Partitioning Planar Graphs with Costs and Weights

LYUDMIL ALEKSANDROV
Bulgarian Academy of Sciences
HRISTO DJIDJEV
Los Alamos National Laboratory
and
HUA GUO and ANIL MAHESHWARI
Carleton University

A graph separator is a set of vertices or edges whose removal divides an input graph into components of bounded size. This paper describes new algorithms for computing separators in planar graphs as well as techniques that can be used to speed up the implementation of graph partitioning algorithms and improve the partition quality. In particular, we consider planar graphs with costs and weights on the vertices, where weights are used to estimate the sizes of the partitions and costs are used to estimate the size of the separator. We show that in these graphs one can always find a small cost separator (consisting of vertices or edges) that partitions the graph into components of bounded weight. We describe implementations of the partitioning algorithms and discuss results of our experiments.

Categories and Subject Descriptors: G2.2 [**Graph Theory**]: Graph Algorithms—*Mathematics of Computation*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Graph algorithms, graph separators, graph partitioning algorithms, implementation

A preliminary version of this paper appeared in the *4th ALENEX Conf.* (San Francisco, January 2002), Lecture Notes in Computer Science 2409:98–107, Springer-Verlag, New York.
This work was partially supported by the NSERC and EPSRC grant GR/M60750, RTDF grant 98/99-0140, and IST Programme of the EU, Contract IST-1999-14186, ALCOM-FT.
Authors' addresses: Lyudmil Aleksandrov, Bulgarian Academy of Sciences, IPOI, Acad. G. Bonc hev Str. Bl. 25-A, 1113 Sofia, Bulgaria; email: lyualeks@bas.bg. Hristo Djidjev, Los Alamos National Laboratory, Basic and Applied Simulation Science (CCS-5), MS M997, Los Alamos, NM 87545, USA; email: djidjev@lanl.gov. Hua Guo and Anil Maheshwari, School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, ON, CANADA K1S 5B6; emails: {hguo2,anil}@scs.carleton.ca.

1. INTRODUCTION

1.1 Motivation

Graph partitioning is a basic tool in the design of efficient algorithms for solving algorithmic as well as application problems. Informally, the graph partitioning problem consists of finding, in an input graph G, a small set of vertices or edges, called a *separator*, whose removal divides G into prescribed number of parts of roughly equal sizes. A variety of algorithms for solving combinatorial problems are based on the divide-and-conquer strategy combined with an efficient use of graph partitioning on planar graphs. Such algorithms include the best known solutions to the shortest path problem for planar or near-planar graphs [Djidjev et al. 2000; Frederickson 1987; Henzinger et al. 1997], divide-and-conquer algorithms in computational geometry [Goodrich 1992], efficient parallel factorization of sparse matrices [Gilbert et al. 1993; Gilbert and Schreiber 1992], in computational complexity [Lipton and Tarjan 1980], and algorithms for finding a VLSI layout design [Bhatt and Leighton 1984; Leiserson 1983]. In scientific computing, graph partitioning is used to find a distribution of the data among the processors in a high-performance computing system that balances the load and minimizes the interprocessor communication.

In general, the quality of graph partitions is measured by the size of the separator and by the structure of the resulting parts, such as the imbalance between their sizes, their connectivity, and others. For example, an ideal partitioning of a graph G into k parts would be to find a minimum size set of vertices (or edges) whose removal divides G into k components (connected subgraphs) of equal size. Such a task, however, is computationally intractable even for simple graphs. Moreover, in many area of application vertices and edges of the graphs to be partitioned represent different objects and consecutively their presence in the separator and in the parts is measured differently. Thus, a good graph-partitioning algorithm ought to take account of these differences. One useful approach is to consider graphs with weights and costs assigned to its elements. The quality of the partitioning is then measured by the total cost of the elements in the separator and by the imbalance between the total weights of the parts.

The goal of this work is to develop and implement partitioning algorithms for planar graphs with weights and costs on their vertices. In addition, we propose and do experiments with techniques that increase the speed of graph-partitioning algorithms and/or improve the quality of the resulting partitions. All of our algorithms have theoretically guaranteed upper bounds both on the quality of the partitions and on the time needed to find them.

1.2 Related Work

Let $G(V, E)$ be a planar graph with real-valued nonnegative weights and costs assigned to its vertices. For a vertex v its weight and its cost are denoted by $w(v)$ and $c(v)$, respectively. For a subgraph $G' \subset G$ the total sum of the weights and total sum of the costs of the vertices in G' are denoted by $w(G')$ and $c(G')$, respectively. The sum of the squares of the costs of the vertices in G is denoted by $\sigma(G)$, i.e., $\sigma(G) = \sum_{v \in V} (c(v))^2$. Throughout the paper t is a real number in $(0, 1)$.

A set of vertices S is called a *vertex t-separator* (or simply t-separator) if the removal of vertices in S from G leaves no component of weight exceeding $tw(G)$. Similarly, a set of edges whose removal partitions G into components of weights not exceeding $tw(G)$ is called an *edge t-separator*. Note that when a vertex is removed, all edges incident to this vertex are removed as well.

There is a vast amount of literature devoted to graph separators. Here we discuss briefly only the work that, in our opinion, is closely related to our results. Initially, the existence and construction of separators in planar graphs with unit costs was studied by Lipton and Tarjan [1979] in their celebrated paper. They showed that in any n-vertex planar graph with nonnegative weights on its vertices, there exists a $(2/3)$-separator of no more than $2\sqrt{2n}$ vertices and proposed a linear time algorithm to construct it. The existence of various separators as well as algorithms for their construction have been proposed for several classes of nonplanar graphs with unit costs. These include graphs of bounded genus [Aleksandrov and Djidjev 1996; Djidjev 1981; Gilbert et al. 1984b], chordal graphs [Gilbert et al. 1984a], and geometric graphs in two and three dimensions [Miller et al. 1997].

Generalizing the notion of the size of a graph separator, Djidjev [2000] considers planar graphs with real-valued nonnegative costs assigned to their vertices. The main theorem in Djidjev [2000] states that in any planar graph G, with weights and costs on its vertices, there exists a $(2/3)$-separator of total cost not exceeding $2\sqrt{2\sigma(G)}$. Moreover, it is shown that this bound is tight up to a constant factor and that such separator can be computed in linear time.

In spite of the extensive algorithmic work on constructing separators and their usage in the design of other combinatorial algorithms, there exists a small amount of work on the implementation of partitioning algorithms. Gilbert et al. [1998] describe a Matlab implementation of a partitioning algorithm for geometric graphs (graphs whose embedding in two or three dimensions satisfies certain requirements) based on the theoretical work of Miller et al. [1997]. A number of researchers, mainly working in the scientific computing area, have designed and implemented graph-partitioning algorithms, which are based on heuristics like, e.g., the Kernighan–Lin method [Kernighan and Lin 1970], spectral methods [Pothen et al. 1990], genetic algorithms [Maini et al. 1994], or combinations of different methods (see Camp et al. [1994] for a survey of results). Although these algorithms usually work well, in practice, for the chosen applications, they can not guarantee asymptotically optimal bounds on the size of the obtained separators in the worst case. Most of them, however, work on general graphs and are based on relatively simple and easy to code routines, which explains their popularity among practitioners. Software packages containing implementations of such type of algorithms include CHACO [Hendrickson and Leland 1994] and METIS [Schloegel et al. 2000; Karypis 1998]. Farrag [1998] in her thesis has proposed an implementation of Aleksandrov and Djidjev's algorithm [Aleksandrov and Djidjev 1996] for planar graphs. One of the main observations was that the quality of the obtained partitions is very much dependent on the choice of the root of the breadth-first search-spanning tree.

1.3 New Results

The main results of this paper are:

1. We show, that in any planar graph G with weights and costs on its vertices, there exists a vertex t-separator whose cost is at most $4\sqrt{2\sigma(G)/t}$, and provide an algorithm for constructing such a separator. The cost of the separator is within a constant factor of the optimal for the class of planar graphs. The running time of the algorithm is $O(|V| + T_{\mathrm{SSSP}}(G))$, where $T_{\mathrm{SSSP}}(G)$ is the time for computing single source shortest path tree in G.[1]
2. We present a technique for tuning the above algorithm that results in considerable reduction of the cost of the produced separators. The computational cost of this technique is theoretically analyzed in the case of integer costs and experimentally tested in the general case.
3. We show that in any planar graph G with weights and costs on its vertices, there exists an edge t-separator of, at most, $4\sqrt{2\Delta(G)/t}$ edges, where $\Delta(G) = \sum_{v \in V}(\deg(v))^2$ and $\deg(v)$ is the degree of a vertex v. The size of the edge separator is asymptotically optimal for the class of planar graphs. Our algorithm constructs such a separator in $O(|V| + T_{\mathrm{SSSP}}(G))$ time.
4. We implement our algorithms and support the theoretical results by extensive experiments. Unlike other implementations of graph-partitioning algorithms based on heuristics, ours always produce separators whose worst-case cost (size) is bounded. Our experiments show that the actual constant in the estimate on the cost of separators is about three times smaller than the one stated in point (1) In addition, experiments indicate that the tuning technique stated in point (2) is effective and computationally efficient.

The result stated in point (1) is a generalization of many existing results on separators, which follow by appropriately choosing weights, costs, and a parameter t. In most of the previous work, the case $t = 2/3$ was considered. Separators for smaller values of t have been constructed by repeated $2/3$ separation. This approach, however, adds an extra $O(\log(1/t))$ factor to the time complexity and, more importantly, results in low-quality partitions. In comparison, the algorithm stated in point (1) constructs t-separators, for any value of t, directly in linear time and produces high-quality partitions. This is supported by our experiments in which the execution times are consistently small for different values of t and, hence, our results are not only theoretically, but also practically, attractive.

1.4 Organization of the Paper

Section 2 consists of three subsections. In the first subsection we outline our approach and algorithm. In the second we present some necessary details, prove the correctness of the algorithm and establish an upper bound on the cost of the resulting separators. In the third subsection we show how the algorithm

[1]In Henzinger et al. [1997] it is shown that SSSP tree in planar graphs can be computed in linear time, but the algorithm is complicated and the hidden constant seems to be large. Alternatively, we use Dijkstra's algorithm and obtain simple and practical algorithm.

for constructing vertex t-separators can be used for obtaining low-cost edge separators. In Section 3, we describe and analyze a technique for reducing the cost of separators. In Section 4, we discuss the implementation of our algorithms and provide relevant experimental results.

2. PARTITIONING OF PLANAR GRAPH WITH WEIGHTS AND COSTS

2.1 Preliminaries and Outline of the Approach

Recall that $G = (V, E)$ is a planar graph with nonnegative weights and costs associated to its vertices. The weight and the cost of a vertex $v \in V$ are denoted by $w(v)$ and $c(v)$, respectively. The total weight and cost of the vertices in a subgraph $G' \subset G$ are denoted by $w(G')$ and $c(G')$, respectively. We denoted by $\sigma(G)$ the sum of the squares of the cost of the vertices of G, i.e., $\sigma(G) = \sum_{v \in V}(c(v))^2$. For a real $t \in (0, 1)$, a vertex t-separator for G was defined as a set of vertices whose removal leaves no component of total weight exceeding $tw(G)$. Without loss of generality, we assume that G is connected. In this subsection, we outline our algorithm that constructs a t-separator S_t in G of total cost

$$c(S_t) = \sum_{v \in S_t} c(v) \leq 4\sqrt{2\sigma(G)/t}, \tag{1}$$

and outputs the partition formed by the connected components of $V \setminus S_t$. The algorithm is based on the approach introduced in Lipton and Tarjan [1979] and extended in Aleksandrov and Djidjev [1996] and Djidjev [2000]. In this approach, the separator construction relies on a single source shortest path (SSSP) tree T rooted at a dummy vertex ρ. The vertex ρ is assigned zero weight and zero cost and is connected to G, so that the planarity is preserved. The SSSP tree T is constructed with respect to the distance, called as the *cost distance*, and is defined as follows: the cost of traveling along a single edge equals the cost of its target vertex and the cost of a path equals the sum of the costs of the edges in that path. Equivalently, the cost of a path equals the sum of the costs of the vertices on that path except for the vertex where it originates. In the uniform case, where all vertices have equal costs, T is simply the breadth-first search tree rooted at ρ.

Our algorithm constructs a t-separator S_t in three phases. In the first phase, named *partitioning by levels*, a set of levels $\mathcal{L}$ in T is selected and vertices in these levels are added to S_t. Intuitively, a level in T consists of the vertices that are at the same cost distance from the root ρ. The removal of the vertices in a level divides the graph into two disjoint subgraphs induced by the vertices "below" and "above" that level (see later Figure 1a). The choice of levels greatly affects the quality of the resulting separator. We propose two alternative methods called "equidistant" and "optimized" choice of levels. Both methods ensure the validity of the estimate (1) on the cost of the resulting separator. While the optimized choice of levels is computationally more demanding, its application usually significantly reduces the cost of the produced separators. Therefore, we carefully tuned the implementation of this method as described in Section 3.

In the second phase, named *partitioning by fundamental cycles*, each connected component G_j, whose weight exceeds $tw(G)$, is further partitioned as

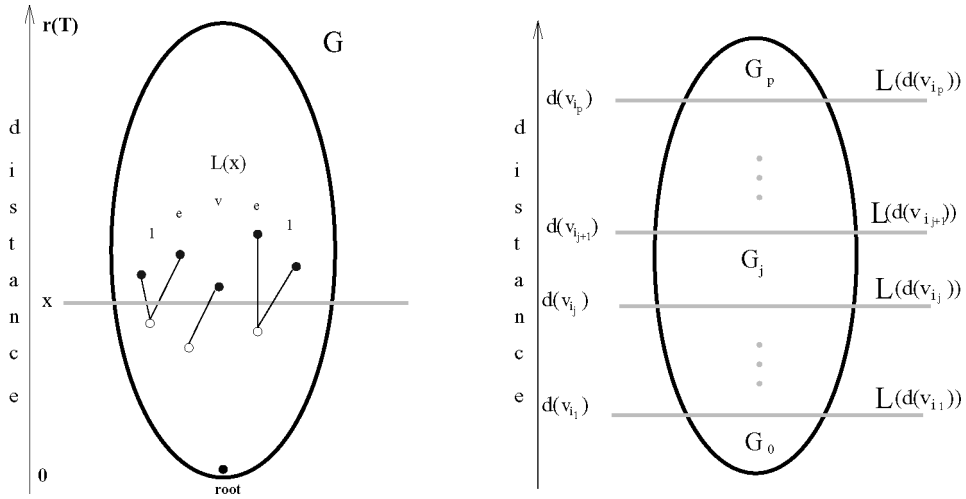

Fig. 1. (a) The five edges shown form the set $E(x)$. The target vertices of these edges form the level $L(x)$. (b) The removal of levels $L(d(v_{i_1})), \ldots, L(d(v_{i_p}))$ partitions G into subgraphs $G_0, G_1, \ldots, G_p$.

follows: first, a dummy vertex ρ_j of zero weight and zero cost is added and connected to the vertices in the lowest level in G_j. Second, a planar representation of this graph is constructed and additional edges are introduced in order to triangulate each of its faces. The obtained graph is called *triangulation* of G_j and denoted by $\tilde{G}_j$. It is straightforward to see that any t-separator of $\tilde{G}_j$ is a t-separator of G_j. Third, a SSSP tree T_j rooted at ρ_j is constructed in $\tilde{G}_j$. Note that by our construction the distance between any two vertices in T_j is, at most, their distance in T. We refer to this property by saying that T_j is *consistent* with T. Fourth, $\tilde{G}_j$ is partitioned by means of fundamental cycles with respect to T_j. Recall that a fundamental cycle in $\tilde{G}_j$ with respect to T_j is a cycle consisting of a single nontree edge plus the unique tree path between its endpoints. We find a set of fundamental cycles $\mathcal{C}$ whose removal partitions $\tilde{G}_j$, and, hence, G_j, into components of weights at most $tw(G)$ by applying the so called *separation tree* technique, which is described in detail in Aleksandrov and Djidjev [1996]. Vertices in this set of fundamental cycles are added to S_t. After the second phase, the vertices in S_t already form a t-separator of G, whose cost satisfies (1).

As our experiments show the quality of the partition corresponding to the separator S_t constructed in the first two phases often can be improved, by "cleaning up" the separator from unnecessary vertices and by merging components of small weight. This is done in the third phase, called *packing*. First, we consequently select separator vertices that are not adjacent to any component of $G \setminus S_t$ and whose weight does not exceed $tw(G)$. These "isolated" vertices are then removed from the separator and constitute new single vertex components of $G \setminus S_t$. Second, we consecutively select separator vertices that are adjacent to exactly one component of $G \setminus S_t$ and absorb them into that component, provided that the weight condition is not violated. Third, we merge neighboring components of small weight into a new component, provided that the weight

Table I. Outline of Our Algorithm Computing a Vertex t-Separator and the Corresponding Partition of a Connected Planar Graph G with Weights and Costs Assigned to Its Vertices

ALGORITHM OUTLINE:

Input: A connected planar graph G with weights and costs. A real $t \in (0, 1)$.
Output: A vertex t-separator S_t and the corresponding partition of G.

Phase I: PARTITIONING BY LEVELS

Step 1. Add a dummy vertex ρ of zero weight and zero cost and connect it to an arbitrary vertex of G.
Step 2. Compute SSSP tree T rooted ρ with respect to the cost-distance in G.
Step 3. Select a set of levels $\mathcal{L}$ in T, using either the "equidistant" or the "optimized" choice of levels.
Step 4. Insert vertices in the levels in $\mathcal{L}$ into S_t and compute connected components $G_1, \ldots, G_m$ of the graph $G \setminus S_t$.

Phase II: PARTITIONING BY FUNDAMENTAL CYCLES

Step 5. For each "heavy" component G_j, such that $w(G_j) > tw(G)$, do Steps 5.1 through 5.4 below.
Step 5.1. Compute a planar triangulation $\tilde{G}_j$ of G_j.
Step 5.2. Construct a SSSP tree T_j consistent with T.
Step 5.3. Find a set of fundamental cycles $\mathcal{C}$ with respect to T_j, whose removal partitions G_j into components of weight $< tw(G)$.
Step 5.4. Insert vertices on the cycles in $\mathcal{C}$ into S_t and compute the connected components of $G_j \setminus \mathcal{C}$.

Phase III: PACKING

Until possible repeat Steps 6, 7 and 8 below.
Step 6. Select a vertex $v \in S_t$ that is not adjacent to any component of $G \setminus S_t$ and $w(v) \leq tw(G)$. Remove v from S_t and update $G \setminus S_t$.
Step 7. Select a vertex $v \in S_t$ that is adjacent to exactly one component K of $G \setminus S_t$ and $w(v) + w(K) \leq tw(G)$, then move v from S_t into K.
Step 8. Select a pair of neighboring components K_1 and K_2 of $G \setminus S_t$, such that $w(K_1) + w(K_2) + w(\partial(K_1, K_2)) \leq tw(G)$. Merge K_1, K_2, and $\partial(K_1, K_2)$ into a new component K and remove $\partial(K_1, K_2)$ from S_t.

of the new component does not exceed $tw(G)$. More precisely, let K_1 and K_2 be components of $G \setminus S_t$. We define *common boundary* of K_1 and K_2 as the set of vertices that are adjacent to K_1 and K_2 and to no other component. The common boundary of K_1 and K_2 is denoted by $\partial(K_1, K_2)$. Two components are called neighboring if they have nonempty common boundary. Two neighboring components K_1 and K_2 are merged into new component K provided that $w(K_1) + w(K_2) + w(\partial(K_1, K_2)) \leq tw(G)$. The new component K is obtained as the subgraph of G induced by the vertices in K_1, K_2, and $\partial(K_1, K_2)$. The third phase terminates when there are no more separator vertices to be removed, to be absorbed, or neighboring components to be merged. The outline of our algorithm is presented in algorithmic form in Table I.

2.2 Construction of a Vertex t-Separator

In this subsection, we present necessary details about the first two phases of the algorithm outlined in the previous subsection. We prove that the set of vertices

S_t formed after the execution of the first two phases is a vertex t-separator for G and establish the estimate (1) on its cost.

For the sake of convenience, we begin with the second phase. Let us consider a "heavy" component G_j obtained after the implementation of the first phase. Let $\tilde{G}_j$ be the triangulation of G_j obtained in Step 5.1 (see Table I). Let T_j be a spanning tree of $\tilde{G}_j$ rooted at a dummy vertex ρ_j of zero weight and zero cost. We define radius of T_j to be the maximum cost of a path in T_j originating at ρ_j and denote it by $r(T_j)$. A fundamental cycle in $\tilde{G}_j$ with respect to the tree T_j is a cycle formed by any nontree edge e, plus the unique path in T_j between the two endpoints of e.

In Aleksandrov and Djidjev [1996], it is shown that there exists a set $\mathcal{C}$ of, at most $\lfloor 2/t \rfloor$, fundamental cycles, $C_1, C_2, \ldots, C_q$, where $q \leq \lfloor 2/t \rfloor$, whose removal leaves no component with weight exceeding $tw(G_j)$. Thus, the vertices in the cycles in $\mathcal{C}$ form a t-separator in G_j, whose cost can be estimated by

$$c(\mathcal{C}) \leq \sum_{i=1}^{q} c(C_i) \leq 2qr(T_j) \leq 2\lfloor 2/t \rfloor r(T_j) \tag{2}$$

Note that any fundamental cycle in T_j has cost, at most, $2r(T_j)$. The above discussion is summarized in the following theorem:

THEOREM 1. *Let $\tilde{G}$ be a triangulated planar graph with nonnegative weights and costs on vertices. Let T be a spanning tree of G rooted at a dummy vertex ρ of zero weight and zero cost. For any $t \in (0, 1)$, there exists a t-separator for $\tilde{G}$, consisting of the vertices in a set of fundamental cycles, whose cost does not exceed $2\lfloor 2/t \rfloor r(T)$, where $r(T)$ is the radius of T with respect to the cost distance in $\tilde{G}$. Such a separator can be found in $O(|\tilde{G}|)$ time.*

Next, we describe the first phase of our algorithm that partitions the input graph G into components with "short" SSSP trees by removal of levels. Recall that the graph G is connected and let T be a SSSP tree rooted at a dummy vertex ρ. Assume that G has n vertices and that they are enumerated by their distance to ρ, i.e., $0 = d(\rho) \leq d(v_1) \leq \ldots \leq d(v_n) = r(T)$, where n is the number of vertices in G and $d(v_i)$ denotes the distance from ρ to v_i (see Figure 1a).

Definition 1. For a real $x \in [0, r(T)]$ we define a set of tree edges $E(x)$ by $E(x) = \{e = (u, v) : e \in E(T),\ d(u) < x \leq d(v)\}$, and a set of vertices $L(x)$ called level by $L(x) = \{v : e = (u, v) \in E(x),\ d(u) < d(v)\}$.

In the following three lemmas, we establish some properties of the levels in G.

LEMMA 1. *Let x, y be real in $(0, r(T))$. If the interval (x, y) does not contain any of the distances $d(v_i)$, then $L(x) = L(y)$.*

PROOF. Presented in the Appendix. □

Hence, there are at most n different levels in G, i.e., for any $x \in (0, r(T))$ the level $L(x)$ coincides with one of the levels $L(d(v_1)), \ldots, L(d(v_n))$. Any of the levels $L(x)$, $x \in (0, r(T))$, cuts G into two subgraphs as stated in the following lemma.

LEMMA 2. *For any real $x \in [0, r(T)]$, the vertices of G are partitioned into three subsets $V_- = \{v : d(v) < x\}$, $L(x)$, and $V_+ = \{v : d(v) > x, v \notin L(x)\}$. No edge in G joins V_- and V_+.*

PROOF. Presented in the Appendix. □

Therefore, the removal of a level $L(x)$ with $x \in (0, r(T))$ partitions G into two subgraphs G_- and G_+ induced by the vertex sets V_- and V_+, respectively. In the next lemma, we obtain an estimate on the integral of $c(L(x))$, which will be used in the estimation of the cost of our separators.

LEMMA 3. $\int_0^{r(T)} c(L(x))dx = \sigma(G) = \sum_{v \in V}(c(v))^2$.

PROOF. Presented in the Appendix. □

Lemma 2 and Theorem 1 can be used together for obtaining a variety of t-separators in G. Namely, let $i_1, \ldots, i_p$ be a sorted sequence of distinct indices, i.e.,

$$0 = i_0 < i_1 < \ldots < i_p < i_{p+1} = n + 1$$

Removal of the levels $L(d(v_{i_1})), \ldots, L(d(v_{i_p}))$ partitions G into subgraphs $G_0, G_1, \ldots, G_p$ (see Figure 1b). For each of the graphs G_j, whose weight exceeds $tw(G)$, we construct a $tw(G)/w(G_j)$-separator S_j consisting of the vertices in a set of fundamental cycles with respect to a spanning tree T_j consistent with T. Recall that a tree T_j is called consistent with T if the distances in T_j are, at most, the distances in T. In the case where $w(G_j) \leq tw(G)$, we assume that $S_j = \emptyset$. Separators S_j are constructed as follows:

Consider the graph G_0. We construct a triangulation of G_0 and then find a SSSP tree T_0 rooted at ρ. Obviously, T_0 is consistent with T. Applying Theorem 1, we obtain a $tw(G)/w(G_0)$-separator S_0, such that

$$c(S_0) \leq 2\lfloor 2w(G_0)/tw(G)\rfloor r(T_0) \leq 2\lfloor 2w(G_0)/tw(G)\rfloor (d(v_{i_1-1}) - d(\rho))$$

Next, consider the graph G_1. Assume, for the sake of simplicity, that G_1 is connected. We connect ρ to all the vertices in G_1 adjacent to $L(d(v_{i_1}))$ in G. The resulting graph is planar. We compute a triangulation of this graph and find a SSSP tree T_1 rooted at ρ. The tree T_1 is consistent with T. Using Theorem 1, we construct a $tw(G)/w(G_1)$-separator S_1 such that

$$c(S_1) \leq 2\lfloor 2w(G_1)/tw(G)\rfloor r(T_1) \leq 2\lfloor 2w(G_1)/tw(G)\rfloor (d(v_{i_2-1}) - d(v_{i_1}))$$

In this way, for $j = 0, 1, \ldots, p$, we obtain a set of vertices S_j that is $tw(G)/w(G_j)$-separator for G_j, such that

$$c(S_j) \leq 2\lfloor 2w(G_j)/tw(G)\rfloor (d(v_{i_{j+1}-1}) - d(v_{i_j})) \tag{3}$$

Clearly, the union of the vertices in the levels $L(d(v_{i_1})), \ldots, L(d(v_{i_p}))$ and separators $S_0, \ldots, S_p$ forms a t-separator for G. We denote this separator by $S(i_1, \ldots, i_p)$. The cost of $S(i_1, \ldots, i_p)$ is estimated by

$$c(S(i_1, \ldots, i_p)) = c\left(\cup_{j=1}^{p} L(d(v_{i_j}))\right) + c\left(\cup_{j=1}^{p} S_j\right) \leq c\left(\cup_{j=1}^{p} L(d(v_{i_j}))\right) + \sum_{j=0}^{p} c(S_j)$$

$$\leq c\left(\cup_{j=1}^{p} L\left(d\left(v_{i_j}\right)\right)\right) + 2\sum_{j=0}^{p}\left\lfloor \frac{2w(G_j)}{tw(G)} \right\rfloor \left(d\left(v_{i_{j+1}-1}\right) - d\left(v_{i_j}\right)\right) \tag{4}$$

where $i_0 = 0$ and $i_{p+1} = n + 1$.

The estimate (4) suggests a method for choosing the sequence of indices $i_1, \ldots, i_p$, so that the cost of the resulting separator $S(i_1, \ldots, i_p)$ can be estimated in terms of the input parameters t and $\sigma(G)$. We call this method *equidistant choice of levels* and we describe it in the following:

We set $h = \sqrt{t\sigma(G)/8}$ and define a set of equidistant points $y_j = jh$, $j = 0, \ldots, p'$ in the interval $[0, d(v_n)]$, where $p' = \lfloor d(v_n)/h \rfloor$. For $j = 0, \ldots, p'$, let $i(y_j)$ be the smallest index, such that $y_j \leq d(v_{i(y_j)})$. For $j = 1, \ldots, p'$ we define L_j to be the level of minimum cost among the levels $L(d(v_{i(y_{j-1})})), \ldots, L(d(v_{i(y_j)})$, i.e.,

$$c(L_j) = \min_{i(y_{j-1}) \leq i \leq i(y_j)} c(L(d(v_i)))$$

Note that there may be repetitions in the sequence $L_1, \ldots, L_{p'}$. We remove repetitions in that sequence and denote the resulting subsequence by $\mathcal{L} = \{L_1^*, \ldots, L_p^*\}$. Each of the levels L_j^* for $j = 1, \ldots, p$ corresponds to unique index i_j such that $L_j^* = L(d(v_{i_j})$. From our construction it follows that $i_1 < \ldots < i_p$.

In the next theorem we prove that the upper bound (1) on the cost of the separators holds when the equidistant choice of levels is used in the first phase of the algorithm (see Step 3, Table I).

THEOREM 2. *Let G be an n-vertex planar graph with nonnegative weights and costs associated to its vertices. For any $t \in (0, 1)$ there exists a t-separator S_t such that $c(S_t) \leq 4\sqrt{2\sigma(G)/t}$. Such a separator S_t can be found in $O(n)$ time in addition to the time for computing a SSSP tree in G.*

PROOF. Consider the t-separator $S(i_1, \ldots, i_p)$ constructed as described above, where indices $i_1, \ldots, i_p$ are chosen using the equidistant method. Recall that $h = \sqrt{t\sigma(G)/8}$, $p' = \lfloor d(v_n)/h \rfloor$ and $y_j = jh$ for $j = 0, \ldots, p'$. By (4), for the cost of this separator we have

$$c(S(i_1, \ldots, i_p)) \leq c\left(\cup_{j=1}^{p} L(d(v_{i_j}))\right) + 2\sum_{j=0}^{p}\left\lfloor \frac{2w(G_j)}{tw(G)} \right\rfloor (d(v_{i_{j+1}-1}) - d(v_{i_j}))$$

We separately estimate the two terms on the right side of this inequality. To estimate the first term, we consider points $x_1, \ldots, x_{p'}$, such that $x_j \in [y_{j-1}, y_j]$ and $c(L(x_j)) = \min_{x \in [y_{j-1}, y_j]} c(L(x))$. From our definitions and Lemma 1, it follows that for $j = 1, \ldots, p'$ we have $L(x_j) = L_j$. Using this and Lemma 3, the first term can be estimated as follows:

$$\begin{aligned} c\left(\cup_{j=1}^{p} L(d(v_{i_j}))\right) &\leq \sum_{j=1}^{p'} c(L(x_j)) \leq \sum_{j=1}^{p'} \frac{1}{h} \int_{y_{j-1}}^{y_j} c(L(x))dx \\ &\leq \frac{1}{h} \int_{0}^{d(v_n)} c(L(x))dx = \sigma(G)/h \end{aligned} \tag{5}$$

To estimate the second term, we observe that $d(v_{i_{j+1}-1}) - d(v_{i_j}) \leq 2h$ for $j = 0, \ldots, p$, where it is assumed $i_0 = 0$ and $i_{p+1} = n + 1$. Thus,

$$2\sum_{j=0}^{p} \left\lfloor \frac{2w(G_j)}{tw(G)} \right\rfloor (d(v_{i_{j+1}-1}) - d(v_{i_j})) \leq 4h \sum_{j=0}^{p} \left\lfloor \frac{2w(G_j)}{tw(G)} \right\rfloor \leq 8h/t \qquad (6)$$

To obtain the theorem, we sum (5) and (6) and substitute h with its value $\sqrt{t\sigma(G)/8}$. The time bound follows straightforward from our discussion. □

2.3 Applications of Theorem 2

Theorem 2, in essence, captures many of the existing results on separators in planar graphs. By appropriately setting the cost and weights of vertices in the given planar graph, we can obtain variety of separator results. This includes the vertex separator of Lipton and Tarjan [1979], edge separators [Diks et al. 1993] and bisectors [Djidjev 2000], to name a few. Below we show how Theorem 2 can be applied to construct an edge t-separator in a weighted planar graph. Recall that a set of edges is called an *edge t-separator* of G, if their removal leaves no component of weight greater than $tw(G)$.

In Diks et al. [1993] it has been shown that any n-vertex planar graph of maximum vertex degree D has a (2/3)-edge separator of cardinality $O(\sqrt{Dn})$. Other versions of (2/3)-edge separator, where edges have cost, have been highlighted in Djidjev [2000]. Here we prove the following theorem:

THEOREM 3. *Let G be an n-vertex planar graph with nonnegative weights on vertices. Let t be a real number in $(0, 1)$ and let no vertices of G have weight exceeding $tw(G)$. There exists an edge t-separator of G consisting of, at most, $4\sqrt{2\Delta(G)/t}$ edges, where $\Delta(G) = \sum_{v \in V}(deg(v))^2$ and $deg(v)$ is the degree of v. Such a separator can be found in $O(n)$ time, in addition to computing SSSP tree in G.*

PROOF. First we set the cost of each vertex as its degree. Then we apply Theorem 2 and obtain a vertex t-separator VS_t, whose cost is at most $4\sqrt{2\Delta(G)/t}$. Since the cost of a vertex is its degree, there are, at most, $4\sqrt{2\Delta(G)/t}$ edges incident to vertices in VS_t. Next, for each vertex $v \in VS_t$, we consider components of $G \setminus VS_t$ adjacent to v and if there is a component whose weight plus the weight of v is less than $tw(G)$, we absorb v in that component. If there is no such component, we make v to constitute its own component. Finally, we form an edge t-separator ES_t by inserting in it all edges incident to vertices in VS_t that join different components. Clearly, ES_t satisfies the requirements of the theorem. □

Notice that in planar graphs $\sum_{v \in V} deg^2(v) \leq 6Dn$, where D is the maximum degree in G. Hence, Theorem 3 with $t = 2/3$ implies the previous result of Diks et al. [1993]. The result of Theorem 3 can be extended to graphs with nonnegative costs on their edges in a straightforward manner.

3. TECHNIQUE FOR REDUCING THE COST OF THE SEPARATORS

In this section, we describe a technique that optimizes the choice of levels to be inserted in the separator during the first phase of the algorithm. Recall that in the first phase we select a sequence of indices $i_1, \ldots, i_p$ and then slice the graph by removing the vertices in the levels $L(d(v_{i_1})), \ldots, L(d(v_{i_p}))$. Let $\Gamma(i_1, \ldots, i_p)$ denote the right-hand side of Eq. (4), i.e.,

$$\Gamma(i_1, \ldots, i_p) = c\left(\cup_{j=1}^{p} L(d(v_{i_j}))\right) + 2\sum_{j=0}^{p}\left\lfloor \frac{2w(G_j)}{tw(G)} \right\rfloor (d(v_{i_{j+1}-1}) - d(v_{i_j}))$$

We have shown that the cost of the separator $S(i_1, \ldots, i_p)$, resulting after the choice of levels corresponding to the sequence $i_1, \ldots, i_p$, is bounded by $\Gamma(i_1, \ldots, i_p)$. To prove our main result in the previous section (Theorem 2), we have chosen the indices $i_1, \ldots, i_p$ by a method we called equidistant choice of levels. We then proved that for such choice of indices the value of Γ does not exceed $4\sqrt{2\sigma(G)/t}$. The selection of indices, when using the equidistant choice of levels, is done in linear time. The cost of the resulting separators is asymptotically optimal and, hence, such a choice is satisfactory from a theoretical point of view.

In practice, we may apply a technique which further reduces the cost of the produced separators. That is, we find a sequence of indices for which the value of Γ is minimum and then use the corresponding levels to partition the graph in the first phase of the algorithm. We name this technique *optimized choice of levels* and use it as an alternative to equidistant choice of levels in Step 3 (Table I). Indeed, the cost of the t-separators resulting when optimized choice of levels is used, satisfies (1). We are not able to argue that optimized choice is superior to equidistant choice in all cases, since we are minimizing a function that is an upper bound on the cost of the resulting separator, but not the cost itself. Nevertheless, our experiments (see later Figure 4) show that this technique significantly improves the cost of the separators and is computationally efficient. In the remainder of this section we describe the optimized choice of levels technique, finding a sequence of indices $i_1, \ldots, i_p$ such that the value $\Gamma(i_1, \ldots, i_p)$ is minimum.

3.1 Reformulation of the Problem for Minimizing Γ as a Shortest Path Problem

Notice that the number of different levels is not always n. If two vertices have the same distance to the root ρ, then their levels coincide. Assume that there are $n' \leq n$ vertices with distinct distances from ρ. We assume that vertices with the same distance are grouped into groups, that for the sake of simplicity, are denoted by $v_1, \ldots, v_{n'}$ so that $0 = d(\rho) < d(v_1) < \ldots < d(v_{n'}) = r(T)$. Thus, we have n' distinct levels $L(d(v_1)), \ldots, L(d(v_{n'}))$ to choose from. We relate to these levels an oriented graph $H = (V(H), E(H))$ with $n'+1$ vertices defined by $V(H) = \{0, 1, \ldots, n'+1\}$ and $(n'+2)(n'+3)/2$ edges $E(H) = \{(i, j) : 0 \leq i < j \leq n'+1\}$. Next, we assign costs to the edges of H so that for any sequence of indices $0 < i_1 < \ldots < i_p < n'+1$ the cost of the path in H from 0 to $n'+1$ through vertices $i_1, \ldots, i_p$ is exactly $\Gamma(i_1, \ldots, i_p)$. This is achieved by the

following assignment of costs to the edges of H

$$cost(i, j) = c(L(d(v_j))\backslash L(d(v_i))) + 2\lfloor 2w(G_{i,j})/tw(G)\rfloor(d(v_{j-1}) - d(v_i)) \quad (7)$$

where $L(v_0) = L(v_{n'+1}) = \emptyset$ and $w(G_{i,j})$ is the total weight of the vertices of G with distances between $d(v_i)$ and $d(v_j)$, but not in $L(d(v_i))$. Now the problem of finding a sequence of indices minimizing the value of Γ is equivalent to finding the shortest path between 0 and $n' + 1$ in the graph H.

3.2 Efficient Computation of Shortest Paths in H

Dijkstra's shortest-path algorithm can be used to find the shortest path between 0 and $n' + 1$ in H and, hence, a sequence of indices for which Γ is minimum. The problem with direct application of Dijkstra's algorithm on H, however, is that it takes $\theta(n'^2)$ time, which can be $\Omega(n^2)$. Below we describe how the special structure of the graph H can be exploited so that Dijkstra's algorithm will not need to explore all the edges of H and, subsequently, will find a shortest path from 0 to $n' + 1$ more efficiently.

The classical Dijkstra's algorithm maintains a priority queue Q storing the vertices of the graph, where the key of a vertex i in Q is the cost of a shortest path from 0 to i found so far. In each iteration, the algorithm extracts the vertex with a minimum key from Q, which we refer to as a *current vertex*, and then *relaxes* the edges whose source is the current vertex. Next, we describe our efficient implementation of Dijkstra's algorithm on H.

Notice that we can not compute in advance the cost of the edges of H, since this will take $\Omega(n'^2)$ time. However, we can compute in a preprocessing step, the costs and weights of the sets of vertices $L(d(v_{i+1}))\backslash L(d(v_i))$ for $i = 0, \ldots, n' - 1$ in $O(n)$ time. Then, given the cost of an edge $(i, j) \in E(H)$, the cost of the edges $(i, j - 1)$ or $(i, j + 1)$ is easily computed in $O(1)$ time. This leads to the following rule for our implementation.

RULE 1. *Edges originating at the current vertex have to be relaxed following their consecutive order* (*increasing or decreasing*).

For a vertex $i \in V(H)$ denote by $dist(i)$ the cost of the shortest path between 0 and i. We know that the cost of an edge (i, j) in H is the sum of two terms (see Eq. 7). We call them the *cost part* and the *weight part* and denote them by $cost_c(i, j)$ and $cost_w(i, j)$, respectively. Observe that the weight part of an edge is an increasing function with respect to the target vertex. Hence, we have

PROPERTY 1. *If $i < j$ and $dist(i) \geq dist(j)$, then there is a shortest path from 0 to $n' + 1$ that does not include i.*

PROOF. Let $P^*(0, j)$ be a shortest path from 0 to j. Thus, we have $dist(i) \geq dist(j) = cost(P^*)$ and, thus, $i \notin P^*$. Consider now an arbitrary path $P = \{0, \ldots, i, \ldots, n' + 1\}$ from 0 to $n' + 1$ through i. Let i' be the last vertex before j in P and k be the vertex immediately after i' in P. We define a path P_1 from 0 to $n' + 1$ consisting of the shortest path $P^*(0, j)$ followed by the edge (j, k) and then followed by the portion $P(k, n' + 1)$, that is, $P_1 = \{P^*(0, j), k, P(k, n' + 1)\}$.

Next we show that $cost(P) \geq cost(P_1)$. We have

$$\begin{aligned} cost(P) - cost(P_1) &= cost(P(0,i')) + cost(i',k) - dist(j) - cost(j,k)) \\ &\geq dist(i) + cost(i',k) - dist(j) - cost(j,k) \\ &= dist(i) + cost_c(i',k) + cost_w(i',k) - dist(j) - cost_c(j,k) - cost_w(j,k) \\ &= dist(i) + cost_w(i',k) - dist(j) - cost_w(j,k) \geq 0 \end{aligned}$$

since $cost_c(j,k) = cost_c(i',k)$ and $cost_w(j,k) \leq cost_w(i',k)$. Thus, $cost(P) \geq cost(P_1)$ and the claim follows from the fact that $i \notin P_1$. □

Property 1 justifies the next rule of our implementation.

RULE 2. *The indices of the consecutive current vertices must increase.*

That is, if extract_min(Q) produces a vertex with smaller index than the previous current vertex, then no relaxation is done. Furthermore, from Theorem 2 it follows that $dist(n'+1) \leq B$, where $B = 4\sqrt{2\sigma(G)/t}$. Therefore, we do not need to consider paths longer than B during our implementation of Dijkstra's algorithm.

RULE 3. *If i is the current vertex and $dist(i) + cost_w(i,j) > B$, then edges (i,j') with $j' > j$ are not relaxed.*

Applying this rule, we do not relax edges "after" an edge (i,j) for which $dist(i) + cost_w(i,j) > B$. Next, property shows that we can also omit the relaxation of all edges "before" a certain edge.

PROPERTY 2. *If $i' < i < j$ and $dist(i') + cost(i',j) \leq dist(i) + cost(i,j)$, then $dist(i') + cost(i',j') \leq dist(i) + cost(i,j')$ for any $i < j' < j$.*

PROOF. From the conditions and the definition of the cost of the edges we have

$$\begin{aligned} dist(i) - dist(i') \;&\geq\; cost(i',j) - cost(i,j) = cost_w(i',j) - cost_w(i,j) \\ &= cost_w(i',j') - cost_w(i,j') = cost(i',j') - cost(i,j') \end{aligned}$$

which implies the property. □

RULE 4. *If i is the current vertex and the relaxation of an edge (i,j) does not result in a smaller distance to j, then edges (i,j') with $j' < j$ are not relaxed.*

The above four rules suggest the following implementation of Dijkstra's algorithm on H. In a preprocessing step, for $i = 0, \ldots, n'$ we compute the smallest index $jump(i)$ such that $w(G_{i,jump(i)}) > tw(G)$. Consider an iteration of Dijkstra's algorithm running on H. Assume that i is the current vertex and edges originating at i are to be relaxed. We relax the edge $(i, jump(i))$ first. The edges before $jump(i)$ are then relaxed in decreasing order until the first unsuccessful relaxation is encountered (Rule 4). Next, edges after $(i, jump(i))$ are relaxed in their increasing order until Rule 3 applies.

Our experiments show that this implementation results in a very efficient procedure and timings suggest that the running time depends linearly on n.

Thus far, we do not have a rigorous proof of this bound, whereas some of the special cases are characterized in the following lemma.

LEMMA 4. *If the cost of vertices are positive integers, then a collection of indices minimizing* Γ *can be found in* $O(\min(n^2, \sigma(G)/t))$ *time. In the uniform case, where all vertices have the same cost, the time bound is* $O(\min(n^2, n/t))$.

PROOF. Presented in the Appendix. □

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

In this section, we provide details on the implementation of our graph-partitioning algorithms based on vertex and edge t-separators and discuss experimental results. The section is organized as follows. We start by discussing some design decisions that we made and a few implementation details. We then describe the objectives of our experiments, the experimental environment, and the test graphs. The section concludes with description of the results and our observations and analysis.

4.1 Design Issues and Implementation Details

For implementing our algorithms, we made extensive use of the data types and basic algorithms provided by LEDA [Mehlhorn and Näher 1999], a C++ library of combinatorial and geometric data types and algorithms. Our choice was motivated by the richness of data structures and algorithms it offers, as well as the fact that it is widely known and used by researchers working in the field of theoretical computer science and experimental algorithmics. The data type *graph* of LEDA is of particular importance to us, since it allows us to associate costs and weights with vertices or edges of the graph in a natural way and has a large collection of implemented basic graph algorithms. During the experiments we also made use of the convenient graph visualization tools offered in LEDA.

Next, we briefly discuss some implementation details. We refer to the outline of the algorithm as presented in Table I. In Step 1, we add a dummy vertex ρ and connect it to an arbitrary chosen vertex in G. Later in Step 2, the vertex ρ is used as root of the SSSP tree T computed with respect to the cost distance in G. The choice of the vertex to which ρ is connected completely determines T and, consequently, the set of levels among which we do our selection in Step 3. Thus, different choices of the vertex (or vertices) to be connected to ρ have strong influence on the quality of the resulting partitions. Although our experiments did not suggest a particular choice to be superior in all cases, this feature of the algorithm can be used to generate a number of alternative partitions and to pick the best one.

We implement Step 2 and Step 5.2 using either Dijkstra's SSSP algorithm or simply breadth-first search in the case of uniform costs. The priority queue in Dijkstra's algorithm is implemented using binary heap. The implementation of the optimized choice of levels in Step 3 follows the description in Section 3 (see Rules 1–4). We also implemented a special priority queue for the case of integer costs, as described in the proof of Lemma 4 (see Appendix). As we already

mentioned, our experiments suggest linear dependence on n of the running time of Step 3.

Step 5.1 is implemented by the aid of the functions provided in LEDA, including construction of planar representations and triangulation of planar graphs. The implementation of Step 5.3 on a "heavy" component G_j involves construction of a data structure called *separation tree*. The separation tree is a weighted binary tree whose edges correspond to fundamental cycles defined with respect to the SSSP tree T_j of G_j. We select a set of edges whose removal partitions the separation tree into components of weight not exceeding $tw(G)$. The corresponding set of fundamental cycles partitions G_j into components whose weight does exceed $tw(G)$. Further details about the separation tree data structure, as well as theoretical analysis, can be found in Aleksandrov and Djidjev [1996].

After the completion of the first two phases, the set of vertices S_t is already a t-separator. In most of the cases, the quality of the partitions obtained at that stage can be further improved, which is done in the third phase. Efficient implementation of the steps in that phase is based on a data structure we call *supergraph* and is denoted by SG. The nodes of SG correspond to the connected components of $G \setminus S_t$. Two nodes in SG are connected by an edge if their corresponding components are neighbors, i.e., have a nonempty common boundary as defined in Section 2.1. (In the case of edge separators, the common boundary of two neighboring components is the set of edges joining these components.)

The nodes of SG are assigned weights that equals the total weight of their corresponding components. The edges of SG are assigned weights and costs. The weight and cost of an edge of SG are the total weight and cost, respectively, of its corresponding common boundary. In addition, for each of the vertices in the separator S_t, we compute and maintain information about its adjacencies to the nodes of SG (components of $G \setminus S_t$). Using the supergraph data structure as an aid, we implement easily and efficiently Steps 6, 7, and 8, as described in Section 2.1.

During our initial experiments, we have encountered a number of cases in which single vertices or very small components could not be absorbed into their neighboring components because of slight violation of the weight condition. The latter resulted into low-quality partitions containing isolated vertices or components of negligible weight. To avoid this undesirable effect, we implemented the third phase with relaxed weight condition. That is, we chose a parameter $\alpha > 1$ and execute Steps 6, 7, and 8, requiring that no component of weight greater than $\alpha tw(G)$ appears. Experimental results presented below are obtained with relaxation parameter $\alpha = 1.5$ for vertex separators and $\alpha = 2$ for edge separators.

4.2 Experiments

In this section we will discuss the goal of our experiments, the parameters by which we measure the performance, the implementation platform, and the test graphs.

Table II. Graphs for Which We Present Detailed Experimental Results in this Section

Graph	$\|V\|$	$\|E\|$	Type
trapezoid	819	2286	Irregular
tapir	1024	2846	Irregular
airfoil1	4253	12289	FEM
america	5000	14787	TIN
whitaker	9800	28989	FEM
big	15606	45878	FEM
crack_dual	20141	30043	FEM
tin50000	25440	75683	TIN
tin100000	50624	150973	TIN

The first goal of our experiments is to test our implementations on a collection of different types of planar graphs and compare the quality of the partitions with the theoretically established bounds. The second goal is to test the efficiency of the optimized choice of levels method described in Section 3.

Recall that we described two different methods to select levels in Phase I of the vertex separator algorithm. These two methods were named equidistant choice of levels and optimized choice of levels and described before Theorem 2 and in Section 3, respectively. Below, we refer to the usage of optimized choice of levels as *cost-reducing* technique.

As above, we denote the resulting vertex separators by S_t and edge separators by ES_t. We collect information about the following parameters:

- The cost of the vertex separators $c(S_t)$ or the size of the edge separators $|ES_t|$.
- The ratio $c(S_t)/\sqrt{\sigma(G)/t}$, in the case of vertex separators and the ratio $|ES_t|/\sqrt{\Delta(G)/t}$, in the case of edge separators.
- The number of connected components of the partition. To compute the number of regions, we sort the components by weight and then pack components together to form regions. We ensure that the total weight of a region is, at most, $tw(G)$, but a region may consist of a collection of one or more (disjoint) components.
- The largest and the smallest components in terms of their weights.
- The ratio between the maximum and the minimum weight of the resulting components, called as the *balance*.
- The total CPU time of Phases I and II and the CPU time of Phase III.

We ran our experiments on Sun Ultra 10 Microsystems with 440 MHz clock and 512 MB memory running Solaris 8 operating system. Our algorithms are implemented in C++ using LEDA version 4.1. The C++ compiler is GNU g++ version 2.95.2. For our experiments, we used well-known planar graphs available on the web,[2] including planar finite element meshes (FEM), irregular planar graphs, and triangular irregular networks (TIN). The choice of these graphs was motivated by their availability, sizes, variety, and whether other separator algorithms have used them. Table II lists a small sample of planar

[2]From http://www.cs.uni-paderborn.de/cs/ag-monien/RESEARCH/PART/graphs.html.

graphs used in our experiments. TIN graphs were constructed from cropped portions of actual digital elevation model data. Once cropped, the obtained grid was then simplified through the removal of vertices, thereby obtaining a terrain. The graph *tapir* is a test case from a two-dimensional mesh generation algorithm of Bern et al. [1995] that produces triangles with sharp angles, but no obtuse angles.

The input to our program consists of the following: the input graph G in the LEDA graph format, the value of t, and the costs and the weights associated with the vertices of G. We have also implemented two converters: one from TIN graphs to the LEDA format and the other from FEM graphs to the LEDA format.

4.3 Experimental Results

In this section we present and discuss the results obtained from the execution of the following sets of experiments:

E1 Partitioning of graphs with unit weights and unit costs by vertex t-separators. The results are presented in Table III.

E2 Partitioning of graphs with random integer weights and random integer costs by vertex t-separators. The results are presented in Table IV.

E3 Partitioning of graphs with unit weights and costs by edge t-separators. The results are presented in Table V.

E4 Evaluation of the efficiency of the cost-reducing technique (Table VI and, later, Figure 4).

Consider, first, the results from the first set of experiments where we test partitioning of graphs with unit weights and unit costs by vertex t-separators (see Table III). There are several conclusions that can be drawn:

- *E1-Separator*: Costs of the obtained separators are significantly smaller than the upper bound derived in Theorem 2. This is illustrated by the ratio between the cost of the separator $c(S_t)$ and $\sqrt{\sigma(G)/t}$ (in this case $c(S_t) = |S_t|$ and $\sqrt{\sigma(G)/t} = \sqrt{n/t}$). By Theorem 2 this ratio is, at most, $4\sqrt{2} \approx 5.657$, whereas we never get the ratio bigger than 2 in our experiments.
- *E1-Balance*: In most of the cases, the ratio between the sizes of the largest and the smallest component is below 2. Exceptions occur when the graph is small and $1/t$ is big, or when there is a small piece that could not be packed to any neighbor component. Generally speaking, our algorithm produces balanced partitions.
- *E1-CPU Time*: Overall running times are small and do not change significantly by altering the value of t. More precisely, the ratios between the total CPU time in milliseconds and the size of the input graph are in the range (0.02, 0.05). Phase III takes only a small fraction of time compared to the time spent in Phase I plus Phase II.

Similar observations can be made from our set of experiments E2. The experiments in this set has been done with random integer weights and costs in

Table III. Experimental Results for Partitioning of Graphs with Unit Weights and Costs by Vertex t-Separators[a]

Graph	$\frac{1}{t}$	$\frac{\lvert S_t\rvert}{\sqrt{n/t}}$	$\lvert S_t\rvert$	# of Components	# of Regions	Min Weight	Max Weight	Time of Ph. I + II	Time of Phase III	Balance
trapezoid	2	0.815	33	2	2	387	399	0.00	0.00	1.03
$\lvert V\rvert = 819$	4	0.716	41	4	4	189	201	0.07	0.00	1.06
$\lvert E\rvert = 2286$	32	1.310	212	27	26	19	32	0.07	0.00	1.68
	128	1.192	386	86	82	2	8	0.06	0.01	4.00
tapir	2	0.376	17	2	2	367	640	0.10	0.00	1.74
$\lvert V\rvert = 1024$	4	0.313	20	4	4	162	301	0.11	0.01	1.86
$\lvert E\rvert = 2846$	32	1.144	207	33	28	14	40	0.09	0.01	2.86
	128	1.058	383	94	87	3	11	0.09	0.01	3.67
airfoil1	2	0.434	40	2	2	1791	2422	0.53	0.03	1.35
$\lvert V\rvert = 4253$	4	0.889	116	5	4	897	1314	0.58	0.03	1.46
$\lvert E\rvert = 12289$	32	1.431	528	31	30	107	163	0.48	0.03	1.52
	128	1.511	1115	110	108	18	42	0.41	0.03	2.33
america	2	0.88	88	3	2	2122	2790	0.62	0.02	1.31
$\lvert V\rvert = 5000$	4	1.245	176	4	4	1180	1236	0.48	0.04	1.05
$\lvert E\rvert = 14787$	32	1.590	636	32	30	113	214	0.52	0.05	1.89
	128	1.574	1259	114	113	21	48	0.47	0.05	2.29
whitaker	2	0.550	77	2	2	4822	4901	0.06	0.08	1.02
$\lvert V\rvert = 9800$	4	1.096	217	4	4	2139	2632	0.67	0.07	1.23
$\lvert E\rvert = 28989$	32	1.627	911	31	31	239	336	1.14	0.08	1.41
	128	1.682	1884	115	114	54	104	1.09	0.09	1.93
big	2	0.951	168	2	2	6001	9437	2.49	0.14	1.57
$\lvert V\rvert = 15606$	4	0.768	192	4	4	3820	3901	2.61	0.11	1.02
$\lvert E\rvert = 45878$	32	1.493	1055	32	31	401	679	2.17	0.15	1.69
	128	1.664	2352	121	120	71	150	1.91	0.16	2.11
crack_dual	2	0.568	114	2	2	9996	10031	0.10	0.14	1.00
$\lvert V\rvert = 20141$	4	0.874	248	4	4	4912	5036	1.13	0.13	1.03
$\lvert E\rvert = 30043$	32	1.185	951	35	31	525	938	1.72	0.15	1.79
	128	1.231	1977	136	121	122	189	1.77	0.17	1.55
tin50000	2	0.705	159	2	2	12639	12642	4.29	0.23	1.00
$\lvert V\rvert = 25440$	4	0.997	318	4	4	6241	6359	3.17	0.21	1.02
$\lvert E\rvert = 75683$	32	1.628	1469	32	31	715	1103	3.88	0.23	1.54
	128	1.735	3130	122	122	144	208	3.44	0.24	1.44
tin100000	2	0.710	226	2	2	24867	25531	9.47	0.46	1.03
$\lvert V\rvert = 50624$	4	0.998	449	4	4	12432	12656	6.68	0.46	1.02
$\lvert E\rvert = 150973$	32	1.637	2083	32	32	1025	1580	8.67	0.46	1.54
	128	1.766	4496	124	124	256	437	8.47	0.48	1.71

[a] The cost-reducing technique is used.

the range $(1, 100)$ (see, Table IV and Figure 2). The results show that the ratios between $c(S_t)$ and $\sqrt{\sigma(G)/t}$ again are far below the bound $4\sqrt{2}$ from Theorem 2 and are consistently smaller than the ratios obtained during the experiments in E1. The balance slightly worsens compared to E1, but still remains below 2, in most cases. The total CPU times are small although larger than those in E1 by about 30%. More precisely, the ratios between the total CPU time in milliseconds and the size of the input graph are in the range $(0.02, 0.08)$

Next, we consider the results of the experiments in the set E3, where we partition graphs with unit weights and costs using edge separators. Recall that

Table IV. Experimental Results for Partitioning of Graphs with Random Integer Weights and Costs in the Range (1, 100) by Vertex t-Separators[a]

Graph	$\frac{1}{t}$	$\frac{\lvert S_t\rvert}{\sqrt{n/t}}$	$\lvert S_t\rvert$	# of Components	# of Regions	Time of Ph. I + II	Time of Phase III	Balance
trapezoid	2	0.192	15	2	2	0.09	0.00	1.04
$\lvert V\rvert = 819$	4	0.491	63	5	4	0.07	0.00	1.69
$\lvert E\rvert = 2286$	32	1.07	224	26	25	0.12	0.01	2.13
	128	0.972	367	79	75	0.13	0.02	3.02
tapir	2	0.282	18	2	2	0.14	0.00	1.75
$\lvert V\rvert = 1024$	4	0.269	35	4	4	0.12	0.01	1.08
$\lvert E\rvert = 2846$	32	0.839	197	31	27	0.12	0.00	2.77
	128	0.904	363	117	100	0.12	0.01	9.33
airfoil1	2	0.487	97	3	2	0.60	0.04	1.01
$\lvert V\rvert = 4253$	4	0.427	115	4	4	0.65	0.04	1.24
$\lvert E\rvert = 12289$	32	1.088	591	33	32	0.69	0.03	1.95
	128	1.325	1246	107	105	0.71	0.04	2.17
america	2	0.495	113	3	2	0.90	0.04	1.29
$\lvert V\rvert = 5000$	4	0.400	120	4	4	0.75	0.03	1.68
$\lvert E\rvert = 14787$	32	1.101	726	32	30	0.85	0.05	2.15
	128	1.211	1317	106	103	0.85	0.06	2.14
whitaker	2	0.242	78	2	2	1.72	0.09	1.01
$\lvert V\rvert = 9800$	4	0.629	290	5	4	1.94	0.09	1.37
$\lvert E\rvert = 28989$	32	1.398	1166	34	33	1.78	0.09	1.85
	128	1.445	2286	115	114	1.83	0.10	2.31
big	2	0.504	201	2	2	4.07	0.14	1.49
$\lvert V\rvert = 15606$	4	0.504	288	4	4	4.16	0.14	1.12
$\lvert E\rvert = 45878$	32	0.906	1148	35	34	4.10	0.14	1.75
	128	1.184	2775	127	125	4.08	0.15	2.87
crack_dual	2	0.200	95	2	2	2.85	0.14	1.07
$\lvert V\rvert = 20141$	4	0.561	237	4	4	2.67	0.15	1.19
$\lvert E\rvert = 30043$	32	0.976	997	34	33	3.46	0.16	1.96
	128	1.016	2037	138	130	3.83	0.18	2.55
tin50000	2	0.310	170	2	2	4.76	0.23	1.69
$\lvert V\rvert = 25440$	4	0.752	602	5	4	5.02	0.23	1.37
$\lvert E\rvert = 75683$	32	1.292	2074	34	33	6.02	0.23	1.96
	128	1.469	4219	131	130	6.05	0.25	2.22
tin100000	2	0.302	246	2	2	9.77	0.48	1.11
$\lvert V\rvert = 50624$	4	0.563	638	4	4	9.91	0.49	1.31
$\lvert E\rvert = 150973$	32	1.324	3258	36	35	13.94	0.47	1.79
	128	1.479	6345	132	130	14.05	0.49	2.38

[a]The cost-reducing technique is used.

we construct edge t-separators applying our vertex t-separator algorithm on graphs with vertex costs. The costs of vertices in the input graph are set to equal their degrees. In this case, $\sigma(G)$ is denoted by $\Delta(G) = \sum_{v \in V} deg(v)^2$. The results are shown in Table V. Two illustrations of edge separators are shown in Figure 3. We make the following observations:

- *E3-Separator*: The ratio between the size of the edge separators $\lvert ES_t\rvert$ and $\sqrt{\Delta(G)/t}$ is significantly smaller than the constant $4\sqrt{2} \approx 5.657$ predicted by Theorem 3. We never get this ratio greater than 1 in our experiments.

Table V. Experimental Results for Partitioning of Graphs with Unit Weights and Costs by Edge t-Separators[a]

Graph	$\frac{1}{t}$	$\frac{\lvert S_t\rvert}{\sqrt{n/t}}$	$\lvert S_t\rvert$	# of Components	# of Regions	Min Weight	Max Weight	Time of Ph. I + II	Time of Phase III	Balance
trapezoid	2	0.122	28	2	2	404	415	0.10	0.00	1.03
$\lvert V\rvert = 819$	4	0.338	110	4	4	137	257	0.10	0.00	1.88
$\lvert E\rvert = 2286$	32	0.485	447	32	27	17	51	0.09	0.01	3.00
	128	0.498	918	102	98	1	12	0.09	0.01	12.00
tapir	2	0.073	20	2	2	487	537	0.13	0.00	1.10
$\lvert V\rvert = 1024$	4	0.231	89	4	4	244	263	0.12	0.01	1.08
$\lvert E\rvert = 2846$	32	0.433	472	32	29	25	48	0.10	0.01	1.92
	128	0.489	1066	128	118	1	16	0.10	0.01	16.00
airfoil1	2	0.149	80	2	2	1778	2475	0.67	0.03	1.39
$\lvert V\rvert = 4253$	4	0.295	224	5	4	899	1388	0.60	0.04	1.54
$\lvert E\rvert = 12289$	32	0.495	1064	32	31	111	196	0.61	0.03	1.77
	128	0.547	2350	118	112	17	63	0.55	0.03	3.71
america	2	0.342	205	3	2	2089	2911	0.74	0.04	1.39
$\lvert V\rvert = 5000$	4	0.374	317	4	4	1020	1392	0.55	0.04	1.36
$\lvert E\rvert = 14787$	32	0.587	1410	33	31	96	286	0.67	0.05	2.98
	128	0.573	2751	121	113	27	64	0.62	0.05	2.37
whitaker	2	0.231	192	2	2	4854	4946	0.23	0.09	1.02
$\lvert V\rvert = 9800$	4	0.364	428	4	4	2398	2534	1.52	0.08	1.06
$\lvert E\rvert = 28989$	32	0.566	1883	32	31	221	409	1.43	0.08	1.85
	128	0.600	3992	127	122	44	129	1.31	0.09	2.93
big	2	0.302	315	2	2	6134	9472	3.54	0.15	1.54
$\lvert V\rvert = 15606$	4	0.270	398	4	4	3881	3934	2.73	0.14	1.01
$\lvert E\rvert = 45878$	32	0.514	2144	35	34	308	514	3.19	0.14	1.67
	128	0.581	4851	128	125	65	201	2.92	0.16	3.09
crack_dual	2	0.179	107	2	2	10046	10095	0.46	0.15	1.00
$\lvert V\rvert = 20141$	4	0.281	238	4	4	4798	5224	1.60	0.15	1.09
$\lvert E\rvert = 30043$	32	0.414	992	35	32	484	915	2.40	0.15	1.89
	128	0.455	2180	138	125	89	223	2.36	0.18	2.51
tin50000	2	0.235	316	2	2	12561	12879	4.81	0.23	1.03
$\lvert V\rvert = 25440$	4	0.378	718	4	4	6312	6389	4.30	0.23	1.01
$\lvert E\rvert = 75683$	32	0.597	3207	32	31	660	1062	4.94	0.23	1.61
	128	0.617	6639	128	124	110	307	4.64	0.25	2.79
tin100000	2	0.235	447	2	2	24504	26120	9.65	0.48	1.07
$\lvert V\rvert = 50624$	4	0.379	1017	4	4	12559	12763	9.42	0.49	1.02
$\lvert E\rvert = 150973$	32	0.598	4545	33	32	1245	1794	11.60	0.49	1.44
	128	0.624	9482	128	125	264	669	11.45	0.51	2.53

[a]The cost-reducing technique is used.

- *E3-Balance*: In most of the cases, the balance is below 2. As in the experiments in E1, exceptions occur when the graph is relatively small compared to $1/t$ or when Phase III fails to pack some small components. In general, produced partitions are balanced.
- *E3-CPU time*: The CPU times in this set of experiments are small and did not change substantially when varying t. The ratios between the total CPU time in milliseconds and the size of the input graph are in the range $(0.02, 0.06)$. The CPU time for Phase III is less than 10% of the total CPU time for the execution of Phases I and II.

Table VI. Experimental Results on the Efficiency of the Cost-Reducing Technique[a]

Graph	$1/t$	4	16	32	64	128	256
trapezoid	equidistant cost	80	217	336	453	560	599
$\|V\| = 819$	optimized cost	43	156	235	322	447	533
$\|E\| = 2286$	equidistant time	0.04	0.06	0.05	0.04	0.02	0.01
	optimized time	0.07	0.07	0.07	0.06	0.06	0.05
tapir	equidistant cost	82	202	328	470	595	665
$\|V\| = 1024$	optimized cost	49	158	260	354	489	583
$\|E\| = 2846$	equidistant time	0.07	0.07	0.07	0.05	0.04	0.01
	optimized time	0.11	0.09	0.09	0.09	0.09	0.08
airfoil1	equidistant cost	125	486	744	1084	1563	2196
$\|V\| = 4253$	optimized cost	120	398	585	842	1191	1624
$\|E\| = 12289$	equidistant time	0.55	0.44	0.39	0.36	0.32	0.25
	optimized time	0.58	0.48	0.48	0.46	0.41	0.40
america	equidistant cost	153	536	834	1222	1721	2409
$\|V\| = 5000$	optimized cost	189	459	720	993	1376	1875
$\|E\| = 14787$	equidistant time	0.60	0.50	0.45	0.45	0.38	0.34
	optimized time	0.48	0.53	0.52	0.48	0.47	0.46
whitaker	equidistant cost	360	829	1254	1791	2504	3532
$\|V\| = 9800$	optimized cost	238	632	925	1390	1932	2669
$\|E\| = 28989$	equidistant time	0.87	1.00	0.98	0.99	0.94	0.88
	optimized time	0.67	1.14	1.14	1.12	1.09	1.09
big	equidistant cost	208	845	1406	1998	2938	4190
$\|V\| = 15606$	optimized cost	198	682	1108	1709	2471	3501
$\|E\| = 45878$	equidistant time	2.51	2.09	2.00	1.86	1.66	1.46
	optimized time	2.61	2.58	2.17	2.09	1.91	1.91
crack_dual	equidistant cost	418	1070	1651	2346	3422	4830
$\|V\| = 20141$	optimized cost	259	737	1088	1611	2301	3272
$\|E\| = 30043$	equidistant time	0.87	1.31	1.64	1.74	1.71	1.36
	optimized time	1.13	1.72	1.72	1.73	1.77	1.75
tin50000	equidistant cost	711	1363	1991	2853	4074	5711
$\|V\| = 25440$	optimized cost	319	962	1508	2216	3175	4446
$\|E\| = 75683$	equidistant time	4.04	3.51	3.28	3.18	2.86	2.60
	optimized time	3.17	3.75	3.88	3.58	3.44	3.25
tin100000	equidistant cost	788	1910	2806	4091	5804	8245
$\|V\| = 50624$	optimized cost	450	1344	2118	3119	4582	6447
$\|E\| = 150973$	equidistant time	6.46	7.81	7.95	7.54	6.98	6.37
	optimized time	6.68	8.66	8.67	8.59	8.47	8.12

[a] We denote by *equidistant cost* and by *optimized cost*, respectively, the cost of separators obtained when equidistant choice levels and optimized choice of levels are used. Similarly, *equidistant time* and *optimized time* denote CPU times in seconds for the execution of Phases I and II when equidistant or optimized choice of levels is used.

An observation that applies to all sets of experiments discussed so far is that ratios $c(S_t)/\sqrt{\sigma(G)/t}$, in the case of vertex separators, and $|ES_t|/\sqrt{\Delta(G)/t}$, in the case of edge separators, increase with the value of $1/t$.

Finally, we conduct experiments on the efficiency of the cost-reducing technique. That is, we compare the cost of the resulting vertex separators and the CPU times when equidistant and optimized choice of levels in Phase I is applied. The results of these tests in graphs with unit weights and costs are presented in Table VI and illustrated in Figure 4. We observe the following:

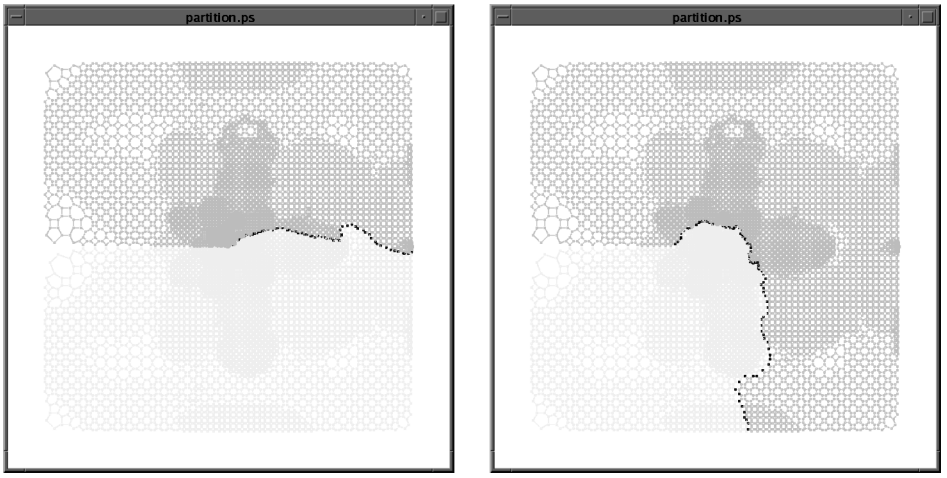

Fig. 2. The first figure illustrates a vertex t-separator of the graph *crack_dual* with unit costs and weights on the vertices. The second figure illustrates a vertex t-separator for the same graph with random vertex costs and weights in the interval (1,100). In both cases $t = 0.5$. Notice that the algorithm detects the crack and constructs the separator as its continuation.

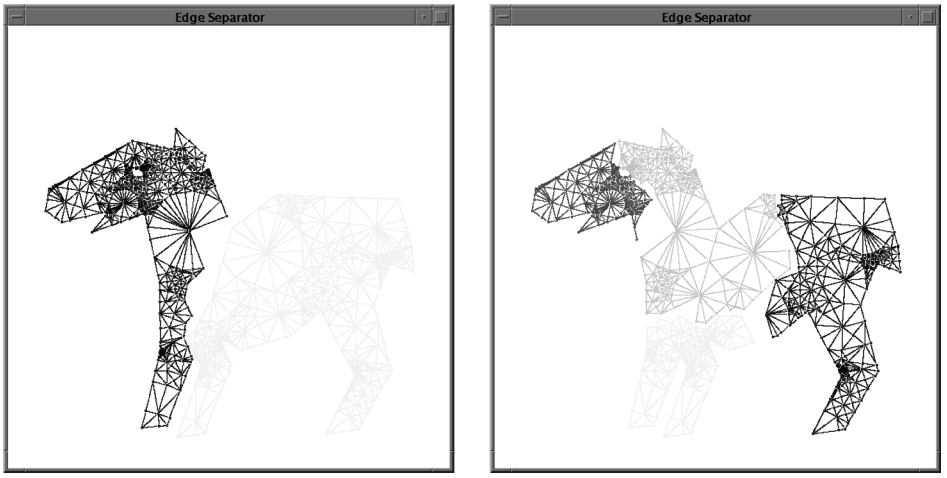

Fig. 3. The left figure illustrates an edge 1/2-separator of the graph *tapir* with unit costs and weights on the vertices. The right figure illustrates an edge 1/4-separator for the same graph.

- *E4-CPU time*: The computation of t-separators using cost-reducing technique takes slightly more time compared to the equidistant choice of levels. These results are consistent when varying the parameter t.
- *E4-Separator*: Generally speaking, the improvement in the cost of the separators when the cost-reducing technique is used is in the range 20–50%, except for the graph *big*, where it is about 15%. The improvement for the graphs *trapezoid* and *tapir*, in the case $1/t = 256$, is about 10%, because the sizes of these graphs are small compared to the value $1/t = 256$, consequently, there is little room for optimization. In the case where $1/t = 4$, in most of the

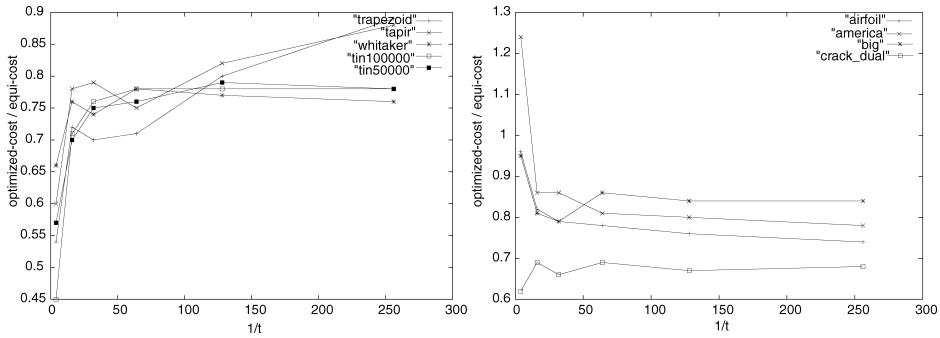

Fig. 4. The ratio between the cost of t-separators computed using optimized choice of levels and equidistant choice of levels. Small ratios correspond to larger improvements.

Table VII. Comparison between Our Edge Separator with Results of Experiments Conducted by Gilbert et al. [1998] with Other Partitioning Algorithms on the Graph *tapir*

Partitions	Spectral (Chaco)	Coordinate Bisection	Default Geometric	Best Geometric	Our Algorithm
2-way ($1/t = 2$)	59	55	37	32	20
128-way ($1/t = 128$)	1278	1387	1239		1066

test graphs we achieved improvements in the range [35%, 55%] except for the graphs *airfoil1* and *big*, where the improvement is about 5%. The improvement for graph *america* is negative, i.e., the cost-reducing technique leads to a larger separator. The reason is that, in this particular case, the equidistant method produces three levels that happen to be the 1/4-separator.

4.4 Comparison with Other Partitioning Algorithms

A natural question that arises is how our algorithms fare with respect to several other algorithms and heuristics proposed for graph partitioning over the last two decades. All of the experimental results reported in the literature are for edge separators. Therefore, we are unable to experimentally compare our vertex separator algorithm with other approaches. Below we discuss how our edge separator algorithm (Theorem 3) derived from the vertex separator algorithm (Theorem 2) performs with respect to others.

Table VII shows a comparison between our edge separator results with that of Gilbert et al. [1998] for the graph *tapir*. In Gilbert et al. [1998], Matlab is used to implement coordinate bisection and Hendrickson and Leland's Chaco package [Hendrickson and Leland 1994] is used to find the spectral bisection. The geometric algorithms are based on the theoretical work of Miller et al. [1997]. A parameter to the geometric algorithm is the number of random trials of great circles to be made. The "default geometric" column reports results for 30 trials and "best geometric" reports the results for 7000 trials.

For the graph that we tested and for which experimental results are reported in the literature, the sizes of the separators computed by our algorithm are better than those obtained by spectral or geometric methods [Gilbert et al. 1998].

The METIS library [Schloegel et al. 2000; Karypis 1998], which is based on the multilevel partitioning technique, computes partitions where the size of the edge separators are slightly better than ours ($\leq$10%) for many graphs where comparable tests can be performed. There are several possible reasons for this. METIS is highly optimized and geared toward edge separators, whereas our algorithms are all based on vertex separators. Moreover, during the multilevel processing in METIS, during each stage the Kernighan–Lin method [Kernighan and Lin 1970] is used, which usually significantly reduces the size of the separators. Simon and Teng [1995] have shown that, in some cases, multilevel partitioning methods using recursive bisection may produce a partition that is far from optimal.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented theoretically optimal algorithms for computing vertex and edge t-separators in a planar graph. We have also demonstrated that the algorithms are easy to implement, efficient, and practical.

Although we provide optimal theoretical results and support them with relevant experiments, there is still room for improvement. In particular, substantial improvements can be made to Phase III. This phase can be viewed as a variant of the bin-packing problem, where combining neighboring partitions is preferred. The general problem is intractable, but, in our case, the instances are relatively small and can be exactly solved. Using an optimal packing and varying the relaxation parameter α should result in partitions of even better quality (see Section 4). Indeed, heuristics like Kernighan–Lin's method can be employed to improve the final partition, as it is done in most of the existing packages. Experiments presented in this paper have been obtained using a straightforward greedy packing procedure and no refinement of the final partitioning is performed.

APPENDIX

Proofs of the Lemmas

Lemma 1. *Let x, y be real in $(0, r(T))$. If the interval (x, y) does not contain any of the distances $d(v_i)$, then $L(x) = L(y)$.*

Proof. According to our definitions, it suffices to show that $E(x) = E(y)$. Let $e = (u, v)$ be an edge in $E(x)$. By the definition, we have $d(u) < x \leq d(v)$. Since (x, y) does not contain $d(v_i)$ for $i = 1, \ldots, n$, then, $d(u) < y \leq d(v)$ holds and thus $e \in E(y)$. Analogously any edge in $E(y)$ must be in $E(x)$ and, therefore, $E(x) = E(y)$. The latter implies $L(x) = L(y)$. □

Lemma 2. *For any real $x \in [0, r(T)]$, the vertices of G are partitioned into three subsets $V_- = \{v : d(v) < x\}$, $L(x)$, and $V_+ = \{v : d(v) > x, v \notin L(x)\}$. No edge in G joins V_- and V_+.*

Proof. Assume for the sake of contradiction, that there exists a tree edge $e = (u, v)$ such that $u \in V_-$ and $v \in V_+$. Then $d(u) < x < d(v)$ and, by definition, $e \in E(x)$. Therefore, $v \in L(x)$, which contradicts the fact that V_+ and $L(x)$ are

disjoint vertex sets. Assume now that there exists a nontree edge $e = (u, v)$, such that $u \in V_-$, and $v \in V_+$. Denote by $P_T(\rho, u) : \rho \rightsquigarrow u$ and $P_T(\rho, v) : \rho \rightsquigarrow v$ the tree paths from ρ to u and to v, respectively. Consider the path from ρ to v through the vertex u, defined by $P(\rho, v) : \rho \overset{P_T(\rho,u)}{\rightsquigarrow} u \overset{e}{\rightarrow} v$. Furthermore, let v' be the predecessor of v on the path $P_T(\rho, v)$, i.e., $P_T(\rho, v) : \rho \rightsquigarrow v' \overset{e'}{\rightarrow} v$. We have shown that $v' \notin V_-$ and thus $d(u) < x \leq d(v')$. For the edge $e' = (v', v)$, $c(e') = c(v) = c(e)$. Therefore, we have:

$$c(P_T(\rho, v)) = d(v') + c(e') > d(u) + c(e) = c(P(\rho, v))$$

which contradicts that $P_T(\rho, v)$ is a path in the SSSP tree. □

LEMMA 3. $\int_0^{r(T)} c(L(x))dx = \sigma(G) = \sum_{v \in V} (c(v))^2$.

PROOF. Let $e = (u(e), v(e))$ be a tree edge. Recall that the distance in G is related to the cost of vertices, and the cost of traveling along a single edge equals the cost of its target vertex. We have $c(e) = c(v(e)) = d(v(e)) - d(u(e))$. Let $\delta_e(x)$ be a characteristic function, defined by $\delta_e(x) = 1$ if $e \in E(x)$, and 0 otherwise. Then

$$\begin{aligned}
\int_0^{r(T)} c(L(x))d(x) &= \int_0^{r(T)} \left(\sum_{e \in E(x)} c(v(e)) \right) dx = \int_0^{r(T)} \left(\sum_{e \in E(x)} c(e) \right) dx \\
&= \int_0^{r(T)} \left(\sum_{e \in E(T)} c(e) \cdot \delta_e(x) \right) dx = \sum_{e \in E(T)} c(e) \int_0^{r(T)} \delta_e(x) dx \\
&= \sum_{e \in E(T)} c(e)(d(v(e)) - d(u(e))) = \sum_{e \in E(T)} (c(e))^2 = \sum_{e \in E(T)} (c(v(e)))^2 \\
&= \sum_{v \in V} (c(v))^2 = \sigma(G)
\end{aligned}$$

□

LEMMA 4. *If the cost of vertices are positive integers, then a collection of indices minimizing* Γ *can be found in* $O(\min(n^2, \sigma(G)/t))$ *time. In the uniform case, where all vertices have the same cost, the time bound is* $O(\min(n^2, n/t))$.

PROOF. Consider first the case when $n^2 < \sigma(G)/t$. In this case, one can apply Dijkstra's algorithm on H, which takes $O(n^2)$ time. In the case when $n^2 > \sigma(G)/t$, we apply our algorithm as described in Section 3. The work done by the algorithm can be separated into three parts. First is the preprocessing, where $jump(i)$ and the costs $cost(i, jump(i))$ are computed in $O(n)$ time. Second is the work to relax edges. Third is the work for Extract-min and Decrease-key operations in the priority queue Q.

Let us evaluate the time taken by the algorithm to relax edges. Since the relaxation of an edge takes $O(1)$ time (Decrease_key time is not counted here), we will estimate the number of edges considered for relaxation.

To begin with, we argue that the total number of the edges before jump-vertices is $O(n)$. This follows from the observation that for any vertex j in H, at most, one such edge ending at j can be successfully relaxed (Property 2).

To estimate the number of after jump-vertex edges, let us estimate how many such edges can be relaxed from a fixed current vertex i. The weight part of any

such edge (i, j) does not exceed $B = 4\sqrt{2\sigma(G)/t}$ (Rule 3), i.e.,

$$cost_w(i, j) = 2\lfloor 2w(G_{i,j})/tw(G)\rfloor(d(v_{j-1}) - d(v_i)) \leq B.$$

Now observe that in the case of integer costs $j - i + 1 \leq d(v_{j-1}) - d(v_i)$ and that for any vertex $j > jump(i)$ we have $\lfloor 2w(G_{i,j})/tw(G)\rfloor \geq 2$. These inequalities imply $j - i < B/2$, which means that at most $B/2$ "after-jump" edges are relaxed from a fixed current vertex.

According to Rule 1 we can have, at most, B current vertices from which we relax edges since their distances increase by at least 1 in each iteration and the algorithm will find a shortest path to vertex $n' + 1$ in at most B iterations. Therefore, the total number of the after-jump edges relaxed by the algorithm is at most $B^2/2$. Thus, in total, edge relaxation takes $O(\sigma(G)/t)$ time.

We next, consider the cost of Extract_min and Decrease_key operations performed in the priority queue. If we use Fibonacci heaps, we will require $O(\log n)$ time per Extract_min and $O(1)$ amortized time per Decrease_key operation. This leads to $O(n \log n + B^2)$ upper bound on the running time on this part of the algorithm. Instead, we can use the fact that the keys are integers bounded by B and implement the priority queue using an array, which will lead to $O(B^2)$ time for priority queue operations as follows. We employ an array Q of size B with its elements initialized to *nil* and $Q[0] = 0$. When a new distance d to a vertex j is found, the algorithm checks the contents $Q[d]$ and if it is less than j updates $Q[d] = j$. This is the Decrease_key operation obeying Rule 1. Extract_min operations are implemented by traversing the array Q from the left to the right and by returning the contents of nonnil elements of Q. This implementation of the priority queue takes B time to traverse the array (Extract_min operations) and $O(1)$ time per Decrease_key operation, which is $O(B^2) = O(\sigma(G)/t)$ total time.

Summing up, in the case when costs are positive integers, our algorithm for finding a collection of indices minimizing Γ runs in $O(\sigma(G)/t)$ time.

When all the vertices have the same cost, the collection of indices minimizing Γ does not depend on the value of this cost. Thus, we can assume that all vertices have cost 1 and $\sigma(G) = n$. Therefore, the optimal collection of indices is found in $O(\min(n^2, n/t))$ time. □

ACKNOWLEDGMENTS

We thank the referees for suggestions that led to the improved writeup of this paper.

REFERENCES

ALEKSANDROV, L. AND DJIDJEV, H. 1996. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Disc. Math. 9*, 1 (Feb.), 129–150.

BERN, M., MITCHELL, S., AND RUPPERT, J. 1995. Linear-size nonobtuse triangulation of polygons. *Discrete Computational Geometry 14*, 411–428.

BHATT, S. N. AND LEIGHTON, F. T. 1984. A framework for solving vlsi graph layout problems. *J. Computer and System Sciences 28*, 300–343.

CAMP, W. J., PLIMPTON, S. J., HENDRICKSON, B. A., AND LELAND, R. W. 1994. Massively parallel methods for engineering and science problems. *Communications of the ACM 37*, 4 (Apr.), 30–41.

DIKS, K., DJIDJEV, H. N., SYKORA, O., AND VRTO, I. 1993. Edge separators of planar graphs and outer-planar graphs with applications. *J. Algorithms 34*, 258–279.

DJIDJEV, H. N. 1981. A separator theorem. *Compt. Rend. Acad. Bulg. Sci. 34*, 643–645.

DJIDJEV, H. N. 2000. Partitioning planar graphs with vertex costs: Algorithms and applications. *Algorithmica 28*, 1, 51–75.

DJIDJEV, H. N., PANTZIOU, G. E., AND ZAROLIAGIS, C. D. 2000. Improved algorithms for dynamic shortest paths. *Algorithmica 28*, 4, 367–389.

FARRAG, L. 1998. Applications of graph partitioning algorithms to terrain visibility and shortest path problems. M.S. thesis, School of Computer Science, Carleton University, Ottawa, Canada.

FREDERICKSON, G. N. 1987. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput. 16*, 6, 1004–1022.

GILBERT, J. R. AND SCHREIBER, R. 1992. Highly parallel sparse cholesky factorization. *SIAM J. Sci. Stat. Comput. 13*, 1151–1172.

GILBERT, J. R., ROSE, D. J., , AND EDENBRANDT, A. 1984a. A separator theorem for chordal graphs. *SIAM J. Alg. Disc. Meth., 5*, 306–313.

GILBERT, J. R., HUTCHINSON, J. P., AND TARJAN, R. E. 1984b. A separator theorem for graphs of bounded genus. *J. Algorithms 5*, 3, 391–407.

GILBERT, J. R., NG, E. G., AND PEYTON, B. W. 1993. Separators and structure prediction in sparse orthogonal factorization. Tech. Rep. CSL-93-15, Palo Alto Research Center, Xerox Corporation, California.

GILBERT, J. R., MILLER, G. L., AND TENG, S. H. 1998. Geometric mesh partitioning: Implementation and experiments. *SIAM J. Sci. Comput. 19*, 6, 2091–2110.

GOODRICH, M. T. 1992. Planar separators and parallel polygon triangulation. In *Proc. 24th Annual ACM Symposium on Theory of Computing*. 507–516.

HENDRICKSON, B. AND LELAND, R. 1994. The chaco user's guide — version 2.0. Tech. Rep. SAND94-2692, Sandia National Laboratories.

HENZINGER, M. R., KLEIN, P., RAO, S., AND SUBRAMANIAN, S. 1997. Faster shortest-path algorithms for planar graphs. *J. Computer and System Sciences 55*, 1 (Aug.), 3–23.

KARYPIS, G. 1998. *METIS: A family of multilevel partitioning algorithms*. EE/CS, UMN, Minneapolis, USA. /www-users.cs.umn.edu/˜karypis/metis/.

KERNIGHAN, B. W. AND LIN, S. 1970. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 291–307.

LEISERSON, C. E. 1983. *Area efficient VLSI computation*. *In Foundations of Computing*. MIT Press, Cambridge, MA.

LIPTON, R. J. AND TARJAN, R. E. 1979. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36*, 177–189.

LIPTON, R. J. AND TARJAN, R. E. 1980. Applications of a planar separator theorem. *SIAM J. Comput. 9*, 615–627.

MAINI, H. S., MEHROTRA, K. G., MOHAN, C. K., AND RANKA, S. 1994. Genetic algorithms for graph partitioning and incremental graph partitioning. Tech. Rep. CRPC-TR-94504, Rice University.

MEHLHORN, K. AND NÄHER, S. 1999. *LEDA, a platform for combinatorial and geometric computing*. Cambridge University Press, Cambridge.

MILLER, G. L., TENG, S. H., THURSTON, W., AND VAVASIS, S. A. 1997. Separators for sphere-packings and nearest neighbor graphs. *J. ACM 44*, 1, 1–29.

POTHEN, A., SIMON, H. D., AND LIOU, K.-P. 1990. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl. 11*, 3 (July), 430–452.

SCHLOEGEL, K., KARYPIS, G., AND KUMAR, V. 2000. Graph partitioning for high performance scientific simulations. In *CRPC Parallel Computing Handbook*, J. D. et al., Ed. Morgan Kaufmann, San Mates, CA, in press.

SIMON, H. D. AND TENG, S.-H. 1995. How good is recursive bisection? *SIAM J. Scient. Comp. 18*, 5, 1436–1445.

Received September 2002; accepted March 2006