A Randomized Parallel Algorithm for Planar Graph Isomorphism

Hillel Gazit and John H. Reif*

Department of Computer Science, Duke University, Durham, North Carolina 27708-0129

Received November 13, 1996; revised April 8, 1998

We present a parallel randomized algorithm running on a *CRCW PRAM*, to determine whether two planar graphs are isomorphic, and if so to find the isomorphism. We assume that we have a tree of separators for each planar graph (which can be computed by known algorithms in $O(\log^2 n)$ time with $n^{1+\epsilon}$ processors, for any $\epsilon > 0$). If n is the number of vertices, our algorithm takes $O(\log(n))$ time with $P = O(n^{1.5} \cdot \sqrt{\log(n)})$ processors and with a probability of failure of $1/n$ at most. The algorithm needs $2 \cdot \log(m) - \log(n) + O(\log(n))$ random bits. The number of random bits can be decreased to $O(\log(n))$ by increasing the number of processors to $n^{3/2+\epsilon}$, for any $\epsilon > 0$. Our parallel algorithm has significantly improved processor efficiency, compared to the previous logarithmic time parallel algorithm of Miller and Reif (*Siam J. Comput.* **20** (1991), 1128–1147), which requires n^4 randomized processors or n^5 deterministic processors.

Key Words: Planar graphs; Graph isomorphism; Parallel processing; BFS trees

1. INTRODUCTION

The *Graph Isomorphism* problem is as follows: given two graphs G_1, G_2, determine if there is a renaming of vertices of G_1 that results in the graph G_2. Although it is not known if the general Graph Isomorphism problem can be solved in polynomial time, there are efficient sequential and parallel algorithms for some classes of graphs. For example, isomorphism of trees can be solved sequentially in linear work [AHU] and can be solved by a randomized parallel algorithm of Miller and Reif [MR 91], using tree contraction in $O(\log(n))$ time with an optimal number of processors (or deterministically with a linear factor more processors), and similar bounds

* E-mail: reif@cs.duke.edu.

are known for isomorphism of graphs with constant size separators, for example, outerplanar graphs [BJM, CDR].

Hopcraft and Tarjan gave an $O(n \cdot \log(n))$ algorithm for isomorphism of planar graphs [HT 73b] that uses the fact that every 3-connected planar graph has a unique planar embedding. First they find isomorphism between the 3-connected components. Then they represent each graph as a tree, in which internal vertices correspond to separating vertices, and leaves correspond to 3-connected components. Thus, after they solve the 3-connected planar isomorphism problem, they reduce the problem to tree isomorphism, which has a simple linear time sequential solution [AHU]. This result was improved later by Hopcroft and Wong [HW] to a linear time algorithm.

The goal of this paper is to present a parallel algorithm for planar graph isomorphism. The parallel model that we use is the Concurrent-Read Concurrent-Write (*CRCW*) Parallel Random Access Machines (*PRAM*). It is a synchronized parallel computation model in which all of the processors can read and write into a common memory. In the case of concurrent writes into the same memory location, it is assumed that one of the processors succeeds arbitrarily. It is further assumed that in one time step every processor can read, write, and do basic arithmetic operations with $\log(n)$-bit numbers, and that the processors have access to a random number generator.

There was a previous logarithmic time parallel algorithm of Miller and Reif [MR 91] for planar graph isomorphism, requiring n^4 randomized processors or n^5 deterministic processors. This paper presents a randomized parallel algorithm that uses an overall approach similar to that of Hopcroft and Tarjan [HT 73b], but with some interesting new techniques, including randomization. We assume that we have a tree of separators for each input planar graph, which can be computed using Gazit and Miller's algorithm [GM] in $O(\log^2(n))$ time with $n^{1+\epsilon}$ processors, for any $\epsilon > 0$. We check for isomorphism in the 3-connected components using a randomized method, then map each component to a tree and compare the trees using a tree contraction method. We determine whether the input planar graphs are isomorphic in $O(\log(n))$ time by using $O(n^{1.5} \cdot \sqrt{\log(n)})$ processors. The probability of failure of our algorithm is less than $1/n$.

1.1. *Organization of the Paper*

Section 2 reviews useful graph definitions and known parallel graph algorithms. Section 3 gives a summary of our planar graph isomorphism algorithm. Section 4 presents some new parallel graph algorithms that will also be used to test the isomorphism of 3-connected components. Subsection 4.1 describes how to construct lexicographically ordered BFS trees from a BFS numbering. Subsection 4.2 gives a parallel algorithm for computing these BFS trees from multiple sources.

Section 5 describes in detail our planar graph isomorphism algorithm. Section 6 explains how to reduce the number of random bits required without increasing the probability of failure. Section 7 concludes the paper with some open problems.

2. USEFUL KNOWN PARALLEL GRAPH ALGORITHMS

2.1. *Parallel Tree Algorithms*

We will later introduce, as needed, various known PRAM algorithms to solve certain tree and graph problems. For example, *Euler tree-tour* and *tree contraction* are methods for solving problems on trees. Tarjan and Vishkin's deterministic algorithm for Euler tree-tour [TV] takes $O(\log n)$ time, using an optimal number of processors (see [J]). There are both randomized [MR 89] and deterministic methods [GMT] for tree contraction, both costing $O(\log(n))$ time with an optimal number of processors (see also the texts of JáJá [J] and Reif [R 93]).

2.2. *Parallel Graph Connectivity*

An *undirected graph* $G = (V, E)$ consists of a set of *vertices* V of size n and a set of *edges* E of size m. Each edge is an unordered pair (v, w) of disjoint vertices v and w.

A *directed graph* is defined similarly to a graph, but every edge is an ordered pair. The *underlying graph* of a directed graph is the graph resulting from the directed graph if the direction of the edges is ignored. A *path* joining v_1 and v_k in G is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i < k$.

A graph $G = (V, E)$ is *connected* if there is a path from every vertex in V to every other vertex in V. We will use known parallel algorithms for the connected components [CV, G 91], which require $O(\log(n))$ time and $(n + m)/\log(n)$ processors for randomized algorithms and $(m + n)/\log(n) \cdot \alpha(m, n)$ for the deterministic algorithm, where $\alpha(m, n)$ is the inverse Ackerman function. (By using a randomized connectivity algorithm for planar graphs, the processor bounds can be improved to optimal, but our isomorphism algorithm will not require this improved efficiency.)

A graph is 2-*connected* if and only if there is no vertex v such that we can disconnect the graph by removing v. A graph is 3-*connected* if and only if there is no pair of vertices u, v such that we can disconnect the graph by removing u and v [HT 73a]. There are several efficient parallel algorithms for finding 2-connected components [TV] and 3-connected components

[FRT, MR87]. Their parallel complexity is the same as that of the connectivity algorithms. It will suffice for our isomorphism algorithm that all of these problems take at most $O(\log(n))$ time, using a sublinear number of processors.

For a general planar graph we use Hopcroft and Tarjan's algorithm to build a tree, which we call the 3-*connected components tree*, such that the internal nodes represent connected components, 2-connected components, separating vertices, and separating pairs (between 3-connected components); and the leaves are the 3-connected components. For brevity, we omit here the well-known explicit description, which can be found in Hopcroft and Tarjan's papers [HT 73a, HT 73b]. We can run this algorithm in parallel because we can find the 2-connected components [TV] and the 3-connected components [FRT] in $O(\log(n))$ time, using the same processor complexity as the parallel connectivity algorithms [CV, G 91] that use a sublinear number of processors. Summarizing these known results, we have the following:

LEMMA 2.1. *Given a graph of n vertices and m edges, there is an algorithm for computing the* 3-*connected components tree in* $O(\log(n))$ *time and* $(n + m)/\log(n)$ *processors by a randomized algorithm and* $(m + n)/\log(n) \cdot \alpha(m, n)$ *processors by a deterministic algorithm.*

2.3. *Parallel Planar Embedding*

A graph is planar if it has a *planar embedding*, with each vertex mapped to a distinct point on the plane and each edge mapped to a 1-path on the plane between these points, with no pair of crossing 1-paths.

The planar embedding of a graph is specified by the cyclic order of edges around every vertex [M]. By following this cyclic order from one edge to the next, we find the *faces* of the graph. We can *flip* an embedding by reversing the cyclic order of edges at every vertex; the number of faces remains the same under flipping. A 3-connected planar graph has only two planar embeddings where one is a flip of the other. [E].

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there exists a one-to-one mapping f of V_1 onto V_2 such that $(v, u) \in E_1$ if and only if $(f(v), f(u)) \in E_2$. Our solution of the isomorphism problem in 3-connected graphs depends on efficient planar embedding. This can be computed in $O(\log(n))$ time in our model using an almost optimal number of processors [RR 89, RR 94].

2.4. *Computing Separators in Planar Graphs*

Given a graph G, a subset of vertices B is a *separator* if the remaining vertices can be partitioned into two sets A and C such that there is no edge from A to C, and $\max(|A|,|C|) \leq \frac{2}{3} \cdot n$. The sets A,B,C form a partition of V.

We can take every subgraph (A and C) and separate it recursively. On every recursion, we find a separator where the size of each resulting separated subgraph is at most two-thirds of the size of the previous subgraph. We call the first separator a level 1 separator; we call the separators of A and C level 2 separators, and so on. We can continue to find separators until no subgraph is larger than some constant size. This generates a *separator tree* (or *separator decomposition*) whose vertices are these separators. The root of the tree is the separator B, and its two children are the separator of A and the separator of C. Each of the separators of the separated subgraphs is a child of its separator. We say that every separator S in level $i > 1$ is a child of the separator in level $i - 1$, which separates S from all of the other separators of level i.

Lipton and Tarjan [LT] showed that for every planar graph we can find a separator B of size $\sqrt{8 \cdot n}$. They gave a linear time sequential algorithm for finding the separator B. Gazit and Miller [GM] presented an algorithm for finding an $O(\sqrt{n})$ separator in a planar graph in $O(\log^2(n))$ time, using $n^{1+\epsilon}$ processors for any constant $\epsilon > 0$.

LEMMA 2.2. (Gazit and Miller [GM]). *Given a planar graph with n vertices, there is an algorithm for computing an $O(\sqrt{n})$ separator tree in $O(\log(n))$ time with $n^{1+\epsilon}$ processors for any constant $\epsilon > 0$.*

(Goodrich [G 95] gave an algorithm for finding an $O(n^{\epsilon+1/2})$ separator tree for a planar graph in $O(\log n)$ time using $n/\log n$ processors, assuming a precomputed BFS tree, for any $\epsilon > 0$. However, this does not seem to be of use here, since we only know how to efficiently compute a BFS tree for a planar graph in parallel by use of a separator tree via Lemma 2.3.)

2.5. *BFS of a Planar Graph via a Planar Separator Tree*

The *minimum path problem* is to compute the minimum path distance between specified pairs of vertices. *BFS* is a graph searching method that begins at a given source vertex s of the input graph G and produces a BFS spanning tree such that for each vertex v, v is a path distance d in the BFS tree from the source s iff d is the minimum path distance in the graph G to the vertex v from the source s. A *BFS vertex numbering* provides this minimum path distance to each vertex from the source.

Pan and Reif presented BFS algorithms for planar graphs [PR 89]. Their idea was to solve the minimum path problem independently in each separated subgraph, and then to union the solutions.

The first version of their algorithm required $O(\log^2(n))$ time using $n^{1.5}$ processors. In a later version [PR 91] they used a method called *parallel* *stream contraction* to speed up the calculation to $O(\log(n))$ time, using the same number of processors, by beginning to calculate the minimum distances between separators in level i before the calculations from level $i + 1$ are complete.

Assume that the last level of separators $L \leq \log_{3/2}(n)$. In every iteration i they find minimum paths that use 2^{i+j-L} or fewer edges between the vertices of all the separators in every level j. This yields an $O(\log(n))$ time complexity with the same processor bound.

LEMMA 2.3. (Pan and Reif [PR 91]). *Given a planar graph with n vertices, there is an algorithm for computing a BFS tree in $O(\log(n))$ time with $n^{1.5}$ processors.*

2.6. *Testing Isomorphism of Planar Graphs*

Hopcroft and Tarjan [HT 73b] presented a sequential planar isomorphism algorithm that runs in $O(n \cdot \log(n))$ time. Hopcroft and Wang [HW] improved their result with a linear time algorithm.

Miller and Reif [MR 91] developed the first NC algorithm for planar isomorphism. Their algorithm for a 3-connected planar graph is based on performing *BFS* from every edge e and for each resulting *BFS* tree T_e they form a list L_e of the nodes of the tree by a fixed tree traversal method (say by preorder). Then they determine the lexicographically smallest list L_e (that is, the list that appears first if the lists are sorted) among all edges e of the graph. They reduce the planar isomorphism problem to tree isomorphism for which they also gave both a NC algorithm requiring $O(\log n)$ time and $O(n^2)$ deterministic processors and a more efficient RNC algorithm requiring $O(\log n)$ time and $O(n)$ randomized processors. Their original planar isomorphism algorithm required $O(\log n)$ time and $O(n^5)$ deterministic processors or $O(n^4)$ randomized processors. It is easy to decrease the processor bounds by a linear factor using Pan and Reif's *BFS* algorithm for planar graphs [PR 89].

3. OUTLINE OF OUR PLANAR GRAPH ISOMORPHISM ALGORITHM

We will represent each of the input graphs as a 3-connected components tree (see Hopcroft and Tarjan [HT 73b]), in which the internal vertices correspond to separating vertices, and the leaves correspond to the 3-connected components. The Appendix in Section 8 gives a detailed definition of the 3-connected components tree and discusses its construction, using the algorithm of Miller and Reif [MR 91], in $O(\log(n))$ time using a sublinear number of processors.

Our algorithm for isomorphism in planar graphs uses the following very well-known series of steps:

Input: planar graphs G_1, G_2 with precomputed $O(\sqrt{n})$ separator trees.

1. Find the 3-connected component of the input graphs.
2. Test isomorphism in the 3-connected components.
3. Build the 3-connected components tree for G_1 and G_2.
4. Check the 3-connected components trees for isomorphism.

It will suffice for our isomorphism algorithm that all of these problems take at most $O(\log(n))$ time using a sublinear number of processors.

Planar isomorphism algorithms based on these series of steps, and the proofs of the correctness of the steps, were previously given by many authors, including Hopcroft and Tarjan [HT 73b], Hopcroft and Wong [HW], and Miller and Reif [MR 91]. The Appendix in Section 9 gives details of the reduction from the problem of graph isomorphism of planar graphs to graph isomorphism of planar 3-connected components, which for planar graphs takes $O(\log(n))$ time using a sublinear number of processors.

We will provide a fast parallel test for isomorphism of 3-connected components. In our algorithm we use separators to reduce the running time of the *BFS* algorithm. We also use random sampling to reduce the number of *BFS* trees that we have to compute.

Before we can further detail our Planar Graph Isomorphism algorithm (which we do in Section 5), we must refine the second step, in which we test isomorphism in the 3-connected components. As we shall see, this involves building a separator tree, taking a random sample edge, building *BFS* trees starting from vertices indicated by the sampling, using the *BFS* tree to determine labelings, and comparing labelings. We will select a random sample of $O(\sqrt{n \log n})$ edges in each graph, and, starting from vertices adjacent to these edges, which we call sources, we build BFS trees in lexicographic order. If the graphs are isomorphic, and if our sample includes some pair of corresponding edges, then we get isomorphic ordered BFS trees. We label the vertices in every BFS tree using the Tree-Tour technique [TV] and then represent each graph as a sorted list of its edges. The set of labels associated with a BFS tree will be called a *labeling* of the tree. To test isomorphism of 3-connected components, we need to resolve here three key remaining problems; these will be solved by the three further parallel algorithms given in the next section:

• Unique labeling of a BFS tree. This will be done using an algorithm for building a lexicographic first BFS tree described in Subsection 4.1.

- Compare these labelings. This is easily done by determining whether two lists are the same, using a constant time and an optimal number of processors as described in Subsection 4.2.

- Building multiple BFS trees with specified source vertices (as indicated by the random sampling). This will be done by a multisource BFS described in Subsection 4.3.

4. FURTHER REQUIRED PARALLEL ALGORITHMS

4.1. *Building a Lexicographic First BFS Tree*

Assume that we have the *BFS* numbering of an embedded planar graph computed by proceeding in clockwise order of edges, beginning with some arbitrarily designated source vertex s and with a specified first edge (s,v) to be explored. Consider the following simple processing.

Replace every vertex u of degree 3 or more by a small cycle as follows. Replace every edge (u,w) with (u_w,w), where u_w is a new vertex on the cycle. If w has degree three or more, we replace (u,w) with (u_w,w_u). Thus u_w has three edges: an outside edge and two cycle edges. The *BFS* label of u_w is the same as that of u. We now create a *BFS* tree as follows:

```
1. Every original edge that connects level $i$ and level $i+1$ vertices is in the tree.
2. for every new vertex (except $s_v$):
     if its outside edge comes from a previous level vertex
       then choose that edge for the tree.
     else choose the edge to the counterclockwise neighbor in the cycle for the tree.
```

Proposition 4.1. *The above algorithm creates a tree.*

Proof. Every new vertex except s_v has exactly one edge that enters it, either from a previous level vertex or from the left neighbor. Thus no directed cycles are created, and therefore by definition [E] this graph is a tree. ∎

Direct each edge of this *BFS* tree as suggested by the wording of the algorithm: either "from" the vertex of the edge of the previous level or "to" the counterclockwise neighboring vertex. The order is thus determined by the planar embedding of the original graph, specified by the cyclic order of edges around every vertex [M]. We call the resulting oriented tree the *lexicographic first BFS tree*. Thus we have the following:

PROPOSITION 4.2. *For every vertex, the list of its children in the tree is ordered uniquely.*

Proposition 4.2 immediately implies: if $G = (V, E)$ and $G' = (V', E')$ are isomorphic and have the same embedding and isomorphic edges e and e', respectively, then applying breadth first search from e in G and e' and G' yields isomorphic lexicographic first BFS trees. It is important to note that vertices that were replaced by cycles may be assigned more than one label, and so we take the lexically minimum label in this case, which ensures that the resulting label is unique. Applying the Euler tree-tour algorithm [TV] of Tarjan and Vishkin to these lexicographic first BFS trees yields identical preorder labels, and this can be done (see [J]) in $O(\log n)$ time using an optimal number of processors. Thus we have the following:

LEMMA 4.1. *If $G = (V, E)$ and $G' = (V', E')$ are isomorphic and have the same embedding, then by applying breadth first search from isomorphic edges e in G and e' in G', respectively, all of the isomorphic vertices of G, G' are labeled with the same label.*

4.2. *Checking the Labelings*

Suppose G_1 and G_2 are isomorphic planar graphs with several possible labelings. We must find the labelings of G_1 and G_2 that map corresponding vertices to the same integers.

An obvious method is simply to compare every labeling of the first graph with every labeling of the second graph. Unfortunately, the complexity of this algorithm is equal to the product of the number of labelings of the first graph, the number of labelings of the second graph, and the number of edges.

Our idea is to assign a number to each labeling of each graph. We make up this number by encoding every edge as its adjacent vertices, sorting the edge list, and considering the sorted list as a single number of $2 \cdot m \cdot \lceil \log(n) \rceil$ bits. Then we sort the labelings and compare consecutive labelings.

PROPOSITION 4.3. *A comparison of two labelings can be done in constant time using n processors.*

Proof. We divide every labeling into n blocks of $2 \cdot \lceil \log(n) \rceil$ bits. Processor i compares block i in the first labeling with block i in the second labeling. Since $m < 3 \cdot n$, every processor has to compare at most three blocks; this can be done in constant time. We can find the index of the first block that differs in constant time by using a parallel *min* algorithm. ∎

The labelings that are the same correspond to automorphic mappings between these elements. Therefore, we sort the labelings and divide them into sets of labelings such that all of the labelings in each set are the same. The number of labelings is clearly polynomial in n, so $O(\log($the number of labelings$)) \leq O(\log(n))$. By Lemma 4.3 and Cole's parallel merge sort algorithm [C], we have the following:

LEMMA 4.2. *There is an algorithm for sorting the combined list of labelings of both graphs and to find which of the labelings are the same, which runs in* $O(\log($*the number of labelings*$) \leq O(\log(n))$ *time, using n processors per labeling.*

4.3. *Multisource BFS*

The *fixed source distance problem* is to find the shortest distance in the input graph from a specified source vertex to every other vertex. We compute the *BFS* trees from a set V' of k source vertices concurrently. For this, we need to solve the fixed source distance problem for each of these k source vertices. We use the *BFS* algorithm of Pan and Reif [PR 89] as a basis, which uses a separator tree ST.

We assume a *level* decomposition of the nodes of ST where the root has level 1 and the leaves have level $\leq L = \log_{3/2}(n)$. We divide the graph using ST and then, in L stages, compute the distance to vertices of separators at each level i of ST. The original implementation [PR 89] of this idea costs $O(\log^2(n))$ time, using $O(n^{1.5})$ processors to solve the fixed source distance problem. Pan and Reif [PR 91] reduced the parallel time complexity of the fixed source distance problem by introducing the parallel stream contraction technique, where in every iteration i we find minimum paths that use 2^{i+j-L} or fewer edges between the vertices of all the separators in every level j. Since after $i = \Omega(\log(n))$ iterations, $2^{i+j-L} \geq n$, and we are done. This reduces the time needed to solve the fixed source distance problem to $O(\log(n))$ time with the same $O(n^{1/5})$ processor bound.

We use a similar idea in this algorithm, except that we also complete a *BFS* from every source vertex in V'.

LEMMA 4.3. *Given a planar graph G of n vertices, and given an* $O(\sqrt{n})$ *separator tree ST for G, there is an algorithm for computing the BFS trees from a set* V' *of k arbitrarily selected sources in* $O(\log(n))$ *time, using* $O(k \cdot n + n^{1.5})$ *processors.*

Proof. The multisource BFS algorithm has four stages:

1. For each separator S of ST, compute the distance between every pair of vertices in S.

2. For every source in V', compute the distances from the source to all vertices of the separators of ST that contain it.

3. For every source in V', compute the distance to all of the vertices of separators that do not contain it.

We now observe that the above information can be computed in $O(\log(n))$ time with $n^{1.5}$ processors.

Stage 1 is done in $O(\log(n))$ time with $n^{1.5}$ processors by the algorithm of Pan and Reif [PR 91].

In Stage 2, for each source v in V', we determine the lowest level operator S' in ST that has a vertex adjacent (in the input graph) to vertex v. Then we compute the distance from v to the vertices that are contained in the separators within $A(S')$, where $A(S')$ is the set of the separators in ST that are the ancestors of S'. Using the algorithm of Pan and Reif [PR 91], we compute the distances (from each source v in V' to all of the other vertices) level by level in constant time per level, for a total time bound of $O(\log(n))$. The processor bound is $O(n^{1.5})$ plus $O(n)$ per source, for a total processor bound of $O(k \cdot n + n^{1.5})$.

Stage 3 requires the most detailed analysis. There will be $O(\log(n))$ steps. At any step i we assume that we know (via Stage 2 and the previous steps) the distance from each source vertex v in V' to all of the vertices in all separators in $A(S')$ of level i. We now want to compute the distances between vertex v and all of the vertices of all of the separators of level $i + 1$, so the cost of determining these distances from v is upper bounded by the number of vertices of all of the separators of level $i + 1$ in $A(S')$. Thus the complexity of this step is the product of the size of every separator of level i in $A(S')$ with the size of the two level $i + 1$ separators, which are its children in ST. We will prove that the number of operations we need for every step is upper bounded by the number of operations needed for the previous step. There is a constant c such that $c\sqrt{z}$ upper bounds the size of a separator of a subgraph of z vertices. Assume that

- A level i separator in $A(S')$ splits a set of z vertices, and so it has size at most $c\sqrt{z}$.
- Its two child separators S_1 and S_2 (of level $i + 1$ in ST) split sets of x and $z - x$ vertices, respectively, and so have sizes in at most $c\sqrt{x}$ and $c\sqrt{z - x}$, respectively.
- The two child separators of S_1 (of level $i + 2$ in ST) split sets of $x - y$ and y vertices, respectively, and so have sizes at most $c\sqrt{x - y}$ and $c\sqrt{y}$, respectively.

The number of operations we need in level $i + 1$ is at most $(c\sqrt{x}) \cdot (c\sqrt{y} + c\sqrt{x - y}) = c^2 \cdot \sqrt{x} \cdot (\sqrt{y} + \sqrt{x - y})$. The expression is maximized when $y = x/2$, and equals $c^2 \cdot \sqrt{2} \cdot x$. The number of operations required in level $i + 1$ is upper bounded by $c^2 \cdot \sqrt{2} \cdot x + c^2 \cdot \sqrt{2} \cdot (z - x) = c^2 \cdot \sqrt{2} \cdot z$. Likewise, the number of operations in level i is bounded by $(c\sqrt{z}) \cdot (c\sqrt{x} + c\sqrt{z - x}) = c^2 \cdot \sqrt{z} \cdot (\sqrt{x} + \sqrt{z - x})$, which is upper bounded by $c^2 \cdot \sqrt{2} \cdot z$. It is obvious that the number of operations in the top level is $O(\sqrt{n} \cdot (2 \cdot \sqrt{n/2})) = O(n)$; therefore we need $O(n)$ processors per source vertex. ∎

5. OUR PLANAR ISOMORPHISM ALGORITHMS

5.1. *Isomorphism between 3-connected Planar Graphs*

Recall that we *flip* a planar embedding by reversing the cyclic order of edges at every vertex, and that the planar embedding of a 3-connected planar graph is unique up to a flip. Our following algorithm checks the isomorphism between two 3-connected planar graphs:

Input: 3-connected planar graphs G_1, G_2.

1. Find a planar embedding of each graph G_1, G_2 (which is unique up to a flip).

2. Find a tree of separators for each graph G_1, G_2.

3. Flip (i.e., reverse the order of edges around the vertices) the embedding of one graph.

4. For both graphs G_1, G_2, choose at random a sample set R of $\lceil \sqrt{m \cdot \log(m)} \rceil$ edges. For every edge in the sample R, choose a direction at random: the vertex into which any sample edge leads will be a *source* vertex.

5. Perform BFS from all sources for both graphs G_1, G_2 and the flipped embedding, computing the lexicographic first BFS tree starting from each edge in the set R of sample edges.

6. Find a labeling for every lexicographic first BFS tree.

7. Sort the labelings using Cole's [C] algorithm.

8. **If** two labelings (from different graphs) are the same, **then** the graphs G_1, G_2 are isomorphic, and **otherwise** are not isomorphic.

Recall that the number of edges of a planar graph with no parallel edges is at most $|E| < 3 \cdot n$ (see [E]). The following bound follows immediately by inspection of the above algorithm, and from Lemma 4.3.

PROPOSITION 5.1. *The planar graph isomorphism algorithm needs $O(n^{1.5} \cdot \sqrt{\log(n)})$ processors and $O(\log(n))$ time.*

PROPOSITION 5.2. *If G_1, G_2 are not isomorphic, then the 3-connected planar graph algorithm is always correct.*

Proof. In this case, we claim that their labelings are always distinct for any choice of the sample R. Otherwise, the labeling provide an isomorphism, a contradiction. ▮

However, if G_1, G_2 are isomorphic, then their labelings may still be distinct, so the algorithm gives an incorrect output, with a small probability that we now bound.

LEMMA 5.1. *If G_1, G_2 are isomorphic, then the 3-connected planar graph is correct with probability* $\geq 1 + 1/n$.

Proof. We now show that the probability that no two isomorphic edges will be picked in both graphs is less than $1/n$. Assume that the number of edges in the graph is m. The probability that an edge in the sample of the first graph will be picked in a sample of size 1 in the second graph is at least $\sqrt{m \cdot \log(m)}\ /m$. The probability that no isomorphic edge is picked is at most $1 - \sqrt{\log(m)/m}$. Thus the probability that for a random sample of size $\lceil \sqrt{m \cdot \log(m)}\ \rceil$ no isomorphic edge is picked is at most $(1 - \sqrt{\log(m)/(m)}\,)^{\sqrt{m \cdot \log(m)}} < (1/e)^{\log(m)} < 1/n$. ▮

By Lemmas 5.1 and 4.2 we have the following:

THEOREM 5.2. *Given two 3-connected planar graphs, with precomputed trees of separators, there is an algorithm for testing if they are isomorphic, with probability* $\leq 1/n$ *of failure, in* $O(\log(n))$ *time with* $O(n^{1.5} \cdot \sqrt{\log(n)}\,)$ *processors.*

5.1.1. *Directed Planar Graphs*

If we have a directed planar graph where the underlying graph is 3-connected, then we run the algorithm with a slight change. We compute the *BFS* trees in the underlying graph, but when we sort the edges we look at them as ordered pairs, Lemmas 5.1 and 4.2 hold for this case also.

5.2. *General Planar Graph Isomorphism*

Our algorithm is thus as follows:

Input planar graphs G_1, G_2.

1. Build a 3-connected components tree for each graph G_1, G_2.

2. For each 3-connected components tree, by use of a parallel sort, separate the leaves into classes indexed by (n_i, m_i) according to the number of vertices n_i and edges m_i they have.

3. For every 3-connected component, for each of the two possible embeddings, pick a random sample of $\min(m, \lceil \sqrt{n \cdot \log(n)} \rceil)$ edges, regardless of the size of the 3-connected component.

4. Using a parallel sort algorithm [C], find all of the 3-connected components with the same class (number of vertices, number of edges).

5. Check if 3-connected components with the same class are isomorphic, and give isomorphic leaves the same label.

6. Check for tree isomorphism on the labeled 3-connected components tree.

The isomorphism of the labeled trees can be computed by a randomized parallel algorithm of Miller and Reif [MR 91] using tree contraction, costing $O(\log(n))$ time with an optimal number of processors.

PROPOSITION 5.3. *After decomposing a graph into* 3*-connected components, the number of edges in all of the components together is upper bounded by* $4 \cdot n$.

Proof. The number of edges in a planar graph with no parallel edges is upper bounded by $3 \cdot n$. The number of edges may increase since the edge between each separating pair will be in more than one 3-connected component. Still, the 3-connected components are in a tree, so the number of these new edges is upper bounded by the number of edges in the tree, which is at most $n - 1$. ∎

THEOREM 5.3. *Given separator trees for two planar graphs, there is an algorithm for testing if they are isomorphic, with probability* $1/n$ *or less of failure, which takes* $O(\log(n))$ *time with* $O(n^{1.5}\sqrt{\log(n)})$ *processors.*

Proof. By Lemma 4.3, it immediately follows that computing *BFS* from k sources in every 3-connected component has at most the same time and processor complexity as computing *BFS* from k sources in the original path. The complexity bound follows readily. The proof of correctness of our algorithm for the isomorphism of 3-connected components is given in Theorem 5.2.

It remains for us to upper bound the probability of failure. Let us assume that there are d distinct 3-connected components $C_1, C_2, \ldots, C_d$, which have sizes $s_1, s_2, \ldots, s_d$. If a component C_i has size $s_i \le \sqrt{n \cdot \log(n)}$, then there can be no failure for that component, since we have picked all of its vertices. So to upper bound the probability of failure, we can assume without loss of generality that each component C_i has size $s_i \ge \sqrt{n \cdot \log(n)}$ and that $m \ge \sqrt{n \cdot \log(n)}$. In component C_i of size $s_i \ge \sqrt{n \cdot \log(n)}$, the probability that a given random edge that is picked is not an isomorphic edge is at most $(1 - \sqrt{n \cdot \log(n)}\,/s_i)$. Hence for that component C_i, the probability of failure when using $\sqrt{n \cdot \log(n)}$ random edges is at most $(1 - \sqrt{n \cdot \log(n)}\,/s_i)^{\sqrt{n \cdot \log(n)}} \cdot < e^{-n \cdot \log(n)/s_i}$. Thus the total probability of failure is upper bounded by

$$\sum_{i=1}^{d} \left(1 - \frac{\sqrt{n \cdot \log(n)}}{s_i}\right)^{\sqrt{n \cdot \log(n)}} < de^{-n \cdot \log(n)/s_i}.$$

This probability is maximized if there is just one connected component, with size $s_1 = n$. Thus the probability of failure in components is at most $e^{-n \cdot \log(n)/n} \leq e^{-\log(n)} \leq 1/n$. ∎

6. REDUCING THE NUMBER OF RANDOM BITS

We would like to achieve the same probability of failure using a smaller number of random bits. We use the following lemma by Luby [L]:

LEMMA 6.1. *Let $E_1, \dots, E_n$ be mutually independent events such that $Pr[E_i] = p_i$. Then the probability that at least one of the events will happen is at least $\frac{1}{2} \cdot \min(1, \sum_{i=1}^{n} p_i)$.*

In one graph we pick an arbitrary sample of $\sqrt{m \cdot \log(m)}$ edges, and we compute *BFS* trees for both directions of each edge. For the other graph we pick $\log(n)$ pairs of random numbers in the domain $[1 \dots m]$. Using a method similar to Luby's, every pair of random numbers is used to pick $\sqrt{m/\log(m)}$ mutually independent edges. For every edge we compute *BFS* trees from both directions. The probability that one edge in the sample is isomorphic to an edge in the arbitrary set is at least $\sqrt{\log(m)/m}$, and we have $\sqrt{m/\log(m)}$ such edges. Luby's lemma implies that the probability that at least one of them is isomorphic is at least one-half. By using $\log(n)$ independent pairs, we decrease the probability of failure to $1/2^{\log(n)} = 1/n$ or less. Finally, we need an additional $O(\log(n))$ random bits for the randomized tree isomorphism parallel algorithm of Miller and Reif [MR 91].

6.1. *The Number of Random Bits for General Planar Graphs*

In a general planar graph we need random numbers for every 3-connected component. We can reduce the number of random bits by computing a *BFS* tree for the whole graph and then using subtrees for every 3-connected component. However, this approach risks producing a collection of branches, rather than a tree, for the 3-connected components.

The solution is to add an edge of length zero between every separating pair of vertices. We will prove that the resulting graph is planar by using arguments similar to those in Lemma 7.4 of [E].

PROPOSITION 6.1. *If an edge is added between a separating pair of vertices of a planar graph, then the resulting graph is planar.*

Proof. We can contract any 3-connected component to a single edge between the separating pair vertices, and the graph will still be planar because contraction preserves planarity [LT]. Therefore, we can add such an edge to each 3-connected component, and the graph will still be planar. We can embed each 3-connected component so that the edge we add will be on the external face, and then compose all of these planar embeddings in the cyclic order of the contracted edges. ▮

The length-zero edge assures us that there will not be more than two branches of the BFS tree in every 3-connected component. We call these branches T_1 and T_2. Let T_3 and T_4 be the flipped (as previously defined) embeddings of T_1 and T_2, respectively. Note that since an isomorphism cannot map a strict subset of the vertices of a branch to vertices of the other branch, the numbering of all vertices of one of these branches must precede the numbering of all vertices of the other branch. That is, we have the following cases (note that the first two need not exclude the latter two):

1. All of the vertices of T_1 are before all of the vertices in T_2.
2. All of the vertices of T_2 are before all of the vertices in T_1.
3. All of the vertices of T_3 are before all of the vertices in T_4.
4. All of the vertices of T_4 are before all of the vertices in T_3.

For every 3-connected component, we will have to calculate at most four numbers, corresponding to each of these above cases.

Summarizing the above discussion, we have shown the following:

THEOREM 6.2. *Given trees of separators for two planar graphs, there is an algorithm for testing the graphs for isomorphism, and if they are isomorphic, the algorithm constructs an isomorphism mapping, with probability* $1/n$ *or less of failure, in* $O(\log(n))$ *time with* $O(n^{1.5} \cdot \sqrt{\log(n)})$ *processors, using* $2 \cdot \log(n) \cdot \log(m) + O(\log(n))$ *random bits.*

Note that we can reduce the number of random bits to $O(\log(n))$ by taking larger samples of $n^{1/2+\epsilon}$ edges, for a constant $\epsilon > 0$. Then we can use the result of Chor and Goldreich [CG] for k-wise independent events. (A *k-wise* independent event has a probability that is the product of the individual probabilities of the k events). In that case we will need $n^{3/2+\epsilon}$ processors.

THEOREM 6.3. *Given trees of separators for two planar graphs, there is an algorithm for testing the graphs for isomorphism, and if they are isomorphic, the algorithm constructs an isomorphism mapping, with probability of* $1/n$ *or less of failure, in* $O(\log(n))$ *time with* $O(\log(n))$ *random bits, using* $O(n^{1.5+\epsilon})$ *processors, for any constant* $\epsilon > 0$.

7. OPEN PROBLEMS

1. Develop a processor efficient deterministic isomorphism *NC* algorithm.
2. Develop an isomorphism algorithm that does not need *BFS* as a subroutine.

8. APPENDIX: THE TREE OF 3-CONNECTED COMPONENTS

Let S be a subset of the vertex set V of graph G. Two edges e and e' are *S-equivalent* if there exists a path from e to e' avoiding S. The induced graphs on the equivalence classes of the S-equivalent edges are called the *bridges* of S. A bridge consisting of a single edge is *trivial*. A *separating pair* is a pair of vertices that have 2 or more nontrivial bridges or three or more bridges.

Let the 3-*sets* of graph G be the maximum subsets of vertices of size ≥ 2 that are pairwise 3-connected. Each bridge of a 3-set S contains at most two vertices in S. If G is 2-connected, then the bridge contains exactly two vertices of S. Thus there is a unique 3-connected graph associated with each 3-set. A 3-set S is *proper* if it is of size ≥ 4.

Hopcroft and Tarjan [HT 73b] define the unique tree of 3-connected components of a graph and give a linear time algorithm for finding the tree. Miller and Reif [MR 91] give a fast parallel algorithm for finding the tree of 3-connected components of a graph.

Let G be a 2-connected graph. The nodes of the tree T of 3-connected components are proper components, cycles, and m-bonds, as defined below.

- A *proper component* C is a 3-connected graph with no multiple edges whose vertices consist of a proper 3-set S. If two vertices of C share one or more bridges in G, then they share an edge in C, as follows. Each edge $e = (u,v)$ in C is either (i) an original edge from G (if e is a trivial bridge of S and e is the only bridge of G common to its vertices u,v), or (ii) a new *virtual edge*: if a pair of vertices x,y share a nontrivial bridge or two or more trivial bridges, then a new virtual edge $e = (u, v)$ is placed between u and v in C, and a copy of the virtual edge is also placed in G.

• A *cycle component* C is a simple cycle whose vertex set is a maximum subset of vertices S, where the bridges of S in G form a simple cycle of size 3 or more (possibly with pairs in S containing multiple bridges). A unique trivial bridge e of S becomes an edge of C; otherwise we place a new virtual edge in C and place a copy of the virtual edge in G, as in the definition of proper components.

• An *m-bond component* C is a multiset of m identical edges (u, v), one for each bridge of (u, v) in G, and C has as its vertex set a pair of vertices (u, v) that are both separating and 3-connected (2-bonds are between two proper components or a proper component and a cycle component). The m-bonds lie between the components (proper components and cycles). For each bridge of $\{u, v\}$ in G, we place the original edge $e = (u, v)$ of G in C if the bridge is trivial, and otherwise we place a new virtual edge (u, v) in C and a copy of it in G.

JáJá and Simon [JS] compute the 3-connected components of a graph and its 3-sets in $O(\log n)$ time and $O(n^{O(1)})$ processors, and Miller and Ramachandran [MR 87] (see also Fussel, Ramachandran, and Thurimella [FRT]) reduced this processor bound to sublinear for planar graphs; and their parallel complexity is the same as that of the connectivity algorithms. Thus the vertex sets of the proper components can be computed in $O(\log n)$ time using a sublinear number of processors.

As described in Ramachandran and Reif [RR 89, RR 94], the bridges of the proper components can be computed by reduction to the known fast 2-connectivity algorithm of Ramachandran [R 92] in $O(\log n)$ time, using a sublinear number of processors given the 3-connected components (again, the parallel complexity is the same as that of the connectivity algorithms).

Two components will be *adjacent* if they both contain a copy of the same virtual edge. Since a separating pair (u, v) has at least two bridges, u,v are 3-connected if either they have three or more bridges or one bridge is 2-connected. Thus the m-bond components can be computed in $O(\log n)$ time using a sublinear number of processors, given the 2-connected and 3-connected components, proper components, and bridges.

Miller and Reif [MR 91] describe how to construct in $O(\log n)$ time the unique tree of 3-connected components of a graph from the above three types of components by applying parallel tree contraction. They used the 3-set algorithm of JáJá and Simon [JS], and their processor bounds were bounded by the $O(\log n)$ processor bounds of that algorithm. If their algorithm instead uses the 3-connectivity and 3-set algorithm of Miller and Ramachandran [MR 87], then it can easily be seen that by use of their same parallel tree contraction process, the tree of 3-connected components of a planar graphs is constructed in $O(\log n)$ time, using only a sublinear number of processors (bounded by the processor bounds of the algorithm of [MR 87]).

9. APPENDIX: CANONICAL FORMS OF TREES AND PLANAR GRAPHS

9.1. *The Canonical Forms Problem*

Let the *darts* be associated with an undirected edge, and let (u, v) be the two directed edges from u to v and from v to u. We define the *canonical forms problem* to be as follows: given a 3-connected graph as input, with n vertices and m edges, and possibly with labels on its darts, determine a canonical (linear) ordering of the vertices that is a linear ordering of the vertices that defines an incidence matrix unique up to isomorphism. The canonical ordering of the n vertices can be given as a numbering of the vertices from 1 to n. Given such a canonical vertex numbering, a canonical form for the graph will be compactly given by a number obtained by (i) encoding every edge as its adjacent vertices, (ii) sorting the edge list, and (iii) considering the sorted list as a single number of $2 \cdot m \cdot \lceil \log(n) \rceil$ bits.

9.2. *Canonical Forms of Trees*

A *canonical node labeling* of a tree is defined to be a labeling of each node x by the canonical labeling of the subtree rooted at that node x. Such a canonical node labeling of a tree can be used to induce a canonical ordering of the vertices of the tree, by ordering the children of each node by their canonical labeling. Miller and Reif [MR 91] give a parallel algorithm for canonical labeling and canonical ordering of trees, which we will very briefly sketch here, since similar techniques will be used for canonical forms of planar graphs. To compute this canonical labeling for each node v, and the resulting canonical ordering of their tree, the algorithm of Miller and Reif [MR 91] uses parallel tree contraction. The RAKE and COMPRESS operations of parallel tree contraction have two purposes: (a) they determine canonical labels of nodes and (b) determine a canonical ordering of nodes, in particular, the ordering of the children of nodes. The *center* of a tree is either a node or edge where the distance to all other nodes is minimized, and so induces a rooted tree of minimum height. The parallel tree contraction executes the following operations:

(1) first compute the center of the tree, which is unique up to isomorphism, and

(2) then recursively compute canonical orderings and canonical labelings for the subtrees induced by deleting the center.

Each of the children u of the center are ordered (i) first by the time when they computed their canonical label and then (ii) second by their label given by the recursively computed canonical labeling of the subtree containing u. Thus the nodes are canonical ordered by a postorder traversal of the tree rooted at the center.

9.3. *Reducing Canonical Forms of Planar Graphs to the* 3*-Connected Case*

We now describe the $O(\log n)$ time reduction of Miller and Reif [MR 91] from finding canonical forms for general graphs to that of canonical forms for 3-connected graphs. We shall observe that by use of processor efficient 3-connectivity and 3-set algorithms, the reduction done by their parallel tree contraction process takes only a sublinear number of processors for planar graphs.

We assume that the input graph is planar, and without loss of generality, we assume that the input graph is 2-connected. The methods of Section 8 allow us to compute a unique decomposition of the input graph into a tree of 3-connected components, where each 3-connected component is either a proper 3-connected component, a simple cycle, or an m-bond. We have already given algorithms for canonical forms of proper 3-connected components; these canonical forms have $O(n \log n)$ bits and consist of an integer encoding the canonical listing of the sorted edges.

Here we will determine the canonical forms of 2-connected planar graphs. We can easily extend (with no asymptotic increase in resource bounds) our canonical form algorithm for proper 3-connected components to the case where some of the edges may be labeled. The canonical forms are given in this case by an integer encoding the canonical sorted listing of the edges paired with the edge labels. Canonical forms for labeled cycles and m-bonds, where some of the edges are labeled with integers, can also easily be constructed in $O(\log n)$ time, using a sublinear number of processors.

Miller and Reif [MR 91] extend the parallel algorithm described in Subsection 9.2 for the canonical labeling and canonical ordering of a tree, to compute the canonical labeling and canonical ordering of the 3-connected component tree. The 3-connected component tree is unique up to isomorphism. In this algorithm, two 3-connected components are related by identifying a virtual oriented edge (a dart) in one with a virtual oriented edge in the other.

[1′] In $O(\log n)$ time they root the 3-connected component tree by either a 3-connected component or an identified virtual edge, which induces a rooted tree of minimum height (the center of the tree) in the tree of 3-connected components. (If the center is an edge, they simply introduce a 2-bond as a new component that will be the center of the tree.) They use parallel tree contraction to find this root in $O(\log n)$ time with a sublinear number of processors.

[2′] Then they again use parallel tree contract to recursively compute canonical orderings and labelings from the canonical orderings and labelings of subtrees. To do this they order the vertices into blocks according to which component they belong to. The separating pair is in the same block as the parent component. The blocks are ordered by postorder, using an ordering of the children determined below.

Recall that each node of the 3-connected component tree is a 3-connected component, and the children of a component are coupled to their parent in the 3-connected component tree by an edge rather than a vertex. The 3-connected component tree is unique up to isomorphism, and so the center of the tree is unique. The 3-connected components will be ordered by a postorder traversal of the 3-connected component tree rooted at the center. Each node C of the 3-connected component tree induces a subgraph G_C of the input graph consisting of the 3-connected components of the subtree rooted at that node.

As in the canonical tree labeling algorithm of Subsection 9.2, the RAKE and COMPRESS operations of the parallel tree contraction have the purpose of determining (a) the ordering of the children of nodes (which are 3-connected components) of the tree, where the children are again (i) first ordered by the time when they computed their canonical label and then (ii) ordered by their label, and (b) the ordering of the vertices within each of the 3-connected components.

The RAKE and COMPRESS operations of the tree contraction determine at each node C of the 3-connected component tree a canonical ordering of the vertices of the induced subgraph G_C.

The RAKE operation also provides (c) a canonical labeling of the induced subgraph G_C, given by a canonical listing of the sorted edges.

In the RAKE and COMPRESS operations given immediately below for canonical labeling, one of several orderings for a component are arbitrarily, but uniquely, picked when the labelings for the component are sorted and the largest is picked. (Thus the tree contraction phase does not determine where each vertex is mapped in the final ordering, but this is not required for canonical labeling. The image of the vertex mapping is completely computed by a subsequent tree expansion phase.)

Let $e = (u, v)$ be the virtual edge of component C common to its parent, and let d_1 and d_2 (the reverse of d_1) be the two darts associated with edge e. For the purpose of canonical labeling, the 3-connected component trees in the tree contraction phase, RAKE and COMPRESS, are defined as follows:

• RAKE: We can assume that C is a 3-connected component that is a leaf of the 3-connected component tree. Note that C may contain edges that have been recursively labeled by the algorithm. We compute a canonical labeling $L(C)$ for C, as previously described. If C is a trivial bridge (that is, it does not correspond to a virtual edge, and so is not a proper 3-connected component), then C corresponds to a single edge in its parent component, and RAKE simply labels that corresponding edge with the canonical labeling $L(C)$ for C, and removes C. Otherwise, in its parent component C corresponds to a virtual edge, and we proceed as follows. RAKE first chooses a new distinct label L. using L, RAKE determines two possible labelings for the 3-connected component C and its virtual edge e:

A canonical label L_1 of C when dart d_1 is labeled with L.

A canonical label L_2 of C when dart d_2 is labeled with L.

As described in Miller and Reif [MR 91], the labels L_1, L_2 correspond to the two possible ways in which the virtual edge e may be flipped in isomorphisms of the input graph. RAKE then labels the dart corresponding to d_i in the parent of C with L_i, for $i = 1,2$ and removes C. The ordering of all of the vertices of C except u, v are given as follows. If $L_1 > L_2$, then RAKE uses the ordering from L_1, and similarly if $L_2 > L_1$. As described in Miller and Reif [MR 91], the labels L_1, L_2 correspond to the two possible ways C may be flipped in automorphisms of the input graph. If $L_1 = L_2$, it does not matter which one RAKE picks, since there is an automorphism sending d_1 to d_2. RAKE executes this operation for each leaf in parallel, and thus has the effect of labeling the corresponding darts in the 3-connected component associated with the parent of each leaf and then removing all of the leaves.

• COMPRESS: Let C be a component with only one child in the 3-connected components tree, with corresponding virtual edge e'. Darts d_1 and d_2 of virtual edge e are common to the parent of C, and there are darts d'_1 and d'_2 of virtual edge that are common to the only child of C. COMPRESS first chooses two new distinct labels L and L'. Using these labels, COMPRESS determines four possible labelings:

A canonical label L_1 of C when dart d_1 is labeled with L and dart d'_1 is labeled with L'.

A canonical label L_2 of C when dart d_1 is labeled with L and dart d'_2 is labeled with L'.

A canonical label L_3 of C when dart d_2 is labeled with L and dart d'_1 is labeled with L'.

A canonical label L_4 of C when dart d_2 is labeled with L and dart d'_2 is labeled with L'.

As described in Miller and Reif [MR 91], the labels $L_1, \ldots, L_4$ correspond to the four possible ways in which the virtual edges e, e' (i.e., each virtual edge can be flipped two ways) may be flipped in automorphisms of the input graph. COMPRESS uses the label L_{i^*} with maximum value to determine the order of the vertices in C, excluding the end vertices x, y of e. (As in RAKE, if two labels are equal, then C has a symmetry and either order is the same up to isomorphism.) The effect of COMPRESS is thus to merge the current node and its only child into a new node and to uniquely restrict the order of the vertices of $C - \{x, y\}$. (Since this new node may now have children, which may be processed by further RAKE or COMPRESS operations, we have determined the canonical order of the vertices in components below C. If this new node has no children, then its canonical label will be computed during the next RAKE operation by using this vertex ordering.) Unlike RAKE, COMPRESS only orders vertices within C, and so does not yet determine the canonical label of the subgraph including the components below C.

For our resource bounds in this reduction, we assume an oracle for canonical labeling of proper 3-connected components in logarithmic time (that is, we do not charge to the processor bounds of the oracle). Since these RAKE and COMPRESS operations with k arguments each take time logarithmic in k and require a number of processors sublinear in k, it follows by the results of Miller and Reif [MR 91] that this parallel tree contraction phase takes $O(\log n)$ time with a sublinear number of processors. Thus we have a randomized reduction from canonical labeling of 2-connected planar graphs to canonical labeling of proper 3-connected components, and have shown that the reduction takes logarithmic time with a sublinear number of processors.

Miller and Reif [MR 91] also give a tree contraction phase, also taking $O(\log n)$ time, which determines a unique ordering on components, up to a flip and the final ordering where each vertex is mapped. To compute the image of each vertex in the new ordering, they determine the orientation induced on the virtual edges by the new ordering, specifying whether a given virtual edge is either (i) left alone or (ii) flipped over in the new ordering. To implement COMPRESS (RAKE is similar) in the parallel tree expansion phase, they show how to compute the coset of canonical orderings for a consecutive pair of components from the coset of canonical orderings for each component. Let C be a component with two virtual edges e and e'. The possible symmetries consist of the Klien 4 group K_4: flipping over e and, independently, e'. The actual symmetries will be one of five possible subgroups, and so the canonical orderings will be a coset of one of these subgroups. There is a total of 13 possible such cosets, and they determine these cosets using the precomputed canonical forms of the components. (This is rather straightforward, and for the sake of space we omit the details.) Since these RAKE and COMPRESS operations with k arguments also each take time logarithmic in k and require a number of processors sublinear in k, it again follows by the results of Miller and Reif [MR 91] that this parallel tree expansion phase also takes $O(\log n)$ time with a sublinear number of processors.

ACKNOWLEDGMENTS

We thank Sandeep Sen for directing us to the paper of Chor and Goldreich [CG]. Supported by National Science Foundation grant NSF-IRI-91-00681, Rome Labs Contracts F30602-94-C-0037, ARPA/SISTO contracts N00014-91-J-1985, and N00014-92-C-0182 under subcontract KI-92-01-0182. This paper appeared in the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, Crete, Greece, July 1990, pp. 210–219.

REFERENCES

[AHU] A. Aho, J. E. Hopcroft, and J. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, MA, 1974.

[BJM] T. Beyer, W. Jones, and S. Mitchell, Linear algorithms for isomorphism of maximal outerplanar graphs, *J. Assoc. Comput. Mach.* **26** (1979), 603–610.

[CDR] B. S. Chlebus, K. Diks, and T. Radzik, Testing isomorphism of outerplanar graphs in parallel, *in* "Mathematical Foundations of Computer Science 1988, Lecture Notes in Computer Science," Vol. 324, (Michael P. Chytil, Ladislav Janiga, and Vclav Koubek, Eds.), pp. 220–230, Carlsbad, Czechoslovakia, August 29–September 2, 1988, Springer-Verlag, Berlin, 1988.

[CG] B. Chor and O. Goldreich, On the power of two-point based sampling, *J. Complexity* **5** (1989), 96–106.

[C] R. Cole, Parallel merge sort, *SIAM J. Comput.* **17** (1988), 770–785.

[CV] R. Cole and U. Vishkin, Approximate parallel scheduling. II. Applications to logarithmic-time optimal parallel graph algorithms. *Inform. and Comput.* **92** (1991), 1–47.

[E] S. Even, "Graph Algorithms," Comput. Sci. Press, New York, 1979.

[FRT] D. Fussel, V. Ramachandran, and R. Thurimella, Finding triconnected components by local replacements, *in* "Proc. 16th ICALP, Lecture Notes in Computer Science No.," Vol. 372, pp. 379–393, Springer-Verlag, Berlin, 1989.

[G 91] H. Gazit, An optimal randomized parallel algorithm for finding connected components in a graph, *SIAM J. Comput.* **20** (1991), 1046–1067.

[GM] H. Gazit and G. L. Miller, A parallel algorithm for finding a separator in planar graphs, *in* "28th Annual Symposium on Foundations of Computer Science," pp. 238–248, IEEE Press, New York, 1987.

[GMT] H. Gazit, G. L. Miller, and S. H. Teng, Optimal tree contraction in the EREW model, *in* "Concurrent Computations: Algorithms, Architecture, and Technology" (Tewksbury, Dickinson, and Schwartz, Eds.), pp. 139–156, Plenum Press, New York, 1988.

[GR] H. Gazit and J. H. Reif, A randomized parallel algorithm for planar graph isomorphism, *in* "2nd Annual ACM Symposium on Parallel Algorithms and Architectures," pp. 210–219, Assoc. Comput. Mach., New York, 1990.

[G] M. T. Goodrich, Planar separators and parallel triangulation, *J. Comput. System. Sci.* **51** (1995), 374–389.

[HT 73a] J. E. Hopcroft and R. E. Tarjan, Dividing a graph into triconnected components, *SIAM J. Comput.* **2** (1973), 135–158.

[HT 73b] J. E. Hopcroft and R. E. Tarjan, A $v \log(v)$ algorithm for isomorphism of triconnected planar graphs, *J. Comput. System Sci.* **7** (1973), 323–331.

[HW] J. E. Hopcroft and J. K. Wong, Linear time algorithm for isomorphism of planar graphs, *in* "6th ACM SIGACT," Assoc. Comput. Mach., New York, 1974.

[IM] O. H. Ibarra and S. Moran, Probabilistic algorithms for deciding equivalence of straight-line programs, *J. Assoc. Comput. Mach.* **30** (1983), 217–228.

[J] J. JáJá, "An Introduction to Parallel Algorithms," Addison-Wesley, Reading, MA, 1992.

[JS] J. JáJá and J. Simon, Parallel algorithms for planar graph isomorphism and related problems, *SIAM J. Comput.* **11** (1982), 314–328.

[LT] R. J. Lipton and R. E. Tarjan, A separator theorem for planar graphs, *SIAM J. Appl. Math.* **36** (1979), 177–189.

[L] M. Luby, A simple parallel algorithm for the maximal independent set problem, *in* "STOC 85," pp. 1–10, Assoc. Comput. Mach., New York, 1985.

[M] G. L. Miller, Finding small simple cycle separator for 2-connected planar graphs, *J. Comput. System Sci.* **32** (1986), 265–279.

[MR 87] G. L. Miller and V. Ramachandran, A new graph triconnectivity algorithm and its parallelization, *in* "19th Annual ACM Symposium on Theory of Computing," pp. 335–344, Assoc. Comput. Mach., New York, 1985.

[MR 89] G. L. Miller and J. H. Reif, Parallel tree contraction and its applications, *in* "26th IEEE Symposium on Foundations of Computer Science," pp. 478–489, Portland, OR, 1985. (Published as Parallel Tree Contraction, Part I, Fundamentals, *in* "Advances in Computing Research," Vol. 5, pp. 47–72, 1989.)

[MR 91] G. L. Miller and J. H. Reif, Parallel tree contraction, Part II. Further applications, *SIAM J. Comput.* **20** (1991), 1128–1147.

[MT] G. L. Miller and S. Teng, Tree based parallel algorithm design, *in* "2nd International Conference on Supercomputing," pp. 392–403, Santa Clara, CA, May 1987.

[PR 89] V. Pan and J. H. Reif, Fast and efficient solution of path algebra problems, *J. Comput. System Sci.* **38** (1989), 494–510.

[PR 91] V. Pan and J. H. Reif, The parallel computation of minimum cost paths in graphs by stream contraction, *Inform. Process. Lett.* **40** (1991), 79–83.

[R 92] V. Ramachandran, "Parallel Open Ear Decomposition with Applications to Graph Biconnectivity and Triconnectivity," Technical Report no. CS-TR-92-02, University of Texas, Austin, TX, January 1, 1992.

[RR 89] V. Ramachandran and J. H. Reif, An optimal parallel algorithm for graph planarity, *in* "30th Annual IEEE Symposium on Foundations of Computer Science," R.T.P., NC, pp. 282–287, 1989.

[RR 94] V. Ramachandran and J. H. Reif, Planarity testing in parallel, *J. Comput. System Sci.* **49** (1994), 517–561.

[R 93a] J. H. Reif, Ed., "Synthesis of Parallel Algorithms," Morgan Kaufmann, San Mateo, CA, 1993.

[Sc] J. T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities, *J. Assoc. Comput. Mach.* **27** (1980), 701–717.

[TV] R. E. Tarjan and U. Vishkin, An efficient parallel biconnectivity algorithm, *SIAM J. Comput.* **14** (1985), 862–874.