

# IMPROVED APPROXIMATIONS OF CROSSINGS IN GRAPH DRAWINGS AND VLSI LAYOUT AREAS\*

GUY EVEN<sup>†</sup>, SUDIPTO GUHA<sup>‡</sup>, AND BARUCH SCHIEBER<sup>§</sup>

**Abstract.** We give improved approximations for two classical embedding problems: (i) minimizing the number of crossings in a drawing on the plane of a bounded degree graph; and (ii) minimizing the VLSI layout area of a graph of maximum degree four. These improved algorithms can be applied to improve a variety of VLSI layout problems. Our results are as follows. (i) We compute a drawing on the plane of a bounded degree graph in which the sum of the numbers of vertices and crossings is  $O(\log^3 n)$  times the optimal minimum sum. This is a logarithmic factor improvement relative to the best known result. (ii) We compute a VLSI layout of a graph of maximum degree four in a square grid the area of which is  $O(\log^4 n)$  times the minimum layout area. This is an  $O(\log^2 n)$  improvement over the best known long standing result.

**Key words.** Approximation Algorithm, Crossing number, Graph Drawing, VLSI layout.

**AMS subject classifications.** 68W25 Approximation algorithms; 05C85 Graph algorithms; 68W35 VLSI algorithms.

**1. Introduction.** In this paper we study two related problems: (1) drawing a bounded degree graph on the plane with the fewest number of crossings of edges and (2) minimization of the VLSI layout area of a graph of maximum degree four in a grid with constant aspect ratio. Considerable attention has been devoted to these problems in the past (see, e.g., [L80, V81, BL84, U84, SSSV97, LR98]).

A *drawing of a graph on the plane* is an injection of the vertices of the graph to points in the plane and a mapping of the edges to simple continuous curves between the vertices' images. A curve may not contain an image of a vertex as an internal point and three (or more) curves may intersect only at an image of a vertex. A *crossing* is an intersection of two curves at a point that is not an image of a vertex. The *size* of a drawing is the sum of the numbers of vertices and crossings in the drawing. The problem of determining the minimum size of a drawing of a graph on the plane is NP-Complete [GJ83].

Bhatt and Leighton, in [BL84], apply a  $B(n)$ -approximate bisection procedure recursively to decompose a bounded degree graph with  $n$  vertices. They prove that this recursive decomposition induces a drawing of size  $O(B^2(n) \log^2 n)$  times the minimum size drawing. Shahrokhi *et al.* [SSSV97] considered straight line drawings induced by decomposition trees and presented a simpler construction of a drawing of the same size as the one achieved in [BL84]. Leighton and Rao [LR98] showed that the above result can be realized with a  $(\frac{1}{3}, \frac{2}{3})$  separator. Leighton and Rao also showed how to find such a separator with size bounded by  $\alpha(n) = O(\log n)$  times the optimal bisector. This implied an  $O(\log^4 n)$  (or  $O(\alpha^2(n) \log^2 n)$ ) approximation algorithm for the drawing size of a bounded degree graph. The bound on the approximation factor

---

\* Part of this work was done while the first two authors visited IBM T.J. Watson Research Center. An extended abstract of this paper appeared in STOC 2000.

<sup>†</sup> Dept. of Electrical Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel.  
E-mail: [guy@eng.tau.ac.il](mailto:guy@eng.tau.ac.il).

<sup>‡</sup> Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA 19104. E-mail: [sudipto@cis.upenn.edu](mailto:sudipto@cis.upenn.edu). This work was done when the author was a graduate student in Stanford University with research supported by IBM cooperative fellowship, an ARO MURI Grant DAAH04-96-1-0007 and NSF Award CCR-9357849.

<sup>§</sup> IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.  
E-mail: [sbar@watson.ibm.com](mailto:sbar@watson.ibm.com).

in this algorithm relies only on the fact that an optimal drawing induces a planar graph which admits small vertex separators. The reason that the approximation algorithm applies only to bounded degree graphs is that, in bounded degree graphs, the vertex separators and the edge separators have roughly the same size. It appears that the ideas of [PSS96, SSSV97] may extend to arbitrary degree graphs. Motivated by graph layout problems, we focus on bounded degree graphs for results related to crossing numbers of graphs and on graphs of maximum degree 4 for results related to layouts.

A *VLSI layout* of a graph of maximum degree 4 is an embedding of the graph in a grid. The vertices of the graph are injected into the grid vertices, and the edges of the graph are mapped into edge disjoint paths in the grid. In fact, we consider the more restrictive “Manhattan” model of routing (in which layer assignment is trivial) and not the “knock-knee” model. Namely, an intersection of two paths (i.e., images of edges) at an internal grid vertex is allowed provided that one path traverses the grid vertex horizontally and the other path traverses the grid vertex vertically. The objective in the VLSI layout area problem is to minimize the area of the grid. Note that to be able to embed a graph in a grid it must be of maximum degree 4. From now on we consider only graphs of maximum degree 4 for the VLSI layout problem.

Leiserson [L80] and Valiant [V81] showed that every planar graph can be embedded in a grid with constant aspect ratio of size  $O(n \log^2 n)$ . Bhatt and Leighton suggested to embed a graph of maximum degree 4 in two steps. First, draw it on the plane, and then apply the approximation algorithm for VLSI layouts of planar graphs to the planar graph resulting from the drawing by adding vertices in crossings. The optimal layout gives a feasible drawing of the same size, therefore the  $O(\log^4 n)$  approximation algorithm for drawing graphs on the plane combined with the  $O(\log^2 n)$  approximation algorithm for VLSI layouts of planar graphs yield an  $O(\log^6 n)$  approximation algorithm for the VLSI layout problem.

A *decomposition tree* of a graph  $G = (V, E)$  is a tree with a mapping of the tree nodes to subsets of vertices as follows. The root is mapped to  $V$ ; every two siblings are mapped to subsets that constitute a partitioning of the subset of  $V$  to which their parent is mapped; and leaves are mapped to subsets containing a single vertex. The cut associated with an internal tree node  $t$  is the set of edges between the subsets to which the children of  $t$  are mapped. Bhatt and Leighton, in [BL84], proposed a special type of decomposition trees, called *bifurcators*. These decomposition trees are binary trees, and the cut sizes associated with their nodes decrease exponentially as a function of the depth of the node. Bhatt and Leighton [BL84] demonstrated that a  $\sqrt{2}$ -bifurcator can be used for a wide variety of problems in VLSI layouts: minimizing capacitive delay, producing fault tolerant layouts, layouts for graphs using prefabricated chips, regular layouts, and layouts minimizing wire crossing, to name a few. (The detailed description of these applications is provided in [BL84].) Using the approximation of drawing size, in retrospect, Bhatt and Leighton provided an  $O(\log^{2.5} n)$  approximation for the optimal  $\sqrt{2}$ -bifurcator.

*Our Results.* In this paper we provide improved approximation algorithms for the above problems. We first provide an  $O(\log^3 n)$  approximation algorithm for the drawing size problem. As a consequence of this result, based on Bhatt and Leighton’s results, we obtain an  $O(\log^2 n)$  approximation for the optimal  $\sqrt{2}$ -bifurcator problem and a corresponding improvement for all its applications.

Our  $O(\log^3 n)$  approximation for the drawing size problem combined with layout algorithms for planar graphs yield an  $O(\log^5 n)$  approximation for the minimum VLSI

layout area. Using further structural properties of the decomposition tree computed in our algorithm, we show that the VLSI layout area problem can be approximated to a factor of  $O(\log^4 n)$ .

In terms of the best approximation factor known for separators, our results can be stated as follows. Let  $\alpha(n)$  denote the smallest known ratio of the separator size (that can be computed efficiently) to the optimal bisector size. We show an  $O(\alpha^2(n) \log n)$ -approximation of the minimum drawing size and an  $O(\alpha^2(n) \log^2 n)$ -approximation of the minimum VLSI layout area.

Our approximation algorithms construct a decomposition tree that can be viewed as an approximation of a decomposition tree  $\tilde{T}$  obtained by recursively bisecting the planar graph induced by an optimal drawing. Such a decomposition tree  $\tilde{T}$  induces a drawing of size  $O(\log n)$  times the optimal drawing size. In addition,  $\tilde{T}$  can be used in Leiserson's embedding algorithm [L80] to compute a layout of area  $O(\log^2 n)$  times the optimal layout area.

The decomposition tree that we compute mimics the useful properties of  $\tilde{T}$ . In the problem of drawing a graph on the plane we attach an estimator  $\phi(t)$  to every tree node that estimates the optimal drawing size of the corresponding subgraph. The estimator quality is one-sided; it may not surpass (twice) the drawing size, but might be much smaller than the drawing size. The estimators have the following two properties that approximate the properties of the drawing sizes of the subgraphs in  $\tilde{T}$ : (a) the cut sizes are bounded by the square root of the corresponding estimators up to a logarithmic error term; and (b) the estimators decrease exponentially as one goes down the tree. The main difficulty in constructing such an "approximated" decomposition tree with estimators is that the only tool we have is approximate separators. To overcome this difficulty we apply a top-down approach with re-balancing to guarantee that the estimators decrease exponentially.

Our technique extends to a framework for computing decomposition trees with estimators of other super-additive functions. (Super-additive functions are functions in which the sum of the function values on disjoint subgraphs is no more than the value of the function on the whole graph). Guha [G00] applied this technique to the fill-in function, yielding better approximations for chordal completion, operation count and certain cases of elimination height and pathwidth.

For the grid embedding algorithm we follow the paradigm suggested by Leiserson [L80]. First, we present a "weighted" version of Leiserson's algorithm that can be implemented when the separation properties of the decomposition tree are given relative to *weights* of subgraphs rather than *sizes* of subgraphs. A tempting approach is to use the decomposition tree computed for approximating the drawing size together with the estimators associated with its nodes as weights for the implementation of Leiserson's algorithm. However, this approach fails since the estimators are not adequate as weights. This is because they do not satisfy a super-additivity property. Therefore, we compute new node weights. These weights require a re-balancing of the decomposition tree before the weighted version of Leiserson's can be applied.

*Organization.* In Section 2 we define the problems, and describe the basic tools: decomposition trees, separators, etc. In Section 3 we describe decomposition trees with estimators and show that they are useful for drawing graphs. In Section 4 we present an algorithm for computing decomposition trees with estimators. In Section 5 an improved approximation of  $\sqrt{2}$ -bifurcators is presented. In Section 6 we present an algorithm for approximating the minimum VLSI layout area.

## 2. Preliminaries.

*Minimum drawing size of a graph.* A drawing  $D$  of a graph  $G$  is a one-to-one mapping of the vertices to points on the plane. We call the image of a vertex its *position*. Every edge is mapped to a simple (non-self intersecting) continuous curve connecting the positions of the end-points of the edge. A curve corresponding to an edge may not contain a position of a vertex as an interior point. A *crossing* is an intersection of the interiors of two curves corresponding to images of edges. (Note that a position of a vertex does not count as a crossing since a position of a vertex cannot be in the interior of a curve.) We do not allow more than two curves to intersect at every crossing. A *point* in a drawing is either a position of a vertex of the graph or a crossing. We denote the minimum number of points, over all drawings of  $G$ , by  $\text{MDS}(G)$ . Note that  $\text{MDS}(G) = n + \text{CR}(G)$ , where  $\text{CR}(G)$  denotes the (minimum) crossing number of  $G$ , and  $n$  is the number of vertices in  $G$ .

The minimum drawing size of a graph satisfies the following *super-additivity* property.

PROPOSITION 2.1. *If the vertex set of  $G$  is partitioned into disjoint sets and the respective graphs induced by them are denoted by  $\{G_i\}_i$  then  $\text{MDS}(G) \geq \sum_i \text{MDS}(G_i)$ .*

*Minimum layout area of a graph.* A *VLSI layout* of a graph of maximum degree 4 is an embedding of the graph in a grid. The embedding consists of an injection of the graph vertices to the grid vertices and a mapping of the graph edges into edge disjoint paths in the grid. The paths into which two edges are mapped may intersect at an internal grid vertex, provided that one edge is mapped to the two horizontal grid edges touching this vertex and the other edge is mapped to the vertical grid edges touching it. The *aspect ratio* of a (rectangular) grid is the ratio of its short dimension to its long one. Let  $\text{AREA}_\sigma(G)$  denote the minimum area required to embed a graph  $G$  in a host grid with aspect ratio  $\sigma$ . Let  $\text{AREA}(G) = \min_{\sigma > 0} \text{AREA}_\sigma(G)$ . Using a “folding” argument, Leiserson[L80] showed that  $\text{AREA}_1(G) \leq 3 \cdot \text{AREA}(G)$ . It follows that considering only embeddings in squares has only a small effect on the minimum area.

Since every layout is also a drawing, and the area of the layout is an upper bound on the size of the drawing, the following property holds.

PROPOSITION 2.2. *For every graph of maximum degree 4,  $\text{AREA}(G) \geq \text{MDS}(G)$ .*

*Decomposition trees.* A (binary) decomposition tree<sup>1</sup>  $T$  of  $G = (V, E)$  is a rooted binary tree  $T$  and a mapping of the tree nodes to subsets of vertices as follows: The root is mapped to  $V$ ; every two siblings are mapped to subsets that constitute a partition of the subset of  $V$  to which their parent is mapped; and leaves are mapped to subsets containing a single vertex. Let  $V_t$  denote the set of vertices to which the tree node  $t$  is mapped, and let  $n_t = |V_t|$ . Let  $G_t = (V_t, E_t)$  denote the subgraph of  $G$  induced by  $V_t$ . To each internal tree node  $t \in T$ , we associate the cut between the vertex sets to which the two children of  $t$  are mapped. Formally, let  $\text{cut}(V_1, V_2)$  denote the edges between vertices in  $V_1$  and vertices in  $V_2$ . The cut corresponding to an internal node  $t$  is denoted by  $\text{cut}_t$ . This cut is defined by  $\text{cut}_t = \text{cut}(V_{t_\ell}, V_{t_r})$ , where  $t_\ell$  and  $t_r$  are the children of  $t$ . For an edge  $e = (u, v) \in E$ , let  $t(e)$  be the tree node for which  $e \in \text{cut}_{t(e)}$ . Note that  $t(e)$  is the lowest tree node that is mapped to a set containing both  $u$  and  $v$ .

---

<sup>1</sup>Decomposition trees may not be binary. However, in this paper we consider only binary decomposition trees.

*Drawings induced by decomposition trees.* In [BL84], Bhatt and Leighton considered (recursive) drawings that are induced by decomposition trees. Shahrokhi *et al.* [SSSV97] described simple drawings that are induced by ordered decomposition trees.<sup>2</sup> Following Shahrokhi and Shi [SS00] we call this drawing a *linear drawing*. An ordered decomposition tree induces a permutation of the vertices by considering the order of the leaves in a preorder traversal. (Recall that each such leaf is mapped to a subset containing just one vertex in  $G$ .) Let  $v_1, \dots, v_n$  denote the vertices of  $G$  in this order. Map the vertices to points along a line (also called the *spine*), unit distances apart, according to their order. All the edges are drawn on one side of the spine as half circles; namely, each edge  $e = (v_i, v_j)$  is drawn as half a circle with diameter  $|i - j|$  and endpoints at  $v_i$  and  $v_j$ .

Shahrokhi *et al.* [SSSV97] count the number of crossings in such an induced drawing using the following observation.

**LEMMA 2.3.** *In a linear drawing, if the curves corresponding to edges  $e = (v_i, v_j)$  and  $e' = (v_k, v_\ell)$  cross, then either  $t(e)$  is an ancestor of  $t(e')$  or  $t(e')$  is an ancestor of  $t(e)$ . (A node is considered both an ancestor and a descendant of itself.)*

*Proof.* Observe that, for each tree node  $t$ , the vertices in  $V_t$  are drawn contiguously along the spine. From the drawing it follows that whenever two edges  $e = (v_i, v_j)$  and  $e' = (v_k, v_\ell)$  cross, then either  $k < i \leq \ell \leq j$  or  $i \leq k \leq j < \ell$ . The lemma follows.  $\square$

A crossing of the curves of  $e$  and  $e'$  is charged to edge  $e$  if  $t(e)$  is an ancestor of  $t(e')$ , and to edge  $e'$  otherwise. From Lemma 2.3 it follows that a crossing of the curves of  $e = (u, v)$  and  $e'$  is charged to  $e$  if  $t(e')$  is either on the path connecting  $t(e)$  to  $u$  in  $T$ , or on the path connecting  $t(e)$  to  $v$ . Recall that for an internal tree node  $t$  the number of edges  $e$  with  $t(e) = t$  is  $|cut_t|$ . We get the following bound on the number of crossings charged to an edge  $e$ .

**COROLLARY 2.4.** *Let  $P(u, v)$  denote the set of nodes in  $T$  on the path from the leaf mapped to  $u$  to the leaf mapped to  $v$ . The number of crossings in a linear drawing that are charged to the edge  $e = (u, v)$  is bounded by  $\sum_{t \in P(u, v)} |cut_t|$ . Shahrokhi *et al.* [SSSV97] also suggested to place the vertices on a circle (or on the corners of a convex polygon) using the same order and draw the edges with straight lines. The number of crossings in this drawing also satisfies Corollary 2.4.*

*Existence of special separators.* In our algorithm we need to compute a *simultaneous* edge separator. Such an edge separator is a cut that partitions a graph in a balanced way according to *two* measures: the number of vertices and their weights. Suppose that we are given a graph  $G$  with vertex weights  $w(v)$ . For a set of vertices  $S$ , let  $w(S)$  denote the sum of the weights of vertices in the set  $S$ .

**Definition 1.** *A cut  $(S, V - S)$  is a  $(\frac{1}{4}, \frac{3}{4})$  separator with respect to the number of vertices if  $\min\{|S|, |V - S|\} \geq \frac{1}{4}|V|$ .*

*A cut  $(S, V - S)$  is a  $(\frac{1}{3}, \frac{2}{3})$  separator with respect to the vertex weights if  $\min\{w(S), w(V - S)\} \geq \frac{1}{3}w(V)$ .*

*A cut  $(S, V - S)$  is a simultaneous (edge) separator if it is a  $(\frac{1}{4}, \frac{3}{4})$  separator with respect to the number of vertices and a  $(\frac{1}{3}, \frac{2}{3})$  separator with respect to the vertex weights.*

The following lemma shows that a small simultaneous separator exists provided that vertex weights are not too big.

**LEMMA 2.5.** *If the maximum vertex weight is bounded by  $\frac{2}{3}$  of the total weight, then there exists a cut of size  $O(\sqrt{\text{MDS}(G)})$  that is a simultaneous edge separator.*

<sup>2</sup>An ordered tree is a tree where the children of each internal node are ordered.

*Proof.* First, for the sake of completeness, we prove the weighted version of the Planar Separator Theorem. Namely, we show that there exists a cut of size  $O(\sqrt{\text{MDS}(G)})$  that separates  $G$  into subgraphs of weight between one third and two thirds of the total weight. Let  $G = (V, E)$ . Consider an optimal drawing of  $G$ , and let  $\tilde{G} = (\tilde{V}, \tilde{E})$  denote the planar graph that is obtained by introducing vertices in the crossings of the optimal drawing. Assign zero weights to vertices in  $\tilde{V} - V$ , and keep the weights of vertices in  $V$  unchanged. Note that  $|\tilde{V}| = \text{MDS}(G)$ . Construct an ordered decomposition tree of  $\tilde{G}$  by applying the Planar Separator Theorem recursively and setting the heavier subgraph as left child in each step. Let  $v_1, \dots, v_n$  denote the order induced by a preorder traversal on the vertices in  $V$ . Define  $S_i = \{v_1, \dots, v_i\}$  and  $S_0 = \emptyset$ . Define  $\ell$  as follows:

$$\ell = \min\{i : w(S_i) \geq w(V)/3\}.$$

Observe that  $w(V)/3 \leq w(S_\ell) \leq 2w(V)/3$ . If  $\ell = 1$ , then this follows from the fact that  $w(v) \leq 2w(V)/3$ , for every  $v \in V$ . If  $\ell > 1$ , then  $w(v_\ell) \leq w(S_{\ell-1}) \leq w(V)/3$  because of the ordering rule in which the heavier subgraph is set as the left child.

We now prove that the size of the cut  $(S_\ell, V - S_\ell)$  is  $O(\sqrt{\text{MDS}(G)})$ . Let  $t_0, t_1, \dots, t_k$  denote the path in the decomposition tree from the root to the leaf corresponding to  $v_\ell$ . The size of the cut  $(S_\ell, V - S_\ell)$  is bounded by  $\sum_{i=0}^k |\text{cut}_{t_i}|$ . The Planar Separator Theorem implies that (a)  $|\text{cut}_{t_i}| \leq O\left(\sqrt{|\tilde{V}_{t_i}|}\right)$ ; and (b)  $|\tilde{V}_{t_i}| \leq (2/3)^i \cdot |\tilde{V}|$ , for every  $i = 0, 1, \dots, k$ . This completes the proof of the weighted version of the Planar Separator Theorem. We note that since all the vertices in  $\tilde{V}$  have constant degree the weighted version of the Planar Separator Theorem follows from a general theorem of Gazit and Miller [GM90]. We detailed the proof for the sake of completeness and also since a similar construction is used later.

A simultaneous separator can be shown to exist as follows. Let  $(S, V - S)$  denote a  $(\frac{1}{3}, \frac{2}{3})$  separator with respect to the weights  $w(v)$ . If  $|S|, |V| - |S| \geq \frac{1}{4}|V|$ , we are done. Otherwise, without loss of generality,  $|S| > \frac{3}{4}|V|$ . Let  $(S_1, S - S_1)$  denote a  $(\frac{1}{3}, \frac{2}{3})$  separator of  $S$  with respect to the number of vertices. Without loss of generality, assume that  $w(S_1) \leq w(S - S_1)$ . In case  $w(S - S_1) \geq \frac{1}{3}w(V)$ , we are done since the partition into  $S - S_1$  and  $V - (S - S_1)$  is a good partition and the size of the cut is  $O(\sqrt{\text{MDS}(G)})$ .

If  $w(S - S_1) < \frac{1}{3}w(V)$ , let  $(S_2, V - S - S_2)$  denote a  $(\frac{1}{3}, \frac{2}{3})$  separator of  $V - S$  with respect to the weights. Without loss of generality, assume that  $w(S_2) \leq w(V - S - S_2)$ . We claim that the partition into  $(S - S_1) \cup S_2$  and  $(V - S - S_2) \cup S_1$  is a good partition. First, note that the size of the cut is  $O(\sqrt{\text{MDS}(G)})$ . Clearly, it is a  $(\frac{1}{4}, \frac{3}{4})$  partition with respect to the number of vertices since  $|S_1|, |S - S_1| \geq \frac{1}{4}|V|$ .

We show that  $\frac{1}{3}w(V) \leq w(V - S - S_2) + w(S_1) \leq \frac{2}{3}w(V)$ , and thus it is a  $(\frac{1}{3}, \frac{2}{3})$  partition with respect to the weights. The lower bound is proved as follows:

$$\begin{aligned} w(V - S - S_2) + w(S_1) &\geq \frac{1}{2}w(V - S) + w(S_1) \\ &= \frac{1}{2} \cdot [(w(V) - w(S - S_1)) + w(S_1)] \end{aligned}$$

Since  $w(V) - w(S - S_1)$  is at least  $\frac{2}{3}w(V)$  the lower bound follows. Also,

$$w(V - S - S_2) + w(S_1) \leq \frac{2}{3}w(V - S) + \frac{1}{2}w(S) \leq \frac{2}{3}w(V) - \frac{1}{6}w(S) \leq \frac{2}{3}w(V)$$

□

*Computing Simultaneous Separators.* The proof of Lemma 2.5 can be made into an efficient algorithm if one could efficiently compute balanced cuts with respect to vertex weights (or number of vertices). The size of the cuts needs to be  $O(\sqrt{\text{MDS}(G)})$ . In this section we present a weaker result; namely, the cut size is  $O(\sqrt{\text{MDS}(G)} \cdot \log n)$ . This additional  $O(\log n)$  factor is added also to the size of the simultaneous separator that we can compute efficiently. The algorithm is based on applying at most twice the Leighton-Rao bi-criteria approximation algorithm for separators.

The Leighton-Rao separator algorithm [LR98] finds separators in unweighted graphs as well as weighted graphs. A cut  $\text{cut}(U, V - U)$  is a  $b$ -balanced cut if the weights of  $U$  and  $V - U$  are at most  $1 - b$  times the total weight. The Leighton-Rao algorithm receives two balance parameters  $b \leq \frac{1}{2}$  and  $b' \leq \min\{b, \frac{1}{3}\}$  and returns a  $b'$ -balanced cut the size of which is  $O(\frac{\log n}{b-b'} \cdot \text{SEP}_b)$ , where  $\text{SEP}_b$  denotes the size of an optimal  $b$ -balanced cut.

We describe how to compute a  $b$ -balanced cut with respect to vertex weights provided that  $b \leq \frac{1}{3}$ . (Computing balanced cuts with respect to the number of vertices is simply the case of uniform weights.) The size of the cut is  $O(\sqrt{\text{MDS}(G)} \cdot \log n)$ .

Set  $b' = 1 - \sqrt{1 - b}$  and apply the Leighton-Rao algorithm. The partitioning yields a  $b'$ -balanced cut with respect to vertex weights  $\text{cut}(U, V - U)$ . If this cut happens to be also  $b$ -balanced, then we are done. Otherwise, assume  $U$  is the heavier part (i.e.,  $w(U) \geq w(V - U)$ ). Since  $\text{cut}(U, V - U)$  is not  $b$ -balanced, it follows that  $w(U) > (1 - b)w(V)$ . Apply the Leighton-Rao algorithm to  $U$  to obtain a  $b'$ -balanced cut  $(U_1, U - U_1)$ . Assume  $U_1$  is the heavier part. Then the cut  $\text{cut}(U_1, V - U_1)$  is  $b$ -balanced. The upper bound follows since  $w(U_1) \leq (1 - b')w(U) \leq (1 - b')^2w(V) = (1 - b)w(V)$ . The lower bound follows since  $w(U_1) \geq \frac{1}{2}w(U) \geq \frac{1}{2}(1 - b) \cdot w(V) \geq b \cdot w(V)$ . (The last inequality holds since  $b \leq \frac{1}{3}$ .) Note that the sizes of both cuts are bounded by  $O(\sqrt{\text{MDS}(G)} \cdot \log n)$ , since the sizes of the optimal  $b$ -balanced cuts of  $V$  and  $U$  are bounded by  $\sqrt{\text{MDS}(G)}$ .

Another way to compute simultaneous separators with “relaxed” balance parameters uses the spreading metric based algorithm [ENRS99] for simultaneous separators.

*Weight functions induced by cuts.* The weight functions used in our decomposition algorithm are defined by cuts. We consider two subsets of vertices  $A, B \subseteq V$  and call  $A$  the *home* set and  $B$  the *outside* set. The weight of  $v \in A$  with respect to its *home* set  $A$  and the *outside* set  $B$ , denoted  $W_{A,B}(v)$ , is defined to be the number of edges connecting  $v$  with vertices in  $B - A$ . In other words,  $W_{A,B}(v)$  equals the number of edges in  $\text{cut}(A, B - A)$  incident to  $v$ . Note that  $\sum_{v \in A} W_{A,B}(v) = |\text{cut}(A, B - A)|$ . Since the graphs are of bounded degree, the weight function is bounded as well. We note that for some weight functions  $W_{A,B}$ , there may be a vertex  $v$  such that  $W_{A,B}(v) > \frac{2}{3} \sum_{v \in A} W_{A,B}(v)$ . Consequently, we cannot apply Lemma 2.5 to find a simultaneous separator in such cases. To keep the presentation simpler we first assume that such cases of unbalanced weight functions do not happen and later note how to cope with them.

### 3. A decomposition tree for minimizing the drawing size.

**3.1. Motivation.** Following Bhatt and Leighton, the drawings that we obtain are induced by decomposition trees. Specifically, these drawings are linear drawings as defined by Shahrokhi *et al.* [SSSV97]. The definition of decomposition trees that are used for approximating the minimum drawing size is motivated by an “ideal” decomposition tree. Loosely speaking, this ideal decomposition tree is obtained by applying the Planar Separator Theorem recursively as in the proof of the weighted

version of the Planar Separator Theorem in Lemma 2.5. More formally, the ideal decomposition tree  $T_I$  is defined as follows. Let  $\tilde{G}$  denote a planar graph that is obtained by introducing vertices in the crossings of an optimal drawing of  $G = (V, E)$ . The decomposition tree  $\tilde{T}$  of  $\tilde{G}$  is obtained by recursively separating  $\tilde{G}$  using the Planar Separator Theorem. The decomposition tree  $\tilde{T}$  induces the decomposition tree  $T_I$  of  $G$  as follows: (a) For every  $t \in \tilde{T}$ , define  $V_t = \tilde{V}_t \cap V$ . (b) Prune the children, if any, of every tree node  $t$  for which  $|V_t| = 1$ .

The size of a linear drawing induced by an ideal decomposition tree  $T_I$  is analyzed as follows. For every internal tree node  $t \in T_I$ ,  $|cut(V_t, V - V_t)| \leq |cut(\tilde{V}_t, \tilde{V} - \tilde{V}_t)|$ . The Planar Separator Theorem implies that  $|cut(\tilde{V}_t, \tilde{V} - \tilde{V}_t)| = O(\sqrt{|\tilde{V}_t|})$ . By Corollary 2.4, in the linear drawing induced by  $T_I$ , an edge  $e = (u, v)$  is charged for at most  $\sum_{t \in P(u, v)} |cut(V_t, V - V_t)| = \sum_{t \in P(u, v)} O(\sqrt{|\tilde{V}_t|})$  crossings. The recursive separation implies that  $|\tilde{V}_t|$  decreases exponentially as one goes down the tree, therefore, an edge  $e = (u, v)$  is charged for  $O(\sqrt{|\tilde{V}_{t(e)}|})$  crossings. To bound the total charges, charge a tree node  $t$  for all the edges  $e$  for which  $t = t(e)$ . Hence, the charging of a tree node  $t$  is bounded by  $O(|cut(\tilde{V}_t, \tilde{V} - \tilde{V}_t)| \cdot \sqrt{|\tilde{V}_t|}) = O(|\tilde{V}_t|)$ . The tree nodes in the same layer of  $\tilde{T}$  induce a partition of  $\tilde{G}$ . It follows that the sum of  $|\tilde{V}_t|$  over all tree nodes  $t$  in the same layer is bounded by  $O(|\tilde{V}|) = O(\text{MDS}(G))$ . Since there are only a logarithmic number of layers in  $T_I$ , the size of the drawing induced by  $T_I$  is  $O(\text{MDS}(G) \cdot \log n)$ .

Our goal is to define a decomposition tree that mimics the properties of an ideal decomposition tree. We attach to every tree node an estimator  $\phi(t)$  that “behaves” like  $\tilde{V}_t$ , so that we can adapt the analysis above for the drawing size that is actually computed. To be able to apply the same analysis  $\phi(t)$  should have the following properties:

1.  $|cut_t| \leq \sqrt{\phi(t)}$ .
2.  $\phi(t)$  decreases exponentially along every path that goes down the tree.
3.  $\phi(t) = O(\text{MDS}(G_t))$ .

Since we do not have a ideal separator procedure, we relax Property 1 to  $|cut_t| \leq \sqrt{\phi(t)} \cdot O(\log n)$ . This degrades the approximation factor by an additional factor of  $O(\log^2 n)$ .

**3.2. A decomposition tree with estimators. Definition 2.** *A decomposition tree  $T$  of a graph  $G$  together with a function  $\phi(t)$  defined over the tree nodes is called a decomposition tree with estimators if the following properties are satisfied for every internal tree node  $t \in T$ :*

- P1:**  $|cut_t| \leq c\sqrt{\phi(t)} \cdot \log n_t$ , for some constant  $c$ .  
**P2:** For every child  $t'$  of  $t$ :  $\phi(t') \leq \frac{2}{3}\phi(t)$ .  
**P3:**  $\phi(t) < 2 \cdot \text{MDS}(G_t)$ .

The following claim shows that drawing induced by a decomposition tree with estimators is within  $O(\log^3 n)$  factor from optimal.

**Claim 3.1.** *The size of a linear drawing of  $G$  that is induced by a decomposition tree  $T$  with estimators  $\phi(\cdot)$  is  $O(\text{MDS}(G) \cdot \log^3 n)$ .*

*Proof.* Consider an edge  $e = (u, v)$ . From Corollary 2.4 it follows that the number of crossings that are charged to  $e$  is bounded by  $\sum_{t \in P(u, v)} |cut_t|$ . Define  $P(e, u)$  and  $P(e, v)$  to be the paths in  $T$  from  $t(e)$  to the leaves that are mapped to  $u$  and  $v$

respectively. It follows that

$$\sum_{t \in P(u,v)} |cut_t| \leq \sum_{t \in P(e,u)} |cut_t| + \sum_{t \in P(e,v)} |cut_t|.$$

We bound each summand in the right hand side as follows. Let  $d(t, t')$  be the number of hops along the path from  $t$  to  $t'$ .

$$\begin{aligned} \sum_{t \in P(e,u)} |cut_t| &\leq \sum_{t \in P(e,u)} c \cdot \sqrt{\phi(t)} \cdot \log(n_t) && \text{(by Property P1)} \\ &\leq c \cdot \log n \cdot \sum_{t \in P(e,u)} \sqrt{\phi(t)} \\ &\leq c \cdot \log n \cdot \sum_{t \in P(e,u)} \sqrt{\phi(t(e)) \cdot \left(\frac{2}{3}\right)^{d(t(e),t)}} && \text{(by Property P2)} \\ &= \log n \cdot O(\sqrt{\phi(t(e))}) \\ &= O(\sqrt{\text{MDS}(G_{t(e)})} \cdot \log n) && \text{(by Property P3)} \end{aligned}$$

By Properties P1 and P3, for each tree node  $t$ ,  $|cut_t| = O(\sqrt{\text{MDS}(G_t)} \cdot \log n)$ , hence we can bound the total number of crossings charged to the edges in  $cut_t$  by  $O(\log^2 n \cdot \text{MDS}(G_t))$ .

Fix some height in  $T$  and consider all the tree nodes  $t_1, \dots, t_r$  that are of this height. Notice that the sets  $V_{t_i}$  are disjoint and thus by the Super-additivity Property (Proposition 2.1),  $\text{MDS}(G) \geq \sum_{i=1}^r \text{MDS}(G_{t_i})$ . It follows that the the total number of crossings charged to the edges in  $\cup_{i=1}^r cut_{t_i}$  is bounded by

$$\sum_i O(\log^2 n \cdot \text{MDS}(G_{t_i})) \leq O(\log^2 n \cdot \text{MDS}(G)).$$

Observe that the tree  $T$  has height  $O(\log n)$ . This is because the estimator  $\phi(\cdot)$  decreases exponentially along each path from the root to a leaf, and the estimator of the root of the tree is  $O(m^2 + n)$  since  $\text{MDS}(G) = O(m^2 + n)$ . It follows that the total number of crossing points is  $O(\log^3 n \cdot \text{MDS}(G))$ .  $\square$

*Remark.* A better approximation algorithm for separators would yield a better approximation factor for the crossing number and the layout area. In particular, an  $\alpha(n)$ -approximation algorithm for separators would imply a tightened Property P1, i.e.,  $cut_t \leq c\sqrt{\phi(t)} \cdot \alpha(n_t)$ , for every tree node  $t$ . This would imply an approximation factor of  $O(\alpha^2(n) \cdot \log n)$  for the crossing number and  $O(\alpha^2(n) \cdot \log^2 n)$  for the layout area (see Sec. 6). This improves the corresponding results of Bhatt and Leighton (Theorems 17 and 19 in [BL84]) by a logarithmic factor and a log-square factor, respectively.

**4. Constructing a decomposition tree with estimators.** Our goal is to construct a decomposition tree  $T$  of  $G$  with estimators  $\phi(t)$ . The challenge in computing the tree is due to the requirements that: (i)  $\sqrt{\phi(t)}$  cannot be too small, since we must be able to find a separator of size  $c\sqrt{\phi(t)} \log n_t$ , for some constant  $c$ ; (ii)  $\phi(t)$  must decay exponentially along every “downward” path; and (iii)  $\phi(t)$  cannot be greater than (twice)  $\text{MDS}(G_t)$ ; The algorithm avoids an exponential search by using “failures” for pruning backtracking and for re-balancing.

Throughout the description, let  $G = (V, E)$  denote the graph for which a decomposition tree is being computed. To simplify notation, we refer to induced subgraphs of  $G$  simply by their vertex sets.

Our algorithm is recursive. The input to the recursive procedures consists of the following inputs:

- a vertex subset  $V'$  (the goal is to compute a decomposition tree for the subgraph  $G' = (V', E')$  induced by  $V'$ ),
- a “guess”  $g$  on the estimate  $\phi(r')$  of the root of this tree, and
- one of the procedures is also input a weight function on the vertices in  $V'$ . This weight function is either the trivial function  $I$  that assign a unit weight to each vertex, or the weight function  $W_{V',B}$  with respect to the home set  $V'$  and an outside set  $B$ . To simplify notation, we sometimes omit the home set and the outside set, and simply denote the non-trivial weight function by  $W$ .

Let  $\text{decompose}(V', g)$  denote our first recursive procedure. The procedure returns one of the following:

- “fail” - if the guess is too small (i.e.,  $g < \text{MDS}(V')$ ); or
- “success” - in which case the procedure returns also a decomposition tree  $T'$  of  $G'$  with estimators  $\phi(t)$ . The estimator of the root  $r'$  of  $T'$  satisfies  $\phi(r') \leq g$ .

Since  $n + \binom{m}{2}$  is an upper bound on  $\text{MDS}(G)$ , the decomposition tree with estimators of the input graph  $G = (V, E)$  is computed by calling  $\text{decompose}(V, n + \binom{m}{2})$ .

The procedure  $\text{decompose}(V', g)$  searches for an approximation of the smallest feasible guess in the range  $[1, g]$ . This search is performed by calling a second procedure  $\text{test-decompose}(V', g', I)$ , for various values of  $g'$ . The output of procedure  $\text{test-decompose}$  is also either “fail” (when the guess is too small) or “success” (when the guess is large enough, but might be too large). In case of success,  $\text{test-decompose}(V', g', I)$  returns a decomposition tree with “lax” estimators. The estimate  $\phi(r')$  assigned to the root by the lax estimators might not satisfy Property P3. That is, it outputs a success also in case the estimators  $\phi(t)$  satisfy Properties P1-P3 for all tree nodes but the root node, and the estimator for the root  $\phi(r')$  satisfies P1 and P2 but  $\phi(r') \geq 2 \cdot \text{MDS}(V')$ . The two procedures could be merged into a single procedure in which a range of guess values is input (the range is  $[1, g]$  for  $\text{decompose}$  and  $[g, g]$  for  $\text{test-decompose}$ ), and a search takes place within the guess range for an approximation of the smallest feasible guess. However, the partition into two procedures seems to simplify the description.

*The procedure  $\text{decompose}(V', g)$ .*

This procedure calls successively  $\text{test-decompose}(V', g/2^i, I)$ , for  $i = 0, 1, 2 \dots$  until  $\text{test-decompose}$  returns “fail”. We distinguish between two cases.

CASE 1: The first call to  $\text{test-decompose}$  (with  $i = 0$ ) returns “fail”. In this case  $\text{decompose}(V', g)$  returns “fail” as well.

CASE 2: Some calls to  $\text{test-decompose}$  returned “success”. Let  $s$  be the index of the last success. In this case  $\text{decompose}(V', g)$  returns “success” with the decomposition tree and estimators computed by  $\text{test-decompose}(V', g/2^s, I)$ .

The failure with  $s+1$  in Case 2 is used to guarantee Property P3 with respect to  $\phi(r')$ . In Lemma 4.2, we prove that if  $\text{test-decompose}(V', g', I)$  fails, then  $\text{MDS}(V') > g'$ . Therefore, the failure with  $s+1$  implies that  $g/2^s < 2 \cdot \text{MDS}(V')$ , and Property P3 is satisfied.

*The procedure  $\text{test-decompose}(V', g, W)$ .*

This procedure has several steps.

STEP 0: If  $V'$  contains a single vertex  $v$ , then return “fail” if  $g < 1$ , “success” if  $g \geq 1$ .

In case of success, the decomposition tree is a single leaf  $t$  with  $V_t = \{v\}$  and  $\phi(t) = 1$ . If  $G'$  contains more than one vertex, go to the next step.

STEP 1: Find a simultaneous separator of  $V'$ . (Recall that such a separator is a  $(\frac{1}{3}, \frac{2}{3})$  separator with respect to the weight and  $(\frac{1}{4}, \frac{3}{4})$  with respect to the

cardinality.) Denote the two parts by  $V'_1$  and  $V'_2 = V' - V'_1$ . Note that when the weight function is the trivial function  $I$ , then a non-weighted  $(\frac{1}{4}, \frac{3}{4})$ -separator suffices.

STEP 2: Let  $c_S$  be the constant such that by Lemma 2.5 there exists a simultaneous separator of  $V'$  of size  $c_S \cdot \sqrt{\text{MDS}(G')}$ . If the size of the separator is greater than  $c_S \cdot \sqrt{g} \cdot \log n'$ , where  $n' = |V'|$ , then return “fail”. Otherwise, go to the next step. The reason for the failure is that we are guaranteed to find a separator of size at most  $c_S \cdot \sqrt{\text{MDS}(V')} \log n'$ , and thus we have a proof that  $\text{MDS}(V') > g$ .

STEP 3: Call  $\text{decompose}(V'_1, 2g/3)$  and  $\text{decompose}(V'_2, 2g/3)$ .

STEP 4: Distinguish between three cases depending on the outcome of the Step 3.

CASE 1: Both recursive calls returned a success. In this case we have computed decomposition trees with estimators for  $V'_1$  and  $V'_2$  that satisfy Properties P1-P3. We construct a decomposition tree for  $V'$  by connecting both subtrees to a root  $r'$ . Set the estimator  $\phi(r')$  to be  $g$ , and return “success” with the resulting tree and estimators. Note that in this case,  $\phi(r')$  might be much larger than  $\text{MDS}(V')$ .<sup>3</sup>

CASE 2: Both recursive calls returned a failure. In this case we return “fail”. The failures imply that  $\text{MDS}(V'_i) > 2g/3$ , for  $i = 1, 2$ , hence by Super-additivity  $\text{MDS}(V') > 4g/3$ , which justifies the failure.

CASE 3: One recursive call returned a success and the other a failure. Assume  $\text{decompose}(V'_1, 2g/3)$  succeeded and  $\text{decompose}(V'_2, 2g/3)$  failed. This is the most complicated case which may occur, even when  $g$  is a good guess, due to lack of balance between  $\text{MDS}(V'_1)$  and  $\text{MDS}(V'_2)$ . Such an imbalance can hamper satisfying P2 (i.e., the exponential decay of the estimators) along the branch from  $V'$  to  $V'_2$ . The algorithm attempts to re-balance the partitioning by recursing on  $V'_2$  and searching for a balanced cut  $\text{cut}(U, V' - U)$ , where  $U \subseteq V'_2$ . To avoid a significant increase in the cut size (compared to  $\text{cut}(V'_1, V'_2)$ ), simultaneous separators are employed as follows.

Define a weight function  $W_2$  as follows. If the procedure  $\text{test-decompose}$  was called with the trivial weight function  $I$ , then  $W_2$  is the weight function defined with respect to the home set  $V'_2$  and the outside set is  $V'$ . Otherwise, that is, if the procedure  $\text{test-decompose}$  was called with a non-trivial weight function  $W$ , then  $W_2$  is weight function defined with respect to the home set  $V'_2$  and the same outside set used for  $W$ . In this way the outside set is always the first vertex set that failed in the current sequence of failures.

In Case 3 continue with the following sub-steps.

STEP 4.1: Call  $\text{test-decompose}(V'_2, g, W_2)$ . If this call fails, then we have proof that  $\text{MDS}(V') \geq \text{MDS}(V'_2) > g$ , therefore, return “fail”.

Suppose that this call succeeded. Consider the recursion tree created

---

<sup>3</sup>In fact,  $\phi(r')$  can be assigned a “tighter” value which could shorten the running time. Namely, set  $\phi(r')$  to be the maximum of  $\frac{3}{2} \cdot \phi(r'_1)$ ,  $\frac{3}{2} \cdot \phi(r'_2)$  and  $\left(\frac{|\text{cut}(V'_1, V'_2)|}{c_S \log n'}\right)^2$ , where  $r'_i$  is the root of the decomposition tree computed for  $V'_i$ . In this case, if  $\phi(r') = \left(\frac{|\text{cut}(V'_1, V'_2)|}{c_S \log n'}\right)^2$ , then by Lemma 2.5, it follows that  $\phi(r') \leq \text{MDS}(V')$ , which means that smaller guesses for  $V'$  are not needed. However, if  $\phi(r') = \frac{3}{2} \cdot \phi(r'_i)$ , then, from Super-additivity, we can only infer that  $\text{MDS}(V') \geq \text{MDS}(V'_1) + \text{MDS}(V'_2) \geq \frac{1}{2} \cdot (\phi(r'_1) + \phi(r'_2)) \geq \frac{1}{2} \cdot \frac{2}{3} \cdot \phi(r')$ . Hence  $\phi(r') \leq 3 \cdot \text{MDS}(V')$ , and one more guess of  $\phi(r')/2$  would still be required for  $V'$ .

by the call to  $\text{test-decompose}(V'_2, g, W_2)$ . Since this call succeeded, the two recursive calls of  $\text{decompose}$  with the parts of  $V'_2$  and guess  $2g/3$  as inputs either both succeeded or at most one of them failed. In case one of the calls failed, we recursively call  $\text{test-decompose}$  on this “failed” part with a guess  $g$ . Since by our assumption the calling procedure  $\text{test-decompose}$  succeeded,  $\text{test-decompose}$  must succeed on this “failed” part with a guess  $g$ . Again, this implies that the two recursive calls of  $\text{decompose}$  with the sub-parts of this “failed” part and the guess  $2g/3$  either both succeeded or at most one failed. It follows that the recursion tree has a chain of “failure” nodes, where each such failure node denotes a failure in a call to  $\text{decompose}$  with the corresponding subgraph and guess  $2g/3$ . We are guaranteed that the calls to  $\text{decompose}$  with the subgraphs that correspond to the siblings of these failed nodes succeeded. This sequence of “mixed” siblings must end with two siblings for which  $\text{decompose}$  succeeded on their corresponding subgraphs and  $2g/3$ .

Figure 4.1 depicts the recursion tree until two successful siblings are encountered. We use the following notation for the vertex sets inducing the subgraphs corresponding to the siblings in this sequence. Let  $F_0 = V'_2$  and  $S_0 = V'_1$  be the vertex sets of the first pair of mixed siblings. Suppose that the sequence of mixed pairs is of length  $x \geq 1$ . Then, for  $1 \leq i < x$ , the sets  $F_i$  (representing failure) and  $S_i$  (representing success) denote the partition of  $F_{i-1}$  computed in Step 1 of  $\text{test-decompose}(F_{i-1}, g, W_{F_{i-1}, V'})$ . The set  $F_i$  is the one for which the call  $\text{decompose}(F_i, 2g/3)$  failed and the set  $S_i$  is the one for which the call  $\text{decompose}(S_i, 2g/3)$  succeeded. This chain of failures ends with the vertex set  $F_{x-1}$  that is partitioned into two sets  $S_x$  and  $S_{x+1}$  for which both calls  $\text{decompose}(S_x, 2g/3)$  and  $\text{decompose}(S_{x+1}, 2g/3)$  succeeded. Let  $S = S_0 \cup S_1 \cup \dots \cup S_{x-1}$ .

**STEP 4.2:** Call  $\text{decompose}(S \cup S_x, 2g/3)$  and  $\text{decompose}(S \cup S_{x+1}, 2g/3)$ . If both fail, then return “fail”. Lemma 4.2 proves that these two failures imply that  $\text{MDS}(V') > g$ .

Suppose that  $\text{decompose}(S \cup S_x, 2g/3)$  succeeded. Recall the fact that  $\text{decompose}(S_{x+1}, 2g/3)$  succeeded as well. Claim 4.1 proves that  $|\text{cut}(S \cup S_x, S_{x+1})| \leq c \cdot \sqrt{g} \cdot \log n'$ . Return “success” with the decomposition tree obtained by connecting the decomposition trees computed for  $S \cup S_x$  and  $S_{x+1}$  with a common root  $r'$  and the estimators returned by the two recursive calls together with  $\phi(r') = g$ . (A tighter assignment of  $\phi(r')$  is possible, as suggested in Footnote 3.)

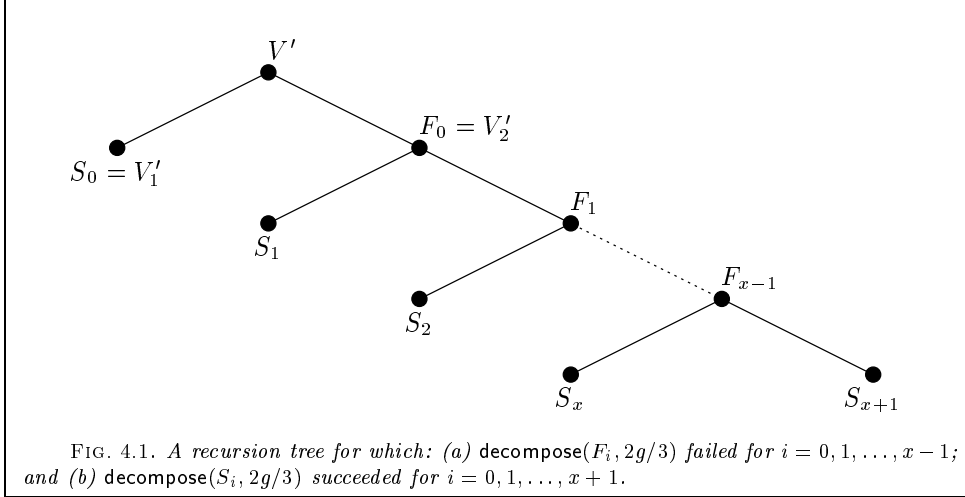
*Correctness.* The following claim proves that the re-balancing performed in Step 4.2 does not increase the cut by much, so that Property P1 holds.

**Claim 4.1.** *If  $\text{test-decompose}(V', g, I)$  reaches Step 4.2, then there exists a constant  $c$  such that*

$$|\text{cut}(S \cup S_x, S_{x+1})| \leq c \cdot \sqrt{g} \cdot \log n'.$$

*Proof.* We use the same notations as in the algorithm. Notice that  $S \cup S_x = V' - S_{x+1}$ , and

$$\text{cut}(S_{x+1}, V' - S_{x+1}) = \text{cut}(S_{x+1}, S_x) \cup \text{cut}(S_{x+1}, V' - F_{x-1}).$$



The success of the  $\text{test-decompose}(F_{x-1}, g, W_{F_{x-1}, V'})$  implies that

$$|\text{cut}(S_{x+1}, S_x)| \leq c_S \cdot \sqrt{g} \cdot \log n'.$$

Since  $(S_{x+1}, S_x)$  is a  $(\frac{1}{3}, \frac{2}{3})$ -separator with respect to the weight function  $W_{F_{x-1}, V'}$ , and since for any subset  $U \subseteq F_{x-1}$ , the total weight  $W_{F_{x-1}, V'}$  over the vertices of  $U$  is  $\text{cut}(U, V' - F_{x-1})$  we get

$$|\text{cut}(S_{x+1}, V' - F_{x-1})| \leq \frac{2}{3} |\text{cut}(F_{x-1}, V' - F_{x-1})|.$$

Continuing in the same manner we get that for  $i = x-1, \dots, 1$ ,

$$|\text{cut}(F_i, V' - F_i)| \leq |\text{cut}(F_i, S_i)| + \frac{2}{3} |\text{cut}(F_{i-1}, V' - F_{i-1})|,$$

and for  $i = 0, \dots, x-1$ ,  $|\text{cut}(F_i, S_i)| \leq c_S \cdot \sqrt{g} \cdot \log n'$ . If we solve this recurrence relation we get

$$|\text{cut}(S_{x+1}, V' - S_{x+1})| \leq \frac{1}{1 - \frac{2}{3}} \cdot c_S \cdot \sqrt{g} \cdot \log n'.$$

Setting  $c = 3 \cdot c_S$  proves the claim.  $\square$

We now prove that failure of  $\text{test-decompose}(V', g, W)$  implies that  $\text{MDS}(V') > g$ . This claim holds regardless of whether  $W$  is a trivial or non-trivial weight function. Notice that the guarantee is one sided; a success might be obtained even if  $g$  is very small. (For example, consider a balanced binary tree as an input graph. All cuts have size 1, so  $g$  can be sub-linear, i.e.,  $(\frac{3}{2})^{\log_2 n}$ , but the drawing size is linear.)

LEMMA 4.2. *If the procedure  $\text{test-decompose}(V', g, W)$  fails, then  $g < \text{MDS}(V')$ .*

*Proof.* The proof is by induction on  $|V'|$ . The induction basis is trivial (see Step 0). In the induction step, we consider the steps in which a failure can be decided upon. Assume that the lemma holds for all graphs with less than  $|V'|$  vertices, and assume that  $g \geq \text{MDS}(V')$  and  $\text{test-decompose}(V', g, W)$  fails.

1. A failure in Step 2 contradicts the assumption because of the guarantee of the partitioning algorithm.

2. A failure in Step 4 Case 2 contradicts the assumption as follows. By the induction hypothesis, failure of  $\text{test-decompose}(V'_i, 2g/3, I)$  implies  $\text{MDS}(V'_i) > 2g/3$ , for  $i = 1, 2$ . From the Super-additivity Property (Proposition 2.1) it follows that  $\text{MDS}(V') > 4g/3$ , a contradiction.
3. A failure in Step 4.1 implies, by the induction hypothesis, that  $\text{MDS}(V'_2) > g$ . Since  $\text{MDS}(V') \geq \text{MDS}(V'_2)$ , we reach a contradiction.
4. The harder case to prove is when a failure is decided upon in Step 4.2. Consider an optimal drawing  $D'$  of  $G'$ , the subgraph of  $G$  induced by  $V'$ . Every point in  $D'$  is either a crossing point between two edges of  $G'$  or a vertex in  $V'$ . Let  $D_{x-1}$  denote the restriction of  $D'$  to a drawing of  $F_{x-1}$ . Recall that  $\text{decompose}(F_{x-1}, 2g/3)$  failed. Therefore, by the induction hypothesis,  $\text{MDS}(F_{x-1}) > 2g/3$ . Since  $D_{x-1}$  is a drawing of the subgraph induced by  $F_{x-1}$ , it follows that  $D_{x-1}$  contains more than  $2g/3$  points.

The points in  $D_{x-1}$  can be partitioned as follows: (a) points of the restriction of  $D_{x-1}$  to  $S_x$ ; namely, vertices of  $S_x$  and crossings between edges of the subgraph induced by  $S_x$  (b) points of the restriction of  $D$  to  $S_{x+1}$ ; and (c) other crossings. Let  $P$  denote the points in  $D_{x-1}$  of type (b) or (c). By swapping  $S_x$  and  $S_{x+1}$  if needed, the number of points in  $P$  is at least half the number of points in  $D_{x-1}$ , and hence, greater than  $g/3$ .

Let  $D_{0..x}$  denote the restriction of  $D'$  to a drawing of  $S \cup S_x$ . The points in the drawing  $D_{0..x}$  are contained in the points of the drawing  $D'$  that are not points in  $P$ . Since the number of points in  $P$  is more than  $g/3$ , we get that the number of points in  $D_{0..x}$  is strictly less than  $\text{MDS}(G') - g/3$ . By our assumption  $g \geq \text{MDS}(G')$ , and thus the number of points in  $D_{0..x}$  is strictly less than  $2g/3$ . Since  $D_{0..x}$  is a drawing of  $S \cup S_x$ ,  $\text{MDS}(S \cup S_x) < 2g/3$ , and by the induction hypothesis,  $\text{decompose}(S \cup S_x, 2g/3, I)$  succeeds. Hence, failure is not decided upon in Step 4.2, a contradiction.

□

The following claim shows that the computed decomposition tree with estimators satisfies Properties P1-P3.

**LEMMA 4.3.** *Let  $T'$  denote the decomposition tree with estimators  $\phi(t)$  returned by a successful call to  $\text{decompose}(G', g)$ . Then  $T$  and  $\phi(t)$  satisfy Properties P1-P3 with respect to  $G'$ .*

*Proof.* We extend the claim to decomposition trees computed by successful calls to  $\text{test-decompose}(G', g, W)$  with the relaxation that Property P3 is not required for the root of  $T'$ .

The proof is by induction on  $|V|$ . It is obvious to show that the induction basis holds. The induction step proceeds as follows:

1. Property P1 is satisfied because  $\text{cut}_t$  is either: (i) a cut computed by the partitioning procedure, in which case the size of the cut is checked in Step 2; or (ii) a cut obtained by re-balancing in Step 4.2, in which case Claim 4.1 guarantees that  $|\text{cut}_t|$  satisfies Property P1.
2. Property P2 is satisfied because by the construction if  $t'$  is a child of  $t$ , then  $\phi(t') \leq \frac{2}{3}\phi(t)$ .
3. Property P3 is satisfied by the induction hypothesis for all internal nodes of  $T$ . It holds for the root  $r'$  of  $T'$  since a success of  $\text{decompose}(G', g)$  implies a failure of  $\text{test-decompose}(G', \phi(r')/2, I)$ . By Lemma 4.2, this implies that  $g/2 < \text{MDS}(G')$ , and Property P3 follows.

□

*Polynomial running time.* Let  $F(n, g)$  and  $G(n, g)$  denote the running times of decompose and test-decompose when the subgraph has  $n$  nodes and the guess equals  $g$ . Note that the weight function does not effect the running time. Let  $p(n)$  denote a polynomial bounding the running time of the partitioning algorithm and the lines in test-decompose that do not call decompose and test-decompose. The functions  $F(n, g)$  and  $G(n, g)$  satisfy the following recurrences: (We make a relaxed upper bound, in assuming that the functions  $F(\cdot)$  and  $G(\cdot)$  are monotone in both parameters.)

$$F(n, g) \leq \sum_{i=0}^{\lceil \log_2 g \rceil} G(n, g/2^i)$$

$$G(n, g) \leq p(n) + 2F\left(\frac{3}{4}n, \frac{2}{3}g\right) + G\left(\frac{3}{4}n, g\right) + 2F\left(n, \frac{2}{3}g\right).$$

The following claim proves that the running time of the decomposition algorithm is polynomial. Recall that  $g$  is always polynomial in  $n$ .

LEMMA 4.4. *There exists constants  $\tau, \gamma, \delta$  such that  $F(n, g) \leq \tau \cdot p(n) \cdot n^\gamma \cdot g^\delta$ .*

**Proof Idea:** One may simply prove this by induction. The reason the proof works is that at each recursion step there are either: (i) a constant number of recursive calls in which one of the parameters (the number of vertices or the guess value) decreases by a constant factor; or (ii) a logarithmic number of recursive calls in which the guess parameters form a geometric sequence. For this reason it was imperative that we went down a path with geometrically decreasing number of vertices. It was also important that in Step 4.2 once the chain of failures ends with two successful siblings, we were able to argue that the guess value should decrease by a constant factor.

Claim 3.1, Lemma 4.3, and Lemma 4.4 imply the following theorem. Note that the the algorithm computes a linear drawing.

THEOREM 4.5. *There exists an  $O(\log^3 n)$ -approximation algorithm for the minimum drawing size of bounded degree graphs.*

*Coping with skewed weight functions.* As mentioned above, for some weight functions  $W_{A,B}$ , there may be a vertex  $v$  such that  $W_{A,B}(v) > \frac{2}{3} \sum_{v \in A} W_{A,B}(v)$ . We call such weight functions *skewed* weight functions. In case of skewed weight functions we cannot apply Lemma 2.5 to find a simultaneous separator. Below, we show how to handle such cases. First, we show that in such cases the total weight is bounded and then prove that because of the bounded weight any cut that is a  $(\frac{1}{4}, \frac{3}{4})$  separator with respect to the number of vertices suffices.

**Claim 4.6.** *Let  $G = (V, E)$  denote a graph with maximum degree  $\Delta$ . For any two subsets of vertices  $A, B \subseteq V$  and a vertex  $v \in A$  such that  $W_{A,B}(v) > \frac{2}{3} \sum_{v \in A} W_{A,B}(v)$ ,  $|cut(A, B - A)| \leq \frac{3}{2} \cdot \Delta$ .*

*Proof.* Recall that  $\sum_{v \in A} W_{A,B}(v) = |cut(A, B - A)|$ . The claim follows since by definition  $W_{A,B}(v) \leq \Delta$ .  $\square$

Simultaneous separators are computed in steps 1 and 4.1 of test-decompose. Modify these steps whenever the weight function is skewed (and non-trivial) to compute a  $(\frac{1}{4}, \frac{3}{4})$  separator with respect to the number of vertices suffices instead of a simultaneous separator. To maintain the validity of the algorithm we need to revise the proof of Lemma 4.1.

**Proof of Lemma 4.1 (revised):** We use the same notations as in the algorithm. Notice that  $S \cup S_x = V' - S_{x+1}$ , and

$$cut(S_{x+1}, V' - S_{x+1}) = cut(S_{x+1}, S_x) \cup cut(S_{x+1}, V' - S_{x-1}).$$

The success of the test-decompose( $F_{x-1}, g, W_{F_{x-1}, V'}$ ) implies that

$$|cut(S_{x+1}, S_x)| \leq c_S \cdot \sqrt{g} \cdot \log n'.$$

If  $cut(S_{x+1}, S_x)$  is a simultaneous separator, then since  $(S_{x+1}, S_x)$  is a  $(\frac{1}{3}, \frac{2}{3})$ -separator with respect to the weight function  $W_{F_{x-1}, V'}$ , and since for any subset  $U \subseteq F_{x-1}$ , the total weight  $W_{F_{x-1}, V'}$  over the vertices of  $U$  is  $cut(U, V' - F_{x-1})$  we get

$$|cut(S_{x+1}, V' - F_{x-1})| \leq \frac{2}{3} |cut(F_{x-1}, V' - F_{x-1})|.$$

Suppose that  $cut(S_{x+1}, S_x)$  is not a simultaneous separator but a  $(\frac{1}{4}, \frac{3}{4})$  separator with respect to the number of vertices. This happens when the weight function  $W_{F_{x-1}, V'}$  is skewed. By Claim 4.6 we have in this case  $|cut(S_{x+1}, V' - F_{x-1})| \leq \frac{3}{2} \Delta$  (where  $\Delta$  is the degree bound).

Combining both cases we have

$$\begin{aligned} |cut(S_{x+1}, V' - F_{x-1})| &\leq \max \left\{ \frac{2}{3} |cut(F_{x-1}, V' - F_{x-1})|, \frac{3}{2} \Delta \right\} \\ &\leq \frac{2}{3} |cut(F_{x-1}, V' - F_{x-1})| + \frac{3}{2} \Delta \end{aligned}$$

Continuing in the same manner we get that for  $i = x-1, \dots, 1$ ,

$$|cut(F_i, V' - F_i)| \leq |cut(F_i, S_i)| + \frac{2}{3} |cut(F_{i-1}, V' - F_{i-1})| + \frac{3}{2} \Delta,$$

and for  $i = 0, \dots, x-1$ ,  $|cut(F_i, S_i)| \leq c_S \cdot \sqrt{g} \cdot \log n'$ . Solving this recurrence relation we get

$$|cut(S_{x+1}, V' - S_{x+1})| \leq \frac{1}{1 - \frac{2}{3}} \left( c_S \cdot \sqrt{g} \cdot \log n' + \frac{3}{2} \Delta \right).$$

Setting  $c = 3c_S + \frac{9}{2} \Delta$  proves the claim.  $\square$

**5. Approximating  $\sqrt{2}$ -bifurcators.** An  $(F, \alpha)$ -bifurcator is a decomposition tree in which  $|cut_t| \leq F \cdot \alpha^{-depth(t)}$  for every tree node  $t$ . An optimal  $\alpha$ -bifurcator is an  $(F, \alpha)$ -bifurcator with the minimum  $F$ . In our algorithm we require that the bounds on the cut sizes decrease geometrically by  $\frac{2}{3}$  relative to *every* node and not only the root. This property was required so that we could bound the sum of the cuts along a path from an internal tree node  $t$  to a leaf by  $O(\phi(t))$ . The following corollary is shown in [BL84, LR98].

**COROLLARY 5.1.** *Given an  $O(\log^k n)$  approximation for the size of drawing on the plane, there exists an  $O(\log^{(k+1)/2} n)$  approximation algorithm for finding the optimal  $\sqrt{2}$ -bifurcator.* Theorem 4.5 implies an improvement of the approximation factor of  $\sqrt{2}$ -bifurcators by a  $\sqrt{\log n}$  factor to  $O(\log^2 n)$ . As mentioned in the introduction, Bhatt and Leighton, [BL84], demonstrated that  $\sqrt{2}$ -bifurcators are useful for minimizing capacitive delay, producing fault tolerant layouts (layouts using prefabricated chip, regular layouts), minimizing wire crossing in layouts and more.

**6. VLSI Layouts.** In this section we present a VLSI layout algorithm for graphs of maximum degree 4. The algorithm embeds a graph  $G$  in a square host grid of size  $O(\text{AREA}(G) \cdot \log^4 n)$ .

An  $O(\log^5 n)$ -approximate VLSI layout area can be derived by applying planar graph embedding algorithms on the drawing computed in Sec. 4, the size of which is  $O(\text{MDS} \cdot \log^3 n)$  [L80, V81]. This is an improvement by a logarithmic factor over the approximation factor of the VLSI layout algorithm of Bhatt and Leighton [BL84]. We show how an additional logarithmic factor can be saved.

**6.1. Weighted version of Leiserson’s Algorithm.** Leiserson’s algorithm for computing layouts of graphs is based on the graphs satisfying a separator theorem. The separator theorem is stated in terms of the *number* of vertices. Namely, every  $n$ -vertex graph in a given family can be separated into two disjoint graphs, each containing at least a constant fraction of the vertices, by removing at most  $f(n)$  edges. For example, in bounded degree planar graphs, the separator is guaranteed to be bounded by the square root of the number of vertices and the size of each part is bounded by  $\frac{2}{3}$  of the original number of vertices. Planar graph embedding algorithms require  $O(n \cdot \log^2 n)$  area.

We consider a “weighted” version of Leiserson’s algorithm in which subgraphs are assigned weights. The input to the weighted version of Leiserson’s algorithm consists of a graph and a decomposition tree with weights  $Wt(t)$  defined over the tree nodes. Loosely speaking, the cut size in every internal tree node is bounded by the square-root of the weight of the tree node, and the weight of each node is divided roughly equally between its two children. The weighted version of Leiserson’s algorithm returns a VLSI layout of area  $O(Wt(r) \cdot \log^2 Wt(r))$ , where  $r$  denotes the root of  $T$ . If  $Wt(r) = O(\text{MDS}(G) \cdot \log^2 n)$ , then  $\log Wt(r) = O(\log n)$ , and by Proposition 2.2 an approximation factor of  $O(\log^4 n)$  follows for the VLSI layout.

The following four properties of the decomposition tree  $T$  and the tree node weights  $Wt(t)$  are sufficient for the “success” of the weighted version of Leiserson’s algorithm:

**Non-triviality:** For each leaf  $\ell$ ,  $Wt(\ell) \geq 1$ .

**Super-additivity:** For any two siblings in the tree  $t_\ell$  and  $t_r$ , with a parent  $t$ ,

$$Wt(t_\ell) + Wt(t_r) \leq Wt(t).$$

**Balance:** For any two siblings in the tree  $t_\ell$  and  $t_r$ ,

$$\max\{Wt(t_\ell), Wt(t_r)\} \leq 2 \cdot \min\{Wt(t_\ell), Wt(t_r)\}.$$

(Note that the above two conditions imply that both the weights  $Wt(t_\ell)$  and  $Wt(t_r)$  are at most  $\frac{2}{3}Wt(t)$ .)

**Separator:** For any internal tree node  $t$ ,  $|cut_t| = O(\sqrt{Wt(t)})$ .

We outline the weighted version of Leiserson’s algorithm. First, the algorithm defines a function  $A(t)$  over the tree nodes that specifies the area allocated for embedding  $G_t$ . The function  $A(t)$  is defined as follows:

$$A(t) = \begin{cases} 1 & \text{if } t \text{ is a leaf} \\ \left( \sqrt{A(t_\ell) + A(t_r)} + \alpha \cdot |cut_t| \right)^2 & \text{otherwise,} \end{cases}$$

where  $t_\ell$  and  $t_r$  denote the children of  $t$  and  $\alpha = 2\sqrt{3}$ .

Given a rectangle  $R_t$  with aspect ratio at least  $\frac{1}{3}$  and area  $A(t)$ , the algorithm computes an embedding of  $G_t$  in  $R_t$  as follows. A sub-rectangle  $R'_t$  of area  $A(t_\ell) + A(t_r)$  that is similar to  $R_t$  is allocated for embedding  $G_\ell$  and  $G_r$ . The sub-rectangle  $R'_t$  is

divided between  $G_{t_\ell}$  and  $G_{t_r}$  in proportion to the weights  $Wt(t_\ell)$  and  $Wt(t_r)$ . After  $G_{t_\ell}$  and  $G_{t_r}$  are embedded in the sub-rectangles allocated for them, the edges of  $cut_t$  are embedded using the unused rows and columns of  $R_t$  (at most 2 columns and 2 rows are used for embedding each edge in the cut). In Claim 6.1 it is proved that there is a sufficient amount of unused rows and columns left in  $R_t$  for routing the edges of  $cut_t$ .

We briefly point out why each of the properties is required: Non-triviality and Super-additivity guarantee that the size of a rectangle allocated for a subgraph is not smaller than the size of the subgraph. Super-additivity also makes sure that the sub-rectangle allocated for the children of  $t$  is not larger than the rectangle allocated for  $t$ . The Balance Property makes sure that the division of the rectangle does not create sub-rectangles whose aspect ratio is below  $\frac{1}{3}$ . Hence, throughout the algorithm, the aspect ratios of the rectangles that serve as hosts for subgraphs are at least  $1/3$ . This also implies that the host grid for the whole graph can be a rectangle of aspect ratio  $1/3$  rather than a square. The Separator Property is used to solve the recurrence and prove the bound on the required area.

The following claim shows that the weighted version of Leiserson’s algorithm succeeds in embedding a graph of maximum degree 4 in a rectangle of area  $A(t)$ .

**Claim 6.1.** *Let  $T$  denote a decomposition tree with node weights  $Wt(t)$  of a graph  $G$  of maximum degree 4. Suppose that the node weights satisfy the Non-triviality, Super-additivity, and Balance Properties. If  $R$  is a rectangle whose aspect ratio is at least  $1/3$  and whose area is at least  $A(t)$ , then the weighted version of Leiserson’s algorithm succeeds in embedding  $G$  in  $R$ .*

*Proof.* The proof follows [L80, Thm 6]. In particular we rely on two observations in Leiserson’s proof. One observation is that aspect ratios do not deteriorate. Namely, suppose we divide a rectangle  $R$  whose aspect ratio is at least  $\frac{1}{3}$  into two sub-rectangles  $R_1$  and  $R_2$  by cutting it along the longer side. Suppose also that this cut is made between a third and two thirds of the side’s length so that the area of the larger sub-rectangle is at most twice the area of the smaller sub-rectangle. Then, the aspect ratios of  $R_1$  and  $R_2$  are also at least  $\frac{1}{3}$ . The second observation is that a new edge can be routed in a given embedding by introducing at most two rows and two columns (this is called “slicing” in [L80]).

The proof is by induction on the height of the tree node. The induction basis obviously holds for leaves. Let  $R'_t$  denote the sub-rectangle of  $R_t$  allocated for  $G_{t_\ell}$  and  $G_{t_r}$ . By definition,  $R'_t$  and  $R_t$  have the same aspect ratio. The area of  $R'_t$  equals  $A(t_\ell) + A(t_r)$ . (There is a “rounding” issue that we ignore, to make the presentation simpler.) We need to show that there are at least  $2 \cdot |cut_t|$  spare rows and columns in  $R_t$  for routing the edges of  $cut_t$ .

Let  $L(R)$  and  $W(R)$  denote the length and width of a rectangle  $R$  (assume  $W(R) \leq L(R)$ ). Denote the aspect ratio of a rectangle  $R$  by  $\sigma(R)$ , namely,  $\sigma(R) = W(R)/L(R)$ . The width of  $R_t$  satisfies:

$$W(R_t) = \frac{area(R_t)}{L(R_t)} = \frac{area(R_t) \cdot \sigma(R_t)}{W(R_t)}.$$

Since the  $area(R_t) \geq A(t)$

$$W(R_t) \geq \sqrt{A(t) \cdot \sigma(R_t)}.$$

Note that since  $R_t$  and  $R'_t$  are similar, they have the same aspect ratio, hence,

$$W(R'_t) = \sqrt{A(t_\ell) + A(t_r)} \sqrt{\sigma(R_t)}.$$

By the last two equations and the definition of  $A(t)$ , it follows that

$$\begin{aligned} W(R_t) &\geq \sqrt{\sigma(R_t) \cdot A(t)} \\ &= \sqrt{\sigma(R_t)} \cdot \left( \sqrt{A(t_\ell) + A(t_r)} + 2\sqrt{3} \cdot |cut_t| \right) \\ &= W(R'_t) + \sqrt{\sigma(R_t)} \cdot 2\sqrt{3} \cdot |cut_t|. \end{aligned}$$

Since  $\sigma(R_t) \geq \frac{1}{3}$ , it follows that  $W(R_t) - W(R'_t) \geq 2 \cdot |cut_t|$ . Hence there are enough spare rows. Since  $L(R_t) - L(R'_t) = (W(R_t) - W(R'_t))/\sigma(R_t)$ , there are also enough spare columns, and the claim follows.  $\square$

The following claim shows that if the decomposition tree satisfies all four properties, then  $A(r) = O(Wt(r) \cdot \log^2 Wt(r))$ , where  $r$  denotes the root of the decomposition tree.

**Claim 6.2.** *Let  $T$  denote a decomposition tree with node weights  $Wt(t)$  of a graph  $G$  of maximum degree 4. Suppose that the node weights satisfy the Non-triviality, Super-additivity, Balance, and Separator Properties. There exists a constant  $c_A$  such that*

$$A(t) = c_A \cdot Wt(r) \cdot \log^2 Wt(r).$$

*Proof.* The proof follows [L80, Sec. 5]. Recall that  $\alpha = 2\sqrt{3}$  and let  $\beta = \max_{t \in T} \{|cut_t|/\sqrt{Wt(t)}\}$ . The Separator Property implies that  $\beta$  is a constant. Define the function  $B(t)$  over the tree nodes as follows:

$$B(t) = \begin{cases} 1 & \text{if } t \text{ is a leaf} \\ \max\{B(t_\ell), B(t_r)\} + \alpha \cdot \beta & \text{otherwise.} \end{cases}$$

We prove by induction that  $A(t) \leq B^2(t) \cdot Wt(t)$ . This obviously holds for leaves. The induction step is proved as follows:

$$\begin{aligned} \sqrt{A(t)} &= \sqrt{A(t_\ell) + A(t_r)} + \alpha \cdot |cut_t| \\ &\leq \sqrt{B^2(t_\ell) \cdot Wt(t_\ell) + B^2(t_r) \cdot Wt(t_r)} + \alpha\beta\sqrt{Wt(t)} \\ &\leq \max\{B(t_\ell), B(t_r)\} \cdot \sqrt{Wt(t)} + \alpha\beta\sqrt{Wt(t)} \\ &= B(t) \cdot \sqrt{Wt(t)}. \end{aligned}$$

The second inequality follows from the induction hypothesis and the definition of  $\beta$ . The third inequality follows from the Super-additivity Property.

The Balance Property and the Super-additivity Property imply that the depth of  $T$  is at most  $\log_{2/3} Wt(t)$ , and hence, it follows that  $B(t) \leq \alpha\beta \log_{2/3} Wt(t)$ , and the claim follows.  $\square$

*Remark.* If the Balance Property is relaxed so that  $\max\{Wt(t_\ell), Wt(t_r)\} \leq (1 - \sigma) \cdot Wt(t)$ , then the aspect ratios of the rectangles allocated in the algorithm are lower bounded by  $\sigma$ . The proof of Claim 6.1 required that  $\alpha \geq 2/\sqrt{\sigma}$ . In the proof of Claim 6.2  $A(t)$  is proportional to  $\alpha^2$ . Therefore, the Balance Property can be relaxed at the cost of increasing  $A(t)$ .

**6.2. Computing a weighted decomposition tree.** Given a graph  $G$ , we show how to construct a decomposition tree  $T$  and a weight function  $Wt(\cdot)$  that obeys the properties sufficient for the success of the weighted version of Leiserson's algorithm.

Observe that recursively separating an optimal drawing of  $G$  generates a decomposition tree that satisfies these properties if the weight function equals the size of the sub-drawing. We show how to compute an “approximation” of such a tree in which the weight of the root is  $O(\text{MDS}(G) \log^2 n)$ .

At first glance, it is tempting to use the decomposition tree computed for the minimal drawing as the decomposition tree required here and to use the estimator function computed before as the required weight function. Two obstacles obstruct this approach: (a) The decomposition tree is not balanced since the ratio of estimators of two siblings is not bounded. (b) The decomposition tree may not obey the Super-additivity Property, as we may have two siblings with estimators that add up to more than the estimator of the parent (i.e., the estimators of each of the children of an internal node  $t$  can be  $2\phi(t)/3$ ). Violating Balance is solved by re-balancing, but violating Super-additivity makes  $\phi(t)$  a bad candidate for a weight function. Super-additivity is obtained by presenting a weight function that is computed bottom-up (to ensure Super-additivity). Balance is obtained afterwards by re-balancing the tree with respect to the weights in a top-down fashion.

**6.2.1. The new weight function.** Let  $T$  denote the decomposition tree of  $G$  with estimator  $\phi(t)$  computed in Section 4. Let  $CW(t)$  be the maximum sum of cut sizes along a path from  $t$  to a leaf in the subtree rooted at  $t$ , where the maximum is taken over all such paths.

**Definition 3.** *The weight function  $Wt(t)$  is defined as follows:*

$$Wt(t) = \begin{cases} 1 & \text{if } t \text{ is a leaf} \\ \max\{Wt(t_\ell) + Wt(t_r), CW^2(t)\} & \text{otherwise.} \end{cases}$$

The following lemma bounds the weight of the root of  $T$ .

**LEMMA 6.3.** *For some constant  $c_W$ ,  $Wt(t) \leq c_W \cdot \text{MDS}(G_t) \cdot \log^2(n_t + 1)$ .*

*Proof.* The proof is by induction. The induction basis for the leaves is obvious. Consider an internal tree node  $t$ . If  $Wt(t) = Wt(t_\ell) + Wt(t_r)$ , then the claim follows by the super-additivity of the minimum drawing size.

If  $Wt(t) = CW^2(t)$ , note that the upper bounds on the sizes of the cuts along any path from  $t$  to any of its leaves decay exponentially and that the size of the first cut (and hence sum of all the cuts along a path from  $t$  to a leaf) is  $O(\sqrt{\phi(t)} \log n_t) = O(\sqrt{\text{MDS}(G_t)} \log n_t)$ . The square of the sum of these cuts is  $O(\text{MDS}(G_t) \log^2 n_t)$ . For an appropriately chosen  $c_W$ , the claim follows.  $\square$

The weight function  $Wt(t)$  satisfies all the properties required except for the Balance Property. We show how to achieve Balance in the following subsection.

**6.2.2. Re-balancing the tree.** Re-balancing proceeds in two steps: First, the tree is rebalanced so that the weight of each node is at most two-thirds the weight of its parent. Second, weights are inflated, if necessary, so that Balanced is achieved.

*Procedure rebalance( $\mathcal{T}$ ).* The  $\text{rebalance}(\mathcal{T})$  procedure balances the tree recursively and assigns weights  $Wt_2(\cdot)$  to tree nodes. The weight of each node is at most two-thirds the weight of its parent. The input consists of a collection of trees (inclusive of the case of a single tree) denoted by  $\mathcal{T}$ , where the initial input consists of a single tree. The procedure outputs a decomposition tree over the union of the leaves of trees in  $\mathcal{T}$ . The following notation is used: the weight of a tree,  $Wt(T')$  is defined to be the weight of its root,  $Wt(r')$ . Given a collection  $\mathcal{T}$  of trees, let  $Wt(\mathcal{T}) = \sum_{T' \in \mathcal{T}} Wt(T')$ .

*The procedure rebalance( $\mathcal{T}$ ).* Assume  $\mathcal{T} = \{T_1, \dots, T_p\}$ .

1. If there exists any  $j$  such that  $\sum_{i=1}^j Wt(T_i)$  is between  $\frac{1}{3}$  and  $\frac{2}{3}$  of  $Wt(\mathcal{T})$ , we create the two collections  $\mathcal{T}_1 = \{T_1, \dots, T_j\}$  and  $\mathcal{T}_2 = \{T_{j+1}, \dots, T_k\}$ . The separating cut is of size 0.
2. If there is no such a  $j$ , then there exists a  $j$  such that  $\sum_{i=1}^{j-1} Wt(T_i)$  and  $\sum_{i=j+1}^p Wt(T_i)$  are both less than  $\frac{1}{3}Wt(\mathcal{T})$ . Starting at the root  $r(T_j)$  of  $T_j$  walk down the tree following the heavier subtree until the first subtree  $T'$  whose weight is less than  $\frac{2}{3}Wt(T_j)$  is reached. Let the subtrees hanging from the path from  $r(T_j)$  to  $T'$  (other than  $T'$ ) be  $T'_1, \dots, T'_q$  (where  $T'_q$  denotes the sibling of  $T'$ ). By Lemma 6.4 below  $\sum_{i=1}^q Wt(T'_i) < \frac{2}{3}Wt(T_j)$ .
3. We have four collections:

$$\begin{array}{ll} \mathcal{T}_a = \{T_1, \dots, T_{j-1}\} & \mathcal{T}_b = \{T_{j+1}, \dots, T_p\} \\ \mathcal{T}_c = \{T'\} & \mathcal{T}_d = \{T'_1, \dots, T'_q\} \end{array}$$

Let  $MIN(a, b) \in \{a, b\}$  be the index of the collection with the smaller weight between collections  $\mathcal{T}_a$  and  $\mathcal{T}_b$ . Recall the weight of a collection  $\mathcal{T}$  is the sum of the weights of the trees in  $\mathcal{T}$ . Similarly,  $MAX(a, b) \in \{a, b\}$  is defined to be index of the larger weighted collection. Define  $MIN(c, d)$  and  $MAX(c, d)$  similarly. Let

$$\begin{array}{l} \mathcal{T}_1 = \mathcal{T}_{MIN(a,b)} \cup \mathcal{T}_{MAX(c,d)} \\ \mathcal{T}_2 = \mathcal{T}_{MAX(a,b)} \cup \mathcal{T}_{MIN(c,d)}. \end{array}$$

4. Call  $\text{rebalance}(\mathcal{T}_1)$  and  $\text{rebalance}(\mathcal{T}_2)$ , and let  $S_i$  denote the decomposition tree computed for  $\mathcal{T}_i$ . Let  $S$  be a decomposition tree for  $\mathcal{T}$  obtained by connecting  $S_1$  and  $S_2$  to a root  $s$ . Set  $Wt_2(s) = Wt(\mathcal{T})$ .

Note that the weight function  $Wt_2(\cdot)$  satisfies all the properties except for Balance. In particular, (a) The Separator Property holds since the size of a cut is either zero or is bounded by  $CW(r(T_j))$  where  $r(T_j)$  is the root of  $T_j$ , which is bounded by  $\sqrt{Wt(T_j)}$ . (b) The Super-additivity Property holds since either  $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$  or  $T_j$  is partitioned into the collections  $\mathcal{T}_c$  and  $\mathcal{T}_d$  and  $Wt(T_j) \geq Wt(\mathcal{T}_c) + Wt(\mathcal{T}_d)$ . Note also that if a single tree is re-balanced, then the weight of the root after re-balancing equals the weight of the root before balancing. So Lemma 6.3 implies that the weight of the root  $s$  of the decomposition tree computed by  $\text{rebalance}(\{\mathcal{T}\})$  satisfies

$$Wt_2(s) \leq c_W \cdot \text{MDS}(G_t) \cdot \log^2(n+1).$$

The following Lemmas prove that the weight of every node is at most two-thirds the weight of its parent.

LEMMA 6.4. *In Step 2 above  $\sum_{i=1}^q Wt(T'_i) < \frac{2}{3}Wt(T_j)$ .*

*Proof.* Consider two cases: (a) If  $Wt(T') > Wt(T_j)/3$ , then it follows by super-additivity. (b) If  $Wt(T') \leq Wt(T_j)/3$ , then also  $Wt(T'_q) \leq Wt(T_j)/3$ . Since the weight of the parent of  $T'$  and  $T'_q$  is not less than  $\frac{2}{3}Wt(T_j)$ , we have  $\sum_{i=1}^{q-1} Wt(T'_i) < Wt(T_j)/3$ , and the claim follows.  $\square$

LEMMA 6.5. *In the decomposition tree returned by  $\text{rebalance}(\mathcal{T})$ , the weight of a non-root node is at most two-thirds the weight of its parent.*

*Proof.* Since  $Wt(\mathcal{T}_{MIN(a,b)}) \leq \frac{1}{2}(Wt(\mathcal{T}) - Wt(T_j))$ , along with  $Wt(\mathcal{T}_{MAX(c,d)}) < \frac{2}{3}Wt(T_j)$ , we have  $Wt(\mathcal{T}_1) < \frac{2}{3}Wt(\mathcal{T})$ . For the bound on  $Wt(\mathcal{T}_2)$  we distinguish two cases. (a)  $Wt(T_j) \leq \frac{2}{3}Wt(\mathcal{T})$ . In this case, since  $Wt(\mathcal{T}_{MIN(c,d)}) \leq \frac{1}{2}Wt(T_j)$ , and  $Wt(\mathcal{T}_{MAX(a,b)}) < \frac{1}{3}Wt(\mathcal{T})$ , we have  $Wt(\mathcal{T}_2) < \frac{2}{3}Wt(\mathcal{T})$ . (b)  $Wt(T_j) > \frac{2}{3}Wt(\mathcal{T})$ .

In this case, since  $Wt(\mathcal{T}_{MAX(a,b)}) \leq Wt(\mathcal{T}) - Wt(T_j)$ , we have  $Wt(\mathcal{T}_2) \leq Wt(\mathcal{T}) - Wt(T_j) + \frac{1}{2}Wt(T_j) < \frac{2}{3}Wt(\mathcal{T})$ .  $\square$

*Inflating weights and achieving balance:*— The Balance Property is obtained by inflating weights using the following local transformation. For every non-root internal tree node  $t$ , let  $t'$  denote its sibling. Define  $Wt_3(t)$  as follows:

$$Wt_3(t) = \max\{Wt_2(t), \frac{1}{2}Wt_2(t')\}.$$

After the inflation all the other properties are preserved. For example, the Super-additivity Property is preserved since if  $Wt_3(t_\ell) > Wt_2(t_\ell)$ , then  $Wt_3(t_\ell) + Wt_3(t_r) \leq \frac{3}{2}Wt_2(t_r) \leq \frac{3}{2} \cdot \frac{2}{3}Wt_2(t) = Wt_2(t) \leq Wt_3(t)$ , where  $t$  denotes the parent of  $t_\ell$  and  $t_r$ .

Having computed a decomposition tree with weights that satisfy all the properties, we conclude with the following theorem.

**THEOREM 6.6.** *Every graph  $G$  of maximum degree 4 can be embedded in polynomial time in any grid of aspect ratio at least  $\frac{1}{3}$  and area  $O(\text{AREA}(G) \log^4 n)$ .*

**Acknowledgments.** The authors would like to thank Seffi Naor, Chandan Deep Nath, János Pach, Satish Rao, and Farhad Shahrokhi for discussions about these problems. The authors wish to thank the anonymous reviewers for many useful comments and suggestions.

#### REFERENCES

- [BL84] S. N. Bhatt and F. T. Leighton, “A Framework for Solving VLSI Graph Layout Problems”, *J. of Computer and System Sciences*, 28:300–343, 1984.
- [ENRS99] G. Even, J. Naor, S. Rao and B. Schieber, “Fast approximate graph partitioning algorithms”, *SIAM J. on Computing*, 28(6):2187–2214, 1999.
- [G00] S. Guha, “Nested graph dissection and approximation algorithms”. In 41st Annual Symposium on Foundations of Computer Science, pp 126–135, Redondo Beach, California, 2000.
- [GJ79] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, California, 1979.
- [GJ83] M.R. Garey and D.S. Johnson, “Crossing number is NP-complete”, *SIAM J. Algebraic Discrete Methods*, 4(3):312–316, 1983.
- [GM90] H. Gazit and G.L. Miller, “Planar separators and the Euclidean norm”, In 1st Int. Symposium on Algorithms (SIGAL), pp. 338–347, LNCS 450, Springer-Verlag, Heidelberg, Germany, 1990.
- [L80] C.E. Leiserson. “Area-efficient graph layouts (for VLSI)”. In 21st Annual Symposium on Foundations of Computer Science, pp. 270–281, Syracuse, New York, 1980.
- [L83] F. T. Leighton, *Complexity Issues in VLSI*, MIT Press, 1983.
- [LR98] T. Leighton and S. Rao, “Multicommodity Max-Flow Min-Cut Theorems and their Use in Designing Approximation Algorithms”, *J. of the ACM*, 46(6):787–832, 1999.
- [LT79] R. J. Lipton and R. E. Tarjan, “A separator theorem for planar graphs”, *SIAM J. on Applied Mathematics*, 36(2):177–189, 1979.
- [PSS96] J. Pach, F. Shahrokhi, and M. Szegedy, “Applications of the crossing number”, *Algebraic Combinatorics* 16:111–117, 1996.
- [SS00] F. Shahrokhi, and W. Shi, “On Crossing Sets, Disjoint Sets, and Pagenumber”, *J. of Algorithms* 34(1):40–53, 2000.
- [SSSV97] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrto, “Crossing Numbers: Bounds and Applications”, *Intuitive Geometry*, Bolyai Society Mathematical Studies 6, Budapest, pp. 179–206, 1997.
- [U84] J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Rockville, Maryland, 1984.
- [V81] L. G. Valiant, “Universality Considerations in VLSI Circuits”, *IEEE Transactions on Computers*, C-30(2):135–140, 1981.