A polylogarithmic approximation of the minimum bisection*

Uriel Feige† Robert Krauthgamer‡

November 19, 2001

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 cost is the number of edges connecting the two sets. It is known that finding a bisection of minimum cost is NP-hard. We present an algorithm that finds a bisection whose cost is within ratio of $O(\log^2 n)$ from the minimum. For graphs excluding any fixed graph as a minor (e.g. planar graphs) we obtain an improved approximation ratio of $O(\log n)$. The previously known approximation ratio for bisection was roughly $\sqrt{n}$.

1 Introduction

Let $G(V, E)$ be an undirected graph with n vertices and m edges, where n is even. For a subset S of the vertices (with $S \neq \emptyset, V$), the *cut* $(S, V \setminus S)$ is the set of all edges in G with one endpoint in S and one endpoints in $V \setminus S$; these edges are said to be cut by $(S, V \setminus S)$. The *cost* of a cut is the number of edges in it.

A cut $(S, V \setminus S)$ is called a *bisection* of G if its two sides, S and $V \setminus S$, are each of size $n/2$. We denote the minimum cost of a bisection of G by b. *Minimum bisection* is the problem of computing b for an input graph G. This problem is NP-hard, see [GJS76], and we address the problem of approximating it.

An algorithm is said to approximate a minimization problem within ratio $f \geq 1$ if it runs in polynomial time and outputs a solution whose cost is at most f times the cost of

*A preliminary version of this paper appears in Proceedings of 41st Symposium on Foundations of Computer Science (FOCS '00), pp. 105-115, November 2000. This Research was supported in part by a Minerva grant.

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. This author is the Incumbent of the Joseph and Celia Reskin Career Development Chair. Email: `feige@wisdom.weizmann.ac.il`

‡Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. This author is currently at the International Computer Science Institute (ICSI) and at the Computer Science Division, University of California, Berkeley, CA 94720, USA. This author's research was supported in part by the Open University of Israel. Email: `robi@wisdom.weizmann.ac.il` or `robi@cs.berkeley.edu`

the optimal solution. The problem is said to have a PTAS (polynomial time approximation scheme) if for every fixed $f > 1$ there is an algorithm with approximation ratio f.

1.1 Previous work

Leighton and Rao [LR88, LR99] showed how to approximate within ratio $O(\log n)$ *minimum-quotient cuts*, which we shall call *min-ratio cuts*. In these cuts, one wishes to minimize the *cut ratio* (also called *edge expansion* or *flux*) $c/|S|$, where c is the number of edges cut, and $|S|$ is the cardinality of the smaller of the two vertex sets.

A *β-balanced cut* is a cut that partitions the graph into two parts, each of size at most βn. Leighton and Rao [LR88] used the approximate min-ratio cuts to find a 2/3-balanced cut (also called *edge separator*) with at most $O(b \log n)$ edges, see also [LR99, Shm97]. Note that such a 2/3-balanced cut does not provide an $O(\log n)$ approximation for the value of b. For example, when the graph consists of 3 disjoint cliques of equal size, an optimal 2/3-balanced cut has no edges, whereas $b = \Omega(n^2)$.

A straightforward approach for obtaining an exact bisection is to first find an almost balanced cut (e.g. using approximate min-ratio cuts) and then move a few low degree vertices from one side to the other. Using this approach one can approximate bisection within a ratio of $\tilde{O}(\sqrt{m/b})$ (we use $\tilde{O}(f)$ to denote $O(f \cdot \text{polylog } n)$) see e.g. [LR99, Footnote 10] and [FKN00]. This is a dramatic improvement over the naive ratio of $O(m/b)$ (achieved by arbitrarily picking $n/2$ vertices), but might still be larger than n.

In terms of n, the best approximation ratio previously known is $\tilde{O}(\sqrt{n})$, due to [FKN00]. Their approach follows, in part, a divide-and-conquer paradigm. Two of their main tools are (i) approximate min-ratio cuts, which are used to recursively break the graph, and (ii) dynamic programming, which is used to combine certain possible parts into an exact bisection. The current work also uses a divide-and-conquer approach, but in a more sophisticated way.

Additional related work include the following. In [SV95], Saran and Vazirani give an algorithm that approximates bisection within a ratio of $n/2$. In [AKK99], Arora, Karger and Karpinski show that bisection has a PTAS for everywhere-dense graphs, i.e. graphs with minimum degree $\Omega(n)$. In [GSV99], Garg, Saran and Vazirani give an approximation ratio of 2 for the problem of finding a 2/3-balanced cut of minimum cost in a planar graph. Their result extends to a β-balanced cut, for any $\beta \geq 2/3$, but does not extend to a bisection, which is a 1/2-balanced cut. In [BJ92], Bui and Jones show that for any fixed $\epsilon > 0$, it is NP-hard to approximate the minimum bisection within an *additive* term of $n^{2-\epsilon}$. In terms of approximation ratio, however, there is no known hardness of approximation result which excludes the possibility that bisection has a PTAS.

1.2 Our results

Our main result is an algorithm for approximating the minimum bisection within a polylogarithmic ratio.

Theorem 1 *A bisection of cost within ratio of $O(\log^2 n)$ of the minimum can be computed in polynomial time.*

In Section 2 we give an overview of the algorithm. On a high level, the algorithm follows a divide-and-conquer approach. The input graph is recursively divided into parts, using a new cut notion which we call an *amortized cut*, and then the parts are combined into a bisection using dynamic programming.

In Section 4 we describe our algorithm for approximating bisection, based on a subroutine for finding an amortized cut. If the subroutine is guaranteed to find a ρ-amortized cut in a graph, the algorithm computes a bisection whose cost is within ratio of $1 + O(\rho \log n)$ of the minimum.

In Section 3 we devise an algorithm for finding an $O(\log n)$-amortized cut in a general graph. By using this algorithm as a subroutine in the $1 + O(\rho \log n)$ approximation algorithm for bisection, we are guaranteed that $\rho = O(\log n)$, proving Theorem 1. The subroutine uses a τ-approximate min-ratio cut in order to find an $O(\tau)$-amortized cut. The best known approximation algorithms for min-ratio cut in general graphs, due to Leighton and Rao [LR88, LR99] and due to [AR98, LLR95], have approximation ratio $\tau = O(\log n)$.

In certain graph families, there is a better approximation ratio τ for the min-ratio cut problem. If these graph families are closed under taking induced subgraphs, then we can approximate bisection within an improved ratio of $O(\tau \log n)$. For example, it is shown in [KPR93] that in graphs excluding any fixed graph as a minor (e.g. bounded-genus graphs) min-ratio cut can be approximated within a constant ratio, i.e. $\tau = O(1)$.

Theorem 2 *In graphs excluding any fixed graph as a minor (e.g. planar graphs), a bisection of cost within ratio of $O(\log n)$ of the minimum can be computed in polynomial time.*

In Section 5 we show that our results extend to several natural generalizations of the bisection problem. These extensions include, for example, bisection of graphs with arbitrary nonnegative edge costs and graph partitioning into three parts of equal size.

1.3 Conventions and notation

We will often denote the two sides of a (not necessarily optimal) bisection as *white* W and *black* B. A graph may have several different bisections of minimum cost. For the analysis, let us fix one of them (arbitrarily) and call it *the fixed optimal bisection* (W^*, B^*).

For V_1, V_2 two disjoint subsets of vertices in a graph, let $e(V_1, V_2)$ denote the number of edges with one endpoint in V_1 and the other endpoints in V_2. Subsets $V_1, V_2 \subset V$ are called a *partition* of V if they are nonempty, disjoint, and their union is equal to V. In our context, V is the vertex set of a graph, and then a partition $V = V_1 \cup V_2$ is equivalent to the cut (V_1, V_2).

A subset of vertices $S \subset V$ with $0 < |S| < |V|$, corresponds to a cut $(S, \overline{S})$ in the graph, where $\overline{S} = V \setminus S$. We denote by $r(S)$ the ratio of this cut, i.e. $r(S) = \frac{e(S,\overline{S})}{\min\{|S|,|\overline{S}|\}}$, and by $r'(S)$ the ratio of this cut towards S, i.e. $r'(S) = \frac{e(S,\overline{S})}{|S|}$. We call S a *part* of the graph, referring either to the set of vertices S or to the subgraph induced on S, depending on the context.

2 Overview and techniques

Our approximation algorithm for minimum bisection has three stages, as outlined below.

Stage 1: Decomposition. This stage consists of a sequence of *divide steps*. The input to a divide step is a *part* of the input graph G, i.e. a vertex set and the subgraph induced on it, and the output is a partition of the vertex set into two nonempty subsets, giving two new parts of the graph. These divide steps are applied on the input graph G recursively, until it is decomposed into individual vertices.

The output of the whole decomposition stage is a binary tree T, that we call the *decomposition tree*. Each node i of the tree contains a part V_i obtained in a divide step, as follows. The root of the tree contains the input graph G, the leaves of the tree contain individual vertices of G, and the two direct descendants of a node i are the two subparts obtained in the divide step of its part V_i.

To complete the description of the decomposition stage, we need to explain how a divide step is performed. This is done using a new notion called an *amortized cut*, which we define later in this section. We devise an algorithm for finding amortized cuts in Section 3. The decomposition stage is described in more detail in Section 4.1.

Stage 2: Labeling. Consider a *labeling* of the decomposition tree T, which labels each (nonleaf) tree node as either white or black. Fixing a parameter $1/2 < \alpha < 1$, we say that a labeling is *α-consistent* with respect to a white-black bisection (W, B) of the input graph if every part V_i (at a tree node i) satisfies that $|W \cap V_i| \leq \alpha|V_i|$ if the label of node i is white, and that $|B \cap V_i| \leq \alpha|V_i|$ if the label of node i is black.

The desired outcome of the labeling stage is a labeling which is α-consistent with the fixed optimal bisection (W^*, B^*), called in short an *opt-consistent labeling*. However, an optimal bisection is not known to the algorithm, so instead of finding an opt-consistent labeling, this stage produces a family of labelings, such that at least one member of the family is opt-consistent. The description of how this is done is deferred to Section 4.2. For the purpose of this overview, it will be convenient to think of the labeling stage as if it produces only one labeling, which is opt-consistent.

Stage 3: Combining. Given a decomposition tree T and an arbitrary (not necessarily opt-consistent) labeling of it, the combining stage assigns to each vertex v of the input graph G a *white charge* and a *black charge*. The two charges are simple to compute based on the labels along the path from the root of T to the leaf that contains the vertex v.

The *charge of a bisection* (W, B) of the input graph G (with respect to the labeling) is defined as the sum of the white charges of the vertices of W and the black charges of the vertices of B. The functions white charge and black charge have the property that for every bisection, charge is an upper bound on cost (regardless of the labeling).

If the charge is defined with respect to an opt-consistent labeling of T then our notion of amortized cut used in the decomposition stage guarantees in addition that the charge of the fixed optimal bisection is within a polylogarithmic factor of its cost b. Hence, using the opt-consistent labeling produced by the labeling stage ensures that the input graph G contains a bisection whose charge is within polylogarithmic ratio of b.

Finding a bisection of minimum charge in G is relatively straightforward. Associate with each vertex a *net-charge*, which is its white charge minus its black charge, and pick the $n/2$ vertices with smallest net-charge to form one side W, leaving the remaining $n/2$ vertices in another side B. The bisection (W, B) that we find has minimum charge, and its cost is thus within a polylogarithmic factor of b, the cost of the minimum bisection.

It is interesting to note that finding a minimum cost bisection is an optimization problem with a quadratic objective function (minimizing the number of edges, where edges are pairs of vertices). Finding a minimum charge bisection (given the decomposition tree and an opt-consistent labeling) is an optimization problem with a linear objective function (sum of net-charges over individual vertices). Hence in a sense, our algorithm performs a linearization of a quadratic function, and loses a polylogarithmic factor in the process.

The above presentation of the combining stage was oversimplified. The output of the labeling stage is not one labeling that is opt-consistent, but rather a large family of labelings, such that at least one of them is opt-consistent. Moreover, this family has exponential cardinality, so we cannot try the above net-charge approach on each labeling separately. Instead, we exploit the structure of this family of labelings and use dynamic programming to compute a labeling from the family and a bisection, such that the charge of this bisection with respect to this labeling is minimum over all labeling-bisection pairs. Details appear in Section 4.4.

In the rest of the overview we shall introduce and discuss the notion of *amortized cut*, which is of central importance in bounding the ratio between the charge and the cost of the fixed optimal bisection. To motivate this new notion we present our algorithm as a divide-and-conquer algorithm. We then suggest a kind of cut that is desirable for the algorithm's divide step and call this cut notion an amortized cut.

Divide and conquer approach

A possible divide and conquer approach for a graph problem is to divide the input graph G into two parts (using a cut), solve a subproblem for each part, and then combine the solutions of the two subproblems into a solution for G. This approach can be applied recursively, and then the input graph G is recursively divided into smaller and smaller parts, where each part is associated with a subproblem. Note that the divide step cut is a tool of this approach, and is not intended to be a solution to the subproblem.

In our context, the graph problem is minimum bisection, and we apply this divide and conquer approach for the more general problem of cutting away an arbitrary number of vertices that is given as part of the input (bisection is the special case where the given number is $n/2$). Similarly, the subproblem of each part requires to cut away (from that part) an arbitrary number of vertices that is given in the subproblem. Note that minimum bisection is a cut problem, and therefore in addition to the *divide step cuts* we have here also *solution cuts* (later called *combined cuts*). Note that the solution cut of a part need not be the same as the divide step cut of this part.

Our three stage algorithm outlined above follows this divide and conquer approach. The task of breaking the input graph into smaller and smaller parts is performed by the decomposition stage, whose decomposition tree T represents the recursive structure of the divide steps.

For such a divide and conquer approach to be successful, it is desirable that (i) each of the two subproblems can be solved separately; and (ii) the solutions of the two subproblems can be combined while incurring a relatively small additional cost. Below we provide an overview of how our algorithms handles these issues.

Consider the problem of cutting away k vertices from a part $U \subseteq V$ of the input graph. The corresponding divide step uses a cut (U_1, U_2) of U to break this problem into the two subproblems of cutting away k_1 vertices from U_1 and of cutting away k_2 vertices from U_2, with $k = k_1 + k_2$. (For the sake of exposition assume that k_1, k_2 can be guessed.) Let us assume that the subproblem associated with each subpart U_i is solved separately (by recursion) and the solution obtained for it is a cut (C_i, F_i) with $|C_i| = k_i$ (see also Fig. 1). The two solution cuts are then combined into a cut of U that separates $k = k_1 + k_2$ vertices, namely $(C_1 \cup C_2, F_1 \cup F_2)$. Let $Cut(U', k')$ denote the cost of the cut of U' that separates k' vertices and is found by the algorithm. Then the cost of the combined cut is given by

$$Cut(U, k) = Cut(U_1, k_1) + Cut(U_2, k_2) + e(C_1, F_2) + e(C_2, F_1). \tag{1}$$

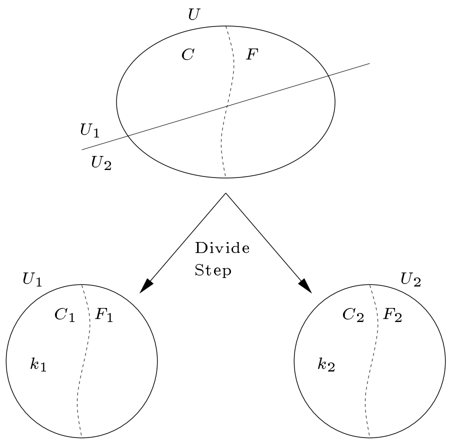

Figure 1: The divide and conquer paradigm

Previous accounting method

The approach of [FKN00] is based on a straightforward upper bound on the cost (1) of the combined cut. The additional cost incurred by the divide step, i.e. $e(C_1, F_2) + e(C_2, F_1)$, is at most the cost of all the edges cut by the divide step, i.e. $e(U_1, U_2)$, yielding the upper bound

$$Cut(U, k) \le Cut(U_1, k_1) + Cut(U_2, k_2) + e(U_1, U_2). \tag{2}$$

We remark that a bound similar to (2) is used in divide and conquer algorithms for many other graph problems, such as minimum cut linear arrangement (a.k.a. cutwidth), see e.g. [LR99].

The divide steps of [FKN00] use an approximate min-ratio cut to break each part U. This cut appears to be suitable for the bound (2) because it minimizes the cost of the cut (U_1, U_2), and at the same time tries to cut the part U into parts of roughly equal size, so as to minimize the depth of the recursion.

It is particularly instructive to evaluate the quality of our upper bound in the case where the computed cut $(C_1 \cup C_2, F_1 \cup F_2)$ is just the cut induced on U by the optimal bisection (W^*, B^*). Intuitively, we analyze the case where the algorithm happens to find the optimal bisection. In fact, we will later use dynamic programming to find a bisection for which the upper bound is minimized, so such an analysis bounds from above the cost of the output bisection.

There are cases where the upper bound (2) is tight (i.e. holds with equality). Indeed, the cuts within each U_i are computed independently of each other, and so it might happen that all the edges between the two parts U_1, U_2 end up in the combined cut. However, this bound is insensitive to cases where only few of the edges that are cut in the divide step end up in the combined cut, leading to a relatively poor approximation ratio.

New accounting method

We introduce a more sophisticated way of bounding the cost of the combined cut. Since $F_1 \subseteq U_1$ and $F_2 \subseteq U_2$ we can bound the cost of the combined cut by

$$Cut(U, k) \leq Cut(U_1, k_1) + Cut(U_2, k_2) + e(C_1, U_2) + e(C_2, U_1). \tag{3}$$

Unlike the actual cost (1), the upper bound (3) can be used in a divide and conquer approach, as follows. Let us call $e(C_1, U_2) + e(C_2, U_1)$ the *charge of the divide step* of U. This charge can be distributed into a charge $e(C_1, U_2)$ of the part U_1, and a charge $e(C_2, U_1)$ of the part U_2. The charge of a part U_i consists of the edges going from C_i to the other part U_{3-i}, and thus depends on the cut (C_i, F_i) chosen in the part U_i, but not on the cut chosen in the other part U_{3-i}. We obtain two separate subproblems (as in each part U_i we want to find a cut (C_i, F_i) for which sum of the cost of this cut and the charge to this part is minimal), enabling a recursive divide and conquer approach. In contrast, the terms $e(C_1, F_2)$ and $e(C_2, F_1)$ of the actual cost of the combined cut depend on the cuts chosen in both parts, and do not allow to break the problem into two separate subproblems.

The new accounting method makes a distinction between the two sides C and F of the combined cut. Unlike e.g. in (2), these two sides have different roles in the upper bound (3), and we will choose in a certain way which side is referred to as C (and which as F). Since we wish to minimize the charge, it makes sense to choose the smaller of the two sides to be C. In our analysis we have a somewhat relaxed condition, requiring that $|C| \leq \alpha |U|$, for a fixed $1/2 < \alpha < 1$. The task of identifying a side C as required above in each divide step (i.e. each node of the decomposition tree) is performed by the labeling stage, as explained in Section 4.2.

The *charge of a bisection* is the upper bound that is obtained by applying the upper bound (3) recursively, i.e. it is the sum of the charges of all the divide steps. In Section 4.3

we discuss this notion in more detail, and in Section 4.4 we show that its current formulation is equivalent to the one from Stage 3 of the algorithm outline (where the identification of a side C at each divide step corresponds to labeling of the decomposition tree T). ¿From the current formulation it is straightforward that the charge of a bisection is always an upper bound on its cost (regardless of the identification of C at each divide step, i.e. the tree labeling).

We call the vertices of $C = C_1 \cup C_2$ *charged* and the vertices of $F = F_1 \cup F_2$ *free*. The edges in the part U can then be classified as charged-charged, charged-free or free-free, according to their two endpoints.

Desired divide step

Rather than find a bisection of minimum cost, our approximation algorithm looks for a bisection of minimum charge. Our desired divide step is therefore one that guarantees that for the fixed optimal bisection, charge can be used to approximate cost. By the labeling stage, it suffices to refer here to charge with respect to an opt-consistent labeling, so from now on we assume that $|C| \leq \alpha|U|$ at each divide step.

Consider the charge of the fixed optimal bisection, and recall that it is the sum of the charges of all the divide steps. The charge of a divide step of a part U is $e(C_1, U_2) + e(C_2, U_1)$ and can be written also as $e(C_1, F_2) + e(C_2, F_1) + 2e(C_1, C_2)$, i.e. the cost of the charged-free edges that the divide step cuts and twice the cost of the charged-charged edges that it cuts. Observe that a charged-free edge is always an edge of the fixed optimal bisection (and vice versa) and that each edge is cut exactly once in the decomposition stage. Hence, all the charged-free edges cut in all the divide steps are exactly all the edges of the fixed optimal bisection. So for the fixed optimal bisection, the difference between charge and cost is twice the cost of all the charged-charged edges cut in all the divide steps.

It is therefore desired that the divide step cuts relatively few charged-charged edges, where relative here is with respect to b, the cost of the fixed optimal bisection. Since b is the total cost of the charged-free edges that are cut in all the divide steps, we seek an *amortization scheme* that amortizes the total cost of all charged-charged edges cut against the total cost of all charged-free edges cut. The partition of vertices to charged and free is not known to the divide step, and we therefore require that the amortization scheme holds for every possible partition of vertices to charged and free.

A simple amortization scheme can consider each divide step separately and amortize the cost of the charged-charged edges cut in a divide step against the cost of the charged-free edges cut in the same divide step. Suppose that in every divide step the amortized cost in this method is at most ρ, i.e. at every part U we have that $e(C_1, C_2) \leq \rho[e(C_1, F_2) + e(C_2, F_1)]$. Then the total cost of charged-charged edges cut in all divide steps is clearly at most ρb, and the charge of the fixed optimal bisection is at most $(1 + 2\rho)b$.

The problem with this simple amortization scheme is that in order to guarantee that the scheme holds for all possible partitions of vertices to charged and free, ρ might be required to be at least n, a value that is too high for our intended application. For example, consider a graph that consists of two cliques of size $n/2$ connected by an edge e. If the divide step breaks any of the cliques, then letting this clique be C and the other clique be F, the amortization cost will be at least n. Otherwise, the divide step consists of the edge e and then letting C consist of the two endpoints of e, the amortization cost will be infinite.

We employ a more complicated amortization scheme that allows a small amortization cost ρ but introduces an additional logarithmic factor. The reason for the logarithmic factor is that this scheme amortizes against the same edge more than once (but, in a sense, not too many times). Another complication is that this scheme actually has two amortization methods, and it uses at each divide step the one that is better (for that divide step).

Amortized cut

We amortize the cost of the charged-charged edges cut in a divide step against the cost of the charged-free edges *in the part being divided*, i.e. in the divide step of a part U we amortize $e(C_1, C_2)$ against $e(C, F)$. The edges that we amortize against are not cut in this divide step, and hence an edge may receive an amortized cost in many divide steps. However, our amortization scheme described below will guarantee that the total cost amortized against a single edge is at most $O(\rho \cdot \log n)$, for a suitable ρ. Since the edges that we amortize against are charged-free edges and hence edges of the fixed optimal bisection, it would follow that the total cost of the charged-charged edges cut in all the divide steps is at most $O(\rho \log n) \cdot b$, and so the charge of the fixed optimal bisection is $(1 + O(\rho \log n)) \cdot b$.

For motivation, consider the case where the divide steps recursion has depth $O(\log n)$, e.g. when all the divide steps are roughly balanced. In this case, an edge can receive an amortized cost in at most $O(\log n)$ divide steps. Suppose that in every divide step the amortized cost is at most ρ, i.e. in every part U we have that $e(C_1, C_2) \leq \rho \cdot e(C, F)$. Then the total cost amortized against a single edge is at most $O(\rho \log n)$.

We do not require that the divide steps are balanced, but rather scale the amortization cost at a part U according to the imbalance of its divide step. Out of the several possible scaling factors we will use only the following two, where we assume, without loss of generality, that $|U_1| \leq |U_2|$. The first scaling factor is $e(C_1, F_1)/e(C, F)$, and its corresponding amortization method requires that

$$e(C_1, C_2) \leq \rho \cdot \frac{e(C_1, F_1)}{e(C, F)} \cdot e(C, F). \tag{4}$$

The second scaling factor is $|C_1|/|C|$, and its corresponding amortization method requires that

$$e(C_1, C_2) \leq \rho \cdot \frac{|C_1|}{|C|} \cdot e(C, F). \tag{5}$$

Alternative formulations. The first amortization method (4) can be written also as $e(C_1, C_2) \leq \rho \cdot e(C_1, F_1)$. A convenient interpretation of this formulation is that we amortize against the charged-free edges inside U_1, the smaller side of the divide step cut (rather than inside U, the part being divided), and the amortized cost is required to be at most ρ.

The second amortization method (5) can be written also as $e(C_1, C_2) \leq \rho \cdot r'(C) \cdot |C_1|$ where $r'(C) = e(C, F)/|C|$ (see Section 1.3 for the difference between $r'(C)$ and $r(C)$). A convenient interpretation of this formulation is that we amortize against the vertices in C_1, the charged vertices inside the smaller side of the divide step cut, and the amortized cost is required to be at most $\rho \cdot r'(C)$.

Total amortized cost. The total cost amortized in the first method (4) is at most $O(\rho \log n) \cdot b$. Indeed, let us use the alternative formulation in which the amortization is only against edges inside U_1, the smaller side of the divide step cut. An edge can be inside U_1 in at most $\log n$ divide steps (since the size of the part it is contained in reduces at each such divide step by a factor of 2). Hence the total cost amortized in this method against a single edge (of the fixed optimal bisection) is at most $O(\rho \log n)$, and the claim follows.

The total cost amortized in the second method (5) is also at most $O(\rho \log n) \cdot b$. Indeed, we show in Section 4.3 that the total cost amortized in this method against a single edge (of the fixed optimal bisection) is at most $O(\rho \log n)$ (essentially by careful summation of the relevant terms of the form $|C_1|/|C|$), and the claim follows.

Our amortization scheme. Our amortization scheme chooses at each divide step the scaling factor that is better for this divide step, and so it suffices to have that at each part U at least one of (4) and (5) holds. It follows from the above discussion (see Section 4.3 for a full proof) that the total cost amortized in both methods together is at most $O(\rho \log n) \cdot b$.

We can now formally define our desired divide step according to the (alternative formulations of) the two amortization methods described above. We call this cut an amortized cut.

Definition (Amortized cut). Let (U_1, U_2) be a cut with $|U_1| \leq |U_2|$ in a graph $G'(U, E')$, and let $U = C \cup F$ be a partition of the graph vertices U to charged vertices C and free vertices F. Let us denote $C_i = U_i \cap C$ and $F_i = U_i \cap C$ for $i = 1, 2$, as in Fig. 1. Let

$$\rho_e = \frac{e(C_1, C_2)}{e(C_1, F_1)} \qquad \text{and} \qquad \rho_v = \frac{e(C_1, C_2)}{|C_1| \cdot r'(C)} \tag{6}$$

where $r'(C) = e(C, F)/|C|$. We call ρ_e the *amortized cost for the edges*, and ρ_v the *amortized cost for the vertices* (note that ρ_e, ρ_v depend on C, F).

The *amortized cost of the cut* (U_1, U_2) is the maximum of $\min\{\rho_e, \rho_v\}$, where the maximum is taken over all partitions $U = C \cup F$ with $0 < |C| \leq \alpha |U|$ for a fixed $\frac{1}{2} \leq \alpha < 1$. We say that the cut (U_1, U_2) is *ρ-amortized* if its amortized cost is at most ρ.

In order us to correctly handle cases where there is no cost to amortize against, we use the convention that $\frac{0}{0}$ is defined to be 0, and that $\frac{t}{0}$ for $t > 0$ is defined to be ∞. In particular, we may extend (6) to the case where $C = \emptyset$ and then ρ_e, ρ_v are defined to be 0.

Convenient characterizations. A convenient characterization of an amortized cut is given in the following proposition, whose proof is straightforward. (We will use this characterization in Section 4.)

Proposition 1 *A cut (U_1, U_2) with $|U_1| \leq |U_2|$ is ρ-amortized if and only if for every $C \subset U$ with $|C| \leq \alpha |U|$ and $F = U \setminus C$,*

$$e(C_1, C_2) \leq \rho \cdot \max\Big\{ e(C_1, F_1)\ ,\ \tfrac{|C_1|}{|C|} \cdot e(C, F) \Big\}$$

where $C_i = U_i \cap C$ *and* $F_i = U_i \cap C$ *for* $i = 1, 2$.

The restriction $|C| \leq \alpha|U|$ implies that the two terms $r(C) = \frac{e(C,F)}{\min\{|C|,|F|\}}$ and $r'(C) = \frac{e(C,F)}{|C|}$ differ by no more than a constant factor. Indeed, $\min\{|C|, |F|\} = \Theta(|C|)$ and hence $r(C) = \frac{e(C,F)}{\min\{|C|,|F|\}} = \frac{e(C,F)}{\Theta(|C|)} = \Theta(r'(C))$.

We can therefore characterize the amortized cost of a cut (up to constant factors) in terms of $r(C)$ rather than $r'(C)$. (We will use this characterization in Section 3).

Proposition 2 *A cut* (U_1, U_2) *with* $|U_1| \leq |U_2|$ *is* $O(\rho)$*-amortized if for every partition* $U = C \cup F$ *with* $0 < |C| \leq \alpha|U|$*,*

$$\min\left\{\frac{e(C_1, C_2)}{e(C_1, F_1)}, \frac{e(C_1, C_2)}{|C_1| \cdot r(C)}\right\} \leq \rho \tag{7}$$

where $C_i = U_i \cap C$ *and* $F_i = U_i \cap C$ *for* $i = 1, 2$.

Remarks. Observe that without the restriction $|C| \leq \alpha|U|$, the amortized cost ρ might be required to be $\Omega(|U|)$, a value that is too high for our intended application. For example, consider a clique on n vertices and a cut (U_1, U_2) in it with $|U_1| \leq |U_2|$. Let one vertex of U_2 be the only free vertex, and the rest of the vertices be charged. The number of charged-charged edges cut is $|U_1| \cdot \Theta(n)$. There are no charged-free edges in U_1, so the amortized cost for the edges is $\rho_e = \infty$. The number of charged vertices in the smaller side is $|U_1|$ and $r'(C) = \frac{n-1}{n-1} = 1$, so the amortized cost for the vertices is $\rho_v = \frac{|U_1|\Theta(n)}{|U_1| \cdot 1} = \Theta(n)$. Therefore, the amortized cost of any cut would be $\rho = \Omega(n)$.

In contrast, we show that the restriction $|C| \leq \alpha|U|$ allows to obtain relatively small values of ρ. Namely, there always exists a cut whose amortized cost is $\rho = O(1)$, and a cut whose amortized cost is $O(\log|U|)$ can be computed efficiently. We remark that our constructions are stronger than those required by Proposition 2, as they satisfy (7) with no restriction on $|C|$. (The point is that we use $r(C)$ rather than $r'(C)$, which makes a significant difference when $|C| \gg |F|$, as in the above clique example.)

Note that the amortized cost ρ is not an approximation ratio. On the one hand, it is not clear from the definition that every graph has an $O(1)$-amortized cut. On the other hand, the amortized cost of a cut may be smaller than 1, as demonstrated by a graph that consists of two cliques of size $n/2$ connected by an edge. The cut that separates the two cliques can be seen to have amortized cost $O(1/n)$.

3 Finding an amortized cut

In this section we devise an algorithm for finding $O(\log n)$-amortized cuts in general graphs, and $O(1)$-amortized cuts in graphs excluding any fixed minor (e.g. planar graphs). The input graph for this algorithm is denoted by G (though it may be just a part of the input graph for bisection). We assume that G is connected, as otherwise we can separate a connected component while cutting no edges at all.

Section 3.1 shows that every optimal min-ratio cut is an $O(1)$-amortized cut. It follows that in every graph there exists an $O(1)$-amortized cut. An optimal min-ratio cut is NP-hard to find in general graphs, and we thus consider approximate min-ratio cuts.

Section 3.2 demonstrates an approximate min-ratio cut which would be a poor divide step for our accounting method. In particular, its amortized cost is high, showing that the arguments of Section 3.1 do not immediately extend from optimal min-ratio cuts to approximate ones.

Section 3.3 presents an algorithm that uses a τ-approximate min-ratio cut in order to find an $O(\tau)$-amortized cut. Known algorithms for the min-ratio cut problem in general graphs [LR99, AR98, LLR95] have approximation ratio $\tau = O(\log n)$, and we can thus find an $O(\log n)$-amortized cut. For certain graph families a better approximation ratio is possible. For example, in graphs excluding any fixed minor, a ratio of $\tau = O(1)$ is known due to [KPR93], and we can thus find an $O(1)$-amortized cut.

3.1 Min-ratio cuts are $O(1)$-amortized

We give an $O(1)$ upper bound on the amortized cost of optimal min-ratio cuts. The proof is based on the characterization given in Proposition 2 for an amortized cut. We remark that our proof satisfies (7) with no restriction on $|C|$.

Lemma 3 *An optimal min-ratio cut in a graph is $O(1)$-amortized.*

Proof. Let (V_1, V_2) be an optimal min-ratio cut in a graph G, and assume, without loss of generality, that $|V_1| \le |V_2|$. Let $V = C \cup F$ be an arbitrary partition of the graph vertices to charged vertices C and free vertices F, with $0 < |C| < |V|$, and denote $C_i = V_i \cap C$ and $F_i = V_i \cap F$ for $i = 1, 2$ (see also Fig. 2). We show below that

$$\min\left\{\frac{e(C_1, C_2)}{e(C_1, F_1)}\,,\ \frac{e(C_1, C_2)}{|C_1| \cdot r(C)}\right\} \le 2, \tag{8}$$

and then by Proposition 2 we will have that (V_1, V_2) is $O(1)$-amortized, which proves the lemma. Note that we can assume that $|C_1| > 0$, as otherwise there is nothing to prove.

One easy case is when $\frac{e(C_1,C_2)}{e(C_1,F_1)}$ (i.e. the amortized cost for the edges ρ_e) is at most 2, which clearly implies (8).

Another easy case is when $\frac{e(C_1,C_2)}{|C_1|} \le 2r(V_1)$. Since (V_1, V_2) is an optimal min-ratio cut, we also have that $r(V_1) \le r(C)$. We obtain that $\frac{e(C_1,C_2)}{|C_1| \cdot r(C)} \le 2\frac{r(V_1)}{r(C)} \le 2$, and therefore (8) holds.

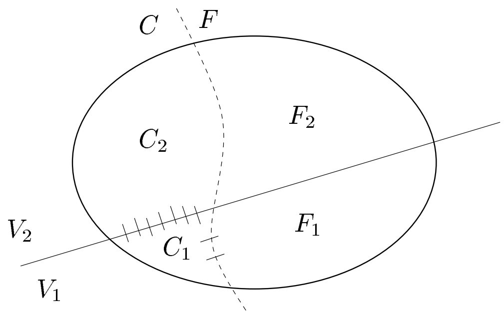

Figure 2: The amortized cost of an optimal min-ratio cut (V_1, V_2)

We next prove that one of the two easy cases above must hold, as otherwise we must have that $r(F_1) < r(V_1)$, in contradiction with (V_1, V_2) being an optimal min-ratio cut. Indeed, assume that $e(C_1, C_2)/e(C_1, F_1) > 2$ and $\frac{e(C_1,C_2)}{|C_1|} > 2r(V_1)$. Since $r(V_1) = \frac{e(V_1,V_2)}{|V_1|}$ is the average degree from V_1 to V_2, it can be represented as the following convex combination of the average degree from C_1 to V_2 and the average degree from F_1 to V_2, namely

$$r(V_1) = \frac{|F_1|}{|V_1|} \cdot \frac{e(F_1, V_2)}{|F_1|} + \frac{|C_1|}{|V_1|} \cdot \frac{e(C_1, V_2)}{|C_1|}.$$

Since $r(F_1) = \frac{e(F_1,V_2)+e(F_1,C_1)}{|F_1|}$ (note that $|F_1| \le |V_1| \le \frac{1}{2}|V|$), we can represent $r(V_1)$ also as

$$r(V_1) = \frac{|F_1|}{|V_1|} \cdot r(F_1) + \frac{|C_1|}{|V_1|} \cdot \left[\frac{e(C_1, V_2) - e(F_1, C_1)}{|C_1|} \right].$$

By the above two assumptions (that exclude the easy cases) we have that

$$\frac{e(C_1, V_2) - e(F_1, C_1)}{|C_1|} \ge \frac{e(C_1, C_2) - e(F_1, C_1)}{|C_1|} \ge \frac{\frac{1}{2}e(C_1, C_2)}{|C_1|} > r(V_1).$$

The last two inequalities imply that

$$r(V_1) > \frac{|F_1|}{|V_1|} \cdot r(F_1) + \frac{|C_1|}{|V_1|} \cdot r(V_1).$$

We obtained that some convex combination of $r(F_1)$ and $r(V_1)$ is smaller than $r(V_1)$, and we can therefore conclude that $r(F_1) < r(V_1)$. This contradicts the fact that (V_1, V_2) is an optimal min-ratio cut, and completes the proof of Lemma 3. $\square$

The converse of Lemma 3 is not true, and an $O(1)$-amortized cut can be an $\Omega(n)$-approximate min-ratio cut, as follows from the next proposition with $t = O(1)$.

Proposition 4 *Fix a constant $1/2 < \alpha < 1$ for the definition of an amortized cut. Then for every $t = o(n)$, there is an $O(1/t)$-amortized cut which is an $\Omega(n/t)$-approximate min-ratio cut.*

Proof. Consider the a graph on n vertices, for a sufficiently large n, that consists of three cliques as follows. V_1 is a clique on t vertices, V_2 is a clique on αn vertices, and V_3 is a clique on the remaining $\Omega(n)$ vertices. In addition, the graph contains one edge connecting V_1 to V_2, and one edge connecting V_2 to V_3.

The cut $(V_1, V_2 \cup V_3)$ has amortized cost $O(1/t)$. Indeed, let $C \cup F$ be a partition of the vertices with $|C| \leq \alpha n$. We may assume that C contains both endpoints of the edge between V_1 and V_2, as otherwise the cut contains no charged-charged edges and its amortized cost is 0. So we have that the cost of the charged-charged edges cut is 1, and that both V_1 and V_2 contain at least one charged vertex. If V_1 contains also at least one free vertex, then the number of charged-free edges in V_1 is at least $t-1$ and hence $\rho_e = \frac{e(C_1,C_2)}{e(C_1,F_1)} \leq 1/(t-1)$. Otherwise, we have $C_1 = V_1$; since there are at most αn charged vertices, and at least one of them is in V_1, we have that V_2 contains also free vertices and thus $e(C,F) \geq \Omega(n)$; it follows that $\rho_v = \frac{e(C_1,C_2)}{e(C,F)} \cdot \frac{|C|}{|C_1|} \leq O(1/t)$.

The cut $(V_1, V_2 \cup V_3)$ is an $\Omega(n/t)$-approximate min-ratio cut. Indeed, the ratio of this cut is $r(V_1) = 1/t$, while the cut $(V_3, V_1 \cup V_2)$ is an optimal min-ratio cut and has ratio $r(V_3) = O(1/n)$. □

The next corollary follows from Lemma 3.

Corollary 5 *In every graph there exists an $O(1)$-amortized cut.*

Corollary 5 is optimal up to constant factors, and there are graphs for which any cut has amortized cost $\Omega(1)$. For example, consider a clique on n vertices. Given a cut (V_1, V_2) with $|V_1| \leq |V_2|$, let α be the constant in the amortized cut definition, and take $(\alpha - 1/2)n$ vertices of V_2 and all of V_1 to be the charged vertices. It can be seen that $\rho_e = \infty$ and $\rho_v = \Theta(1)$, and so the amortized cost of the cut (V_1, V_2) is $\Omega(1)$, as claimed.

3.2 Approximate min-ratio cuts might be poor amortized cuts

We demonstrate that an approximate min-ratio cut of a graph might be a poor divide step, and in particular a poor amortized cut. Consider, for example, the following graph G on $2n + 2\sqrt{\epsilon n}$ vertices for a fixed $0 < \epsilon < 1$ (see also Fig. 3). The vertex set of the graph is $F_1 \cup F_2 \cup C_1 \cup C_2$ where each of F_1, F_2 are of size n, each of C_1, C_2 are of size $\sqrt{\epsilon n}$, and each of the four subsets forms a clique. These four cliques are connected as follows. Between F_1 and F_2 there are n edges that form a matching (i.e. have no common endpoint). Between C_1 and C_2 there are all possible ϵn edges, thus $C_1 \cup C_2$ forms a clique. There are also $2\sqrt{\epsilon n}$ edges between F_i and C_i (for $i = 1, 2$) so that their endpoints at F_i are distinct and each vertex of C_i is an endpoint of exactly two of these edges.

Let $C = C_1 \cup C_2$ be the charged vertices, and $F = F_1 \cup F_2$ the free vertices. Such a partition to charged and free may reflect the "right" cut of $2\sqrt{\epsilon n}$ vertices from the graph G (if, e.g., the input graph for bisection consists of this graph G and a clique on $2n - 2\sqrt{\epsilon n}$ vertices).

Consider a divide step based on the cut $(F_1 \cup C_1, F_2 \cup C_2)$, whose ratio is nearly optimal. Indeed, an optimal min-ratio cut in this graph is $(F_1, C_1 \cup F_2 \cup C_2)$ and its ratio is $1 + 2\sqrt{\epsilon}/\sqrt{n}$.

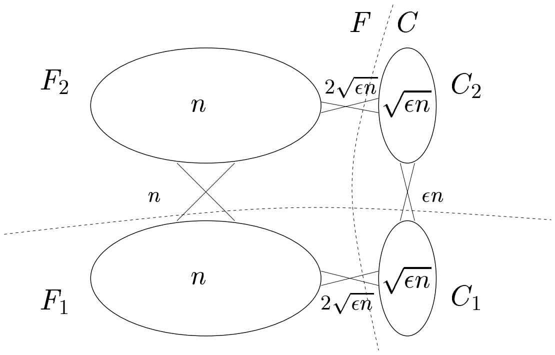

Figure 3: A poor divide step by an approximate min-ratio cut

The cut $(F_1 \cup C_1, F_2 \cup C_2)$ has a slightly higher ratio of $(1+\epsilon)(1-o(1))$, and so it is a $1+\epsilon$ approximate min-ratio cut.

Observe that the cut $(F_1 \cup C_1, F_2 \cup C_2)$ is a poor divide step. It cuts ϵn charged-charged edges while the total number of charged-free edges in G (and the bisection cost in the input graph) is only $4\sqrt{\epsilon n}$. According to the new accounting method, such a divide step does not give an approximation ratio better than $\Omega(\sqrt{\epsilon n})$.

The observation that the cut $(F_1 \cup C_1, F_2 \cup C_2)$ is a poor divide step is supported by its high amortized cost. The amortized cost for the edges is $\rho_e = \epsilon n / 2\sqrt{\epsilon n} = \sqrt{\epsilon n}/2$. The ratio of the cut (C, F) is $r(C) = r'(C) = 2$, so the amortized cost for the vertices is $\rho_v = \epsilon n / (\sqrt{\epsilon n} r'(C)) = \sqrt{\epsilon n}/2$. We conclude that a $1+o(1)$ approximate min-ratio cut might have amortized cost $\rho \geq \min\{\rho_e, \rho_v\} = \sqrt{\epsilon n}/2$.

3.3 Finding $O(\tau)$-amortized cut

We present an algorithm that finds an $O(\tau)$-amortized cut, given a subroutine for computing a τ-approximate min-ratio cut. The algorithm is motivated by the $O(1)$ upper bound on the amortized cost of a min-ratio cut shown in Section 3.1. In particular, we examine what additional properties are required in order to extend the analysis of Lemma 3 from optimal min-ratio cuts to approximate ones.

The proof of Lemma 3 uses *twice* the fact that (V_1, V_2) is an optimal min-ratio cut. In the first usage we had that $\frac{e(C_1,C_2)}{|C_1| \cdot r(C)} \leq 2\frac{r(V_1)}{r(C)} \leq 2$, which extends to the case where (V_1, V_2) is an approximate min-ratio cut with the approximation ratio carried over to the amortized cost, i.e. if (V_1, V_2) is a τ-approximate min-ratio cut then we have $\frac{e(C_1,C_2)}{|C_1| \cdot r(C)} \leq 2\frac{r(V_1)}{r(C)} \leq 2\tau$.

The second time we used the fact that (V_1, V_2) is an optimal min-ratio cut was to say that $r(F_1) < r(V_1)$ cannot hold and gives a contradiction. In general, this usage does not extend to an approximate min-ratio cut, as demonstrated by the example in Section 3.2. However, the proof does extend to an approximate min-ratio cut if we have the additional property that the ratio of V_1 is minimal over all its subsets F_1, i.e. $r(V_1) \leq r(F_1)$ for all $F_1 \subset V_1$. We therefore obtain that the proof of Lemma 3 extends to approximate min-ratio cuts as follows.

Lemma 6 *Let (V_1, V_2) be a τ-approximate min-ratio cut in a graph, with $|V_1| \leq |V_2|$. If $r(V_1) \leq r(F_1)$ for every $F_1 \subset V_1$ then (V_1, V_2) is an $O(\tau)$-amortized cut.*

Note that the proof of Lemma 6 is not symmetric with respect to the two amortization methods. It guarantees that either $e(C_1, C_2)/e(C_1, F_1) \leq 2$ (i.e. the amortized cost for the edges ρ_e is at most 2), or $\frac{e(C_1,C_2)}{|C_1| \cdot r(C)} \leq 2\tau$ (i.e. the amortized cost for the vertices ρ_v is $O(\tau)$). In contrast, in the proof of Lemma 3 for optimal min-ratio both amortization costs are $O(1)$.

The amortized cut algorithm. We use Lemma 6 to devise an algorithm that finds an $O(\tau)$-amortized cut based on a τ-approximate min-ratio cut. The algorithm, described in Fig. 4, starts with a τ-approximate min-ratio cut (V_1, V_2) and then "fixes" it so that it would also be "minimal" with respect to containment, as required by Lemma 6. It then follows that the output cut is $O(\tau)$-amortized.

In order to "fix" the cut (V_1, V_2), the algorithm uses minimum (s,t)-cuts in a related graph G', which is defined in step 2. The related graph G' contains edges of the input graph G, as well as new edges. The edges from G have unit capacity, while the capacity of the new edges is some parameter $p > 0$. Step 3 then finds the optimal value of p with respect to the minimum (s,t)-cut. Before discussing implementation issues of step 3, let us analyze the algorithm correctness.

Algorithm FINDAMORTIZED.

1. Find in the input graph $G = (V, E)$ a τ approximate min-ratio cut (V_1, V_2) with $|V_1| \leq |V_2|$.
2. Create a related graph G':

   – Merge all vertices of V_2 into a single vertex t, removing self loops at t, and keeping all edges to V_1, including parallel edges.

   – Add a new vertex s which is connected to each vertex of V_1 by an edge whose capacity (weight) is a parameter $p > 0$.
3. Let S denote the vertices of V_1 which are on the same side with s in a minimum (s,t)-cut of G'.

   – Find (e.g. by binary search) the minimum $p > 0$ for which $S \neq \emptyset$. (Possibly, $S = V_1$).
4. Output the cut $(S, V \setminus S)$ of the input graph.

Figure 4: Algorithm for amortized cuts

Lemma 7 *The cut $(S, V \setminus S)$ output by algorithm* FINDAMORTIZED *is a τ-approximate min-ratio cut. In addition, every nonempty subset of V_1 has ratio at least as large as S, i.e. $r(S) = \min\{r(S') : \emptyset \neq S' \subseteq V_1\}$.*

Proof. Consider an arbitrary value p and an arbitrary (s,t)-cut in the related graph G' with the corresponding set $S \subset V_1$ (see Fig. 5). The cut consists of (i) edges between s and $V_1 \setminus S$ (each of capacity p) (ii) edges between S and $V_1 \setminus S$ (these are edges from the input graph G) and (iii) edges between S and t (these are the edges between S and V_2 in the input graph G). The capacity of this (s,t)-cut is thus

$$\mathrm{cap}(S) = p \cdot |V_1 \setminus S| + e(S, V \setminus S)$$

where, as usual, $e(\cdot,\cdot)$ denotes the number of corresponding edges in the input graph G. In the special case of the empty set $S = \emptyset$, the capacity of the (s,t)-cut is

$$\mathrm{cap}(\emptyset) = p \cdot |V_1|$$

Fixing the value of p, let us compare the capacity of the cut defined by the empty set $\emptyset$ with that of an arbitrary set $S \neq \emptyset$, i.e. $\mathrm{cap}(\emptyset)$ vs. $\mathrm{cap}(S)$. The empty set $\emptyset$ yields a smaller capacity whenever

$$\begin{array}{rcl} p \cdot |V_1| & < & p \cdot |V_1 \setminus S| + e(S, V \setminus S) \\ & \Updownarrow & \\ p & < & \dfrac{e(S, V \setminus S)}{|S|} = r(S) \end{array}$$

where $r(S)$ is the ratio of the cut $(S, V \setminus S)$ in the input graph G (note that $|S| \leq |V_1| \leq \frac{1}{2}|V|$ and that $r(S) > 0$ if G is connected).

We claim that the value of p found at step 3 is essentially $p^* = \min\{r(S) : \emptyset \neq S \subseteq V_1\}$. Indeed, when $p < p^*$, a minimum (s,t)-cut in G' corresponds to $S = \emptyset$, and when $p > p^*$, a minimum (s,t)-cut yields a set $S \neq \emptyset$. When $p = p^*$, a minimum (s,t)-cut can be obtained either by $S = \emptyset$, or by (one or more) $S \neq \emptyset$ with $r(S) = p^*$.

When $p = p^* + \epsilon$ for a very small $\epsilon > 0$, only the sets $S \neq \emptyset$ with $r(S) = p^*$ give smaller capacity than the empty set, and thus a minimum (s,t)-cut is obtained by one of these sets S. By the definition of p^*, this set $\emptyset \neq S \subset V_1$ has minimal ratio $r(S)$ over all nonempty subsets of V_1, i.e. $r(S) = \min\{r(S') : \emptyset \neq S' \subseteq V_1\}$, as claimed. Furthermore, since $S = V_1$ is

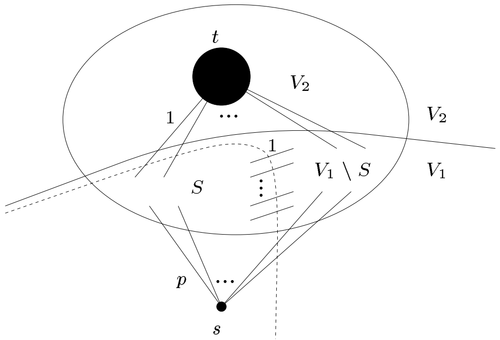

Figure 5: An (s,t)-cut in the related graph G'

included in this range, we get that $r(S) \leq r(V_1)$ and hence $(S, V \setminus S)$ is a τ-approximate min-ratio cut, finishing the proof. We remark that a slightly modified algorithm can guarantee in addition that $r(S) < r(S')$ for every $S' \subset S$ with $S' \neq \emptyset, S$. Details omitted. $\square$

Theorem 3 *Given a subroutine for computing a τ-approximate min-ratio cut, algorithm* FindAmortized *finds an $O(\tau)$-amortized cut.*

Proof. Lemma 7 guarantees that the cut found by the algorithm satisfies the requirements of Lemma 6, from which it follows that the cut is $O(\tau)$-amortized. $\square$

We now address the issue of implementing step 3. Observe that p^* is the maximum value p for which the empty set $\emptyset$ gives a minimum (s,t)-cut. Since, by definition, p^* is the ratio $r(S)$ of a set S, it has only n^3 possible values, which can be exhaustively searched. Alternatively, p^* can be found in $O(\log n)$ iterations of binary search, since as an exact multiple of $1/|S|$ it is bounded between 0 and n, and the difference between any two of its possible values is more than $1/n^2$.

Once we find p^*, we need to find a set $S \neq \emptyset$ that gives a minimum (s,t)-cut for p^*. We can either guess a vertex of V_1 and merge it with s before computing the minimum (s,t)-cut for p^*, or alternatively compute a minimum (s,t)-cut for $p = p^* + \epsilon$ with e.g. $\epsilon = 1/n^2$.

4 The bisection algorithm

In this section we describe our approximation algorithm for bisection and prove the following theorem. (See Section 2 for the definition of an amortized cut.)

Theorem 4 *Given a subroutine that finds a ρ-amortized cut, a bisection within ratio of $1 + O(\rho \log n)$ of the minimum can be found in polynomial time.*

4.1 Decomposition stage

The decomposition stage recursively divides the input graph $G = (V, E)$ into smaller and smaller parts using a ρ-amortized cut subroutine (e.g. the one devised in Section 3). Each part is further divided unless it consists of a single vertex.

The decomposition stage builds a rooted binary tree T, called the *decomposition tree*, which corresponds to the recursive decomposition of the input graph G in a natural way, as follows. (Throughout, we call the vertices of T *nodes*, to avoid confusion with the vertices of the input graph G.) Each tree node i contains a part $V_i \subseteq V$ that was found during the recursive decomposition. The root node of T contains V, i.e. the whole input graph G. Let us denote the two children of a nonleaf node i by $L(i)$ and $R(i)$. Then their two parts $V_{L(i)}, V_{R(i)}$ are the result of dividing V_i, i.e. the ρ-amortized cut found in V_i is $(V_{L(i)}, V_{R(i)})$. A leaf of the tree T contains a part that consists of a single vertex of G. Therefore T contains exactly n leaves and $n - 1$ nonleaf nodes.

4.2 Labeling stage

Recall the following definitions from Section 2. A *labeling* of the decomposition tree T labels each nonleaf node of the tree as either white or black. Fixing a parameter $1/2 < \alpha < 1$, we say that a labeling is *α-consistent* with respect to a white-black bisection (W, B) of G if every tree node i satisfies that: If the label of node i is white then $|W \cap V_i| \leq \alpha|V_i|$, and if the label of node i is black then $|B \cap V_i| \leq \alpha|V_i|$ (where V_i is the part contained in node i). A labeling is called *opt-consistent* if it is α-consistent with the fixed optimal bisection (W^*, B^*).

The labeling stage produces a family $\mathcal{F}$ of labelings. The cardinality of $\mathcal{F}$ is exponential in n, so rather than listing its members explicitly, the labeling stage produces an implicit representation of $\mathcal{F}$. The actual work of the labeling stage is to *mark* certain nodes of T, and these nodes implicitly define the family $\mathcal{F}$, as described below.

The labeling stage marks some of the nodes of T in a process that goes from the root of T towards its leaves, as follows. The root of T is always marked, and any other node i in the tree is marked in this process if its closest marked ancestor j satisfies $|V_i| \leq \frac{1}{2\alpha}|V_j|$ (as before, V_i and V_j are the parts contained in the nodes i and j, respectively). Note that the constant α is chosen so that $\frac{1}{2} < \alpha < 1$, implying $\frac{1}{2} < \frac{1}{2\alpha} < 1$.

A labeling of T is said to be *derived* from the marked nodes, if the label of every unmarked node is the same as the label of its closest marked ancestor (there is no restriction on the labels of the marked nodes). Note that in this case the labels of the marked nodes uniquely define the labels of all the internal tree nodes.

The family $\mathcal{F}$ produced by the labeling stage consists of all the labelings that can be derived from the marked nodes. Since each of the $\Omega(n)$ marked nodes can be labeled arbitrarily by one of two colors, the resulting family of labelings has exponentially large cardinality, and we cannot explicitly list all the family members. Instead, the algorithm implicitly represents this family $\mathcal{F}$ by identifying which are the marked nodes.

Lemma 8 *The family of labelings $\mathcal{F}$ contains at least one opt-consistent labeling.*

Proof. Let the white-black cut (W, B) be the fixed optimal bisection. Consider the labeling that is derived from the marked nodes, with the label of each marked node i being the color in minority among the vertices of V_i.

This labeling is clearly in the family $\mathcal{F}$, and we claim that it is also opt-consistent. Indeed, the label of a marked node i is by definition the minority color in V_i. The label of an unmarked node i is the same as the label of its closest marked ancestor j. Suppose, without loss of generality, that this label (of i and j) is white. Then at most half the vertices of V_j are white, i.e. $|W \cap V_j| \leq \frac{1}{2}|V_j|$. Observe that $V_i \subset V_j$ and $|V_i| > \frac{1}{2\alpha}|V_j|$ and hence $|W \cap V_i| \leq |W \cap V_j| \leq \frac{1}{2}|V_j| < \alpha|V_i|$. Hence, this labeling of $\mathcal{F}$ is opt-consistent. □

4.3 The charge of a bisection

We now formally define the *charge of a bisection* (W, B) with respect to the decomposition tree T and a labeling of it. The reference to T will later be omitted, as we always refer to the tree computed in the decomposition stage.

Definition (Charge). Let (W, B) be a bisection of the input graph, and assume we are given a decomposition tree T and a labeling of it. For each (nonleaf) node i of T, if i is labeled white then we let (see Fig. 6) $C_i = W \cap V_i$ and $F_i = B \cap V_i$, and if i is labeled black then we let $C_i = B \cap V_i$ and $F_i = W \cap V_i$. We obtain a cut (C_i, F_i) of the part V_i, and say that C_i is *charged* and F_i is *free*. The *charge of the divide step* of a (nonleaf) node i is defined as

$$e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)}).$$

The *charge of the bisection* (W, B) is defined as the sum of all the divide steps charges, i.e.

$$\sum_{i \in T} e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)}).$$

(These charges are defined with respect to T and a labeling of it.)

Bisection charge vs. cost

In certain conditions, a bisection charge can approximate its cost. As shown below, the charge of a bisection upper bounds its cost, and the gap between them is not too large if the charge is taken with respect to an α-consistent labeling (as in the case of the fixed optimal bisection and an opt-consistent labeling).

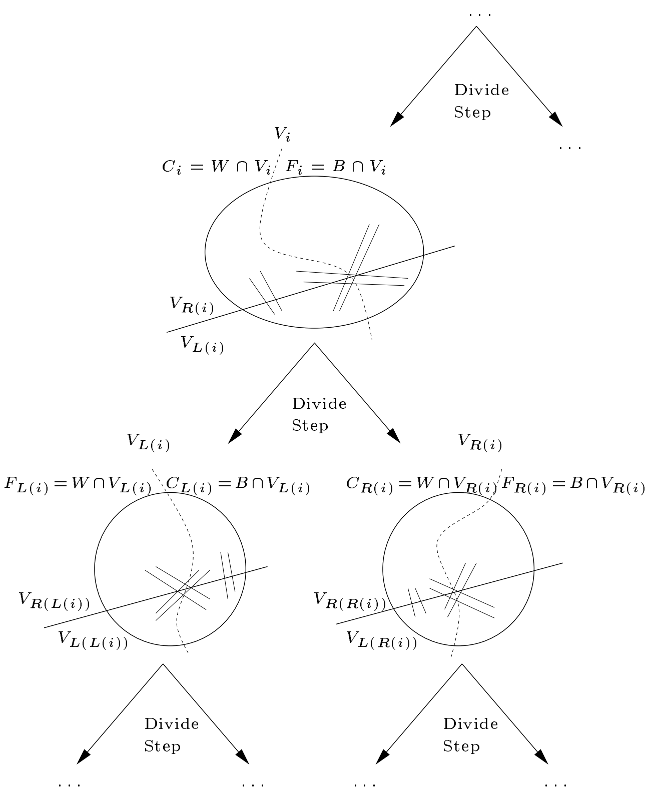

Figure 6: The charge of a bisection (W, B) throughout the decomposition tree

Lemma 9 *The charge of a bisection (W, B) with respect to any labeling is at least as large as its cost.*

Proof. As we have seen in section 2, the true cost of the (W, B) edges cut in a divide step i is $e(C_i \cap V_{L(i)}, F_i \cap V_{R(i)}) + e(C_i \cap V_{R(i)}, F_i \cap V_{L(i)})$, and is therefore not larger than the charge of this step. The proof follows by summing over all divide steps, since the decomposition stage eventually divides the graph into individual vertices, and so every edge of the bisection (W, B) is cut at some divide step. $\square$

Lemma 10 *The charge of a bisection (W, B) with respect to a labeling that is α-consistent with it is at most $e(W, B) \cdot (1 + O(\rho \log n))$.*

Proof. Consider a bisection (W, B) and a labeling of T that is α-consistent with it. As we have seen in Section 2 and in Lemma 9 the charge of a divide step is larger than the true cost of the (W, B) edges cut in that step by the cost of the charged-charged edges cut in that divide step. Summing over the divide steps we get that the charge of (W, B) the fixed optimal bisection is larger than its cost by $2 \sum_i e(C_i \cap V_{L(i)}, C_i \cap V_{R(i)})$, where i ranges over all (nonleaf) nodes i in T. We use the shorter notation $C_L = C_i \cap V_{L(i)}$ and $C_R = C_i \cap V_{R(i)}$, where i is clear from the context.

To upper bound $2\sum_i e(C_L, C_R)$, observe that each part V_i is divided using a ρ-amortized cut, and that the α-consistent labeling guarantees that $|C_i| \le \alpha|V_i|$ for all nodes i, so we can use the amortization scheme of Section 2. Namely, let us assume, without loss of generality, that the decomposition stage places in the left child of a node i the smaller of the two subparts of V_i, i.e. $|V_{L(i)}| \le |V_{R(i)}|$ for every nonleaf node i. Then by Proposition 1 we can upper bound

$$e(C_L, C_R) \le \rho \cdot \max\left\{e(C_L, F_L)\ ,\ \tfrac{|C_L|}{|C_i|} \cdot e(C_i, F_i)\right\},$$

and obtain

$$2\sum_i e(C_L, C_R) \le 2\rho \cdot \left\{\sum_i e(C_L, F_L) + \sum_i \tfrac{|C_L|}{|C_i|} \cdot e(C_i, F_i)\right\} \ . \tag{9}$$

Therefore, to complete the proof of Lemma 10 it suffices to upper bound the sums in the curly brackets (i.e. the total cost amortized in each of the two methods) by $e(W, B) \cdot O(\log n)$.

Consider first $\sum_i e(C_L, F_L)$. The edges that contribute to this sum are charged-free edges and hence edges of the bisection (W, B). An edge in the cut (C_L, F_L) must be inside $V_{L(i)}$, the smaller side of the cut of V_i, and any single edge can be inside $V_{L(i)}$ in at most $\log n$ divide steps i throughout the tree T. Hence, $\sum_i e(C_L, F_L)$ consists of at most $\log n$ times the cost of every edge of the bisection (W, B), and therefore this sum is at most $e(W, B) \cdot \log n$.

Consider next $\sum_i \frac{|C_L|}{|C_i|} \cdot e(C_i, F_i)$, and recall our convention that $\frac{0}{0}$ is defined to be 0. The edges of $e(C_i, F_i)$ contribute to the sum their cost scaled by a factor of $\frac{|C_L|}{|C_i|}$. Each edge of $e(C_i, F_i)$ is a charged-free edge and hence an edge of the bisection (W, B). However, an edge of the bisection (W, B) belongs to $e(C_i, F_i)$ if and only if this edge is inside V_i. The nodes i for which this edge is inside V_i are all on a path from the root to a leaf of the decomposition tree T, and therefore the total contribution of this edge is at most its cost scaled by the sum of $\frac{|C_L|}{|C_i|}$ over that path in T.

We claim that the sum of $\frac{|C_L|}{|C_i|}$ over any path from the root to a leaf is bounded by $O(\log n)$. It follows from this claim that $\sum_i \frac{|C_L|}{|C_i|} \cdot e(C_i, F_i)$ can be described as the cost of every edge of the bisection (W, B) scaled by at most $O(\log n)$, and therefore this sum is at most $e(W, B) \cdot O(\log n)$.

To prove the claim, consider an arbitrary path from the root to a leaf, and denote the path nodes by $1, 2, \ldots, p+1$. At each node i the charged side (i.e. C_i) may be either W or B, depending on the label of the node, so denoting $w_j = |W \cap V_j|$ and $b_j = |B \cap V_j|$, we have that $\frac{|C_L|}{|C_i|}$ is either $\frac{w_{L(i)}}{w_i}$ or $\frac{b_{L(i)}}{b_i}$, and clearly at most their sum. Hence,

$$\sum_{i=1}^{p} \frac{|C_L|}{|C_i|} \le \sum_{i=1}^{p} \frac{w_{L(i)}}{w_i} + \sum_{i=1}^{p} \frac{b_{L(i)}}{b_i}$$

Consider first $\sum_1^p \frac{w_{L(i)}}{w_i}$, and observe that w_i is a nonincreasing sequence, since in the tree, node i is a parent of node $i+1$. If node $i+1$ is a left child (of its parent node i), then $w_{L(i)} = w_{i+1}$ and hence $\frac{w_{L(i)}}{w_i} = \frac{w_{i+1}}{w_i} \le 1$. The number of such nodes i is at most $\log n$, since the path from the root to a leaf can contain at most $\log n$ left children i (recall that $|V_{L(i)}| \le |V_{R(i)}|$). The contribution of all such nodes i to $\sum_1^p \frac{w_{L(i)}}{w_i}$ is therefore at most $\log n$.

If node $i+1$ is a right child (of its parent i), then $w_{L(i)} = w_i - w_{i+1}$, and the contribution of all such nodes i is at most $\sum_1^p \frac{w_i - w_{i+1}}{w_i}$. Clearly, $\frac{w_i - w_{i+1}}{w_i} \leq \frac{1}{w_i} + \ldots + \frac{1}{w_{i+1}+1}$ and hence the contribution of all such nodes i to $\sum_1^p \frac{w_{L(i)}}{w_i}$ is at most $\sum_1^p \frac{w_i - w_{i+1}}{w_i} \leq \frac{1}{w_1} + \ldots + \frac{1}{2} + 1 = H(w_1) \leq H(n)$ where $H(k) = \sum_1^k \frac{1}{j}$ is the k-th harmonic number.

We conclude that $\sum_1^p \frac{w_{L(i)}}{w_i} \leq \log n + H(n) \leq O(\log n)$. Similarly, $\sum_1^p \frac{b_{L(i)}}{b_i} = O(\log n)$, and together we get that $\sum_1^p \frac{|C_L|}{|C_i|} \leq O(\log n)$, proving the claim and the lemma. □

Corollary 11 *The charge of the fixed optimal bisection (W^*, B^*) with respect to an opt-consistent labeling is at most $b(1 + O(\rho \log n))$.*

Distributing charge to vertices

It will be convenient (algorithmically) to distribute the charge of a bisection (W, B) (with respect to T and a labeling) to the vertices of the input graph, as follows. For each vertex $v \in V_i$ let the *cross-degree* of v at node i, denoted $cross_i(v)$, be the cost of the edges that are incident at v and are cut in divide step i. We define the *charge of a vertex $v \in V$* as the sum of the cross-degree of v at all nodes i for which v belongs to the charged side, i.e. $\sum_{i:v \in C_i} cross_i(v)$. The next lemma proves that distributing the charge of a bisection to the graph vertices is indeed correct.

Lemma 12 *The charge of a bisection (W, B) is the sum of the charges of all vertices in G.*

Proof. The charge of a divide step of node i is equal to the sum of the cross-degrees at node i of all vertices $v \in V_i$, i.e.

$$e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)}) = \sum_{v \in C_i} cross_i(v) \ .$$

Summing over all nodes i in the tree T, the lefthand side is, by definition, the bisection charge, and the righthand side is the sum of the charges of all vertices in G. The proof follows. □

Distributing the charge to the vertices of G is important algorithmically. The charge of a vertex depends on (and can be easily computed from) the side of this vertex in the bisection (W, B), the decomposition tree T, and the labeling of T, but it does not depend on the side of the cut (W, B) that other vertices of the graph belong to. It follows that the charge of a bisection (W, B) with respect to a given decomposition tree T and a labeling of it, depends *linearly* on the placement of vertices into W and B. This formulation of charge will be exploited by (the dynamic programming in) the combining stage.

4.4 Combining stage

The combining stage computes a bisection of the input graph G and a labeling of the decomposition tree T, such that the bisection charge with respect to the labeling is at most $b \cdot (1 + O(\rho \log n))$. It then follows from Lemma 9 that the cost of the computed bisection is at most $b \cdot (1 + O(\rho \log n))$, as desired.

Consider first the case where an opt-consistent labeling is known. Then it suffices to compute a bisection of G whose charge with respect to this opt-consistent labeling is minimal, because Corollary 11 guarantees that the charge of the computed bisection is at most $b \cdot (1 + O(\rho \log n))$. Below we describe a simple procedure for finding a bisection of G with minimal charge with respect to a given labeling.

However, we do not know how to efficiently find an opt-consistent labeling, and therefore we go over all the labelings in the family $\mathcal{F}$. Specifically, using a more complicated procedure described below the combining stage finds a bisection of G and a labeling from $\mathcal{F}$, such that the charge of the bisection with respect to the labeling is minimal over all such bisection-labeling pairs. Lemma 8 guarantees that at least one of these labelings is opt-consistent, in which case Corollary 11 applies. Hence, the bisection-labeling pair computed by this procedure satisfies that the charge of the bisection with respect to the labeling is indeed at most $b \cdot (1 + O(\rho \log n))$.

Minimizing charge over a given labeling

Finding a bisection of minimum charge with respect to a given labeling is relatively straightforward. By Lemma 12, the charge of a bisection (W, B) is the sum of the vertex charges. Since the decomposition tree T and the labeling are fixed, the charge of a vertex depends only on its side in the bisection (W, B). We can therefore compute for each vertex v what is its charge when it belongs to W, called the *white charge* of v, and what is its charge when it belongs to B, called the *black charge* of v. (Note that summing the white charge and the black charge of a vertex gives the degree of that vertex in G.)

The charge of a bisection (W, B) is then the sum of the white charges of W and the black charges of B. To find a bisection (W, B) with minimum charge with respect to the given labeling, we can thus compute for each vertex its net-charge (white charge minus black charge), and take W to be the $n/2$ vertices with smallest net-charge. (This algorithm for the case where a labeling is given was used in the algorithm outline in Section 2, where we assumed that the labeling stage produces an opt-consistent labeling.)

Minimizing charge over the family $\mathcal{F}$

The combining stage uses dynamic programming to find a bisection and a labeling from the family $\mathcal{F}$, so that the charge of the bisection with respect to this labeling is minimum over all such bisection-labeling pairs.

The dynamic programming table Q has entries of the form $Q(i, k, g)$, where i is a node of the decomposition tree T, k is an integer between 0 and $|V_i|$, and g is a *guess list* that contains the labels of the marked ancestors of node i. Throughout, i is considered an ancestor of itself.

An entry $Q(i, k, g)$ in the table contains the optimal solution to the following problem: Choose k vertices of V_i and a labeling from $\mathcal{F}$ that agrees with g, so that when these k vertices are placed in the side W and the remaining vertices of V_i are placed in the side B, the sum of the charges of all the vertices of V_i with respect to the chosen labeling, is minimal over all such choices. Note that when we only consider labelings from the family $\mathcal{F}$

that agree with g, the labels of all the ancestors of i are uniquely defined from g, while the marked descendants of i can have arbitrary labels.

For a leaf node i, the table entry $Q(i,k,g)$ can be computed directly, as follows. Since i is a leaf node, the part V_i consists of a single vertex, say v, and k can be either 0 or 1. If $k=0$ then v is necessarily in B, and if $k=1$ then v is necessarily in W. The guess list g gives the labels of all the nodes on the path from the leaf i to the root, and hence all the labels that can possibly affect the charge of v. Since k and g uniquely define all the data that the charge of v depends on, $Q(i,k,g)$ is just the charge of v, and can be computed directly as $\sum_j cross_j(v)$ where j ranges over all ancestors of i whose label (according to g) agrees with the side of v (as follows from k).

For a nonleaf node i, the table entry $Q(i,k,g)$ can be efficiently computed from table entries of its children nodes $L(i), R(i)$. Indeed, choosing k vertices from V_i is equivalent to choosing j vertices from one child part $V_{L(i)}$ and $k-j$ vertices from the other child part $V_{R(i)}$, so we need to add up two entries, each corresponding to one child node. The optimal value of j is not known, but it can be exhaustively searched. The guess list g can be extended into lists g_L, g_R for the children nodes, in possibly more than one way. Therefore,

$$Q(i,k,g) = \min_{0\le j\le k} \min_{g_L,g_R} \{Q(L(i),j,g_L) + Q(R(i),k-j,g_R)\}$$

where g_L, g_R range over all possible extensions of g, as described below. If a child node $L(i)$ is a marked node, then there are two possible ways to extend the list g into a list g_L (by adding a label for $V_{L(i)}$), and the optimum $Q(i,k,g)$ is achieved by taking the one which is better. If a child node $L(i)$ is not a marked node, then the only extension is $g_L = g$, because i and $L(i)$ have the same marked ancestors. The possible extensions of the child node $R(i)$ are similar. It follows that each table entry of a nonleaf node i can be computed from table entries of its children $L(i), R(i)$ in time $O(|V_i|) = O(n)$.

To fill all the table entries, start from the entries that correspond to leaf nodes i and go upwards the decomposition tree T. In particular, the entries $Q(i_{root}, n/2, g)$ will be computed for the root node i_{root}. At the root node, the guess list g contains the label of the root, and thus has only two possible values. (In fact, the two entries must be the same due to symmetry.) The combining stage outputs $\min_g Q(i_{root}, n/2, g)$, which by definition, is the minimum charge of all bisections of the input graph with respect to any labelings from $\mathcal{F}$, as desired. A bisection that achieves this minimum charge can also be computed. Simply go over the table entries in the reversed order of computation, and recover at each entry the values of j, g_L, g_R that gave the optimum. Alternatively, associate with each entry $Q(i,k,g)$ a set of k vertices of V_i which is optimal for it, and its corresponding labels.

Lemma 13 *The combining stage finds in polynomial time a bisection of the input graph G and a labeling from the family $\mathcal{F}$, so that the charge of the bisection with respect to the labeling is minimal over all such bisection-labeling pairs.*

Proof. The above discussion shows that the algorithm correctly computes every entry $Q(i,k,g)$, and a bisection-labeling pair as desired.

The size of the table Q is polynomial in n. Indeed, there are only $O(n)$ tree nodes i. For each tree node i, the range of k contains $O(|V_i|) = O(n)$ possible values. In addition, at each tree node i the guess list g contains labels of at most $O(\log n)$ ancestor nodes, and thus g assumes polynomially many values. The polynomial bound on the size of the table Q follows.

An entry for a leaf nodes i is computed efficiently. An entry for a nonleaf node is efficiently computed from previously computed entries. By the upper bound on the table size we conclude that all the table entries are computed in polynomial time, and in particular $Q(i_{root}, n/2, g)$. □

Corollary 14 *The combining stage finds bisection of the input graph (and a labeling of T) such that bisection charge (with respect to the labeling) is at most $b(1 + O(\rho \log n))$.*

Proof. By Lemma 13 and Corollary 11 there exists a bisection of G and a labeling of $\mathcal{F}$ such that the bisection charge with respect to the labeling is at most $b(1 + O(\rho \log n))$. The proof then follows by applying Lemma 8. □

This corollary completes the proof of Theorem 4, since by Lemma 9 the charge of a bisection is an upper bound on its actual cost.

5 Extensions

Our results extend to several variants (and generalizations) of the minimum bisection problem, including the case of edges with arbitrary nonnegative costs (Section 5.1), the case of vertices with polynomially bounded nonnegative integer weights (Section 5.2), the variant that requires, in addition, to separate a given pair of vertices s and t (Section 5.3), the case of cutting away from the graph an arbitrary number of vertices (instead of $n/2$) that is given as part of the input (Section 5.4), the case of cutting the input graph into a fixed number of equal-size parts (Section 5.5), and the case of finding a 2/3-balanced cut whose cost is small relative to the minimum bisection cost b (Section 5.6).

In what follows, the *basic bisection problem* refers to the minimum bisection problem that was defined in Section 1. In contrast, the *extended bisection problems* refer to the variants of the problem specified above. We discuss each extended problem separately, but it is straightforward to combine together several extensions (e.g. to allow both edge costs and vertex weights as described above, and require that the total weight of the vertices cut away is a number k that is given in the input).

We consider two approaches for extending our approximation algorithm from the basic bisection problem to an extended problem. One approach is to *reduce* the extended problem to the basic one. Another approach is to *modify the algorithm* that we devised for the basic bisection problem so that it handles also the extended variant. As we discuss below, each approach has its own advantages and so it is valuable to show both approaches for each extended problem. We indeed show that for almost all the extended problems specified above both approaches can be applied, although for a few problems we provide only a modified algorithm.

A major advantage of the reduction approach is that it is self contained and not restricted to the particular algorithm that we devise, so future improvement in the approximation ratio for the basic problem may lead to an immediate improvement also for the extended problem. Most of our reductions transform an approximation ratio $f(n)$ for the basic problem into an approximation ratio $f(n^{O(1)})$ for the extended problem (because they increase the number of vertices n by a polynomial), and so for the current approximation ratio $f(n)$, which is polylogarithmic, these reductions increase the approximation ratio by at most a constant factor. The techniques used in our reductions are similar to those devised in [BJ92, BCLS87] for the (different) purpose of proving NP-hardness results.

The advantages of the algorithm modification approach are that it preserves aspects that are specific to our algorithm, such as an improved $O(\log n)$ approximation ratio for planar graphs, and that it is usually more efficient (and therefore practical) than the reduction approach. A drawback of the algorithm modification approach is that it requires to go again through the algorithm's analysis. In particular, we might be required to verify that the approximate min-ratio cut algorithm (that we use as a black-box) can be extended accordingly. However, the necessary changes in the algorithm and its proof are usually straightforward.

5.1 Edge costs

Suppose that the edges of the input graph G have arbitrary nonnegative costs, and that the cost of a bisection is the total cost (i.e. sum of the costs) of its edges, and we wish to find a bisection of G of (approximately) minimum cost.

Reduction. We reduce the extended problem of bisection with edge costs (described above) to the basic bisection problem, as follows. Given a graph G with edge costs as an input, we first guess the most costly edge in a minimum cost bisection of G, by exhaustively trying all $O(n^2)$ edges in the input graph. By scaling all edge costs, we can assume, without loss of generality, that the cost of the guessed edge is n^2. It follows that the cost b of the optimum bisection is at least n^2 but smaller than n^4. We then round down all edge costs to their closest integer, which can decrease the cost of any bisection by at most $\binom{n}{2} \leq b/2$ and therefore by a factor of at most 2. We next change to n^5 every edge cost that is larger than n^5, which does not affect the cost of nearly optimal bisections (i.e. whose original cost was within ratio of roughly n from the minimum). Finally, we replace each vertex of the graph by a clique of size n^5, and each edge (u, v) of cost t by t unit cost edges placed arbitrarily between the clique of u and the clique of v (since $t < n^{10}$ we can do that with no parallel edges).

The bisection of minimum cost b in G corresponds to a bisection of cost $\Theta(b)$ in the resulting graph. Hence, applying our algorithm for the basic problem on the resulting graph (which has n^6 vertices) yields a bisection whose cost is $O(b(\log n^6)^2) = O(b \log^2 n)$. This bisection cannot split any of the cliques that we created, as otherwise its cost will be at least $n^5 - 1 \gg b \log^2 n$, and it therefore must correspond to a bisection of G, whose cost is roughly the same, namely $O(b \log^2 n)$, as required.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem with edge costs, as follows. Rather than considering the number of edges we always consider their cost, e.g. $e(V_1, V_2)$ denotes the sum of the costs of the edges with one endpoint in V_1 and one endpoint in V_2. The corresponding changes in our algorithm and analysis are straightforward. Note that the amortized cut algorithm (see Fig. 4) requires (in step 1) a subroutine that computes an approximate min-ratio cut with respect to the edge costs, but known algorithms (e.g. due to [LR99]) provide this subroutine. Note also this algorithm's binary search (step 3) takes $O(M \log n)$ iterations, where M is the number of bits used to represent an edge cost, and so the running time is polynomial in the input size. The resulting approximation ratio is the same as for the basic problem, i.e. $O(\log^2 n)$.

5.2 Polynomial vertex weights

Suppose that the vertices of the input graph G have nonnegative integer weights that are bounded by a polynomial n^c (where n is the number of vertices in G), and let a bisection be a cut that separates half of the total weight (i.e. sum of the weights) of the vertices of V. We wish to find a bisection of G of (approximately) minimum cost. Note that if the weights are allowed to be exponential in n, finding any bisection of the graph is equivalent to the partition (or subset-sum) problem, and therefore NP-hard.

Reduction. We reduce the extended problem of bisection with vertex weights (described above) to the basic bisection problem, as follows. Given a graph G with vertex weights as an input, we replace each vertex of cost w in G by a clique of $\max\{1, w \cdot n^3\}$ unit weight vertices, and replace each edge (u, v) in G by one edge placed arbitrarily between the clique of u and the clique of v. In addition, for each vertex of weight 0 in G we place in the graph a new isolated vertex of unit weight.

A bisection of minimum cost b in G corresponds to a bisection of the same cost b in the resulting graph. Hence, applying our algorithm for the basic problem on the resulting graph (which has at most n^{c+4} vertices) yields a bisection whose cost is $O(b(c+4)^2 \log^2 n)$. This bisection cannot split any of the cliques that we created, as otherwise its cost will be at least $n^3 - 1 \gg b \cdot (c+4)^2 \log^2 n$. Furthermore, the vertices of the created cliques of size at least n^3 must be partitioned evenly by this bisection, as otherwise their partition deviates from an even one by at least n^3 (these clique sizes are multiples of n^3) which is much more than the total number of remaining vertices, $2n^2$ (recall that we added isolated vertices for vertices of weight 0 in G). The computed bisection of the resulting graph therefore corresponds to a bisection of G, whose cost is the same, namely $O(b(c+3)^2 \log^2 n)$, as required.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem with vertex weights, as follows. Rather than considering the number of vertices in a part we always consider their total weight, e.g. $r(S)$ denotes the cost of the cut $(S, V \setminus S)$ divided by the minimum between the weight of S and the weight of $V \setminus S$. The corresponding changes in our algorithm and analysis are straightforward. Note that the amortized cut algorithm (see Fig. 4) requires (in step 1) a subroutine that computes an approximate min-ratio cut with respect to the vertex weights, but known algorithms (e.g. due to [LR99]) provide this subroutine. Note also that in this algorithm's related graph G' (step 2) the capacity of an edge between a vertex $v \in V_1$ and the new vertex s is p times the weight of v_1. The resulting approximation ratio is the same as for the basic problem, i.e. $O(\log^2 n)$.

5.3 Separating two vertices from each other ($s - t$ cut)

Suppose that the input graph G contains two special vertices s and t, and we wish to find a bisection that separates s from t and has minimum cost. (Note that the converse restriction, namely that s, t will not be separated, is equivalent to merging them into one vertex of weight 2, and therefore follows from Section 5.2).

Reduction. We reduce the extended problem of a bisection that separates s from t to the extended problem of bisection with vertex weights (described in Section 5.2), as follows. Given an input graph G with special vertices s, t as above, we let the vertices s, t have weights n and let all other vertices of G have weight 1. The total weight of s and t together is $2n$, while the total weight of all other vertices is $n - 2$ (and thus smaller), so every bisection of the resulting graph must separate s from t. It follows that every bisection of the resulting graph corresponds to a bisection of G that separates s from t and has the same cost, and vice versa. We can therefore find a bisection of G that separates s from t and its cost is within $O(\log^2 n)$ from the minimum.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem of a bisection that separates s from t, as follows. We change the dynamic programming table Q of the combining stage, so that every entry $Q(i, k, g)$ contains two solutions (if they exist); one solution with the k chosen vertices containing s but not t, and the other solution with the k chosen vertices not containing any of s and t. Computing the table entries is straightforward, and the output of the algorithm is $\min_g Q(i_{root}, n/2, g)$, where the minimum is taken only over solutions that contain s and not t. The necessary changes in our analysis are straightforward. The resulting approximation ratio is the same as for the basic problem, i.e. $O(\log^2 n)$.

5.4 Cutting an arbitrary given number of vertices

Suppose that the input consists of a graph G and a number k, and we wish to find a minimum cost cut that separates exactly k vertices.

Reduction. We reduce the problem of cutting away a given number k of vertices to the problem of bisection with vertex weights (described in Section 5.2), as follows. Given an input graph G and a number k (assume, without loss of generality, that $k \leq n/2$), we let the vertices of G have weight 1, and add to the graph an isolated vertex of weight $n - 2k$. It is clear that every bisection of the resulting graph corresponds to a cut of G that separates k vertices and has the same cost, and vice versa. We can therefore find a cut of G that separates k vertices and its cost is within $O(\log^2 n)$ from the minimum.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem of cutting a given number of vertices, as follows. The only change in the algorithm is in the combining stage, that now outputs $\min_g Q(i_{root}, k, g)$, where Q is the dynamic programming table (see Section 4.4). The necessary changes in our analysis are straightforward. The resulting approximation ratio is the same as for the basic problem, i.e. $O(\log^2 n)$.

5.5 Cutting into a fixed number of parts

Suppose that we wish to find a cut that separates the input graph G into a fixed number p of parts of equal size.

We do not know of a reduction from this extended problem to the basic bisection problem. A recursive bisection approach has a poor performance in general, although it may be useful in some special cases and if some requirements are relaxed, see [ST97] and the references therein.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the problem of cutting the graph into p parts of equal size, as follows. The cost of a cut that partitions V into p parts $V^1, \ldots, V^p$ is

$$\sum_{j<l} e(V^j, V^l) = \frac{1}{2} \sum_j e(V^j, V \setminus V^j).$$

Therefore, by scaling the value of every possible solution by a factor of 2 (which clearly does not affect any approximation ratio issues), we obtain that the objective function of the extended problem has the convenient form $\sum_j e(V^j, V \setminus V^j)$. Observe that each cut $(V^j, V \setminus V^j)$ corresponds to separating V^j from the other parts, which are grouped into one part $V \setminus V^j$. Thus, each summand $e(V^j, V \setminus V^j)$ in the objective function is similar to the basic bisection problem (with the minor exception that the two sides are not of the equal sizes). Below we describe the modifications to the three stages of the algorithm, which works simultaneously on all p cuts $(V^j, V \setminus V^j)$. Its analysis is based on applying the new accounting method of Section 2 separately to each of these p cuts.

The decomposition stage computes a decomposition tree T exactly as in the algorithm for the basic problem (see Section 4.1). Observe that the amortized cut notion does not depend on the cut that we seek, and so the obtained decomposition (and its tree T) can be used for all cuts $(V^j, V \setminus V^j)$.

We extend the notion of a labeling of the decomposition tree, as follows. An extended labeling of T assigns to every tree node a vector of p "basic" labels, one label for each cut $(V^j, V \setminus V^j)$. An extended labeling corresponds to deciding at each tree node i and for each j, which of V^j and $V \setminus V^j$ is considered charged (and which is considered free) in the part V_i. Note that an extended labeling can be viewed as a vector, whose coordinate j forms a basic labelings for $(V^j, V \setminus V^j)$.

The labeling stage marks some nodes of the tree T exactly as in the algorithm for the basic problem (see Section 4.2). This stage implicitly defines a family $\overline{\mathcal{F}}$ that consists of all extended labelings in which every unmarked node has the same label as its closest marked ancestor (there is no restriction on the labels of the marked nodes). It is straightforward that $\overline{\mathcal{F}}$ contains at least one extended labeling, for which every coordinate j (forms a basic labeling that) is α-consistent with the cut $(V^j, V \setminus V^j)$. We can restrict the number of possible labels at the marked (and hence also unmarked) nodes from 2^p to $p+1$ values, as follows. Similar to the proof of Lemma 8 it is sufficient for our purposes that $\overline{\mathcal{F}}$ contains the labeling where V^j is considered free at a marked node i if more than half the vertices of the part V_i are from V^j. At any part V_i, the latter can happen for at most one value of j, and so it suffices to consider only labelings where at most one V^j is free.

We extend the notion of a charge of a vertex, as follows. The extended charge of a vertex v with respect to an extended labeling is the sum of the basic charges of v with respect to each of the p coordinates of this extended labeling.

The combining stage uses dynamic programming on a table $\overline{Q}$, whose entries are of the form $\overline{Q}(i, \overline{k}, \overline{g})$, as follows. i is a tree node. $\overline{k} = (k_1, \ldots, k_p)$, where k_j is the desired size of the jth part and $\sum_j k_j = |V_i|$. $\overline{g} = (g_1, \ldots, g_p)$ where g_j is a guess list that contains the jth label of every marked ancestor of i. An entry $\overline{Q}(i, \overline{k}, \overline{g})$ contains the optimal solution to the following problem: Choose a partition of V_i into subsets with sizes according to $\overline{k}$, and choose a labeling from $\overline{\mathcal{F}}$ that agrees with g, so that the sum of the extended charges of all the vertices of V_i with respect to the chosen labeling, is minimal over all such choices. Note that this problem requires some correlation between p cuts, and therefore $\overline{Q}(i,\overline{k},\overline{g})$ is generally not equal to $\sum_j Q(i,k_j,g_j)$ (where Q is the basic table).

The rules for computing the entries of the table $\overline{Q}$ are a straightforward extension of those for the table Q (see Section 4.4). The algorithm computes all the table entries and then outputs $\min_{\overline{g}} \overline{Q}(i_{root},\overline{k},\overline{g})$ where $\overline{k}=(n/p,\ldots,n/p)$.

The running time of this modified algorithm is polynomial in n (for fixed p). Indeed, the decomposition stage and the labeling stage are exactly as in the algorithm for the basic bisection problem, so let us consider the dynamic programming table $\overline{Q}$ of the combining stage. The number of tree nodes i is $O(n)$, and the range of $\overline{k}$ contains at most n^p possible values. The vector $\overline{g}$ contains one of $p+1$ possible values for each of the $O(\log n)$ marked ancestors (of the relevant tree node i), so $\overline{g}$ assumes one of $n^{O(\log p)}$ values. It follows that the size of the table $\overline{Q}$ is $n^{p+O(\log p)}$. Each table entry is computed efficiently from previously computed entries, and hence the combining stage takes polynomial time.

To analyze the approximation ratio, let $V^1,\ldots,V^p$ be the optimal partition of the input graph into p parts of equal size. Recall that the extended charge of a vertex is the sum of its basic charges with respect to each cut $(V^j, V\setminus V^j)$, and we can therefore apply the analysis of the basic algorithm for each cut $(V^j, V\setminus V^j)$ separately. It follows that the output value is guaranteed to be at most $O(\log^2 n)\cdot\sum_j e(V^j, V\setminus V^j)$. Furthermore, one can obtain from the table $\overline{Q}$ a cut (into p parts of equal size) whose cost is at most (half) this value, i.e. within a ratio of $O(\log^2 n)$ from the minimum.

5.6 Bicriteria approximation and balanced cuts

Suppose that we wish to find a 2/3-balanced cut (recall that a cut is called β-balanced if it partitions the graph into two parts, each of size at most βn) whose cost is guaranteed to be small relative to the minimum cost b of a bisection (i.e. a 1/2-balanced cut). Here, the minimum bisection problem is relaxed in two respects, as the solution cut is allowed to have cost larger than b and also to deviate from the cardinality constraints (for its two sides). Algorithms for such problems are sometimes referred to as bicriteria approximation and sometimes as pseudo-approximation.

Known bicriteria approximation algorithms find a 2/3-balanced cut of cost $O(b\log n)$. Leighton and Rao [LR88, LR99] show how an algorithm that finds a τ approximate min-ratio cut can be used to find a 2/3-balanced cut of cost $O(b\tau)$; the approximation ratio $\tau=O(\log n)$ that they achieve is the best currently known, see also [Shm97]. Even, Naor, Rao and Schieber [ENRS97] devise a different algorithm that also finds a 2/3-balanced cut of cost $O(b\log n)$.

We show below that amortized cuts can be used to obtain also bicriteria approximation algorithms (in addition to approximation algorithms) for minimum bisection. In fact, our algorithm is similar to the one of [LR88, LR99], except that we use amortized cuts instead of approximate min-ratio cuts.

Lemma 15 *An algorithm that finds a ρ-amortized cut can be used to find a 2/3-balanced cut of cost $b(1+O(\rho))$.*

Proof. Given an input graph $G(V, E)$ on n vertices, use the algorithm that finds a ρ-amortized cut, as follows. Repeatedly find (in the graph) a ρ-amortized cut and remove (from the graph) the smaller of its two sides, until the graph contains no more than $2n/3$ vertices. Denoting by S the set of vertices that remain in the graph after the last iteration, output the cut $(S, V \setminus S)$.

It is straightforward to see that $n/3 < |S| \leq 2n/3$, and hence the output cut $(S, V \setminus S)$ is a 2/3-balanced cut. We prove below that the total cost of all edges cut by the amortized cuts (throughout the iterations) is at most $b(1 + O(\rho))$. It would then follow immediately that $e(S, V \setminus S) \leq b(1 + O(\rho))$, as required.

We now upper bound the total cost of all edges cut in the amortized cuts. Let (W, B) be a fixed optimal bisection of cost b, and call the vertices of W white, and the vertices of B black. The total cost of white-black edges cut is clearly at most b. We show below that the total cost of all white-white edges cut is $O(b\rho)$. By the symmetry between W and B, we will then have a similar upper bound on the total cost of the black-black edges cut, and obtain the desired upper bound of $b(1 + O(\rho))$ on the total cost of all edges cut.

To show that the total cost of white-white edges cut in the amortized cuts is $O(b\rho)$, we consider the white vertices W as charged in all the amortized cuts, and then white-white edges are charged-charged edges. The algorithm applies a ρ-amortized cut in parts of G that contain at least $2n/3$ vertices. At least $n/2 - n/3 = n/6$ of the vertices in such a part are black, while at most $n/2$ of them are white, and hence at most 3/4 of the vertices in this part are considered charged. Taking a constant $\alpha \geq 3/4$ in the definition of an amortized cut, we have that the cost of the charged-charged edges cut can be amortized in one of two amortization methods (see Section 2).

In one amortization method the cost of the charged-charged edges cut is amortized against charged-free edges in the smaller side of the cut, with amortized cost at most ρ. Observe that an edge can be in the smaller side of the amortized cut (the side that is removed) in at most one iteration, so the total cost amortized in this method (in all the iterations) against one charged-free edge is at most ρ. Hence, the total cost amortized in this method (in all the iterations) is at most $b\rho$.

In the other amortization method the cost of the charged-charged edges cut is amortized against charged-free edges in the part being divided, with amortized cost at most $\rho|C_1|/|C|$, where C denotes the charged vertices in the part being divided and C_1 denotes the charged vertices in the smaller side of the cut. The total cost amortized in this method (in all the iterations) against one charged-free edge is then upper bounded by ρ times the sum of $|C_1|/|C|$ over all iterations. Recall that the charged vertices are the white vertices, and so $|C| \geq n/6$ in all amortized cuts (i.e. iterations). Furthermore, each vertex is in the smaller side of the cut (the side that is removed) in at most one iteration, and so the sum of $|C_1|$ over all iterations is at most $n/2$. It follows that the total cost amortized in this method (in all the iterations) against one charged-free edge is at most 3ρ, and hence the total cost amortized in this method is at most $b \cdot 3\rho$.

We conclude that the total cost of all charged-charged (i.e. white-white) edges cut in all the iterations is at most $b \cdot 4\rho$. As described above, this proves that the total cost of all edges cut in all the iterations is at most $b(1 + 8\rho) = b(1 + O(\rho))$, and the lemma follows. $\square$

We remark that a 2/3-balanced cut of cost $b(1 + O(\rho))$ can be found also by modifying the algorithm we devised for the basic bisection problem so that its combining stage outputs $\min_{g,n/3\leq k\leq n/2} Q(i_{root}, k, g)$ (and its corresponding cut). Indeed, the proof of Lemma 15 shows a 2/3-balanced cut whose charge (with respect to a certain labeling in $\mathcal{F}$) is at most $b(1+O(\rho))$. Details omitted.

Concluding remarks. Designing an algorithm that finds a cut of amortized cost better than $O(\log n)$ remains an important open question. An efficient algorithm that accomplishes that will not only improve the approximation ratio for minimum bisection (by Theorem 4), but also the bicriteria approximation ratio for minimum bisection (by Lemma 15), which will lead, in turn, to improved approximation ratios for many other problems, see [LR99, Section 3].

Finding a cut whose amortized cost is better than $O(\log n)$ is, in a sense, no harder (and possibly easier) than approximating min-ratio cuts within a ratio better than $O(\log n)$, as the former problem is reducible (by Theorem 3) to the latter. Furthermore, an $O(1)$-amortized cut always exists (by Corollary 5), and we know of no hardness result for the problem of finding such a cut.

Acknowledgements

We thank Kobbi Nissim for his part in bootstrapping this research, and the anonymous referees for comments that improved the presentation.

References

[AKK99] 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.

[AR98] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, 1998.

[BCLS87] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[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.

[ENRS97] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 639–648. ACM, New York, 1997.

[FKN00] U. Feige, R. Krauthgamer, and K. Nissim. Approximating the minimum bisection size. In *32nd Annual ACM Symposium on 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.

[GSV99] N. Garg, H. Saran, and V. V. Vazirani. Finding separator cuts in planar graphs within twice the optimal. *SIAM J. Comput.*, 29(1):159–179, 1999.

[KPR93] P. Klein, S. A. Plotkin, and S. Rao. Excluded minors, network decomposition, and multicommodity flow. In *25th Annual ACM Symposium on Theory of Computing*, pages 682–690, May 1993.

[LLR95] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[LR88] 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, October 1988.

[LR99] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[Shm97] D.B. Shmoys. Cut problems and their applications to divide-and-conquer. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997.

[ST97] H. D. Simon and S. Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, 1997.

[SV95] H. Saran and V. V. Vazirani. Finding k cuts within twice the optimal. *SIAM J. Comput.*, 24(1):101–108, 1995.