Theoretical Computer Science 71 (1990) 401-411
North-Holland

NOTE

OPTIMALLY EDGE-COLOURING OUTERPLANAR GRAPHS IS IN NC

Alan GIBBONS

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

Wojciech RYTTER

Institute of Informatics, Warsaw University, Poland

Communicated by D. Perrin
Received November 1987
Revised May 1989

Abstract. We prove that every outerplanar graph can be optimally edge-coloured in polylogarithmic time using a polynomial number of processors on a parallel random access machine without write conflicts (P-RAM).

1. Introduction

Vizing's well-known theorem [15] states that the minimum number of colours required to (properly) edge-colour a graph 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 checking whether a particular graph is Δ-edge-colourable is NP-hard [8]. However, it is known that bipartite and outerplanar graphs are Δ-edge-colourable and there are polynomial-time algorithms colouring these graphs with the minimum (that is, optimal) number of colours [2, 3, 5, 10, 12]. In the case of bipartite graphs the nonexistence of odd cycles enables us to apply the technique of Euler partitioning (for the parallel case particularly, see [10]), and in the case of outerplanar graphs an associated tree structure of the graph is employed [12].

NC is the class of problems computable in polylog ($\log^k s$, for some constant k and problem size s) parallel time with a polynomial number of processors. It is known [9, 13] how parallelisation of the algorithm implicit in the proof of Vizing's theorem can provide a $(\Delta+1)$-edge-colouring of an arbitrary simple graph with n vertices in parallel time P, where P is a polynomial in Δ and $\log(n)$. In [9], this specifically led Karloff and Shmoys to the result that the problem of finding a $(\Delta+1)$-edge-colouring is in NC for simple graphs which have $\Delta = O(\log^{O(1)}(n))$. It was shown in [10] that for bipartite graphs, the tighter problem of optimally edge-colouring is also in NC. We show here that the similarly tighter problem of finding a Δ-edge-colouring of outerplanar graphs is also in NC. This paper is a full version of part of an extended abstract presented in [7]. In [7] it was implied that

the problems of optimally edge-colouring outerplanar and Halin graphs are both in NC. In co-authorship with Amos Israeli and since presenting [7], the present authors discovered an improved algorithm for Halin graphs which is described in [6].

It was shown in [1] that the problem of optimal vertex-colouring of outerplanar graphs is in NC. In the case of vertex-colouring, the minimum number of colours is at most 3. Our parallel algorithm implicitly describes a new linear time sequential algorithm for edge-colouring outerplanar graphs. For graphs with $\Delta = 3$ the sequential algorithm in [12] is parallelised. However, for $\Delta > 3$ we have to design an entirely new algorithm to reduce the problem to the case of $\Delta = 3$. In this reduced case, a tree of internal faces of the graph is constructed, each face is independently edge-coloured and a technique similar to that used in [1] can be applied. The initial edge-colouring of faces is more subtle than in the case of vertex-colouring, the crucial point is to satisfy a certain invariant (property P3 from [12]). The reduction from the case $\Delta > 3$ to the case $\Delta \leq 3$ is based on the following properties of outerplanar graphs: there is a node of degree at most 2, if the graph is biconnected then it has a Hamiltonian cycle and the maximum number of edges is $2n - 3$. Trees and unicycle graphs (see [11]) are special cases of outerplanar graphs.

Our model of computation is a parallel random access machine without write conflicts (P-RAM). Such a machine consists of a number of synchronously working processors (which are uniform cost RAMs) using a common memory. No two processors can write simultaneously to the same location. On the other hand, many processors can read at the same time from the same location. The action of the parallel instruction

for each x satisfying a given condition **do in parallel** instruction(x)

consists of assigning a processor to each x (for which the specified condition holds) and executing instruction(x) for all such x simultaneously. See [4] for further details.

2. Optimal edge-colouring of biconnected outerplanar graphs with $\Delta = 3$

Any biconnected outerplanar graph with at least three nodes has a planar embedding which is a polygon with noncrossing (internal) diagonals. Such a graph has exactly one Hamiltonian cycle (bounding the polygon). We call the edges of this cycle *sides* and the remaining edges are called *diagonals*. Following [1], it is easy to find the Hamiltonian cycle using the following observation: deletion of an edge (together with its endpoints) disconnects the polygon if and only if it is a diagonal. The test for connectivity can be performed in $\log^2 n$ time using $O(n)$ processors, since the number of edges is linear. $O(n)$ such tests are needed. The edges on the Hamiltonian cycle can be consecutively ordered and this ordering can be used to compute the internal faces (the set of edges entering a given node can be ordered clockwise, these local orderings can be used to compute the internal faces in logarithmic parallel time). This enables us to easily compute a structured form of

the graph: a graph of its internal faces. In this graph two faces are adjacent if and only if they have a common diagonal. This graph is a tree, denoted by TF, and it is the basis of the algorithm in this section. The parallel construction of TF together with the computation of the set of faces has been fully described in [1].

If the faces of the graph G are edge-coloured independently then many edges will be coloured inconsistently, because one edge belongs to two distinct faces. Even if each edge is coloured consistently in this sense and has the same colour in both faces, the whole colouring can be improper, because two similarly coloured but distinct edges (in separate faces) can have a common endpoint and this will violate the requirement that edges incident to the same node must have distinct colours. We start with a "locally good" colouring of faces and step by step we shall remove the inconsistencies described above. The following crucial invariant will enable us to do it.

Property P. *Each face C is properly coloured as a cycle and if there are in C three consecutive edges $e1$, $e2$, $e3$ such that $e2$ belongs to some other cycle then these edges are coloured by three distinct colours.*

It was shown in [9] that each cycle can be (independently) coloured to satisfy Property P. We parallelise the method from [9].

The first step of our algorithm consists of simultaneously and independently colouring all faces to satisfy Property P. For a given face we start from the edge joining this face to its father face and colour its edges consecutively with 1, 2, 3 (if the face has no father then we start from any edge common with some other face). After that, Property P can be violated on edges (a, b), (b, c) and (c, d) (see Fig. 1). We recolour these edges depending on the value of k modulo 3 (where k is the number of edges on the cycle). If k modulo $3 = 2$ then two situations are possible depending upon whether (a, b) belongs to some other face. If this is so then we colour edges (a, b), (b, c), (c, d) by 2, 1, 3, respectively. Otherwise we colour them 1, 3, 1. Figure 1 indicates the recolouring schemes. For the outerplanar graph G of Fig. 3, the initial colouring of the faces is presented in Fig. 4.

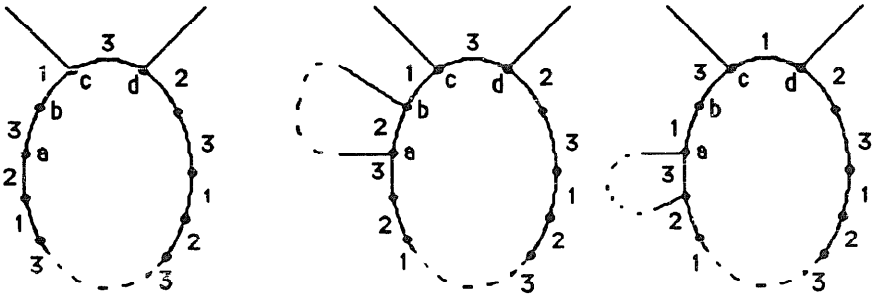

Fig. 1. The initial colouring of faces. The edges are coloured 1, 2, 3, 1, 2 ... starting with the upper edge (which is shared with the father face). Then a suitable recolouring of the edges (a, b), (b, c) and (c, d) is carried out.

For one face such a colouring can easily be done in $\log(n)$ time with $O(n)$ processors. The length of the cycle has to be computed and the edges numbered consecutively by 1, 2, 3, 1, ... This can be done by directing the cycle and breaking it at the point c. Such a numbering can be achieved for an (open) list by computing the distances (modulo 3) from each element to the end of the list using a standard doubling technique. Hence the initial colouring of faces can be done in logarithmic time with $O(n^2)$ processors, using $O(n)$ processors for each face.

Let C be a face which is not the root of the tree TF and let (e, a), (a, b), (b, f) be consecutive edges coloured $x1$, $y1$, $z1$, respectively. Here (a, b) is the edge common to C and its father face C'. The edges (d, a), (a, b), (b, c) are consecutive edges of the face C'. We denote their colours by x, y, z, respectively (see Fig. 2).

Let sub(C, h) be a subgraph of the outerplanar graph consisting of all faces which are in the subtree (of TF) rooted at C and whose distance from C (measured as a number of faces on the path to C in the tree TF) is not larger than h. For example sub$(C, 1) = C$.

We define the operation recolour(C, h). This operation consists of simultaneously replacing in sub(C, h) each occurrence of colour $x1$ by z, $y1$ by y and $z1$ by x. In other words we perform the assignment of colours $(x1, y1, z1) \leftarrow (z, y, x)$ in the set of faces which are in the maximal subtree of height at most h rooted at C (in TF). For example if we execute recolour$(C, 1)$, for one face C only, then C and its father

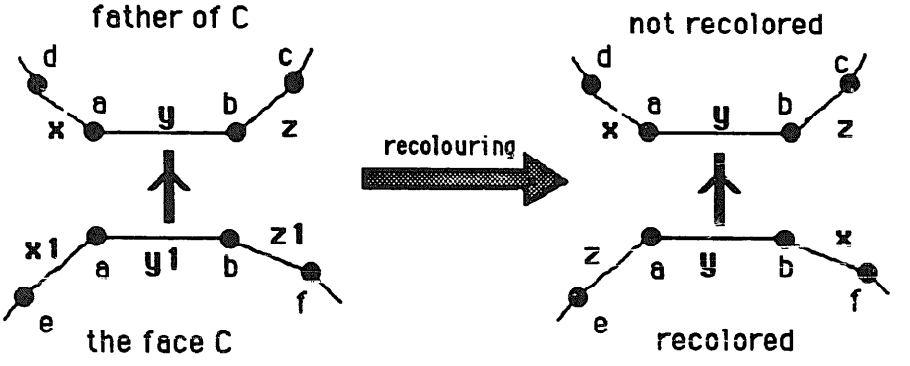

Fig. 2. A subtree rooted at the lower face is recoloured by executing the assignment $(x1, y1, z1) \leftarrow (z, y, x)$.

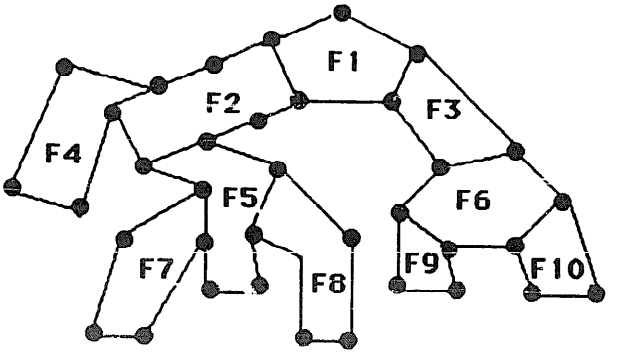

Fig. 3. The outerplanar graph G.

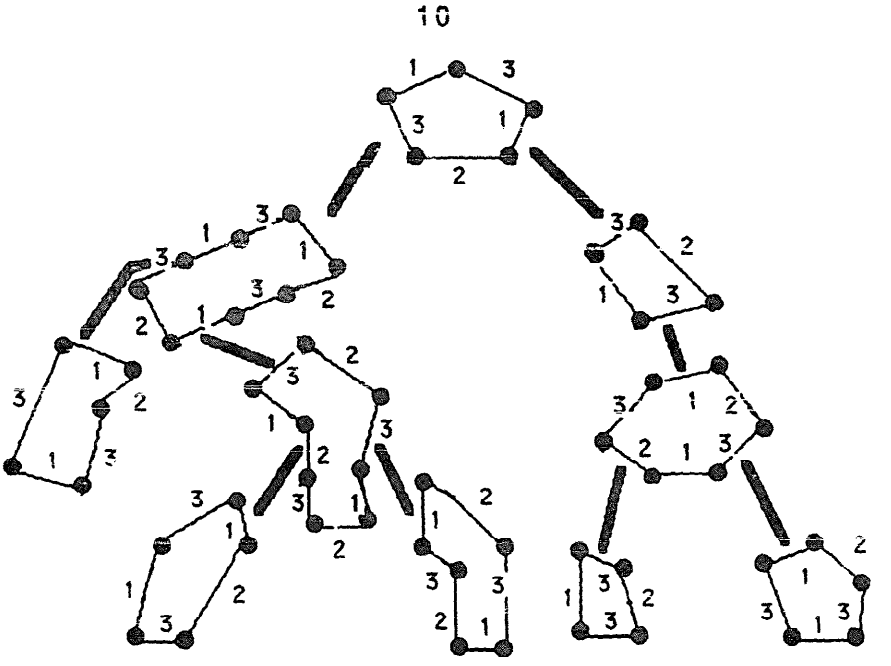

Fig. 4. The tree of faces rooted at $F1$.

face are consistently coloured. Notice that the operation recolour is well defined if the invariant P holds. Moreover, this operation preserves the invariant P. Also P guarantees that the assignment of colours defined above is a permutation of colours.

Now we use a divide-and-conquer approach to consistently colour the whole graph G. We decompose TF into smaller subtrees of faces, colour them recursively by the same method and then use the operation recolour to agree between colours of these subtrees of faces. The tree TF can be decomposed in many ways and some variations of the same schema are possible (for example one can find a node which splits the tree into much smaller subtrees). We use the following method. First compute the depth of each node (face) of TF, as a number of faces to the root. Let h be the height of TF. Assume for simplicity that h is a power of two (some dummy layers can be added if necessary). If $h = 1$ then the graph consists of only one face which is already properly coloured during the initialization and the algorithm stops. Otherwise we disconnect all faces of depth $h/2$ from their sons. TF is decomposed into a set of subtrees, each of depth at most $h/2$. We colour these subtrees recursively. Denote by R the set of faces of depth $(h/2+1)$. After that we execute in parallel for each face C belonging to R: recolour(C, $h/2$). Now the whole tree of faces is consistently coloured and this gives the edge-colouring of G and the algorithm stops.

The recursion has depth $\log(h) = O(\log(n))$, and the decomposition and recolouring can be done in $\log(n)$ time. Thus such an algorithm runs in $\log^2 n$ time with $O(n^2)$ processors.

An iterative version of this algorithm is provided by implementing the recursion in a bottom-up manner (in the recursion tree), by first combining the trees of height

1, then the trees of the height 2, 4, 8, ..., as follows:

for $k = 1$ **to** $\log(h)$ **do**
  for each face C such that $(\text{depth}(C) - 2^{k-1} - 1) \bmod 2^k = 0$
    do in parallel recolour(C, 2^{k-1}).

Figure 5 presents the colouring of faces after performing one iteration with $k = 1$, and Fig. 6 shows the final colouring. Observe how recolour($F5$, 2) works when proceeding from the colouring of Fig. 5 to that of Fig. 6.

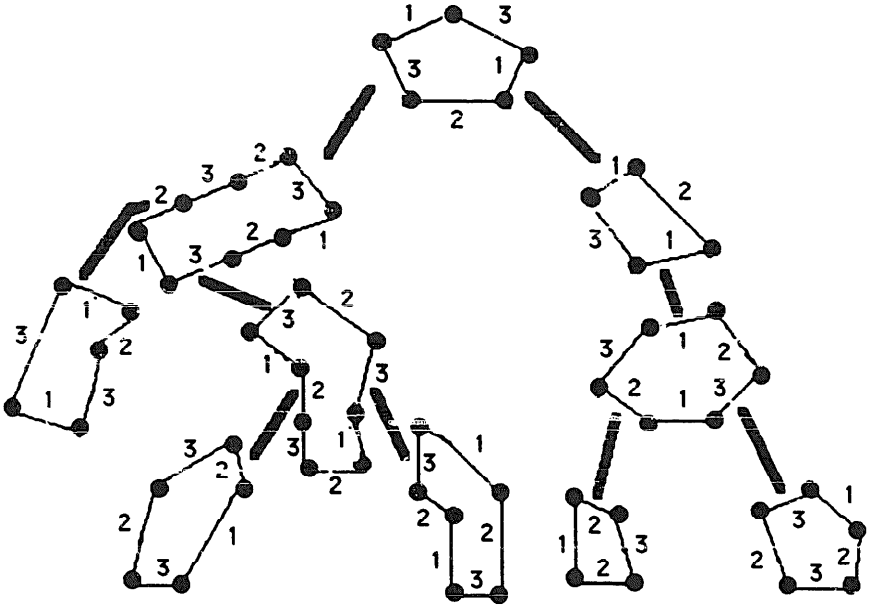

Fig. 5. The colouring of faces after executing recolour(F, 1) for $F = F2, F3, F7, F8, F9, F10$ (in parallel).

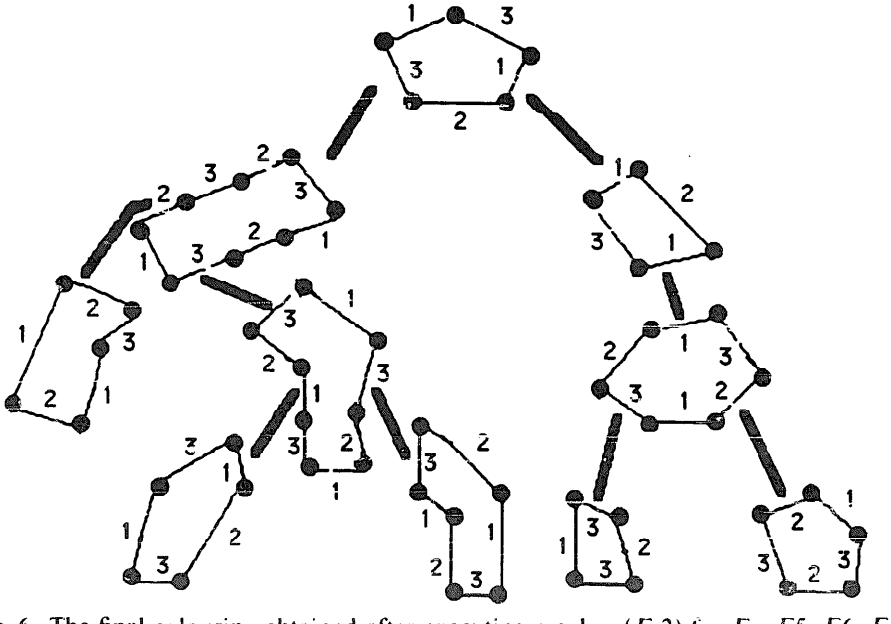

Fig. 6. The final colouring obtained after executing recolour(F, 2) for $F = F5, F6, F4$.

The operation recolour(C, q) takes O(1) parallel time, if we preprocess the tree in such a way that for each face we can test in O(1) time whether it is a member of sub(C, q). It is enough to compute the table of distances between each pair of faces of TF. This problem can be generally solved for a tree in $\log(n)$ parallel time with $O(n)$ processors. As the tree of faces can be computed in $O(\log^2 n)$ parallel time with n^2 processors (see [1] or our discussion above, when describing the tree TF), we now have the following result.

Lemma 2.1. *Every outerplanar biconnected graph with $\Delta = 3$ can be edge-coloured using three colours in* $O(\log^2)$ *parallel time with* $O(n^2)$ *processors on a* P-RAM.

3. Optimal edge-colouring of general outerplanar graphs

Let G be a biconnected outerplanar graph with $\Delta > 3$. The outerplanar graph reduced(G) is obtained in the following way. We find a node c of degree 2. Let (a, c), (c, b) be the edges incident with c. Let $H = (c, b), (b, d), (d, f), (f, g), \ldots,$ $(q, a), (a, c)$ be the sequence of consecutive sides of the polygon. We mark each second edge of this sequence starting with (b, d). Notice that the edge (c, b) is always unmarked and that the edge (a, c) is marked if and only if n is even. Each node, except perhaps c, is incident with exactly one marked edge (see Fig. 7a). The set of marked edges is a maximum matching (possibly not matching node c) which we denote by $M(G)$. The graph reduced(G) is obtained by removing all the edges of $M(G)$ from G.

Figure 7a shows an example outerplanar graph with $\Delta > 3$ and the vertices are labelled as described in the preceding paragraph. The edges of $M(G)$ are heavily scored. The first such edge is (b, d) and the last is (q, a). In this example both (a, c) and (c, b) are unmarked. Hence the degree of node c in Fig. 7a is not decreased in reduced(G). However the degree of every other vertex is decreased by 1, and the maximal degree of the whole graph is therefore also decreased by 1. The graph G has 28 edges, but reduced(G) has only 20 edges. Such a high reduction in the number of edges is a general property of the operation.

Lemma 3.1. *Let Δ' be the maximal degree and m' be the number of edges of the graph reduced(G), for a biconnected outerplanar graph G having maximal degree $\Delta > 3$ and m edges. Then $\Delta' = \Delta - 1$ and $m' \leq \frac{3}{4}m$. The graph reduced(G) can be constructed in* $O(\log n)$ *parallel time with* $O(n)$ *processors, where n is the number of nodes of G.*

Proof. Each node of G is an endpoint of some edge in $M(G)$, except perhaps node c which has degree 2. Hence the maximal degree decreases, since the degree of every other node is reduced by 1. There are at least $(n-1)/2$ edges in $M(G)$ and all of them are removed. The maximal number of edges of the outerplanar graph

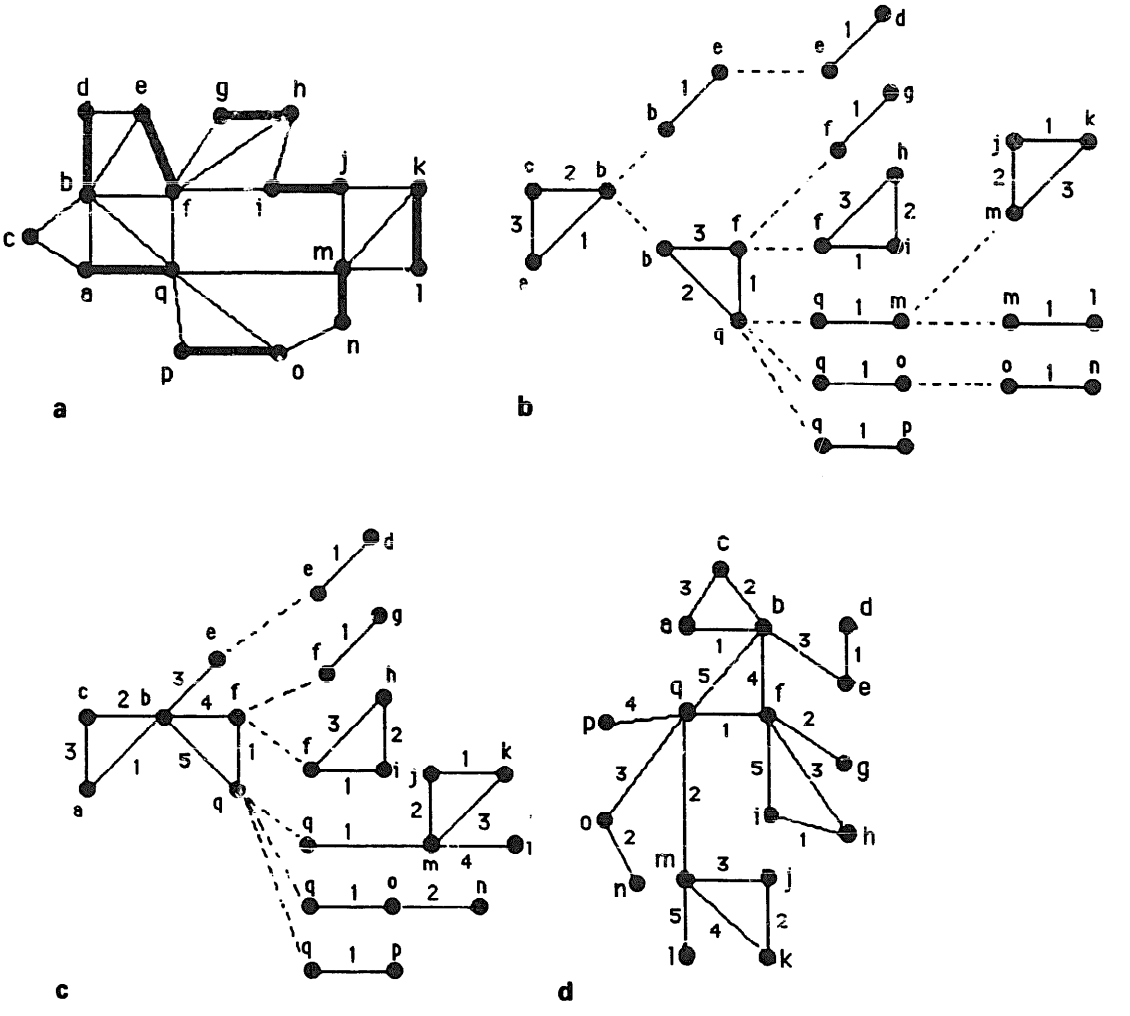

Fig. 7.

is $2n-3$. This implies $m' \leq \frac{3}{4}m$. A suitable node c can be easily found in $\log(n)$ time. The marking can be made in $\log(n)$ time with n processors using a standard doubling technique. This completes the proof. □

Notice that after removing $M(G)$ the maximum degree of G decreases by 1, however degrees in biconnected components can decrease by more than 1. For example the maximal degree of biconnected components after deleting edges of $M(G)$ in Fig. 7a decreases by 2, as can be seen in Fig. 7b which shows the tree of biconnected components. Notice that this tree can easily be found within the complexity constraints we require, by employing the biconnected components algorithm of [14]. Let TB denote the tree of biconnected components. We use the tree TB to agree colour inconsistencies between two connected components which we presume to have been independently coloured. The method we use is one of iterating the process of making components at odd levels in the tree agree with their fathers and then merging these components with their fathers. This halves the height

of the tree in each iteration and so a logarithmic number of iterations is sufficient. Therefore we just need to see how we can bring about agreement between a biconnected component and its father. Let Δ be the maximum degree of reduced(G) for which TB has been constructed. We assume that each biconnected component is already edge-coloured using colours from within the set $\{1, \ldots, \Delta\}$. Consider a father vertex F in TB and an articulation point v which F has in common with the sons $S_1, S_2, \ldots, S_r$. Let $P(S_i)$ denote the set of colours present at S_i in F and let $d(S_i)$ be the degree of v in S_i. The problem is to extract from the set $\{1, \ldots, \Delta\} - P(F)$ a number of disjoint subsets with cardinalities $d(S_i)$ for all i, $1 \leq i \leq r$. Then taking a one to one correspondence between colours in the subset with cardinality $d(S_i)$ and colours in $P(S_i)$ will define a colour exchange in S_i which for all i will remove inconsistencies. This is because no colour in $\{1, \ldots, \Delta\}$ can then appear more than once at v. Within a single iteration in which the height of TB is halved, this agreeing of colours between all components at odd levels with their fathers can be achieved in $O(\log(n))$ time using n^2 processors. The details, although tedious, are not difficult and so we omit them. Thus we can remove colouring inconsistencies over the whole of TB in $O(\log^2(n))$ time using n^2 processors.

If we call this process ADJUST, then the procedure for optimally edge-colouring a biconnected outerplanar graph with maximum degree Δ is:

```
procedure edge-colour(G, D)
begin {Δ ≥ 3, G is a biconnected graph, the procedure colours G using
  colours from [1, ..., D], D ≤ Δ}
if Δ ≤ 3 then use the algorithm described earlier for this case else
  begin find M(G)
  colour every edge in M(G) with the colour D
  G ← reduced(G)
  find the biconnected components of G and construct TB
  for each biconnected component X in parallel do
    edge-colour(X, D − 1)
  ADJUST
  end
end
```

We illustrate the procedure ADJUST with reference to the tree of biconnected components shown in Fig. 7b. We take the root of the tree to be the component which is the circuit (a, b, c). Let the edges of the biconnected components be coloured as indicated in Fig. 7b. In Fig. 7c the inconsistencies between components at odd levels in the tree have been removed and components at these odd levels have been merged with their fathers. In Fig. 7d the process has been repeated once more and an optimal colouring of reduced(G) has been obtained, where G is the graph of Fig. 7a. The edges of $M(G)$ can now be coloured with the colour 6 so that an optimum colouring of G is obtained.

We are now in a position to present our main result.

Theorem 3.2. (a) *Every outerplanar graph can be optimally edge-coloured in* $O(\log^3(n))$ *time with* n^2 *processors.*

(b) *If* G *is a biconnected outerplanar graph with* $\Delta \geq 3$, *then edge-colour*(G, Δ) *optimally edge-colours* G *in* $O(\log^3(n))$ *time with* n^2 *processors.*

Proof. (b): Lemma 3.1 implies that the depth of the recursion is $O(\log(n))$. The first operation, if $\Delta > 3$, is the removal of $M(G)$ and this decreases the number of edges by a factor of $\frac{3}{4}$. Within one recursive level of edge-colour, for a graph with n vertices, all operations (constructing the tree of biconnected components and the operation ADJUST) take $O(\log^2(n))$ time with n^2 processors. An optimal colouring is produced because, as we saw earlier, the base cases of the recursion produce optimal colourings and in other cases when Δ is increased by 1 (with the addition of the edges in $M(G)$) so is the number of colours used. Thus (b) is proved.

(a): If $\Delta \geq 3$ and the graph is not biconnected then we can decompose G into biconnected components and execute edge-colour for each component in parallel. Then we can apply the operation ADJUST. Then (a) follows from (b). That completes the proof. □

Remark 3.3. Consider replacing in the procedure edge-colour all instructions of the form **for each ... do in parallel** by the corresponding sequential instruction **for each ... do**. Then edge-colour(G, Δ) can be easily implemented to colour G in linear time. The crucial point is that after each removal of $M(G)$ the number of edges decreases by a factor $\frac{3}{4}$. The number of edges is linear with respect to n (the number of nodes) and the tree TB can be computed in linear time. The operation ADJUST can be easily implemented to run in linear time by traversing the tree TB in breadth first manner, making suitable recolourings of the biconnected components encountered (nodes of TB). Moreover, the sequential colouring of graphs with $\Delta = 3$ can be done much easier than using the approach in Section 2. The inductive argument used in [3] for the case $\Delta = 3$ can be used to obtain a very simple linear time algorithm for this case (by removing, inductively, a node with degree two and identifying neighbours of such node). This gives, for the general case, quite a different linear time algorithm than that presented in [12]. The trick of traversing the tree of faces in a suitable order (described in [12]) is omitted.

References

[1] K. Diks, A fast parallel algorithm for six-colouring of planar graphs, in: J. Grunska, B. Rovan and J. Wiedermann, eds., *Proc. 12th Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science **233** (Springer, Berlin, 1986) 273-282.

[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. Combin. Theory (B)* **18** (1975) 35-38.

[4] S. Fortune and J. Wyllie, Parallelism in random access machines, in: *Proc. 10th ACM Symp. Theory of Computing* (1978) 114-118.

[5] H. Gabow and O. Kariv, Algorithms for edge colouring bipartite graphs and multigraphs, *SIAM J. Comput.* **11** (1) (1982) 117-129.

[6] A.M. Gibbons, A. Israeli and W. Rytter, Parallel O(log(n)) time edge-colouring of trees and Halin graphs, *Inform. Process. Lett.* **27** (1988) 43-51.

[7] A.M. Gibbons and W. Rytter, Fast parallel algorithms for optimal edge colouring of some tree structured graphs, in: *Proc. Foundations of Computation Theory*, Kazan, Russia, 1987, Lecture Notes in Computer Science **278** (Springer, Berlin, 1987) 155-162.

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

[9] H.J. Karloff and D.B. Shmoys, Efficient parallel algorithms for edge coloring problems, *J. Algorithms* **8** (1987) 39-52.

[10] G.F. Lev, N. Pippenger and L.G. Valiant, A fast parallel algorithm for routing in permutation networks, *IEEE Trans. Comput.* **30** (2) (1981) 93-100.

[11] S.L. Mitchell and S.T. Hedetniemi, Linear algorithms for edge-colouring trees and unicycle graphs, *Inform. Process. Lett.* **9** (3) (1979) 110-112.

[12] A. Proskurowski and M. Syslo, Efficient vertex- and edge-colouring of outerplanar graphs, *SIAM J. Algebraic Discrete Methods* **7** (1986) 131-136.

[13] O.A. Ogunyode and A.M. Gibbons, Parallel algorithms for approximate edge-colouring of simple graphs, in: *Proc. 7th Internat. Conf. on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science **285** (Springer, Berlin, 1987).

[14] R.E. Tarjan and U. Vishkin, Finding biconnected components and computing tree functions on logarithmic parallel time, *SIAM J. Comput.* **14** (4) (1985) 862-874.

[15] V.G. Vizing, On an estimate of the chromatic class of a p-graph (in Russian), *Diskret. Anal.* **3** (1964) 25-30.