Partitioning Graphs with Costs and Weights on Vertices: Algorithms and Applications *

Hristo N. Djidjev

Department of Computer Science, Rice University,
P.O. Box 1892, Houston, TX 77251, USA,
email: hristo@cs.rice.edu

Abstract

We prove separator theorems in which the size of the separator is minimized with respect to non-negative vertex costs. We show that for any planar graph G there exists a vertex separator of total vertex cost at most $c\sqrt{\sum_{v\in V(G)}(cost(v))^2}$ and that this bound is optimal within a constant factor. Moreover such a separator can be found in linear time. This theorem implies a variety of other separation results discussed in the paper. We describe application of our separator theorems to graph embedding problems, graph pebbling, and multi–commodity flow problems.

1 Introduction

Background. A separator is a small set of vertices or edges whose removal divides a graph into two roughly equal parts. The existence of small separators for some important classes of graphs such as planar graphs can be used in the design of efficient divide-and-conquer algorithms for problems on such graphs. Formally, a separator theorem for a given class of graphs S states that any n-vertex graph from S can be divided into components of no more than αn vertices, for some constant $\alpha < 1$, by removing a set called an α *separator* of no more than $f(n)$ vertices, where $f(n) = o(n)$. A classic result of Lipton and Tarjan [20] states that a separator theorem with $f(n) = O(\sqrt{n})$ holds for the class of planar graphs and that such a

*This work is partially supported by National Science Foundation grant CCR–9409191.

separator can be found in linear time. Lipton and Tarjan also described various applications of their separator theorem [21].

Numerous new results on graph separation and many new application have been found since. Separator theorems exist for the class of graphs of bounded genus [6, 10], for the class of chordal graphs [11], for the three-dimensional graphs [24], for the class of graphs that exclude a given minor [1], and for some geometrically defined graphs that include many three dimensional finite element graphs [23, 28]. Separators with special structure have been studied, including simple cycle separators [22, 9] and separators consisting of edges [5]. Leighton and Rao [17] have studies approximation algorithms for constructing separators of minimum size.

Separator theorems have been used for solving large sparse systems of linear equations [19, 12], for developing algorithms for VLSI layout design [4, 18], for shortest path problems [8], in parallel computing [13], and in computational complexity [21].

Our results. In many applications it is useful to associate costs with the vertices or the edges of the graph. For example, the costs might correspond to different sizes of circuit elements to be laid out or different amounts of data that have to be exchanged between communicating processes in a distributed system. For such applications, the size of the separator has to be measured by some function of the costs associated with its vertices (instead by its cardinality).

In this paper we consider graphs with costs and weights on the vertices or edges: the costs are used to evaluate the size of the separator and the weights are used to evaluate the sizes of the components into which the graph is divided. We give tight bounds on the minimum cost of a separator of a planar weighted graph and study related algorithmic problems. Our main theorem states that for any planar graph G with a non-negative real cost $cost(v)$ on each vertex v and non-negative real vertex weights adding to one there exists a vertex set of total cost no more than $\sqrt{8}\sqrt{\sum_{v\in V(G)}(cost(v))^2}$ whose removal leaves no component of weight exceeding 2/3. For the special case of graphs with equal vertex costs our theorem yields the same bound as the Lipton and Tarjan's planar separator theorem. We show that our bound is tight within a constant factor and give a linear algorithm for constructing such a separator.

Previously, weights on vertices and edges have been studied mostly for measuring the sizes of the components into which the graph is divided. Separator theorems of such type can usually be proved by generalizing the corresponding non-weighted versions in a straightforward way. However, these generalizations do not readily extend to the case where the size of the separator has also to be measured as a function of the weights.

In order to prove our separator theorem we define a new type of spanning tree called a *steep spanning tree* whose levels are real numbers and which generalizes a breadth first spanning tree in the case of graphs with vertex costs. The property of a steep spanning tree that is essential for separating a planar graph is that, intuitively, if for any real r one cuts along level r all edges either in the original graph, or in any of its steep spanning trees, the resulting components will have identical vertex sets.

Our constructive proofs can be transformed into an $O(n \log n)$ algorithm for finding a separator of an n-vertex planar graph in a straightforward manner. We show that actually it is possible to find a separator in $O(n)$ time, by avoiding the additional $O(\log n)$ factor that comes from sorting. Our technique illustrates that in certain cases sorting can be replaced by a binary search of a set of n reals that uses the linear algorithm for finding the median and exploits the structure of the graph.

We prove some other separator theorems that follow from our main result, some of which are needed in the application section. These include a separation into parts of weight not exceeding $1/2$, finding a separator that is a set of edges, and separation of graphs of bounded genus.

Applications. We discuss two applications of our separator theorems: for the problems of embedding planar graphs into various communication structures, for pebbling a planar directed acyclic graph, and for constructing a minimum routing tree. The graph embedding problem arises in the design of VLSI chips and in the simulation of different interconnection structures. We consider graphs with costs on the edges that model the volumes of information that has to be transfered between different processes. The goal is to minimize the total amount of traffic (viz., the average weighted dilation), if the original graph is implemented on the target structure. We show that any n-vertex planar graph of maximum degree d and non-negative edge costs whose squares sum up to n can be embedded into a binary tree, a hypercube, a butterfly graph, a shuffle-exchange graph, a cube-connected-cycles graph, or a de Bruin graph with $O(\log d)$ average weighted dilation.

Graph pebbling is used to study memory requirements of straight-line programs, for register allocation, and for deriving space-time tradeoffs. We show that any n-vertex planar acyclic directed graph G with non-negative vertex costs can be pebbled so that, at any time, the sum of the costs of all vertices with pebbles be $O(\sqrt{\sum_{v \in V(G)} (cost(v))^2} + I \log n)$. (Here I denotes the maximum sum of costs on the predecessors of any vertex of

G.) This result implies that, if G is a flow graph with costs on its vertices corresponding to the amounts of memory used by the corresponding steps, then it is possible to substantially reduce the total amount of memory needed to complete the task.

Our results on the above two problems match the best previously known bounds in the special case of graphs with unit vertex costs. No results have been known for these problems for graphs with costs on the vertices.

Finally, we consider applications of our results to the tree congestion problem. This a version of the multi–commodity flow problem, where one is given a set of demands represented by an edge-weighted graph G and the goal is to construct a network of small cost that can accommodate all demands. In the case studied in the paper, the network is restricted to be a tree and the cost is measured by the maximum edge capacity (called a *congestion*) of the tree. For a general graph G the problem of finding a tree with a minimum congestion (called a *minimum routing tree*) is NP–complete. We study the case for planar graphs G and provide estimation on the congestion of minimum routing trees for planar graphs and an $O(n \log n)$ algorithm that constructs such trees.

Paper organization. In Section 2 we prove a vertex separator theorem for graphs with non-negative vertex costs and weights and describe a linear algorithm for constructing the separators. In Section 3 we show that our theorem is tight within a constant factor. More graph separation results that follow from our vertex separator theorem by divide-and-conquer are given in Section 4. Applications of the separator theorems of the paper on graph embedding problems, for pebbling directed planar graphs, and for construction of minimum routing trees are discussed in Section 5.

2 A vertex separator theorem

In this section we prove our main result. As with other separator theorems for graphs embedded on surfaces, the proof relies on the fact that in any planar graph with a spanning tree of small radius there exists a small cycle that divides the graph into two roughly equal parts. In order to use this result for graphs with vertex costs we will define a special type of spanning tree and show how it can be used for slicing the graph into appropriately "thin" pieces. If not stated otherwise, we assume throughout this section that G denotes a connected planar graph with non-negative vertex costs and non-negative vertex weights adding to one.

Constructing a steep spanning tree

Definition 2.1 *Let T be a spanning tree of G with root rt and let for any vertex v of G $level(v)$ denote the sum of the vertex costs on the simple tree path from v to the root of T. T is called a* steep spanning tree *if for any vertex $w \neq rt$ the parent of w is a vertex adjacent to w with a minimum level.*

By definition, T is a steep spanning tree of G iff for any non-tree edge (v, w) of G such that $level(v) \leq level(w)$ and $w \neq rt$

$$level(v) \geq level(parent(w)). \tag{1}$$

The next lemma establishes a close relationship between the steep spanning trees and the breadth-first spanning (BFS) trees.

Lemma 2.1 *If all costs of G are unit, then a spanning tree T of G is a steep spanning tree if and only if T is a BFS tree of G.*

Proof: The proof of sufficiency follows directly from (1). To prove the necessity assume that G has unit costs and T is a steep spanning tree of G. By (1), for any non-tree edge (v, w) such that $level(v) \leq level(w)$ and $w \neq rt$

$$level(w) - level(v) \leq level(w) - level(parent(w)) = cost(w) = 1. \tag{2}$$

We will prove that T is a BFS tree by induction on the number of levels. Assume that for some integer level $l \geq 0$ and any $l_0 \leq l$ the set of vertices on level l_0 contains all vertices at distance l_0 to the root of T. We have to prove that any vertex on level $l + 1$ is at distance $l + 1$ from the root rt and that any vertex at distance $l + 1$ from rt is on level $l + 1$.

Let w be any vertex at level $l + 1$ and let x be the parent of w. Then, by the definition of levels, the level of x is l and thus the distance d between w and rt is at most $l + 1$. But d can not be less than $l + 1$ because by the induction hypothesis levels $0, 1, \cdots, l$ contain all vertices at distance at most l from rt.

Let now x be any vertex at distance $l + 1$ from rt. We have to prove that the level of x is $l + 1$. There exists a path between x and rt of length $l + 1$. Let (x, y) be the first edge of this path. Then the distance between y and rt is l and by the induction assumption the $level(y) = l$. If (x, y) is a tree edge, then the claim follows by the definition of levels.

Let (x, y) be a non-tree edge. Clearly $level(x) \geq l + 1$ since by the inductive assumption the vertices at levels less or equal to l are at distance at most l from rt. Then $level(x) \geq l+1 > l = level(y)$ and by (2) $level(x) - level(y) \leq 1$. Since all levels are integers, by the last inequality $level(x) - level(y) = 1$ and thus $level(x) = l + 1$. ∎

A steep spanning tree can be constructed in linear time by the following algorithm.

Algorithm STEEP–TREE
{Constructs a steep spanning tree T of G}

1. Replace any edge (v, w) of G by a pair of directed edges, one from v to w of *length* $cost(w)$ and one from w to v of length $cost(v)$.

2. Choose an arbitrary vertex rt to be the root of T. Run the linear time single source shortest path algorithm from [15] on the resulting planar graph, G', with source any vertex rt of G'.

3. Define T to be the resulting shortest path tree of G'.

Lemma 2.2 *Algorithm* STEEP–TREE *constructs a steep spanning tree of G in $O(|G|)$ time, where $|G| = |V(G)| + |E(G)|$.*

Proof: Let (v, w) be a non-tree edge of G with $level(v) \leq level(w)$, where the levels are computed with respect to T. By the definitions of a level and a shortest path tree, the shortest path between rt and w in G' has length $level(w) = level(parent(w)) + cost(w)$. Assume that (1) does not hold for (v, w). Then the path between rt and w consisting of the shortest path between rt and v and the edge (v, w) has length $level(v) + cost(w) < level(parent(w)) + cost(w) = level(w)$. Hence this path is shorter than the tree path between rt and w, contradicting the definition of T in Step 3 of the algorithm. ∎

It is an interesting open problem to construct a linear time algorithm for constructing steep spanning trees for arbitrary (not necessarily planar) graphs.

Slicing the graph

Given a spanning tree T of G, each non-tree edge induces a (unique) simple cycle. We call such cycle a *T-cycle*. Using Jordan's Curve Theorem, Lipton and Tarjan showed in [20] that if G is triangulated, then at least one T-cycle is a weighted $2/3$-separator of G.

For graphs with vertex costs we have the following result.

Lemma 2.3 *Let G be a triangulated planar graph with non-negative real vertex costs and a steep spanning tree T of maximum level l_{max}. A vertex $2/3$ separator of G that is a T-cycle with cost not exceeding $2l_{max} + cost(rt)$ can be found in $O(|G|)$ time, where rt is the root of T.*

Define the *width* of any subgraph of G to be the maximum difference of the levels of any two vertices of G. Our next goal is, by removing a set of vertices of small total cost, to cut from G components of small weights so that the remaining part has appropriately small width (so that Lemma 2.3 can be efficiently applied on that component).

For any tree edge $e = (v, w)$ of G assign a non-negative edge cost $cost(e) = |level(v) - level(w)|$ and assign cost 0 to each non-tree edge. Denote by l_{max} the maximum level of G. For any real x define a set

$$E_x = \{(v, w) \in E(G) \mid level(v) \leq x < level(w)\}.$$

Consider the function f defined for $x \geq 0$ by

$$f(x) = \sum_{e \in E_x} cost(e) \tag{3}$$

($f(x) = 0$ if $x > l_{max}$). The next lemma shows that certain integrals on f can be computed efficiently. It will be used in Algorithm Level–l' below.

Lemma 2.4 *Let $L_1 \leq L_2$, where $L_1 \in \Re$ and $L_2 \in \Re \cup \{\infty\}$. Let S_1 be the set of all edges with at least one endpoint in $[L_1, L_2]$ and let S_2 be the set of all edges (v, w) such that $level(v) \leq L_1 \leq L_2 \leq level(w)$. Given S_1 and $\Delta = \sum_{e \in S_2} cost(e)$, the integral $\int_{L_1}^{L_2} f(x)\,dx$ can be computed in $O(|S_1|)$ time.*

Proof: Define a function $\kappa_{v,w}(x)$ equal to 1, if $level(v) \leq x \leq level(w)$ or if $level(w) \leq x \leq level(v)$, or equal to 0, otherwise. Then

$$f(x) = \sum_{(v,w) \in E_x} cost(v, w) = \sum_{(v,w) \in E(G)} cost(v, w)\kappa_{v,w}(x)$$

$$= \sum_{(v,w) \in E(T)} cost(v, w)\kappa_{v,w}(x).$$

Thus

$$\int_{L_1}^{L_2} f(x)\,dx = \int_{L_1}^{L_2} \sum_{(v,w) \in E(T)} cost(v, w)\kappa_{v,w}(x)\,dx$$

$$= \sum_{(v,w)\in E(T)} cost(v,w) \int_{L_1}^{L_2} \kappa_{v,w}(x)\,dx.$$

Let $I_{v,w}$ denote $\int_{L_1}^{L_2} \kappa_{v,w}(x)\,dx$. If $(v,w) \in S_2$, then $I_{v,w} = L_2 - L_1$. If $(v,w) \in E(G) \setminus (S_1 \cup S_2)$, then $I_{v,w} = 0$. Hence

$$\int_{L_1}^{L_2} f(x)\,dx = \sum_{(v,w)\in E(T)\cap S_1} cost(v,w) \int_{L_1}^{L_2} \kappa_{v,w}(x)\,dx + \Delta(L_2 - L_1). \quad (4)$$

By the definition of $\kappa_{v,w}$, assuming w.l.o.g. that $level(v) \leq level(w)$, $I_{v,w} := \int_{L_1}^{L_2} \kappa_{v,w}(x)\,dx$ is equal to the length of the intersection of the intervals $[level(v), level(w)]$ and $[L_1, L_2]$, i.e., $I_{v,w}$ can be computed using $O(1)$ comparisons and at most one subtraction. The lemma follows. ∎

Denote $N(G) = \sum_{v\in V(G)}(cost(v))^2$. From the proof of the previous lemma the next statement follows.

Lemma 2.5 $\int_0^{\infty} f(x)\,dx = N(G) - (cost(rt))^2$.

Proof: By the definition of $\kappa_{v,w}$

$$\int_0^{\infty} \kappa_{v,w}(x)\,dx = |level(v) - level(w)|.$$

Then by (4)

$$\int_0^{\infty} f(x)\,dx = \sum_{(v,w)\in E(T)} cost(v,w)|level(v) - level(w)|$$

$$= \sum_{(v,w)\in E(T)} (level(v) - level(w))^2$$

$$= \sum_{w=parent(v)} (cost(w))^2 = \sum_{w\in V(G)\setminus\{rt\}} (cost(w))^2. \ \blacksquare$$

Our algorithm for finding a pair of levels whose removal cuts the graph into components with a small width or a small weight is based on the following lemma (see Figure 1).

Lemma 2.6 *Let $h(l)$ be a continuous non-negative function defined on the interval $[0, L]$ such that $h(0) = 0$. Then*

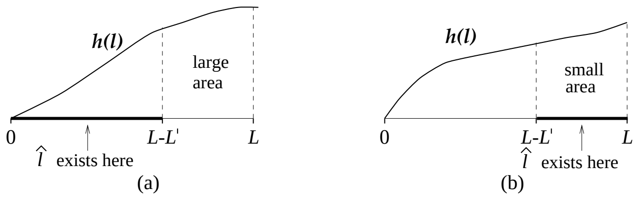

Figure 1: Illustration of Lemma 2.6.

(i) There exists a number $\hat{l}$ in $[0, L]$ such that

$$h(\hat{l}) + 2(L - \hat{l}) \le 2\sqrt{\int_0^L h(l)\,dl}; \tag{5}$$

(ii) Let $L' \in [0, L]$. If $L'^2 \le \int_{L-L'}^L h(l)\,dl$, then such a number $\hat{l}$ exists in the interval $[0, L - L']$, otherwise such a number $\hat{l}$ exists in the interval $(L - L', L]$.

Proof: We will prove only (ii) since it implies (i). Denote $S = \int_0^L h(l)\,dl$. Consider the following cases.

(a) $L'^2 \le \int_{L-L'}^L h(l)\,dl$ (Figure 1 (a)). Assume that no $\hat{l} \in [0, L - L']$ satisfies (5). Then

$$h(l) > 2\sqrt{S} - 2(L - l) \quad \text{for } 0 \le l \le L - L'. \tag{6}$$

Since $h(0) = 0$ then (6) implies $L > \sqrt{S}$. Moreover, from $L'^2 \le \int_{L-L'}^L h(l)\,dl$ it follows $L' \le \sqrt{S}$ and thus $L > L'$. Hence

$$S = \int_0^L h(l)\,dl \ge L'^2 + \int_0^{L-L'} h(l)\,dl$$

$$> L'^2 + \int_0^{L-L'} (2\sqrt{S} - 2(L - l))\,dl \ge L'^2 + \int_{L-\sqrt{S}}^{L-L'} (2\sqrt{S} - 2(L - l))\,dl$$

$$\ge L'^2 + (\sqrt{S} - L')2\sqrt{S} - \int_{L-\sqrt{S}}^{L-L'} 2(L - l)\,dl$$

$$\ge L'^2 - \int_{L-\sqrt{S}}^{L-L'} 2(L - l)\,dl = L'^2 + \int_{L'}^{\sqrt{S}} 2l\,dl = S,$$

which is a contradiction. Thus (5) holds for some $\hat{l} \in [0, L - L']$.

(b) $L'^2 > \int_{L-L'}^{L} h(l)\,dl$ (Figure 1 (b)). Assume that no $\hat{l} \in (L-L', L]$ satisfies (5). Then

$$h(l) > 2\sqrt{S} - 2(L-l) \quad \text{for } L-L' < l \le L.$$

Hence

$$L'^2 > \int_{L-L'}^{L} h(l)\,dl \ge \int_{L-L'}^{L} [2\sqrt{S} - 2(L-l)]\,dl$$

$$\ge 2\sqrt{S}L' - \int_{L-L'}^{L} 2(L-l)\,dl \ge \int_{0}^{L'} 2l\,dl = L'^2,$$

a contradiction. Thus (5) holds for some $\hat{l} \in (L-L', L]$. ∎

Lemma 2.7 *If cost$(rt) = 0$ then there exists a pair of levels l' and l'' such that*

(i) Both the weight of all vertices of G on levels below l' and the weight of all vertices of G on levels above l'' do not exceed 1/2;

(ii) $f(l') + f(l'') + 2(l'' - l') \le \sqrt{8N(G)}$.

Such pair of levels can be found in $O(|G|)$ time.

Proof: First we will prove that such levels exist and then we will discuss how they can be found efficiently.

Existence

Let l^* be the lowest level such that the weight of the vertices of G on levels l^* or below is at least 1/2. Denote $N(G)' = \int_0^{l^*} f(l)\,dl$ and $N(G)'' = \int_{l^*}^{\infty} f(l)\,dl = N(G) - N(G)'$ (by Lemma 2.5). Denote by l' and l'' two non-negative numbers such that $l' \le l^* \le l''$ and

$$f(l') + 2(l^* - l') \le 2\sqrt{N(G)'} \tag{7}$$

$$f(l'') + 2(l'' - l^*) \le 2\sqrt{N(G)''}. \tag{8}$$

The existence of such number l' follows directly from Lemma 2.6. To show that a suitable number l'' exists, let $\hat{l}$ be a number satisfying (5) with respect to $L = R - l^*$ and $h(x) = f(R-x)$, where R is any number greater than the radius of T. By using the substitution $l = R - u$

$$\int_0^{R-l^*} h(l)\,dl = \int_0^{R-l^*} f(R-l)\,dl = \int_{l^*}^{R} f(u)\,du = \sqrt{N(G)''}.$$

Denote $l'' = R - \hat{l}$. Then by the above equalities and (5)

$$2\sqrt{N(G)''} \geq h(\hat{l}) + 2(R - l^* - \hat{l}) = f(R - \hat{l}) + 2(R - l^* - \hat{l}) = f(l'') + 2(l'' - l^*).$$

Thus numbers l' and l'' satisfying inequalities (7) and (8) exist. Adding (7) and (8) gives $f(l') + f(l'') + 2(l'' - l') \leq 2(\sqrt{N(G)'} + \sqrt{N(G)''}) \leq 2(\sqrt{N(G)/2} + \sqrt{N(G)/2}) \leq \sqrt{8N(G)}$. Thus l' and l'' satisfy condition (ii). Condition (i) follows from the choice of l^*.

Algorithms

It is easy to find levels l' and l'' satisfying the lemma in $O(|G| \log |G|)$ time by sorting the sequence of levels of the vertices of G and then following the proof from the first part of the lemma. We will show here that one can avoid sorting and find the levels in linear time.

First we describe an algorithm for finding the lowest level l^* such that the weight of the vertices of G on levels l^* or below is at least $1/2$. Note that the main difficulty for finding l^* is that we can not compute in linear time, for all levels l, the sum of the weights of the vertices on levels not exceeding l. The algorithm given below finds a level l^* by carrying out a binary search and computing at each step the median of a set of levels.

We will maintain in our algorithm a subset S of vertices of G that contains all vertices of G whose levels belong to a certain interval. At each iteration the size of S will be reduced at least by a factor of two.

Algorithm LEVEL–l^*

1. Set initially $S := V(G)$ and $Wt^< := 0$, where $Wt^<$ denotes the sum of the weights of the vertices of G on levels lower than the minimum level of a vertex in S.

2. Find in $O(|S|)$ time a level l_{med} that is the median of of the multiset of the levels of the vertices of S.

3. Compute the sum $wt^<$ of the weights of all vertices on levels less than l_{med} and the sum $wt^=$ of the weights of all vertices on levels equal to l_{med}.

4. Check which of the following cases applies.

   (a) If $Wt^< + wt^< + wt^= < 1/2$, then let S be the set of vertices on levels greater than l_{med} and let $Wt^< := Wt^< + wt^< + wt^=$. Repeat Steps 2–4.

(b) If $Wt^{<} + wt^{<} < 1/2$ and $Wt^{<} + wt^{<} + wt^{=} \geq 1/2$, then $l^* := l_{med}$.

(c) If $Wt^{<} + wt^{<} \geq 1/2$, then let S be the set of vertices on levels less than l_{med} (and $Wt^{<}$ be unchanged). Repeat Steps 2–4.

Since all steps require $O(|S|)$ time and after each iteration the size of S is reduced by a factor of at least 2, the time $T(n)$ of the algorithm satisfies the recurrence

$$T(n) \leq T(n/2) + O(n), \text{ if } n > 1,$$

whose solution is $T(n) = O(n)$.

Next we discuss how to find levels l' and l'' satisfying (7), (8), and the inequalities $0 \leq l' \leq l^* \leq l''$. Note that the main difficulty here is that we can not compute in linear time the values of the function f for the levels of all vertices of G, if we are not given the sorted sequence of levels.

We implement a binary search using again the linear algorithm for finding the median and Lemma 2.6 (ii) for making the choice in which half to continue the search. We describe only the algorithm for finding the level l'; the algorithm for finding the level l'' is similar.

In the algorithm below, S denotes the set of edges with at least one endpoint in $[L_1, L_2]$ and Δ denotes the sum of the costs of all edges (v, w) such that $level(v) \leq L_1 \leq L_2 \leq level(w)$. I_{old} denote the value of $\int_{L_2}^{l^*} f(l)\,dl$. See Figure 2 for illustration.

Algorithm Level–l'

1. Set initially $L_1 := 0$, $L_2 := l^*$, and $I_{old} := 0$. Compute $N(G) := \int_0^\infty f(l)\,dl$, construct S, and compute Δ.

2. Find in $O(|S|)$ time a level l_{med} that is the median of the multiset of the levels of the endpoints of the edges of S. Find also the closest to l_{med} levels l'_{med} and l''_{med} such that $l'_{med} < l_{med}$ and $l''_{med} > l_{med}$.

3. Compute $I_{new} := \int_{l_{med}}^{L_2} f(l)\,dl$ using the algorithm from Lemma 2.4. Check which of the following cases applies (see Figure 2).

   (a) If l_{med} satisfies (7), i.e., if $f(l_{med}) + 2(l^* - l_{med}) \leq 2(N(G) - I_{old} - I_{new})$, then let $l' := l_{med}$ and halt.

   (b) If $(l^* - l_{med})^2 \leq I_{new} + I_{old}$, then let $I_{old} := I_{new} + I_{old}$ and $L_2 := l'_{med}$.

   (c) If $(l^* - l_{med})^2 > I_{new} + I_{old}$, then let $L_1 := l''_{med}$.

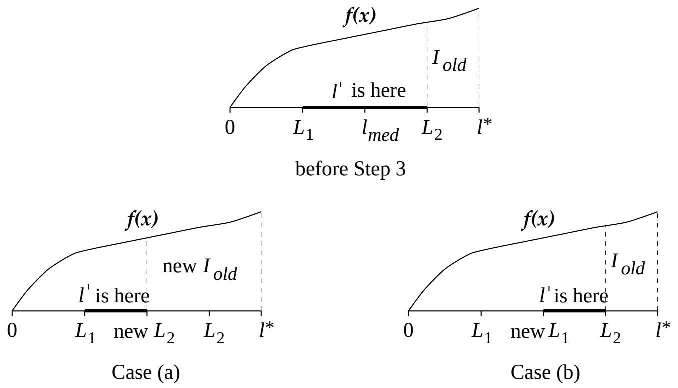

Figure 2: Cases in Step 3 of Algorithm LEVEL–l'.

4. Update S and Δ as follows. For each edge $(v, w) \in S$ check which of the following cases applies. Assume w.l.o.g. that $level(v) \leq level(w)$.

   (a) If $[level(v), level(w)] \cap [L_1, L_2] = \emptyset$ then remove (v, w) from S.

   (b) If $[level(v), level(w)] \supseteq [L_1, L_2]$ then remove (v, w) from S and set $\Delta := \Delta + cost(v, w)$.

Step 1 requires $O(n)$ time. Step 3(a) requires $O(|S|)$ time since $f(l_{med})$ can be computed using the equality $f(l_{med}) = \sum_{e \in (E_x \cap S)} cost(e) + \Delta$ (see (3)). The other steps of Algorithm LEVEL–l' clearly also require $O(|S|)$ time. Moreover, after each iteration the size of S is reduced by a factor of at least 2. Thus, the time $T(n)$ for all executions of Steps 2–4 satisfies the recurrence

$$T(n) \leq T(n/2) + O(n), \text{ if } n > 1,$$

whose solution is $T(n) = O(n)$. ∎

Next we prove the main theorem of this section.

Theorem 1 *Let G be a planar graph with non-negative vertex weights that sum up to one and non-negative vertex costs whose squares sum up to $N(G)$. There exists a $2/3$ vertex separator of G with total vertex cost no more than $\sqrt{8N(G)}$. Such a separator can be found in $O(|G|)$ time.*

Proof: First we will partition the graph into components by removing a set of edges of small cost so that Lemma 2.3 can be applied to a component of small width. Then we will replace that edge set by a vertex set of low cost.

Embed G in the plane and add new edges (if necessary) in order to triangulate the embedding. We will need to identify a vertex of cost 0 to be the root of a spanning tree of G. If such a vertex rt does not exist in G, we define a new vertex rt of cost 0 in any face (triangle) f of the embedding of G and join it by 3 new edges to the vertices on f. We will use G to denote the resulting graph.

Given a subgraph H of G, let the weight (resp. the cost) of H be the sum of the weights (resp. costs) of the vertices in H. Similarly, we define weight and cost of vertex and edge sets.

Let l' and l'' be a pair of levels satisfying Lemma 2.7. By definition, the set of edges $E_{sl} = E_{l'} \cup E_{l''}$ has cost $f(l') + f(l'')$ (see (3)). Let K be a component of a maximum weight into which E_{sl} divides G.

Assume that the weight of K is greater than $2/3$. Then, by Lemma 2.7(i), the level of any vertex in K will be in the interval (l', l''). Thus, the difference between the levels of any two vertices of K is less than $l'' - l'$.

Next we will replace E_{sl} by a vertex set. Define

$$M_{sl} = \{v \mid (v, parent(v)) \in E_{sl}\}.$$

By our definition of edge costs, M_{sl} and E_{sl} have equal costs. By the definition of M_{sl}, any *tree* edge from E_{sl} is incident to a vertex from M_{sl}.

Thus, in order to show that M_{sl} can be used instead of E_{sl} for partitioning G, we have also to show that any *non-tree* edge from E_{sl} is incident to a vertex from M_{sl}.

Let (v, w) be a non-tree edge of E_{sl}. Assume w.l.o.g. that $level(v) \leq level(w)$. Then $level(v) \leq l \leq level(w)$ for some $l \in \{l', l''\}$. By (1), $level(v) \geq level(parent(w))$. Then $l \geq level(v) \geq level(parent(w))$ and $level(w) \geq l \geq level(parent(w))$. Hence the tree edge $(w, parent(w)) \in E_{sl}$ and $w \in M_{sl}$.

To summarize, we found a vertex set M_{sl} such that for a pair of levels l' and l'' satisfying Lemma 2.7

(i) the cost of M_{sl} is $f(l') + f(l'')$;

(ii) if the heaviest component K induced by $V(G) \setminus M_{sl}$ has weight greater than $2/3$, then the width of K is at most $l'' - l'$.

Now apply Lemma 2.3 to G. Let c be a simple cycle with exactly one non-tree edge and that is a weighted $2/3$-separator of G. Denote by M_c the set of vertices on c that belong to K. Then the cost of M_c is at most $2(l'' - l')$ (we used here the fact that $cost(rt) = 0$).

If the weight of K is less than $2/3$ we define $M_c = \emptyset$.

The set $M = M_{sl} \cup M_c$ is clearly a $2/3$-separator of G. The cost of M is no more than $f(l') + f(l'') + 2(l'' - l')$, which by Lemma 2.7 does not exceed $\sqrt{8N(G)}$.

By Lemmas 2.2, 2.7, and 2.3, constructing a steep spanning tree, finding levels l' and l'', or constructing a cycle c requires $O(|G|)$ time. All other steps can be implemented in linear time in a straightforward manner. ∎

We note that a more involved proof would yield even a better constant in Theorem 1. For instance, using the techniques of [7] we can replace the constant $\sqrt{8}$ by 2 and also guarantee that the separator C is a simple cycle, provided the original graph is triangulated.

3 Tightness of the result

In this section we will show that Theorem 1 is optimal within a constant factor. First we note that, given a function $f(x) = o(\sqrt{x})$, there does not exist a separator as in Theorem 1 where the cost of the separator is bounded by $f(\sum_{v \in V(G)} (cost(v))^2)$. If such a version of Theorem 1 existed, then one could assign to all vertices of an arbitrary planar n-vertex graph G equal costs and weights and find a separator of G of cardinality $o(\sqrt{n})$. This will contradict the fact that no separator theorem exists for the class of planar graphs with cardinality of the separator $o(\sqrt{n})$ [20].

The above fact shows that if the sum of the squares of the vertex costs is chosen to be used as a measure of the cost (i.e., as a norm) of the graph, then one can not improve the bound of Theorem 1 except for the constant factor. However, the above argument does not imply that by using some other measure one could not obtain an asymptotically stronger version of the theorem. Here we will prove that Theorem 1 is optimal also in the stronger sense.

Theorem 2 *There exists a constant $c > 0$ such that for any integer $n > 0$ and any n non-negative reals $cost_1, \cdots, cost_n$ there exists a planar n-vertex weighted graph G with costs the given numbers such that any $2/3$ vertex separator of G has total vertex cost at least $c\sqrt{\sum_{i=1}^{n} cost_i^2}$.*

Proof: Let n be any positive integer and let $cost_1, \cdots, cost_n$ be non-negative reals. We will define a graph G such that any weighted vertex $2/3$ separator of G has cost $\Omega(\sqrt{Q})$, where $Q = \sum_{i=1}^{n} cost_i^2$.

We define the *distance* between any two points of a polygon P the length of the shortest Euclidean path joining the points that stays inside P. The

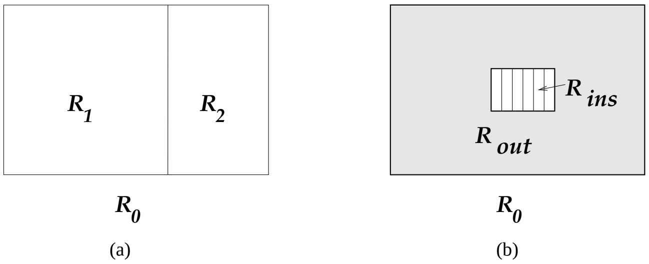

Figure 3: Subdivisions of the rectangle in Cases (a) and (b).

aspect ratio of P is the ratio of the maximum distance between any pair of points of P and the square root of the area of P.

The idea of our proof is to construct a square in the plane whose area is Q and that is subdivided into polygons with areas the numbers $c_i^2, i = 1, \cdots, n$. We will aim that the polygons are of bounded aspect ratio. Then we will define G to be the dual graph of the resulting embedding (excluding the outer face).

Let R_0 be any square with area Q and thus with aspect ratio $\sqrt{2}$ (actually we will use only the fact that the aspect ratio of R_0 is no more than 2). Consider the following two cases.

(a) $costs_i^2 < Q/3$ for any $1 \leq i \leq n$. Find the smallest j such that $Q_1 = \sum_{i=1}^{j} cost_i^2 > Q/3$. Then $Q_1 \leq 2Q/3$ and $Q_2 = Q - Q_1 < 2Q/3$. Divide R_0 into two rectangles R_1 and R_2 of areas Q_1 and Q_2 as shown on Figure 3 (a). Then the aspect ratio of each of R_1 and R_2 is no more than 2 (assuming that the aspect ratio of R_0 is no more than 2). Recursively, find a subdivision of R_1 with respect to the numbers $cost_1, \cdots, cost_j$ and a subdivision of R_2 with respect to the numbers $cost_{j+1}, \cdots, cost_n$.

(b) $costs_j^2 \geq Q/3$ for some $1 \leq j \leq n$. Construct a rectangle R_{ins} with area $Q - costs_j^2$ and the same aspect ratio as R_0 and draw it inside R_0 as illustrated on Figure 3 (b). Denote by R_{out} the part of R_0 that is outside R_{ins}. Assign a cost $cost_j$ to R_{out}. Divide R_{ins} into $n - 1$ equal rectangles, which we call *internal* (see Figure 3 (b)), and assign the remaining costs arbitrarily to these rectangles. While the aspect ratio of the internal rectangles is unbounded, it will be sufficient for our purposes that the aspect ratio of R_{out} is bounded.

The resulting subdivision has n elementary polygons (i.e., which are not further subdivided) to which the numbers $cost_1, \cdots, cost_n$ have been assigned and each elementary polygon P that is not an internal rectangle has a bounded aspect ratio and cost equal to the square root of the area of P. For each elementary polygon R to which a number (say) $cost_j$ is assigned construct a vertex j in a graph G and connect j by an edge to any vertex of G whose corresponding elementary polygon shares a common segment with R. Assign a weight $cost_j^2$ to vertex j. We will show that the resulting planar graph satisfies the requirements of the theorem.

Assume that G has a $2/3$ vertex separator of cost $o(\sqrt{Q})$. Then, by the construction of G, there exists a set M of elementary polygons with total cost $o(\sqrt{Q})$ that divide a square with area Q (namely R_0) into two disjoint parts each with area at most $2Q/3$. Remove from M all internal rectangles and denote by M' the resulting set of polygons. Note that if R' is an internal rectangle of cost c' corresponding to a vertex v of G, then the removal of R' from M can increase the weight of the biggest part of the partition by no more than the weight of v, which, by definition, is equal to $(c')^2$ (see Figure 3 (b)). Thus removal of all internal rectangles from M can increase the weight of the biggest part by no more than the weight of the separator of G, which is $o(Q)$.

By construction, M' is a set of disjoint polygons with bounded aspect ratios and with total cost $o(\sqrt{Q})$. Since the polygons in M' are with bounded aspect ratios, the cost assigned to each polygon in M' is proportional to the maximum length of a shortest line joining any two points of the polygon. As discussed above, removal of the polygons of M' divides the square R_0 of area Q into two parts each with area $\Theta(Q)$. Then there exists a line of total length $o(\sqrt{Q})$ that divides R_0 into two polygons of areas $\Theta(Q)$ each, which is impossible. The contradiction shows that G has no vertex 2/3-separator of cost $o(\sqrt{Q})$. ■

One could conjecture that a lower bound as in Theorem 2 can be proved for an arbitrary set of n vertex weights that are independent of the vertex costs. Such conjecture, however, is false. For example, for graphs with weights $1/n$ on all vertices and costs arbitrary non-negative numbers $cost_1, \cdots, cost_n$, no $\Omega(\sqrt{\sum_{i=1}^{n} cost_i^2})$ lower bound can be proved since in the case $cost_1 = cost_2 = \cdots = cost_k = 0$, where $k = \lceil n/3 \rceil$, and arbitrary other costs, any graph with such costs and weights has a trivial separator of k vertices and cost 0.

4 Related separator theorems

4.1 Separation into equal parts

We will show that Theorem 1 implies a 1/2-separator theorem with a bound on the cost of the separator increased only by a constant factor.

Theorem 3 *Let G be a planar graph with non-negative vertex weights that sum up to one and non-negative vertex costs whose squares sum up to $N(G)$. There exists a $1/2$ vertex separator of G of cost $O(\sqrt{N(G)})$, which can be found in $O(|G|)$ time.*

Proof: Let $cost(v)$ denote the cost on vertex v. Our algorithm maintains a partition A, B, C, D of the vertices of G such that no edge joins a vertex from A with a vertex from B. D contains initially all vertices of G and at each step a portion of the vertices of D is added to A, B, or C. During each iteration, we apply twice our 2/3 separator theorem (Theorem 1). The first time Theorem 1 is applied we assign equal weights to the vertices. This guarantees that the cardinality of D is reduced by a factor of 2/3, implying a linear running time. The second time Theorem 1 is applied we assign weights proportional to the squares of the vertex costs. This guarantees a small cost of the final separator. The algorithm ends when D is empty.

Here is the algorithm in a greater detail.

Algorithm 1/2-SEPARATOR

Input: A planar weighted n-vertex graph G
Output: A set C of vertices that is an 1/2-separator of G

1. Define an initial partition of the vertices of G in sets A, B, C, D as follows: $A, B, C := \emptyset,\ D := V(G)$.

2. Apply the 2/3-separator theorem using unit vertex weights as described below. (The goal of this step is to reduce the number of vertices of D by a factor 3/2 so that the time complexity of the algorithm be linear.)

   (a) Find by Theorem 1 a separator X of the subgraph of G induced by D by using the original costs and vertex weights $1/|D|$ and let Y, Z be a partition of the vertices of $D \setminus X$ such that $weight(Y), weight(Z) \leq (2/3)$ (which is equivalent to $|Y|, |Z| \leq (2/3)|D|$).

(b) $C := C \cup X$.

(c) If (say) the original weight of Y is not greater than original the weight of Z then add Y to the lighter set between A and B and set $D := Z$. (It will be shown that the current weights of A and B never exceed $1/2$.)

3. Apply again Step 2 to the resulting partition A, B, C, D, but replace the vertex weight $1/|D|$ on each vertex v with a weight $cost^2(v)/cost(Q)$, where Q is the sum of the squares of all costs on vertices from D. (The goal of this step is to reduce the sum of the squares of vertex costs of D by a factor $3/2$ so that the cost of the final separator be small.)

4. Repeat Steps 2 and 3 until $D = \emptyset$.

Correctness

Note that before any execution of Step 2 or 3 the sets A, B, C, D form a partition of $V(G)$ and the sets X, Y, Z form a partition of D. We will prove that the weight of any of A and B never exceed $1/2$. Assume that, at the beginning of Step 2, each of A and B has weight not more than $1/2$ (with respect to the original weights in G) and in Step 2(c) Y is added to A. Then Y is lighter than Z and A is lighter than B. Moreover A, B, C, X, Y, Z is a partition of $V(G)$. Thus the weight of $A \cup Y$ is not greater than half of the weight of $V(G)$ (which is one) and the weight of B is not changed. We showed that, if at the beginning of Step 2 A and B have weights less than $1/2$, after Step 2 their weights are still less than $1/2$. The analysis of Step 3 is similar. By induction, the weight of each of A and B never exceeds $1/2$.

Time bound

After each execution of Step 2 the cardinality of D is reduced by a factor of $3/2$. Thus, the maximum time $T(n)$ needed by the algorithm to find a separator of an n-vertex graph satisfies the recurrence

$$T(n) \leq T(\lfloor 2n/3 \rfloor) + O(n) , \quad \text{for } n > 2,$$

which has a solution $T(n) = O(n)$.

Bound on the cost of C

As a result of Step 3, after each iteration the sum of the squares of the costs of the vertices of D is reduced by a factor of $3/2$. Thus the cost $cost(C)$ of C is bounded by

$$cost(C) \leq \sum_{i=0}^{\infty} \left(\frac{2}{3}\right)^{i/2} O(\sqrt{N(G)}) = O(\sqrt{N(G)}). \quad \blacksquare$$

4.2 Edge separators

In [5] it has been proved that any n-vertex planar graph of maximum vertex degree d has a $2/3$ edge separator of cardinality $O(\sqrt{dn})$. We will show that, using Theorem 1, this result can easily be generalized for the case of graphs with vertex costs.

Theorem 4 *Let G be a planar graph of maximum vertex degree d, with non-negative vertex weights that do not exceed $2/3$ and sum up to one, and non-negative edge costs whose squares sum up to $N(G)$. There exists a $2/3$ edge separator of G of total cost not exceeding $4\sqrt{dN(G)}$, which can be found in $O(|G|)$ time.*

Proof: Let $cost(e)$ denote the cost on edge e. For any vertex v of G define a cost $cost(v) = \sum_{e\in E(v)} cost(e)$, where $E(v)$ is the set of edges incident to v. By Theorem 1, there exists a vertex set of cost no more than $\sqrt{8}\sqrt{\sum_{v\in V(G)}(cost(v))^2}$ whose removal leaves no component of weight greater than $2/3$. Then the set of all edges incident to a vertex of that vertex separator is an edge separator C, which can be found in linear time using the algorithm from Theorem 1.

Let us estimate the cost of C. For each vertex v

$$(cost(v))^2 = (\sum_{e\in E(v)} cost(e))^2.$$

By the inequality

$$(\sum_{i=1}^{n} x_i)^2 \leq n \sum_{i=1}^{n} x_i^2, \tag{9}$$

valid for arbitrary positive reals $x_1, \cdots, x_n$,

$$(cost(v))^2 \leq |E(v)| \cdot \sum_{e\in E(v)} (cost(e))^2 \ \leq d \cdot \sum_{e\in E(v)} (cost(e))^2.$$

Since each edge contributes to the cost of exactly two vertices of G, the claimed bound follows. ∎

4.3 Separation of graphs of bounded genus

For graphs of bounded genus, one can combine the technique of Section 2 with results from [6, 10] to prove separator theorems of the type of Theorems 1 and 4. We will use the following lemma, which is an adaptation of a result from [10].

Lemma 4.1 *Let G be a triangulated graph embedded on a surface of genus g with non-negative vertex weights that sum up to one and non-negative vertex costs. Let T be a steep spanning tree of G of maximum level l_{max} and root rt. A weighted vertex $2/3$-separator of G that is a union of $2g+1$ T-cycles with total cost not exceeding $(2g+1)l_{max} + cost(rt)$ can be found in $O(|G|)$ time, given the embedding of G.*

Theorem 5 *Let G be an n-vertex graph embedded on a surface of genus g with non-negative vertex weights that sum up to one and non-negative vertex costs whose squares sum up to $N(G)$. There exists a $2/3$ vertex separator of G with total vertex cost no more than $\sqrt{(16g+8)N(G)}$. Such a separator can be found in $O(|G|)$ time, given the embedding of G.*

Proof: Similar to the proof of Theorem 1, but using Lemma 4.1 now instead of Lemma 2.3. ∎

Theorem 6 *Let G be an n-vertex graph of maximum vertex degree d embedded on a surface of genus g, with non-negative vertex weights that do not exceed $2/3$ and sum up to one, and non-negative edge costs whose squares sum up to $N(G)$. There exists a $2/3$ edge separator of G of total cost not exceeding $\sqrt{(32g+16)dN(G)}$, which can be found in $O(|G|)$ time, given the embedding of G.*

Proof: Similar to the proof of Theorem 4. ∎

5 Applications

5.1 Graph embeddings

Many problems in VLSI design and in network simulation can be formulated as graph embedding problems. We show here that, in the case of planar graphs or graphs of bounded genus, one can use our graph separation results to find embeddings with good properties.

Formally, an embedding of a graph G_1 into a graph G_2 is an injective mapping $\mu : V(G_1) \to V(G_2)$. One measure for characterizing an embedding μ is the *average dilation* of μ defined as

$$AD(\mu) = \Big(\sum_{(u,v)\in E(G_1)} d_{G_2}(\mu(u), \mu(v)) \Big) \;/\; |E(G_1)|,$$

where $d_{G_2}(x, y)$ denotes the distance between vertices x and y in G_2. In the applications, G_1 can model communicating processes of a computer program, a circuit, or a communication network, where vertices denote communicating objects and edges denote existence of communication links. G_2 represents a target structure (e.g., a mesh of processors, or a VLSI chip) and usually has a regular structure. Then $AD(\mu)$ will correspond to the average increase of communication time if G_1 is implemented or simulated by G_2.

In many applications the amounts of information that need to be exchanged between the individual communicating objects are different. For such problems non-negative costs $cost(u, v)$ are assigned to the edges of G_1 and the goal is to minimize the total increase of traffic if G_1 is implemented on G_2. In such a case we use as a measure the *average weighted dilation (AWD)* of μ defined as

$$AWD(\mu) = \sum_{(u,v)\in E(G_1)} d_{G_2}(\mu(u), \mu(v)) \cdot cost(u, v) \;/\; |E(G_1)|.$$

Another parameter that is used to evaluate an embedding μ is the *expansion* of μ equal to $|V(G_2)|/|V(G_1)|$. Good embeddings are required to have $O(1)$ expansion. We will show that weighted separators can be used to construct embeddings of planar graphs with small AWD and expansion one.

Theorem 7 *Every n-vertex planar graph of maximum degree d and with non-negative edge costs whose squares sum up to n can be embedded into a binary tree with average weighted dilation $O(\log d)$ and expansion one.*

Proof: Let G be an n-vertex planar graph of maximum degree d with non-negative edge costs whose squares sum up to n. Assume that G is connected. Denote for any subgraph H of G $N(H) = \sum_{(u,v)\in E(H)}(cost(u, v))^2$, where $cost(u, v)$ denotes the cost on edge (u, v). Use the following recursive procedure to embed G into a binary tree T.

The embedding algorithm

If G has a single vertex, then define T to be a tree of a single vertex. Otherwise, assign a weight $1/n$ to each vertex of G. Construct in $O(n)$ time a partition A, B of $V(G)$ and an edge separator C of total cost $O(\sqrt{dN(G)}) = O(\sqrt{dn})$ as in Theorem 4. From the proof of Theorem 4, C consists of *all* edges incident to the vertices of a certain vertex separator M. Choose any $x \in M$. Embed recursively the subgraph induced by $A \setminus \{x\}$ into a binary tree T_1 and the subgraph induced by $B \setminus \{x\}$ into another binary tree T_2.

Make the roots of T_1 and T_2 children of x and denote by T the resulting tree. Label x by G (we will need the labels for the analysis of the algorithm).

Analysis

The height of T is $O(\log n)$ since, by Theorem 8, at each level of recursion the number of the vertices of the graph is reduced by a factor of at least $3/2$.

Denote

$$s(G) = \sum_{(u,v)\in E(G)} d_T(\mu(u), \mu(v)) \cdot cost(u,v), \tag{10}$$

where μ is the above defined embedding of G into T. Since the endpoints of each edge of C are embedded into vertices at distance $O(\log n)$ (at most twice the height of T), then

$$\begin{aligned} s(G) &= O(1), \quad \text{if } n = 1 && (11)\\ s(G) &\leq s(G_1) + s(G_2) + O(\sqrt{dN(G)}\log n), \quad \text{if } n > 1. && (12) \end{aligned}$$

Before solving the recurrence for $s(G)$, we note that (12) does not give a good bound for $s(G)$ in the case where G has less than d vertices, since the degree of a graph with less than d vertices will clearly be less than d. By (10), $s(G) \leq \log n \sum_{(u,v)\in E(G)} cost(u,v)$ for an arbitrary n. By inequality (9), $\sum_{(u,v)\in E(G)} cost(u,v) \leq \sqrt{|E(G)|}\sqrt{N(G)}$. Thus $s(G) = O(\sqrt{nN(G)}\log n)$ and we can consider the following recurrence instead of (11)–(12):

$$\begin{aligned} s(G) &= O(\sqrt{nN(G)}\log n), \quad \text{if } d \leq n \leq 2d, && (13)\\ s(G) &\leq s(G_1) + s(G_2) + O(\sqrt{dN(G)}\log n), \quad \text{if } n > 2d, && (14) \end{aligned}$$

where $|V(G_1)|, |V(G_2)| \leq (2/3)n$ and $|V(G_1)| + |V(G_2)| \leq n$.

Solving the recurrence

Our next goal is to show that the solution to the recurrence (13)–(14) is $s(G) = O(\sqrt{nN(G)}\log d)$. We consider the binary tree T constructed by the embedding algorithm. There is a one-to-one correspondence between the labels of the vertices of T and the subgraphs of G produced during all recursive calls. For instance, the root of T has a label G and its two children have labels G_1 and G_2. Next, we assign weights to the vertices of T that will be used during the estimation of $s(G)$. Define a weight $wt(t) = O(\sqrt{dN(G)}\log n)$ on the root t of T that corresponds to the rightmost term of (14). Define in the same way weights on the other vertices of T whose labels are graphs with more than $2d$ vertices. For any vertex v of T whose corresponding graph H has k vertices, where $d \leq k \leq 2d$, and no ancestor of v in T has such a property, assign a weight $wt(v) = O(\sqrt{kN(H)}\log k)$ according to (13). Denote by M_0 the set of all such vertices v. Assign a weight 0 on all remaining vertices of T. According to (13)–(14), the sum of the weights of all vertices of T gives an upper bound for $s(G)$.

We note that the set of the vertices of the subgraphs of G corresponding to M_0 have pairwise empty intersections. This follows from the fact that no vertex of M_0 is an ancestor in T of another vertex of M_0. Define sets $M_1, M_2, \cdots$, where M_i, for $i \geq 1$, is the set of the vertices of T that are parents of vertices from M_{i-1}. By the same argument as above, the set of the vertices of the subgraphs of G corresponding to M_i have pairwise empty intersections. Denote by $wt(M_i)$ the sum of the weights of the vertices of M_i. Our goal is to show that

$$\sum_{i=0}^{\infty} wt(M_i) = O(\sqrt{nN(G)}\log d).$$

We will need an upper and a lower bound for $|M_i|$. For any vertex w of T denote by $n(w)$ and $c(w)$ the number of the vertices and the sum of the squares of the costs of the subgraph of G corresponding to w. Recall that, by definition, each subgraph of G corresponding to a vertex of M_0 has at least d and at most $2d$ vertices. The subgraphs corresponding to vertices of M_1 are at most 2 times larger, by the definition of M_i, and at least 3/2 times larger, according to Theorem 8. By induction, for any $v \in M_i$

$$(3/2)^i d \leq n(v) \leq 2^i(2d)$$

and thus

$$|M_i| \leq n/((3/2)^i d) = n(2/3)^i/d.$$

Now for any $i \geq 1$ we can estimate the weight of M_i as follows

$$wt(M_i) = \sum_{v\in M_i} wt(v) = \sum_{v\in M_i} O(\sqrt{dc(v)}\log n(v))$$

$$= \sum_{v\in M_i} O(\sqrt{dc(v)}(\log d + i + 1)) = \sum_{v\in M_i} O(\sqrt{d}(\log d + i)\sqrt{c(v)})$$

$$= O(\sqrt{d}(\log d + i)) \sum_{v\in M_i} \sqrt{c(v)} = O(\sqrt{d}(\log d + i)) \sum_{v\in M_i} \sqrt{N(G)/|M_i|}$$

$$= O(\sqrt{d}\log(d + i))\sqrt{|M_i|N(G)} = O(\sqrt{dN(G)}(\log d + i)\sqrt{n(2/3)^i/d})$$

$$= O(\sqrt{nN(G)}(\log d + i)(2/3)^{i/2}).$$

The above analysis is valid also for $i = 0$, since for any $v \in M_0$ the weight of v is $O(\sqrt{n(v)c(v)}\log n(v)) = O(\sqrt{dc(v)}\log n(v))$. Thus we have

$$\sum_{i=0}^{\infty} wt(M_i) = \sum_{i=0}^{\infty} O(\sqrt{nN(G)}(\log d + i)(2/3)^{i/2})$$

$$= \sum_{i=0}^{\infty} O(\sqrt{nN(G)}\log d(2/3)^{i/2}) + \sum_{i=0}^{\infty} O(\sqrt{nN(G)}i(2/3)^{i/2})$$

$$= O(\sqrt{nN(G)}\log d) + O(\sqrt{nN(G)}) = O(\sqrt{nN(G)}\log d).$$

Thus $S(G) = O(\sqrt{nN(G)}\log d) = O(n \log d)$. Since G is connected, then G has at least $n-1$ edges and $AWD(G) = S(G)/(n-1) = O(n \log d/(n-1))$ $= O(\log d)$. Thus, G can be embedded into a binary tree with AWD $O(\log d)$ and expansion one.

If G is not connected, then embed each component of G by this algorithm and then combine the resulting trees into one binary tree. ■

If the squares of the edge costs of G do not add to n, then Theorem 7 can still be applied on G after scaling of the costs. Note that in case of unit costs Theorem 7 yields a bound on the average dilation that is independent of the size of the graph and, by [5], is optimal within a constant factor.

As shown below Theorem 7 implies that any planar graph can be embedded into a variety of other important communication structures with a small AWD.

Corollary 1 *Every n-vertex planar graph G of maximum degree d and with non-negative edge costs whose squares sum up to n can be embedded into a hypercube, a butterfly graph, a shuffle-exchange graph, de Bruin graph, or a cube-connected-cycles graph with average weighted dilation* $O(\log d)$ *and expansion one.*

Proof: According to [3, 2, 25], any binary tree can be embedded into any of the above graphs with average dilation $O(1)$ and expansion one. ■

Similar results hold for embedding a planar graph into a linear, a 2-dimensional, or a d-dimensional ($d \geq 3$) array. The proofs of these results are similar to the proof of Theorem 7.

5.2 Pebbling

Pebbling a directed graph is a well studied problem in programming with applications to register allocation and to deriving space-time tradeoffs (see [26, 29] for a survey of results). Pebbling is a one-person game that involves placing pebbles on the vertices of a directed acyclic graph G. During any step, one can remove a pebble from a currently pebbled vertex or place a pebble onto a vertex that currently has no pebble. An additional requirement is that a vertex can be pebbled only if all its predecessors (i.e., all vertices w such that $(w, v) \in E(G)$) are pebbled. The goal is that any vertex be pebbled at least once while minimizing the number of pebbles used at any particular moment. In applications, pebbles correspond to memory units and the objective is to minimize the total memory needed in order to carry out the computation represented by G.

We will study here the weighted version of this problem, where the vertices of G have non-negative costs. Such graphs are used in cases where individual computations involve different amounts of data. The goal is to minimize the maximum sum of the costs on all vertices of G with pebbles placed on them at the same time. We denote this value by $p(G)$.

First, we will need to modify (strengthen) Theorem 1. The next statement shows that one can divide a weighted planar graph by removing a set of vertices of small cost so that both the number of vertices and the weight of each part be small.

Theorem 8 *Let G be a planar n-vertex graph with non-negative weights $weight(v)$ on its vertices that sum up to one and non-negative vertex costs whose squares sum up to $N(G)$. A set C of vertices of total cost $O(\sqrt{N(G)})$ can be found in $O(n)$ time that partitions $V(G)$ into sets A and B such that no edge joins A and B, $weight(A), weight(B) \leq 5/6$, and $|A|, |B| \leq (5/6)n$.*

Proof: Assign a weight $1/n$ to each vertex of v. Find by Theorem 1 a set C_1 of vertices of total cost $O(\sqrt{N(G)})$ that partitions $V(G)$ into two edge disjoint sets X_1 and Y_1 such that $|X_1|, |Y_1| \leq (2/3)n$. Assume w.l.o.g. that $weight(X_1) \geq weight(Y_1)$. Then

$$(1/3)n - |C_1| \leq |X_1| \leq (2/3)n \tag{15}$$

and

$$weight(X_1 \cup C_1) \geq 1/2. \tag{16}$$

Find by Theorem 1 a set of vertices C_2 of total cost $O(\sqrt{N(G)})$ that partitions X_1 into two edge disjoint sets X_2 and Y_2 such that

$$weight(X_2), weight(Y_2) \leq (2/3)weight(X_1). \tag{17}$$

By (16) and (17) we have

$$weight(X_2 \cup C_2) = weight(X_1) - weight(Y_2)$$

$$\geq (1/3)weight(X_1) \geq 1/6 - (1/3)weight(C_1).$$

Similar inequalities hold for $weight(Y_2 \cup C_2)$. Thus

$$weight(X_2 \cup C_1 \cup C_2), weight(Y_2 \cup C_1 \cup C_2) \geq 1/6. \tag{18}$$

Define $C = C_1 \cup C_2$, A to be the set among X_2 and Y_2 with a maximum number of vertices, and $B = V(G) \setminus (A \cup C)$. Then, by (15), $|A| \leq (2/3)n$. Further by (15), $|X_2|+|Y_2|+|C_2| \geq (1/3)n-|C_1|$ and thus $|A|+|C_1|+|C_2| \geq |A| + (|C_1| + |C_2|)/2 \geq (1/6)n$ and $|B| \leq (5/6)n$. We proved $|A|, |B| \leq (5/6)n$.

By (17), $weight(A) \leq 2/3$. By (18) $weight(A \cup C) \geq 1/6$ and therefore $weight(B) \leq 5/6$. Thus $weight(A), weight(B) \leq (5/6)$. ∎

We state also the following corollary of Theorem 8 which we will use later and whose proof is similar to the proof of Theorem 4.

Corollary 2 *Let G be a planar n-vertex graph of maximum vertex degree d, with non-negative weights $weight(v)$ on its vertices that do not exceed $2/3$ and that sum up to one, and non-negative edge costs whose squares sum up to $N(G)$. There exists a $2/3$ edge separator of G of total cost not exceeding $O(\sqrt{dN(G)})$ that partitions $V(G)$ into sets A and B such that $weight(A)$, $weight(B) \leq 5/6$, and $|A|, |B| \leq (5/6)n$. The separator can be found in $O(n)$ time.*

Next we describe a pebbling algorithm based on the method from [21]. Let G be an n-vertex planar graph and Q be the sum of the squares of the vertex costs of G. Recall that our goal is to minimize $p(G)$, the maximum sum of the costs on all vertices of G with pebbles placed on them at the same time. Let $in\_cost(G)$ denote the maximum sum of costs on all predecessors of any vertex of G. Assign a weight on each vertex of G equal to the square of the cost on that vertex. Using Theorem 8, find a partition A, B, C of the vertices of G such that the cost of C is $O(\sqrt{Q})$, A and B are edge disjoint, the cost of neither A nor B exceeds αQ, and $|A|, |B| \leq \alpha n$, for $\alpha = 5/6$.

Pebble the vertices of G in topological order as follows. To pebble the next vertex v remove all pebbles except those on vertices of C. Pebble all predecessors w of v recursively. For this end, for any predecessor w of v, apply recursively the same procedure on the graph G_w induced by w and all its unpebbled predecessors. Since the vertices are pebbled in topological order and since pebbles on vertices of C are never removed, the cost of G_w does not exceed αQ. Thus we have the following recurrence for the maximum value $p(Q,n)$ for $p(G)$ over all n-vertex graphs G whose cost squares sum up to Q and for which $in\_cost(G)$ does not exceed some upper bound I

$$p(Q,n) \leq c\sqrt{Q} + I + p(\alpha Q, \lfloor \alpha n \rfloor), \quad \text{if } n > 1,$$

where c is a positive constant. (Recall that $p(G)$ is the maximum sum of the costs on all vertices of G with pebbles placed on them at the same time.) By solving the recurrence we find $p(Q,n) = O(\sqrt{Q} + I \log n)$.

Thus we proved the following theorem.

Theorem 9 *Let G be an n-vertex planar acyclic directed graph with non-negative vertex costs whose squares sum up to $N(G)$. Let I be the maximum sum of the costs on the predecessors of any vertex of G. The vertices of G can be pebbled so that, at any time, the sum of the costs of all vertices with pebbles be $O(\sqrt{N(G)} + I \log n)$.*

We note that it is possible to develop an algorithm for pebbling G that requires a polynomial time, while the maximum cost of vertices with pebbles placed on them at the same time is only slightly increased. Specifically, the running time of such an algorithm can be reduced to $O(n^{5/3})$ and the maximum cost of vertices with pebbles at any time can be bounded by $O((N(G))^{5/3} + I)$, where I is as in Theorem 9. The algorithm has a similar structure to the algorithm from [21] and uses an $\varepsilon(n)$ separators for $\varepsilon(n) = o(1)$ (instead of 2/3 separators as in Theorem 1). The interested reader is referred to [21] for more details.

5.3 Designing multi–commodity flow trees

The multicommodity flow problem is related to optimally routing several different commodities in a network in order to satisfy a given set of demands. The total amount of flow through any of the edges should not exceed the capacity of that edge. Approximation algorithms exist that find whether there exists a *feasible flow* (flow that satisfies the demands and obeys all capacity constraints) [16]. We consider a related version of the problem, where given the set of demands, one has to construct a network for which a feasible flow exists and whose maximum edge capacity (called a *congestion of the network*) is small. The case where the network is required to be a tree is called the *tree congestion problem* [14].

Formally, let $G = (V, E)$ be a digraph with non-negative real weights (*demands*) $dem(v, w)$ on its edges. The goal is to find a spanning tree T of G with edge capacities for which (i) a feasible flow satisfying the demands determined by the edges of G exists, and (ii) whose congestion is minimum compared to the congestion of *any* tree that meets the requirements. For example, the demands may correspond to the expected volume of phone calls between corresponding pairs of locations and the goal will be to minimize the maximum bandwidth required [27]. Such a tree T is called a *minimum routing tree*. If T is fixed (but not its capacities), one can determine the smallest possible capacity $load(e)$ on any edge e of T as follows. Delete e from T and let T_1 and T_2 be the resulting trees. Let E_1 and E_2 be the sets of edges of G with both endpoints in $V(T_1)$ and $V(T_2)$, respectively. Then

$$load(e) = \sum_{(v,w) \in E(G) \setminus (E_1 \cup E_2)} dem(v, w).$$

For a general weighted demands digraph G, the tree congestion problem is NP–complete [27]. A polynomial approximation algorithm with a logarithmic approximation factor is described in [14]. Seymour and Thomas [27] show that exact solution can be found in polynomial ($O(|G|^4)$) time if G is planar and all demands are integers.

Using our separator theorems, we will provide the first known non-trivial upper bound on the worst-case congestions of minimum routing trees for the class of planar digraphs and will also give an $O(n \log n)$ algorithm that given any n-vertex planar digraph constructs a routing tree with congestion not exceeding the claimed bound.

Theorem 10 *For any n-vertex planar digraph G of maximum degree d and with non-negative edge demands whose squares sum up to n there exists a balanced binary routing tree of G with congestion $O(\sqrt{dn})$.*

Proof: Let G be an n-vertex planar graph of maximum degree d with non-negative edge demands whose squares sum up to n. (We ignore the orientation of the edges of the original graph.) Denote for any subgraph H of G $N(H) = \sum_{(u,v) \in E(H)} (dem(u, v))^2$, where $dem(u, v)$ denotes the demand on edge (u, v). Apply to G the following algorithm.

The routing algorithm

If G consists of a single vertex, then define T to be the tree of that vertex. Otherwise, assign a weight to each vertex v of G equal to the sum of the squares of the demands on all edges incident to v divided by $2N(G)$

(so that the sum of all weights is one). Construct in $O(n)$ time a partition A, B of $V(G)$ and an edge set C of total demand $O(\sqrt{dN(G)}) = O(\sqrt{dn})$ (as in Corollary 2) such that any edge joining A and B belongs to C, the weight of neither A nor B exceeds $\alpha N(G)$, and $|A|, |B| \leq \alpha n$, for $\alpha = 5/6$. Assume w.l.o.g. that $|A| \geq |B|$. Choose a vertex x in A with a minimum weight. Embed recursively the subgraph induced by $A \setminus \{x\}$ into a binary tree T_1 and the subgraph induced by B into another binary tree T_2. Make the roots of T_1 and T_2 children of x and denote by T the resulting tree. Let edges e_1 and e_2 connect x to the roots of T_1 and T_2, respectively.

Analysis

For any edge e in T_1, the load on e with respect to G is not more than the load on e in G_1 plus the sum σ of the demands on all edges of G that join a vertex in T_1 to a vertex in T_2. Similar argument is valid for the load on any edge in T_2. From the choice of x, the load on e_1 or e_2 does not exceed $N(G)/|T_1| + \sigma$. Recall that T_1 has a vertex set $A \setminus \{x\}$. Since $|T_1| \geq n/2 - 1$ and $N(G) = n$, then $N(G)/|T_1| = O(1)$. Furthermore, $\sigma = \sum_{(u,v)\in C}(dem(u,v))^2 = O(\sqrt{dn})$.

Thus the maximum congestion $cong(n)$ of the routing tree constructed by this algorithm for any n vertex planar graph satisfies the recurrence

$$
\begin{aligned}
cong(n) &\leq \max\{cong(n_1), cong(n_2) \mid n_1 + n_2 = n, \quad n_1, n_2 \leq \alpha n\} + O(\sqrt{dn}) \\
&= cong(\alpha n) + O(\sqrt{dn}),
\end{aligned}
$$

whose solution is $cong(n) = O(\sqrt{dn})$. ∎

We formulated the results from this section (Theorems 7, 9, and 10) for the class of planar graphs. The generalizations for the class of graphs of bounded genus are straightforward.

References

[1] Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and its applications. *Proceedings of the 22nd Symp. on Theory of Computing*, pages 293–299, 1990.

[2] S. Bhatt, F. Chung, T. Leighton, and A. Rosenberg. Optimal simulations of tree machines. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 274–282, 1986.

[3] S. N. Bhatt, F. R. K. Chung, J. W. Hong, F. T. Leighton, and A. L. Rosenberg. Optimal simulations by butterfly networks. In *Proc. 20th ACM Symposium on Theory of Computing (STOC)*, pages 192–204, 1988.

[4] S.N. Bhatt and F.T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28:300–343, 1984.

[5] Krzystof Diks, Hristo N. Djidjev, Ondrej Sykora, and Imrich Vrto. Edge separators of planar and outerplanar graphs with applications. *J. Algorithms*, 14:258–279, 1993.

[6] Hristo N. Djidjev. A separator theorem. *Compt. rend. Acad. bulg. Sci.*, 34:643–645, 1981.

[7] Hristo N. Djidjev and Shankar Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 1995, in print.

[8] G.N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.

[9] Hillel Gazit and Gary L. Miller. Planar separators and the Euclidean norm. In *SIGAL 90, Lecture Notes in Computer Science, vol. 450*, pages 338–347. Springer-Verlag, Berlin, Heidelberg, New York, Tokio, 1990.

[10] John R. Gilbert, Joan P. Hutchinson, and Robert E. Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5:391–407, 1984.

[11] John R. Gilbert, Donald J. Rose, and Anders Edenbrandt. A separator theorem for chordal graphs. *SIAM Journal on Algebraic and Discrete Methods*, pages 306–313, 1984.

[12] John R. Gilbert and Robert E. Tarjan. The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50:377–404, 1987.

[13] Michael T. Goodrich. Planar separators and parallel polygon triangulation. *Proceedings of 24th Symp. on Theory of Computing*, pages 507–516, 1992.

[14] Samir Khuller, Balaji Raghavachari, and Neal Young. Designing multicommodity flow trees. *Information Processing Letters*, 50:49–55, 1994.

[15] P. Klein, S. Rao, M. Rauch, and S. Subramanian. Faster shortest-path algorithms for planar graphs. In *26th ACM Symp. Theory of Computing*, pages 27–37, 1994.

[16] Leighton, Makedon, Plotkin, Stein, Tardos, and Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50:228–243, 1995.

[17] F. Thomas Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29^{th} IEEE Symposium on the Foundations of Computer Science*, pages 422–431, 1988.

[18] C.E. Leiserson. Area efficient VLSI computation. In *Foundations of Computing*. MIT Press, Cambridge, MA, 1983.

[19] Richard J. Lipton, D. J. Rose, and Robert E. Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.*, 16:346–358, 1979.

[20] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math*, 36:177–189, 1979.

[21] Richard J. Lipton and Robert E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9:615–627, 1980.

[22] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, pages 265–279, 1986.

[23] Gary L. Miller, Shang-Hua Teng, and Stephen A. Vavasis. A unified geometric approach to graph separators. *Proceedings of the 32nd FOCS*, pages 538–547, 1991.

[24] Gary L. Miller and William Thurston. Separators in two and three dimensions. *Proceedings of the 22nd Symp. on Theory of Computing*, pages 300–309, 1990.

[25] B. Monien and H. Sudborough. Comparing interconnection networks. In *Symposium on Mathematical Foundations of Computer Science*, volume 320, pages 138–153, 1988.

[26] N. Pippenger. Advances in pebbling. In *Annual International Colloquium on Automata, Languages and Programming*, pages 407-417, 1982.

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

[28] Shang-Hua Teng. Points, spheres, and separators: A unified geometric approach to graph partitioning. Ph.D.-Thesis CMU-CS-91-184, Carnegie Mellon University, Pittsburgh, 1991.

[29] H. Venkateswaran and M. Tompa. A new pebble game that characterizes parallel complexity classes. *SIAM Journal on Computing*, 18:533–549, 1989.