 2$	2.68	0.18	15.15
PA, $n = 30,\ d = 3$	29.74	3.12	9.53
PA, $n = 50,\ d = 5$	921.65	45.80	20.12
PA, $n = 30,\ d = 6$	62.41	6.35	9.83
PA, $n = 30,\ d = 15$	79.16	28.19	2.81
PA, $n = 30,\ d = 27$	68.06	60.21	1.13

Fig. 6. Comparison of running times on test instances. The running times displayed are an average of 5 runs of the same types of graphs shown in Fig. 5.

efficient algorithm is much faster than our algorithm, but for denser graphs the running times are closer to each other.

7. Conclusion

The Lagrangean relaxation of the k-cut problem yields a good lower bound, and also leads to a 2-approximation algorithm. We also provide comparisons with other algorithms and lower bounds in the literature, demonstrating the competitiveness of our approach. The efficacy of the algorithm is further validated using tests on random graphs.

The gap between the best known approximation ratio $(2 - \frac{2}{k})$ and the best hardness result (NP-completeness) remains open. In particular, it is unknown whether or not the k-cut problem admits a polynomial time approximation scheme.

Acknowledgments

This work was done while Amitabh Sinha was a graduate student at the Tepper School of Business at Carnegie Mellon University. R. Ravi was supported by NSF Grant CCR-1120168 and Amitabh Sinha was supported by a William Larimer Mellon Fellowship.

A preliminary version [9] of this paper appeared in the proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms. We thank the reviewers of the conference and several other readers and reviewers for their comments. We also thank an anonymous reader for pointing out the work of Barahona [2]. Finally, we thank an anonymous referee for pointing out a simpler proof for Theorem 3.4, which appears in this paper.

References

[1] A. Barabasi, R. Albert, Emergence of scaling in random networks, Science 286 (1999) 509–512.
[2] F. Barahona, On the k-cut problem, Operations Research Letters 26 (2000) 99–105.
[3] C. Chekuri, S. Guha, J. Naor, Approximating Steiner k-cuts, in: Proceedings of the 30th International Colloquium on Automata, Languages and Computation, 2003, pp. 189–199.
[4] W. Cunningham, Optimal attack and reinforcement of a network, Journal of the ACM 32 (3) (1985) 549–561.
[5] P. Erdös, A. Renyi, On random graphs, Publicationes Mathematicae 6 (1959) 290–297.
[6] O. Goldschmidt, D. Hochbaum, Polynomial algorithm for the k-cut problem, in: Proceedings of the 29th Annual IEEE Symposium on the Foundations of Computer Science, 1988, pp. 444–451.
[7] D. Karger, C. Stein, A new approach to the minimum cut problem, Journal of the ACM 43 (4) (1996) 601–640.

[8] J. Naor, Y. Rabani, Tree packing and approximating k-cuts, in: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 26–27.
[9] R. Ravi, A. Sinha, Approximating k-cuts via network strength, in: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 621–622.
[10] H. Saran, V. Vazirani, Finding k-cuts within twice the optimal, SIAM Journal of Computing 24 (1) (1995) 101–108.
[11] A. Sinha, Approximating k-cuts using network strength as a Lagrangean relaxation: A Java implementation (2004). Available from: <http://www.umich.edu/~amitabh/kcut>.