Research report 83

FAST PARALLEL ALGORITHMS FOR OPTIMALLY EDGE AND VERTEX COLOURING HALIN GRAPHS

Alan Gibbons & Wojciech Rytter*

(RR83)

Abstract

We show that every Halin graph can be optimally edge-coloured or optimally vertex-coloured in polylog time using a polynomial number of processors on a parallel random access machine without write conflicts (P-RAM). Our algorithm is designed using the tree-structure of Halin graphs which allows an application of the Divide and Conquer technique in a parallel setting.

Department of Computer Science
University of Warwick
Coventry CV4 7AL, England

* and Institute of Informatics, Warsaw University, Poland

Oct 1986

FAST PARALLEL ALGORITHMS FOR OPTIMALLY EDGE AND VERTEX COLOURING HALIN GRAPHS

Alan Gibbons
Department of Computer Science, University of Warwick
Coventry CV4 7AL, England.
and
Wojciech Rytter
Department of Computer Science, University of Warwick
and Institute of Informatics, Warsaw University, Poland

Abstract
We show that every Halin graph can be optimally edge-coloured or optimally vertex-coloured in polylog time using a polynomial number of processors on a parallel random access machine without write conflicts (P-RAM). Our algorithm is designed using the tree-structure of Halin graphs which allows an application of the Divide and Conquer technique in a parallel setting.

Key Words: Halin graphs, Edge-colouring, Vertex-colouring, Parallel Algorithm.

1. Introduction.

A general approach to designing fast parallel algorithms for problems dealing with well structured objects is to reflect their structure in the divide and conquer technique. The resulting algorithms are recursive. The first step is a decomposion of the input object into similarly structured objects whose sizes decrease by a constant proportion p less than one. (In this paper p = 1/2). The second step is a recursive and parallel application of the same algorithm to each of the objects independently. This independence may result in some inconsistencies between individual objects (in our case different colours for the same edge) which have to be resolved. The third step of the algorithm removes these inconsistencies for each object making suitable adjustments (in our case recolouring). This approach was also used for fast parallel edge-colouring [6] and vertex- colouring [1] of outerplanar graphs.
In graph algorithms the above approach is best suited to trees. Halin graphs and outerplanar graphs fall between trees and more general graphs in that trees are reflected in their structure. A Halin graph is planar and consists of a tree T with no vertices of degree two and a circuit C (called the skirt) which consists precisely of a sequence of all the leaf vertices of T. A graph is outerplanar if it is planar and every vertex lies on the same (which we can take to be the external) face. We can construct a graph in which each vertex corresponds to an internal face of a particular oterplanar graph and which has an edge between such face vertices iff the corresponding faces are adjacent. The graph so constructed is a tree and is a partial dual of the outerplanar graph. (In fact it is the graph obtained from the dual by deleting the vertex (w, say) corresponding to the external face of

the outerplanar graph. If we add to this tree a circuit of length degree(w), bounding the tree in the plane, and if we add, in planar fashion, an edge from each circuit vertex to a corresponding leaf of the tree then we obtain a Halin graph.) It is this tree of faces which is algorithmically taken advantage of in [6].

It is well known that the minimum number of colours with which it is possible to edge-colour a graph so that no two adjacent edges are similarly coloured is either Δ or $(\Delta+1)$, depending upon the graph. Here and throughout the paper Δ is the maximum vertex degree of the graph. The problem of determining whether or not a graph is Δ-colourable is NP-hard [7]. However it is known, for example, that bipartite, outerplanar and Halin graphs (except for odd cycles) are Δ-colourable, moreover there are polynomial-time sequential algorithms to obtain optimal (that is, using a minimum number of colours) colourings [2,3,5,10,12]. NC is the class of problems solvable in polylog (i.e. $O(\log^k n)$, for some k) parallel time with a polynomial number of processors. It is known that the problems of optimal edge-colouring of bipartite graphs and of outerplanar graphs are in NC [6,9]. We show that optimal edge-colouring of Halin graphs is also in NC.

The problem of vertex-colouring an arbitrary graph using the minimum number of colours so that no two adjacent vertices are similarly coloured is well known to be NP-hard. However, polynomial-time sequential algorithms are known for the classes of graph mentioned earlier [12]. It was shown in [1] that the problem of optimal vertex-colouring of outerplanar graphs is in NC. We observe (in Remark 2) that the problem of optimally vertex-colouring Halin graphs is also in NC.

We take the definition of a Halin graph (sometimes called a skirted tree) from [12]. In any planar embedding of the Halin graph, C forms the boundary of some face. Without loss of generality, we can always take this to be the external face. In this paper we presume a natural representation for Halin graphs as follows. For each vertex v we have an ajacency list in which the neighbours of v are ordered in their (say, clockwise) rotational occurence about v in the planar embedding of the graph. Moreover we also have an explicit representation of the skirt C (thus T is also available in a single parallel step by removing skirt edges from the Halin graph).

Remark 1.

Given a more commonly used representation we can obtain a planar embedding, as represented by the set of faces, in polylog time with a polynomial number of processors [8]. The problem of then proceeding to our representation is in NC. We omit the details but observe that C will be a face which has exactly one edge in common with every other face and that if (u,v), (v,w) are consecutive edges of some face (in anticlockwise order) then u and w will consecutive vertices in our adjacency list representation for v.

2. Edge Colouring.

Before describing our edge colouring algorithm we record the following fact . Here and throughout the paper ND(v) is the number of descendants of v (including v) in a rooted version tree.

Fact 1.

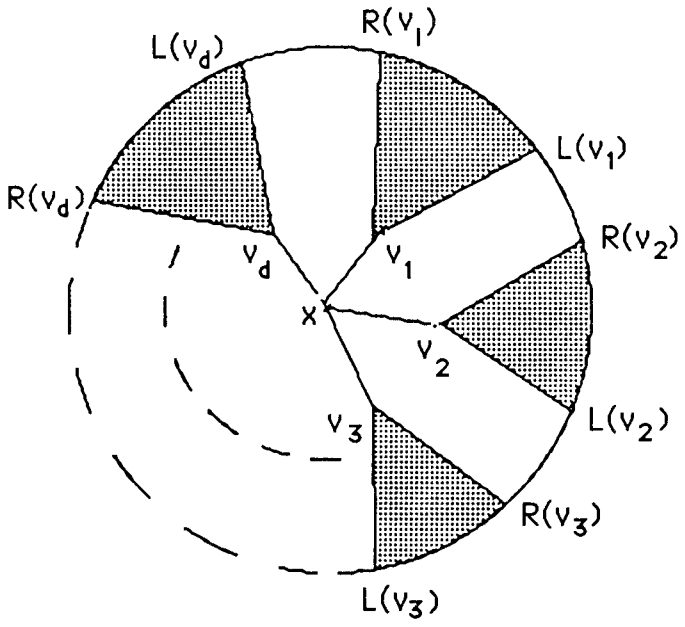
For every tree T with n vertices there exists at least one vertex x such that for every component K of (T-x), $|K| \leq n/2$.

Proof. Let T be arbitrarily rooted at some vertex u. It is easy to see that at least one vertex x will exist for which $ND(x) > n/2$ and for which $ND(w) \leq n/2$ for every son w of v. Moreover it is easy to see that x satisfies the thesis.

Tarjan and Vishkin in [14] have shown how to compute ND(v) in log n parallel time with n processors (also father(v) which we shall presume to know later). Hence we can find x efficiently as follows. Given ND(v) for all v, we proceed (in parallel) to compute MND(v), defined to be max{ ND(w):w is a son of v}. This can be done in logarithmic time using a standard doubling argument. If $ND(v) > n/2$ and if $MND(v) \leq n/2$ then v satisfies fact 1. It is possible that v is not unique in this respect. It is a simple matter to identify a single vertex satisfying fact 1 in logarithmic time by, for example, choosing the first such vertex in a locally constructed list of vertices (we assume they have a suitable alphabetic labelling) using, again, a standard doubling argument. Thus a vertex satisfying fact 1 can be found in O(log n) time on O(n) processors.

Given x as defined in fact 1, we imagine (without loss of generality) that T is rooted at x. We shall require to find, for each son(x), its leftmost descendant leaf L(son(x)) and its rightmost descendant leaf R(son(x)). Such vertices are indicated in figure 1. In that figure the circuit C of the Halin graph bounds the exterior face. The degree of x is d and its sons are $v_1, v_2 \ldots v_d$. Again, it is a simple matter to find L(v) and R(v) for all v in log n parallel time using standard doubling arguments based on leftmost and rightmost sons (in our representation these are known, being on either side of father(v) in the adjacency list of v).

FIGURE 1.

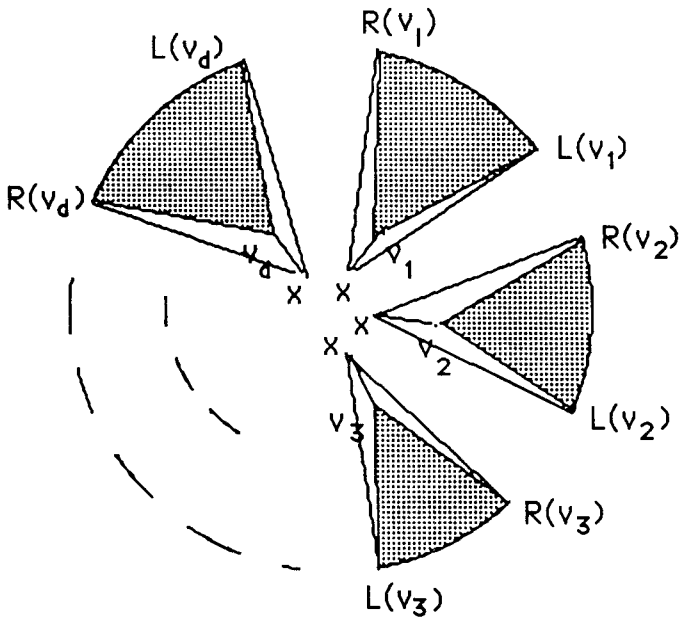


Algorithm 1.

STEP 1 {Decomposition}.

In constant parallel time on O(m)=O(n) processors, we decompose the Halin graph as follows. Remember that, by definition of a Halin graph, d(v)>2 for all v. It follows that d(x)>2 and that L(v)≠R(v) for all v unless v is a leaf, in which case v=L(v)=R(v). If (u,w) is an edge such that u=L(v_i) and v=R(v_{i+1}) where we assume modulo d, then (u,w) is deleted. [In fact, such edges have to be stacked for reconstruction purposes later]. In addition, for each L(v_i) and R(v_i) we add edges (x,L(v_i)) and (x,R(v_i)). This deletion and addition of edges results in a graph in which x is an articulation point, we complete the decomposition by separating the graph at x, passing a copy of x to each component. The decomposition for figure 1 is shown schematically in figure 2. Notice that each component of the decomposition will be a Halin graph unless v_i is a leaf of T. In this case the component associated with v_i will be the multigraph with n=2 and m=3. Notice that, in constant time,we preserve our form of the representation of Halin graphs during the decomposition.

FIGURE 2.



STEP 2. {Recursive application of algorithm 1}

for each component created in step 1 do in parallel

if the size of the component > 2 then apply Algorithm 1 to this component

else {n=2} colour the edges of this {multigraph} component with 1, 2 and 3.

STEP 3. {Reconstruction and Colouring}

Given an optimum colouring of each of the components of the decomposition described we can find an optimum colouring of the reconstructed component as follows. For the ith component of the decomposition (which contains v_i, the ith son of v), let $C1_i$, $C2_i$ and $C3_i$ (in clockwise rotational order about x) be the edge colours present at x. For k = 1,2 ...d we intend that colour C_k be assigned to e_k=(x,v_k) in H. In order to achieve this we permutate the colours in the ith component according to any permutation in which $C1_i$ becomes C_{i-2}, $C2_i$ becomes C_i and $C3_i$

becomes C_{i-1}. Such a permutation can be found within our time constraints with little difficulty. These permutations additionally ensure that when H is reconstructed there are no colour clashes on replaced edges because $C3_i$ and $C1_{i+1}$ are replaced by the same colour which is assigned to the skirt edge $(L(v_i), R(v_{i+1}))$ which is replaced after the edges $(x,L(v_i))$ and $(x,R(v_{i+1}))$ have been removed. Such a colouring scheme is shown schematically in figure 3.

End of Algorithm 1

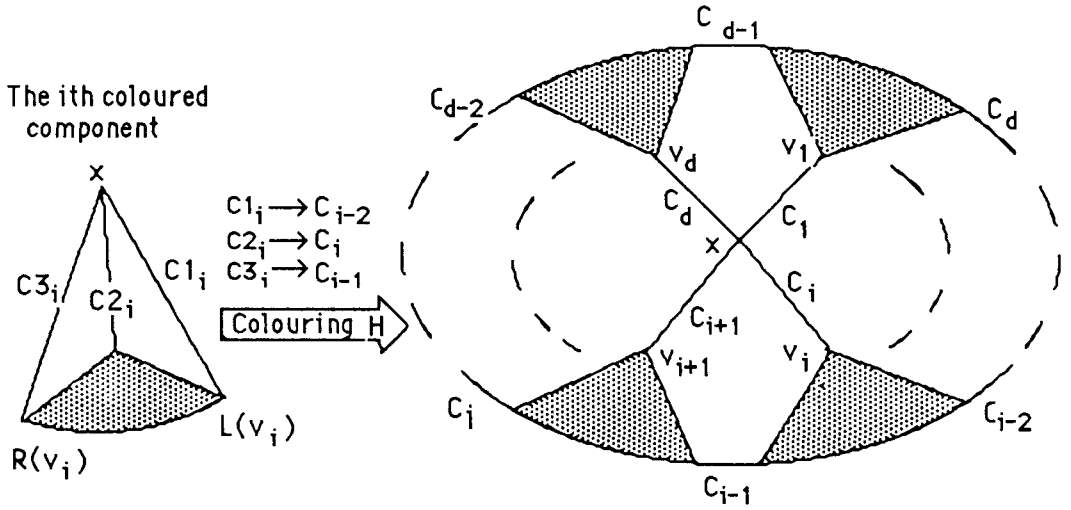


FIGURE 3.

Theorem

Any Halin graph can be optimally edge coloured in $O(\log^2 n)$ time with O(n) processors.

Proof.

Our algorithm is simply the recursive application of the decomposition and reconstruction process described. At the lowest level of recursion we colour a multigraph with n=2 and m=3, the three edges being assigned colours C_1, C_2 and C_3 in constant time. The cost of a single decomposition and reconstruction phase is O(log n). Because of fact 1, the depth of the recursion does not exceed log n and so, overall, we have an $O(\log^2 n)$ algorithm running on O(n) processors.

We need to show that the algorithm uses Δ colours. At the lowest level of recursion each multigraph uses 3 ($=\Delta$) colours, which are labelled from 1 to Δ. We therefore have a basis for an inductive proof of this statement for all n. Consider the merging of several components into a larger component. By the induction hypothesis each of the several components uses its respective value of Δ colours and these are labelled from 1 to Δ. In forming the larger component the edges adjacent to the vertex x, which is common to the several components, are coloured from 1 to d(x). New colours are introduced only if d(x) is greater than all of the Δs of the several components. In this case the larger component has Δ=d(x) and again precisely Δ colours are used and they are labelled from 1 to d(x).

This completes the proof.

Figure 4 illustrates an application of the algorithm to the specific Halin graph shown in (a). From (a) through to (d) we illustrate the decomposition of the graph for successively deeper levels of the recursion. For reasons of clarity we do not separate the graph at the articulation points; at successive levels of the recursion these are (3), (1,5) and (2,4). Also for clarity reasons we represent the multigraph with n=2 and m=3, which appears at the lowest level of recursion, as follows:

Here x is the articulation vertex which caused the creation of this multigraph. In figure 4 we understand that the edges of each such multigraph are coloured 1, 2 and 3 in a clockwise fashion about x as shown here. From (d) back to (a) figure 4 illustrates the reconstruction of the Halin graph and, at each level of recursion, the colours that are assigned to the edges. In the figure each component created in the the decomposition has an identifying letter, this is used in the following statement of the colour permutations used within the components as they are merged to form larger components. The notation is as follows, X->Y: 'permutation' means that we apply the permutation to the colours of component X in the reconstruction of component Y.

from (d) to (c)

E->J: 1->2, 2->4, 3->3
F->J: 1->3, 2->1, 3->4
G->J: 1->4, 2->2, 3->1
H->J: 1->1, 2->3, 3->2

A->D: 1->1, 2->3, 3->2
B->D: 1->2, 2->1, 3->3
C->D: 1->3, 2->2, 3->1

from (c) to (b)

D->S:1->4, 2->3, 3->2. J->T:1->3, 2->1, 3->4, 4->2. V->Y:1->2, 2->1, 3->3
K->S:1->3, 2->1, 3->4. P->T:1->1, 2->3, 3->2. W->Y: 1->3, 2->2, 3->1
L->S:1->4, 2->2, 3->1. Q->T:1->2, 2->4, 3->3. X->Y: 1->1, 2->3, 3->2
M->S:1->1, 2->3, 3->2. R->T:1->3, 2->1, 3->4

from (b) to (a)

S->Z: 1->1, 2->2, 3->4, 4->3.
T->Z: 1->4, 2->3, 3->2, 4->1.
U->Z: 1->3, 2->1, 3->4
Y->Z: 1->3, 2->2, 3->1

Remark 2.

A similar theorem holds for vertex colouring. In this case the known sequential algorithms [12] are very easily parallelisable. If all but one of the vertices are skirt vertices then the colouring is quite easy. We therefore assume that this is not the case it what follows. It is easy to colour T with two colours 1 and 2 depending on the parity of their distances from the root. Then colour clashes on the skirt C may occur. These are resolved as follows. If C has even length then we colour each second vertex on C in a single parallel step with the colour 3. Otherwise we find a vertex v all of whose sons are skirt vertices. Such a vertex is called a fan in [12]. If C is odd then there is always

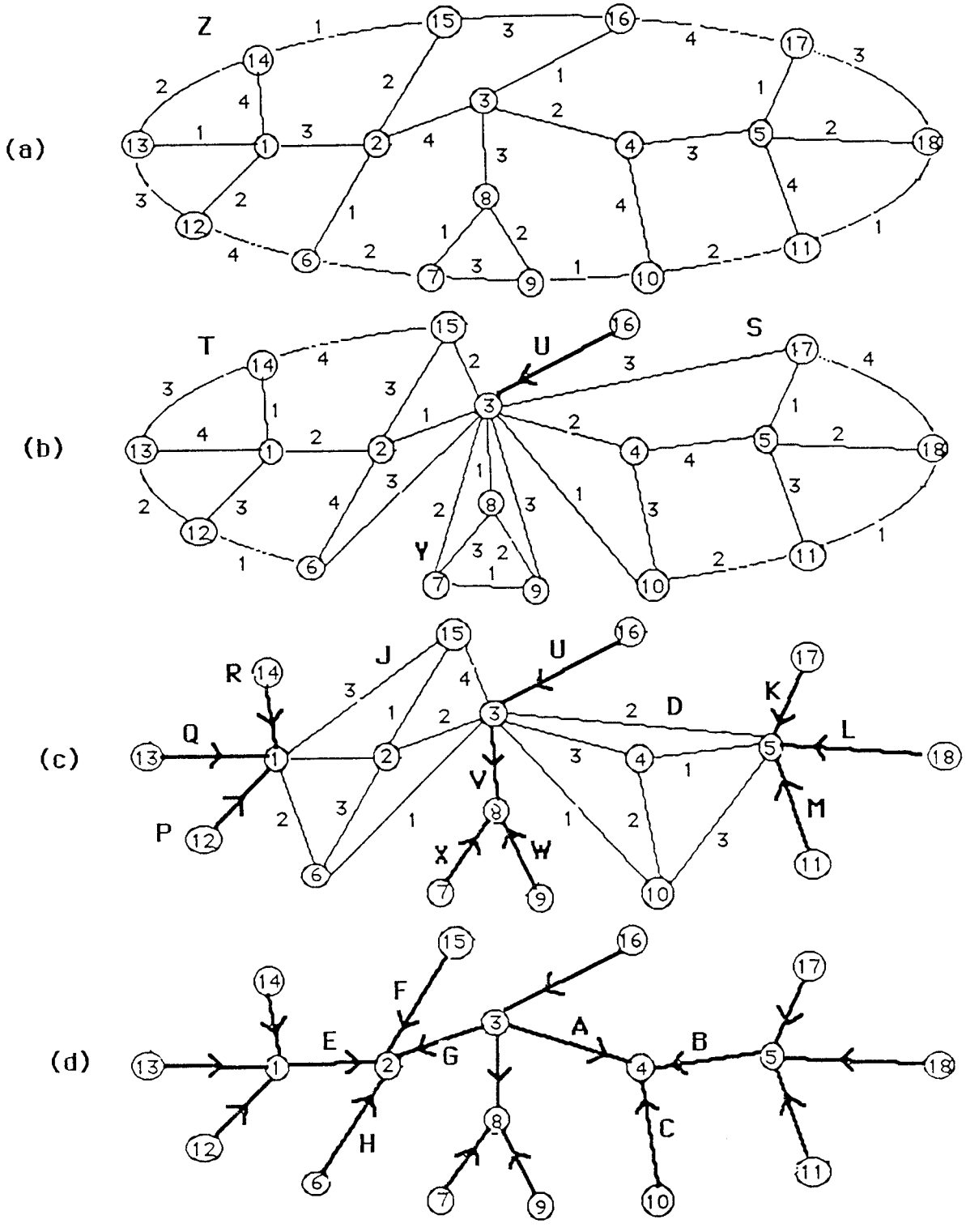


FIGURE 4.

An Application of the Edge Colouring Algorithm.

a fan v with an odd number of sons. For convenience, we imagine that the skirt vertices have been consecutively numbered so that the sons of v are the vertices 1,2....(d(v)-1), where d(v) is the degree of v. In a single parallel step, we colour v with 3, even sons of v with the colour of skirt vertex L, odd sons of v with 1 or 2 (whichever is not the colour of skirt vertex L) and even non-son-of-v skirt vertices with 3. All steps are easily achieved in log n time with O(n) processors.

References

[1] K.Diks. A Fast Parallel Algorithm for Six Colouring of Planar Graphs. To be presented at the Conference on Math.Foundations of Computer Science, Bratislava, 1986. To appear in Lecture Notes in Computer Science, Springer Verlag.

[2] S.Fiorini and R.J.Wilson. Edge-colouring of Graphs, Research Notes in Mathematics 16, Pitman, London 1977.

[3] S.Fiorini. On the Chromatic Index of Outerplanar Graphs, J.Combinatorial Theory (B) 18 (1975), 35-38.

[4] S. Fortune and J. Wyllie. Parallelism in Random Access Machines, Proceedings of the 10th ACM Symp. Theory of Comp.(1978) 114-118.

[5] H.Gabow & O.Kariv. Algorithms for Edge Colouring Bipartite Graphs and Multigraphs, SIAM J. Computing 11,1 (1982), 117-129

[6] A.M.Gibbons and W.Rytter. A fast Parallel Algorithm for Optimally Edge-Colouring of Outerplanar Graphs. Research Report No.80. Dept. of Computer Science. University of Warwick. June 1986

[7] I. Holyer. The NP-completeness of edge-colouring, SIAM J. Computing 10(1981),718-720.

[8] J.Ja'Ja' and J.Simon. Parallel Algorithms in Graph Theory: Planarity Testing. SIAM J. COMPUT. Vol.11.No.2, May 1982.

[9] G.F.Lev, N.Pippenger, and L.G.Valiant. A fast Parallel Algorithm for Routing in Permutation Networks. IEEE Tran.on Computers, C-30 (1981), 93-100.

[10].A.Proskurowski & M. Syslo. Efficient Vertex- and Edge-Colouring of Outerplanar Graphs, SIAM J.Algebraic and Discrete Methods (1986).

[11] M. Skowronski. Efficient Vertex- and Edge- Colouring of Halin Graphs, Proceedings of the 3rd Czechoslovak Symp. on Graph Theory, Prague 1982.

[12] M.Syslo. NP-Complete Problems on Some Tree-Structured Graphs: a Review, Proc. WG'83 International Workshop on Graph Theoretic Concepts in Computer Science, (M.Nagl and J.Perl Editors), 342-353 (1983).

[13] M. Syslo, A. Proskurowski. On Halin Graphs, in M. Borowiecki, J. W. Kennedy, M. Syslo (Editors). Graph Theory-Lagow 1981, LN in Mathematics, Springer-Verlag, Berlin-Heidelberg, 1983

[14] R.E.Tarjan and U.Vishkin. An Efficient Parallel Biconnectivity Algorithm. SIAM J. Comput. 14:4 (1985), 862-874.