On cutting a few vertices from a graph*

Uriel Feige Robert Krauthgamer Kobbi Nissim
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science
Rehovot 76100, Israel
{feige,robi,kobbi}@wisdom.weizmann.ac.il

February 28, 2002

Abstract

We consider the problem of finding in an undirected graph a minimum cut that separates exactly a given number k of vertices. For general k (i.e. k is part of the input and may depend on n) this problem is NP-hard.

We present for this problem a randomized approximation algorithm, which is useful when k is relatively small. In particular, for $k = O(\log n)$ we obtain a PTAS (polynomial time approximation scheme), and for $k = \Omega(\log n)$ we obtain an approximation ratio $O(k/\log n)$.

1 Introduction

Let $G(V, E)$ be an undirected graph on n vertices, and assume that each edge has a nonnegative *cost*. For a subset S of the vertices, let $(S, V \setminus S)$ denote, as usual, the *cut* of G that consists of the edges with exactly one endpoint in S. The cost of a cut is the sum of capacities of its edges. In the case where

*This is a preprint of an article accepted for publication in Discrete Applied Mathematics© (copyright owned as specified in the journal).

all edges have a unit cost, the cost of a cut is simply the number of edges in it. A cut $(S, V \setminus S)$ with $|S| = k$ is called a $(k, n-k)$ *cut* of G. Let b_k denote the minimum cost of a $(k, n-k)$ cut in G.

In the *minimum* $(k, n-k)$ *cut problem*, we are given the graph G with the edge capacities and a number $k \in \{1, \ldots, n-1\}$, and we wish to compute b_k, i.e. the minimum cost of a $(k, n-k)$ cut. The minimum $(k, n-k)$ cut problem, i.e. computing b_k for general k, is NP-hard. Indeed, Garey, Johnson, and Stockmeyer [GJS76] show that it is NP-hard to compute b_k in the special case $k = n/2$, known as the *minimum bisection* problem, even in unit cost graphs. It is straightforward to see that the proof of Bui and Jones [BJ92] for the NP-hardness of finding edge separators, actually shows that it is NP-hard to compute b_k in graphs of maximum degree 3 and for $k = \alpha n$ (and even $k = n^\alpha$) for any fixed $0 < \alpha < 1$.

It is not known whether b_k is polynomial time computable when k is a slowly growing function of n, say $k = \log n$. Note that a straightforward exhaustive search on all vertex subsets of size k can find b_k in time $n^{k+\Theta(1)}$, which is polynomial only if $k = O(1)$, i.e. a fixed constant independent of n.

We address the problem of approximating b_k when k is relatively small compared to n. An algorithm is said to approximate a minimization problem within ratio $r \geq 1$ if it runs in polynomial time and outputs a solution whose value is at most r times the value of the optimal solution. The problem is said to have a PTAS (polynomial time approximation scheme) if for every fixed $r > 1$ it has an approximation algorithm with approximation ratio r.

Related work. For general k, the current authors presented in the extended abstract [FKN00] the first sublinear (in n) approximation ratio for the minimum $(k, n-k)$ cut problem, giving an $O(\sqrt{n} \log n)$ approximation ratio. Feige and Krauthgamer [FK00] improved over this approximation ratio by devising an algorithm that achieves an $O(\log^2 n)$ approximation ratio for the minimum $(k, n-k)$ cut problem. However, for the case of a relatively small k the result of [FK00] does not improve over an algorithm that we briefly described in Section 5 of [FKN00]. The current paper is the full version of our approximation ratio for relatively small k from [FKN00], which is currently the best known approximation ratio for this case.

There is no known hardness of approximation result that excludes the possibility of a PTAS for the minimum $(k, n-k)$ cut problem. For additional related work (e.g. results for restricted graph families and related problems) we refer the reader to [FKN00] and [FK00].

Our results. We present a simple randomized algorithm that is aimed towards approximating the minimum $(k, n-k)$ cut problem when k is relatively small. The algorithm and its analysis are described in Section 2, where we prove the following theorem.

Theorem 1 *For every fixed $\epsilon > 0$, there is a polynomial time randomized algorithm that finds, with high probability, a $(k, n-k)$ cut whose cost is at most $(1 + \epsilon k/\ln n)b_k$.*

The above approximation ratio implies, in particular, the following results for $k = O(\log n)$ and for $k = \Omega(\log n)$.

Corollary 1 *For any $k = O(\log n)$, there is a PTAS for the minimum $(k, n-k)$ cut problem.*

Corollary 2 *For any $k = \Omega(\log n)$, the minimum $(k, n-k)$ cut problem can be approximated within a ratio of $O(k/\log n)$.*

Note that Corollary 2 should be used only when k is slightly larger than $O(\log n)$, while for larger k the approximation ratio of [FK00] is preferable.

In Section 3 we discuss how our approximation algorithm extends to vertex weights and to $s - t$ cuts.

Techniques. Our algorithm utilizes random edge contraction and dynamic programming. Random edge contraction was introduced by Karger and Stein [KS96] to devise efficient algorithms for the minimum cut problem. Each iteration of their algorithm selects an edge at random and merges its endpoints, so as to form clusters of vertices. If no edge of a fixed minimum cut $(S, V \setminus S)$ is ever contracted, then every cluster is contained entirely either in S or in $V \setminus S$. When only two clusters remain, they correspond to the fixed minimum cut. It can be shown that there is a noticeable probability that no edge of the fixed minimum cut is ever contracted, and then the algorithm succeeds.

Our algorithm also applies random edge contractions iteratively, but instead of requiring that only two clusters remain, we stop at an earlier point, in which we are guaranteed that dynamic programming will find a nearly minimum $(k, n-k)$ cut. The algorithm actually does not know the "right" stopping point, and therefore tries all possible stopping points (taking the best solutions).

2 The algorithm

Our algorithm for finding a $(k, n-k)$ cut (of nearly minimum cost) uses the random edge contraction technique of Karger and Stein [KS96]. It consists of repeating the following algorithm Contract sufficiently many times in order to amplify its success probability.

Algorithm Contract works in iterations, where each iteration consists of (i) a random edge contraction stage followed by (ii) a combining stage that computes a cut of the graph that corresponds to a $(k, n-k)$ cut of the input graph. (Both stages are described below). The algorithm proceeds with the iterations until there are no edges in the graph (to contract) and then it outputs a cut of minimum cost among all $(k, n-k)$ cuts found throughout the iterations (if any).

Let us now describe in more detail the two stages that form an iteration of algorithm Contract. A schematic description of the algorithm appears in Figure 1.

In the *contraction stage* we choose an edge at random and contract it, as follows. The probability of choosing an edge is proportional to its cost, so if all edges have a unit cost then an edge is selected uniformly at random. To contract the chosen edge, we merge its two endpoints. If as a result there are several edges between some pairs of (newly formed) vertices (i.e. parallel edges), we retain them all. Edges between vertices that were merged are removed, so that there are never any self-loops.

We refer to the vertices of the formed graph as *clusters*. Each cluster is a set of vertices (of the input graph) merged together. Note that the edges inside a cluster are removed from the graph. The *size* of a cluster is the number of vertices in it, and its *degree* is the cost of edges leaving the cluster.

In the *combining stage* we find in the graph (of the current iteration) a set of clusters whose total size is exactly k, and for which the sum of cluster degrees is minimal. (For example, if all clusters are of size 1, i.e. contain a single vertex, then we actually take the k vertices of smallest degree in the graph.) Note that any subset of clusters corresponds to cut (of the input graph) whose cost is no more than the sum of degrees of these clusters. In particular, the subset of clusters that we find is of total size k and thus corresponds to a $(k, n-k)$ cut of the input graph.

The combining stage can be implemented in polynomial time using dynamic programming (note that k is polynomially bounded), as follows. Label the clusters by $1, 2, \ldots, L$ in an arbitrary way, and define a dynamic programming table T. Each table entry $T(i, k')$ (with $1 \leq i \leq L$ and $1 \leq k' \leq k$) describes the minimal sum of cluster degrees, over all subsets Q of the clusters $1, 2, \ldots, i$ for which the total size (of Q) is exactly k'. It is straightforward that each entry $T(i, k')$ can be easily computed from entries of the form $T(i-1, \cdot)$, and that the entries $T(1, \cdot)$ are easy to initialize. The size of the table is $L \cdot k = O(n^2)$, so $T(L, k)$, which is the desired output of the combining stage, can be computed in polynomial time.

Algorithm CONTRACT.
Input: Graph on n vertices and a number k.
Output: $(k, n-k)$ cut in the graph.

1. While the graph contains edges, do
   - 1.1. edge contraction stage;
   - 1.2. combining stage to find a $(k, n-k)$ cut.
2. Output the cut of minimum cost among the cuts found in step 1.2 (if any).

Figure 1: Algorithm for finding a $(k, n-k)$ cut

Lemma 3 *The running time of algorithm* CONTRACT *is polynomial in n.*

Proof: Each edge contraction decreases the number of vertices by 1, and thus the number of iterations is bounded by n. Each iteration takes a polynomial time and the proof follows. $\square$

We analyze the success probability of the algorithm based on the following desired scenario. Suppose that the edges chosen to be contracted do not belong to a fixed optimum cut $(S, V \setminus S)$, i.e. these edges are either inside S or inside $V \setminus S$, until at some point the edges inside S (that remain in the graph) have a small cost relative to the cost of the optimum cut. At this point, it can be seen that the combining stage must find a $(k, n-k)$ cut (of the input graph) whose cost is nearly optimal.

The next lemma lower bounds the probability that the algorithm outputs a $(k, n-k)$ cut of relatively small cost. The parameter $\rho > 0$ in the lemma can have an arbitrary value, but we later set $\rho = \epsilon / \ln n$ for a fixed $\epsilon > 0$.

Lemma 4 *For every (not necessarily fixed) $\rho > 0$, algorithm* Contract *outputs a $(k, n-k)$ cut whose cost is at most $(1+\rho k)b_k$ with probability at least $e^{-2/\rho}$.*

Proof: For the analysis, fix one cut $(S, V \setminus S)$ with $|S| = k$ whose cost b_k is minimum. Note that algorithm Contract is not aware of this cut.

Consider a run of the algorithm, and let A_t (for $0 < t < n$) be the event that the graph G_t resulting from the first t contractions satisfies the following two conditions:

(a) The total cost of edges of G_t with both endpoints in S is at most $\rho k b_k / 2$. (Note that the number of such edges may be reduced at each iteration, since edges that become self-loops are removed.)

(b) No cluster of G_t contains vertices both from S and from $V \setminus S$.

Condition (b) is equivalent to:

(b') None of the first t contracted edges belongs to the optimum cut $(S, V \setminus S)$.

We claim that if the event $A = \cup_t A_t$ happens then the algorithm succeeds, i.e. finds a $(k, n-k)$ cut of cost at most $(1+\rho k)b_k$. Indeed, assume that the event A_t happens and consider the combining stage of iteration t, which is performed on G_t. From (b) we have that every cluster in G_t is either a subset of S or a subset of $V \setminus S$. Therefore, the clusters contained in S have together all the vertices of S, and thus their total weight is k. From (a) it follows that the sum of degrees of these clusters (in G_t) is at most $b_k + 2(\rho k b_k/2) = b_k(1+\rho k)$. The combining stage of iteration t will therefore find a set of clusters of total weight k and whose sum of cluster degrees is no larger, which gives a $(k, n-k)$ cut (of the input graph), with cost at most $b_k(1+\rho k)$.

We next lower bound the probability of the event A. Let us say that an iteration is *successful* if the edge chosen to be contracted is inside S, a *ruin* if it is from the optimum cut $(S, V \setminus S)$, and *void* if it is inside $V \setminus S$. By (a) and (b'), the event A is equivalent to saying that the cost of edges inside S reduces to $\rho k b_k/2$ or less before any ruin iteration occurs. In this sense, the event A is affected by the successful and ruin iterations, but not by the void iterations. In other words, we need to compute the probability that an iterations is successful conditioned on the iteration not being void. As long as the cost of edges inside S, denoted $|E_S|$, is more than $\rho k b_k/2$, the conditioned probability for a successful iteration is

$$\frac{|E_S|}{|E_S|+b_k} = \left(1+\frac{b_k}{|E_S|}\right)^{-1} \geq \left(1+\frac{1}{\rho k/2}\right)^{-1}.$$

For the event A to happen we need that the first $k-1$ or less iterations that are not void will all be successful, and thus

$$\mathbf{Pr}[A] \geq \left(1+\frac{1}{\rho k/2}\right)^{-(k-1)} > e^{-\frac{2(k-1)}{\rho k}} > e^{-2/\rho}.$$

The probability that the algorithm outputs a $(k, n-k)$ cut of cost at most $b_k(1+\rho k)$ is at least $\mathbf{Pr}[A] > e^{-2/\rho}$, as claimed. $\square$

The next corollary follows from Lemma 4 by taking $\rho = \epsilon/\ln n$ for a fixed $\epsilon > 0$.

Corollary 5 *For every fixed $\epsilon > 0$, with probability at least $n^{-2/\epsilon}$, algorithm* Contract *outputs a $(k, n-k)$ cut whose cost is at most $(1+\epsilon k/\ln n)b_k$.*

We can amplify the above success probability by repeating algorithm Contract polynomially many (roughly $n^{2/\epsilon}$) times and taking from all the repetitions the cut of minimum cost. We then obtain Theorem 1.

3 Extensions

$s-t$ cuts. Suppose that the graph contains two special vertices s, t that must be separated, i.e. we wish to find a minimum cost cut $(S, V \setminus S)$ with $|S| = k$, $s \in S$ and $t \in V \setminus S$.

Unlike the minimum cut algorithm of Karger and Stein [KS96] that does not extend to $s-t$ cuts (see e.g. [MR95, Problem 1.8]), our approximation ratio does extend to this $s-t$ cut variant of the problem. The proof follows by modifying the combining stage to consider only clusters that do not contain t and such that at least one of them contains s.

Vertex weights. Suppose that the vertices of G have nonnegative integer weights. A *w-cut* cuts away vertices of total weight w, i.e. it is a cut $(S, V \setminus S)$ for which the sum of weights of S is w. Let b_w be the minimum cost of a w-cut.

We consider the problem of finding a nearly optimal w-cut, i.e. whose cost approximates b_w. We assume that the vertex weights are bounded by a polynomial in n, since for exponential vertex weights it is NP-hard to decide whether G contains a w-cut (as this is simply the subset-sum problem).

Let $b_{w,k}$ be the minimum cost of a cut that cuts away $k \in \{1, \ldots, n-1\}$ vertices of total weight w (and ∞ if no such cut exists). Modifying the combining stage to find a w-cut (using dynamic programming), it is straightforward to extend the proof of Lemma 4 and show that if $b_{w,k}$ is finite then with probability at least $e^{-2/\rho}$ algorithm CONTRACT finds a cut of cost at most $(1+\rho k)b_{w,k}$. By taking sufficiently many repetitions with $\rho = \epsilon / \log n$ for a fixed $\epsilon > 0$, we conclude that one can find in polynomial time a w-cut whose cost is at most $\min_k\{(1 + \epsilon k/\log n)b_{w,k}\}$ with high probability. Note that the minimum in the latter bound is not necessarily obtained at a value of k for which $b_{w,k} = b_w$.

Acknowledgements

The first author is the Incumbent of the Joseph and Celia Reskin Career Development Chair. The work of the second author was supported in part by a Dora Ostre memorial scholarship. The work of the third author was supported in part by the Joseph Brainin and Sally Brainin scholarship, the Inez P. and David N. Myers scholarship and the Harry and Mae Schetzen and Prof. Martin Schetzen scholarship. This research was supported in part by a Minerva grant.

References

[BJ92] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Inform. Process. Lett.*, 42(3):153–159, 1992.

[FK00] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *41st Annual IEEE Symposium on Foundations of Computer Science*, pages 105–115, November 2000.

[FKN00] U. Feige, R. Krauthgamer, and K. Nissim. Approximating the minimum bisection size. In *32nd Annual ACM Symposium on the Theory of Computing*, pages 530–536, May 2000.

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1(3):237–267, 1976.

[KS96] D. R. Karger and C. Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.