Extending Planar Graph Algorithms to $K_{3,3}$-free Graphs

Samir Khuller*

88-902
March 1988

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Supported by NSF grant DCR 85-52938 and PYI matching funds from AT&T Bell Labs.

Extending Planar Graph Algorithms to $K_{3,3}$-free Graphs

Samir Khuller *
Computer Science Department, Cornell University

March 17, 1988

Abstract

For several problems, restricting attention to special classes of graphs has yielded better algorithms. In particular, restricting to planar graphs yields efficient parallel algorithms for several graph problems. In this paper we extend these algorithms to $K_{3,3}$-free graphs, showing that the restriction of planarity is not important. The three problems dealt with are : graph coloring, depth first search and maximal independent sets. As a corollary we show that $K_{3,3}$-free graphs are five colorable (this bound is tight).

1 Introduction

For several graph-theoretic problems, solutions are known only for special classes of graphs and not for general graphs. For example, it is an open question if finding a depth first search tree (from now on referred to as DFS) for general graphs is in the complexity class NC (the class of problems which can be solved in polylog time with polynomially many processors). However, an NC algorithm is known for the case of planar graphs [XiYe 88]. A parallel algorithm has been developed for DFS on general graphs [AgAn 87] using randomization, showing the problem to be in the class Random NC.

A natural question is: is planarity fundamentally important in getting fast parallel algorithms for these problems ? Planar graphs have been characterized by Kuratowski's theorem in terms of the forbidden homeomorphs

*supported by NSF grant DCR 85-52938 and PYI matching funds from AT&T Bell Labs.

K_5 and $K_{3,3}$. We give a negative answer to the above question by extending several planar graph NC algorithms to $K_{3,3}$-free graphs (i.e. graphs not containing any subgraph homeomorphic to $K_{3,3}$).

The problem of coloring a graph has attracted a lot of attention and study. The problem is defined as follows: given a graph $G(V, E)$, obtain a k-coloring for the graph (where k is an integer) such that each node is assigned a color and no two adjacent nodes have the same color. No more than k colors should be used. In particular find a function f:$V \rightarrow \{1, 2, .., k\}$ such that f(u) $\neq$ f(v) whenever (u,v) $\in E$.

It has been shown by [Naor 87] how to five color a planar graph in parallel. Using Naor's algorithm we develop a five coloring NC algorithm for $K_{3,3}$-free graphs. Our algorithm runs in time $O(log^5 n)$ and uses $O(n)$ processors. The sequential complexity of our algorithm is $O(n)$. As a corollary we show that $K_{3,3}$-free graphs are five colorable (this bound is tight since K_5 requires exactly five colors). The corresponding tight bound for coloring planar graphs was an open problem for many years until it was proven by [ApHa 77] by a complicated case analysis that every planar graph is four colorable. Notice the relative ease of proving the tight bound for $K_{3,3}$-free graphs. The four coloring of a planar graph can be obtained by a polynomial time algorithm, although it is an open question if this four coloring can be obtained in NC.

We also show how to find a *Maximal Independent Set* in $K_{3,3}$ free graphs. Luby's algorithm for MIS takes $O(log^2 n)$ time but uses $O(n^3)$ processors. For the case of planar graphs it has been shown in [XinH 87] that only $O(n)$ processors are sufficient to achieve the same running time. We develop a MIS algorithm for $K_{3,3}$-free graphs. Our algorithm runs in $O(log^2 n)$ time using only $O(n)$ processors.

Smith [Smit 86] discovered a deterministic NC algorithm to find the DFS tree in planar graphs in $O(log^3 n)$ time and $O(n^4)$ processors. The result was improved by [XiYe 88] to show that for planar graphs, the problem can be solved in $O(log^2 n)$ time by using only $O(n)$ processors. The main idea of the algorithm lies in obtaining a separating cycle in the planar graph using Miller's algorithm [Mill 86]. In this paper it is shown how to decompose $K_{3,3}$-free graphs by finding a separating cycle in them. Using this separating cycle and ideas from [XiYe 88] it is relatively easy to construct a DFS tree in the graph. The algorithm takes $O(log^3 n)$ time and uses $O(n)$ processors.

Our algorithms are based on a special decomposition of $K_{3,3}$-free graphs obtained by Vazirani in [Vaz 87]. This was used in [Vaz 87] to show that the problem of counting the number of perfect matchings in $K_{3,3}$-free graphs is in NC. The model of computation assumed is the CRCW (Concurrent Read Concurrent Write) PRAM model. The model consists of a number of identical processors and a common globally shared memory. In each time unit, a processor can read from a memory cell, perform an arithmetic or logical computation and write into a memory cell. Both concurrent reads and concurrent writes into the same memory cell by different processors are permitted. If a write conflict occurs, an arbitrary processor succeeds.

2 Terminology and Background

Let $G(V, E)$ denote a simple undirected graph . Let c_v denote the color of vertex v. A *legal coloring* of the graph is an assignment of colors to the vertices such that no two adjacent vertices have the same color. A set of vertices I, is said to be independent if no two of them are adjacent. Let $N(\mathrm{v})$ denote the set of vertices which are adjacent to vertex v. Let $\{a, b\}$ be a pair of vertices in a biconnected simple graph G. Suppose the edges of G are divided into equivalence classes $E_1, ..., E_n$ such that two edges which lie on a common path not containing any vertex of $\{a, b\}$ except as an endpoint are in the same class.The classes E_i are called the *separation classes* of G with respect to $\{a, b\}$. If there are at least two separation classes,then $\{a, b\}$ is a *separation pair* of G unless there are exactly two separation classes, and one class consists of a single edge.

If G is biconnected such that no pair $\{a, b\}$ is a separation pair of G, then G is *triconnected.* Let $\{a, b\}$ be a separation pair of G. Let the separation classes of G with respect to $\{a, b\}$ be $E_1, ..., E_n$. Let $E' = \cup_{i=1}^{k} E_i$ and $E'' = \cup_{i=k+1}^{n} E_i$ be such that $\mid E' \mid \geq 2$, $\mid E'' \mid \geq 2$. Let $G_1 = (V(E'), E' \cup \{(a, b)\})$, $G_2 = (V(E''), E'' \cup \{(a, b)\})$.

The Graphs G_1 and G_2 are called the *split* graphs of G with respect to $\{a, b\}$. Replacing a graph G by two split graphs is called *splitting* G. A new edge (a,b) called a *virtual edge* is added in the split components if such an edge (a,b) is not already present in the graph G. Suppose G is split, the split graphs are split, and so on, until no more splits are possible (each remaining graph is triconnected). The graphs constructed in this way are called the *split components* of G.

A planar embedding of a planar graph G is a drawing of G in the plane such that no two edges intersect with each other in the drawing. A *Depth First Spanning tree* is a spanning tree T in G if and only if all edges which are not edges in T are between a pair of nodes, one of which is an ancestor of the other in the tree (assuming the tree has been rooted at a particular vertex).

3 Decomposition of $K_{3,3}$-free graphs

The heart of the parallel algorithms lie in a special decomposition of $K_{3,3}$-free graphs. This decomposition is made possible by a theorem due to [Vaz 87], which is based on the following lemma due to Hall in [Hall 43] (see also [Asan 85]).

Lemma 1 (Hall) *Each triconnected component of a $K_{3,3}$-free graph is either planar or exactly the graph K_5.*

Let G be a $K_{3,3}$-free graph. First obtain the decomposition of G into triconnected components. Then keep merging two planar components if they share a pair of vertices. The components obtained in the end will either be planar pieces or K_5'*s* (merging two planar components keeps them planar). We call these components *pieces*. In [Vaz 87] it is shown that the pieces are invariants of the graph and do not depend on the order in which these components are merged. Pairs of vertices shared by two or more pieces are called *connecting pairs.* Two connecting pairs have at most one vertex in common. Each connecting pair is also a separating pair.

Theorem 1 (Vazirani) *There is a unique decomposition of a $K_{3,3}$-free graph into pieces. Let P be the set of pieces and C the set of connecting pairs of such a decomposition. Construct a new graph H on the vertex set $P \cup C$. If a connecting pair p is contained in a piece h then there is an edge (p,h) in H. The graph H is a tree.*

Our decomposition tree T (also called a tree of pieces) has as its vertex set P, with two vertices having an edge between them if they share a connecting pair. We can assume that G is biconnected since if not, we can find the biconnected components and solve the problem for each biconnected component independently. Then it is an easy task to put the solutions together using similar ideas to the ones presented in this paper.

4 The Coloring Algorithm

The first step in the algorithm is to obtain the decomposition of the graph into its triconnected components. This can be easily done by using the parallel algorithm of [MiRa 87]. The components can be checked for planarity

using the algorithm in [KlRe 86] and the planar and K_5 pieces can thus be identified. Root the tree T of pieces at a K_5 piece and call the root T_r. Each piece in the tree can now be colored independently in parallel.
For the

(i) Planar pieces : use Naor's algorithm

(ii) K_5 pieces : give each vertex a different color.

The colors are assumed to be drawn from the set $\{1, 2, 3, 4, 5\}$ and each vertex is given a color c_v. The next step is to "put" these colors together. Each piece in the tree T, has a unique path to the root T_r (K_5 piece). Each piece has many connecting pairs incident on it, but only one which separates it from its parent piece in the tree. Call this connecting pair the *parent connecting pair.* Each piece (except for the root T_r) has a unique parent connecting pair. We first describe the sequential algorithm to "put" the color's together and then show how it can be parallelized.

Definition 1 *cflip(c_1,u,H) :*
Basically the color-flip operation (cflip) exchanges the colors of vertices colored c_u with the vertices colored c_1 in the piece H.

Definition 2 *cmatch(H_1, H_2) : If H_1 and H_2 are two pieces sharing a parent connecting pair (u,v) (H_1 being the parent of H_2 in the tree T rooted at T_r). In H_1 the color's of u and v are c_u' and c_v' .In H_2 the color's are c_u and c_v.*
Cmatch does the following two operations in sequence : cflip(c_u', u, H_2) ; cflip(c_v', v, H_2) .

The operation ensures that the new color's of u and v in H_2 are c_u' and c_v'. Thus the piece $H_1 \cup H_2$ maintains a proper graph coloring.

Lemma 2 *If a graph G has a valid coloring, performing a cflip(c,u,G) maintains the graph coloring as a valid one.*

Proof: Consider any two vertices v_1 and v_2 such that $(v_1, v_2) \in E$. Now there are various cases to consider:

i) cflip did not change the color of v_1 or v_2 since $c_{v1} \neq c_{v2}$ initially, after the cflip $c_{v1} \neq c_{v2}$.

ii) color of v_1 changed if c_{v1} was initially c(case for c_u is identical) , then $c_{v2} \neq c$. If $c_{v2} = c_u$ then the new colors of c_{v1} and c_{v2} are c_u and c respectively.

iii)both colors changed if both have the same new colors ,then both must have had the same old colors (c or c_u) which is not possible. □

The next step is to "match" the colors of all the pieces. we use the fact that the decomposition of G has a tree structure (on the pieces). Direct all the edges away from the root T_r. If $H_1 \rightarrow p$ and $p \rightarrow H_2$ are the two edges in the directed tree (p is a connecting pair (u,v)), then the piece H_2 is a 'son' of the piece H_1. In order to match the colors of the two pieces we do a cmatch(H_1, H_2) operation. This causes a change in the colors of piece H_2. The piece $H_1 \cup H_2$ now has a valid coloring. This change of colors will also be carried over to all the descendents of H_2 by doing cmatch(H_2,H_k) operations when H_k is a child of H_2. We can thus piece the entire graph back together from its decomposition, ensuring that the coloring remains a valid one.

Thus T_r fixes its color and the rest of the pieces adjust their colors level by level in the tree T. This is the sequential algorithm for obtaining a five coloring for G.

Now we show how to parallelize the color matchings between pieces which was done level by level on the tree T in the sequential algorithm. Note that, on a graph G the cflip operation can be done in constant time on a PRAM by using only $O(n)$ processors with one processor for each vertex. Thus the Cmatch operation can also be done in constant time in parallel.

In the rooted tree T, the new colors of each piece T_i are determined by the colors of the pieces on the unique path from T_i to the root T_r. After rooting the tree at T_r, assign levels to each node T_i.

$$level(T_i) = \begin{cases} 1 & \text{if } T_i = T_r \\ 1 + level(parent(T_i)) & \text{otherwise} \end{cases}$$

The *depth* of the tree T is defined to be Max(level(T_i)) over all nodes T_i of the tree T.

Let $OddT = \{T_i \mid \text{level}(T_i) \text{ is odd }\}$. OddT is essentially the set of pieces having an odd level in the tree T.

Now for each $T_i \in$ OddT and $T_j \in$ Children(T_i) do a Cmatch(T_i,T_j) operation. This takes constant time using $O(n)$ processors. Now merge all the $T_j \in$ Children(T_i) pieces with T_i.

Thus all the nodes at an even level in the tree get merged with their parent nodes, to yield a new tree T^1 from T^0(T).

Obviously depth(T^1) = $\lceil$ depth(T^0)/2 $\rceil$.
The Cmatch ensures that when a node and all of its children get merged, the colors of the children are appropriately flipped to achieve a valid coloring for the new merged piece $T_i \cup$ Children(T_i) ($T_i \in$ OddT). Now perform the above Cmatch and Merging step on the new decomposition tree T^1 to get tree T^2. In $O(log\ n)$ steps the entire tree T, will be reduced to a single node with a valid coloring. Thus the entire graph can be colored in $O(log\ n)$ time once all the planar pieces have been colored.

Summarizing the steps of the Algorithm:
1) Obtain the triconnected components of G and from this obtain the decomposition tree T, after merging adjacent planar pieces.

2) Root the tree at an arbitrary K_5 piece and direct all edges towards their parents in the rooted tree.

3) Color each planar and K_5 piece independently.

4) Put all the colors together doing the Cmatch operations between adjacent pieces and keep shrinking the decomposition tree until we achieve a valid coloring for the entire graph.

ANALYSIS:

Analysis for the Parallel Algorithm:
The algorithm of [MiRa 87] takes $O(log^2 n)$ time and $O(n)$ processors. Planarity checking takes time $O(log^2 n)$ and $O(n)$ processors using the algorithm by [KlRe 86]. Naor's algorithm takes $O(log^5 n)$ time and $O(n^3)$ processors. This algorithm uses Luby's Maximal Independent Set algorithm (for general graphs) as a subroutine which uses $O(n^3)$ processors and $O(log^2 n)$ time. We can instead use a Maximal Independent Set algorithm for planar graphs which runs in $O(log^2 n)$ time and uses only $O(n)$ processors [XinH 87]. Finding the decomposition tree of pieces and obtaining the rooted tree of pieces can be done in $O(log^2 n)$ time using only $O(n)$ processors by finding a DFS tree rooted at the vertex chosen as the root [XiYe 88]. Matching up all the colors can easily be done in $O(log\ n)$ time with $O(n)$ processors. Thus the entire parallel algorithm runs in $O(log^5 n)$ time with $O(n)$ processors.

Analysis for the Sequential Algorithm:
If we use the algorithm of [ChNS 81] to color all the planar pieces of the

decomposition tree T we can achieve a running time of $O(n)$. Since the triconnected components can be obtained by [HoTa 73] in $O(n)$ time and planarity checking can also be done in $O(n)$ time using [HoTa 74]. Obtaining the tree of pieces and rooting it is a relatively easy task and takes no more that $O(n)$ time. Performing the color matchings level by level in the tree also takes $O(n)$ time. Thus a five coloring of a $K_{3,3}$-free graph can be obtained in $O(n)$ time.

Theorem 2 *Every $K_{3,3}$-free graph is five colorable and there is an NC^5 algorithm to obtain the five coloring.*

Proof: Trivial from the correctness of the algorithm and the above analysis. It should be noted that the bound of five colors is tight, since K_5 is a $K_{3,3}$-free graph which needs exactly five colors for a valid coloring. □

5 Depth First Search

Let C be a simple cycle in a planar graph G. The vertices of G are partitioned into three parts: the vertices on C, the vertices in the interior of C and the vertices in the exterior of C. If no connected component of the interior of C or the exterior of C contains more than $\frac{2}{3}n$ vertices, C is called a *separating cycle* of G. The DFS algorithm for planar graphs in [XiYe 88] relies on the fact that all planar graphs have a separating cycle which can be obtained very fast in parallel as shown in [Mill 86]. In fact Miller showed that a separating cycle of size $O(\sqrt{n})$ always exists and can be obtained in NC. To solve the DFS problem the length of the cycle is not important, allowing us to construct a cycle of arbitrary length more efficiently. The separating cycle allows us to decompose the graph and find a DFS tree on the smaller pieces recursively as shown in [XiYe 88]. To find the DFS tree of G rooted at vertex r, we obtain a separating cycle C for the graph. Now find a path P_{rx} from r to any node x on C such that no other vertex of C is on P_{rx}. Let y be one of the neighbours of x on C. Consider a path P_{ry} from r to y going through x and all around the cycle C. Suppose $G - P_{ry}$ is the union of connected components $\{G_i\}$. Let e be an edge of G. We call e a *touching edge* of G_i if one end vertex of e is in P_{ry} and the other end is in G_i. A touching edge of G_i is called an *essential touching edge* if there does not exist another touching edge e' of G_i which touches P_{ry} at a point further (in P_{ry}) from r than the point at which e touches P_{ry}. Let e_i be an essential touching edge (if there are many such, pick one) of G_i and let x_i

be the end vertex of e_i in G_i. Suppose T_i is the DFS tree of G_i rooted at x_i. It is shown in [Smit 86] that the union of P_{ry}, $\{e_i\}$ and $\{T_i\}$ forms a DFS tree for the graph G.

Here we show that all $K_{3,3}$-free graphs have a separating cycle which can be found in parallel, allowing us to find a DFS tree on $K_{3,3}$-free graphs in parallel.

To show how to construct the separating cycle in the $K_{3,3}$-free graph in parallel we first state the following lemma.

Lemma 3 *If T is a tree and each node $v_i \in T$ has a nonnegative weight w_i. Define $W = \sum_{v_i \in T} w_i$. Then there exists a vertex v in T, such that every connected component of T-v has total weight $\leq \frac{2}{3}W$. v is called a* separating vertex *for the tree.*

Proof: See [Mehl 84]. □

Theorem 3 *Every $K_{3,3}$-free graph has a separating cycle which can be obtained in NC.*

Proof: Consider the decomposition tree T of the graph G. To each node T_i assign a weight w_i = number of vertices in piece T_i. Now using the above lemma find the separating vertex for the decomposition tree. Call this piece (vertex in tree T) T_p. We need to find a cycle within the piece T_p whose removal breaks up the graph into connected components each of which is no larger than $\frac{2}{3}$n. Note that, we may not be able to disconnect all the triconnected components hanging off from the piece T_p. The only components which get disconnected are the ones which have their connecting pairs on the cycle, the others get into one of two groups: Interior group and Exterior group. The triconnected components in each group may remain connected to one another. We need to consider two cases:

(i) T_p is a K_5 piece.

Choose C to be the cycle consisting of all the five vertices in the K_5 piece. Some of the edges (a,b) on the cycle may be virtual edges, with a piece hanging off from the connecting pair (a,b). In which case we can find a path from a to b in the piece hanging off and splice it into the cycle C, replacing the virtual edge (a,b).

(ii) T_p is a planar piece.

Consider every connecting pair (u,v) in T_p. Removal of (u,v) from the graph yields at least two connected components. Let n_{uv} be the sum of the number of vertices in all the connected components formed except for the component containing the piece T_p. Introduce n_{uv} new nodes on the virtual edge (u,v) in T_p. Now in the resulting planar graph find a separating cycle C (using Miller's parallel algorithm). Each group of nodes added on an edge will either get removed (being on the cycle), belong to the Interior group or the Exterior group. The total number of vertices in the graph with the new nodes added is still $O(n)$. The separating cycle in this graph ensures that the sizes of the connected components in the Interior and Exterior groups is bounded by $\frac{2}{3}n$. □

ANALYSIS:
The decomposition tree can be obtained in time $O(log^2 n)$ with $O(n)$ processors (shown in the analysis for five coloring). Using the planar graph DFS algorithm we can root the tree in the same time using only a linear number of processors. To compute the separating vertex of the tree we use the RAKE and COMPRESS operations in [MiRe 85]. These operations allow us to propogate information very easily from the leaf nodes to the root of the tree. Using these two operations we can compute the sizes of the connected components formed when a vertex is removed from the tree (this can be done in $O(log\ n)$ time using only $O(n)$ processors for all the vertices in the tree). Once the sizes of the components formed are known it is easy to identify (in parallel) a separating vertex. If the separating vertex is a K_5 piece, then we can trivially identify a five cycle, replace the virtual edges by splicing in paths (which can be found in $O(log\ n)$ time with $O(n)$ processors as in [XiYe 88]). If the piece is planar we find a separating cycle using Miller's algorithm [Mill 86], which takes time $O(log\ n)$ and $O(n)$ processors if we do not care about the length of the separating cycle. Thus the separating cycle for the $K_{3,3}$-free graph can be obtained in $O(log^2 n)$ time using only $O(n)$ processors. Obtaining the DFS tree for the graph by using this separating cycle adds a factor of $O(log\ n)$ to the running time since we need to solve the problem recursively on smaller subgraphs, yielding a running time of $O(log^3 n)$ using $O(n)$ processors.

Theorem 4 *The Depth First Spanning Tree problem for $K_{3,3}$-free graphs is in NC^3 and uses only $O(n)$ processors.*

Proof: Trivial from the correctness of the above algorithm and the analysis. □

6 Maximal Independent Sets

The construction of a MIS on these graphs relies on first obtaining the decomposition tree of the graph (as obtained for coloring). The essential idea is to find Maximal Independent Sets for each piece and then combine these sets together to produce a MIS for the entire graph. The difficulty arises in handling cases when the vertices of a connecting pair belonging to many pieces, are in the MIS's for some of the pieces and not in the MIS's for the others. As before, root the decomposition tree at some vertex T_r. Each piece has a parent connecting pair (u,v). There are four possibilities:
1) u and v are in the MIS (only if they do not have a real edge)
2) u is in the MIS and v is not
3) v is in the MIS and u is not
4) both u and v are not in the MIS
Essentially we need to construct four MIS's for each piece, one for each of the above cases (this is done by forcing the vertices either into the set or out of it, then constructing a MIS on the rest of the piece). Call the MIS's corresponding to each of the above cases A,B,C,D respectively. A K_5 piece is a constant sized piece and we can easily construct a MIS for it considering only the real edges. For constructing a MIS on the planar piece use the algorithm by [XinH 87].

Now consider the following tree of macro nodes. Each macro node denotes a piece in the decomposition of the graph. Each macronode T_i contains four nodes in it, namely T_i^A,T_i^B, T_i^C,T_i^D. Each node within a macronode denotes a MIS for the corresponding piece. Let $X, Y \in \{A, B, C, D\}$. In the tree, we put an edge (directed) from T_i^X to T_j^Y if T_i is the parent of T_j in the macro node tree and the MIS's corresponding to X and Y "agree" in their containment of u and v (where (u,v) is the parent connecting pair of T_j. More formally, ($u \in T_i^X \Leftrightarrow u \in T_j^Y$ and $v \in T_i^X \Leftrightarrow v \in T_j^Y$) iff there is an edge (directed) from T_i^X to T_j^Y. We can arbitrarily pick one of the nodes in T_r as the chosen MIS for piece T_r and call it T_r^A.

To construct a MIS for the entire graph, we chose all the MIS's T_i^U if there is a directed path from T_r^A to T_i^U. By a pointer jumping technique it is possible to determine for every node if there is a path to it from the root node in $O(log^2 n)$ time with $O(n)$ processors. It is easy to show by induction on the levels of the macro node tree, that in every macro node only one node gets selected as the chosen MIS for that piece. The basic idea is to keep compressing the tree by merging its odd and even level macro nodes. We add an edge from $T_i^X to T_k^Z$ if there is an edge from $T_i^X to T_j^Y$ and an edge from $T_j^Y to T_k^Z$ (T_i and T_k are at even levels in the macro node tree, level(T_k)= level(T_i) + 2). The operation keeps halving the height of the tree at each step and at the end of $O(log\ n)$ steps, all nodes at level = $\frac{depth}{2^k}$ for k=0,1,2,..,*log depth* which have a path from the root node have been identified. Now we can recursively solve the problem for the parts of the tree formed by deleting the nodes (call these selected nodes) at levels which have been identified as having paths from the root to them. We need to find paths from the selected nodes to the other nodes in the trees rooted at the selected nodes, which is done recursively using the same algorithm.

ANALYSIS:
Constructing the decomposition tree, rooting it and finding out all the MIS's for the pieces takes time $O(log^2 n)$ using the algorithm by [XinH 87] and $O(n)$ processors. The pointer jumping technique as shown above takes time $O(log^2 n)$ with $O(n)$ processors (the extra $O(logn)$ factor comes due the recursive step). Hence a MIS for the entire graph takes $O(log^2 n)$ time using only $O(n)$ processors.

Theorem 5 *The Maximal Independent Set problem for $K_{3,3}$-free graphs is in NC^2 and uses $O(n)$ processors.*

Proof: Trivial from the above algorithm and its analysis. □

7 Acknowledgements

I would like to thank Vijay Vazirani for ideas, discussions and encouragement without whom this work would not have been possible. Thanks also to Esther Arkin, Rafael Hassin and Joseph Mitchell for commenting on earlier drafts of the paper.

References

[AgAn 87] A.Aggarwal and R.J.Anderson, 'A Random NC algorithm for depth first search', *Proceedings of STOC conference*, (1987),pp 325-334.

[ApHa 77] K.Appel and W.Haken, 'Every planar map is four-colorable' , *Illinois Journal of Maths* 21(1977) 429-567.

[Asan 85] T.Asano, 'An Approach to the subgraph homeomorphism Problem', *Theoretical Computer Science* 38(1985), pp 249-267.

[ChNS 81] N.Chiba, T.Nishizeki and N.Saito, 'A linear algorithm for five coloring a planar graph ', *Graph Theory and Algorithms,LNCS* Vol 108 (Springer,Berlin 1981), pp 9-19.

[GaJo 78] M.R.Garey and D.S.Johnson, 'Computers and Intractability : A Guide to the Theory of NP-Completeness', *Freeman,San Francisco.*

[Hall 43] D.W.Hall, ' A note on primitive skew curves ', *Bull. Amer. Math. Soc.*, 49 (1943), pp 935-937.

[HoTa 73] J.E.Hopcroft and R.E.Tarjan, 'Dividing a graph into triconnected components',*SIAM Journal of Computing* 2:3,Sept (1973) , pp135-158.

[HoTa 74] J.E.Hopcroft and R.E.Tarjan, 'Efficient planarity testing', *Journal of the Assoc for Comp Mach* 21,(1974), pp 549-568.

[JaSi 82] J.JaJa and J.Simon, 'Parallel algorithms in graph theory : Planarity Testing' ,*SIAM Journal of Computing* ,11 (1982), pp 314-328.

[KlRe 86] P.N.Klein and J.H.Reif, 'An Efficient parallel algorithm for planarity ', *Proceedings of FOCS conference* (1986), pp 465-477 .

[Mehl 84] K.Mehlhorn, 'Graph algorithms and NP-Completeness', *Springer Verlag*, (1984).

[Mill 86] G.L.Miller, 'Finding small simple separators for 2-connected planar graphs', *Journal of the Assoc for Comp Mach* ,32 (1986), pp 265-279.

[MiRa 87] G.L.Miller and V.Ramachandran, 'A New graph triconnectivity algorithm and its Parallelization ',*Proceedings of STOC conference* (1987), pp 335-344.

[MiRe 85] G.L.Miller and J.Reif, 'Parallel tree contraction and its applications', *Proceedings of FOCS conference* ,(1985), pp 478- 489.

[Naor 87] J.Naor, 'A Fast parallel coloring of planar graphs with five colors' , *Information Proceedings Letters* 25 (1987), pp 51-53 .

[Smit 86] J.Smith, 'Parallel algorithms for depth first searches I:Planar Graphs', *SIAM Journal of Computing*,15,1986, pp 814-830.

[Tutt 66] W.T.Tutte, 'Connectivity in graphs' , *University of Toronto Press* 1966 .

[Vaz 87] V.V.Vazirani, 'NC Algorithms for computing the number of perfect matchings in $K_{3,3}$ free graphs and related problems ', *to appear* .

[XinH 87] X.He, 'A nearly optimal parallel algorithm for constructing maximal independent sets in planar graphs', *to appear.*

[XiYe 88] X.He and Y.Yesha, 'A nearly optimal parallel algorithm for constructing depth first spanning trees in planar graphs', *SIAM Journal of Computing*, vol 17, no 3, June 1988.