Finding separator cuts in planar graphs within twice the optimal

*Naveen Garg** *Huzur Saran** *Vijay V. Vazirani**

Abstract

Building on the works of Rao [9, 10] and Park and Phillips [8], we present a factor 2 approximation algorithm for the problem of finding a minimum cost b-balanced cut in planar graphs, for $b \leq \frac{1}{3}$, if the vertex weights are given in unary (using scaling, a psuedo-approximation algorithm is also presented for the case of binary vertex weights). This problem is of considerable practical significance, especially in VLSI design.

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. For $b = \frac{1}{3}$, this problem has been called the *minimum separator problem*. In this paper, we present a factor 2 approximation algorithm for the problem of finding a minimum cost b-balanced cut in planar graphs, for $b \leq \frac{1}{3}$ (for this and other results stated below, unless stated otherwise, we will assume that the vertex weights are given in unary). We give examples to show that our analysis is tight.

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 [6], 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, 5, 4, 7]. Several of these applications pertain to planar graphs, the most important being circuit partitioning in VLSI design.

Until recently, no approximation algorithms were known for b-balanced cuts in general graphs. For planar graphs, Rao [9, 10] had given an $O(\log n)$ factor approximation algorithm. Recently, this situation has been rectified by the remarkable result of Chung and Yau [2], who have given a constant factor approximation algorithm for general graphs. The exact constant has not been calculated, but it appears to be around 10^6. The techniques used in these two results are quite different, and our work is based on Rao's approach.

The *quotient cost* of a cut is defined to be the quotient of the cost of the cut and the weight on its smaller side. A cut achieving minimum quotient cost in the graph is called a *flux cut*, and the quotient cost of such a cut is called the *flux* of the graph. Rao [10] gave a factor 1.5 algorithm for the problem of finding a flux cut in planar graphs, and recently Park and Phillips [8] showed that this problem is in P. A flux cut limits multicommodity flow in the same way that a min cut limits max flow. Leighton and Rao [5] derive an approximate max-flow min-cut theorem for uniform multicommodity flow, and in the process, they give an $O(\log n)$ factor algorithm for finding a flux 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 a constant factor ($O(\log n)$ factor) of the optimal b'-balanced cut for $b < b'$, and $b \leq \frac{1}{3}$ [9, 10, 5]. For instance, using the Park-Phillips algorithm, this gives a $\frac{1}{3}$-balance 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 be arbitrarily more costly than the best $\frac{1}{3}$-balanced cut.

We use several ideas from the important works of Rao [9, 10], and Park and Phillips [8]. Our algorithm also iteratively accumulates "sparse" cuts in the graph. One of the novel aspects of our work is the specific definition of "sparsity," that takes into consideration the cost and weight of cuts already picked. However, picking such cuts greedily may lead to a high cost b-balanced cut. We get around this by modifying the greedy strategy so that in each iteration it looks not only for a "sparse" cut, but also a low cost completion of the currently accumulated cut. We find "sparse" cuts by extending the work of Park and Phillips to our definition of sparsity. One of the ideas here is a method of assigning weights to the edges of a planar graph, so that the sum on any cycle is equal to the weights of the faces enclosed by the cycle. Such an idea has been used in the past by Kasteleyn [3], for computing the number of perfect matchings in a planar graph in polynomial time. For finding completions, we use Rao's algorithm.

Park and Phillips have shown that the problems of finding a flux cut and a b-balanced cut in planar graphs are weakly NP-complete, i.e., these problems are NP-complete if the vertex weights are given in binary. Park and Phillips have given a pseudo-polynomial time algorithm for finding flux cuts, on the other hand it is not known whether the b-balanced cut problem in planar graphs is strongly NP-complete or there is a pseudo-polynomial time algorithm for it. Park and Phillips also leave open the question of finding a fully polynomial approximation scheme for finding flux cuts in planar graphs. Using scaling, we give such an algorithm. Further, using this idea we give an algorithm to find a b-balanced cut that is within twice the best $(b + \frac{1}{n})$-balanced

*Department of Computer Science & Engineering, Indian Institute of Technology, New Delhi 110016, India.

0272-5428/94 $04.00 © 1994 IEEE

cut, for $b < \frac{1}{3}$ if the vertex weights are given in binary.

In this paper, we shall present our algorithm only for the case $b = \frac{1}{3}$; the ideas extend in a straight-forward manner for $b < \frac{1}{3}$. Here is an outline of the rest of the paper: In Section 3 we introduce one of our notions of sparsity: *true-net-sparsity*, and we give a high-level description of our algorithm under the assumption that we can obtain the "best" *true-net-sparsity* cut in each iteration. In Section 4 we introduce a weaker notion of sparsity, called *net-sparsity*, and show that because of the manner in which our modified greedy schema is organized, it suffices to obtain the "best" *net-sparsity* cut in each iteration. In this section, we also show that our algorithm achieves an approximation guarantee of 2. In Section 5 we show how to find minimum *net-sparsity* cuts; this is again made possible because of the manner in which our high-level algorithm is organized. In Section 6, we show how to find the best completion of the current cut. So far we were assuming that the costs and weights are given in unary. In Section 7, we first remove this assumption on costs, by scaling. Then, we remove the assumption on weights, but as stated above, this only results in a pseudo-approximation algorithm.

2 Preliminaries

Let $G = (V, E)$ be a connected undirected planar graph, with an edge cost function $cost : E \rightarrow \mathbf{Z}^+$, and a vertex weight function $wt : V \rightarrow \mathbf{Z}^+$. Let W be the sum of weights of all vertices in G. The cost and weight of subsets of E and V respectively are defined in the obvious manner. 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 $\overline{S}$ and we denote this cut by $\bigtriangledown(S)$. The cost of this cut, $cost(S)$, is the sum of costs of all edges in the cut.

Let G^D be the dual of G. Then, the faces of G^D correspond to the vertices of G, and inherit the same weight function. Also, the edges of G^D correspond to the edges of G, and inherit the same cost function.

A simple cycle in G^D partitions V into two sets, the sets lying on either side of this cycle, say S and $\overline{S}$. The cut $\bigtriangledown(S)$ defined in this manner will be called a *simple cycle cut*. Such a cut has the property that the induced subgraphs on both S and $\overline{S}$ are connected. Cut $\bigtriangledown(S)$ is said to be a *simple cut* if the subgraph induced on $\overline{S}$ is connected. Such a cut is defined by the disjoint union of simple cycle cuts.

Let OPT denote the optimal separator cut in G. It is easy to show that OPT must be a simple cut, and moreover, if it is not a simple cycle cut, then each simple cycle cut defining it has weight $< \frac{W}{3}$. This is true even for general graphs, as shown below.

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

Proof: Let $(S, \overline{S})$ be a minimum cost separator cut in G. Suppose to the contrary, $(S, \overline{S})$ does not satisfy the conditions of the lemma. Suppose both S and $\overline{S}$ are not connected. If some component has weight $\geq \frac{W}{3}$, then, by keeping this component on one side and moving the rest of the components to the other side, we have a cheaper separator cut, a contradiction. If each component has weight $< \frac{W}{3}$, then, by swapping components appropriately, we can ensure that each side still has weight $\geq \frac{W}{3}$, thereby getting a cheaper separator. Finally, if one side, say $\overline{S}$, is connected and there is a connected component in S having weight $\geq \frac{W}{3}$, then moving all the other components to $\overline{S}$, we get a cheaper separator, a contradiction. ■

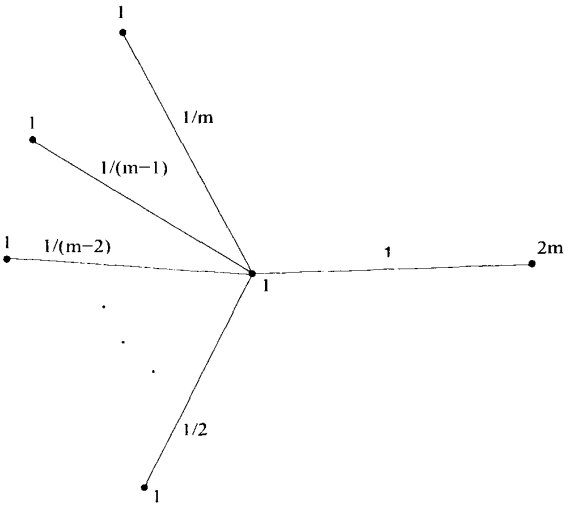

Figure 1: Graph with vertex weights and edge costs showing how sparsity increases. Here $m = \frac{n}{3}$.

3 Overview of algorithm using true-net-sparsity

The usual definition of *sparsity* of a cut is the quotient of the cost of the cut and the weight on the smaller side, i.e.,

$$sparsity(S) = \frac{cost(S)}{\min(wt(S), wt(\overline{S}))}$$

A natural approach to finding good separator cuts is to repeatedly find a sparsest cut and remove its smaller side from the graph; eventually reporting the union of the removed vertices. However, in the worst case, the sparsity available in the remaining graph may keep increasing, because the picked vertices may always come from the smaller side of the optimal separator cut. 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 $\frac{1}{2}$. An obvious way out is to "shrink" the picked vertices into one vertex whose weight is the combined weight of the shrunk vertices, and continue the search in this graph (we will always pick simple cycle cuts, so the shrunk graph will also be planar). The problem now is that our picked pieces may be nested, and we will be charged the cost of picking each of them, rather than the cost of only the last set. We get around this by introducing the notion of *true-net-cost*, which compensates for the cost of previous shrinkings.

Let $\bigtriangledown(T) = \bigtriangledown(C_1 \cup C_2 \ldots \cup C_k)$ be a simple cut in G^D, where C_i's are the connected components of T. Then, G^D_T is obtained from G^D by "shrinking" the C_i's i.e., by merging the faces corresponding to the vertices in C_i into a single

face whose weight is the sum of the weights of the faces merged. Let $\nabla(S)$ be a simple cycle cut in G_T^D. Define

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

Further, define

$$\textit{true-net-wt}_T(S) = wt(T \cup S) - wt(T)$$

and

$$\textit{true-net-sparsity}_T(S) = \frac{\textit{true-net-cost}_T(S)}{\min(\textit{true-net-wt}_T(S), \textit{true-net-wt}_T(\bar{S}))}$$

Our algorithm incorporates one more idea. To motivate it, let us give an example to show that it is not sufficient to just keep picking cuts of minimum true-net-sparsity: suppose the optimal separator is $T_1 \cup T_2$, where T_1 is a very sparse simple cycle cut of weight $\frac{W}{3} - \epsilon$, and T_2 is a high sparsity simple cycle cut of weight ϵ, for a small number ϵ. Now, having picked T_1, our algorithm may pick another cut, T_3 of sparsity almost that of T_2, and weight $\frac{W}{3} - \epsilon$, and hence, the cost incurred may be arbitrarily higher than that of OPT. We get around this difficulty by modifying the greedy schema as follows: in each iteration, we not only look for the minimum true-net-sparsity simple cycle cut that keeps the total accumulated weight under $\frac{W}{3}$, but also the minimum true-net-cost simple cycle cut that together with the previously picked cut is a separator.

Let T_i be the cut picked at the beginning of the i^{th} iteration (we will show later that this will be a simple cut in G^D). Let $\nabla(S)$ be any simple cycle cut in G_{T_i}. We will say that $\nabla(S)$ is a *dot cut*, denoted as $\bullet$ *cut*, if $wt(T_i \cup S) < \frac{W}{3}$, and it is a *box cut*, denoted as, $\square$ *cut* if $\frac{W}{3} \leq wt(T_i \cup S) \leq \frac{2W}{3}$. The "best" $\bullet$ cut is the cut having minimum *true-net-sparsity* among all $\bullet$ cuts, and the "best" $\square$ cut is the cut having minimum *true-net-cost* among all $\square$ cuts in G_{T_i}.

We will show that the cuts T_i are simple cuts in G^D, and this algorithm finds a separator cut of cost within a factor of 2 of the optimal.

4 Net-sparsity suffices

We will define a weaker notion of sparsity than true-net-sparsity, called *net-sparsity*, and we will show that because of the exact manner in which our algorithm picks cuts, in each iteration, the minimum true-net-sparsity cut is also the minimum net-sparsity cut. We will show in Section 6 how to find a minimum net-sparsity $\bullet$ cut. For the $\square$ step, we will show later that a much weaker notion of cost than true-net-cost suffices.

Let $\nabla(T) = \nabla(C_1 \cup C_2 \ldots \cup C_k)$ be a simple cut in G^D. where C_i's are the connected components of T. Let $\nabla(S)$ be a simple cycle cut in G_T^D. Define $\textit{trapped-cost}_T(S)$ to be the sum of the costs of those C_i's such that $C_i \subseteq S$. We define $\textit{trapped-wt}_T(S)$ in an analogous manner, i.e.,

$$\textit{trapped-cost}_T(S) = \sum_{T_i : T_i \subseteq S} \textit{cost}(T_i)$$

$$\textit{trapped-wt}_T(S) = \sum_{T_i : T_i \subseteq S} wt(T_i)$$

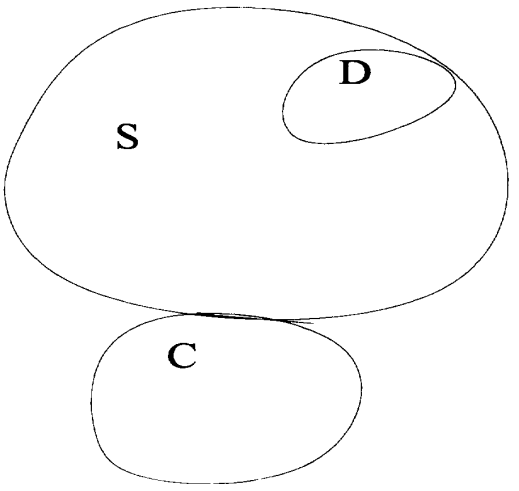

Figure 2: Simple brushing of cycle C against S.

Further, define

$$\begin{aligned} \textit{net-cost}_T(S) &= \textit{cost}(S) - \textit{trapped-cost}_T(S) \\ \textit{net-wt}_T(S) &= wt(S) - \textit{trapped-wt}_T(S) \\ \textit{net-sparsity}_T(S) &= \frac{\textit{net-cost}_T(S)}{\textit{net-wt}_T(S)} \end{aligned}$$

Notice that $\textit{true-net-wt}_T(S) = \textit{net-wt}_T(S)$. On the other hand,

$$\textit{cost}(T \cup S) \leq \textit{cost}(T) + \textit{net-cost}_T(S)$$

The reason for the inequality is that there may be a cycle in $\nabla(T)$ "brushing" against $\nabla(S)$, i.e., if for some $T_i \subseteq \bar{S}$, $\nabla(T_i)$ shares edges with $\nabla(S)$. The cost of the common edges will not appear in $\textit{cost}(T \cup S)$, but it appears in both terms on the right hand side of the inequality. brushings are illustrated in Figure 2. Here cycle C brushes against the picked cut S. Notice that since the entire cost of cycle D will be subtracted from $\textit{cost}(S)$ in $\textit{net-cost}_{T_i}(S)$, the touching of D with S is taken into consideration, and we do not call it a brushing.

Recall that $\nabla(OPT)$ is a simple cut. If $\nabla(T_i)$ is a simple cut in G^D, let $\hat{T}_i$ denote the components of T_i that are not completely contained in OPT. Then $OPT - \hat{T}_i$ is also a simple cut in G, and will be called the *leftover* OPT at the beginning of the i^{th} iteration.

Theorem 4.1 *If the* DOT-BOX ALGORITHM *does not halt in the first k iterations, and for $1 \leq i < k$,*

- $\text{net-sparsity}_{T_i}(S_i) = \text{true-net-sparsity}_{T_i}(S_i)$
- $T_{i+1} = T_i \cup S_i$ *is a simple cut,*
- *each cycle of leftover* OPT *at the beginning of the i^{th} iteration is a $\bullet$ cut*

then $\text{cost}(\nabla(T_k)) < \text{cost}(OPT)$.

Before proving the theorem, we prove:

Algorithm DOT-BOX ALGORITHM;

1. $min\text{-}sol \leftarrow \infty$, $i \leftarrow 1$, $T_1 \leftarrow \emptyset$

2. while true do

2.1. Find the best $\bullet$ and $\square$ cuts, S_i and B_i respectively.

2.2. $min\text{-}sol \leftarrow \min(min\text{-}sol, cost(T_i) + true\text{-}net\text{-}cost_{T_i}(B_i))$

2.3. $T_{i+1} \leftarrow T_i \cup S_i$, $i \leftarrow i+1$

2.4. if $min\text{-}sol < cost(T_{i+1})$ **then**

HALT

end.

Lemma 4.2 *The* net-sparsity *of S_i's is non-decreasing, $0 \leq i < k$, i.e.* net-sparsity$_{T_{i-1}}(S_{i-1}) \leq$ net-sparsity$_{T_i}(S_i)$.

Proof: For contradiction assume that $net\text{-}sparsity_{T_i}(S_i) < net\text{-}sparsity_{T_{i-1}}(S_{i-1})$. Since S_{i-1} is shrunk in G_{T_i}, either $S_{i-1} \subseteq S_i$ or $S_{i-1} \subseteq \overline{S_i}$.

1. If $S_{i-1} \not\subseteq S_i$ then

$$\begin{aligned} net\text{-}cost_{T_{i-1}}(S_i) &= net\text{-}cost_{T_i}(S_i) \\ net\text{-}wt_{T_{i-1}}(S_i) &= net\text{-}wt_{T_i}(S_i) \end{aligned}$$

and so

$$\begin{aligned} net\text{-}sparsity_{T_{i-1}}(S_i) &= net\text{-}sparsity_{T_i}(S_i) \\ &< net\text{-}sparsity_{T_{i-1}}(S_{i-1}) \end{aligned}$$

Clearly, $true\text{-}net\text{-}sparsity_{T_{i-1}}(S_i) \leq net\text{-}sparsity_{T_{i-1}}(S_i)$, and by the statement of the theorem, $net\text{-}sparsity_{T_{i-1}}(S_{i-1}) = true\text{-}net\text{-}sparsity_{T_{i-1}}(S_{i-1})$. Hence, $true\text{-}net\text{-}sparsity_{T_{i-1}}(S_i) < true\text{-}net\text{-}sparsity_{T_{i-1}}(S_{i-1})$. Further, since S_i is a $\bullet$ cut in G_{T_i}, it is also a $\bullet$ cut in $G_{T_{i-1}}$. But this contradicts the fact that S_{i-1} is the best $\bullet$ cut in $G_{T_{i-1}}$.

2. If $S_{i-1} \subseteq S_i$ we have

$$\begin{aligned} net\text{-}cost_{T_i}(S_i) &= net\text{-}cost_{T_{i-1}}(S_i) - net\text{-}cost_{T_{i-1}}(S_{i-1}) \\ net\text{-}wt_{T_i}(S_i) &= net\text{-}wt_{T_{i-1}}(S_i) - net\text{-}wt_{T_{i-1}}(S_{i-1}) \end{aligned}$$

and so by Remark 1,

$$\begin{aligned} &net\text{-}sparsity_{T_{i-1}}(S_i) \\ &< \max(net\text{-}sparsity_{T_i}(S_i), net\text{-}sparsity_{T_{i-1}}(S_{i-1})) \\ &= net\text{-}sparsity_{T_{i-1}}(S_{i-1}) \end{aligned}$$

Once again this contradicts the fact that S_{i-1} is the best $\bullet$ cut in $G_{T_{i-1}}$.

■

Remark 1 *It is easy to show that for positive real numbers a, b, c, d, if $\frac{a}{b} < \frac{c}{d}$, then, $\frac{a}{b} < \frac{a+c}{b+d} < \frac{c}{d}$. Further, let $(a_1, b_1) \cdots (a_k, b_k)$ be pairs of positive real numbers. Then, $\exists i, 1 \leq i \leq k$ such that $\frac{a_i}{b_i} \leq \frac{\sum_{j=1}^{k} a_j}{\sum_{j=1}^{k} b_j}$.*

Proof: (of Theorem 4.1) Since $\triangledown(OPT)$ and $\triangledown(T_i)$ are simple cuts in G^D, $P_i = OPT \cap T_i, 1 \leq i \leq k-1$ is a simple cut and denotes the part of OPT that is included in T_i. Let $P_k = OPT - P_{k-1}$. Since $\phi = T_1 \subset T_2 \subset T_3 \ldots \subset T_k$, $\phi = P_1 \subseteq P_2 \subseteq \ldots \subseteq P_k \subset OPT$. At each iteration we are including portions of OPT into T_i with $P_i - P_{i-1}$ being the piece of OPT included into T_i at the $(i-1)^{\text{th}}$ iteration. We can view this procedure as two processes – one picking vertices of OPT and the other picking vertices of T_k.

1. The process picking vertices of OPT picks a weight of $wt(P_i - P_{i-1}) = net\text{-}wt_{P_{i-1}}(P_i) = net\text{-}wt_{T_{i-1}}(P_i)$ at the $(i-1)^{\text{th}}$ iteration. It picks this weight at a cost of $true\text{-}net\text{-}cost_{P_{i-1}}(P_i - P_{i-1}) = true\text{-}net\text{-}cost_{P_{i-1}}(P_i) = true\text{-}net\text{-}cost_{T_{i-1}}(P_i)$ and hence the rate at which it picks weight at the $(i-1)^{\text{th}}$ iteration is equal to $true\text{-}net\text{-}sparsity_{T_{i-1}}(P_i)$.

2. The second process picks vertices of T_k, picking a weight of $wt(T_i - T_{i-1}) = net\text{-}wt_{T_{i-1}}(S_{i-1})$ in the $(i-1)^{\text{th}}$ iteration at a cost of $true\text{-}net\text{-}cost_{T_{i-1}}(T_i - T_{i-1}) = true\text{-}net\text{-}cost_{T_{i-1}}(S_{i-1})$. Hence the rate at which this process picks weight at the $(i-1)^{\text{th}}$ iteration is equal to $true\text{-}net\text{-}sparsity_{T_{i-1}}(S_{i-1})$

Notice that $P_i - P_{i-1}$ is a simple cut each of whose cycles is a $\bullet$ cut. Further, since S_{i-1} is the best $\bullet$ cut in the $(i-1)^{\text{th}}$ iteration,

$$\begin{aligned} net\text{-}sparsity_{T_{i-1}}(S_{i-1}) &= true\text{-}net\text{-}sparsity_{T_{i-1}}(S_{i-1}) \\ &\leq true\text{-}net\text{-}sparsity_{T_{i-1}}(P_i) \end{aligned}$$

Also,

$$net\text{-}wt_{T_{i-1}}(S_{i-1}) = net\text{-}wt_{T_{i-1}}(T_i) \geq net\text{-}wt_{T_{i-1}}(P_i)$$

Thus in each iteration the second process picks weight at a cheaper rate than the first process, and the rate at which the second process picks only increases with each iteration (Lemma 4.2). Furthermore, in each iteration $i \leq k-2$, the second process picks more weight than the first process, and in the last, i.e., $(k-1)^{th}$ iteration, the first process picks enough weight so that the overall weight picked by it exceeds that picked by the second process, since $wt(T_k) \leq \frac{W}{3} \leq wt(OPT)$. Hence, the total cost incurred by the second process is less than that incurred by the first. ■

Remark 2 *Notice that Theorem 4.1 continues to hold even if the last cut picked, i.e., S_{k-1} has brushings (and consequently* net-sparsity$_{T_{k-1}}(S_{k-1}) >$ true-net-sparsity$_{T_{k-1}}(S_{k-1})$*), and T_k is not a simple cut.*

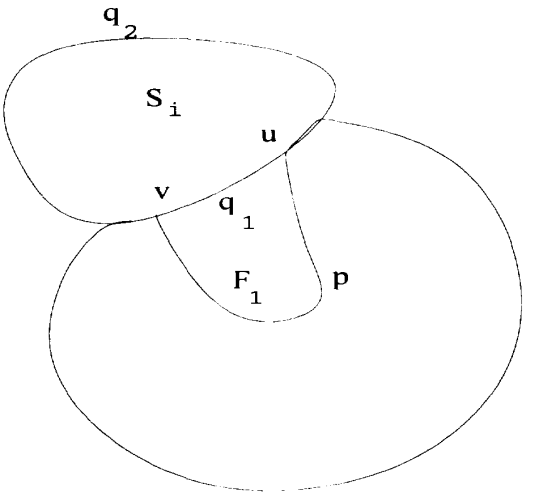

Figure 3: A non-simple brushing, and figure for Theorem 4.3

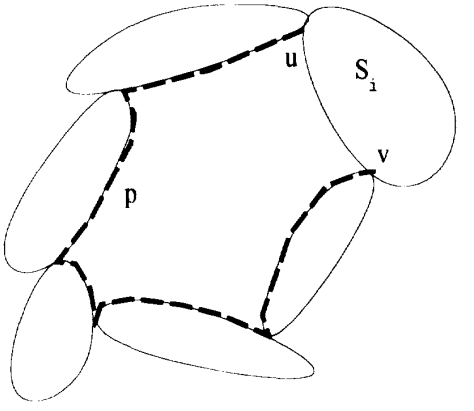

Figure 4: A degenerate example for Theorem 4.3

Let us say that a brushing of a simple cycle cut $\bigtriangledown(S)$ with a picked cycle C_i is *simple* if the cut $S \cup C_i$ is a simple cycle cut. The brushing illustrated in Figure 2 is simple. Figure 3 illustrates a non-simple brushing. We may assume w.l.o.g. that each picked cycle has no simple brushings, since we can modify any picked cycle to include C_i, without changing the picked cut. Notice that this modification does not change the true-net-sparsity of the cycle.

Theorem 4.3 *Let k be the first iteration in which the leftover* OPT *has a cycle that is a $\square$ cut. Then, for $0 \leq i < k$,*

- net-sparsity$_{T_i}(S_i)$ = true-net-sparsity$_{T_i}(S_i)$
- *$T_{i+1} = T_i \cup S_i$ is a simple cut*

Proof: We shall prove this by induction on i. This is trivially true for $i = 0$, since there are no shrinkings so far. Assuming the statement for $i-1$, let us prove it for i. First observe that $cost(T_i \cup S_i) < cost(OPT)$, by Theorem 4.1 and Remark 2. Suppose $net\text{-}sparsity_{T_i}(S_i) > true\text{-}net\text{-}sparsity_{T_i}(S_i)$. Then there must be a picked cycle brushing against S_i in a non-simple manner. Split each brushing cycle into *segments*, i.e., simple paths that start and end at vertices of S_i, and are internally disjoint from S_i. Consider such a segment p. It partitions S_i into two paths, q_1 and q_2 such that $F_1 = p \cup q_1$ and $F_2 = p \cup q_2$ are simple cycle cuts (see Figure 3). Let F_1 be the cycle that does not enclose S_i. We will call F_1 the *hole trapped by p*. Pick segment p such that the hole trapped by it is minimal by containment. Clearly, there is no brushing along path q_1 of S_i. There are three cases:

1. $wt(F_1) > \frac{2W}{3}$:
   We claim that $true\text{-}net\text{-}cost_{T_i}(\overline{F_1}) < true\text{-}net\text{-}cost_{T_i}(S_i)$, since the boundary of $T_i \cup \overline{F_1}$ is a proper subset of the boundary of $T_i \cup S_i$. Further, $net\text{-}wt_{T_i}(\overline{F_1}) \geq net\text{-}wt_{T_i}(S_i)$ since $S_i \subseteq \overline{F_1}$. Now, if $\overline{F_1}$ is a $\bullet$ cut, then it is sparser than S_1, a contradiction. If $\overline{F_1}$ is a $\square$ cut, then $cost(T_{i-1} \cup \overline{F_1}) < cost(T_i \cup S_i) < cost(OPT)$, a contradiction.
2. $\frac{2W}{3} \geq wt(F_1) \geq \frac{W}{3}$:
   Since the boundary of F_1 is a subset of the boundary of $T_i \cup S_i$, $cost(F_1) < cost(T_i \cup S_i) < cost(OPT)$, a contradiction, since F_1 is a separator cut.
3. $wt(F_1) < \frac{W}{3}$:
   Notice that the boundary of $S_i \cup F_1$ consists of $q_2 \cup p$, whereas that of S_i consists of $q_2 \cup q_1$. Since p is also the boundary of a cycle in T_i, $true\text{-}net\text{-}cost_{T_i}(S_i \cup F_1) < true\text{-}net\text{-}cost_{T_i}(S_i)$. Also, $net\text{-}wt_{T_i}(S_i \cup F_1) \geq net\text{-}wt_{T_i}(S_i)$. Now we obtain a contradiction as in case 1.

Finally, we show that $\bigtriangledown(T_{i+1})$ is a simple cut. Suppose not. Since S_i does not brush against any cycle of T_i, the only possibility is degeneracies of the type shown in Figure 4, where the cycles of T_i and S_i form a "necklace". As shown in the figure, one can still find a segment p in T_i, and conduct the same argument as above to derive a contradiction. ■

Theorems 4.1 and 4.3 give:

Corollary 4.4 *If k is the first iteration in which the leftover* OPT *has a cycle that is a $\square$ cut, then* cost(T_k) < cost (OPT).

Notice that the *true-net-cost* of the $\square$ cycle in leftover OPT is $\leq cost(OPT)$. Consider the following weaker notion of *cost*: set the costs of the edges on the boundary of T_k to 0, and keep the other edge costs of $G^D_{T_k}$ as before. Call the costs of cycles under this measure *simple-net-cost*. Notice that the *simple-net-cost* of the $\square$ cycle in leftover OPT is $\leq cost(OPT)$. In fact, we will find $\square$ cuts under this measure of cost, as shown in Section 6. Further, our algorithm will not halt before iteration k: if it does halt after the i^{th} iteration, then the cost of the solution obtained (in step 2.4 of the algorithm) $\leq cost(T_{i+1}) < cost(OPT)$, a contradiction. Since the solution obtained by the algorithm in the k^{th} iteration is within twice the optimal, and the algorithm returns the overall minimum solution found, we have:

Theorem 4.5 *The separator returned by the* DOT-BOX ALGORITHM *has cost at most twice that of OPT.*

We need to incorporate one more change in the algorithm, in step 2.4, to ensure that we can find the best • cut in each iteration. In view of Corollary 4.4, this change will not affect the approximation guarantee. We give below the final algorithm:

5 Finding the best dot cuts

The central notion needed for finding good • cuts is that of a *transfer function*. Such a function was introduced by Park and Phillips [8], and can be viewed as an extension of a function given by Kasteleyn [3]. Below we give a simpler procedure for computing this function.

Let G be a connected planar graph, with a weight function on its vertices, $wt : V \to \mathbf{Z}$. Pick an embedding for G. Let G^D be the dual graph; its faces will inherit the weight function. 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 $(v \to u)$. The transfer function corresponding to wt is an anti-symmetric function, f, on the edges of $\vec{G^D}$, i.e., $f(u \to v) = -f(v \to u)$, satisfying: for the traversal of any clockwise (anticlockwise) cycle in this directed graph, the sum of the edge weights add up to the (negative of the) sum of the weights of the faces enclosed by the corresponding cycle in G^D.

The transfer function is computed as follows: pick a spanning tree in G^D, and set the corresponding edges in $\vec{G^D}$ at zero. Now, add the remaining edges of G^D in an order so that with each edge added, one face of this dual graph is completed. Notice that all the edges of this face are already set, and the weight of this face is known. So, corresponding to this edge, the two directed edges in $\vec{G^D}$ can be set; one of these edges is used in the clockwise traversal of the face, and the other in the anticlockwise traversal. Using the fact that f is antisymmetric and that any simple cycle in G_T^D is a $GF[2]$ sum of the faces contained in the cycle we obtain,

Lemma 5.1 ([8]) *The function f defined above satisfies: for the traversal of any clockwise (anticlockwise) cycle in $\vec{G^D}$, the sum of the edge weights add up to the (negative of the) sum of the weights of the faces enclosed by the corresponding cycle in G^D.*

Notice that if for a choice of values of *net-wt*, *trapped-wt*, and *trapped-cost*, we can find a minimum cost simple cycle cut in G_T^D with these parameters, then, by carrying this out for all the suitable choices of these parameters and choice for the infinite face of G_T^D, we can find the best • cut in G_T^D. All these three parameters are defined on the faces of G_T^D, and so we can compute the corresponding transfer functions. Call these w_1, w_2 and c' respectively. Actually, we will not be able to find a simple cycle in G_T^D with these parameters, but we will be able to find "walks" as described below, and extract cycles from these walks.

Next, we construct a network as follows. Each vertex of this network is a 4-tuple of the kind (v, i, j, k) where $v \in V$, the second dimension corresponds to the *net-wt*, $-W \le i \le W$, the third dimension to the *trapped-wt*, $-W \le j \le W$, and the last to the *trapped-cost*, $-U \le k \le U$ (U is the sum of the edge costs). For each directed edge $e = (u \to v)$ in $\vec{G_T^D}$ we add an edge from (u, i, j, k) to $(u, i + w_1(e), j + w_2(e), k + c'(e))$ of cost equal to $cost(e)$, for all choices of i, j, k. Now, for finding a cheapest simple cycle in $\vec{G_T^D}$ containing the edge $e = (u \to v)$ (the direction of the edge is along the clockwise traversal of the cycle) of *net-wt*, *trapped-wt*, and *trapped-cost* w, w' and t' respectively, find the shortest path from $(v, 0, 0, 0)$ to $(u, w - w_1(e), w' - w_2(e), t' - c'(e))$ in the network. Consider this path along with the edge $(u \to v)$. Notice that in general, this may not be a simple cycle, but is an Eulerian circuit in $\vec{G_T^D}$, because the network contains multiple copies corresponding to each edge of $\vec{G_T^D}$. We will call this circuit a *walk*. Define the *net-wt*, *trapped-wt* and *trapped-cost* of this walk, Q, to be w, w' and t' respectively. The time required to compute all these shortest paths is $O(n^2 N \log N)$, where N is the size of the network, $N = nUW^2$.

Decompose Q into a set of cycles; each cycle is a simple cycle cut in G_T and is denoted by (S, ψ) where S is the side of the cut that has smaller *net-wt*, and ψ is the direction of traversal of this cycle; it being $+1$ if it is traversed in the clockwise sense and -1 if it is traversed in the anti-clockwise sense. We use the notation $(S, \psi) \in Q$ to say that (S, ψ) is a cycle in the decomposition of Q.

Let $\tilde{S}$ denote the side of the cycle (S, ψ) that does not contain the infinite face. The *net-wt* of this walk is the signed sum of the *net-wt*'s of the sets $\tilde{S}$, i.e.

$$net\text{-}wt_T(Q) = \sum_{(S,\psi)\in Q} \psi \cdot net\text{-}wt_T(\tilde{S})$$

A cycle (S, ψ) is *large* if the side of the cycle that does not contain the infinite face is the larger side with respect to *net-wt*, i.e., $S \neq \tilde{S}$.

Let β be the difference in the number of large clockwise cycles and large anti-clockwise cycles then

$$net\text{-}wt_T(Q) = \beta(W - wt(T)) + \sum_A net\text{-}wt_T(C) - \sum_B net\text{-}wt_T(C),$$

where A is the set of small clockwise and large anticlockwise cycles and B is the set of small anticlockwise and large clockwise cycles in the decomposition of Q.

Claim 1 *If $\beta = 0$ then*

$$\text{net-wt}_T(Q) \le \sum_{(S,\psi)\in Q} \text{net-wt}_T(S)$$

If $\beta > 0$ then

$$\text{net-wt}_T(Q) \ge W - wt(T) - \sum_{(S,\psi)\in Q} \text{net-wt}_T(S)$$

and if $\beta < 0$ we have

$$\text{net-wt}_T(Q) \le -(W - wt(T)) + \sum_{(S,\psi)\in Q} \text{net-wt}_T(S)$$

Let $cost(Q)$ be the sum of the costs of the edges included in the walk. Thus, $cost(Q) = \sum_{(S,\psi)\in Q} cost(S)$. The *trapped-wt*, *trapped-cost* of this walk is the signed sum of the *trapped-wt*, *trapped-cost* of the sets $\tilde{S}$, $\tilde{S} : (S, \psi) \in Q$, i.e.

$$trapped\text{-}wt_T(Q) = \sum_{(S,\psi)\in Q} \psi \cdot trapped\text{-}wt_T(\tilde{S})$$

$$trapped\text{-}cost_T(Q) = \sum_{(S,\psi)\in Q} \psi \cdot trapped\text{-}cost_T(\tilde{S})$$

Algorithm DOT-BOX ALGORITHM;

1. $min\text{-}sol \leftarrow \infty$, $i \leftarrow 1$, $T_1 \leftarrow \emptyset$

2. while true do

2.1. Find the minimum *net-sparsity* $\bullet$ and the minimum *simple-net-cost* $\square$ cuts, S_i and B_i respectively.

2.2. $min\text{-}sol \leftarrow \min(min\text{-}sol, cost(T_i) + true\text{-}net\text{-}cost_{T_i}(B_i))$

2.3. $T_{i+1} \leftarrow T_i \cup S_i$, $i \leftarrow i+1$

2.4. **if** $min\text{-}sol < 2 \cdot cost(T_{i+1})$ **then** HALT

end.

Analogous to our definition of the *net-cost* of a set, we define the *net-cost* of a walk as

$$net\text{-}cost_T(Q) = cost(Q) - trapped\text{-}cost_T(Q)$$

Lemma 5.2 *If* $\beta = 0$ *then*

$$\text{net-cost}_T(Q) \geq \sum_{(S,\psi)\in Q} \text{net-cost}_T(S)$$

Proof: For a cycle $(S,\psi) \in Q$, the *trapped-cost* on the two sides of the cycle are related as

(S,ψ)	$\psi \cdot trapped\text{-}cost_T(\tilde{S},\psi)$
small clockwise cycle	$trapped\text{-}cost_T(S)$
small anticlockwise cycle	$-trapped\text{-}cost_T(S)$
large clockwise cycle	$cost(T) - trapped\text{-}cost_T(S)$
large anticlockwise cycle	$-(cost(T) - trapped\text{-}cost_T(S))$

Thus when the number of large clockwise cycles is equal to the number of large anti-clockwise cycles we have

$$\begin{aligned} trapped\text{-}cost_T(Q) &= \sum_{(S,\psi)\in Q} \psi \cdot trapped\text{-}cost_T(\tilde{S}) \\ &\leq \sum_{(S,\psi)\in Q} \psi \cdot trapped\text{-}cost_T(S) \end{aligned}$$

and so

$$\begin{aligned} net\text{-}cost_T(Q) &= cost(Q) - trapped\text{-}cost_T(Q) \\ &\geq \sum_{(S,\psi)\in Q} cost(S) - \sum_{(S,\psi)\in Q} trapped\text{-}cost_T(S) \\ &= \sum_{(S,\psi)\in Q} net\text{-}cost_T(S) \end{aligned}$$

■

Our goal is to find a "good" simple cycle cut in G_T^D. However, via our approach, we are only guaranteed to find "good" walks. Fortunately, we can prove that in any decomposition of a walk, there must be a cycle whose net-sparsity and net-cost are bounded by that of the walk. The proof of this fact is quite elaborate. This fact is analogous to a fact proven by Park and Phillips, though their setting was much simpler, since they were working with the original graph, G^D, i.e., without any shrunk faces. They showed that there must be a cycle $C \in \mathcal{C}$ whose sparsity is at most that of the walk. Define the weight of a cycle to be the weight on its smaller side. The cost of the walk is equal to the sum of the costs of the cycles in its decomposition, and its weight is the signed sum of the weight inside each cycle. In general, the weight of the walk may be larger than the sum of the weights of its cycles (since a cycle may contribute its wrong, i.e., larger, side to the weight of the walk), and so no cycle may be as sparse as the walk. However, Park and Phillips prove that this cannot be the case if the weight of the walk is at most $\frac{W}{2}$ and hence the existence of a cycle of sparsity bounded by that of the walk follows from Remark 1. Our setting is more involved because a cycle may contribute trapped-cost on its wrong side towards the trapped-cost of the walk.

5.1 Extracting cycles from walks

Since the network does not provide us with simple cycle cuts, we need to look at walks; we can however restrict our attention to such walks for which the sum of the *net-cost* and the total cost of T_i is strictly less than *min-sol*. We refer to such walks as 'good' walks. Further we shall restrict our attention to only those walks with *trapped-sparsity* $= \frac{trapped\text{-}cost}{trapped\text{-}wt}$ at most the sparsity of S_{i-1}. We can now make the following claim about these 'good' walks.

Lemma 5.3 *If* Q *is a 'good' walk then for every cycle* $(S,\psi) \in Q$, $\text{net-cost}_{T_i}(S) > 0$.

Proof: Let i be the first iteration for which the statement of the lemma is not true. Clearly $wt(S) \leq \frac{2W}{3}$.

If $wt(S) \geq \frac{W}{3}$, then S is a simple cycle separator of cost given by

$$cost(S) \leq cost(T_i) + net\text{-}cost_{T_i}(S) \leq cost(T_i)$$

and hence we would have halted at the previous iteration; a contradiction.

We therefore assume that $wt(S) \leq \frac{W}{3}$. Since $net\text{-}cost_{T_{i-1}}(S) \geq 0$, it should be the case that $S_{i-1} \subset S$. Therefore,

$$net\text{-}cost_{T_i}(S) = net\text{-}cost_{T_{i-1}}(S) - net\text{-}cost_{T_{i-1}}(S_{i-1})$$

and so $net\text{-}cost_{T_{i-1}}(S_{i-1}) > net\text{-}cost_{T_{i-1}}(S)$. Since $S_{i-1} \subset S$,

$$net\text{-}wt_{T_{i-1}}(S_{i-1}) \leq net\text{-}wt_{T_{i-1}}(S)$$

and hence

$$net\text{-}sparsity_{T_{i-1}}(S_{i-1}) > net\text{-}sparsity_{T_{i-1}}(S)$$

Since S_{i-1} is the best $\bullet$ cut at the $(i-1)^{\text{th}}$ iteration, S could not have been a $\bullet$ cut at this iteration. Hence S was a $\square$ cut at the $(i-1)^{\text{th}}$ iteration. But since it has a *net-cost* less than the *net-cost* of the best $\bullet$ cut, by Theorem 4.1, $cost(T_{i-1} \cup S) < cost(OPT)$, a contradiction. ■

Theorem 5.4 *A 'good' walk, Q, has a simple cycle cut S such that* net-sparsity$_{T_i}(S) \leq$ net-sparsity$_{T_i}(Q)$, *and* net-cost$_{T_i}(S) \leq$ net-cost$_{T_i}(Q)$.

Proof: The walk Q can be decomposed into cycles. Let $\nabla(S)$ be a cycle in this decomposition and let S be its smaller side, with respect to *net-wt*.

Any simple cycle cut, S, of weight more than $\frac{W}{3}$ is a simple cycle separator of cost

$$cost(S) \leq cost(Q) \leq cost(T_i) + net\text{-}cost_{T_i}(Q) < min\text{-}sol,$$

a contradiction. So, for the rest of the proof, we assume that each simple cycle cut in the walk has $net\text{-}wt_{T_i}(S) \leq wt(S) < \frac{W}{3}$ and hence each such cut is either a $\bullet$ cut or a $\square$ cut. Further, let w' be the *trapped-wt* and t the *trapped-cost* of Q.

Case 1: $\sum_S net\text{-}wt_{T_i}(S) < \frac{2W}{3}$.

We will first prove that the number of large clockwise cycles in the decomposition is equal to the number of large anticlockwise cycles. If β, the difference in the number of large clockwise and large anticlockwise cycles, is positive then by Claim 1

$$\begin{aligned} net\text{-}wt_{T_i}(Q) &\geq W - wt(T_i) - \sum_S net\text{-}wt_{T_i}(S) \\ &> W - wt(T_i) - \frac{2W}{3} \\ &= \frac{W}{3} - wt(T_i), \end{aligned}$$

contradicting the fact that Q is a $\bullet$ walk.

If $\beta < 0$, then by Claim 1

$$\begin{aligned} net\text{-}wt_{T_i}(Q) &\leq -W + wt(T_i)) + \sum_{(S,\psi)\in Q} net\text{-}wt_{T_i}(S) \\ &\leq -\frac{W}{3} + wt(T_i) \\ &< 0. \end{aligned}$$

Since the *net-wt* of the walk is positive, we have another contradiction. Hence $\beta = 0$ and therefore by Lemma 5.2

$$\sum_{(S,\psi)\in Q} net\text{-}cost_{T_i}(S) \leq net\text{-}cost_{T_i}(Q)$$

Since for all $(S,\psi) \in Q$, *net-cost* is positive we have

$$\forall (S,\psi) \in Q \quad net\text{-}cost_{T_i}(S) \leq net\text{-}cost_{T_i}(Q)$$

Further, by Claim 1

$$\sum_{(S,\psi)\in Q} net\text{-}wt_{T_i}(S) \geq net\text{-}wt_{T_i}(Q)$$

and so

$$\frac{\sum_{(S,\psi)\in Q} net\text{-}cost_{T_i}(S)}{\sum_{(S,\psi)\in Q} net\text{-}wt_{T_i}(S)} \leq net\text{-}sparsity_{T_i}(Q)$$

and so there exists a cycle $(S,\psi) \in Q$ such that

$$\begin{aligned} net\text{-}sparsity_{T_i}(S) &\leq \frac{\sum_{(S,\psi)\in Q} net\text{-}cost_{T_i}(S)}{\sum_{(S,\psi)\in Q} net\text{-}wt_{T_i}(S)} \\ &\leq net\text{-}sparsity_{T_i}(Q) \end{aligned}$$

Case 2: $\sum_{(S,\psi)\in Q} net\text{-}wt_{T_i}(S) \geq \frac{2W}{3}$.

Let C be the cycle minimizing the ratio of *cost* and *net-wt* among cuts $(S\psi) \in Q$. Then,

$$\begin{aligned} \frac{cost(C)}{net\text{-}wt_{T_i}(C)} &\leq \frac{\sum_{(S,\psi)\in Q} cost(S)}{\sum_{(S,\psi)\in Q} net\text{-}wt_{T_i}(S)} \\ &\leq \frac{cost(Q)}{\frac{2W}{3}} \\ &= \frac{net\text{-}cost_{T_i}(Q) + t}{\frac{W}{3} + \frac{W}{3}} \end{aligned}$$

Now there are two cases:

Case a: $\frac{cost(C)}{net\text{-}wt_{T_i}(C)} < \frac{t}{W/3} < \frac{t}{w'}$.

If C is a $\bullet$ cut in the previous iteration, we have a contradiction since we didn't pick the sparsest cut in the previous iteration. So, C must be a $\square$ cut in the previous iteration. Since $nwt_{T_i}(C) < \frac{W}{3}$, by the condition of this case, $cost(C) < t \leq cost(T_i)$. But then, $C \cup T_{i-1}$ was a solution available at the previous iteration, of cost $< 2 \cdot cost(T_i)$, and the algorithm would not have proceeded to the i^{th} iteration.

Case b: $\frac{cost(C)}{net\text{-}wt_{T_i}(C)} < \frac{net\text{-}cost_{T_i}(Q)}{\frac{W}{3}}$.

Since $net\text{-}wt_{T_i}(Q) < \frac{W}{3}$, and $cost(C) < net\text{-}cost_{T_i}(Q)$, we have that

$$net\text{-}sparsity_{T_i}(C) \leq \frac{cost(C)}{net\text{-}wt_{T_i}(C)} < net\text{-}sparsity_{T_i}(Q)$$

Now, C cannot be a $\square$ cut, since by Corollary 4.4, $cost(T_i) + net\text{-}cost_{T_i}(C) < cost(OPT)$. So, C is a $\bullet$ cut meeting the requirements of the theorem. ■

6 Finding the best box cuts

We will use Rao's [9, 10] algorithm to find $\square$ cuts. As stated earlier, we need to find the best $\square$ cut with respect to *simple-net-cost*, hence we set to 0 the costs of all edges that are boundaries of picked cuts in $G^D_{T_i}$. In this graph, a minimum b-balanced simple cycle cut, for $b = \frac{\frac{W}{3} - wt(T_i)}{W - wt(T_i)}$, is the best $\square$ cut.

Rao gives an algorithm for finding a b-balanced *connected circuit* cut of minimum cost. A connected circuit cut is a set of cycles in G^D connected by an acyclic set of paths. Intuitively, we view a connected circuit as a simple cycle with 'pinched' portions corresponding to the paths. In confirmity with this viewpoint, define the cost of a connected circuit cut to be the cost of the closed walk that goes through each pinched portion twice and each cycle once. Note that the real cost of the cut defined by a connected circuit is just the sum of the costs of the cycles in it. Hence, this definition overstates the cost of the underlying cut when the connected circuit is not a simple cycle.

Notice that we do not have to restrict ourselves to simple cycle cuts, and can use connected circuit cuts (provided they are cheaper) in the $\square$ step. Hence we can use Rao's algorithm to find $\square$ cuts.

A key step in Rao's work is to demonstrate that an optimal connected circuit cut is 'well behaved' with respect to a shortest path tree T rooted at some vertex in G. By 'well behaved' we mean that C visits (or crosses) each branch of T at most once and all paths in T from the root to C lie

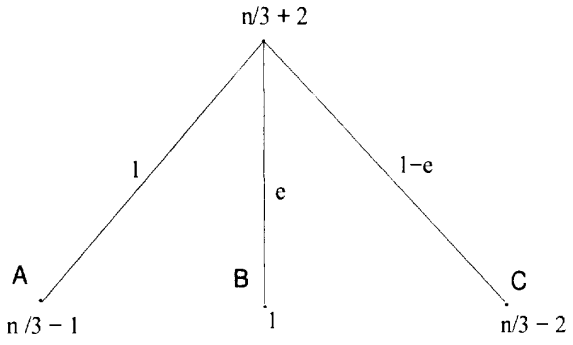

Figure 5: A tight example for our analysis. Vertex weights and edge costs are given, and e is a small constant.

on one side of C. For any vertex v on a connected circuit cut C, we say that v is *discovered* from the inside if the last edge in the path from the root to v lies inside the connected circuit C. Discovery from C, and from the exterior of C is analogously defined.

Theorem 6.1 (Rao) *Consider a planar graph G and a $b \leq 1/3$, and a node r that lies on some optimal connected circuit cut, C_{opt}. For any shortest path tree, T, of G rooted at r, there exists an optimal b-balanced connected circuit cut, C, that contains r and every node on C is discovered by a branch of T either from C or from the interior of C.*

Given a root vertex r, and a face F_r adjacent to r as the exterior face, Rao gives an algorithm based on dynamic programming to find a minimum cost b-balanced connected circuit cut (for $b \leq 1/3$) which is well-behaved with respect to the shortest path tree rooted at r. A minimum cost b-balanced connected circuit cut is obtained by trying every vertex in G as the root and every face adjacent to r as the exterior face and choosing the minimum solution. The total time taken by Rao's algorithm to obtain an optimal b-balanced connected circuit cut is $O(n^2 \min(W, U))$ where U is the total cost of G.

Theorem 6.2 *The* DOT-BOX ALGORITHM *finds a separator cut in a planar graph of cost within twice the optimal, and runs in time $O(n^3 N \log N)$, $N = nUW^2$, where U is the total cost of all edges in the graph, and W is the sum of the weights of all vertices in the graph.*

Our analysis is tight, as can be seen from the example in Figure 5. The optimal cut picks pieces A and B, and has cost $1 + e$, whereas our algorithm will first pick piece C and then A, incurring a cost of $2 - e$. Here, $e \geq \frac{4}{n}$ is a small constant.

7 Cost and weight scaling

Since one of the dimensions of the network is the *trapped-cost*, the size of the network (and hence the running time of the DOT-BOX ALGORITHM) depends upon the sum of the costs of the edges. To make this algorithm polynomial time, we scale the costs of the edges.

For each $i, 0 < i \leq \log U$, we divide the edge costs by 2^i and truncate the fractional parts. Edges of cost more than $2^{i+(\epsilon+\log n)}$ ($\epsilon > 1$ is a constant to be chosen later) are assigned an infinite cost (by identifying their end-points), Note that this operation does not destroy planarity. We now run the DOT-BOX ALGORITHM for this graph G_i.

Let i be the smallest number such that the costliest edge in the optimum solution has cost less than $2^{i+(\epsilon+\log n)}$. Then $OPT \geq 2^{i+(\epsilon+\log n)-1}$. Let G_i denote the graph with edge costs restricted to the range $(2^i, 2^{i+(\epsilon+\log n)})$. Since all edges in G of cost less than 2^i are assigned a zero cost in G_i, the solution found by the DOT-BOX ALGORITHM in G_i has lesser cost than the cost of this solution in G. Further, since the number of edges in any separator is no more than $3n$, the difference in these costs is at most

$$2^i \cdot 3n = \frac{3 \cdot 2^{i+(\epsilon+\log n)}}{2^\epsilon} \leq 3 \cdot 2^{1-\epsilon} \cdot OPT$$

and hence the solution found by the DOT-BOX ALGORITHM is of cost no more than

$$2 \cdot OPT + 3 \cdot 2^{1-\epsilon} \cdot OPT$$

Choosing ϵ as $1 + \log n$ then gives us an approximation guarantee of $2 + \frac{3}{n}$. Hence we get an approximation algorithm even if the edge-costs are given in binary:

Theorem 7.1 *The* DOT-BOX ALGORITHM, *with cost scaling, finds a separator cut in a planar graph of cost within $2 + \frac{3}{n}$ of the optimal, and runs in time $O(n^7 W^2 \log nW \log U)$, where U is the total cost of all edges in the graph, and W is the sum of the weights of all vertices in the graph.*

A simlar technique can be used for making our algorithm polynomial when the weights are given in binary. Let M be the weight of the heaviest vertex in G. Obtain G' by setting the weight of each vertex v to $\lfloor \frac{wt(v) * 2^{\log n + \epsilon}}{M} \rfloor$. We run the DOT-BOX ALGORITHM on G'. Unfortunately, this does not give a true approximation algorithm, since after truncation of weights the optimum separator cut may not have enough weight on the smaller side. The weight of the smaller side of an optimal separator cut in G is at least $M/2$. When we truncate weights, the loss of weight for any vertex is no more than $\frac{M}{2^{\log n + \epsilon}}$ and so the total loss of weight of the smaller side is at most $\frac{M}{2^\epsilon}$. Thus if we start with a $(\frac{1}{3} + \frac{1}{2^\epsilon})$-balanced cut in G we get that it must be a separator cut in G'. Also, a separator cut in G' is not necessarily a separator cut in G since when the truncated weights are added back, they may all contribute to the larger side of the cut, destroying balance. Thus this technique gives us an algorithm which finds a b-balanced cut within a factor 2 of the optimum $(b + \frac{1}{2^{\epsilon-1}})$-balanced cut for b strictly less than a third.

Theorem 7.2 *The* DOT-BOX ALGORITHM, *with cost and weight scaling, finds a b-balanced cut in a planar graph of cost within $2 + \frac{3}{n}$ of the cost of an optimal $(b + \frac{1}{n})$-balanced cut for $b \leq \frac{1}{3} - \frac{1}{n}$, in $O(n^{13} \log n \log U)$ time, where U is the total cost of all edges in the graph, and W is the sum of the weights of all vertices in the graph.*

Notice that the quality of our pseudo-approximation guarantee is stronger than that of Rao; his algorithm will find a b-balanced cut within a factor of $O(\log n)$ of the best $(b + \frac{1}{n})$-balanced cut, provided the vertex weights are given in unary.

Finally, a similar technique to the above can be used to give a fully polynomial approximation scheme for obtaining

a minimum sparsity cut in planar graphs when the vertex weights are in binary, thereby solving an open problem from [8]. Here we generate graphs G_i by truncating vertex weights to a range of $[2^i, 2^{i+2\log n+\epsilon}]$ for each value of $i, 0 \leq i \leq \lceil \log M \rceil$. Vertices having weights larger than the upper limit of the range are assigned a weight equal to the upper limit. We then use the algorithm of Park and Phillips to obtain a minimum sparsity cut in each of the graphs G_i and report the sparsest cut found. As in previous cases, the sparsity of the cut obtained can be shown to be within a factor $(1+\frac{1}{2^\epsilon})$ of the optimum cut. The running time of the algorithm of [8] is $O(n^2 W \log nW)$.

Theorem 7.3 *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 $O(\frac{1}{\delta} n^5 \log M \log n)$ time.*

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] F.R.K. Chung and S.T. Yau. A near optimal algorithm for edge separators. In *Proceedings, 26th Annual ACM Symposium on Theory of Computing*, 1994.

[3] P. W. Kasteleyn. Dimer statistics and phase transitions. *J. Math. Physics*, 4:287–293, 1963.

[4] C.E. Leiserson. Area-efficient layouts (for VLSI). In *Proceedings, 21st Annual IEEE Symposium on Foundations of Computer Science*, 1980.

[5] 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, 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 422–431, 1988.

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

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

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

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

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