Approximating the minimum bisection size

(Extended Abstract)*

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

February 22, 2000

Abstract

A bisection of a graph with n vertices is a partition of its vertices into two sets, each of size $n/2$. The bisection size is the number of edges connecting the two sets. Finding the bisection of minimum size is NP-hard. We present an algorithm that finds a bisection that is within $O(\sqrt{n}\log n)$ of optimal. No sublinear approximation ratio for bisection was previously known.

1 Introduction

Let $G(V,E)$ be a graph with n vertices and m edges, where n is even. A bisection of G is a set of vertices $S \subset V$ with cardinality $|S| = n/2$. The size of the bisection S is the number of edges connecting S to its complement $V \setminus S$. The minimum size of the bisection of a graph is denoted by b. Computing b is NP-hard, cf. [8, 6]. We address the problem of approximating b.

An algorithm is said to approximate a minimization problem within ratio f if it runs in polynomial time and outputs a solution whose cost is at most f times the cost of the optimal solution. The problem is said to have a PTAS (polynomial time approximation scheme) if for every fixed $f > 1$ it has an approximation algorithm with approximation ratio f.

Our main motivation for approximating b is to better understand approximation algorithms in general. Finding the minimum bisection stands out as a well known problem for which there are no published approximation algorithms with sublinear (in n) approximation ratios, nor are there any hardness of approximation results that exclude the possibility that it has a PTAS.

*This is a preprint of a paper that will appear in STOC 2000

1.1 Where is the problem?

Recall that the minimum cut problem, partitioning the graph into two sets of vertices with the minimum number of edges connecting them, can be solved in polynomial time (e.g., by using the min-cut max-flow duality). Bisection is a similar problem, but with cardinality constraints on the two sets. Why do we find it difficult to approximate it?

A natural approach to approximate bisection is to first find a "good" cut (that cuts relatively few edges, say c) that is nearly balanced: one part contains $(n/2)+k$ vertices, the other $(n/2)-k$, for some small k. Thereafter this good cut can be fixed to be a bisection by transfering the k vertices of lowest degree from the big side to the small side. It appears that if both c and k are "small", we may obtain a good approximation ratio.

This approach does not guarantee an approximation ratio sublinear (in n), as illustrated in the following example. Let the input graph consist of 4 cliques whose sizes are $n/4, n/4+1, n/4-1$,n/4 where n is a multiple of 4. The first two cliques are connected to each other by one edge, and no other edges connect the cliques. The first step might result with a nearly balanced cut that cuts no edges, where the first two cliques are on one side of the cut, and the last two cliques are on the other side. The second step would then move one vertex to fix the imbalance, and result with a bisection of size $n/4-1$, whereas the optimal bisection contains only one edge.

1.2 Previous work

When b is a constant, the optimal bisection can be found in polynomial time, by guessing the bisection edges, removing them, partitioning the graph into connected components, and recombining the parts using dynamic programming. It is not known whether the optimal bisection can be found in polynomial time when the bisection size may be a slowly growing function of n (such as $\log n$).

Picking an arbitrary set S of $n/2$ vertices gives a bisection of size at most m, and hence approximation ratio at most m/b. This is a very poor approximation ratio when b is very small, especially when b is smaller than the average degree in G. Interestingly, Boppana shows that for most graphs that have bisections significantly smaller than $m/2$, the minimum bisection is unique and can be found exactly [5]. However, his algorithm does not appear to offer a good approximation ratio for graphs on which it fails to find the optimal bisection.

In [12], Saran and Vazirani give an algorithm that approximates bisection within a ratio of $n/2$, and further show that their particular algorithm does not approximate bisection within a better ratio. In [2], Arora, Karger and Karpinski show that bisection has a PTAS on everywhere-dense graphs (graphs with minimum degree $\Omega(n)$).

Bisection is a problem motivated by divide-and-conquer approaches for solving other graph problems, cf. [4]. By finding a small bisection, a graph can be partitioned into two equal size parts with very little interaction between the parts. Then a solution for the graph problem can be found in each part, and they can be combined to one solution at low additional cost. Continuing this approach recursively, there are only $\log n$ levels of recursion until the graph is broken into individual vertices.

For the divide-and-conquer paradigm, exact bisections are not necessary. Leighton and Rao [11] showed how to approximately find *minimum-quotient separators*, which we shall call *minimum-ratio cuts.* In these cuts, one wishes to cut the graph into two (not necessarily equal) parts, while minimizing the ratio $c/|S|$, where c is the number of edges cut, and $|S|$ is the cardinality of the smaller of the two vertex sets. (This ratio is also called *edge expansion* and *flux*).

Leighton and Rao showed (using weak duality with multicommodity flow problems) that this problem can be approximated within a ratio of $O(\log n)$. Minimum-ratio cuts are essentially as effective as bisections for divide-and-conquer algorithms. In particular, they can be used to find "balanced separators", partitioning the graph into parts no larger than $2n/3$ while cutting at most $O(b \log n)$ edges, cf. [10, Section 3] and [13], and the references therein.

Using minimum ratio cuts together with the approach outlined in Section 1.1, one can approximate bisection within a ratio of $O(\sqrt{m/b}\log n)$ (see Section 4.1). This is a dramatic improvement over the naive ratio of $O(m/b)$, but might still be larger than n. A somewhat stronger $O(\sqrt{m \log n/b})$ ratio is claimed in Footnote 10 in [10]. The proof sketched there apparently uses a private communication of Goemans, Schieber, Sudan and Williamson. An adaptation of this private communication is given in [10], but does not suffice for proving the $O(\sqrt{m \log n/b})$ ratio. (The $O(\sqrt{m \log n/b})$ ratio is apparently based on Claim 1, stated in Section 2. See also Section 4.1.)

We remark that there has been some confusion regarding the best approximation ratio known for bisection. For example, Arora in his survey [1] says:

The min-bisection problem is particularly intriguing. We currently know neither an algorithm that achieves an approximation ratio better than around $n^{1/2}$, nor an inapproximability result of any sort.

However, this quotation should not be misunderstood to imply that an approximation ratio of roughly $\sqrt{n}$ was known. (Tracing the roots of this quotation, our most plausible explanation for it is that the roughly $\sqrt{m}$ ratio of [10] was confused with $\sqrt{n}$.)

1.3 Our results

We present three approximation algorithms for bisection.

Theorem 1 *The minimum bisection problem can be approximated within the following ratios:*

1. $O(\sqrt{\frac{m}{b}}\log n)$
2. $O(n^{2/3}(\log n)^{4/3}/b^{1/3})$
3. $O(\sqrt{n}\log n)$

The third of these ratios gives the best approximation guarantee, unless either $b > m/n$ (in which case the first ratio is better) or $b > \sqrt{n}\log n$ (in which case the second ratio is better).

We use as building blocks approximation algorithms for the problem of cutting away exactly j vertices in a graph, i.e. finding a set of vertices $S \subset V$ of cardinality $|S| = j$ with the minimum number of edges connecting S to its complement $V \setminus S$. The following approximation guarantee that we obtain for this problem is particularly useful when j is small (e.g. when $j \ll \sqrt{n}$), and may be of independent interest.

Theorem 2 *The problem of cutting j vertices can be approximated within ratio $O(1 + j/\log n)$.*

Our approximation algorithms, with the exception of the second ratio in Theorem 1, work also for weighted graphs, in which edges have nonnegative weights and we wish to find a bisection that minimizes the total weight of edges cut.

The main contribution of our work is that it shows that unlike problems such as clique and chromatic number, bisection can be approximated within ratios of $O(n^\delta)$ for some δ strictly smaller than 1. Moreover, it indicates which algorithmic techniques are useful for approximating bisection. Specifically, we make extensive use of minimum-ratio cuts and of dynamic programming. Recently, modifications and extensions of these techniques were used in a more complicated algorithm that approximates bisection within a polylogarithmic ratio [7].

1.4 Techniques

At a high level, a plausible approach is as follows. An algorithm with approximation ratio $O(\sqrt{m/b}\log n)$ follows from known techniques. Modify the input graph G to a new graph G' for which the size of the minimum bisection is preserved, and the number of edges is reduced to $O(nb)$. Then apply the algorithm with approximation ratio $O(\sqrt{m/b}\log n)$ on G' to find a bisection within $O(\sqrt{n}\log n)$ of optimal.

A complication introduced by such a sparsification approach is the need to deal with weighted graphs, whereas the $O(\sqrt{m/b}\log n)$ approximation ratio works for unweighted graphs. For example, Benczur and Karger [3] show how the number of edges in a graph can be reduced to $O(n\log n)$ while roughly preserving all cut sizes. However, these edges are weighted and the total weight of the edges remains roughly m (the number of original edges). A more careful examination of the $O(\sqrt{m/b}\log n)$ approximation ratio shows that for edge weighted graphs m should be taken as the sum of weights of the edges, and hence the sparsification technique of [3] does not seem to help.

Alternatively, we may try to contract random edges as in [9]. It can easily be shown that as long as $m > nb$, we can contract edges essentially with no risk of ever merging two vertices from opposite sides of the optimal bisection. Hence we end up with a graph G' in which the total weight of remaining edges is $O(nb)$, as desired, and the minimum bisection is preserved. However, now vertices have weights. The notion of bisection changes to that of partitioning the graph into two parts of equal weight (rather than equal cardinality). The $O(\sqrt{m/b}\log n)$ algorithm is inapplicable for this problem.

We use a different procedure for merging vertices (this is done in algorithm SV in Section 3). In principle, this procedure can be applied to reduce the number of edges to nb. This fact can serve as intuition why such a procedure can potentially lead to an $O(\sqrt{n}\log n)$ approximation algorithm. However, we arrive at an $O(\sqrt{n}\log n)$ approximation in an indirect way, without ever reducing the number of edges in the graph to $O(nb)$. First we cut the graph into parts using min-ratio cuts. We then use the procedure for merging vertices in order to cut j vertices from a part by using a number of edges that is at most a factor of j

more than optimal (this is a slight generalization of a result of [12]). Then the parts are combined into two roughly equal parts, using a fairly elaborate dynamic programming. Finally, there is a balancing stage in which some vertices switch sides to get an exact bisection.

When bounding the approximation ratio of our algorithm, we count every edge that was cut by one of the min-ratio cuts, even though some of these edges do not end up in the bisection found by the algorithm. Likewise, we overcount the number of edges added to the bisection during the balancing stage. The approximation ratio acheived is the ratio of the total number of edges ever counted by our analysis, divided by b.

1.5 Conventions and notation

We assume that the optimal value of b is known to our algorithms (up to a constant multiplicative error). This assumption can be made with only $O(\log n)$ multiplicative overhead in the running time, because there are less than n^2 possible values for b.

For the sake of analysis, fix one optimal bisection and assume that its two sides are colored with different colors, e.g., blue and red. The algorithm does not see this coloring. The *color* of a set $S \subset V$ is the color of the majority of its vertices (and blue in case of equality). A vertex $v \in S$ is said to be a *minority* vertex if its color differs from the color of S. A set $S \subset V$ of vertices is said to be *monochromatic* if it lies in one side of the optimal bisection.

A *part* of the graph is either a set of vertices $S \subset V$, or the subgraph induced on S, depending on the context in which the term "part" is used.

1.6 Roadmap

In Section 2 we reproduce the known results about cutting graphs into nearly (rather than exactly) equal parts. Our analysis is somewhat different than that previously used for this problem, and extends more easily to the approximation algorithms that we later give for (exact) bisection. In Section 3 we give approximation algorithms for cutting exactly j vertices in a graph. Their approximation ratios are linear in j, and we shall later use them as subroutines only for small values of j. In Section 4 we present three algorithms for bisection, proving Theorem 1. In Section 5 we prove Theorem 2, which slightly improves over the results of Section 3, and hence indirectly can also be used to slightly improve the approximation ratio for bisection. We chose not to present various minor improvements that we have to the approximation ratio of bisection (affecting only the logarithmic terms), mainly because of the recent major improvements obtained in [7].

2 A pseudo-approximation for bisection

A pseudo-approximation for bisection cuts the graph into two nearly equal parts, rather than exactly equal. If the size of the smaller part is n' we call such a cut an $(n', n - n')$ cut. Leighton and Rao [11] have shown how to obtain an $(n', n - n')$-cut that cuts at most $O(b \log n)$ edges and for which $n/3 \leq n' \leq n/2$ (the algorithm does not control the exact size of n', just the range of values in which it falls). Goemans, Schieber, Sudan and Williamson (private communication) showed how the lower bound on n' can be pushed closer to $n/2$ at the expense of cutting more edges. Their result is understood to be of the following form:

Claim 1 *For every $1 \le k \le n/2$, one can produce an $(n', n-n')$ cut with $n/2-k \le n' \le n/2$ and the number of edges cut is $O(b(n/k)\log n)$.*

To the best of our knowledge, the proof of Claim 1 is based on *spreading metrics*, and was never published.

Leighton and Rao [10] observe that Claim 1 implies that bisection can be approximated within a ratio of $O(\sqrt{m \log n/b})$ (we may assume that $b \neq 0$), by choosing k as $n\sqrt{b \log n/m}$, and then moving the $n/2 - n'$ vertices of the lowest degree from the large part to the small part (we shall elaborate on this later). We note that as a function of n, this does not give an approximation guarantee better than the $n/2$ of [12], because m/b may be $\Omega(n^2)$.

In our algorithms for approximating bisection, we also use a pseudo-approximation for bisection, and then balance it so as to get an exact bisection. However, Claim 1 as a blackbox will not suffice for our purpose. Hence we prove here a claim that is slightly weaker than Claim 1. We shall use additional properties of the proof in later parts of the paper.

Lemma 1 *For every $1 \le k \le n/2$, one can produce an $(n', n-n')$ cut with $n/2 - k \le n' \le n/2$ and the number of edges cut is $O(b(n/k)(\log n)^2)$.*

To prove the lemma, we present an algorithm consisting of a cutting stage and a combining stage.

Cutting stage. Using the approximation algorithm for min-ratio cut, repeatedly break the graph into parts. Each part may be broken into smaller parts if a cut with ratio $O(\frac{b}{k}\log n)$ is found. Stop when no part of the graph can be cut with this ratio, and denote the resulting parts by $S_1, \ldots, S_l$.

Proposition 2 *After the cutting stage, the total number of minority vertices in all parts is at most k.*

Proof: Let r be the number of minority vertices. We prove that $r \le k$.

Let r_i be the number of minority vertices in S_i ($r_i = 0$ if S_i is monochromatic). Let b_i be the number of edges from the optimal bisection present in S_i. Then $\sum r_i = r$ and $\sum b_i \le b$. Hence for at least one of the S_i, $b_i/r_i \le b/r$. If $r > k$, then $b/r < b/k$, implying that a cut with such a ratio still exists. But then the approximation algorithm for min-ratio cut would have found in that part a cut with ratio $O(\frac{b}{k}\log n)$. $\square$

Combining stage. Use dynamic programming to select parts whose cardinalities add up to a value between $n/2-k$ and $n/2+k$. Such a selection exists, by Proposition 3. Combine these parts to one set A, and all other parts to a set B.

Proposition 3 *The sum of cardinalities of the blue parts is between $n/2 - k$ and $n/2 + k$.*

Proof: At most k red vertices are in blue sets. At most k blue vertices are not in blue sets. $\square$

To complete the proof of Lemma 1, we need to upper bound the number of edges cut by our algorithm. Edges are only cut in the cutting stage. For each min-ratio cut, the amortized cost of cut per vertex in the smaller part is $O(\frac{b}{k}\log n)$. Each vertex is in the smaller part of a cut at most $\log n$ times. Hence the number of edges cut is $O(n\frac{b}{k}(\log n)^2)$.

3 Cutting a small number of vertices

Let G be a graph on n vertices and m edges. For $j \leq n/2$, let b_j be the size of the minimum cut that cuts away exactly j vertices from G. Of course, $b_{n/2}$ is the size of the minimum bisection. We present two algorithms that find a $(j, n-j)$ cut whose size is small (relative to the optimum value b_j). These algorithms are important building blocks in our approximation algorithms for the bisection.

Algorithm Greedy: Greedily pick the j vertices of smallest degree.

Proposition 4 *The number of edges cut by algorithm Greedy is at most* $\min\{2jm/n\,,\, b_j + 2\binom{j}{2}\}$.

Proof: The average degree in G is $2m/n$, and hence there exists a set of j vertices (e.g. a random one) whose degrees sum up to at most $2jm/n$. The sum of degrees of the j vertices in the optimal cut is at most $b_j + 2\binom{j}{2}$.

This upper bounds the sum of degrees of the j vertices of smallest degree, and hence the size of the cut produced. $\square$

Our second algorithm is a modification of an approximation algorithm proposed for bisection in [12]. We assume that b_j is known to the algorithm.

Algorithm SV: Compute for all pairs of vertices u, v the size of the minimum cut between u and v. Merge (transitively) those pairs of vertices for which the minimum cut is greater than b_j. In the resulting graph, keep parallel edges but remove self loops. The resulting graph G' is composed of clusters, where a cluster is a set of vertices merged together. Each cluster has a weight equal to the number of vertices from which it is composed, and degree equal to the number of edges leaving the cluster.

Use dynamic programming to select a set of clusters whose total weight is exactly j and for which the sum of cluster degrees is smallest. This defines a cut of j vertices in the original graph.

Lemma 5 *The number of edges cut by algorithm SV is at most* $2jb_j$.

The proof of Lemma 5 follows from the following propositions.

Proposition 6 *Only vertices on the same side of the optimal* $(j, n-j)$ *cut are merged.*

Proof: For vertices on opposite sides of the optimal cut, their minimum cut is of size at most b_j. $\square$

Proposition 7 *The size of the minimum cut between every two clusters in G' is at most b_j.*

Proof: Let C_1 and C_2 be two clusters. Take arbitrary $v_1 \in C_1$ and $v_2 \in C_2$. The minimum cut between them in G is at most b_j, because otherwise they would belong to the same cluster. All vertices of C_1 must be on one side of this minimum cut and all vertices of C_2 must be on the other side (otherwise either C_1 or C_2 contain pairs of vertices whose minimum cut is at most b_j). Likewise, every other cluster must lie entirely on one side of the cut. Hence this is also a cut in G'. $\square$

Proposition 8 *Consider l arbitrary clusters. The total number of edges connecting between them is at most $(l-1)b_j$.*

Proof: Consider a graph C whose vertices are the clusters $C_1, \ldots, C_l$ and whose edges are the edges joining these clusters in G'. Iteratively use minimum cuts to cut C into individual clusters. $l-1$ minimum cuts are needed, and each such cut cuts at most b_j edges. $\square$

We now complete the proof of Lemma 5. Each cluster lies entirely on one side of the minimum $(j, n-j)$-cut. Let l be the number of clusters on the smaller side of the cut. The total number of edges connecting these clusters to other clusters is exactly b_j, and the total number of edges connecting between these clusters is at most $(l-1)b_j$. As $l \leq j$, the total sum of degrees of these clusters is at most $b_j + 2(l-1)b_j \leq (2j-1)b_j$.

We can achieve a slight improvement over algorithm SV and find a $(j, n-j)$ cut of size $O(1 + j/\log n)b_j$. To simplify the exposition, this algorithm is deferred to Section 5, as it improves the approximation ratio for bisection only by logarithmic factors.

4 Approximation algorithms for bisection

We present here three approximation algorithms for the minimum bisection problem.

4.1 An approximation ratio of $O(\sqrt{m/b}\log n)$

Our first algorithm is the one presented in a footnote in [10]. It uses Lemma 1 as a blackbox. (A factor of $\sqrt{\log n}$ can be saved by using Claim 1 instead.)

Cutting stage + combining stage. Following Section 2, with $k = n \log n \sqrt{b/m}$, produce an $(n', n-n')$-cut for which $n/2 - k \leq n' \leq n/2$ and the number of edges cut is $O(b(n/k)(\log n)^2)$.

Balancing stage. Move $j = n/2 - n' \leq k$ vertices of lowest degree from one set to another to obtain a bisection.

Lemma 9 *The number of edges in the bisection found is $O(b\sqrt{m/b}\log n)$.*

Proof: For our choice of k, the number of edges cut in the cutting+combining stage is $O(\sqrt{mb}\log n)$. To bound the number of edges cut in the fixing stage, observe that the bigger side in G has $n - n' \geq n/2$ vertices and at most m edges. Proposition 4 then implies that the number of edges cut in the fixing stage is at most $2jm/(n-n') \leq 4jm/n \leq 4km/n = O(\sqrt{mb}\log n)$. $\square$

The value of the parameter k was chosen so as to balance the number of edges cut in the two stages of the algorithm. For the second stage, we estimated the number of edges cut using the $2jm/n$ bound from Proposition 4, but did not use the $b_j + 2\binom{j}{2}$ bound from the same proposition. The reason for not using the latter bound (in the context of the current algorithm) is that we do not have any upper bounds on the value of b_j for the graph induced on the vertices of the larger side of the $(n', n-n')$-cut, other than the $2jm/(n-n')$ bound that we use anyhow.

4.2 Approximation within $O(n^{2/3}(\log n)^{4/3}/b^{1/3})$

In this section we wish to have an algorithm similar to that in Section 4.1, but to be able to use the bound $b_j + 2\binom{j}{2}$ from Proposition 4, with $b_j = b$. Then choosing $k \simeq n^{1/3}$, we cut roughly $bn^{2/3}$ edges in each stage of the algorithm.

The problem with this approach is how to ensure that we can indeed assume that $b_j \leq b$. In order to do this we separate the cutting stage and the combining stage and add in between a marking stage.

Cutting stage. Following Section 2, cut the graph into parts $S_1, \ldots, S_l$, using the approximation algorithm for min-ratio cut. As we have seen in Proposition 2, the total number of minority vertices in all parts is at most k. The number of edges cut in this stage is $O(b(n/k)(\log n)^2)$.

Marking stage. For each part S_i and every $0 \leq j \leq k$ create a version of S_i, denoted by S_i^j, in which the j vertices of lowest degree in S_i are *marked*. Note that in exactly one version, the number of vertices marked is exactly equal to the number of minority vertices in S_i. We call this version the *true* version of S_i.

Combining stage. Use dynamic programming to select at most one version from each part such that the parts cardinalities add up to a value n' where $n/2 \leq n' \leq n/2 + k$, the number of marked vertices among them is between $n' - n/2$ and k, and the sum of degrees of the marked vertices is at most $b + 2\binom{k}{2}$. Such a selection exists, by Proposition 10. It can be found by dynamic programming because after each part is considered, the parameters to remember are the current sum of cardinalities, number of marked vertices, and sum of their degrees, and each parameter ranges over polynomially many values. Combine these parts to one set A, and all other parts to a set B.

Proposition 10 *A selection of parts as postulated in the combining stage exists.*

Proof: Take the union of all blue sets. W.l.o.g., its cardinality is at least as large as that of the union of all red sets. Hence its cardinality is at least $n/2$. It is at most $n/2 + k$ because it contains at most k red vertices. In the true version of each blue part, the j vertices marked are of total degree not larger than that of the j red vertices which are the minority of this part. Hence, the total degree of the marked vertices as chosen by the dynamic programming is not larger than that obtained by marking red vertices and choosing the true versions of the blue parts. This total degree would have been at most $\sum_{parts}(b_j + 2\binom{j}{2})$ by the second part of Proposition 4. Since $\sum_{parts} b_j \leq b$ and $\sum_{parts} j \leq k$, the total degree of marked vertices in this case is bounded by $b + 2\binom{k}{2}$. $\square$

Balancing stage. Move $n' - n/2$ marked vertices from set A to set B.

The total number of edges cut in the balancing stage is at most $b + 2\binom{k}{2}$. The number of edges cut in the cutting stage was $O(nb(\log n)^2/k)$. Choosing $k = (nb)^{1/3}(\log n)^{2/3}$, the number of edges cut by our algorithm is $O((nb)^{2/3}(\log n)^{4/3})$.

Remark: Unlike sections 4.1 and 4.3, the analysis presented in this section does not extend easily to edge weighted graphs. The difficulty is in adapting the $b_j + 2\binom{j}{2}$ bound of Proposition 4 to edge weighted graphs.

4.3 Approximation within $O(\sqrt{n}\log n)$

In this section we wish to have an algorithm similar to that of Section 4.2, but to use the bound $2jb_j$ (from Lemma 5) on the number of edges cut in the balancing stage. We would like to argue that we can assume that $b_j \leq b$. An argument of this type was used in Section 4.2. However, now we will need to refine the argument. This will result in a different balancing stage, in which we do not only move vertices from the large side to the small side, but also move vertices from the small side to the large side. This becomes necessary because we move from side to side whole clusters, rather than individual vertices, and cluster sizes in the large side may not add up to give the exact number of vertices that need to be moved to create a bisection.

Cutting stage. Following Section 2, cut the graph into parts $S_1, \ldots, S_l$, using the approximation algorithm for min-ratio cut. As we have seen in Proposition 2, the total number of minority vertices in all parts is at most k. The number of edges cut in this stage is $O(b(n/k)(\log n)^2)$.

Marking stage. For each part S_i and every $0 \leq j \leq k$ create a version S_i^j in which exactly j vertices in S_i are *marked*. The j vertices are chosen using algorithm SV and satisfy Lemma 5 with $b_j = b$, namely, that the number of edges cut is at most $2jb$. (In fact, with some extra work we can find a cut of size at most $2jb_j$, where b_j is measured relative to S_i. Details omitted.) Note that for exactly one version, the number of vertices marked is exactly equal to the number of minority vertices in S_i. We call this version the *true* version.

Combining stage. Use dynamic programming to create two sets A and B with the following properties. From each part exactly one version is selected and put in either A or B. The number of vertices in A is n' where $n/2 - k \leq n' \leq n/2$, and the number of vertices in B is $n - n'$. The number of marked vertices in A is m_A and in B is m_B, satisfying $n' + m_B - m_A = n/2$ and $m_A + m_B \leq k$. The sum of the cut sizes of the marked vertices (in their corresponding parts) is at most $2bk$. Such a selection exists, by Proposition 11. It can be found by dynamic programming because after each part is considered, the parameters to remember are the current cardinalities of sets A and B, the number of marked vertices in each set, and sum of their cut sizes (only the minimum needs to be retained here), and each parameter ranges over polynomially many values.

Proposition 11 *A selection of parts as postulated in the combining stage exists.*

Proof: Take the union of all blue sets. W.l.o.g., its cardinality is not larger than that of the union of all red sets. Hence its cardinality is at most $n/2$. It is at least $n/2 - k$ because it misses at most k blue vertices. As the j vertices marked in the true version of each part induce a cut at most $2j$ times larger than that induced by the j minority vertices (by Lemma 5), and the size of a cut induced by minority vertices is at most b, the sum of cut sizes for minority vertices is at most $b \sum_{parts} 2j \leq 2bk$. □

Balancing stage. Exchange the marked vertices between sets A and B.

The total number of edges cut in the balancing stage is at most $2bk$. The number of edges cut in the cutting stage was $O(b(n/k)(\log n)^2)$. Choosing $k = \sqrt{n} \log n$, the number of edges cut by our algorithm is $O(b\sqrt{n} \log n)$.

5 Better approximation ratios

We presented an algorithm that approximates bisection within a ratio of $O(\sqrt{n} \log n)$. It follows from our work that a necessary and sufficient condition for improving the approximation ratio significantly beyond $\sqrt{n}$ (i.e. ignoring logarithmic terms), is to have an algorithm that finds $(j, n-j)$ cuts with approximation ratios significantly better than j, for $j \simeq \sqrt{n}$. Algorithm SV achieves an approximation ratio of $2j$ for this last problem. This can be slightly improved to $O(1 + j/\log n)$, by combining the idea of randomly contracting edges (as in [9]) with dynamic programming, as sketched below.

Algorithm Contract: The algorithm proceeds in iterations. Each iteration includes two stages (i) random edge contraction and (ii) dynamic programming to find a $(j, n-j)$ cut, both described below. The iterative process stops when there are no edges to contract, and from all $(j, n-j)$ cuts found throughout the iterations, the one of minimum size is taken. Each contraction decreases the number of vertices by 1, and thus the number of iterations is bounded by n.

Edge contraction stage. Choose an edge at random and merge its endpoints, keeping parallel edges but removing self loops. The resulting graph is composed of clusters, where a cluster is a set of vertices merged together. Each cluster has a weight equal to the number of vertices from which it is composed, and degree equal to the number of edges leaving the cluster.

Dynamic programming stage. Select in the graph of the corresponding iteration, a set of clusters whose total weight is exactly j and for which the sum of cluster degrees is the smallest. If such a set is found, it defines a cut of j vertices in the original graph.

To amplify the success probability, repeat the above iterative procedure polynomially many times, taking the best cut found.

Lemma 12 *With high probability, algorithm Contract finds a $(j, n-j)$ cut of size at most $O(1 + j/\log n)b_j$.*

Proof: It clearly suffices to show that the success probability of one repetition of the algorithm is at least some inverse polynomial in n. For the sake of analysis, fix one optimal cut of size b_j which cuts away a set S^* of exactly j vertices. Intuitively, as long as there are many edges connecting between clusters of S^*, a random edge is more likely to be chosen from these edges than from the b_j edges of the cut. When there are only few edges connecting clusters of S^*, dynamic programming on the cluster degrees will find a relatively small cut.

We claim that with at least an inverse polynomial probability, none of the b_j edges of the optimal cut is contracted before the clusters of S^* will have sum of degrees at most $O(1 + j/\log n)b_j$. Indeed, note that we can *ignore* the iterations in which the randomly chosen edge is not adjacent to any of the vertices of S^*. We thus consider only the iterations in which the chosen edge either connects between two clusters of S^*, or it is one of the b_j edges of the optimal $(j, n-j)$ cut. As long as the number of edges connecting between clusters of S^* is more than $(1 + j/\log n)b_j$ (and none of the b_j cut edges was contracted yet), the probability for a successful iteration, in which an edge connecting two clusters of S^* is chosen, is at least $\frac{1+j/logn}{2+j/logn} \geq \max\{\frac{1}{2}, 1 - \frac{\log n}{j}\}$. We need no more than $j-1$ successful iterations, and therefore the probability that one repetition will find a $(j, n-j)$ cut of size $O(1 + j/\log n)b_j$ is at least some inverse polynomial in n. $\square$

Algorithm Contract extends to weighted graphs, as follows. For edge weighted graphs, choose edges at random with probabilities proportional to their weights, to get the same approximation ratio $O(1 + j/\log n)$.

For graphs in which vertices have polynomially bounded nonnegative integer weights, a w-cut cuts away vertices of total weight w. Let b_w be the minimum cost (weight of edges cut) of a w-cut. We wish to find a w-cut with cost approximating b_w. (The polynomial bound on vertex weights is required so as to be able to check in polynomial time whether a w-cut exists at all. This last problem is simply the subset-sum problem, which has pseudopolynomial time algorithms, but is NP-hard in general.) For $1 \leq j \leq \min[w, n-1]$, let $b_{w,j}$ be the minimum cost of a cut that cuts away j vertices of total weight w (and ∞ if no such cut exists). Fixing a value of j, the proof of Lemma 12 extends to show that with high probability algorithm Contract will find a cut of size $O(1 + j/\log n)b_{w,j}$. Hence we can find a cut of weight w and cost $O(\min_j[(1 + j/\log n)b_{w,j}])$. Note that the minimum in the latter bound is not necessarily obtained at a value of j for which $b_{w,j} = b_w$.

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

[1] S. Arora. The approximability of NP-hard problems. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 337–348. ACM, 1998.

[2] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *J. Comput. System Sci.*, 58(1):193–210, 1999.

[3] A. A. Benczúr and D. R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 47–55. ACM, 1996.

[4] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *J. Comput. System Sci.*, 28(2):300–343, 1984.

[5] R. B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In *28th Annual Symposium on Foundations of Computer Science*, pages 280–285. IEEE, Oct. 1987.

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

[7] U. Feige and R. Krauthgamer. Manuscript in preparation.

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

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

[10] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, to appear.

[11] F. T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *29th Annual Symposium on Foundations of Computer Science*, pages 422–431. IEEE, Oct. 1988.

[12] H. Saran and V. V. Vazirani. Finding k cuts within twice the optimal. *SIAM J. Comput.*, 24(1):101–108, 1995.

[13] D.B. Shmoys. Cut problems and their applications to divide-and-conquer. In *Approximation Algorithms for NP-Hard Problems.* (Editor: D. Hochbaum) PWS Publishing Company, 1997.