Constant Factor Approximation of Vertex-Cuts in Planar Graphs

[Extended Abstract]

Eyal Amir
Computer Science Division
University of California
Berkeley, CA 94720.
eyal@cs.berkeley.edu

Robert Krauthgamer*
Computer Science Division
University of California
Berkeley, CA 94720.
robi@cs.berkeley.edu

Satish Rao
Computer Science Division
University of California
Berkeley, CA 94720.
satishr@cs.berkeley.edu

ABSTRACT

We devise the first constant factor approximation algorithm for minimum quotient vertex-cuts in planar graphs. Our algorithm achieves approximation ratio $1+\frac{4}{3}(1+\epsilon)$ with running time $O(W \cdot n^{3+2/\epsilon})$, where W is the total weight of the vertices. The approximation ratio improves to $\frac{4}{3}(1+\epsilon+o(1))$ if there is an optimal quotient vertex-cut (A^*, B^*, C^*) where the weight of C^* is of low order compared to those of A^* and B^*; this holds, for example, when the input graph has uniform weights and costs. The ratio further improves to $1+\epsilon+o(1)$ if, in addition, $\min\{w(A^*), w(B^*)\} \leq \frac{1}{3}W$.

We use our algorithm for quotient vertex-cuts to achieve the first constant-factor pseudo-approximation for vertex separators in planar graphs.

Our technical contribution is two-fold. First, we prove a structural theorem for planar graphs, showing the existence of a near-optimal quotient vertex-cut whose high-level structure is that of a bounded-depth tree. Second, we develop an algorithm that optimizes over such complex structures in running time that depends (exponentially) not on the size of the structure, but rather only on its depth. These techniques may be applicable in other problems.

Categories and Subject Descriptors

F.2.0 [**Theory of Computation**]: Analysis of algorithms and problem complexity—*General*

General Terms

Algorithms, Theory

Keywords

Approximation algorithms, balanced cut, planar graph, planar separator, quotient cut, vertex separator

*Supported in part by NSF grants CCR-9820951 and CCR-0121555 and DARPA cooperative agreement F30602-00-2-0601.

STOC'03, June 9–11, 2003, San Diego, California, USA.

1. INTRODUCTION

Graph partitioning is extensively used in many areas, including scientific computing, VLSI design, task scheduling, machine vision, and automated reasoning, see e.g. [21, 12, 23, 4, 1]. One important graph partitioning problem that emerges naturally in a variety of applications is the vertex-separator problem defined as follows. Let $G(V, E)$ be an n-vertex graph with vertex costs $c : V \to \mathbb{N}$ and vertex weights $w : V \to \mathbb{N}$. A *vertex-cut* of G is a partition of V into three disjoint sets A, B, C such that no edge in E has one endpoint in A and one in B. The cost of the vertex-cut is $c(C)$ where, throughout, $c(S) := \sum_{v \in S} c(v)$ for $S \subseteq V$.

A vertex-cut is called a *vertex-separator* (aka $2/3$-*balanced*) if $\max\{w(A), w(B)\} \leq \frac{2}{3}w(V)$ where, throughout, for $S \subseteq V$ we write $w(S) := \sum_{v \in S} w(v)$, and $W := w(V)$ denotes the total weight of the vertices. The *vertex-separator problem* is to find a minimum-cost vertex separator in an input graph G. This problem was shown to be NP-hard even in graphs of degree 3 by Bui and Jones [5]. Leighton and Rao [13, 14] give an $O(\log n)$ pseudo-approximation algorithm for this problem in general graphs.[1] A closely related problem is the quotient vertex-cut problem. The *quotient cast* of the vertex-cut is defined as[2]

$$q(A, B, C) := \frac{c(C)}{\min\{w(A), w(B)\} + w(C)}.$$

The *minimum quotient vertex-cut problem* is to find a vertex-cut with minimum quotient in an input graph. Also this problem is NP-hard (see below), and the best approximation ratio known for this problem is $O(\log n)$, due to [14].

For planar graphs, the celebrated Planar Separator Theorem, due to Lipton and Tarjan [15] (see also [2, 16, 25]), shows that every planar graph has a vertex-separator (A, B, C) with $|C| = O(\sqrt{n})$. (This corresponds to the uniform cost case $c(v) = 1$ for all $v \in V$). Furthermore, they give efficient algorithms for finding such a vertex-separator. However, these methods are only guaranteed to find separators of size $O(\sqrt{n})$. Many planar graphs have much

[1]A *ρ-approximation algorithm* for a minimization problem is a polynomial-time algorithm that outputs, for a problem instance, a solution whose cost is at most ρ times that of a minimum-cost solution for the instance. A *pseudo-approximation* algorithm for $2/3$-balanced cuts is allowed to output, say, a $3/4$-balanced cut (its cost is still compared to that of a $2/3$-balanced cut).

[2]Notice the term $w(C)$ in the denominator of the quotient. In general, we can expect the $w(C)$ to be small compared to $w(A), w(B)$ in cuts of small quotient; this natural assumption is provably true for planar graphs with uniform weights and uniform costs. We discuss this issue in more detail in Section 2.2.

smaller separators, so for any particular graph these algorithms may find separators that are far from optimal. We address this issue by devising a constant factor pseudo-approximation for the vertex-separator problem in planar graphs. No approximation better than that for general graphs was previously known for the problem of vertex-separators in planar graphs, even in the uniform weights and costs case; Interestingly, this problem is not known to be NP-hard.

1.1 Related work

Vertex-cut variants of graph partitioning problems are usually closely related to the edge-cut variants of the same problems. For instance, an edge-cut variant (either directed or undirected) of a problem frequently reduces to a vertex-cut variant of the same problem by replacing every edge by a path of length two, letting the new vertex have zero weight and the same cost as the edge it replaces. Notice that this reduction preserves planarity of the input graph. There is also a well-known reduction from the (say undirected) vertex-variant to the directed edge-cut variant, see [14]. However, this reduction does not preserve planarity.

There is a significant amount of research on approximating various graph partitioning problems, and many of these results extend from one variant of the problem to another by using the aforementioned general reductions. An $O(\log n)$–approximation for the minimum quotient edge-cut problem (aka sparse cut) in undirected graphs was first devised in [13]. Their results and techniques were expanded and extended to other graph partitioning problems, see e.g. [24, 14] and the references therein.

For planar graphs, significantly better approximation ratios are usually known. Park and Phillips [18] devise a pseudo-polynomial time algorithm for solving (exactly) the minimum quotient edge-cut problem (improving over a constant factor approximation in [19, 20]). Klein, Plotkin and Rao [11] give a constant factor approximation for the directed version of this problem. These results immediately imply a constant factor pseudo-approximation for edge-separators, see e.g. [19, 24, 14]. Garg, Saran and Vazirani [9] give factor 2 (true) approximation for edge-separators (i.e., $2/3$-balanced cuts). Feige and Krauthgamer [8] give an $O(\log n)$ (true) approximation for minimum-bisection.

As mentioned before, these results do not translate to similar approximation ratios for quotient vertex-cuts, since the well-known reduction of vertex-cut problems to edge-cut problems in directed graphs might create nonplanar graphs even if the original graph is planar. It was erroneously claimed in [20] that the techniques that were devised there for approximating quotient edge-cuts extend to a $3/2$–approximation for quotient vertex-cuts, but these techniques do not tackle the most basic problem, as is discussed in Section 2.3.

1.2 Our results

We devise the first constant factor approximation algorithm for minimum quotient vertex-cuts in planar graphs. Our algorithm achieves approximation ratio $1+\frac{4}{3}(1+\epsilon)$ with running time $O(W \cdot n^{3+2/\epsilon})$. The approximation ratio improves to $4/3\cdot(1+\epsilon+o(1))$ if in a vertex-cut (A^*, B^*, C^*) of optimal quotient cost, C^* has relatively small weight compared to A^* and B^* (namely, $w(C^*) \leq \epsilon \min\{w(A^*), w(B^*)\}$); this holds, for example, when the input graph has uniform (or close to uniform) weights and costs. If, in addition, $\min\{w(A^*), w(B^*)\} \leq \frac{1}{3}W$ then the approximation ratio further improves to $1+\epsilon+o(1)$.

We also achieve the first constant factor pseudo-approximation for vertex-separators in planar graphs. This follows from the approximation for quotient vertex-cuts by extending a well-known technique (see [19, 24, 14]) from edge-cuts to vertex-cuts.

1.3 Techniques

No constant-factor approximation was known previously for the problem of minimum-quotient vertex-cuts in planar graphs even in the uniform weight uniform cost weight case. This is, perhaps, due to subtle but fundamental differences between minimum quotient edge-cuts and vertex-cuts. The results for finding optimal quotient edge-cuts rely on a well-known lemma [17, 20] that there is an optimal quotient edge-cut which divides a graph into two connected graphs. For planar graphs, this implies that there is a simple Jordan curve in the plane that corresponds to an optimal quotient edge-cut. The algorithms proceed by searching for simple cycles in the dual of the planar graph. In particular, Park and Phillips [18] define a cost and weight for each edge in the dual graph such that the cost of any simple cycle in the dual graph is equal to the cost of the corresponding cut in the input graph, and the weight of the cycle corresponds to the weight that the cut separates. Then, they can modify methods for finding minimum mean cycle, due to [10], to find the best such cycle.

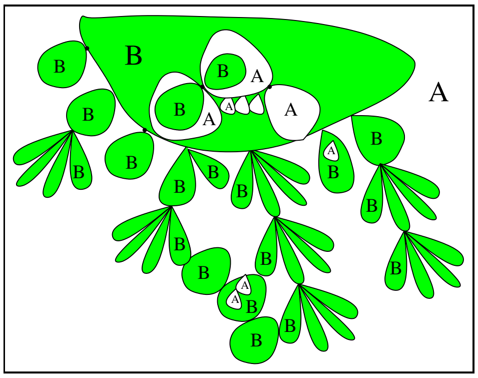

Figure 1: Jordan curve in the plane corresponding to an optimal quotient vertex-cut (A, B, C).

In contrast, an optimal quotient vertex-cut in a graph might divide the graph into an arbitrary number of connected components. In fact, both sides of the cut can be an arbitrary set of components (consider, e.g., a star). For planar graphs, the "Jordan" curve in the plane corresponding to an optimal (or near-optimal) quotient vertex-cut might be quite complex. See e.g. Figure 1 and Section 2.3.

We partially address this difference by showing that in any plane graph there exists a vertex-cut whose quotient cost is within $4/3$ of optimal, and in which *one* side is connected in the plane. We then show that such a separator must correspond to a Jordan-type curve that is essentially a tree of cycles, see e.g. Figure 2.

Still, this structure is more complex than a simple cycle; any walk along the structure visits many vertices numerous times where any revisits are *free* in terms of the cost of the walk. Thus, standard reductions to the minimum mean cycle problem do not work. To address this, we show that there is a near-optimal quotient vertex-cut whose associated tree of cycles has constant depth.

Of course, one can easily extend the mean cycle methods to "guess" a constant number of vertices that are allowed to be revisited. Thus, if a near-optimal walk has a simple structure and only revisits a constant number of vertices, we could find it. Unfortunately, this is not true even for a bounded-depth tree of cycles. For example, the walk separating the tree of cycles in Figure 2 revisits $\Omega(n)$ vertices.

Hence, we need to work a bit more. We consider a walk "around"

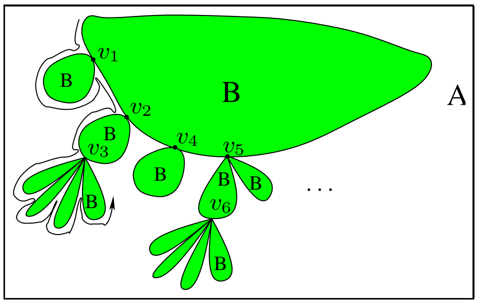

Figure 2: A tree of cycles corresponding to a near-optimal quotient vertex-cut. Midway through a walk around it, v_1 is completely visited, v_2, v_3 are partially visited, and v_5, v_6 have not been visited yet.

a tree of cycles, and notice that at any point in the walk each vertex is either completely visited, has yet to be visited at all, or has been visited and will be visited again. However, the number of vertices in this partially visited state is bounded by the depth of the tree of cycles (see Figure 2). Thus, we devise an algorithm that optimizes over walks of this form. The algorithm needs only "remember" a constant number of vertices at a time (the partially visited ones). Hence, it has a polynomial running time.

This outline is a simplification of our methods. We need to also enforce some topological conditions due to the relationship between the edge weight of our walk and the vertex weight separated. In the technical sections, we also discuss the case where the separator vertices themselves have a significant amount of weight, as well as some generalizations of our structure theorems that we believe will lead to the removal of the $4/3$ factor in our approximation.

Organization. Section 2 presents background notions and concepts; Section 3 shows that restricting the search for quotient vertex-cuts to trees of cycles in an appropriately defined triangulation of the input plane graph guarantees constant factor approximation; Section 4 presents an algorithm based on dynamic programming that searches for quotient vertex-cuts that form a tree of cycles; finally, Section 5 combines these results to present a constant factor approximation for quotient vertex-cuts and, in turn, to a pseudo-approximation for the vertex separator problem.

2. PRELIMINARIES

Throughout this paper we denote by G the input planar graph, by G_0 its plane graph and by G_T its face-vertex graph (see Section 2.1 for definitions). We assume without loss of generality that G (and thus G_0) have no parallel edges or self-loops. Notice, however, that G_T may have parallel edges (e.g., if G is a tree).

A (directed) *cycle* in a graph G is a sequence of edges, denoted $(u_0, v_0), \ldots, (u_{k-1}, v_{k-1})$, such that $v_i = u_{(i+1) \bmod k}$ for all $i < k$. A cycle is *simple* if it contains k distinct vertices and k distinct edges. Note that two parallel edges define a simple cycle of length 2.

2.1 Planar and plane graphs

A graph G is *planar* if it can be drawn in the plane $\mathbb{R}^2$ such that no two edges intersect, except possibly, at their endpoints. A *plane graph* is a planar graph together with a particular drawing of it. A plane graph G partitions $\mathbb{R}^2 \setminus G$ into *faces*; a face is a region that can be linked by an arc in $\mathbb{R}^2 \setminus G$. The *boundary* of a face is the subgraph whose point set in the drawing is the frontier of the face. The boundary of a face is not necessarily a simple cycle. For example, if G is a tree then the boundary of the outer face (the only face of G) is not a simple cycle (but rather traverses every edge of G once in each direction).

A plane graph is called *triangular* if the boundary of every face in it consists of three distinct edges.

Adjacency and connectivity of faces. Two faces of a plane graph G are *adjacent* if their boundaries have a common edge. Note that non-adjacent faces may contain a common vertex in their boundaries.

DEFINITION 2.1. *Let R be a set of faces of a plane graph G. A* faces-path *in R between two faces $f, f' \in R$ is a sequence of faces $f = f_1, \ldots, f_k = f'$ all from R such that for all $i < k$ the faces f_i and f_{i+1} are adjacent. We say that the set of faces R is* connected *if there is a faces-path in R between every two faces $f, f' \in R$. A connected set of faces is also called a* connected region.

Notice that every two faces in a connected region R are linked by a path in the plane $\mathbb{R}^2$ that goes only through faces of R and edges of G (that are on the boundary of two distinct faces of R), without going through any vertex of G. A maximal connected subset of a set of faces F is called a *connected component* of F. Clearly, any set of faces F can be partitioned (uniquely) into its connected components $F_1, \ldots, F_t$; we let $\mathrm{CC}(F) := \{F_1, \ldots, F_t\}$ be the set of connected components of F.

Face-Vertex Graphs. We create a triangular graph (every face has exactly three distinct edges in its boundary) that extends G_0, and call this graph the *face-vertex graph* of G_0 (in fact, every triangular extension of G_0 will do; in particular, if G_0 is triangular, we can assume the face-vertex graph is G_0 itself).

Let $G_0(V_0, E_0)$ be a plane graph. A *face-vertex graph* of G_0 is a plane graph $G_T(V_T, E_T)$ that is obtained from G_0 by adding for each face f of G_0 a new vertex (placed inside f) and connecting (in the plane) this new vertex to every vertex in the boundary of f (by order of their appearance on the boundary). Notice that $V_T \supseteq V_0$ and $E_T \supseteq V_0$, and that G_T may contain parallel edges (e.g., if G_0 is a tree). An example of a plane graph and its face-vertex graph is shown in Figure 2.1(A),(C).

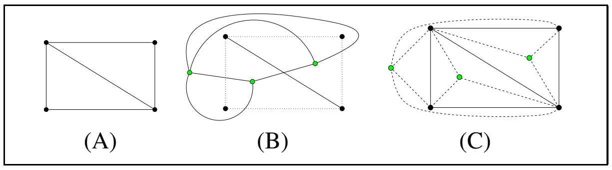

Figure 3: (A) A plane graph, (B) its dual, and (C) its face-vertex graph.

We extend any vertex weights and vertex costs of G_0 to G_T by letting the weight and cost of every vertex of $V_T \setminus V_0$ be 0. (Vertices of V_0 have the same weight and cost in G_T as in G_0.) It is easy to see that any vertex-cut of G_0 corresponds to a vertex-cut in G_T with the same weights and costs, and vice versa. However, G_T has some structural properties (such as being triangular), as stated in the easy to verify proposition below. Our results in Section 3 use these properties to show that for the purpose of finding near-optimal quotient vertex-cuts, one may focus on vertex-cuts induced by certain (not necessarily simple) cycles. It is straightforward to see that if G_0 is already triangular, then every vertex-cut (of G_0)

that is induced by a cycle in G_T there is a corresponding cycle in G_0 that induces the same cut (of G_0), and thus one may apply the algorithm of Section 4 directly on G_0 (instead of G_T).

By the definition of the face-vertex graph, we have the following properties.

PROPOSITION 2.2. *Let G_T be a face-vertex graph of a plane graph G_0. If G_0 has no parallel edges or self-loops then:*
(1) The boundary of every face of G_T is a cycle of length 3 containing one edge from E_0 and two edges from $E_T \setminus E_0$.
(2) Every edge in $E_T \setminus E_0$ has exactly one of its endpoints in V_0. (In particular, this edge is not a self-loop.)

REMARK. The face-vertex graph of a plane graph is not its *dual* (in the classical sense, see e.g. [7, 6]). It is well-known that minimal edge-cuts in a plane graph correspond to simple cycles in the dual (plane) graph; this correspondence was exploited in [19, 20, 18] to devise algorithms that find approximate (or even optimal) quotient edge-cuts and edge-separators in planar graphs. In contrast, there is no exact correspondence between simple cycles in G_T and vertex-cuts in G_0 (see e.g. Section 2.3). The goal of Section 3 is to provide a partial analogue to the well-known correspondence in edge-cuts.

2.2 Balanced and quotient vertex-cuts

In the rest of this paper we allow a vertex-cut to be *trivial*, i.e., to have either A or B empty. This is convenient for our purposes, and does not affect applications, as will be seen later in the paper.

Balanced vertex-cuts. It will be convenient to define the balance of a vertex-cut with respect to two parameters, $0 \leq \alpha \leq 1$ and $0 \leq b \leq 1$. We say that a vertex-cut (A, B, C) is (α, b)*-balanced* if

$$\min\{w(A), w(B)\} + \alpha w(C) \geq bW.$$

where $W := w(V)$ is the total weight of the vertices. For example, the case $\alpha = 1$ and $b = 1/3$ is equivalent to $\max\{w(A), w(B)\} \leq \frac{2}{3} w(V)$, the classical balance condition mentioned in Section 1.

Quotient vertex-cuts. We define the quotient cost of a vertex-cut with respect to some parameter $\alpha \geq 0$. The α*-quotient cost* of a vertex-cut (A, B, C) is

$$q^\alpha(A, B, C) := \frac{c(C)}{\min\{w(A), w(B)\} + \alpha \cdot w(C)}.$$

The *b-limited α-quotient cost* of a vertex-cut (A, B, C) is

$$q_b^\alpha(A, B, C) := \frac{c(C)}{\min\{w(A) + \alpha w(C), w(B) + \alpha w(C), bW\}}.$$

Notice that $q_1^\alpha(A, B, C) = q^\alpha(A, B, C)$ for all α. Throughout, the quotient is defined to be ∞ whenever the denominator is 0 (even if the numerator is also 0). We will use the following relationships between different quotient costs later in the paper. In particular, they allow us to translate results achieved for one quotient into another.

LEMMA 2.3. *Let $0 \leq \alpha < \alpha'$ and $b \leq 1$. Suppose that (A^*, B^*, C^*) is a vertex-cut of G with the minimum b-limited α'-quotient cost such that $w(A^*), w(B^*) \geq \lambda \cdot w(C^*)$ for some $\lambda > 0$. If the b-limited α-quotient cost of some vertex-cut (A, B, C) is at most ρ times the minimum (for such cuts) in G, then the b-limited α'-quotient cost of (A, B, C) is at most $\rho \cdot (1 + \frac{\alpha' - \alpha}{\lambda + \alpha})$ times the minimum (for such cuts) in G.*

Our typical application of this lemma argues that a vertex-cut (A, B, C) having near-optimal $\alpha = 0$ quotient cost must also have near-optimal $\alpha' = 1$ quotient cost. One interesting special case is that of unit costs and weights in planar graphs, where the Lipton-Tarjan theorem guarantees the existence of a vertex-cut with 1-quotient cost $O(1/\sqrt{n})$, and hence any optimal 1-quotient cost vertex-cut (A^*, B^*, C^*) must satisfy $\frac{\min\{|A^*|, |B^*|\} + |C^*|}{|C^*|} \geq \Omega(\sqrt{n})$. The next lemma shows that in this case, there is negligible difference between these two quotients.

LEMMA 2.4. *Let $\alpha' > \alpha \geq 0$. Given a vertex-cut (A, B, C) whose b-limited α-quotient cost is within factor ρ of the minimum, one can find a vertex-cut (A', B', C') whose b-limited α'-quotient cost is within factor $\rho + 1$ of the minimum. (In fact, (A', B', C') is either the given cut (A, B, C) or some trivial cut with $|C'| = 1$.)*

2.3 Where is the problem?

The algorithms of [19, 20, 18] for finding (or approximating) an edge-cut of minimum quotient all rely on the correspondence between minimum quotient edge cuts and simple cycles in the dual. In the vertex-cut problem such a correspondence does not hold. Specifically, we now exemplify that limiting ourselves to vertex-cuts that correspond to simple cycles leads to vertex-cuts whose quotient is arbitrarily far from the optimal. Consider the graph presented in Figure 4 and divide the vertices in this graph into four types, t_1, t_2, t_3, t_4. If our graph has d cycles touching the center vertex, then we set $w(t_1) = \frac{bW}{d}$, $w(t_2) = 0$, $w(t_3) = (1 - b)W$, $w(t_4) = 0$, $c(t_1) = \infty$, $c(t_2) = 10$, $c(t_3) = \infty$, and $c(t_4) = 0$. This graph is triangular, so taking the face-vertex graph of this graph does not add any new cycle-induced vertex-cuts. Clearly, the optimal weight is given when vertices of type t_1 are in one side of the cut and that of type t_3 is in the other side. Vertices of types t_1, t_3 cannot be in the cut because of their large cost. The optimal quotient is achieved by selecting the vertices on the the cycles touching the center vertex. Any simple cycle in this graph separates at most one of the vertices of type t_1 from the rest, thus separating a factor d less weight at the same cost as the optimal quotient vertex-cut.

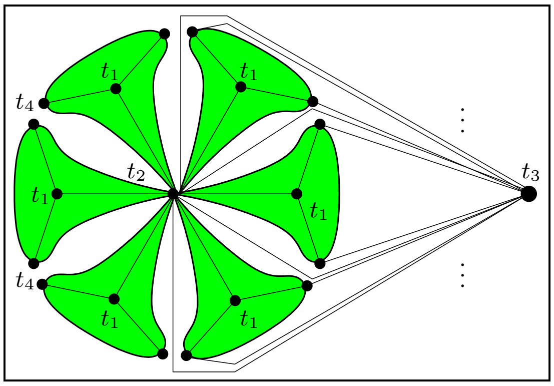

Figure 4: An example where any vertex-cut defined by a simple cycle has quotient cost that is arbitrarily far from the optimum.

3. STRUCTURAL THEOREM

The main result of this section is that in every plane graph G_0 there is a vertex-cut that is nearly optimal in terms of quotient cost and also has a certain "nice" structure. Informally, this structure means that the vertex-cut corresponds to a collection of directed cycles in G_T that are arranged in a tree-like structure of constant depth; in short, we shall call it CAST (Cycles Arranged in a Shallow Tree). The intuition behind this structure is given in Section 2.3.

The precise statement of our basic structural theorem and the relevant definitions (including CAST) are given in Section 3.1. Its proof is then given in Section 3.2. Finally, we devise various extensions and examine special cases in Section 3.3.

3.1 Definitions and basic structural theorem

A face f (vertex v) of G_T is said to be *enclosed* by a directed simple cycle D (in G_T) if f (respectively, v) is contained in the region of $\mathbb{R}^2 \setminus D$ that is to the left of one (and thus all) of the edges in D. A set $\mathcal{D}$ of directed simple cycles (in G_T) is *proper* if every face of G_T is enclosed by at most one cycle in $\mathcal{D}$. Such a proper set $\mathcal{D} = \{D_1, \dots, D_k\}$ defines a vertex-cut (A, B, C) of G_0, as follows. Let C_i be the vertices of V_0 that appear in the cycle D_i, and let B_i be the vertices of V_0 that are enclosed by D_i. Then, let $C := \cup_{i=1}^k C_i$, $B := (\cup_{i=1}^k B_i) \setminus C$ and $A := V_0 \setminus (C \cup B)$. To see that (A, B, C) is a vertex-cut, observe that an edge in G_0 between a vertex of A and a vertex of B must have its two endpoints in different regions of $\mathbb{R}^2 \setminus D_i$ for some i. Also, $B \cap C = \emptyset$ because $\mathcal{D}$ is proper.

The *auxiliary graph* of a proper set $\mathcal{D}$ (of directed simple cycles) is the bipartite graph $\tilde{G}(\tilde{V}_1, \tilde{V}_2, \tilde{E})$ where $\tilde{V}_1 := \mathcal{D}$, $\tilde{V}_2 := V_T$, and $\tilde{E} := \{(d, v) : v \in V_T \text{ belongs to the cycle } d \in S\}$. Such a proper set $\mathcal{D}$ is called a *d-CAST* if its auxiliary graph $\tilde{G}$ is a forest (i.e. contains no cycles) and each of its connected components is a tree that can be rooted at some vertex $v \in \tilde{V}_2 = V_T$ so that its depth is at most $2d$. For example, the cycles drawn in Figure 4 form a 1-CAST. An $O(1)$-CAST is called in short a CAST.

The following theorem is proved in Section 3.2. This result easily extends to α-quotient cost for arbitrary α by applying Lemma 2.4, and then the factor becomes $\frac{4}{3}\frac{d+1}{d} + 1$.

THEOREM 3.1 (BASIC STRUCTURAL THEOREM). *In every plane graph G_0 there is a vertex-cut that is defined by a d-CAST in G_T and its 0-quotient cost (i.e. $\alpha = 0$) is within a factor of $\frac{4}{3}\frac{d+1}{d}$ from the minimum (over all vertex-cuts) in G_0.*

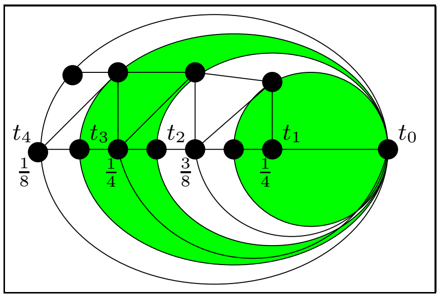

Figure 5: $4/3$ is tight

We note that the constant $4/3$ is tight, as shown by Figure 5 (t_1, t_2, t_3, t_4 have cost ∞ and the noted weights, t_0 has cost 1 and weight 0, and all other vertices have cost 0 and weight 0). However, we can improve over this constant by allowing a more complicated structure than a CAST. We can also show that the constant loss in the α-quotient cost for $\alpha > 0$ can be improved by relying on certain assumptions that include the uniform weight uniform cost case. In fact, in the uniform cost case the CAST structure can be replaced with just one simple cycle (in G_T) at the expense of a larger constant than $4/3$. These extensions are described in Section 3.3.

3.2 Proof of basic structural theorem

The proof of Theorem 3.1 consists of three steps, as follows. Let $(\hat{A}, \hat{B}, \hat{C})$ be any vertex-cut in a plane graph G_0 (e.g., a minimum quotient cut). We first show in Section 3.2.1 that there exists in G_0 a vertex-cut $(\hat{A}', \hat{B}', \hat{C}')$ that has nearly the same quotient cost and for which $\hat{A}'$ corresponds to some "connected region" in the plane. We then prove in Section 3.2.2 that $\hat{B}'$ must correspond to a union of "connected regions" in the plane, each described by a simple directed cycle (in G_T), with the important property that these simple cycles form a forest (in a sense to be defined). Using averaging arguments, we show in Section 3.2.3 that every such forest must contain a shallow subtree (namely, subtree of constant depth) that defines a vertex-cut $(\hat{A}'', \hat{B}'', \hat{C}'')$ whose quotient cost is nearly the same.

Letting $(\hat{A}, \hat{B}, \hat{C})$ be a minimum quotient cut in G_0, the three steps above yield a cut $(\hat{A}'', \hat{B}'', \hat{C}'')$ that has a nearly optimal quotient cost and also has a CAST structure, thus proving the main result of this section. The three steps above apply for 0-quotient cost (i.e. $\alpha = 0$), but the results easily extend to 1-quotient cost (i.e., $\alpha = 1$) by applying Lemma 2.4.

3.2.1 A connected region in the plane

The first step in the proof of the basic structural theorem is to exhibit in G_0 a vertex-cut $(\hat{A}', \hat{B}', \hat{C}')$ that has nearly the same quotient cost as $(\hat{A}, \hat{B}, \hat{C})$ and for which $\hat{A}'$ corresponds to some "connected region" in the plane. Intuitively, the vertex-cut $(\hat{A}, \hat{B}, \hat{C})$ partitions the plane into connected regions, each associated with one of $\hat{A}, \hat{B}, \hat{C}$; a careful rearrangement of the connected regions associated with $\hat{A}$ and $\hat{B}$ yields the desired vertex-cut $(\hat{A}', \hat{B}', \hat{C}')$. Below we give the formal argument.

We start by labeling each face of G_T by $\hat{A}$ if its boundary contains a vertex of $\hat{A}$, by $\hat{B}$ if its boundary contains a vertex of $\hat{B}$, and by $\hat{C}$ if neither event happens. It is easy to verify that every face of G_T has exactly one label.

Let $\mathrm{CC}(\hat{A})$ denote the set of connected components of the faces labeled $\hat{A}$ (recall Definition 2.1), and similarly for $\hat{B}$ and $\hat{C}$. Let $\mathrm{CC}(G_T) := \mathrm{CC}(\hat{A}) \cup \mathrm{CC}(\hat{B}) \cup \mathrm{CC}(\hat{C})$ be the set of all these connected regions. Each connected region $R \in \mathrm{CC}(G_T)$ corresponds to a set of vertices in G_0, namely,

$$V(R) := \{v \in V_0 : v \text{ is in the boundary of a face in } R\}. \quad (1)$$

We define the *weight of a connected region* R to be $\tilde{w}(R) := \sum_{v \in V(R) \setminus C} w(v)$. Notice that ignoring the weight of vertices from C corresponds to 0-quotient cost (i.e., $\alpha = 0$). For a set S of connected regions, define $V(S) := \cup_{R \in S} V(R)$ and $\tilde{w}(S) := \sum_{R \in S} \tilde{w}(R)$. We say that a face f is in S if $f \in R$ for some connected region $R \in S$. It is easy to verify that

$$\tilde{w}(\mathrm{CC}(\hat{A})) = w(\hat{A}),\ \tilde{w}(\mathrm{CC}(\hat{B})) = w(\hat{B}),\ \tilde{w}(\mathrm{CC}(\hat{C})) = 0,$$

and it then immediately follows that $\tilde{w}(\mathrm{CC}(G_T)) = \tilde{w}(\mathrm{CC}(\hat{A})) + \tilde{w}(\mathrm{CC}(\hat{B})) + \tilde{w}(\mathrm{CC}(\hat{C})) = w(\hat{A} \cup \hat{B})$.

PROPOSITION 3.2. *There is a partition $\mathrm{CC}(G_T) = S \cup \bar{S}$ such that the faces of S form a connected region and the balance between $S, \bar{S}$ corresponds to the balance between $\hat{A}, \hat{B}$ as follows: $\min\{\tilde{w}(S), \tilde{w}(\bar{S})\} \geq \frac{3}{4}\min\{w(\hat{A}), w(\hat{B})\}$.*

PROOF. Without loss of generality assume that $w(\hat{A}) \geq w(\hat{B})$ and let $\beta := w(\hat{B})/w(\hat{A} \cup \hat{B}) \leq \frac{1}{2}$. We say that two connected regions R_1, R_2 are *adjacent* if they contain two faces $f_1 \in R_1$, $f_2 \in R_2$ that are adjacent in G_T. Apply the following procedure on $\mathrm{CC}(G_T)$. Start with $S_1 = \{R_{\max}\}$ where $R_{\max} :=$

$\mathrm{argmax}_{R \in \mathrm{CC}(G_T)} \tilde{w}(R)\}$. If $\beta > \frac{3}{8} w(\hat{A} \cup \hat{B})$ then let $S_2 = \{R'_{\max}\}$ where $R'_{\max} := \mathrm{argmax}_{R \in \mathrm{CC}(G_T) \setminus S_1} \tilde{w}(R)\}$; otherwise, set $S_2 = \emptyset$, thus ignoring it throughout this procedure. Now repeatedly add to either S_1 or S_2 a connected region $R \in \mathrm{CC}(G_T) \setminus (S_1 \cup S_2)$ that is adjacent to at least one connected region in S_1 or S_2, respectively. Stop this iterative process when $\tilde{w}(S_i) \geq \max\{\frac{1-\beta}{2}, \frac{1}{8} + \frac{\beta}{2}\} w(\hat{A} \cup \hat{B})$ for some $i \in \{1, 2\}$, letting $S := S_i$ and $\bar{S} := \mathrm{CC}(G_T) \setminus S$. By construction, the faces in S forms a connected region.

We first claim that at every iteration there exists a connected region $R \in \mathrm{CC} \setminus (S_1 \cup S_2)$ that is adjacent to some connected region in either S_1 or S_2. To this end, observe that $0 \leq \beta \leq \frac{1}{2}$, so whenever $\tilde{w}(S_1), \tilde{w}(S_2) < \max\{\frac{1-\beta}{2}, \frac{1}{8} + \frac{\beta}{2}\} w(\hat{A} \cup \hat{B})$, we also have that $\tilde{w}(S_1) + \tilde{w}(S_2) < \max\{1-\beta, \frac{1}{4} + \beta\} w(\hat{A} \cup \hat{B}) \leq \tilde{w}(\mathrm{CC}(G_T))$, and thus $\mathrm{CC}(G_T) \setminus (S_1 \cup S_2)$ is not empty, i.e., contains at least one connected region. Let f be a face in such a connected region of $\mathrm{CC} \setminus S$, and let f' be a face in a connected region of S_1. Since the set of all faces in G_T is connected (in the sense of Definition 2.1), there is a sequence of faces $f = f_1, \ldots, f_k = f'$ such that f_i and f_{i+1} are adjacent for all $i < k$. It follows that there must exist two successive faces f_i in $\mathrm{CC} \setminus (S_1 \cup S_2)$ and f_{i+1} in either S_1 or S_2. The claim follows.

The above procedure always terminates since the number of iterations is bounded by the number of connected regions in $\mathrm{CC}(G_T)$, and in turn, by the (finite) number of faces in G_T. Finally, some calculation shows that $\min\{\tilde{w}(S), \tilde{w}(\bar{S})\} \geq \frac{3}{4} \min\{w(\hat{A}), w(\hat{B})\}$. In fact, if $\beta \leq \frac{1}{3}$ then the stronger result that $\min\{\tilde{w}(S), \tilde{w}(\bar{S})\} \geq \min\{w(\hat{A}), w(\hat{B})\}$ holds. $\square$

Finally, let $S, \bar{S}$ be as in Proposition 3.2. We use (1) to define the following subsets of V_0:

$$\hat{A}' := V(S) \setminus V(\bar{S}), \;\; \hat{B}' := V(\bar{S}) \setminus V(S), \;\; \hat{C}' := V(S) \cap V(\bar{S}). \tag{2}$$

The next easy to verify proposition summarizes the first step in the proof of the basic structural theorem.

PROPOSITION 3.3. *$(\hat{A}', \hat{B}', \hat{C}')$ is a vertex-cut of G_0 with $\hat{C}' \subseteq \hat{C}$ and $\min\{w(\hat{A}'), w(\hat{B}')\} \geq \frac{3}{4} \min\{w(\hat{A}), w(\hat{B})\}$, where $\hat{A}'$ corresponds to the connected region S.*

3.2.2 A forest of simple cycles

The second step in the proof of the basic structural theorem is to show that $\hat{B}'$ corresponds to a union of connected regions in the plane, each described by a simple cycle (in G_T), with the important property that these simple cycles form a "forest". Recall that $\hat{A}'$ and $\hat{B}'$ are defined in (2) so that they essentially correspond to S and $\bar{S}$, respectively. We show that since S is a connected region (by construction), the connected regions of $\bar{S}$ are arranged in a "forest". Below we give the formal argument. (Recall the definitions in Section 3.1.)

Slightly abusing notation, we denote the set of connected components of all the faces in the connected regions in $\bar{S}$ by $\mathrm{CC}(\bar{S}) := \mathrm{CC}(\cup_{R \in \bar{S}} \mathrm{CC}(R))$. It can be shown that for every connected region $R \in \mathrm{CC}(\bar{S})$ there is a directed simple cycle D_R such that R is exactly the set of faces (of G_T) that are enclosed by D_R. (We remark that the fact that each cycle is simple relies on S being a connected region.) Let $\mathcal{D}(\bar{S})$ be the set of directed simple cycles D_R that enclose the connected regions $R \in \mathrm{CC}(\bar{S})$. It is easy to verify that $\mathcal{D}(\bar{S})$ is a proper set of cycles that defines (in the sense of Section 3.1) the vertex-cut $(\hat{A}', \hat{B}', \hat{C}')$ described in (2).

The description of the cut $(\hat{A}', \hat{B}', \hat{C}')$ using the directed cycles $\mathcal{D}(\bar{S})$ allows us to investigate the arrangement of these cycles in the plane, and prove the following proposition. The main idea is that the existence of a "cycle of cycles" implies that S contains at least two connected regions (see Figure 6).

PROPOSITION 3.4. *Let $\tilde{G}(\tilde{V}_1, \tilde{V}_2, \tilde{E})$ be the auxiliary graph of $\mathcal{D}(\bar{S})$. Then, $\tilde{G}$ is a forest (i.e., contains no cycles).*

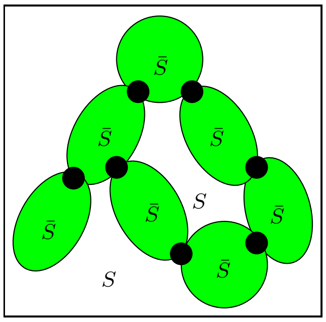

Figure 6: Cycle of cycles

3.2.3 A shallow tree of simple cycles

The third step in the proof of the basic structural theorem is to show that the forest $\tilde{G}$ must contain a subtree that is shallow (i.e., has small depth) and its cycles define a vertex-cut whose quotient cost is nearly the same (as that of $(\hat{A}', \hat{B}', \hat{C}')$). The main idea here is to remove every $2d$-th level in the forest $\tilde{G}$ (so that we lose at most $1/d$ of the cut's weight and do not increase its cost) and then apply averaging arguments on the resulting connected components. We can thus prove the following proposition.

PROPOSITION 3.5. *Let $d \geq 1$ be an integer. Then, there is a d-CAST in $\mathcal{D}(\bar{S})$ that defines a vertex-cut $(\hat{A}'', \hat{B}'', \hat{C}'')$ whose 0-quotient cost (i.e., $\alpha = 0$) is at most $(1 + \frac{1}{d})$ times that of $(\hat{A}', \hat{B}', \hat{C}')$.*

We can now prove Theorem 3.1.

PROOF OF THEOREM 3.1. Let $(\hat{A}, \hat{B}, \hat{C})$ be a vertex-cut of minimum 0-quotient cost (i.e., $\alpha = 0$). The result then follows from Propositions 3.3 and 3.5. $\square$

3.3 Extensions

We can extend Theorem 3.1 in various directions. First, as mentioned just prior to Theorem 3.1, for $\alpha > 1$ we get a factor of $\frac{4}{3} \frac{d+1}{d} + 1$ in the general case. The loss in the quotient cost is actually smaller, if there exists an optimal quotient cut (A^*, B^*, C^*) with $w(C^*) \ll w(A^*), w(B^*)$; in particular, in the uniform weight uniform cost case there is always a cut (A^*, B^*, C^*) with optimal quotient such that $w(C^*) \leq O(1/\sqrt{n}) \leq \min\{w(A^*), w(B^*)\}$ (see Section 2.2), and thus the loss in the 1-quotient cost (i.e., for $\alpha = 1$) becomes arbitrarily close to $4/3$. This result follows by combining Theorem 3.1 and Lemma 2.3.

Second, for general weights we reduce the loss in the quotient cost to $1 + \epsilon$, for arbitrary fixed $\epsilon > 0$, by somewhat relaxing the structural requirement and allowing a more complicated structure than a CAST. Technically, we generalize Proposition 3.3 by allowing $\hat{A}'$ (and S) to correspond to $O(1/\epsilon)$ connected regions. The proof is a modification of that of Proposition 3.2 that distinguishes between large weight and small weight connected regions.

Third, we show for the uniform cost case that the CAST structure can be replaced with just one simple cycle (in G_T) at the expense of a larger constant factor loss in the quotient cost. Technically, we replace a 1-CAST structure with one of its simple cycles.

Finally, we consider b-limited quotient cuts and show that in the uniform weight uniform cost case the loss in the b-limited 1-quotient cost is arbitrarily close to 1.

PROPOSITION 3.6. *If (A^*, B^*, C^*) is an optimal α-quotient cut for G_0 with $\lambda w(C^*) \leq w(A^*), w(B^*)$ for some $\lambda > 0$, then the factor in Theorem 3.1 can be improved to $\frac{4}{3}\frac{d+1}{d}(1 + \frac{\alpha}{\lambda})$.*

PROPOSITION 3.7. *There exists a vertex-cut $(\hat{A}', \hat{B}', \hat{C}')$ of G_0 where $\hat{A}'$ corresponds to $O(1/\epsilon)$ connected regions, $\hat{C}' \subseteq \hat{C}$ and $\min\{w(\hat{A}'), w(\hat{B}')\} \geq (1-\epsilon)\min\{w(\hat{A}), w(\hat{B})\}$.*

PROPOSITION 3.8. *Let $\alpha \geq 0$. Then, every uniform cost planar graph G_0 has a vertex-cut that is either defined by a directed cycle or is trivial such that its α-quotient cost is within a factor of $6\frac{1}{3}$ from the minimum for $\alpha \geq 0$.*

PROPOSITION 3.9. *Let $b \leq \frac{1}{3}$, $\alpha \geq 0$. If (A^*, B^*, C^*) is an optimal b-limited α-quotient cut with $\lambda w(C^*) \leq w(A^*), w(B^*)$ for some $\lambda > 0$ for G_0, then the factor in Theorem 3.1 can be improved to $\frac{d+1}{d}(1 + \frac{\alpha}{\lambda})$ for b-limited α-quotient cost.*

4. ALGORITHM FOR STRUCTURED CUTS

In this section we use the structural theorems of Section 3 to devise an algorithm for finding in planar graphs a cut with near-optimal 1-quotient cost (i.e., $\alpha = 1$). Given a planar input graph G, we first fix an embedding G_0 of it in the plane and compute its face-vertex graph G_T (see Section 2). By applying the algorithm described below we then, roughly speaking, search for a d-CAST structure in G_T (see Section 3.1 for definition) with minimum α-quotient. The structural theorems guarantee that a search restricted to d-CASTs will find a vertex-cut whose 1-quotient is within a small constant factor of the minimum. Formally, we prove the following theorem.

THEOREM 4.1. *Let $d \geq 1$ and let G_0 be a plane graph. Then, there is an algorithm that runs in time $O(W \cdot n^{3+2d})$ and finds a vertex-cut (A', B', C') in G_0 whose 1-quotient cost (i.e. $\alpha = 1$) is at most the 0-quotient cost of any vertex-cut (A, B, C) that is defined by a d-CAST in G_T, i.e.,*

$$\frac{c(C')}{\min\{w(A'), w(B')\} + w(C')} \leq \frac{c(C)}{\min\{w(A), w(B)\}}.$$

Algorithm outline. We start by translating vertex weights in G_T to face weights in G_T (Section 4.1); these face weights are defined so that the total weight of the faces enclosed by any simple directed cycle approximates the total weight of the vertices enclosed by this cycle; actually, the same holds for any proper set of such cycles. Now, similarly to [18], we construct a search graph G_s, which is just a directed version of G_T with edge weights (Section 4.2); these edge weights are defined so that the total weight of the edges on the cycle on any simple directed cycle is equal to the total weight of the faces that this cycle encloses. Finally, our main routine finds among a certain family of closed walks (i.e., cycles, but not necessarily simple ones) in G_s, a walk that has minimum cost to weight quotient; here, the weight of a walk is the sum of the weights of its edges and the cost of a walk is (in principle) the sum of the costs of the vertices it visits.

This family of walks (defined in Section 4.3) has three important properties. First, for any d-CAST there is in this family a corresponding walk whose weight and cost are equal to that of the d-CAST (Section 4.4). Second, we can efficiently optimize the cost to weight quotient over this family using dynamic programming (Section 4.5). However, this family contains also walks that do not correspond to a CAST, so the third important property is that from any walk in this family we can extract efficiently a vertex-cut (that need not be a d-CAST) whose quotient cost is no larger than that of the whole walk (Section 4.6).

Our algorithm is inspired by those of [19, 20, 18] for finding (or approximating) the minimum quotient edge-cut, and we point out some of the similarities and differences throughout the description.

4.1 Assigning weight to faces

We now translate vertex weights into face weights (in G_T), by distributing the weight of every vertex from V_T (or actually, V_0) equally among the faces of G_T incident to it. Formally, the weight of face f of G_T is $\sum_{v \in f} w(v)/\deg(v)$, where $\deg(v)$ is the degree of v in G_T. The analogous transformation for edge-quotient cuts [19, 20, 18] is simpler, since there each face in the dual of G_0 corresponds to a vertex in G_0, and is thus assigned the weight of that vertex. The next proposition shows that the weight transformation allows, in principle, to approximate the total weight cut by a cycle (i.e. by a cut that is defined using a cycle) by using the total weight of the faces that are cut by this cycle.

PROPOSITION 4.2. *Let D be a (not necessarily simple) cycle in G_T and let C denote the set of vertices on it. Let l denote the number of regions in $\mathbb{R}^2 \setminus D$, and let V_i and F_i denote the sets of vertices and faces, respectively, in the ith region. Then, for all $I \subseteq \{1, \ldots, l\}$, we have $w(\cup_{i \in I} V_i) \leq w(\cup_{i \in I} F_i) \leq w(\cup_{i \in I} V_i) + w(C)$.*

4.2 The Search Graph

Similar to [18], we define the *search graph* G_s to be the digraph obtained from G_T by replacing every undirected edge (u, v) by a pair of directed edges (u, v) and (v, u). Let the vertices of G_s have the same cost they have in G_T. In the rest of this subsection we associate weight with the (directed) edges of G_s.

Let T be any spanning tree of the graph G_T. Choosing any vertex r on the outer face as a root, orient the tree according to this embedding. That is, order the children of any vertex in a counterclockwise direction; for the root vertex r start with the outer face, and for any other vertex start with the edge leading to its parent.

The tree edges incident to a vertex v divide the plane immediately surrounding v into $\deg(v)$ regions. We give each such region a unique labeling as follows: Starting at the root, traverse the tree in a Depth First Search manner, using the order on children defined above. Each step in this traversal corresponds to visiting a vertex from either its parent or from a child. Thus, step 1 is visiting the root, step 2 is visiting the root's first child and so on. Each time a vertex is visited in this traversal, one immediate region around v, namely, the one between the edges used to enter and to exit v, is labeled with the traversal's step number. Formally, for a vertex v let e_1 be the edge to its parent and let $e_2, \ldots, e_{\deg_T(v)}$ be edges to its children, where $\deg_T(v)$ is the degree of v in the tree T. We then label the region between e_i and $e_{(i+1) \bmod \deg_T(v)}$ by the step of traversal at which vertex v is visited for the ith time, denoted $t(v, i)$. See for example Figure 7.

We give weight to every directed edge (u, v) in G_s as follows. If the underlying undirected edge belongs to the tree T, we define $w(u, v) = 0$; otherwise, (u, v) is embedded in some region relative to each of its endpoints. Let $t(u, i)$ and $t(v, j)$ be the labels of

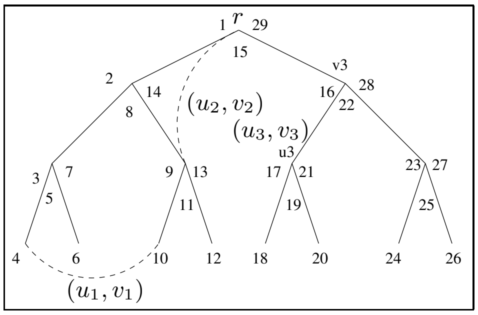

Figure 7: Labeling the regions of the tree T.

these regions. Assume without loss of generality that $t(u,i) < t(v,j)$, and let $w(u,v)$ be the total weight of the faces enclosed by the simple cycle that (u,v) closes in the tree T (i.e., take the unique path in T between u and v and add to it the edge (u,v)); let $w(v,u) = -w(u,v)$. Thus, edges going left-to-right have positive weight and their reverse edges have negative weight.

Park and Phillips [18] showed that the weight of a simple counter-clockwise (respectively clockwise) cycle in G_s is equal to (respectively, minus) the weight of the faces enclosed by the cycle.

4.3 Walks with pebbles

We now define the family of walks that will be used to prove Theorem 4.1. We shall also define for each such walk certain quantities such as total weight, true cost, total cost and counted cost.

A *walk* of *length* k in G_s is a sequence $(u_1, v_1), \dots, (u_k, v_k)$ of (directed) edges, such that $v_i = u_{i+1}$ for all $1 \le i < k$. The walk is *closed* if $v_k = u_1$. (Notice that a closed walk is just a directed cycle.) A closed walk is *simple* if it contains k distinct edges and k distinct vertices; A non-closed walk is *simple* if it contains k distinct edges and $k+1$ distinct vertices. For a walk π, let π_{e_i} (for $i = 1, \dots, k$) denote the ith edge in the walk (namely, (u_i, v_i)), and let π_i (for $i = 1, \dots, k+1$) denote the ith vertex in the walk (namely, $u_i = v_{i-1}$). The *total weight* of a walk π of length k is $w(\pi) = \sum_{i=1}^{k} w(\pi_{e_i})$, and its *total cost* is $\sum_{i=1}^{k} c(\pi_i)$. Notice that in these definitions, if the walk is not simple then some costs and weights may be counted more than once. We define the *actual cost* of a walk π of length k as the sum of costs of all the *distinct* vertices in $\{\pi_1, \dots, \pi_k\}$. For edge quotient cuts, the total (edge) cost is used in [18], since there it suffices to seek a simple cycle in G_s, and for such a walk the total cost is clearly equal to the actual cost. In contrast, for vertex-cuts we shall seek a walk that corresponds to a CAST, and thus might visit some vertices more than once. For such a walk, the total cost might be a poor upper bound on the actual cost. We thus define below a family of walks that allows, for a limited number of vertices, additional visits to these vertices at no additional cost; however, we shall limit these free visits to be in a counter-clockwise order, without completing a full cycle.

A *d-pebble walk* is a walk that uses at most d pebbles as follows. In the beginning of the walk all d pebbles are unassigned, and during the walk a pebble may be assigned (to some vertex) or unassigned only according to these rules: (i) when the walk enters a vertex v that has no pebble (assigned to it), the walk must assign to v one of the unassigned pebbles; (ii) when the walk leaves a vertex v that has a pebble (assigned to it), the walk may, but does not have to, unassign the pebble from v. Of course, one pebble allows to "traverse" any walk, by simply assigning the pebble upon entrance to a vertex and then immediately unassigning it when leaving the vertex. However, the pebbles allow for a tighter upper bound of the actual cost of the walk, as follows. Let us call a visit to a vertex that already has a pebble (assigned to it) a *free visit*, and let the *counted cost* of a walk π be $\sum c(\pi_i)$, where the summation is over all the non-free visits $1 \le i \le k$. Clearly, the counted cost of a walk is always an upper bound on its actual cost.

In d-pebble walks we restrict the free visits to have a certain order in the plane. Specifically, between the moment that a pebble is assigned to v and the moment in which it is unassigned from v, the walk uses edges incident to v only in a counter-clockwise order around v without completing a full cycle (this includes both incoming and outgoing edges). Formally, every pebble is represented by a *tuple* $\langle v, e_{\text{first}}, e_{\text{last}} \rangle$, where v is the vertex to which the pebble is assigned, e_{first} is the incoming edge to v that was used by the walk to assign the pebble to v, and e_{last} is the last edge that was used by the walk among all edges incident to v (incoming and outgoing edges). An unassigned pebble is represented by an empty tuple $\langle\rangle$.

4.4 A CAST yields a pebble walk

We next show that every d-CAST yields a d-pebble walk with a certain correspondence between their weights and costs. Notice that a directed cycle (in the CAST) may enclose the infinite face in G_T, and then the total weight of a walk along this cycle is negative.

PROPOSITION 4.3. *Let (A, B, C) be a cut corresponding to a d-CAST, for some $d \ge 1$. Then, there exists a $(d+1)$-pebble walk with counted cost $c(C)$ and with total weight w_0 such that $\min\{|w_0|, W - |w_0|\} \ge \min\{w(A), w(B)\}$. Furthermore, the length of this walk is at most $|E_T| \le 9n$ and the total weight up to any point in the walk is always in the range $[-W, W]$.*

4.5 A dynamic programming algorithm

We now use dynamic programming to find a walk of minimum counted cost among all $(d+1)$-pebble walks of length at most $|E_T| \le 9n$ and total weight $w \in \{-W, \dots, W\}$. Formally, we define the following dynamic programming table T. We define $T(l, s, t, \{p_1, \dots, p_{d+1}\}, w)$ to be the minimum cost of a $(d+1)$-pebble walk of length $0 \le l \le |E_T|$ that starts at s, ends at t, its pebble assignments (at the end of the walk) are $p_1, \dots, p_{d+1}$, and its total weight is exactly $w \in \{-W, \dots, W\}$. Recall that each pebble p_i is represented by a tuple that takes one of $O(n^2)$ different values.

PROPOSITION 4.4. *Every entry for $l = 0$ can be computed in time $O(1)$, and every entry of T for $l+1$ can be computed from (some of the) entries of T for l in time $O(n)$. Thus, we can compute all entries of the form $T(l, v, v, \{\langle v, e_{\text{first}}, e_{\text{last}} \rangle, \langle\rangle, \dots, \langle\rangle\}, w)$ with $1 \le l \le |E_T|$ and $w \in \{-W, \dots, W\}$ in time $O(W \cdot n^{2d+6})$.*

REMARK. The running time can be improved by using specific properties of the walk guaranteed by Proposition 4.3 in order to avoid redundancies in the table T. For instance, the starting vertex s can be encoded by the first pebble, and the end vertex t can encode the last (currently used) pebble. This results in an algorithm that takes $O(W \cdot n^{2d+3})$ time. The algorithm can be made even faster by using Dijkstra's shortest path algorithm, but the dependency of the running time on d remains the same. The full details are omitted from this version of the paper and resemble a construction in [18].

When the table T is complete, we take the entry that minimizes $T(l, v, v, \emptyset, w) / \min\{|w|, W - |w|\}$ over all $v \in V_T$ and $-W < w < W$. We can compute from this entry a d-pebble walk π that has total weight w and counted cost $T(l, v, v, \emptyset, w) / \min\{|w|, W - |w|\}$.

4.6 A pebble walk yields a vertex-cut

We now show that any pebble walk (and, in particular, the one found in the dynamic program) gives rise to a vertex-cut with equal or smaller quotient. The main hurdle here is that a pebble walk need not correspond directly to a CAST or any vertex-cut; even if it does, the total weight of the walk need not correspond to the total weight that this cut separates. For instance, if the walk is the union of several directed simple cycles, some faces may be enclosed by more than one of these (counter-clockwise) cycles.

We shall prove that any pebble walk can be decomposed into sub-walks such that the counted cost and total weight decompose in an appropriate way. A careful averaging argument due to [18] then shows that the quotient in at least one of these sub-walks must be not larger than that of the complete walk. This decomposition also ensures that every sub-walk corresponds to a vertex-cut (with the suitable weight and cost). In fact, this decomposition can be found efficiently, and thus one can extract from the walk found in the dynamic program a vertex-cut of small quotient cost. For edge quotient cuts, the total (edge) cost is used in [18] and then any decomposition of the walk into sub-walks clearly decomposes the total cost appropriately. For quotient vertex-cuts we need to make sure that the counted cost is decomposed appropriately.

The *net-count* of a face f by a walk π (in G_s), denoted $\mathrm{nc}_\pi(f)$, is the (net) number of times that the weight of the face f contributes to $w(\pi)$, the total weight of π. (See Section 4.3 for definition of $w(\pi)$.) The *number of visits* of a walk π to a directed edge e (of G_s), denoted $\mathrm{nv}_\pi(e)$, is the number of times that e appears in the walk π.

For a set A of integers, let $\Delta(A) := \max A - \min A$. For a closed walk π, let $\Delta_\pi(G_s) := \Delta(\{\mathrm{nc}_\pi(f) : f \text{ a face in } G_s\})$. For a vertex v, let $\Delta_\pi(v)$ be the difference between the least-counted face and the most-counted face among those incident to v, i.e., $\Delta_\pi(v) := \Delta(\{\mathrm{nc}_\pi(f) : f \in F_v\})$ where F_v is the set of faces of G_s incident on v.

Our procedure for decomposing a pebble closed walk into closed sub-walks can be sketched as follows. First, we label each face f of G_s with $\mathrm{nc}_\pi(f)$. Then, we create a set S that includes all faces of G_s that have the highest label. Viewing S as a cut (every vertex incident with a face in S is either in C or A, and everything else is in B) we check if its quotient is no higher than that found for π in T. If so, then we are done (S defines the sought cut). If the quotient of S is worse, then we decrease the labels of faces in S by 1. Now we repeat the process by selecting a new set S in the resulting graph and so forth.

We next argue in Proposition 4.5 about two properties of this decomposition procedure. Then, the first property and Proposition 4.6 together imply that the counted cost of the walk is decomposed appropriately to the sub-walks. (The appropriate decomposition of the total weight is immediate.) Finally, the second property and Proposition 4.7 together imply that every such sub-walk corresponds to a vertex-cut.

PROPOSITION 4.5. *Every closed pebble walk π in G_s is the union of closed (but not necessarily pebble) walks π^j, such that (i) every vertex v appears in at most $\Delta_\pi(v)$ of these walks, and (ii) $\Delta_{\pi^j}(G_s) \le 1$ for every walk π^j.*

PROPOSITION 4.6. *Let π be a closed pebble walk. Then, every vertex v contributes its cost to the counted cost of π at least $\Delta_\pi(v)$ times.*

PROPOSITION 4.7. *Every walk π with $\Delta_\pi(G_s) \le 1$ defines a vertex-cut (A, B, C) such that $c(C)$ is the actual cost of π and $\min\{w(A), w(B)\} + w(C) \ge \min\{w(\pi), W - w(\pi)\}$.*

By combining the three propositions above and the averaging arguments of Park and Phillips [18, Theorem 2.2] we obtain the next corollary.

COROLLARY 4.8. *Let π be a closed pebble walk with total weight w' and counted cost c'. Then, there is in G_T a vertex-cut whose 1-quotient cost (i.e., $\alpha = 1$) is at most $\frac{c'}{\min\{|w'|, W-|w'|\}}$.*

Finally, we prove Theorem 4.1.

PROOF OF THEOREM 4.1. Let (A, B, C) be a cut defined by a d-CAST and consider the algorithm described throughout Section 4. By Proposition 4.3 there exists a $(d+1)$-pebble walk π of total weight w_0 and counted cost c_0, such that $\frac{c_0}{\min\{|w_0|, W-|w_0|\}} \le \frac{c(C)}{\min\{w(A), w(B)\}}$. By the definition of the dynamic programming table, there is $v \in V_T$ and $l \le |E_T|$ such that $T(l, v, v, \emptyset, w_0) \le c(C)$. Let π' be the $(d+1)$-pebble walk found by the dynamic program, and denote its total weight by w' and its counted cost by c'. Then, we have that $\frac{c'}{\min\{|w'|, W-|w'|\}} \le \frac{c_0}{\min\{|w_0|, W-|w_0|\}}$. We conclude by Corollary 4.8 that the algorithm finds in G_T a vertex-cut whose 1-quotient cost is at most $\frac{c'}{\min\{|w'|, W-|w'|\}}$, and in particular is at most $1 + \frac{1}{\lambda}$ times that of (A, B, C). $\square$

5. IMPLICATIONS AND APPLICATIONS

5.1 Quotient vertex-cuts

Combining Theorem 3.1, and Theorem 4.1 we obtain the following result.

THEOREM 5.1. *There is an algorithm that runs in time $O(W \cdot n^{3+2d})$ and finds in plane graphs a vertex-cut whose 1-quotient cost is within a factor of $1 + \frac{4}{3}\frac{d+1}{d}$ from the minimum. If, in addition, there is a minimum 1-quotient cost vertex-cut (A^*, B^*, C^*) with $\lambda w(C^*) \le \min\{w(A^*), w(B^*)\}$ then the factor is at most $\frac{4}{3}\frac{d+1}{d}(1 + \frac{1}{\lambda})$.*

PROOF. Let (A^*, B^*, C^*) be a vertex-cut of minimum 1-quotient cost. We start by proving the second part of the theorem. Assume that $\lambda w(C^*) \le \min\{w(A^*), w(B^*)\}$. Then, this cut's 0-quotient cost is $\frac{c(C^*)}{\min\{w(A^*), w(B^*)\}} \le \frac{(1+1/\lambda)c(C^*)}{\min\{w(A^*), w(B^*)\}+w(C)}$, so by Theorem 3.1 there is a vertex-cut (A, B, C) that is defined by a d-CAST in G_T and its 0-quotient cost is at most $\frac{4}{3}\frac{d+1}{d}\frac{c(C^*)}{\min\{w(A^*), w(B^*)\}}$. Theorem 4.1 then guarantees that the algorithm of Section 4 finds a vertex-cut whose 1-quotient cost is not larger, namely, at most $\frac{4}{3}\frac{d+1}{d}(1 + \frac{1}{\lambda})\frac{c(C^*)}{\min\{w(A^*), w(B^*)\}+w(C)}$.

Now, for the first part, if $\lambda w(C^*) \ge \min\{w(A^*), w(B^*)\}$ then there is a trivial cut, (A, B, C), of 1-quotient cost $\frac{c(C)}{w(C)} \le \frac{c(C^*)}{w(C^*)} \le \frac{(1+\lambda)c(C^*)}{\min\{w(A^*), w(B^*)\}+w(C)}$, and an exhaustive search finds such a cut in time $O(n)$. If $\lambda w(C^*) \le \min\{w(A^*), w(B^*)\}$ then the above algorithm finds a cut whose 1-quotient cost is within a factor of $\frac{4}{3}\frac{d+1}{d}(1 + \frac{1}{\lambda})$ from the minimum.

Therefore, for any $\lambda > 0$, we can approximate the minimum 1-quotient cost within a factor of $\max\{1 + \lambda, \frac{4}{3}\frac{d+1}{d}(1 + \frac{1}{\lambda})\}$. To minimize this term we choose $\lambda = \frac{4}{3}\frac{d+1}{d}$, and then the approximation ratio we obtain is $1 + \frac{4}{3}\frac{d+1}{d}$. $\square$

Theorem 5.1 can be extended as follows to b-limited α-quotient cuts using Proposition 3.9 and extensions to Corollary 4.8 and Theorem 4.1.

THEOREM 5.2. *Let $b \le \frac{1}{3}$. There is an algorithm that runs in time $O(W \cdot n^{3+2d})$ and finds in plane graphs a vertex-cut whose* *b-limited* 1*-quotient cost is within a factor of* $1+\frac{d+1}{d}$ *from the minimum. If, in addition, there is a minimum b-limited* 1*-quotient cost vertex-cut* (A^*, B^*, C^*) *with* $\lambda w(C^*) \leq \min\{w(A^*), w(B^*)\}$ *then the factor is at most* $\frac{d+1}{d}(1+\frac{1}{\lambda})$.

5.2 From quotient cuts to separators

The following shows that we can find a pseudo-approximation to a vertex separator using an approximation algorithm for a minimum-quotient vertex-cut.

THEOREM 5.3. *Let* $1 \geq b \geq b' > 0$ *and* $0 < \alpha \leq 1$ *such that* $b'/\alpha \leq \min\{\frac{1}{3}, b\}$. *Given a* ρ*-approximation algorithm for minimum b-limited* α*-quotient cost vertex-cuts one can find a* (b', α)*-balanced cut that is within a factor of* $\frac{\rho}{b-b'/\alpha}$ *from the minimum* (b, α)*-balanced cut.*

As a simple result, for uniform weight uniform cost planar graphs, setting $\alpha = 1$, $b = \frac{1}{3}$ and $b' = \frac{1}{6}$, we obtain a $\frac{5}{6}$-balanced cut whose cost is within a factor of $6\frac{d+1}{d}(1+\frac{O(1)}{\sqrt{n}})$ from the optimal $2/3$-balanced separator. This is done using our algorithm from Section 4 with total running time $O(n^{2d+5})$.

6. CONCLUDING REMARKS

In most of this paper we restricted ourselves to finding a bounded depth tree of cycles (a CAST), which gave rise to a factor of $\frac{4}{3}$ in our approximation (see Theorem 3.1). However, Proposition 3.7 suggests that a more general structure allows to discard this factor. We suspect that this structural result can be exploited algorithmically by a suitable extension of our dynamic programming algorithm, yielding a polynomial-time approximation scheme for the problem of minimum quotient vertex-cuts in planar graphs of uniform cost and uniform weight. We hope to present such an algorithm in the full version of this paper.

One application known for approximate vertex separators and quotient vertex-cuts are closely related to minimum width tree decompositions (of a graph) (aka treewidth) [21]. A known method for finding a tree decomposition of near optimal width is to recursively use a balanced vertex-cut algorithm, see e.g. [3]). Using this technique and Proposition 3.8 we can can find for planar graphs a tree decomposition whose width is within a constant factor from the optimal. The constant factor that we achieve is worse than the 1.5 given by an algorithm of [22], but we suspect that its running time can be made better than the $O(n^4)$ that is reported in [22]. We hope to present such a result in the full version of this paper.

7. REFERENCES

[1] C. Ababei, N. Selvakkumaran, K. Bazargan, and G. Karypis. Multi-objective circuit partitioning for cutsize and path-based delay minimization. In *International Conference on Computer-Aided Design*, pages 181–186, 2002.

[2] N. Alon, P. Seymour, and R. Thomas. Planar separators. *SIAM J. Discrete Math.*, 7(2):184–193, 1994.

[3] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01)*, pages 7–15. Morgan Kaufmann, 2001.

[4] E. Amir and S. McIlraith. Partition-based logical reasoning. In *Proc. 7th Int'l Conference on Principles of Knowledge Representation and Reasoning*, pages 389–400. Morgan Kaufmann, 2000.

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

[6] R. Diestel. *Graph theory*. Springer-Verlag, New York, second edition, 2000.

[7] S. Even. *Graph algorithms*. Computer Science Press Inc., 1979.

[8] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4):1090–1118, 2002.

[9] 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.

[10] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[11] 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.

[12] S. L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society B*, 50(2):157–224, 1988.

[13] 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.

[14] 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.

[15] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[16] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997.

[17] B. Mohar. Isoperimetric numbers of graphs. *J. Combin. Theory Ser. B*, 47(3):274–291, 1989.

[18] J. K. Park and C. A. Phillips. Finding minimum-quotient cuts in planar graphs. In *25th Annual ACM Symposium on Theory of Computing*, pages 766–775, May 1993.

[19] S. Rao. Finding near optimal separators in planar graphs. In *28th Annual Symposium on Foundations of Computer Science*, pages 225–237. IEEE, 1987.

[20] S. Rao. Faster algorithms for finding small edge cuts in planar graphs. In *24th ACM Symp. on Theory of Computing*, pages 229–240. ACM, 1992.

[21] N. Robertson and P. D. Seymour. Graph minors. II: algorithmic aspects of treewidth. *J. Algorithms*, 7:309–322, 1986.

[22] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[23] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[24] 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.

[25] D. A. Spielman and S.-H. Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *37th Annual Symposium on Foundations of Computer Science*, pages 96–105. IEEE, 1996.