TECHNICAL REPORT NO. 308

A Linear-Processor Algorithm for Finding Small Cycle Separators on Undirected Planar Graphs

by

Fang Wan

May 1990

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY

Bloomington, Indiana 47405-4101

A Linear-Processor Algorithm for Finding Small Cycle Separators on Undirected Planar Graphs

Fang Wan
Department of Computer Science
Indiana University at Bloomington, Indiana 47405
wan@iuvax.cs.indiana.edu

May 2, 1990

Abstract

A parallel algorithm is presented for finding a sublinear size cycle separator on an undirected planar graph in $O(\log^2 n)$ time on the CRCW PRAM. Let $O(n^M)$ denote the number of processors required to carry out a breadth first search on a graph of size n in $O(\log^2 n)$ time and let k denote $n^{(M-P)/M}$, where n^P is the number of processors available to our algorithm and $1 \leq P \leq M$. The size of the cycle separator is the best possible $O(\sqrt{d \cdot n})$ when the maximum face size d is large, $d \geq k$, and is $O(n^{(2M-P)/2M})$ when d is small, $d \leq k$. If the algorithm uses $O(n)$ processors, the size of the separator is the optimal $O(\sqrt{d \cdot n})$ when $d \geq n^{0.58}$ and $O(n^{0.79})$ otherwise because the best M achieved so far is 2.376 [8]. Pan and Reif's parallel nested dissection algorithm and our small cycle separator algorithm are combined into a parallel algorithm for finding a BFS tree on a planar graph in $O(\log^3 n)$ time using $O(n^{1.84})$ processors.

1 Introduction

Divide and conquer is one of the fundamental techniques used in parallel algorithms. To apply this strategy to solve a graph theory problem, we usually need to find proper separators. Given a graph G of size n and a constant $C \geq 2$, a $(\frac{1}{C}, \frac{C-1}{C})$ vertex separator on G is a vertex set which divides G into connected components no larger than $\frac{C-1}{C}n$. The most commonly used C is 3. The feasibility and the efficiency of divide and conquer techniques depend on separators' sizes and topological properties. For many parallel algorithms, the smaller separators are, the better their processor-time complexities.

Lipton and Tarjan show that there is a vertex separator of size $\sqrt{8 \cdot n}$ for every planar graph [12]. This result is *optimal* up to a multiplicative constant in the worst case. Their result was later improved to $\sqrt{6 \cdot n}$ by Djidjev [5] and then to $\frac{7}{3}\sqrt{n}$ by Gazit [6]. In many cases, vertex separators with certain topological properties are essential for developing efficient parallel algorithms. A *cycle separator* is a simple cycle whose

vertices constitute a vertex separator. If a separator has only one vertex, it is called a *cut point*. Cycle separators have proven very useful for developing parallel algorithms to solve graph theory problems such as depth first search [10,1]. In addition, finding small cycle separators is a major means of finding small vertex separators. A graph G can be triangulated and treated as an undirected graph G' so that a small cycle separator on G' is a small vertex separator on G. Not necessarily small, a cycle separator on an undirected planar graph can be found in $O(\log n)$ time using linear processors [16]. Kao showed that every strongly connected digraph has a cycle separator and a cycle separator on a planar digraph can be found by a DNC algorithm [10]. Kao and Wan showed that not every strongly connected planar graph or 2-strongly-connected planar graph has a sublinear size cycle separator or path separator [11]. Miller presented an algorithm to find a cycle separator of size $2\sqrt{d \cdot n}$ on an undirected planar graph in $O(\log n)$ time when a BFS tree on the graphs face incidence graph is part of the algorithm's input [13]. However, this algorithm's processor-time complexity is $O(n^3 \log n)$ when an $O(n^3)$ processor BFS algorithm is used and is $O(n^{2.376} \log^2 n)$ when an $O(n^{2.376})$ processor BFS algorithm is used [8]. Gazit and Miller have claimed that an $O(\sqrt{d \cdot n})$ size cycle separator on any undirected planar graph can be found in $O(\sqrt{n} \log n)$ time using $O(\sqrt{n}/\log n)$ processors [9]. While giving optimal processor-time complexity, it is not a polylogarithmic time algorithm. Gazit and Miller also presented a randomized algorithm which can find an $O(\sqrt{d \cdot n})$ size cycle separator on an undirected planar graph in $O(\log^2 n)$ time using $O(n + f^{1+\epsilon})$ processors, where f is the number of faces [7]. So it is interesting to know whether there is a linear processor DNC parallel algorithm to find small cycle separators in undirected planar graphs.

Another motivation of looking for a more efficient parallel small cycle separator algorithm is to find a more efficient parallel BFS algorithm. As mentioned above, Gazit and Miller presented a DNC BFS algorithm which has achieved the best processor-time complexity $O(n^{2.376} \log^2 n)$ so far [8]. If a family of $s(n)$ size separators on G is given, Pan and Reif have a parallel nested dissection algorithm to find a BFS tree on G in $O(I(n) \log^2 n)$ time using $s^3(n)/(I(n) \log n)$ processors, where $I(n)$ denotes parallel time of computing the sum of n values [14]. Thus an efficient parallel small separator algorithm is indispensable for making use of Pan and Reif's reduction. Gazit and Miller's randomized parallel separator algorithm [7] and Pan and Reif's reduction can be combined into a randomized parallel algorithm to perform BFS on a planar graph in $O(\log^3 n)$ time using $O(n^{1.5}/\log n)$ processors. But no known DNC BFS algorithm uses less than $O(n^{2.376})$ processors, even if it is used for planar graphs.

This paper presents a parallel algorithm which finds a sublinear size cycle separator on an undirected planar graph in $O(\log^2 n)$ deterministic time on the CRCW PRAM. The size of the cycle separator depends on how many processors are available to our algorithm and how many processors are required for matrix multiplication. We let $O(n^M)$ denote the number of processors required to multiply two $n \times n$ matrices over the ring of integers in $O(\log^2 n)$ time and let $O(n^P)$ denote the number of processors used for our algorithm, where $1 \leq P \leq M$. The size of the cycle separator is $O(\sqrt{d \cdot n})$ when the maximum face size d is larger or equal to $k = n^{(M-P)/M}$ and it is $O(n^{(2M-P)/2M})$ when $d \leq k$. If $O(n)$ processors are available to the algorithm, the size of the separator is the optimal $O(\sqrt{d \cdot n})$ when

$d \geq n^{0.58}$ or is $O(n^{0.79})$ otherwise because the best M achieved so far is 2.376 [4]. Since our algorithm can find a $O(n^{0.61})$ size vertex separator on every planar graph when $O(n^{1.84})$ processors are available, a BFS on any planar graph can be done in $O(\log^3 n)$ deterministic time using $O(n^{1.84})$ processors. This result is a substantial improvement over previous DNC BFS algorithms for planar graphs.

This algorithm is similar to that developed by Gazit and Miller [7]. Adjacent faces of an undirected graph will be repeatly united until the number of faces is small enough to apply Miller's algorithm [13] directly with the available processors. A new approach is adopted to make face uniting easier while maintaining the existence of a sublinear cycle separator.

The rest of the paper contains four parts. The first part is an outline of our small cycle separator algorithm. The second and the third parts describe and analyze two important subroutines in the algorithm. The two subroutines are interesting in their own right. The last part analyzes the algorithm as a whole and extends the result to a parallel BFS algorithm for planar graphs.

2 A General Outline

An *undirected graph* $G=(V,E)$ consists of a *vertex* set V and an *edge* set E. Each of its edges is identified by a pair of unordered vertices. A graph G is a *planar* graph if G can be drawn on a plane without any edges crossing. Symbol n represents the number of vertices, m represents the number of edges, and f represents the number of faces. Therefore, Euler's Theorem is written as $f+n-m=2$. A face is *regular* if its boundary is a simple cycle and is *irregular* otherwise. The *size* of a face is the number of vertices on the boundary of the face. The *shape* of a face is the number of vertices which are on the boundary of the face and have a degree no less than 3. An vertex set is *independent* if it has no two adjacent vertices.

An *n-weighted* graph is defined by the following inductive steps. Initially, every biconnected undirected planar graph that has n vertices is an n-weighted graph and each of its faces is assigned a weight equal to the size of the face. The rest of the n-weighted graphs are constructed from existing n-weighted graphs by uniting some of latters' faces in a way described as the followings. Two faces are eligible to be united into one face if their boundaries share at least one edge. The two faces are united into one face by first deleting all edges shared on their boundaries and then deleting all isolated vertices. The weight of the new face is the sum of the two original faces' weights minus the number of vertices shared by the two original faces. In another word, a face's weight is the face's size plus the number of vertices *merged* into the face in the different stages when the face is constructed. Thus a face's size is no larger than the face's weight. Although an n-weighted graph may have fewer than n vertices, it owes its origin to a biconnected planar graph that has n vertices. For the sake of convenience, a face is called *k-face* if its weight is no larger than k and is called *k^+-face* otherwise. A pair of k-faces is called a *k-uniting* if they can be united. A k-uniting produces either a k-face or a k^+-face no larger than $2k$. An algorithm *exhausts* k-unitings of an n-weighted graph G if it repeatly performs

k-unitings on G till G has no remaining k-uniting. We will prove that an n-weighted graph without any k-unitings has $O(n/k)$ faces. First, we will show that an n-weighted graph has at most $O(n/k)$ k^+-faces.

Lemma 1 *An n-weighted graph G cannot have more than $\frac{2(n-2)}{k-1}$ k^+-faces.*

Proof: For a specific n, we show that at least one n-weighted graph which has n vertices and has no k-face has the maximum number of faces any n-weighted graph can have. Given any n-weighted graph G, if we add as many vertices as the weight of each face minus the size of the face to the boundary of the face, we get a new graph G' from G without changing face weights. G' has as many k^+-faces as G and is an n-weighted graph which has exactly n vertices by the definition of n-weighted graphs. The k-faces of G' can be removed by deleting edges shared by a k-face and a k^+-face repeatly. The result is a graph G'' which has at least as many k^+-faces as G' and still has exactly n vertices. Since each face of G'' has at least $k+1$ edges on its boundary, $(k+1)f \leq 2m = 2(f+n-2)$ by Euler's Theorem. Because there is such a G'' for every n-weighted graph G, no n-weighted graph has more than $\frac{2(n-2)}{k-1}$ k^+-faces. ∎

If an n-weighted graph G has no k-uniting, k-faces of G turn out to be an independent set in the dual graph of G. We will use this fact to prove that G cannot have too many k-faces.

Lemma 2 *If I is an independent set of a planar graph $G=(V,E)$ and each element of I has a degree no less than 3, $|I| \leq \frac{2}{3}(n-2)$.*

Proof: Because I is an independent set and each element of I has a degree no less than 3, $3|I| \leq m$. If we remove edges between vertices in set $V-I$ so that $V-I$ becomes an independent set, neither V nor I is changed. In this case, no face size is less than 4, otherwise either I or $V-I$ will not be independent any more. By Euler's Theorem, $m \leq 2n-4$ which implies $|I| \leq \frac{2}{3}(n-2)$. ∎

We are now in a position to prove that an n-weighted planar graph only has $O(n/k)$ faces if its k-unitings are exhausted and all its small faces have shapes larger than 2.

Theorem 1 *If a biconnected n-weighted graph G has neither a k-uniting nor a k-face whose shape is less than 3, G has no more than $\frac{6(n-2)}{k-1} - 4$ faces.*

Proof: Let f_1 denote the number of k-faces of G and let f_2 denote the number of the rest, we have $f = f_1 + f_2$. Let $\bar{G} = (\bar{V}, \bar{E})$ be the dual graph of G with multiple edges being united as a single edge. It is clear that $\bar{G}$ is a planar graph and $|\bar{V}|$ is equal to f. Because G has no k-uniting and every k-face's shape is larger than 2, k-faces of G become an independent set whose elements have degree no less than 3 in $\bar{G}$. By Lemma 2, $f_1 \leq \frac{2}{3}(f_1+f_2-2)$ which leads to $f_1 \leq 2(f_2-2)$. By Lemma 1, $f_2 \leq \frac{2(n-2)}{k-1}$. So $f = f_1 + f_2 \leq 2(f_2-2) + f_2 = 3f_2 - 4 \leq \frac{6(n-2)}{k-1} - 4$. ∎

The bottle neck of previous small cycle separator algorithms has been the $O(f^M)$ processors required to perform BFS on a face incident graph. To get around it when only n^P processors are available, where $1 \leq P \leq M$, we reduce a graph's faces to a feasible number. If we let the k in Theorem 1 be $n^{\frac{M-P}{M}}$, G will have no more than

$f=6(n-2)/(n^{\frac{M-P}{M}}-1)-4=O(n^{\frac{P}{M}})$ faces. Under this circumstance, Miller's algorithms [13,8] can be used to find a small cycle separator on G in $O(\log^2 n)$ time using only $O(f^M)=O(n^P)$ processors. The size of the separator is $O(\sqrt{max\{d,k\}\cdot n})$. When $d\geq k$, the best $O(\sqrt{d\cdot n})$ size cycle separator still can be found. So Theorem 1 suggests a paradigm of finding small cycle separators on undirected planar graphs.

Algorithm 1 *Find-Simple-Cycle-Separator*
Input: *A biconnected undirected planar graph* $G=(V,E)$
Output: *A simple cycle separator of* G

1. Find a planar embedding of G by Ramachandran and Reif's algorithm [15];
2. Exhaust k-unitings of G and call the new n-weighted graph G';
3. Simplify G' into a biconnected graph G'' without increasing either face number or face size;
4. Find a cycle separator on G'' by Miller's algorithm [13,8].

In the above algorithm, Step 1 needs $O(\log n)$ time and $O(n)$ processors [15]. The planar embedding can be easily maintained or be calculated again when necessary. When k equals $n^{\frac{M-P}{M}}$, Step 4 can be done in $O(\log^2 n)$ time using $O(n^P)$ processors as mentioned before. So the processor-time complexity of Algorithm 1 mainly depends on Step 2 and Step 3 which are the main issues of the following sections.

3 Exhaustion of k-Unitings

As usual, face unitings will be conducted on dual graphs. Because we only unite k-faces, we define a class of dual graph like graphs which we call *k-dual graphs* of n-weighted graphs. A k-dual graph $\bar{G}_k=(\bar{V}_k,\bar{E}_k)$ of an n-weighted graph G is constructed from the dual graph $\bar{G}$ of G in three steps. First, each vertex of $\bar{G}$ inherits the weight of the corresponding face in G and each edge of $\bar{G}$ is assigned a weight equal to 2. Secondly and repeatly, unite two parallel edges if they form a face and let the weight of the new edge equal the total weight of the two original edges minus one. Lastly, delete all vertices weightier than k. Apparently, the total weight on the edges between two vertices of $\bar{G}_k$ equals the number of vertices shared by the two corresponding faces in G and a connected component of $\bar{G}_k$ can be contracted into one vertex whose weight equals the total weight on the vertices of the component minus the total weight on all edges between these vertices. Component contractings on $\bar{G}_k$ are equivalent to face unitings on G and can easily be imitated on G. Since we want to simulate exhausting k-unitings, no vertex weightier than $2k$ should be formed and a vertex weightier than k should be dropped from $\bar{G}_k$ when it is formed. An example of G, $\bar{G}$, and $\bar{G}_k$ is given in Fig 3. In the figure, (1) is G, (2) is $\bar{G}$, (3) is $\bar{G}_k$, (4) and (5) are what $\bar{G}_k$ and G look like after their k-unitings are exhausted. The numbers in Fig 3 are face weights, vertex weights, and edge weights accordingly.

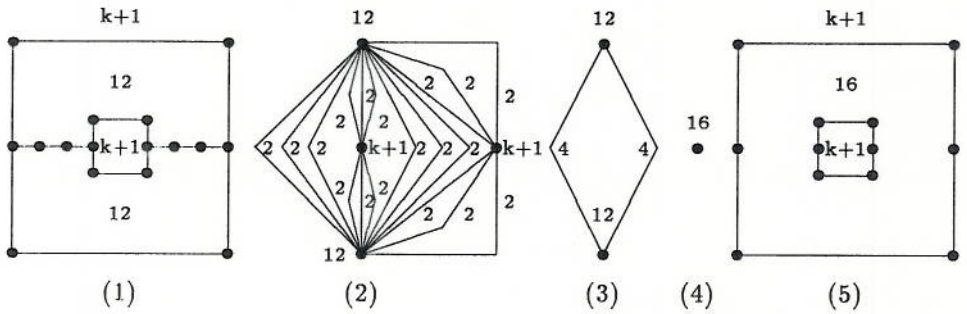

Figure 1: Example of G, $\bar{G}$, and $\bar{G}_k$.

A vertex of $\bar{G}_k$ is said to be *satiated* when a connected component containing the vertex is contracted into a vertex weightier than k or when the maximal connected component containing the vertex is contracted into a vertex no weightier than k. When a vertex is satiated, it is contracted into a vertex which either should be dropped from $\bar{G}_k$ or becomes an isolated vertex in $\bar{G}_k$. We use divide-conquer techniques to contract vertices of $\bar{G}_k$ in $\log n$ stages. In each stage, the vertices on separators of connected components of $\bar{G}_k$ will be satiated first so that $\bar{G}_k$ is divided into smaller connected components whose k-unitings can be exhausted separately and recursively.

First, we show how to satiate a cut point of a connected $\bar{G}_k$. If we label vertices of $\bar{G}_k$ from 1 to $|V_k|$ so that the cut point is labeled 1 and the top x vertices constitute a connected component for any $x \leq |V_k|$, we can satiate the cut point by uniting the first x vertices which can be contracted into a vertex weightier than k if possible or uniting the whole graph otherwise.

Algorithm 2 *Satiate-Cut-Point*
Input: *A connected k-dual graph $\bar{G}_k$ and a cut point*
Output: *Separated components of $\bar{G}_k$*

1. Find a spanning tree in $\bar{G}_k$, label vertices in preorder from the cut point, and sort vertices of $\bar{G}_k$ in ascending order by their labels;

2. Compute *increments* for all vertices, where the increment of a vertex is the weight of the vertex minus the weights on edges between the vertex and other vertices labeled smaller;

3. Find a label x, if there is one, so that the total increment of the first x vertices is larger than k and unite them. Otherwise, unite the whole $\bar{G}_k$ into one vertex.

Clearly, the above processing only leaves behind connected components smaller than one third of the original size of $\bar{G}_k$.

Lemma 3 *Algorithm Satiate-Cut-Point can be performed in $O(\log n)$ time with $O(n)$ processors.*

Proof: In Step 1, finding a spanning tree, computing depths for vertices, and sorting n numbers need $O(\log n)$ time and $O(n)$ processors when Tarjan and Vishkin's tree tour

technique and Cole's sorting algorithm are used [17,3]. Step 2 is trivial. The preorder guarantees that the top x vertices constitute a connected component. The increment of the xth vertex is exactly the difference between the weights of the vertex contracted from the top x vertices and the vertex contracted from the top $x-1$ vertices. So we can satiate the cut point in Step 3 by using doubling and binary searching techniques on increments. ∎

Algorithm 3 *Satiate-Cycle*
Input: *A k-dual graph $\bar{G}_k$ and a cycle separator*
Output: *Left components of $\bar{G}_k$*

1. Unite concatenate vertex pairs on the cycle into vertices no weightier than k till there is no more such pair left;
2. Label the vertices on the current cycle in order and unite every odd labeled vertex with its successor, if there is one.

Apparently, this algorithm either totally separates $\bar{G}_k$ or only leaves a cut point.

Lemma 4 *When algorithm Exhaust-Cycle is applied to a k-dual graph $\bar{G}_k$ and a cycle separator, all vertices on the cycle separator are satiated except at most one vertex. The algorithm needs $O(\log n)$ running time and $O(n)$ processors.*

Proof: Step 1 is easy when doubling technique is carefully used and will not break the cycle separator because no vertex weightier than k is formed. Step 2 uses $O(\log n)$ time and $O(n)$ processors to reorder and unite the cycle. In Step 2, two concatenate vertices are united into a vertex which must be weightier than k and is removed from $\bar{G}_k$, otherwise the two vertices should be united in Step 1. The last odd labeled vertex in Step 2 may be left as a cut point. ∎

Algorithm 4 *Exhaust-k-Unitings*
Input: *A planar embedding of an undirected planar graph $G = (V, E)$*
Output: *An n-weighted graph G' formed by exhausting k-unitings of G*

1. Find the k-dual graph $\bar{G}_k = (\bar{V}_k, \bar{E}_k)$ of G;
2. Go to next step if $\bar{E}_k$ is not empty, Exit otherwise;
3. Find a conventional cycle separator for every maximal connected component of $\bar{G}_k$;
4. Apply Exhaust-Cycle to all cycle separators;
5. Apply Satiate-Cut-Point to all cut points including those left in Step 3 and jump back to Step 1.

Theorem 2 *Algorithm Exhaust-k-Unitings can exhaust k-unitings of an undirected planar graph in $O(\log^2 n)$ time using $O(n)$ processors.*

Proof: In Algorithm 4, each iteration runs in $O(\log n)$ time using $O(n)$ processors. Step 1 can utilize standard doubling technique to unit multiple edges. Step 2 is trivial. In Step 3, a conventional cycle separator on an undirected planar graph can be found in $O(\log n)$ time by a linear-processor algorithm [16]. Step 4 and 5 are covered by Lemmas 3 and 4. Since each iteration reduces the size of the maximum connected component of $\bar{G}_k$ by one third, $\bar{E}_k$ will be empty within $O(\log n)$ iterations. ∎

4 Simplification of n-Weighted Graphs

In this section, G' denotes an n-weighted graph as the result of applying algorithm Exhaust-k-Unitings to a biconnected undirected planar graph G. We have to simplify irregular faces of G' without increasing either the largest face size or the number of faces. And the simplified graph should have no face weightier than $\frac{1}{3}n$. Although a similar problem was studied before [7], our simplification method is simpler because a tree representation is used to decide relations between positions of irregular faces.

A planar graph without irregular face is biconnected [2]. So a graph G' will be a biconnected graph when its irregular faces are properly simplified. In this paper, all simple cycles on the boundary of a face are called the *eigencycles* of the face and one eigencycle is called the *out-cycle* of the face while the rest are called the *in-cycles* of the face. For an out-cycle, the side of the corresponding face is its *inside*. For an in-cycle, the side of the corresponding face is its *outside*. The inside weight of an out-cycle (in-cycle) is the weight of a face formed by uniting all faces inside the out-cycle (in-cycle). A face *surrounds* another face if and only if the latter is inside one of the former's in-cycles. If two irregular faces do not surround each other, they are *independent*. The proof of the following lemma provides an efficient algorithm to simplify a group of independent irregular faces.

Lemma 5 *Given G, G', and a set of independent irregular faces of G', we can either find a cycle separator no larger than $2k$ or simplify given irregular faces in $O(\log n)$ time using $O(n)$ processors.*

Proof: Since given irregular faces are independent, the inside weights of their out-cycles can be calculated simultaneously by counting vertices inside those cycles on graph G in $O(\log n)$ time with no more than $O(n)$ processors. The weight computation has three possible outcomes.

1. If the inside weight of an out-cycle is in $[\frac{1}{3}n, \frac{2}{3}n]$, this out-cycle is a cycle separator and is at most $2k$ long.

2. If all inside weights of the out-cycles are less than $\frac{1}{3}n$, unite all faces inside each of the out-cycles into one face which is regular because its boundary is single eigencycle.

3. If the inside weight of one out-cycle is larger than $\frac{2}{3}n$, unite all faces outside this cycle into one face whose weight is less than $\frac{1}{3}n$, find the inside weights of the in-cycles of the face, and perform following case analysis.

(a) If the inside weight of an in-cycle is in $[\frac{1}{3}n, \frac{2}{3}n]$, this in-cycle is a cycle separator no longer than $2k$.

(b) If all inside weights of the in-cycles are less than $\frac{1}{3}n$, unite all faces inside each of the in-cycle into one simple face and recover all vertices and edges in this irregular face from G. Because G is biconnected and the weights of irregular faces of G' are controlled to be no weightier than $2k$, the newly recovered subgraph of G is biconnected and has at most $2k$ vertices. Therefore, any conventional cycle separator on the new n-weighted graph is not larger than $2k$ and is a cycle separator of G! Such a conventional cycle separator can be found in $O(\log 2k)$ time with $O(2k)$ processors by Shannon's algorithm [16].

(c) If the inside weight of an in-cycle is larger than $\frac{2}{3}n$, unite all faces outside this cycle into one face lighter than $\frac{1}{3}n$.

If the above processing doesn't find a small cycle separator, the independent irregular faces are either united into simple faces inside their out-cycles or united into a simple face outside an in-cycle of one of them. The processing has constant steps and each step needs no more than $O(\log n)$ time and $O(n)$ processors. ∎

In order to use the processing of Lemma 5 to simplify G' systematically, out-cycle selection is important. The following algorithm explains how to select out-cycles for G' and get a forest representation on which vertices represent irregular faces and a vertex is another vertex' descendant if and only if the former is surrounded by the later. The forest is called an *irregular-face-forest* of G'. In the following algorithm, we assume that the eigencycles of the same irregular face are disjointed, otherwise we can split joint points first.

Algorithm 5 *Find-Irregular-Face-Forest*
Input: *G and G'*
Output: *An irregular-face-forests of G'*

1. Select any eigencycle as the first out-cycle, find a spanning tree on G, select any vertex on the first out-cycle as root;

2. Modify the spanning tree by following operations and unite multiple edges into one whenever they are formed. Bypass and delete all vertices not on eigencycles of irregular faces. Unite vertices on the same eigencycle into one vertex. Now there is an one-to-one relation between vertices in current graph and eigencycles of irregular faces. Each irregular face has an eigencycle met first by a path from the root, select this eigencycle as the out-cycle of the face. Unite vertices representing the in-cycles of the same face into one vertex. Link each out-cycle vertex to the first in-cycle vertex on the path from the out-cycle vertex to the root, if there is one. Delete all edges between out-cycle vertices;

3. Unite vertices representing the same irregular face into one vertex and use current forest as an irregular-face-forest of G'.

In Fig 4(1), the boundary of an irregular face consists of cycles having the same prefix in their labels. Fig 4(2) is a branch of a spanning tree represented by the horizon cut line in

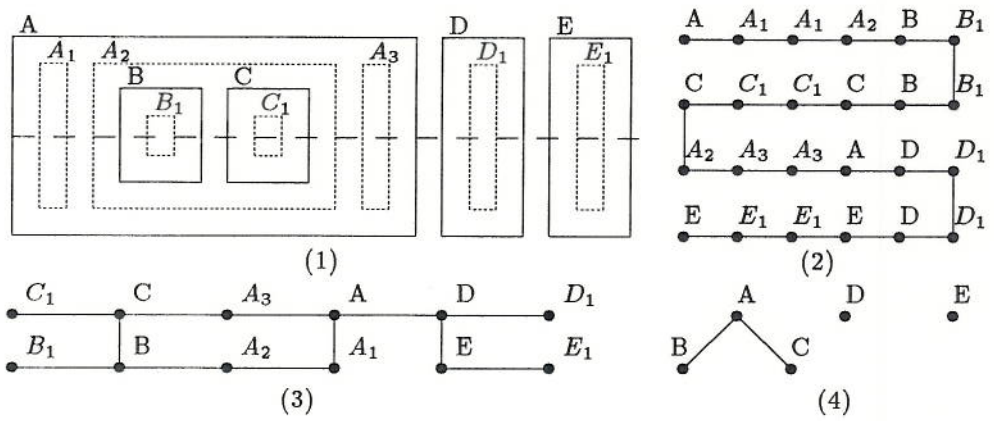

Figure 2: Example of finding irregular-dual forest of G

Fig 4(1). Fig 4(3) is the result of uniting each eigencycle together and uniting multiple edges. Fig 4(4) is the irregular-face-forest obtained in the above algorithm. According to the irregular-face-forest, the out-cycles are denoted with solid lines and the in-cycles are denoted with dot lines in Fig 4(1).

Lemma 6 *Given G and G', algorithm Find-Irregular-Face-Forest finds an irregular-face-forest of G' in $O(\log n)$ time using $O(n)$ processors.*

Proof: It is easy to see that all the nontrivial computations in the algorithm use at most $O(\log n)$ time and $O(n)$ processors. The correctness of the algorithm is based on the simple fact that an irregular face surrounds another face if and only if a path that starts from outside the former has to intersect an in-cycle of the former odd times before it reaches the latter's boundary. An irregular face that can surround other faces must have more than one eigencycles but only has one eigencycle which is always met first by paths from the root of the spanning tree and is selected as the out-cycle of the face. No two faces can surround each other at the same time under our out-cycle selection, otherwise there is a path that starts from the root, passes one face's out-cycle and one of its in-cycles in order, and meets the other face's in-cycles first because they surround each other, this path violates our out-cycle selection rule. So a forest representation of irregular faces is already decided when out-cycles are selected by the above method, what left to do is just simplifying the graph into a forest in corresponding to the surrounding relation. One last detail should be pointed out, an eigencycle may be on boundaries of two irregular faces so we need to maintain two labels on the vertex representing this eigencycle to guide further operations. Clearly, this detail does not damage above process and analysis. ∎

Algorithm 6 *Simplify-to-Biconnected*
Input: *G and G':=Exhaust-k-Unitings(G)*
Output: *Biconnected graph G'' or a small cycle separator of G*

1. Apply Find-Irregular-Face-Forest to G and G';
2. **While** current irregular-face-forest is not empty **do**
   Suppose the depth of current irregular-face-forest is h, select all irregular faces

having depth $\lfloor h/2 \rfloor$ and apply the processing of Lemma 5 to this set of independent irregular faces; (Comments: By Lemma 5, each iteration either prunes all branches underneath selected faces or cuts the forest to a subtree underneath one of the irregular faces when no cycle separator is found. In both cases, the remind irregular-face-forest is about one half shorter.)

3. If the shape of a face is 2, unite it with another face when they share the longer part or an equal part of the former's two part boundary. If the weight of a new face is larger than $\frac{1}{3}n$, use its boundary as a cycle separator.

Theorem 3 *Given G and G', algorithm Simplify-to-Biconnected either simplifies G' into a biconnected planar graph without increasing both the maximum face size and the number of faces or finds a cycle separator no larger than $2k$. The algorithm needs $O(\log^2 n)$ time and $O(n)$ processors.*

Proof: Since Step 2 needs at most $O(\log n)$ iterations, $O(\log^2 n)$ time and $O(n)$ processors are enough by Lemma 5. In Step 2, the size of a new face cannot be larger than the maximum face size of G', because the boundary of the new face is always part of the boundary of a face of G'. Step 2 doesn't creat a face weightier than $\frac{1}{3}n$ unless a cycle separator no larger than $2k$ is found, so a face created in Step 3 is lighter than $\frac{2}{3}n$ and its size is no larger than the size of one of the two faces from which it is formed. ∎

5 Analysis and Conclusion

Now we can use algorithm *Exhaust-k-Unitings* and *Simplify-to-Biconnected* for Step 2 and Step 3 of algorithm *Find-Simple-Cycle-Separator* respectively and claim the following theorem without proof.

Theorem 4 *Parallel algorithm Find-Simple-Cycle-Separator can find a cycle separator on an undirected planar graph in $O(\log^2 n)$ time using $O(n^P)$ processors on the CRCW PRAM, where $1 \leq P \leq M$. The size of the cycle separator is $O(\sqrt{max\{d, n^{\frac{M-P}{M}}\} \cdot n})$, where d is the maximum face size.*

Theorem 5 *A BFS tree on a planar graph can be found in $O(\log^3 n)$ time with $O(n^{1.84})$ processors on the CRCW PRAM.*

Proof: By Theorem 4, we can use Algorithm 1 to find a family of $n^{0.612}$-separators on a planar graph in $O(\log n)$ iterations using $O(n^{1.84})$ processors. Then Pan and Reif's $s^3(n)/(I(n)\log n)$ processor BFS algorithm can take over. ∎

Although the processor-time complexity of our parallel small cycle separator algorithm is optimal to a factor of $O(\log^2 n)$, it is still interesting to find $O(\log n)$ time small cycle separator algorithms. It is still an open issue regarding how to find linear-processor DNC cycle separator algorithms which have tighter cycle separator upper bound when $d \leq n^{0.58}$ and how to find more efficient parallel BFS algorithms for planar graphs.

6 Acknowledgement

The author would like to thank Gregory Shannon for helpful discusses in relation to this work.

References

[1] A. Aggarwal, R. Anderson, and M. Kao. Parallel depth-first search in general directed graphs. In *Proceedings of the 21th Annual ACM Symposium on Theory of Computing*, pages 297–308, 1989.

[2] A. Aho, J. Hopcroft, and J. Ullman. *Data structures and algorithms.* Addison-Wesley, 1983.

[3] R. Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4), August 1988.

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progression. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 1–6, 1987.

[5] H. N. Djidjev. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic and Discrete Methods*, 3(2):229–240, June 1982.

[6] H. Gazit. *Processor Efficient Parallel Graph Algorithms.* PhD thesis, University of Southern California, Los Angeles, CA, August 1988.

[7] H. Gazit and G. Miller. A parallel algorithm for finding a separator in planar graphs. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 238–251, 1987.

[8] H. Gazit and G. L. Miller. An improved parallel algorithm that computes the bfs numbering of a directed graph. *Information Processing Letters*, 28(2):61–65, June 1988.

[9] H. Gazit and G. L. Miller. An $O(\sqrt{n}\log n)$ optimal parallel algorithm for a separator for planar graphs. In preparation, 1989.

[10] M. Kao. All graphs have cycle separators and planar directed depth-first search is in DNC. In *Proceedings of the 3rd Agean Workshop on Computing: VLSI Algorithms and Architectures*, pages 53–63, 1988.

[11] Ming-Yung Kao and Fang Wan. On the size of planar directed path separators. Manuscript, August 1989.

[12] R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Algebraic and Discrete Methods*, 36(2):177–189, April 1979.

[13] G. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, June 1986.

[14] V. Pan and J. H. Reif. Fast and efficient solution of path algorithm problems. *Journal of Computer and System Sciences*, 38(3):494–591, June 1989.

[15] V. Ramachandran and J. Reif. An optimal parallel algorithm for graph planarity. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 282–287, 1989.

[16] G. Shannon. A linear processor algorithm for depth-first search in planar graphs. *Information Processing Letters*, 29(3):119–124, October 1988.

[17] R. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, November 1985.