Finding separator cuts in planar graphs within twice the optimal

Naveen Garg [*] *Huzur Saran* [†] *Vijay V. Vazirani* [‡]

Abstract

A factor 2 approximation algorithm for the problem of finding a minimum-cost b-balanced cut in planar graphs is presented, for $b \leq \frac{1}{3}$. We assume that the vertex weights are given in unary; for the case of binary vertex weights, a pseudo-approximation algorithm is presented. This problem is of considerable practical significance, especially in VLSI design.

The natural algorithm for this problem accumulates sparsest cuts iteratively. One of our main ideas is to give a definition of sparsity, called *net-sparsity*, that reflects precisely the cost of the cuts accumulated by this algorithm. However, this definition is too precise: we believe it is **NP**-hard to compute a minimum-net-sparsity cut, even in planar graphs. The rest of our machinery is built to work with this definition and still make it computationally feasible. Towards this end, we use several ideas from the works of Rao [8, 9] and Park and Phillips [7].

1 Introduction

Given an undirected graph with edge costs and vertex weights, the *balance* of a cut is the ratio of the weight of vertices on the smaller side to the total weight in the graph. For $0 < b \leq \frac{1}{2}$, a cut having a balance of at least b is called a b-balanced cut; a $\frac{1}{3}$-balanced cut is given the special name of a *separator*. In this paper, we present a factor 2 approximation algorithm for finding a minimum-cost b-balanced cut in planar graphs, for $b \leq \frac{1}{3}$, assuming that vertex weights are given in unary. We also give examples to show that our analysis is tight. For the case of binary vertex weights, we use scaling to give a pseudo-approximation algorithm: for each $\alpha > 2/b$, it finds a $(b - 2/\alpha)$-balanced cut of cost within twice the cost of an optimal b-balanced cut, for $b \leq 1/3$, in time polynomial in n and α. The previous best approximation guarantee known for b-balanced cuts in planar graphs was $O(\log n)$, due to Rao [8, 9]; for general graphs, no approximation algorithms are known.

The problem of breaking a graph into "small" sized pieces by removal of a "small" set of edges or vertices has attracted much attention since the seminal work of Lipton and Tarjan [5], because this opens up the possibility of a divide-and-conquer strategy for the solution of several problems on the graph. Small balanced cuts have numerous applications, see for example, [1, 3, 4, 6]. Several

[*]Max-Planck-Institut fur Informatik, Im Stadtwald, 66123 Saarbrucken, Germany.

[†]Department of Computer Science & Engineering, Indian Institute of Technology, New Delhi 110016, India.

[‡]College of Computing, Georgia Institute of Technology, Atlanta, GA 30332. Supported by NSF Grant CCR-9627308.

of these applications pertain to planar graphs, the most important one being circuit partitioning in VLSI design.

The *sparsity* of a cut is defined to be the ratio of the cost of the cut and the weight on its smaller side, and a cut having minimum sparsity in the graph is called the *sparsest cut*. Rao [9] gave a $\frac{3}{2}$-approximation algorithm for the problem of finding a sparsest cut in planar graphs, and recently Park and Phillips [7] showed that this problem is polynomial time solvable. A sparsest cut limits multicommodity flow in the same way that a min-cut limits max-flow. Leighton and Rao [3] derived an approximate max-flow min-cut theorem for uniform multicommodity flow, and in the process gave an $O(\log n)$-approximation algorithm for finding a sparsest cut in general graphs. By finding and removing these cuts iteratively, one can show how to find in planar (general) graphs, a b-balanced cut that is within an $O(1)$ factor ($O(\log n)$ factor) of the optimal b'-balanced cut for $b < b'$, and $b \leq \frac{1}{3}$ [8, 9, 3]. For instance, using the Park-Phillips algorithm for sparsest cut in planar graphs, this approach gives a $\frac{1}{3}$-balanced cut that is within 7.1 times the cost of the best $\frac{1}{2}$-balanced cut in planar graphs. Notice however, that these are not true approximation algorithms, since the best $\frac{1}{2}$-balanced cut may have a much higher cost than the best $\frac{1}{3}$-balanced cut.

This iterative algorithm has shortcomings due to which it does not lead to a good true approximation algorithm; these are illustrated via an example in Section 3. One of our main ideas is to give a definition of sparsity, called *net-sparsity*, that overcomes these shortcomings. The notion of *net-cost*, on which this definition of net-sparsity is based, reflects *precisely* the cost of the cuts accumulated iteratively. Indeed, it is too precise to be directly useful computationally – we believe that computing the sparsest cut under this definition is **NP**-hard even in planar graphs. The rest of our machinery is built to work with this definition and still make it computationally feasible, and we manage to scrape by narrowly!

Planarity is exploited in several ways: First, a cut in a planar graph corresponds to a set of cycles in the dual. Secondly, the notion of a *transfer function* turns out to be very useful. Given a planar graph with weights on faces, this notion can be used to define a function on the edges of the graph so that on any cycle it evaluates to the sum of the weights of the faces enclosed by the cycle. Such an idea has been used in the past by Kasteleyn [2], for computing the number of perfect matchings in a planar graph in polynomial time. Kasteleyn defined his function over $GF[2]$. Park and Phillips [7] first defined the function over reals, thereby demonstrating the full power of this notion.

Park and Phillips [7] have shown that the problems of finding a sparsest cut and a minimum b-balanced cut in planar graphs are weakly **NP**-hard, i.e., these problems are **NP**-hard if the vertex weights are given in binary. Indeed, the algorithm they give for finding the sparsest cut in planar graphs is a pseudo-polynomial time algorithm. As a consequence of this algorithm, it follows that if $\mathbf{P} \neq \mathbf{NP}$, finding sparsest cuts in planar graphs is not strongly **NP**-hard. On the other hand it is not known if the b-balanced cut problem in planar graphs is strongly **NP**-hard or if there is a pseudo-polynomial time algorithm for it (the present paper only gives a pseudo-polynomial approximation algorithm). Park and Phillips leave open the question of finding a fully polynomial approximation scheme for sparsest cuts in planar graphs, i.e., if the vertex weights are given in binary. We give such an algorithm using a scaling technique.

2 Preliminaries

Let $G = (V, E)$ be a connected undirected graph, with an edge cost function $c : E \to \mathbf{R}^+$, and a vertex weight function $wt : V \to \mathbf{Z}^+$. Any function that we define on the elements of a universe, extends to sets of elements in the obvious manner; the value of the function on a set is the sum of its values on the elements in the set. Let W be the sum of weights of all vertices in G. A partition $(S, \overline{S})$ of V defines a *cut* in G; the cut consists of all edges that have one end point in S and the other in $\overline{S}$. A set of vertices, S, is said to be *connected* when the subgraph induced on it is connected. If either S or $\overline{S}$ is connected then cut $(S, \overline{S})$ will be called a *simple cut* and when both S and $\overline{S}$ are connected then the cut $(S, \overline{S})$ is called a *bond*.

Given a set of vertices $S \subset V$, we define the cost of this set, cost(S), as the sum of the costs of all edges in the cut $(S, \overline{S})$. The weight of the set S, wt(S), is the sum of the weights of the vertices included in S.

A cut $(S, \overline{S})$ is a *separator* if $\frac{W}{3} \leq \text{wt}(S), \text{wt}(\overline{S}) \leq \frac{2W}{3}$. The cost of a separator is the sum of the costs of the edges in the separator.

Lemma 2.1 *For any connected graph G there exists a minimum-cost separator, $(S, \overline{S})$, which is a simple cut. Further, if S is the side that is not connected, then each connected component of S has weight strictly less than $\frac{W}{3}$.*

Proof: Let $(S, \overline{S})$ be a minimum-cost separator in G. Consider the connected components obtained on removing the edges of this separator. Clearly, no component has weight strictly larger than $\frac{2W}{3}$. If all components have weight strictly less than $\frac{W}{3}$ then both S and $\overline{S}$ are not connected and we can arrive at a contradiction as follows. We first pick two components that have an edge between them and then pick the remaining, one by one, in an arbitrary order till we accumulate a weight of at least $\frac{W}{3}$. The accumulated weight can not exceed $\frac{2W}{3}$ since each component has weight at most $\frac{W}{3}$. Thus we obtain a separator of cost strictly less than the cost of the separator $(S, \overline{S})$; a contradiction. Hence, at least one component has weight between $\frac{W}{3}$ and $\frac{2W}{3}$. If there are two such components then these are the only components since by switching the side of a third component we obtain a cheaper separator. If there is only one component of weight between $\frac{W}{3}$ and $\frac{2W}{3}$ then this separator is optimum iff this component forms one side of the cut and the remaining components the other side. Thus $(S, \overline{S})$ is a bond and if some side of the cut is not connected then all components on that side have weight strictly less than $\frac{W}{3}$. ■

Hence there always exists a minimum-cost separator that is a simple cut. Let OPT denote the set of vertices on the side of this separator that is not connected.

3 Overview of the algorithm

Let S be a set of vertices such that wt$(S) \leq$ wt$(\overline{S})$. The *sparsity* of S is usually defined as the quotient of the cost and the weight of this set, i.e.

$$\text{sparsity}(S) = \frac{\text{cost}(S)}{\text{wt}(S)}$$

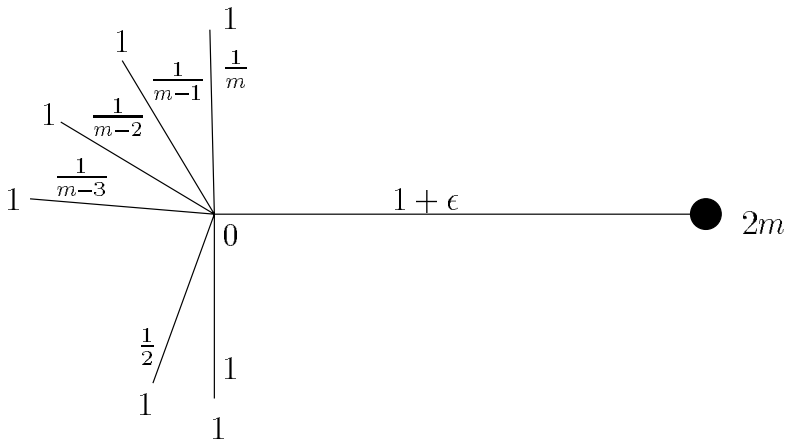

Figure 1: Graph with vertex weights and edge costs showing how minimum sparsity increases. Here $m = \frac{n}{3}$.

A natural approach to finding good separators is to repeatedly find a set of minimum sparsity and remove it from the graph; eventually reporting the union of the removed vertices. It is easy to concoct "bad" examples for this approach by ensuring that the picked vertices always come from the smaller side of the optimal separator, and thereby ensuring that the minimum sparsity available in the remaining graph keeps increasing. This is illustrated in Figure 1; here the first cut picked has a sparsity of $\frac{1}{m}$ whereas the last cut has a sparsity of $\frac{1}{2}$.

This approach has two shortcomings: it removes the vertices picked in each iteration and deals only with the remaining graph in subsequent iterations, and it assumes that edges of the cuts found in each iteration are picked permanently, even though they may not be needed in the final cut. One of our main ideas is to give a definition of "sparsity" under which this algorithm does not suffer from either of these shortcomings.

Let $S, T \subset V$ be two sets of vertices. Define, the *net-cost* of S with respect to T as,

$$\text{net-cost}_T(S) = \text{cost}(T \cup S) - \text{cost}(T)$$

and the *net-weight* of S with respect to T as,

$$\text{net-weight}_T(S) = \text{wt}(T \cup S) - \text{wt}(T)$$

Thus, if we have already picked the set of vertices T, then $\text{net-cost}_T(S)$ measures the extra cost incurred and $\text{net-weight}_T(S)$ the weight added, in picking the set $S \cup T$. Finally, define the *net-sparsity* of S with respect to T as

$$\text{net-sparsity}_T(S) = \frac{\text{net-cost}_T(S)}{\text{net-weight}_T(S)}.$$

For any algorithm that picks a cut by accumulating sets of vertices, the notion of net-cost gives precisely the extra cost incurred in each iteration. But is it so precise that computing the sparsest cut under this definition turns out to be **NP**-hard even in planar graphs? Although we do not have an answer to this question, we believe that it is "yes"! Indeed, the rest of our machinery is built to

work with this definition and still make it computationally feasible, and we manage to scrape by narrowly!

Let us first show that it is not sufficient to just keep picking sets of minimum net-sparsity. Consider the following example: Suppose OPT $= S_1 \cup S_2$, where S_1 is a very sparse set of weight $\frac{W}{3} - \epsilon$, and S_2 is a set of high sparsity and weight ϵ, for a small ϵ. Having picked S_1, we might pick another set, S_3 of sparsity almost that of S_2, and weight $\frac{W}{3} - \epsilon$, and hence, the cost incurred would be arbitrarily high compared to the optimum.

We get around this difficulty by ensuring that in each iteration the set of vertices we pick is such that the total weight accumulated is strictly under $\frac{W}{3}$. More formally: Let T_{i-1} be the set of vertices picked by the end of the (i-1)$^{\text{th}}$ iteration ($T_0 = \phi$). In the i$^{\text{th}}$ iteration we pick a set D_i such that

[weight] $\text{wt}(T_{i-1} \cup D_i) < \frac{W}{3}$.

[net-sparsity] $\forall S : \text{wt}(T_{i-1} \cup S) < \frac{W}{3}, \quad \text{net-sparsity}_{T_{i-1}}(D_i) \leq \text{net-sparsity}_{T_{i-1}}(S)$.

[minimality] D_i has minimum net-weight among all sets satisfying the above conditions.

We call set D_i a *dot* and denote it by $\bullet$. Thus, at the end of the i$^{\text{th}}$ iteration the set of vertices we have picked is given by $T_i = T_{i-1} \cup D_i$. This is how we augment the "partial solution" $(T_1, T_2 \ldots T_i)$ in each iteration.

How do we ever obtain a "complete solution" (a separator)? In the i$^{\text{th}}$ iteration, besides augmenting the partial solution T_{i-1} to the partial solution T_i we also augment it to a complete solution, i.e. we pick a set of vertices B_i such that

[weight] $\frac{W}{3} \leq \text{wt}(T_{i-1} \cup B_i) \leq \frac{2W}{3}$.

[cost] $\forall S : \frac{W}{3} \leq \text{wt}(T_{i-1} \cup S) \leq \frac{2W}{3}, \quad \text{cost}(B_i) \leq \text{cost}(S)$.

Since $T_0 = \phi$, finding the set B_1 corresponds to finding the minimum-cost separator. To avoid this circularity in the argument we restrict B_i to a smaller class of sets:

[cost] $\forall S : (S, \overline{S})$ is a bond and $\frac{W}{3} \leq \text{wt}(T_{i-1} \cup S) \leq \frac{2W}{3}, \quad \text{cost}(B_i) \leq \text{cost}(S)$.

We call the set B_i a *box* and denote it by $\square$. Notice that a box set need not be a bond, and that we count a $\square$ at its cost rather than its net-cost. This is done only to simplify the algorithm and its analysis. The example which shows that the analysis of our algorithm is tight also shows

that counting the $\Box$ at its net-cost would not have led to any improvement in the approximation guarantee.

So, in each iteration we obtain a separator. The solution reported by the algorithm is the one of minimum cost from among all these separators. The algorithm, which we call the *dot-box algorithm* is the following.

Algorithm DOT-BOX ALGORITHM;

1. MINSOL $\leftarrow \infty$, $i \leftarrow 0$, $T_0 \leftarrow \phi$

2. while $\text{wt}(T_i) < \frac{W}{3}$ **do**

2.1. $i \leftarrow i + 1$

2.2. Find $\bullet$ and $\Box$ sets, D_i and B_i respectively.
If there is no $\bullet$ set, **exit**.

2.3. MINSOL $\leftarrow \min(\text{MINSOL}, \text{cost}(T_{i-1}) + \text{cost}(B_i))$

2.4. $T_i \leftarrow T_{i-1} \cup D_i$

end.

We make two remarks regarding step (2.2): First, we conjecture that finding the $\bullet$ set is **NP**-hard. Our procedure to find $\bullet$ sets may not always succeed; however, we will prove that if it fails, then the $\Box$ set found in the current iteration gives a separator within twice OPT. Second, at some iteration it might be the case that no subset of vertices satisfies the weight criterion for a $\bullet$, since each set takes the total weight accumulated to $W/3$ or more. In this case, the DOT-BOX ALGORITHM halts, and outputs the best separator found so far.

4 Analysis of the DOT-BOX ALGORITHM

We first prove some properties of net-cost and net-weight which will be useful for the analysis. From the definition of net-cost and net-weight we have that $\text{net-cost}_T(S) = \text{net-cost}_T(S - T)$ and $\text{net-weight}_T(S) = \text{net-weight}_T(S - T) = \text{wt}(S - T)$. The following property also follows from the definitions

Property 4.1 *Let S_1, S_2 be two sets of vertices not necessarily disjoint. Then*

$$\begin{aligned} \text{net-cost}_T(S_1 \cup S_2) &= \text{net-cost}_T(S_1) + \text{net-cost}_{T \cup S_1}(S_2) \\ \text{net-weight}_T(S_1 \cup S_2) &= \text{net-weight}_T(S_1) + \text{net-weight}_{T \cup S_1}(S_2) \end{aligned}$$

Property 4.2 $\text{net-cost}_T(S) \leq \text{net-cost}_{S \cap T}(S)$.

Proof: Figure 2 shows the edges between the sets $S \cap T$, $S - T$, $T - S$ and $\overline{S \cup T}$ from which the above property is immediate. The net-cost of S with respect to $S \cap T$ may be higher because it includes the cost of edges from $S - T$ to $T - S$. ■

The following property is immediate from Figure 3.

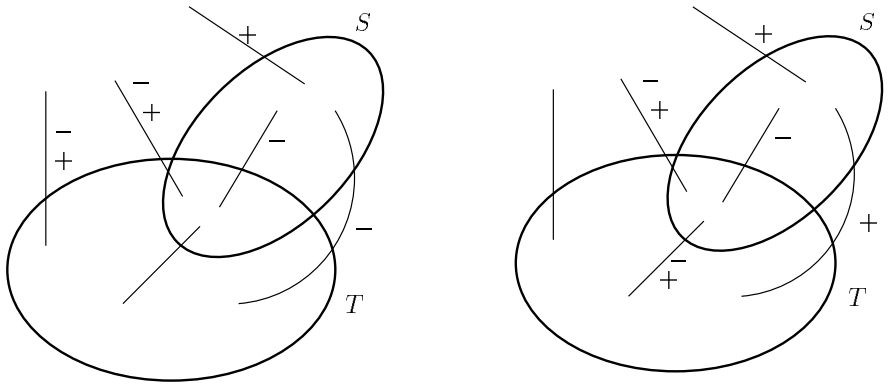

Figure 2: Computation of net-cost$_T(S)$ and net-cost$_{S\cap T}(S)$. A +/- on an edge signifies that the edge is counted in the positive/negative term in the net-cost computation.

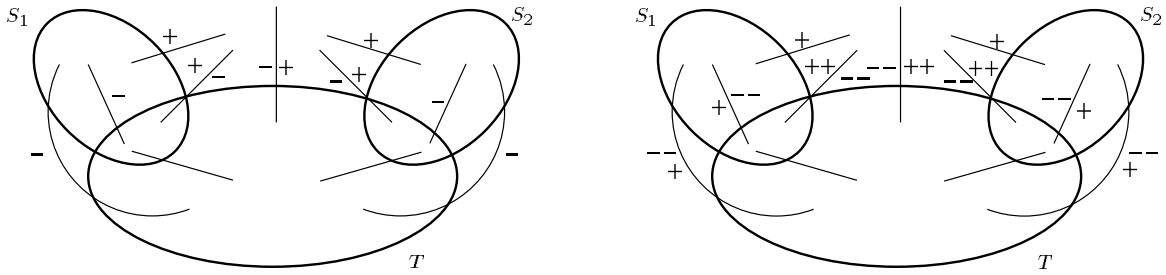

Figure 3: Computation of net-cost$_T(S_1 \cup S_2)$ and net-cost$_T(S_1)$+net-cost$_T(S_2)$.

Property 4.3 *Let S_1, S_2 be two disjoint sets of vertices with no edges between them. Then*

$$\text{net-cost}_T(S_1 \cup S_2) = \text{net-cost}_T(S_1) + \text{net-cost}_T(S_2)$$

Remark 4.1 *For positive real numbers a, b, c, d,*

$$\min(\frac{a}{b}, \frac{c}{d}) \leq \frac{a+c}{b+d} \leq \max(\frac{a}{b}, \frac{c}{d})$$

Further, let $a_i, b_i, 1 \leq i \leq k$ be positive real numbers. Then, $\exists i, 1 \leq i \leq k$ such that $\frac{a_i}{b_i} \leq \frac{\sum_{j=1}^{k} a_j}{\sum_{j=1}^{k} b_j}$.

Lemma 4.1 *The net-sparsity of the $\bullet$'s is increasing, i.e.*

$$\forall i : i \geq 1, \quad \text{net-sparsity}_{T_{i-1}}(D_i) \leq \text{net-sparsity}_{T_i}(D_{i+1})$$

Proof: Since the set $D_i \cup D_{i+1}$ satisfies the weight requirement for a $\bullet$ at the i$^{\text{th}}$ iteration,

$$\text{net-sparsity}_{T_{i-1}}(D_i) \quad \leq \quad \text{net-sparsity}_{T_{i-1}}(D_i \cup D_{i+1}).$$

By Property 4.1,

$$\begin{aligned} \text{net-cost}_{T_{i-1}}(D_i \cup D_{i+1}) &= \text{net-cost}_{T_{i-1}}(D_i) + \text{net-cost}_{T_i}(D_{i+1}) \\ \text{net-weight}_{T_{i-1}}(D_i \cup D_{i+1}) &= \text{net-weight}_{T_{i-1}}(D_i) + \text{net-weight}_{T_i}(D_{i+1}), \end{aligned}$$

which using Remark 4.1 gives us:

$$\begin{aligned} \min(\text{net-sparsity}_{T_{i-1}}(D_i), \text{net-sparsity}_{T_i}(D_{i+1})) &\leq \text{net-sparsity}_{T_{i-1}}(D_i \cup D_{i+1}) \\ &\leq \max(\text{net-sparsity}_{T_{i-1}}(D_i), \text{net-sparsity}_{T_i}(D_{i+1})). \end{aligned}$$

Now, by the first inequality, it must be the case

$$\max(\text{net-sparsity}_{T_{i-1}}(D_i), \text{net-sparsity}_{T_i}(D_{i+1})) = \text{net-sparsity}_{T_i}(D_{i+1}).$$

The lemma follows.

■

Let k be the first iteration at which some connected component of OPT meets the weight requirement of a $\square$.

Lemma 4.2 $\forall i : 1 \leq i \leq k-1, \quad \text{net-sparsity}_{T_{i-1}}(D_i) \leq \text{net-sparsity}_{T_{i-1}}(\text{OPT})$

Proof: Since OPT $\not\subseteq T_{i-1}$ there are connected components of OPT which are not completely contained in T_{i-1}. By assumption, none of these components satisfies the weight requirement for a $\square$; hence each of these components meets the weight requirement for a $\bullet$. Hence the $\bullet$ picked in this iteration should have net-sparsity at most that of any of these components.

By Property 4.3, the net-cost of OPT is the sum of the net-costs of these components of OPT. The same is true for net-weight and hence the component of OPT with minimum net-sparsity has net-sparsity less than that of OPT. The lemma follows. ■

The above two lemmas imply that the net-sparsity at which the $\bullet$'s are picked is increasing and that for any iteration before the k$^{\text{th}}$, this net-sparsity is less than the net-sparsity of OPT in that iteration.

Lemma 4.3 $\forall i : 1 \leq i \leq k-1 \quad \text{cost}(T_i) < \text{cost}(\text{OPT})$

Proof: To establish this inequality for i we consider two processes:

1. The first process is our algorithm which picks the set of vertices D_j at the j$^{\text{th}}$ step, $1 \leq j \leq i$.
2. The second process picks the vertices $D_j \cap \text{OPT}$ at the j$^{\text{th}}$ step, $1 \leq j < i$. At the i$^{\text{th}}$ step it picks the remaining vertices of OPT.

Let P_j be the set of vertices picked by the second process in the first j steps. Then $P_j = \text{OPT} \cap T_j, 1 \leq j < i$ and $P_i = \text{OPT}$. At the j$^{\text{th}}$ step the second process picks an additional weight of net-weight$_{P_{j-1}}(P_j)$ at a cost of net-cost$_{P_{j-1}}(P_j)$. By the fact that the second process picks a subset of what the first process picks at each step we have:

Claim 4.1 *For* $1 \leq j \leq i-1 \quad \text{net-weight}_{T_{j-1}}(D_j) \geq \text{net-weight}_{P_{j-1}}(P_j)$.

Claim 4.2 *For* $1 \leq j \leq i, \quad \text{net-sparsity}_{T_{j-1}}(D_j) \leq \text{net-sparsity}_{P_{j-1}}(P_j)$.

Proof: Since $P_j \cap T_{j-1} = P_{j-1}$, by Property 4.2 we have

$$\text{net-cost}_{T_{j-1}}(P_j) \leq \text{net-cost}_{P_{j-1}}(P_j)$$

Further,

$$\text{net-weight}_{T_{j-1}}(P_j) = \text{net-weight}_{P_{j-1}}(P_j)$$

and hence net-sparsity$_{T_{j-1}}(P_j) \leq$ net-sparsity$_{P_{j-1}}(P_j)$.

For $j < i$, the claim follows since P_j satisfies the weight requirement for a $\bullet$ and D_j was picked as the $\bullet$. For $j = i$, $P_j = \text{OPT}$ and the claim follows from Lemma 4.2. ■

The above claims imply that in each iteration (1 through i) the first process picks vertices at a lower net-sparsity than the second process. If both these processes were picking the same additional weight in each iteration then this fact alone would have implied that the cost of the vertices picked by the first process is less than the cost of the vertices picked by the second. But this is not the case. What is true, however, is the fact that in iterations 1 through i-1 the first process picks a larger additional weight than the second process. In the i$^{\text{th}}$ iteration, the second process picks enough additional weight so that it now has accumulated a total weight strictly larger than that picked by the first process (since $\text{wt}(\text{OPT}) \geq \frac{W}{3} > \text{wt}(T_i)$). But the net-sparsity at which the second process picks vertices in the i$^{\text{th}}$ iteration is more than the maximum (over iterations 1 through i) net-sparsity at which the first process picks vertices. So it follows that the cost of the vertices picked by the first process is strictly less than the cost of the vertices picked by the second, i.e. $\text{cost}(T_i) < \text{cost}(\text{OPT})$. ■

Consider the separator found in the k$^{\text{th}}$ iteration, i.e. the cut $(T_{k-1} \cup B_k, \overline{T_{k-1} \cup B_k})$. This solution is formed by picking $\bullet$'s in the first k-1 steps and a $\square$ in the k$^{\text{th}}$ step.

Lemma 4.4 $\text{cost}(T_{k-1} \cup B_k) \leq 2 \cdot \text{cost}(\text{OPT})$.

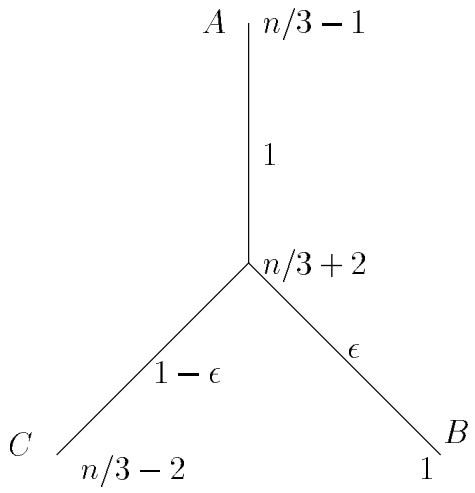

Figure 4: A tight example for our analysis. Vertex weights and edge costs are given and $\epsilon = 3/n$.

Proof: Any connected component of OPT is a bond. In the k$^{\text{th}}$ iteration there exists a connected component of OPT, say OPT_j such that $\frac{W}{3} \leq \text{wt}(T_{k-1} \cup \text{OPT}_j) \leq \frac{2W}{3}$. Hence the $\Box$ at the k$^{\text{th}}$ step should have cost at most $\text{cost}(\text{OPT}_j)$, i.e. $\text{cost}(B_k) \leq \text{cost}(\text{OPT}_j) \leq \text{cost}(\text{OPT})$.

From Lemma 4.3 we know that $\text{cost}(T_{k-1}) < \text{cost}(\text{OPT})$. Hence

$$\text{cost}(T_{k-1} \cup B_k) < \text{cost}(T_{k-1}) + \text{cost}(B_k) < 2 \cdot \text{cost}(\text{OPT})$$

■

Since the Dot-Box Algorithm outputs the best separator found, we have:

Theorem 4.5 *The cost of the separator found by the* Dot-Box Algorithm *is at most twice the cost of* OPT.

Our analysis of the Dot-Box Algorithm is tight; when run on the example in Figure 4, it picks a separator of cost almost twice the optimum. In this example, $\text{OPT} = A \cup B$ and so $\text{cost}(\text{OPT}) = 1 + \epsilon$. For $\epsilon > 3/n$, the $\bullet$ in the first iteration is the set C and the $\Box$ is the set A. This separator, which is also the one returned by the Dot-Box Algorithm, has cost $2 - 2\epsilon$ and hence the approximation ratio is $2\left(\frac{n-3}{n+3}\right)$.

5 Structural properties of our solution, and a computationally easier definition of net-cost

In this section we prove some structural properties of the solution found by the Dot-Box Algorithm. This allows us to redefine net-cost in such a manner that it becomes computationally easier and yet the analysis from the previous section continues to hold.

Lemma 5.1 *For* $1 \leq i \leq k-1, \overline{T_i}$ *is connected.*

Proof: For contradiction assume that $\overline{T_i}$ is not connected. Let A be a connected component of $\overline{T_i}$. There are three cases:

$\mathrm{wt}(T_i \cup A) < \mathbf{W/3}$ The set A satisfies the weight requirement for a $\bullet$ at the (i+1)$^{\text{th}}$ iteration. Since A has edges only to vertices in T_i, the net-cost of A with respect to T_i is negative. Hence net-sparsity$_{T_i}(A)$ is negative, contradicting Lemma 4.1.

$\mathbf{W/3} \leq \mathrm{wt}(T_i \cup A) \leq \mathbf{2W/3}$

The cut $(T_i \cup A, \overline{T_i \cup A})$ is a separator of cost, $\mathrm{cost}(T_i \cup A) < \mathrm{cost}(T_i) < \mathrm{cost}(\mathrm{OPT})$, a contradiction.

$\mathrm{wt}(T_i \cup A) > \mathbf{2W/3}$

Since $\mathrm{wt}(T_i) < W/3$, the condition of this case implies that $\mathrm{wt}(A) > W/3$. If $\mathrm{wt}(A) \leq 2W/3$ then once again we have a contradiction since now the cut $(A, \overline{A})$ is a separator of cost, $\mathrm{cost}(A) < \mathrm{cost}(T_i) < \mathrm{cost}(\mathrm{OPT})$. Thus it must be the case that $\mathrm{wt}(A) > 2W/3$.

Since the above argument implies that each connected component of $\overline{T_i}$ has weight greater than $2W/3$, $\overline{T_i}$ must have only one connected component. ■

Lemma 5.2 *For $1 \leq i \leq k-1$, the set $T_i - T_{i-1}$ is connected.*

Proof: For contradiction assume that $T_i - T_{i-1}$ is not connected. Let A be a connected component of $T_i - T_{i-1}$ and B be the rest of $T_i - T_{i-1}$ (we also denote by A, B the corresponding sets of vertices).

If D_i is the $\bullet$ at the i$^{\text{th}}$ iteration then net-cost$_{T_{i-1}}(D_i)$ = net-cost$_{T_{i-1}}(A \cup B)$. Since A and B are disjoint set of vertices with no edges between them, by Property 4.3

$$\text{net-cost}_{T_{i-1}}(D_i) = \text{net-cost}_{T_{i-1}}(A) + \text{net-cost}_{T_{i-1}}(B)$$

Further,

$$\text{net-weight}_{T_{i-1}}(D_i) = \text{net-weight}_{T_{i-1}}(A) + \text{net-weight}_{T_{i-1}}(B)$$

Thus either it is the case that one of A, B has smaller net-sparsity than D_i which contradicts the assumption that D_i is a $\bullet$ or else, both A, B have the same net-sparsity as D_i but this contradicts the minimality requirement on D_i. ■

Lemma 5.3 *For every iteration $i : 1 \leq i \leq k-1$, there exists a $\bullet$, D_i, satisfying*

1. *$(D_i, \overline{D_i})$ is a bond.*
2. *Each connected component of T_{i-1} is contained in D_i or $\overline{D_i}$ and there is no edge between D_i and components of T_{i-1} in $\overline{D_i}$.*

Proof: The set $T_i - T_{i-1}$ together with any subset of T_{i-1} is also a $\bullet$ for the i$^{\text{th}}$ iteration. We form a new $\bullet$, D_i, by merging with $T_i - T_{i-1}$ all connected components of T_{i-1} which have an edge to $T_i - T_{i-1}$. Since $T_i - T_{i-1}$ is connected, so is D_i. Further, since the graph is connected, every remaining component of T_{i-1} has an edge to $\overline{T_i}$ so that $\overline{D_i}$ is also connected. Thus $(D_i, \overline{D_i})$ is a bond. It follows from the definition of D_i that there is no edge between D_i and components of T_{i-1} in $\overline{D_i}$. ■

Since every $\bullet$ in iterations 1 through $k-1$ satisfies the conditions in Lemma 5.3, we can restrict our search for the $\bullet$ at the i$^{\text{th}}$ iteration to sets that satisfy these conditions as additional requirements.

$(D_i, \overline{D_i})$ is a bond.

Each connected component of T_{i-1} is contained in D_i or $\overline{D_i}$ and there is no edge between D_i and components of T_{i-1} in $\overline{D_i}$.

Let $G_i = (V_i, E_i)$ be the graph obtained by shrinking each connected component of T_{i-1} into a single vertex, removing the self-loops formed and replacing each set of parallel edges by one edge of cost equal to the sum of the cost of the edges in the set. For finding a $\bullet$ at the i$^{\text{th}}$ iteration we consider only such sets, S, such that no connected component of T_{i-1} is split across $(S, \overline{S})$ and $(S, \overline{S})$ is a bond. Therefore we need to look only at subsets of V_i that correspond to bonds in G_i.

Let S be a subset of vertices in G_i. The *trapped cost of S with respect to T_{i-1}*, denoted by trapped-cost$_{T_{i-1}}(S)$, is the sum of the costs of the components of T_{i-1} that are contained in S. We now redefine the *net-cost of S with respect to T_{i-1}* as

$$\text{net-cost}_{T_{i-1}}(S) = \text{cost}(S) - \text{trapped-cost}_{T_{i-1}}(S)$$

Note that for any subset of vertices in G_i, the net-cost under this new definition is at least as large as that under the previous definition. However, and this is crucial, the net-cost of the $\bullet$ set D_i remains unchanged. This is so because by Lemma 5.3 there are no edges between D_i and the components of T_{i-1} not in D_i. Therefore, a $\bullet$ under this new definition of net-cost will also be a $\bullet$ under the previous definition, and so our analysis of the Dot-Box Algorithm continues to hold.

6 Onto planar graphs

We do not know the complexity of computing $\bullet$ sets; we suspect that it is **NP**-hard even in planar graphs. Yet, we can implement the Dot-Box Algorithm for planar graphs — using properties of cuts in planar graphs, and by showing that if in any iteration, the algorithm does not find the $\bullet$ set, then in fact, the separator found (using the $\square$ set found in this iteration) is within twice the optimal (this is proven in Theorem 7.1).

6.1 Associating cycles with sets

Let G^D be the planar dual of G and we fix an embedding of G^D.

Proposition 6.1 *There is a one-to-one correspondence between bonds in G and simple cycles in G^D.*

Proof: Let $(S, \overline{S})$ be a bond in G. Since S is connected, the faces corresponding to S in G^D are adjacent and so the edges of G^D corresponding to $(S, \overline{S})$ form a simple cycle.

For the converse, let C be a simple cycle in G^D which corresponds to the cut $(S, \overline{S})$ in G. Let u, v be two vertices in G that are on the same side of the cut $(S, \overline{S})$. To prove that $(S, \overline{S})$ is a bond it suffices to show a path between u and v in G that does not use any edge of $(S, \overline{S})$.

Embed G and G^D in $\mathbf{R} \times \mathbf{R}$, and consider the two faces of G^D corresponding to vertices u and v. Pick an arbitrary point in each face, for instance, the points corresponding to u and v. Since C is a simple cycle in G^D (and hence in $\mathbf{R} \times \mathbf{R}$) there is a continuous curve (in $\mathbf{R} \times \mathbf{R}$) that connects the two points without intersecting C. By considering the faces of G^D that this curve visits, and the edges of G^D that the curve intersects, we obtain a path in G that connects vertices u, v without using any edge of $(S, \overline{S})$.

■

Since for finding a $\bullet$ and $\square$ we only need to consider sets, S, such that $(S, \overline{S})$ is a bond, we can restrict ourselves to simple cycles in G^D. Furthermore, the two orientations of a simple cycle can be used to distinguish between the two sides of the cut that this cycle corresponds to. The notation we adopt is: with a cycle C directed clockwise we associate the set of faces in G^D (and hence vertices in G) *enclosed* by C (the side that does not include the infinite face is said to be enclosed by C and the side containing the infinite face is said to be *outside* C).

Let $\vec{G^D}$ be the graph obtained from G^D by replacing each undirected edge (u, v) by two directed edges $(u \to v)$ and $(v \to u)$. By the preceding discussion, there exists a correspondence between sets of vertices, S, in G such that $(S, \overline{S})$ is a bond and directed simple cycles in $\vec{G^D}$.

6.2 Transfer function

We associate a cost function, c, with the edges of $\vec{G^D}$ in the obvious manner; an edge in $\vec{G^D}$ is assigned the same cost as the corresponding dual edge in G. Thus, for a directed cycle, C, $c(C)$, denotes the sum of the costs of the edges along the cycle. We would also like to associate functions, t_i, w_i with the edges of $\vec{G^D}$ so that if S is the set corresponding to a directed simple cycle C, then $t_i(C) = \text{trapped-cost}_{T_{i-1}}(S)$ and $w_i(C) = \text{net-weight}_{T_{i-1}}(S)$. We achieve this by means of a *transfer function*.

The notion of a transfer function was introduced by Park and Phillips [7], and can be viewed as an extension of a function given by Kasteleyn [2]. A function g defined on the edges of $\vec{G^D}$ is *anti-symmetric* if $g(u \to v) = -g(v \to u)$. (Notice that the function c defined above is *symmetric*.) Let $f : V \to \mathbf{R}$ be a function on the vertices of G. The transfer function corresponding to f is an anti-symmetric function, f_t, on the edges of $\vec{G^D}$ such that the sum of the values that f_t takes on the edges of any clockwise (anticlockwise) simple cycle in $\vec{G^D}$, is equal to the (negative of the) sum of the values that f takes on the vertices corresponding to the faces enclosed by this cycle.

That a transfer function exists for every function defined on the vertices of G and that it can be computed efficiently follows from the following simple argument. Pick a spanning tree in G^D, and

set f_t to zero for the corresponding edges in $\vec{G^D}$. Now, add the remaining edges of G^D in an order so that with each edge added, one face of this graph is completed. Note that before the edge e is added, all other edges of the face that e completes have been assigned a value under f_t. One of the two directed edges corresponding to e is used in the clockwise traversal of this face, and the other in the anti-clockwise traversal. Since the value of f for this face is known and since f_t should sum to this value (the negative of this value) in a clockwise (anti-clockwise) traversal of this face, the value of f_t for the two directed edges corresponding to e can be determined. Note that the function f_t obtained in this manner is anti-symmetric and this together with the fact that the edges of any simple cycle in G^D can be written as a $GF[2]$ sum of the edges belonging to the faces contained in the cycle implies that f_t has the desired property.

7 Finding • sets

Recall that a • at the i$^{\text{th}}$ iteration is a bond in the graph $G_i = (V_i, E_i)$ and hence we can restrict our search for a • at the i$^{\text{th}}$ iteration to directed simple cycles in $\vec{G_i^D}$.

7.1 Obtaining net-weight and net-cost from transfer functions

Let $\tilde{w_i} : V_i \to \mathbf{Z}^+$, $\tilde{t_i} : V_i \to \mathbf{R}^+$ be two functions defined on the vertices of G_i as follows. The vertices in V_i obtained by shrinking connected components of T_{i-1} have $\tilde{w_i} = 0$ and $\tilde{t_i}$ equal to the cost of the corresponding component of T_{i-1}. The remaining vertices have $\tilde{w_i} = wt$ and $\tilde{t_i} = 0$. Let w_i, t_i denote the transfer functions corresponding to functions $\tilde{w_i}, \tilde{t_i}$. We now relate the values of the functions c, w_i, t_i on a directed simple cycle to the net-cost, net-weight and net-sparsity of the set corresponding to the cycle.

Let C be a directed simple cycle in $\vec{G_i^D}$ and $S \subset V$ the set corresponding to it. If C is clockwise then the net-weight and trapped-cost of S are given by the values of the transfer functions on C, i.e.

$$\begin{aligned} \text{net-weight}_{T_{i-1}}(S) &= w_i(C) \\ \text{trapped-cost}_{T_{i-1}}(S) &= t_i(C) \end{aligned}$$

If C is anti-clockwise then the values of the transfer functions w_i, t_i on C equal the negative of the net-weight and the trapped-cost of the set enclosed by C (which is $\overline{S}$ in our notation). Hence

$$\text{net-weight}_{T_{i-1}}(S) = W - \text{wt}(T_{i-1}) - \text{net-weight}_{T_{i-1}}(\overline{S}) = W - \text{wt}(T_{i-1}) + w_i(C)$$

$$\text{trapped-cost}_{T_{i-1}}(S) = \text{cost}(T_{i-1}) - \text{trapped-cost}_{T_{i-1}}(\overline{S}) = \text{cost}(T_{i-1}) + t_i(C)$$

Recalling our new definition of net-cost

$$\text{net-cost}_{T_{i-1}}(S) = \text{cost}(S) - \text{trapped-cost}_{T_{i-1}}(S)$$

We conclude that if C is clockwise

$$\text{net-sparsity}_{T_{i-1}}(S) = \frac{c(C) - t_i(C)}{w_i(C)}$$

and for anti-clockwise C,

$$\text{net-sparsity}_{T_{i-1}}(S) = \frac{c(C) - (t_i(C) + \text{cost}(T_{i-1}))}{w_i(C) + W - \text{wt}(T_{i-1})}$$

Hence, for a simple directed cycle C, once we know the values of the transfer functions w_i, t_i it is easy to determine the net-weight and net-sparsity of the corresponding set S. Note that the orientation of C can be determined by the sign of $w_i(C)$ since $w_i(C) > 0$ ($w_i(C) < 0$) implies that C is clockwise (anti-clockwise).

7.2 The approach to finding a •

For a fixed value of $w_i(C)$, net-sparsity$_{T_{i-1}}(S)$ is minimized when $c(C) - t_i(C)$ is minimum. This suggests the following approach for finding a •:

> For each w in the range $(0 \leq w \leq \frac{W}{3} - \text{wt}(T_{i-1})) \cup (-W + \text{wt}(T_{i-1}) \leq w \leq \frac{-2W}{3})$, compute **min-cycle($w$)**: a directed simple cycle with minimum $c(C) - t_i(C)$ among all directed cycles C with $w_i(C) = w$. Find the net-sparsity of the set corresponding to each of these cycles. The set with the minimum net-sparsity is the • for this iteration.

However, we can implement only a weaker version of procedure **min-cycle.** Following [7], we construct a graph H_i whose vertices are 2-tuples of the kind (v, j) where v is a vertex in $G_i^{\vec{D}}$ and j is an integer between $-nW$ and nW. For an edge $e = u \to v$ in $G_i^{\vec{D}}$ we have, for all possible choices of j, edge $(u, j) \to (v, j + w_i(e))$ of length $c(e) - t_i(e)$. The shortest path between $(v, 0)$ and (v, w) in H_i gives the shortest cycle among all directed cycles in $G_i^{\vec{D}}$ which contain v and for which $w_i(C) = w$. By doing this computation for all choices of v, we can find the shortest cycle with $w_i(C) = w$.

Two questions arise:

1. Is H_i free of negative cycles? This is essential for computing shortest paths efficiently.
2. Is the cycle obtained in $G_i^{\vec{D}}$ simple?

The answer to both questions is "no". Interestingly enough, things still work out. We will first tackle the second question (in Theorem 7.1), and then the first (in Lemma 7.2).

7.3 Overcoming non-simple cycles

Before we discuss how to get over this problem, we need to have a better understanding of the structure of a non-simple cycle, C. If C is not a simple cycle in $G_i^{\vec{D}}$, decompose it arbitrarily into a collection of edge-disjoint directed simple cycles, $\mathcal{C}$. Let $(S_j, \overline{S_j})$ be the cut (in G_i) corresponding to a cycle $C_j \in \mathcal{C}$ and let S_j be the side of the cut that has smaller net-weight. Further, let $\mathcal{S}$ be the collection of sets S_j, one for each $C_j \in \mathcal{C}$.

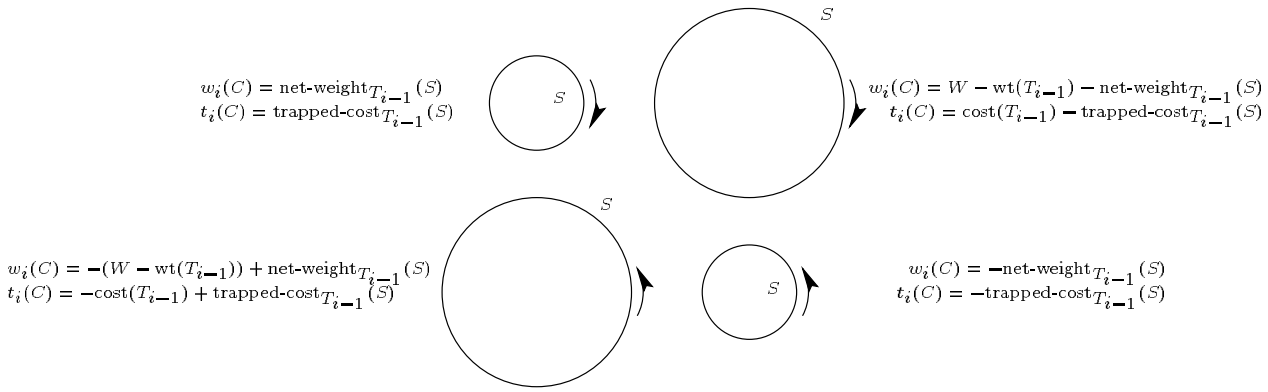

Figure 5: Relationship between net-weight$_{T_{i-1}}(S)$ and $w_i(C)$ for the four cases.

The value of the transfer functions w_i, t_i over C is the sum of their values over the cycles C_j in the collection $\mathcal{C}$. Also,

$$c(C) = \sum_{S_j \in \mathcal{S}} \text{cost}(S_j)$$

For each cycle $C_j \in \mathcal{C}$ we need to relate the net-weight, trapped-cost of S_j to the value of the transfer functions w_i, t_i on C_j. C_j might either be clockwise or anti-clockwise. Further S_j might either be inside C_j or outside C_j. This gives us a total of four different cases. The relationship between net-weight$_{T_{i-1}}(S_j)$ and $w_i(C_j)$, and trapped-cost$_{T_{i-1}}(S_j)$ and $t_i(C_j)$ is given in Figure 5 and can be captured succinctly as follows

$$\begin{aligned} w_i(C_j) &= x_j(W - \text{wt}(T_{i-1})) + y_j \cdot \text{net-weight}_{T_{i-1}}(S_j) \\ t_i(C_j) &= x_j \cdot \text{cost}(T_{i-1}) + y_j \cdot \text{trapped-cost}_{T_{i-1}}(S_j) \end{aligned}$$

where $x_j \in \{+1, 0, -1\}$ and $y_j \in \{+1, -1\}$.

Hence we get a *decomposition rule* relating the value of the functions w_i, t_i on a non-simple cycle C to the net-weight and trapped-cost of the sets induced by this cycle.

$$\begin{aligned} w_i(C) &= x(W - \text{wt}(T_{i-1})) + \sum_{S_j \in \mathcal{S}} y_j \cdot \text{net-weight}_{T_{i-1}}(S_j) \\ t_i(C) &= x \cdot \text{cost}(T_{i-1}) + \sum_{S_j \in \mathcal{S}} y_j \cdot \text{trapped-cost}_{T_{i-1}}(S_j) \end{aligned}$$

where $x = \sum_j x_j$ is an integer.

7.4 A key theorem

Let D_i be a $\bullet$ at the i$^{\text{th}}$ iteration ($i \leq k-1$) and C^* the directed simple cycle in $\vec{G}_i^D$ corresponding to it. Further, let C be the directed cycle reported by **min-cycle**$(w_i(C^*))$.

Theorem 7.1 *If C is not simple then the separator found in this iteration has cost at most* $2 \cdot \text{cost(OPT)}$*, i.e.*

$$\text{cost}(T_{i-1} \cup B_i) \leq 2 \cdot \text{cost(OPT)}$$

Proof: Since C is the directed cycle for which $c(C) - t_i(C)$ is minimum among all cycles with $w_i(C) = w_i(C^*)$ we can claim the following.

If C^* is clockwise, i.e. $w_i(C^*) > 0$ then

$$\begin{aligned} w_i(C) &= \text{net-weight}_{T_{i-1}}(D_i) \\ c(C) - t_i(C) &\leq \text{net-cost}_{T_{i-1}}(D_i) \end{aligned}$$

and if C^* is anti-clockwise, i.e. $w_i(C^*) < 0$ then

$$\begin{aligned} W - \text{wt}(T_{i-1}) + w_i(C) &= \text{net-weight}_{T_{i-1}}(D_i) \\ c(C) - t_i(C) - \text{cost}(T_{i-1}) &\leq \text{net-cost}_{T_{i-1}}(D_i) \end{aligned}$$

Substituting for $w_i(C)$ and $t_i(C)$ by the decomposition rule we get

$$z(W - \text{wt}(T_{i-1})) + \sum_{S_j \in \mathcal{S}} y_j \cdot \text{net-weight}_{T_{i-1}}(S_j) = \text{net-weight}_{T_{i-1}}(D_i) \quad (1)$$

$$-z \cdot \text{cost}(T_{i-1}) + \sum_{S_j \in \mathcal{S}} (\text{cost}(S_j) - y_j \cdot \text{trapped-cost}_{T_{i-1}}(S_j)) \leq \text{net-cost}_{T_{i-1}}(D_i) \quad (2)$$

where z is x if C^* is clockwise and $x + 1$ if C^* is anti-clockwise.

We now prove that there exists $S_j \in \mathcal{S}$ which meets the weight requirement for a $\square$ and has cost no more than the cost of T_i, i.e,

1. $\frac{W}{3} \leq \text{wt}(T_{i-1} \cup S_j) \leq \frac{2W}{3}$
2. $\text{cost}(S_j) \leq \text{cost}(T_{i-1}) + \text{net-cost}_{T_{i-1}}(D_i)$

Assume for contradiction that no such S_j exists. The following observations about the cost/net-cost of a set $S_j \in \mathcal{S}$ are immediate.

Observation 7.1 *if* $\text{net-weight}_{T_{i-1}}(S_j) \geq W/3 - \text{wt}(T_{i-1})$ *then*

$$\text{cost}(S_j) > \text{cost}(T_{i-1}) + \text{net-cost}_{T_{i-1}}(D_i)$$

which implies

$$\text{net-cost}_{T_{i-1}}(S_j) > \text{net-cost}_{T_{i-1}}(D_i)$$

Observation 7.2 *if* $\text{net-weight}_{T_{i-1}}(S_j) < W/3 - \text{wt}(T_{i-1})$ *then*

$$\text{net-sparsity}_{T_{i-1}}(S_j) \geq \text{net-sparsity}_{T_{i-1}}(D_i)$$

and hence

$$\text{net-cost}_{T_{i-1}}(S_j) \geq \text{net-sparsity}_{T_{i-1}}(D_i) \cdot \text{net-weight}_{T_{i-1}}(S_j)$$

From the above two observations it follows that

Observation 7.3 *All sets $S_j \in \mathcal{S}$ have non-negative net-cost, i.e.* $\text{net-cost}_{T_{i-1}}(S_j) \geq 0$.

The idea behind obtaining a contradiction is as follows. For every integral choice of z we use equation 1 to provide a lower bound on the total net-weight of the sets $S_j \in \mathcal{S}$ and equation 2 to provide an upper bound on the total net-cost of the sets $S_j \in \mathcal{S}$. We then use the above observations on the cost/net-cost of sets $S_j \in S$ to argue that there is no way of having sets with so large a total net-weight at so little a total net-cost.

We shall consider 3 cases depending upon whether z is positive/negative/zero.

$\mathbf{z = 0}$

Equation 2 implies

$$\begin{aligned} \text{net-cost}_{T_{i-1}}(D_i) &\geq \sum_{S_j \in \mathcal{S}} (\text{cost}(S_j) - y_j \cdot \text{trapped-cost}_{T_{i-1}}(S_j)) \\ &\geq \sum_{S_j \in \mathcal{S}} \text{net-cost}_{T_{i-1}}(S_j) \end{aligned}$$

and from equation 1 we have

$$\begin{aligned} \sum_{S_j \in \mathcal{S}} \text{net-weight}_{T_{i-1}}(S_j) &\geq \sum_{S_j \in \mathcal{S}} y_j \cdot \text{net-weight}_{T_{i-1}}(S_j) \\ &= \text{net-weight}_{T_{i-1}}(D_i) \end{aligned}$$

Since the net-cost of each set is non-negative (Observation 7.3) each set in $\mathcal{S}$ has net-cost no more than $\text{net-cost}_{T_{i-1}}(D_i)$. This in turn implies that every set in $\mathcal{S}$ has net-weight strictly less than $W/3 - \text{wt}(T_{i-1})$ (Observation 7.1). Thus every set in $\mathcal{S}$ meets the weight requirement for a $\bullet$. Since the net-cost of every set in $\mathcal{S}$ is non-negative, Remark 4.1 applied to the above two inequalities implies that either there exists $S_j \in \mathcal{S}$ of lower net-sparsity than D_i or that every set in $\mathcal{S}$ has the same net-sparsity as D_i and that the sum of the net-weight of the sets in $\mathcal{S}$ is equal to the net-weight of D_i. The first setting leads to a contradiction since every set in $\mathcal{S}$ satisfies the weight requirement for a $\bullet$ and D_i is the $\bullet$ at this iteration. The second setting in turn contradicts the minimality requirement on D_i.

$\mathbf{z > 0}$

Let $\mathcal{S}^-$ denote the collection of sets $S_j \in \mathcal{S}$ with $y_j = -1$. Equation 2 now yields

$$\begin{aligned} \text{net-cost}_{T_{i-1}}(D_i) + z \cdot \text{cost}(T_{i-1}) &\geq \sum_{S_j \in \mathcal{S}} (\text{cost}(S_j) - y_j \cdot \text{trapped-cost}_{T_{i-1}}(S_j)) \\ &\geq \sum_{S_j \in \mathcal{S}^-} (\text{cost}(S_j) + \text{trapped-cost}_{T_{i-1}}(S_j)) \\ &\geq \sum_{S_j \in \mathcal{S}^-} \text{cost}(S_j) \end{aligned}$$

where the second inequality follows from the fact that all sets in $\mathcal{S} - \mathcal{S}^-$ have non-negative net-cost. We shall develop a contradiction by showing that the costs of the sets in $\mathcal{S}^-$ is more

than the left hand side of the above inequality. A lower bound on the total net-weight of the sets in $\mathcal{S}^-$ can be obtained from equation 1 as follows

$$\begin{aligned} z(W - \text{wt}(T_{i-1})) - \text{net-weight}_{T_{i-1}}(D_i) &= -\sum_{S_j \in \mathcal{S}} y_j \cdot \text{net-weight}_{T_{i-1}}(S_j) \\ &\leq \sum_{S_j \in \mathcal{S}^-} \text{net-weight}_{T_{i-1}}(S_j) \end{aligned}$$

What is the cheapest way of picking sets so that their net-weight is at least $z(W - \text{wt}(T_{i-1})) - \text{net-weight}_{T_{i-1}}(D_i)$? By Observation 7.2 a set S_j such that $\text{net-weight}_{T_{i-1}}(S_j) < W/3 - \text{wt}(T_{i-1})$ can be picked only at a net-sparsity of at least $\text{net-sparsity}_{T_{i-1}}(D_i)$. On the other hand Observation 7.1 says that we could be picking sets with large net-weight for cost little more than $\text{cost}(T_{i-1}) + \text{net-cost}_{T_{i-1}}(D_i)$. Since any set in $\mathcal{S}$ has net-weight at most $\frac{W - \text{wt}(T_{i-1})}{2}$ the unit cost of picking these large sets could be as small as

$$\begin{aligned} \frac{\text{cost}(T_{i-1}) + \text{net-cost}_{T_{i-1}}(D_i)}{\frac{W - \text{wt}(T_{i-1})}{2}} &< \frac{\text{cost}(T_{i-1}) + \text{net-cost}_{T_{i-1}}(D_i)}{\text{wt}(T_{i-1}) + \text{net-weight}_{T_{i-1}}(D_i)} \\ &\leq \max\left\{ \frac{\text{cost}(T_{i-1})}{\text{wt}(T_{i-1})}, \frac{\text{net-cost}_{T_{i-1}}(D_i)}{\text{net-weight}_{T_{i-1}}(D_i)} \right\} \\ &\leq \text{net-sparsity}_{T_{i-1}}(D_i) \end{aligned}$$

where the last inequality follows from the fact that $\text{sparsity}(T_{i-1}) \leq \text{net-sparsity}_{T_{i-1}}(D_i)$ which in turn is a consequence of Lemma 4.1.

Thus the cheapest possible way of picking sets is to pick sets of net-weight $\frac{W - \text{wt}(T_{i-1})}{2}$, incurring a cost of little more than $\text{cost}(T_{i-1}) + \text{net-cost}_{T_{i-1}}(D_i)$ for each set picked. Since we need to pick a net-weight of at least $z(W - \text{wt}(T_{i-1})) - \text{net-weight}_{T_{i-1}}(D_i)$, we would have to pick at least $2z - 1$ such sets and so the cost incurred is at least

$$\begin{aligned} \sum_{S_j \in \mathcal{S}^-} \text{cost}(S_j) &> (2z-1)(\text{cost}(T_{i-1}) + \text{net-cost}_{T_{i-1}}(D_i)) \\ &\geq z \cdot \text{cost}(T_{i-1}) + \text{net-cost}_{T_{i-1}}(D_i) \end{aligned}$$

where the last inequality follows from the fact that $z \geq 1$ (hence $2z - 1 \geq z$ and $2z - 1 \geq 1$). This however contradicts the upper bound on the sum of the costs of the sets in $\mathcal{S}^-$, that we derived at the beginning of this case.

$\mathbf{z < 0}$

Let $\mathcal{S}^+$ denote the collection of sets $S_j \in \mathcal{S}$ with $y_j = 1$. Equation 2 now yields

$$\begin{aligned} \text{net-cost}_{T_{i-1}}(D_i) &\geq \sum_{S_j \in \mathcal{S}} (\text{cost}(S_j) - y_j \cdot \text{trapped-cost}_{T_{i-1}}(S_j)) \\ &\geq \sum_{S_j \in \mathcal{S}^+} \text{net-cost}_{T_{i-1}}(S_j) \end{aligned}$$

The total net-weight of the sets in $\mathcal{S}^+$ can be bounded using equation 1 as follows.

$$\begin{aligned}\sum_{S_j \in \mathcal{S}^+} \text{net-weight}_{T_{i-1}}(S_j) &\geq \sum_{S_j \in \mathcal{S}} y_j \cdot \text{net-weight}_{T_{i-1}}(S_j) \\ &= \text{net-weight}_{T_{i-1}}(D_i) - z(W - \text{wt}(T_{i-1})) \\ &\geq -z(W - \text{wt}(T_{i-1}))\end{aligned}$$

where the last inequality follows from the fact that the net-weight of D_i is non-negative.

What is the cheapest way of picking sets so that their net-weight is at least $-z(W - \text{wt}(T_{i-1}))$? Once again by Observation 7.2 a set S_j of net-weight less than $\frac{W}{3} - \text{wt}(T_{i-1})$ can be picked only at a net-sparsity of at least net-sparsity$_{T_{i-1}}(D_i)$. On the other hand Observation 7.1 says that we could be picking a set of net-weight as large as $(W - \text{wt}(T_{i-1}))/2$ for a net-cost that is only strictly larger than net-cost$_{T_{i-1}}(D_i)$. Since

$$\begin{aligned}\frac{\text{net-cost}_{T_{i-1}}(D_i)}{(W - \text{wt}(T_{i-1}))/2} &< \frac{\text{net-cost}_{T_{i-1}}(D_i)}{\text{net-weight}_{T_{i-1}}(D_i)} \\ &= \text{net-sparsity}_{T_{i-1}}(D_i)\end{aligned}$$

the cheapest possible way of picking sets is to pick sets of net-weight $\frac{W - \text{wt}(T_{i-1})}{2}$ and incur a net-cost strictly larger than net-cost$_{T_{i-1}}(D_i)$ for each set picked. Since we need to pick a net-weight of at least $-z(W - \text{wt}(T_{i-1}))$, we should pick at least $-2z$ such sets. Since $z \leq -1$, the total net-cost of these sets is strictly larger than net-cost$_{T_{i-1}}(D_i)$ contradicting the upper bound derived at the beginning of this case.

We have thus established that there exists a set $S_j \in \mathcal{S}$ which meets the weight requirement for a $\square$ and has cost no more than cost(T_i). Further, S_j corresponds to a directed simple cycle in $G_i^{\vec{D}}$. Our procedure for finding a $\square$ returns a set of cost less than the cost of any set that meets the weight requirement for a $\square$ and corresponds to a directed simple cycle in $G^{\vec{D}}$. Hence

$$\text{cost}(B_i) \leq \text{cost}(S_j) \leq \text{cost}(T_i)$$

Therefore

$$\text{cost}(T_{i-1} \cup B_i) \leq \text{cost}(T_{i-1}) + \text{cost}(B_i) \leq \text{cost}(T_{i-1}) + \text{cost}(T_i) \leq 2 \cdot \text{cost}(\text{OPT})$$

where the last inequality follows from Lemma 4.3 and the fact that $i \leq k - 1$. ■

For each w in the range $[0..\frac{W}{3} - \text{wt}(T_{i-1})] \cup [-W + \text{wt}(T_{i-1})..\frac{-2W}{3}]$, it suffices to find in G_i a directed cycle (not necessarily simple) with minimum $c(C) - t_i(C)$ among all directed cycles C with $w_i(C) = w$. If for some w the shortest cycle is not simple we discard the cycle and do not consider that w for the purpose of computing the $\bullet$. If in the process we discard the cycle with $w_i(C) = w_i(C^*)$ then by the above theorem the separator found in this iteration is within twice the optimum. Else, we obtain a simple cycle C with $w_i(C) = w_i(C^*)$ and the set corresponding to this cycle is a $\bullet$.

Finally we have to deal with the case that there are negative cycles in H_i. A negative cycle in H_i corresponds to a cycle C in $G_i^{\vec{D}}$ such that $w_i(C) = 0$ and $c(C) - t_i(C) < 0$.

Lemma 7.2 *If C is a cycle in $G_i^{\vec{D}}$ such that $w_i(C) = 0$ and $c(C) - t_i(C) < 0$ then the separator found in this iteration has cost at most $2 \cdot \text{cost(OPT)}$.*

Proof: The proof of this lemma is along the lines of Theorem 7.1. We decompose C into a collection $\mathcal{C}$ of directed simple cycles. For $C_j \in \mathcal{C}$ let S_j be the side of the cycle with smaller net-weight and let $\mathcal{S}$ be the collection of sets S_j, one for each $C_j \in \mathcal{C}$. Using the decomposition rule we have

$$z(W - \text{wt}(T_{i-1})) + \sum_{S_j \in \mathcal{S}} y_j \cdot \text{net-weight}_{T_{i-1}}(S_j) = 0 \quad (3)$$

$$-z \cdot \text{cost}(T_{i-1}) + \sum_{S_j \in \mathcal{S}} (\text{cost}(S_j) - y_j \cdot \text{trapped-cost}_{T_{i-1}}(S_j)) < 0 \quad (4)$$

For contradiction we assume that every $S_j \in \mathcal{S}$ which satisfies the weight requirement for a $\square$ has cost more than $\text{cost}(T_i)$. By Observation 7.3 every set $S_j \in \mathcal{S}$ has non-negative net-cost. Hence equation 4 yields

$$\begin{aligned} z \cdot \text{cost}(T_{i-1}) &> \sum_{S_j \in \mathcal{S}} (\text{cost}(S_j) - y_j \cdot \text{trapped-cost}_{T_{i-1}}(S_j)) \\ &\geq \sum_{S_j \in \mathcal{S}^-} (\text{cost}(S_j) + \text{trapped-cost}_{T_{i-1}}(S_j)) \\ &\geq \sum_{S_j \in \mathcal{S}^-} \text{cost}(S_j) \end{aligned}$$

which implies that $z > 0$.

A lower bound on the total net-weight of sets in $\mathcal{S}^-$ can be obtained using equation 3.

$$\begin{aligned} z(W - \text{wt}(T_{i-1})) &= -\sum_{S_j \in \mathcal{S}} y_j \cdot \text{net-weight}_{T_{i-1}}(S_j) \\ &\leq \sum_{S_j \in \mathcal{S}^-} \text{net-weight}_{T_{i-1}}(S_j) \end{aligned}$$

Beyond this point the argument is almost identical to that for the case when $z > 0$ in the proof of Theorem 7.1. This contradicts our assumption that every set $S_j \in \mathcal{S}$ which meets the weight requirement of a $\square$ has cost more than $\text{cost}(T_i)$. As in the proof of Theorem 7.1, the $\square$ picked in this iteration has cost at most $\text{cost}(T_i)$ and hence the cost of the separator output is at most $2 \cdot \text{cost(OPT)}$. ■

By Lemma 7.2, we need to compute shortest paths in graph H_i only if it has no negative cycles.

8 Finding $\square$ sets

We will use Rao's algorithm [8, 9] to find a $\square$ set. Let $w : V \to Z^+$ be a weight function on the vertices of G such that $w(v) = 0$ if $v \in T_{i-1}$ and $wt(v)$ otherwise. If B_i is a $\square$ in the i$^{\text{th}}$

iteration, then $(B_i, \overline{B_i})$ is a b-balanced bond in G when the weights on the vertices are given by w and $b = \frac{W/3-\mathrm{wt}(T_{i-1})}{W-\mathrm{wt}(T_{i-1})}$. So, to find the $\Box$ we need to find the minimum-cost simple cycle in G^D which corresponds to a b-balanced bond in G.

Rao [8, 9] gives an algorithm for finding a minimum cost b-balanced *connected circuit* in G^D. A connected circuit in G^D is a set of cycles in G^D connected by an acyclic set of paths. Intuitively, a connected circuit can be viewed as a simple cycle with 'pinched' portions corresponding to the paths. The cost of a connected circuit is defined to be the cost of the closed walk that goes through each pinched portion twice and each cycle once. A connected circuit in G^D defines a simple cut in G; the vertices corresponding to faces included in the cycles of the connected circuit form one side of the cut. A connected-circuit is b-balanced if the cut corresponding to it is b-balanced. Note that the cost of the cut defined by a connected circuit is just the sum of the costs of the cycles in it. Hence, the definition of cost of a connected circuit is an upper bound on the cost of the underlying cut; the two are equal if the connected circuit is a simple cycle.

Notice that for a $\Box$ we do not really need to find a minimum-cost b-balanced bond in G; any cut that is b-balanced and has cost no more than the minimum-cost b-balanced bond will serve our purpose. Hence we can use Rao's algorithm to find a $\Box$. The total time taken by Rao's algorithm to obtain an optimal b-balanced connected circuit cut is $O(n^2W)$.

9 Running time

Clearly, the algorithm terminates in at most n iterations. The running time of each iteration is dominated by the time to find a $\bullet$. In each iteration, computing a $\bullet$ involves $O(n)$ single source shortest path computations in a graph with $O(n^2W)$ vertices and $O(n^2W)$ edges; the edge-lengths may be negative. This requires $O(n^4W^2 \log nW)$ time using Johnson's extension of the all-pairs shortest path algorithm of Floyd and Warshall. Hence, the total running time of the Dot-Box Algorithm is $O(n^5W^2 \log nW)$. This is polynomial if W is polynomially bounded.

Theorem 9.1 *The* Dot-Box Algorithm *finds an edge-separator in a planar graph of cost within twice the optimum, and runs in time $O(n^5W^2 \log nW)$, where W is the sum of weights of the vertices.*

10 Dealing with binary weights

The size of the graph in which we compute shortest paths (and hence the running time of the Dot-Box Algorithm) depends on the sum of the vertex weights. Using scaling we can make our algorithm strongly polynomial; however, the resulting algorithm is a pseudo-approximation algorithm in the sense that it compares the cut obtained with an optimal cut having a better balance. Finally, we use our scaling ideas to extend the algorithm of Park and Phillips into a fully polynomial approximation scheme for finding sparsest cuts in planar graphs with vertex weights given in binary, thereby settling their open problem.

10.1 b-balanced cut

Let us scale the vertex weights so that the sum of the weights is no more than αn ($\alpha > 1$). This can be done by defining a new weight function $\hat{wt} : V \to \mathbf{Z}^+$ as

$$\hat{wt}(v) = \lfloor \frac{\mathrm{wt}(v)}{W} \cdot \alpha n \rfloor$$

The process of obtaining the new weights can be viewed as a two step process: first we scale the weights by a constant factor $\frac{\alpha n}{W}$ and then truncate. The first step does not affect the balance of any cut since all vertex weights are scaled by the same factor. However, the second step could affect the balance of a cut. Thus a cut $(S, \overline{S})$ with balance b under the weight function wt might have a worse balance under $\hat{wt}$ since all vertices on the side with smaller weight might have their weights truncated. However, the total loss in weight due to truncations is at most n (1 for each vertex). The balance would be worst when the total weight stays at αn (not drop by the truncations) and then the loss of weight of the smaller side is a $1/\alpha$ fraction of the total weight. Thus the balance of the cut $(S, \overline{S})$ under $\hat{wt}$ might be $b - 1/\alpha$ but no worse.

Similarly a cut $(S, \overline{S})$ with balance $\hat{b}$ under $\hat{wt}$ might have a worse balance under wt. It is easy to show by similar a argument that under wt, $(S, \overline{S})$ has a balance no worse than $\hat{b} - 1/\alpha$.

Let OPT denote the cost of the optimum b-balanced cut under the weight assignment wt. Since this cut might be $(b - 1/\alpha)$-balanced under $\hat{wt}$, we use the Dot-Box Algorithm to find a $(b - 1/\alpha)$-balanced cut of cost within 2OPT. The cut returned by our algorithm, while being $(b - 1/\alpha)$-balanced under $\hat{wt}$, might only be $(b-2/\alpha)$-balanced under wt. Thus we obtain a $(b-2/\alpha)$-balanced cut of cost within twice the optimum b-balanced cut.

Theorem 10.1 *For $\alpha > 2/b$, the* Dot-Box Algorithm, *with weight scaling, finds a $(b - 2/\alpha)$-balanced cut in a planar graph of cost within twice the cost of an optimum b-balanced cut for $b \leq \frac{1}{3}$, in $O(\alpha^2 n^7 \log n\alpha)$ time.*

10.2 Sparsest cut

Assume that vertex weights in planar graph G are given in binary. Let 2^p be the least power of 2 that bounds the weight of each vertex and let W be the sum of weights of all vertices. We will construct $p + 1$ copies of G, $G_i, 0 \leq i \leq p$ each having the same edge costs as G. In G_i, vertex weights are assigned as follows: Let α be a positive integer; α determines the approximation guarantee as described below. Vertices having weights in the range $[2^i, 2^{i+2\log n+\alpha+2}]$ are assigned their original weight, those having weight $< 2^i$ are assigned weight 0, i.e., they can be deleted from the graph, and those having weight $> 2^{i+2\log n+\alpha+2}$ are assigned weight $2^{i+2\log n+\alpha+2}$. The sparsest cut is computed in each of these graphs using the algorithm of Park and Phillips. For the purpose of this computation, the weights of all vertices in G_i are divided by 2^i; notice that this leaves the weights integral, and the total weight of vertices is at most $O(2^\alpha n^3)$. The running time of [7] is $O(n^2 w \log nw)$, where w is the total weight of vertices in the graph. So, this computation takes time $O(\alpha 2^\alpha n^5 \log W \log n)$, which is polynomial in the size of the input, for fixed α. The sparsity of the $p + 1$ cuts so obtained is computed in the original graph, and the sparsest one is chosen.

Let $(S, \overline{S})$ be an optimal sparsest cut in G, and let S be its lighter side. Let the weight of S be t, and let q be the weight of the heaviest vertex in S. Pick the smallest integer i such that $2^{i+\log n+\alpha+1} \geq q$.

Lemma 10.2 *The cut found in G_i has cost at most that of $(S, \overline{S})$, and weight at least $(1 - \frac{1}{2^\alpha})t$. Therefore this cut has sparsity within a factor of $\frac{1}{1-\frac{1}{2^\alpha}}$ of the sparsity of $(S, \overline{S})$.*

Proof: The algorithm of Park and Phillips searches for the cheapest cut of each weight, for all choices of weight between 0 and half the weight of the given graph. It then outputs the sparsest of these cuts.

First notice that for the choice of i given above, the weight of S in G_i, say t', satisfies $(1 - \frac{1}{2^\alpha})t \leq t' \leq t$. Indeed, any set of vertices whose weights are at most $2^{i+2\log n+\alpha+2}$ in G satisfies that its weight drops by a factor of at most $(1-\frac{1}{2^\alpha})$ in G_i. On the other hand, any set of vertices containing a vertex having weight $> 2^{i+2\log n+\alpha+2}$ in G has weight exceeding t in G_i. Therefore, the cut found in G_i for the target weight of t' satisfies the conditions of the lemma. ■

For a given choice of $\delta > 0$, pick the smallest positive integer α so that $1 + \delta \geq \frac{1}{(1-\frac{1}{2^\alpha})}$. Then, we get the following:

Theorem 10.3 *The above algorithm gives a fully polynomial time approximation scheme for the minimum-sparsity cut problem in planar graphs. For each $\delta > 0$, this algorithm finds a cut of sparsity within a factor of $(1 + \delta)$ of the optimal in $O(\frac{1}{\delta}\log(\frac{1}{\delta})n^5 \log W \log n)$ time.*

11 Open problems

Several open problems remain:

1. Is the problem of finding the cheapest b-balanced cut in planar graphs strongly NP-hard, or is there a pseudo-polynomial time algorithm for it?
2. What is the complexity of finding a minimum net-sparsity cut in planar graphs, assuming that the vertex weights are given in unary?
3. What is the complexity of finding $\bullet$ sets in planar graphs, assuming that the vertex weights are given in unary?
4. Can the algorithm given in this paper be extended to other submodular functions?
5. Can it be extended to other classes of graphs? In particular, can the notion of transfer function be extended to other classes of graphs?

References

[1] S.N. Bhatt and F.T. Leighton. A framework for solving vlsi graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984.

[2] P.W. Kasteleyn. Dimer statistics and phase transitions. *Journal of Mathematical Physics*, 4:287–293, 1963.

[3] F.T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[4] C.E. Leiserson. Area-efficient layouts (for VLSI). In *Proceedings, IEEE Symposium on Foundations of Computer Science*, 1980.

[5] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Mathematics*, 36(2):177–189, 1979.

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

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

[8] S.B. Rao. Finding near optimal separators in planar graphs. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 225–237, 1987.

[9] S.B. Rao. Faster algorithms for finding small edge cuts in planar graphs. In *Proceedings, ACM Symposium on Theory of Computing*, pages 229–240, 1992.