A polylogarithmic approximation of the minimum bisection

(Extended Abstract) *

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

August 31, 2000

Abstract

A bisection of a graph with n vertices is a partition of its vertices into two sets, each of size $n/2$. The bisection cost is the number of edges connecting the two sets. Finding the 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 optimal. 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 a graph with n vertices and m edges, where n is even. A *bisection* of G is a set of vertices $S \subset V$ with cardinality $|S| = n/2$. The edges connecting S to its complement $V \setminus S$ are said to be *cut* by the bisection. The *cost* of a bisection is the number of edges that it cuts. The minimum cost of a bisection of the graph is denoted by b. Computing b is NP-hard, cf. [GJS76]. We address the problem of approximating b.

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 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.

*This is a preprint of a paper that will appear in FOCS 2000 (copyright owned as specified in the conference proceedings)

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[1] $\tilde{O}(\sqrt{m/b})$, cf. [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

[1] We use $\tilde{O}(f)$ to denote $O(f \cdot \text{polylog } n)$.

new cut notion which we call an *amortized cut*, and then combines the parts 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, in graphs excluding any fixed graph as a minor (e.g. bounded-genus graphs) min-ratio cut can be approximated [KPR93] 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.*

Our results extend to finding bisections in graphs with edge weights and polynomially bounded nonnegative integer vertex weights. Details are omitted from this version.

1.3 Conventions and notation

We often denote the two sides of a bisection as *white* W and *black* B. A graph may have several different bisections of minimum cost. In our analysis, we use the term *the fixed optimal bisection* to denote one of them (arbitrarily).

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. $r(S) = \frac{e(S,\overline{S})}{\min\{|S|,|\overline{S}|\}}$, while $r'(S)$ denotes 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*. The root of the tree contains the input graph, the leaves of the tree contain individual vertices, and each internal node of the tree contains a part obtained in a divide step, where the two direct descendents of a node are the two subparts that were obtained in the divide step of this part.

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. In Section 4.1 we describe the decomposition stage in more detail.

Stage 2: Labeling. Consider a *labeling* of the decomposition tree T, which labels each node of the tree as either white or black, and let $1/2 < \alpha < 1$ be a fixed parameter (in this paper we choose $\alpha = 0.9$ arbitrarily). A labeling is called *α-consistent* with respect to a white-black bisection (W, B) of the input graph, if for every set of vertices V_i in a tree node i, a white label at node i implies that $|W \cap V_i| \leq \alpha |V_i|$, and a black label at node i implies that $|B \cap V_i| \leq \alpha |V_i|$.

The desired outcome of the labeling stage is a labeling which is α-consistent with the fixed optimal bisection, called in short a *consistent labeling*. However, an optimal bisection is not known to the algorithm, so instead of finding a consistent labeling, this stage produces a family of labelings, such that at least one member of the family is a consistent labeling. 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 a consistent labeling.

Stage 3: Combining. Assume that we have a decomposition tree T (the output of the decomposition stage) and a consistent labeling of it (the output of the labeling stage). Then in the combining stage, each vertex v of the input graph G is assigned a *white charge* and a *black charge*. The two charges are computed in a simple way, based on the labels along the path from the root of T to the leaf that contains the vertex v.

The *charge* of a cut (W, B) of the input graph G is defined to be 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 cut (W, B), its charge is an upper bound on its cost. Moreover, our notion of amortized cut used in the decomposition stage together with the consistent labeling of the labeling stage ensure that the charge of the fixed optimal bisection is within a polylogarithmic factor of its cost, b. Hence, the input graph G contains a bisection whose charge is within polylogarithmic ratio of b.

Finding a bisection of minimum charge 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. Its cost is thus within a polylogarithmic factor of the cost of the minimum cost 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 the consistent labelling) 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 a single labeling that is consistent, but rather a large family of labelings, such that at least one of them is 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 other labeling-bisection pairs. Details appear in Section 4.3.

The rest of the overview motivates a new notion of an *amortized cut*, which is formally defined at the end of this section. This new notion is of central importance in bounding the ratio between the charge and the cost of the minimum cost bisection.

Previous accounting method

Consider a divide and conquer approach for approximating minimum bisection, as follows. Suppose that we want to cut k vertices from a part $U \subseteq V$ of the input graph. For example, a bisection of the input graph is the case of $U = V$ and $k = n/2$. Let $U = U_1 \cup U_2$ be a partition of the part U, e.g. according to the decomposition stage. Then we clearly need to cut some k_1 vertices from U_1 and some k_2 vertices from U_2, with $k_1 + k_2 = k$. In other words, if we find (e.g. recursively) a cut of U_1 that separates k_1 vertices and a cut of U_2 that separates k_2 vertices, then we can combine the two cuts into a cut of U that separates $k = k_1 + k_2$ vertices.

To analyze the approximation guarantee of such a divide and conquer algorithm, we need to bound the cost of the combined cut. The combined cut consists of edges within each of the parts U_1, U_2 and of edges connecting between the two parts. The edges inside each part U_i can be considered as the contribution of the recursion in that part. We thus need to bound the cost of the edges connecting between the two parts, which we consider to be the contribution of the divide step. For the sake of exposition, we assume that k is given and that k_1, k_2 can be guessed, e.g. exhaustively searched in an efficient way. Note that we deal with arbitrary k (rather than $k = |U|/2$) in order to allow recursive application of the divide and conquer scheme.

Let us be more specific (see also Fig. 1). Suppose that for each $i = 1, 2$ we find in U_i the cut (C_i, F_i) with $|C_i| = k_i$. Combining the two cuts yields a cut of U, namely $(C_1 \cup C_2, F_1 \cup F_2)$, that separates $k = k_1 + k_2$ vertices. Formally, let $Cut(U', k')$ denote the cost of the cut of U' that separates k' vertices and is found by the algorithm. The cost of the combined cut is then

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

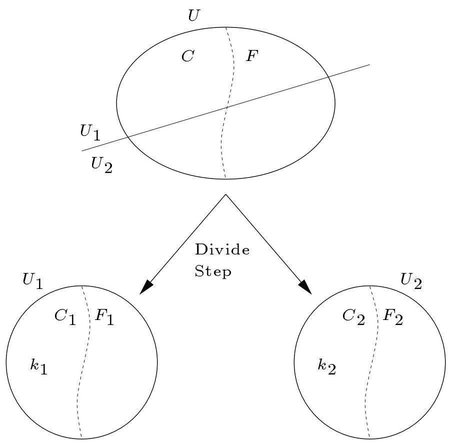

Figure 1: The divide and conquer paradigm

The approach of [FKN00] is based on a straightforward upper bound on the cost of the combined cut. The edges of the combined cut that connect between the two parts U_1, U_2 are clearly no more than all the edges between the two parts. In other words, 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 bound

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

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

The divide steps of [FKN00] partition a part U using an approximate min-ratio cut. This cut is 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. Note that the bound $e(U_1, U_2)$ is tight because the cuts within each U_i are computed independently of each other, and 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

This paper introduces 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

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

The upper bound (3) is more suitable for a divide and conquer approach than the actual cost (1). Let us call $e(C_1, U_2) + e(C_2, U_1)$ the *charge* of the divide step. 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, and it 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. The latter property gives rise to two independent subproblems (as in each part U_i we want to minimize the cost of its cut and the charge to this part), enabling therefore a recursive divide and conquer approach. In contrast, each of the terms $e(C_1, F_2)$ and $e(C_2, F_1)$ that appear in the actual contribution of the divide step, depends on the cuts chosen in both parts, and we cannot break the problem into two independent subproblems.

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

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

The *charge* of a bisection is the upper bound obtained by applying the upper bound (3) recursively, i.e. the sum of all the divide steps charges throughout the decomposition tree. In Section 4.3 we discuss this notion in more detail, and show it is equivalent to the one defined in the algorithm outline.

Desired divide step

Rather than find a bisection of minimum cost, our approximation algorithm looks for a bisection of minimum charge. Therefore, our desired divide step is one that guarantees that for an optimal bisection, charge can be used to approximate cost.

Consider the charge of the optimal fixed bisection. It is the sum of the charges of all the divide steps throughout the tree T. The charge of a single divide step can be written as $e(C_1, U_2) + e(C_2, U_1) = e(C_1, F_2) + e(C_2, F_1) + 2e(C_1, C_2)$, 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. For this bisection, the difference between charge and cost is thus twice the cost of all the charged-charged edges cut in all the divide steps along the tree T. The desired divide step thus cuts relatively few charged-charged edges, where relative here is with respect to the bisection cost b. Since b is the cost of the charged-free edges that are cut in all the divide steps, we would like to amortize the cost of the charged-charged edges cut against the cost of the charged-free edges cut.

Amortized cut

We amortize the cost of the charged-charged edges cut in a divide step in (one of) two methods, as follows. In one method the above cost is amortized against the charged-free edges inside the smaller side of the cut of the divide step. Suppose that the amortized cost for an edge is always at most ρ_e. An edge can be inside the smaller side of the cut in at most $\log n$ divide steps throughout the recursive decomposition, and the amortization is only against charged-free edges, i.e. edges of the optimal fixed bisection, and hence the total cost amortized in this method is at most $O(b \cdot \rho_e \log n)$.

In the second method the cost is *implicitly* amortized against the charged-free edges in U, the part that is being divided. Note that in the worst case, an edge might appear in as many as $\Omega(n)$ (very imbalanced) divide steps throughout the recursive decomposition. Therefore, the amortization mechanism is more complicated.

Formally, in the second method the above cost is amortized against the charged vertices inside the smaller side of the cut, say C_1, and then we factor out $r'(C) = e(C,F)/|C|$ (see Section 1.3 for the difference between $r'(C)$ and $r(C)$). That is, if the cost amortized against a vertex in C_1 is at most $\rho_v \cdot r'(C)$, then the cost implicitly amortized against a charged-free edge in U is $\rho_v \cdot |C_1|/|C|$. In Section 4.3 we bound the total cost amortized in this second method by $O(b \cdot \rho_v \log n)$.

We can now formally define the notion of an amortized cut, based on the two charging methods described above.

Definition (amortized cut). Let (U_1, U_2) be a cut in a graph $G'(U, E')$, and assume, without loss of generality, that $|U_1| \leq |U_2|$. Let $U = C \cup F$ be a partition of the graph vertices 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

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

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 they 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 $0 < |C| \leq \alpha|U|$. We say that the cut (C_1, C_2) is *ρ-amortized* if its amortized cost is at most ρ.

The following proposition characterizes the amortized cost of a cut in terms of $r(C) = \frac{e(C,F)}{\min\{|C|,|F|\}}$.

Proposition 1 *A cut (U_1, U_2) is $O(\rho)$-amortized if and only if for every partition $U = C \cup F$ with $0 < |C| \leq \alpha|U|$ it holds that*

$$e(C_1, C_2) \leq O(\rho) \cdot \max\{e(C_1, F_1), |C_1| r(C)\} \tag{4}$$

Proof. When $|C| \leq \alpha|U|$ for a fixed $\alpha < 1$, 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, i.e. $r'(C) = \Theta(r(C))$. The proof then follows from the amortized cut definition. $\square$

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 to approximate min-ratio cuts.

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

The next lemma gives an upper bound of $O(1)$ on the amortized cost of optimal min-ratio cuts.

Lemma 2 *A min-ratio cut in a graph is $O(1)$-amortized.*

Proof. Let (V_1, V_2) be a min-ratio cut in a graph G, and assume without loss of generality that $|V_1| \leq |V_2|$. Let $V = C \cup F$ be an arbitrary partition of the graph vertices to charged C and free F, with $0 < |C| < |V|$, and denote $C_i = V_i \cap C$ and $F_i = V_i \cap F$ for $i = 1, 2$. We will show that[2] $\min\{\rho_e, \rho_v\} \leq O(1)$.

If the amortized cost for the edges $\rho_e = e(C_1, C_2)/e(C_1, F_1)$ is at most 3 then we're done, so from now on assume $\rho_e > 3$. If $|F_1| \leq |C_1|$ then $\frac{e(C_1,C_2)}{|C_1|} \leq \frac{e(V_1,V_2)}{\frac{1}{2}|V_1|} = 2r(V_1) \leq 2r(C)$ where the last inequality is due to (V_1, V_2) being an optimal min-ratio cut, and similar to Proposition 1, the amortized cost for the vertices is $\rho_v \leq 2r(C)/r'(C) \leq O(1)$, and we're done. So from now on assume that $|F_1| > |C_1|$. We further assume that $|C_1| > 0$ (and thus $|F_1| > 0$), or otherwise we're done because no charged-charged edges are cut, and then $\rho_v = 0$ by definition.

The cut (V_1, V_2) is a min-ratio cut in G, so $r(V_1) \leq r(F_1)$, which can be rewritten as

$$\frac{e(V_1, V_2)}{e(F_1, C_1) + e(F_1, V_2)} \leq \frac{|V_1|}{|F_1|}$$

[2] The choice of constants in the proof is not optimal.

By manipulating each side of the inequality, we get

$$1 + \frac{e(C_1,V_2) - e(F_1,C_1)}{e(F_1,C_1) + e(F_1,V_2)} \le 1 + \frac{|C_1|}{|F_1|}$$

which we rewrite as

$$e(C_1,V_2) - e(F_1,C_1) \le \frac{|C_1|}{|F_1|}\left[e(F_1,C_1) + e(F_1,V_2)\right]$$

By using our assumption that $|F_1| > |C_1|$, we get

$$e(C_1,V_2) - 2e(F_1,C_1) \le \frac{|C_1|}{|F_1|} e(F_1,V_2)$$

We get rid of the term $e(F_1,C_1)$ on the lefthand side with $e(F_1,C_1) = \frac{1}{\rho_e} e(C_1,C_2) \le \frac{1}{3} e(C_1,V_2)$, since $\rho_e > 3$. Rearranging the terms we get

$$\frac{1}{3} \cdot \frac{e(C_1,V_2)}{|C_1|} \le \frac{e(F_1,V_2)}{|F_1|}$$

Clearly, also

$$\frac{1}{3} \cdot \frac{e(C_1,V_2)}{|C_1|} \le \frac{e(C_1,V_2)}{|C_1|}$$

It is not difficult to see that $r(V_1)$ is a convex combination of the righthand sides of the two last inequalities, so we can conclude from these two inequalities that

$$\frac{e(C_1,V_2)}{|C_1|} \le 3r(V_1)$$

Hence

$$\frac{e(C_1,C_2)}{|C_1|} \le \frac{e(C_1,V_2)}{|C_1|} \le 3r(V_1) \le 3r(C)$$

and the amortized cost for the vertices is $\rho_v \le O(1)$. □

3.2 A motivating example

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 $2n + 2n^{1/3}$ vertices (see also Fig. 2). 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 $n^{1/3}$, and each of the four subsets forms a clique. To connect the four pieces, add at random n edges between F_1, F_2, $2n^{1/3}$ edges between F_1, C_1, and $2n^{1/3}$ edges between F_2, C_2. Add also all possible $n^{2/3}$ edges between C_1, C_2, thus making $C_1 \cup C_2$ a clique.

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 $2n^{1/3}$ vertices from the graph G

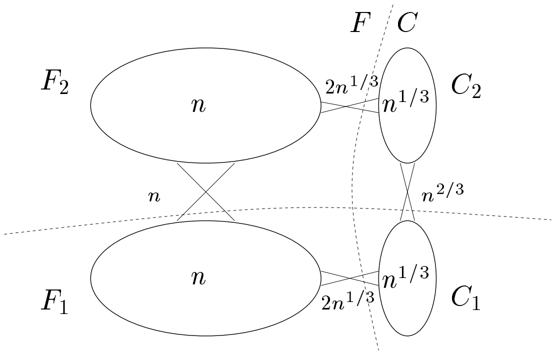

Figure 2: A poor divide step by an approximate min-ratio cut

(if, say, the input graph for bisection consists of this graph G and a clique on $2n - 2n^{1/3}$ 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 $1+\Theta(n^{-2/3})$. The cut $(F_1 \cup C_1, F_2 \cup C_2)$ has a slightly higher ratio of $1 + \Theta(n^{-1/3})$, and so it is a $1 + o(1)$ 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 $n^{2/3}$ charged-charged edges while the total number of charged-free edges in G (and the bisection cost in the input graph) is only $4n^{1/3}$. According to the new accounting method, such a divide step cannot give an approximation ratio better than $\Omega(n^{1/3})$.

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 $\rho_e = n^{2/3}/2n^{1/3} = n^{1/3}/2$. The ratio of the cut (C, F) is $r(C) = r'(C) = 2$, so the amortized cost for the vertices is $\rho_v = n^{2/3}/(n^{1/3}r'(C)) = n^{1/3}/2$. We conclude that a $1 + o(1)$ approximate min-ratio cut might have amortized cost $\rho \geq \min\{\rho_e, \rho_v\} = n^{1/3}/2$.

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 motivation for the algorithm comes from the additional properties that are needed to extend the arguments of Section 3.1 from optimal to approximate min-ratio cuts. The proof of Lemma 2 uses *three times* the fact that (V_1, V_2) is an optimal min-ratio cut. In the first application and in the third one, we conclude that $r(V_1) \leq r(C)$. These applications extend to the case where (V_1, V_2) is an approximate min-ratio cut, with the approximation ratio carried over to the amortized cost of the cut. In the second application we conclude that $r(V_1) \leq r(F_1)$, which does not extend to an approximate min-ratio cut, as demonstrated by the example in Section 3.2. The additional property needed for this application is that the ratio of V_1 is minimal over all its subsets F_1 (in fact, it suffices to consider only subsets F_1 of size at least e.g. $|V_1|/2$). The next lemma thus follows from the same proof as Lemma 2.

Lemma 3 *Let (V_1, V_2) be a τ-approximate min-ratio cut in a graph, with $|V_1| \leq |V_2|$. If for every $F_1 \subset V_1$ it holds that $r(F_1) \geq r(V_1)$, then (V_1, V_2) is an $O(\tau)$-amortized cut.*

Our algorithm, described in Fig. 3, uses a subroutine for computing a τ-approximate min-ratio cut, in order to find an $O(\tau)$-amortized cut. The algorithm 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 3. 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 AMORTIZEDCUT.

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 3: Algorithm for amortized cuts

Lemma 4 *The cut $(S, V \setminus S)$ output by algorithm* AMORTIZEDCUT *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. 4). 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, $e(\cdot,\cdot)$ is the corresponding number of 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 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. □

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

Proof. Lemma 4 guarantees that the cut found by the algorithm satisifes the requirements of Lemma 3, from which it follows that the cut is $O(\tau)$-amortized. □

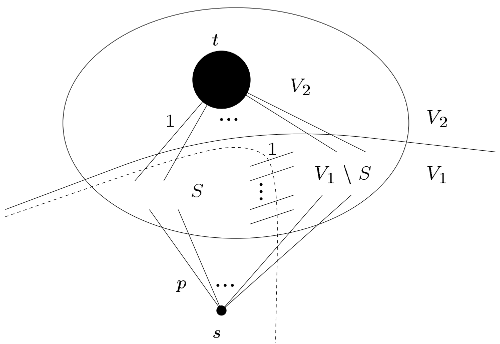

Figure 4: An (s,t)-cut in the related graph G'

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 algorithm for bisection, and prove its apprximation ratio as stated in the next theorem.

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 node i in the tree T contains a part $V_i \subseteq V$ that was found during the recursive decompostion. The root node of the tree T contains V, i.e. the whole input graph G. For a non-leaf node i, we denote its two children nodes by $L(i)$ and $R(i)$, and then node i contains a part V_i which is cut into two smaller parts $V_{L(i)}, V_{R(i)}$. A leaf of the tree T contains a part that consists of a single vertex of G, and therefore T contains exactly n leaves and less than $2n$ nodes.

4.2 Labeling stage

Recall the notions of labelings and α-consistent labelings from Section 2.

The labeling stage *marks* some of the nodes of the decomposition tree T, in a process that goes from the root of the tree 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 $|V_i| \leq \frac{1}{2\alpha}|V_j|$, where V_i and V_j are the parts contained in the nodes i and j, respectively. The constant α satifies $\frac{1}{2} < \frac{1}{2\alpha} < 1$.

A labeling 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 such a labeling is uniquely defined by its labels to the marked nodes.

The family of labelings $\mathcal{F}$ that is produced by the labelings 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 is of 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 5 *The family of labelings $\mathcal{F}$ is guaranteed to contain at least one 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, and the label of each a marked node i is the same as the color of the minority of V_i. We claim that this labeling of $\mathcal{F}$ is consistent. Indeed, 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 it 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 $|W \cap V_i| \leq |W \cap V_j| \leq \frac{1}{2}|V_j| < \alpha|V_i|$. A similar argument is immediate for a marked node, and hence this labeling of $\mathcal{F}$ is consistent. $\square$

4.3 Combining stage

Let us formally define the *charge of a cut* (W, B) of the input graph, given the decomposition tree T and a labeling of it. Consider a node i of the tree T. If it is labeled white then we let $C_i = W \cap V_i$ and $F_i = B \cap V_i$, and if it 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* (here we allow $C_i = \emptyset$, which introduces some trivial cases). Following the new accounting method (3) from Section 2, the *charge of the divide step* of node i is $e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)})$.

Definition (charge). Let (W, B) be a cut of the input graph. Given a labeling of the decomposition tree T, let (C_i, F_i) be the cut of V_i as above. The *charge of the cut* (W, B) is

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

The next two lemmas show that under certain conditions, a bisection charge can approximate its cost. Their proofs are deferred to Section 4.4.

Lemma 6 *The charge of a cut is at least as large as its cost.*

Lemma 7 *The charge of the optimal fixed bisection with respect to a consistent labeling is $b \cdot (1 + O(\rho \log n))$.*

The charge of a cut (W, B) can be distributed to the vertices of the input graph, as follows. For a vertex $v \in V_i$ define the *cross degree* of v at node i, denoted $cross_i(v)$, as the cost of the edges incident to v that are cut in the divide step of i. The *charge of a vertex* $v \in V$ is the sum of the cross degree of v at all nodes i for which v is contained in the charged side, i.e. $\sum_{i: v \in C_i} cross_i(v)$. The next lemma proves that distributing the charge of a cut to the vertices as above is correct.

Lemma 8 *The charge of a cut (W, B) is the sum of the vertex charges of all $v \in V$.*

Proof. The charge of a divide step of node i can be distributed to the vertices of V_i, as follows. Each vertex v in C_i is charged with its cross degree at node i, and each vertex v in F_i is not charged at all. The lemma follows by summing the above over all the nodes i. □

The charge of a vertex can be computed from the side it belongs to in the cut (W, B), the decomposition tree T, and the labeling of T. Note, however, that the charge of a vertex does not depend on the side of the cut (W, B) that other vertices of the graph belong to.

Minimizing charge when a labeling is given

Finding a bisection of minimum charge with respect to a particular labeling given to us is relatively straightforward. By Lemma 8, the charge of a bisection (W, B) is the sum of the vertex charges. The charge of a vertex only depends on its side in the cut (W, B), since the decomposition tree T and the labeling are fixed. We can compute for each vertex v what will be its charge if it will belong to W, called the *white charge*, and what will be its charge if it will belong to B, called the *black charge*. As described in Section 2, 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 cut (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 pick the $n/2$ vertices with smallest net-charge.

Minimizing charge with respect to $\mathcal{F}$

The decomposition 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.

We fill a table Q with 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 non-leaf 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 non-leaf 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 9 *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 the 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 non-leaf 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 10 *The combining stage finds a bisection of the input graph with charge at most $b(1 + O(\rho \log n))$.*

Proof. Follows immediately from Lemmas 9,5 and 7. $\square$

This corollary completes the proof of Theorem 4, since by Lemma 6 the charge of a bisection is an upper bound on its actual cost.

4.4 Charge vs. cost

Proof. (of Lemma 6). As the decomposition stage eventually divides the graph into individual vertices, every edge of the graph is cut by some divide step, and this includes in particular all edges of the cut (W, B). We have seen in section 2 that the charge for a divide step is always at least as large as the true cost of the (W, B) edges cut in this step. Summing up over all divide steps, the proof follows. $\square$

Proof. (of Lemma 7). Let (W, B) be the fixed optimal bisection, and consider its charge with respect to a consistent labeling. As described in Section 2, the difference between the charge of a divide step and the true cost of the (W, B) edges cut in this step is the cost of the charged-charged edges cut in this divide step. Summing up over all the divide steps we get that the difference between the charge of the fixed optimal bisection and its cost is $\sum_i e(C_i \cap V_{L(i)}, C_i \cap V_{R(i)})$ where i ranges over all (non-leaf) 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 bound the sum of $e(C_L, C_R)$ over all nodes i in T, observe that each part V_i is divides using a ρ-amortized cut, and that the consistent labeling guarantees that $|C_i| \leq \alpha|V_i|$ for all nodes i. Hence, each term $e(C_L, C_R)$ can be amortized against either (i) the charged-free edges in the smaller child part, or (ii) the charged vertices in the smaller child part. It therefore suffices to bound the total cost that is amortized in both methods (i) and (ii).

Method (i) amortizes cost against charged-free edges which are always edges of the fixed optimal bisection, and there are only b such edges in the input graph. Each of these edges belongs to the smaller side of the cut in at most $\log n$ divide steps throughout the tree T, and each time the amortized cost is no larger than ρ. Hence, the total charge in method (i) is at most $b \cdot \rho \log n$.

In method (ii) the amortized cost for a charged vertex of the smaller child part is at most $\rho \cdot r'(C_i)$, where $r'(C_i) = \frac{e(C_i, F_i)}{|C_i|}$ is the ratio of the cut (C_i, F_i) towards C_i. Note that if $|C_i| = 0$ then no charged-charged edges are cut, and there is no cost to amortize, so for simplicity, terms whose denominator is 0 are defined to be equal to 0 in this proof.

Without loss of generality let the decomposition stage place the smaller of the two subparts in the left child, i.e. $|V_{L(i)}| \leq |V_{R(i)}|$ for every non-leaf node i. Suppose that instead of amortizing the cost against the vertices of C_L, we amortize it against the $e(C_i, F_i)$. The amortized cost for an edge would then be at most $\rho \cdot \frac{|C_L|}{|C_i|}$. The cost is only amortized against charged-free edges, which are always edges of the fixed optimal bisection, and there are only b such edges in the input graph. A cost is amortized against such an edge only in parts V_i which contain that edge, and all the corresponding nodes i lie on a path from the root to a leaf of the decomposition tree. We claim that the sum of $\frac{|C_L|}{|C_i|}$ over any path from the root to a leaf, is bounded by $O(\log n)$.

Let us first finish the proof of the lemma conditioned on the claim. By the claim, the total cost amortized against an edge of the fixed optimal bisection is at most $O(\rho \log n)$, and hence the total cost amortized in method (ii) is $O(b \cdot \rho \log n)$. The total cost amortized in the method (i) is $b \cdot \rho \log n$, and thus the sum of $e(C_L, C_R)$ over the tree T is $O(b \cdot \rho \log n)$. Since the cost of the fixed optimal bisection is b, we conclude that its charge is at most $b \cdot (1 + O(\rho \log n))$.

To prove the claim, consider an arbitrary path from the root to a leaf. Without loss of generality, the path nodes are $1, 2, \ldots, p+1$. 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 $\frac{|C_L|}{|C_i|}$ is either $\frac{w_{L(i)}}{w_i}$ or $\frac{b_{L(i)}}{b_i}$, where $w_j = |W \cap V_j|$ and $b_j = |B \cap V_j|$. Clearly, we can bound $\frac{|C_L|}{|C_i|}$ by their sum $\frac{w_{L(i)}}{w_i} + \frac{b_{L(i)}}{b_i}$. (Recall that terms with denominator 0 are defined to be equal to 0 and can be ignored). Hence,

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

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 $\frac{w_{L(i)}}{w_i} = \frac{w_{i+1}}{w_i} \leq 1$. The number of such nodes i is at most $\log n$, since a path from the root to a leaf contains at most $\log n$ left childs. The contribution of all such nodes 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 $\sum_1^p \frac{w_i - w_{i+1}}{w_i} \leq \frac{1}{w_1} + \ldots + \frac{1}{2} + 1 = H(w_1) \leq H(n)$ 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 $\sum_{path} \frac{|C_L|}{|C_i|} \leq O(\log n)$, proving the claim and the lemma. □

Acknowledgements

The first author is the Incumbent of the Joseph and Celia Reskin Career Development Chair. Work of the second author was supported in part by the Open University of Israel. This research was supported in part by a Minerva grant.

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.

[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.

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

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

[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] Philip Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th Annual ACM Symposium on the 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. IEEE, 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 *Approximation Algorithms for NP-Hard Problems.* (Editor: D. Hochbaum) PWS Publishing Company, 1997.

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