Research report 105

PARALLEL O(LOG(N)) TIME EDGE-COLOURING OF TREES AND HALIN GRAPHS

Alan Gibbons[1], Amos Israeli[2] and Wojciech Rytter[1,3]

(RR105)

Abstract

We present parallel O(log(n))-time algorithms for optimal edge colouring of trees and Halin graphs with n processors on a a parallel random access machine without write conflicts (P-RAM). In the case of Halin graphs with a maximum degree of three, the colouring algorithm automatically finds every Hamiltonian cycle of the graph.

(1) Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK

(2) Aiken Computation Lab, Harvard University, Cambridge MA 02138. This work was in part supported by the Weizmann fellowship and NSF grant DCR-86-0636.

(3) Institute of Informatics. Warsaw University, 00-901 Warsazawa, PO Box 1210, Poland.

June 1987

PARALLEL O(LOG(N)) TIME EDGE-COLOURING OF TREES AND HALIN GRAPHS

Alan Gibbons[1], Amos Israeli[2] and Wojciech Rytter[1,3]

Abstract

We present parallel O(log(n))-time algorithms for optimal edge colouring of trees and Halin graphs with n processors on a parallel random access machine without write conflicts (P-RAM). In the case of Halin graphs with a maximum degree of three, the colouring algorithm automatically finds every Hamiltonian cycle of the graph.

Key Words: edge colouring, trees, Halin graphs, parallel computations.

1. Introduction

Many parallel algorithms are based on efficient parallel computations on trees. We describe a simple method for edge colouring trees and show how an optimal parallel edge colouring of trees can be extended to optimal edge colouring of Halin graphs. 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 (an example of such a graph is presented in figure 1).

Halin graphs are related to another class of tree-like graphs: outerplanar graphs. These and Halin graphs fall between trees and more general graphs in that trees are reflected in their structure. 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 outerplanar 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 form 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]. Halin graphs have many interesting combinatorial properties, see [12].

(1) Department of Computer Science, University of Warwick, Coventry CV4 7AL, England.

(2) Aiken Computation Lab., Harvard University, Cambridge Ma 02138. This work was in part supported by the Weizmann fellowship and NSF grant DCR-86-0636.

(3) Institute of Informatics, Warsaw University, 00-901 Warszawa, PO Box 1210, Poland.

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 polylogarithmic (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 bipartitie 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 [11]. It was shown in [1] that the problem of optimal vertex-colouring 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 [11]. We assume that the Halin graph is described as a tree with its skirt C. The edges of the cycle C can be ranked and later (using the techniques of [13]) the sons of each vertex of T can easily be ordered according to the order of their leaf-descendents on C.

2. Edge colouring trees

Let Δ denote the maximum degree of the graph. We start with edge-colouring of the tree T using Δ colours 0, 1,..., Δ-1. Let $Q = \{0, 1, ..., \Delta-1\}$ and let $x \otimes y$ denote the operation $(x+y) \bmod \Delta$. We will be using the following obvious fact as a basis for legal edge colourings.

Fact 1

For every element x in Q all the elements $x, x\otimes 1, x\otimes 2, ..., x\otimes(\Delta-1)$ are different.

One of the most useful and general techniques for parallel computations on trees is the Euler tour technique due to Tarjan and Vishkin [13]. Using this technique the tree T can be rooted and for each vertex we can know its father and its sons. We can add one additional ingoing edge for the root and colour it using the colour 0. Hence each node v has exactly one ingoing edge (father(v),v) and up to Δ-1 outgoing edges. Let us label for each vertex v its outgoing edges (to the sons) by different elements from $\{1,..., \Delta-1\}$. Let F(v) be the sum (under $\otimes$) of all labels of edges on the path from the root to v, including the additional ingoing edge of the root. For each vertex v, in

parallel, we colour the edge entering v by F(v). Notice that the added incoming edge to the root may increase Δ, however we shall still obtain an optimal colouring of the original tree.

Insert Figure 1 and Figure 2.

Such a colouring from the labelling of figure 1 is shown in figure 2. At this moment ignore the bold edges of the skirt and the fact that (in figure 1) the outgoing edges are labelled 1,2 or 2,1 depending upon the parity of their distances from the root. The example tree has Δ=3. For such a tree any labelling which initially uses 1,2 for outgoing edges will provide a proper colouring through F(v). However we shall see later that for Δ=3 it is crucial to have a labelling which allows an extension of the colouring of the tree edges to skirt edges.

It follows from Fact 1 that this colouring of tree edges (for all Δ) is a legal edge colouring.

Now we just have to show how the function F can be computed efficiently. We describe two algorithms using two classical methods: parallel prefix computation for the first algorithm and the doubling technique for the second.

The first algorithm is based on the Euler tour technique. In this technique a tree T is turned into a directed Eulerian graph by replacing each edge of T by two antiparallel edges. Then the list $L=(e_1,e_2,...e_m)$ of directed edges is constructed in order of an Euler tour. This list can be turned into a vector easily using n processors in log(n) time. Each undirected edge (v,father(v)) is present twice on L, once as (v,father(v)) and once as (father(v),v). Let label(father(v),v) be the label initially assigned to the undirected edge (father(v),v) and let label(v,father(v)) = -label(father(v),v). Observe that the sum of labels on any cycle is zero. If the ingoing edge for the vertex is e_k, then it is easy to see the following:

$$F(v) = label(e_1) \otimes label(e_2) \otimes label(e_3) ..... \otimes label(e_k).$$

However this reduces to a prefix computation which can easily be performed in log(n) parallel time with n processors. The most time consuming part of the algorithm is the construction of the list L, however such a list should be constructed anyway to root the tree.

In the second algorithm we introduce the table pred(v), which stands for predecessor of v. Initially pred(v)=father(v) and F(v)=label(v,father(v)) for each nonroot node v. The following now computes the value of F(v):

```
repeat log(n) times
    for each nonroot node v in parallel do
        begin
        F(v)←F(pred(v))⊗F(v),
        pred(v)←pred(pred(v))
        end
```

This finishes the description of parallel edge colouring trees in O(log(n)) time using n processors. Notice that while the algorithm using the doubling technique is perhaps conceptually easier, it does essentially involve concurrent reads. On the other hand the Euler tour method can be implemented on an EREW P-RAM.

3. Edge colouring Halin graphs with $\Delta \geq 4$

We proceed now to the edge colouring of Halin graphs. The algorithm will first edge colour the tree and then the skirt. For edge-colouring of the tree T we can use either of the two previously decribed algorithms.

Any vertex of T which is of distance one from C is chosen to be the root of T. The edge (root',root) from the skirt C to the root is the special edge entering the root. We can image that this edge is temporarily removed from the tree. The colour of this edge is 0.

After the edge-colouring of T, in the case $\Delta \geq 4$, the colouring can be extended to the skirt C. This may require recolouring some edges of T which join one of its vertices to the skirt.

Insert Figure 3

Such an extension is not always possible for $\Delta=3$ (see figure 3).

Theorem 1

Every n-vertex Halin graph with $\Delta \geq 4$ can be edge-coloured with Δ colours in O(log(n)) time using n processors.

Proof.

In view of the previous section we can assume that the tree T is already coloured with Δ colours. We show how to extend this to the skirt. First we consider the case $\Delta \geq 5$, for which a particularly simple algorithm is possible. However, the (more complicated) method for $\Delta=4$ works also for any $\Delta \geq 4$.

Case 1: $\Delta \geq 5$

The edges of C are ranked, starting from any arbitrary node, and coloured in three stages. In the first stage all odd numbered edges except the last one, if it is odd, are coloured. In the second stage all even numbered edges are coloured. In the last stage the last edge, if its number was odd, is coloured. Each of these sets is composed of non adjacent edges, and each of these edges has at most four neighbouring edges. Hence each of these edges can be coloured by one colour from {0,1,2,3,4}. Since the edges in each set are not adjacent all edges in each set can be coloured in constant parallel time.

Insert figure 4 and figure 5

Case 2: $\Delta=4$

Take any vertex v of T all of whose sons lie on C (that is, are leaves of T), see figure 5. Assume that v has three sons v1, v2 and v3 (the case of two sons can be considered analogously). Let w_1 be the first vertex to the right of v3. Also let $v3=w_0,w_1,w_2,\ldots,w_k,w_{k+1}=v1$ be the sequence of skirt vertices between v3 and v1 in anticlockwise order. Let a_i be the colour of the tree edge incident to w_i. For each edge $e_i=(w_i,w_{i+1})$ we construct a function $f_i:\{0,1,2\}\rightarrow\{0,1,2\}$, associated with this edge. If $f_i(x)=y$, then the colouring of the edges e_{i-1} and e_i respectively with the colours x and y is legal at vertices w_i and w_{i+1}. This assumes that the skirt edge following e_i is not coloured at this moment (see figure 4).

Let F be the composition of functions $f_1 \bullet f_2 \bullet \ldots \bullet f_k$. The function F can be computed using a parallel prefix computation.

We remove colours from the edges (v,v1), (v,v2) and (v,v3). Let y be the colour of the edge entering v. We colour the edge (v3,w1) with any colour x different from y and a_1. Then we colour the edge $e_k=(w_k,v1)$ with the colour z=F(x). We have a legal colouring. However, the edges outgoing from v have yet to be coloured. It is an easy matter to see that we can always extend the colouring to these edges, see figure 5. This additional operation requires O(1) time. This completes the proof. □

4. Edge colouring and finding Hamiltonian cycles for Halin graphs with $\Delta=3$

In the case $\Delta=3$ the initial labelling of outgoing edges of T is crucial. Let Left(v) and Right(vl) denote, respectively, the left and right son of v. For each edge (u,w) of C, where u≠root' and w≠root', we denote by LCA(u,w) the lowest common ancestor of u and w in T. Let dist(v) be the distance of the vertex v from the root. The structure of the algorithm is as follows:

Algorithm EDGECOLOUR; {the algorithm edge colours Halin graphs with $\Delta=3$}

Step 0

Using the Euler tour technique [13] root the tree T at a vertex root of distance one from C. Let root'

be the vertex on C adjacent to root. Colour the edge (root,root') with colour 0.

Step 1

for each vertex v≠root' of T in parallel do:

label the outgoing edges from v using numbers 1,...Δ-1 in such a way that if dist(v) is even then the label of Left(v) is 1 and the label of Right(v) is 2, otherwise the label of Left(v) is 2 and the label of Right(v) is 1.

Step 2

Edge colour T using any of the two previously presented algorithms for tree colouring.

Step 3

for each skirt edge (u,w) where u≠root' and w≠root' in parallel do:

colour (u,w) with the colour of the edge entering the vertex LCA(u,w).

Step 4

Colour consistently the two skirt edges incident with root' {which is always possible, theorem 2}

end of the algorithm.

Theorem 2

Algorithm EDGECOLOUR computes a legal edge colouring of any Halin graph G with Δ=3 in O(log(n)) time using n processors.

Proof

Consider any tree vertex v. Let a be the colour of the edge entering v. Also let b,c be respectively the colours of the edges to left and right sons v1, v2 of v. Let p,w respectively be the leftmost leaf descendants of v1, v2. Also let u,q be the rightmost descendants of v1, v2. Let S(x,x') denote the set of colours of the edges on the branch from vertex x to x'. We refer to figure 6 for other notations.

Insert figure 6

After completing the algorithm the following invariant is satisfied (see figure 6):

$$a \notin S(v1,u),\ a \notin S(v2,w),\ c \notin S(v,p),\ b \notin S(v,q).$$

Also the colour of (u,w) is the same as that of the edge entering vertex v=LCA(u,w).

This invariant follows from our initial labelling of outgoing edges. If we have two consecutive edges (x1,x2) and (x2,x3) such that x2,x3 are both left or both right sons then one of these edges has label 1 and the other has label 2. Hence the colour of the edge (x2,x3) is the same as the colour of the edge entering x1, because (1+2) mode 3 = 0. This implies that on left (right) branches we have two colours alternating and the invariant follows by considering the sets S(v1,u), S(v2,w), S(v,p) and S(v,q), see figure 6.

Let u2, w2 be respectively the father of u,w. The colour of the edge (u1,u), (w,w1) is respectively the same as the colour of the edge entering the vertex u2, w2. However the colours of edges

entering u2 and w2 are, respectively in S(v1,u), S(v2,w). According to the invariant none of these sets contains the colour a. Hence none of the edges (u1,u) and (w,w1) has colour a. This proves that the colour a of the edge (u,w) is locally legal. The same reasoning applies for each skirt edge not incident with root'. If v=root then we have now p1=q1=root', see figure 6. The edge (p,root') can be coloured with c and the edge (q,root') can be coloured with b. This proves that the colouring of step 4 is possible. This completes the proof of theorem 2. □

Theorem 3

Assume that Δ=3. Then a Hamiltonian cycle of a given Halin graph G can be found in O(log(n)) time using n processors. After executing the algorithm EDGECOLOUR the set of edges coloured by 0 or by 1 gives such a cycle.

Proof

It is enough to prove the second statement. Assume that G is a Halin graph with Δ=3 and is properly coloured using three colours. Let C(a,b) be a component of G containing all vertices reachable from the root by edges coloured a or b. First we prove the following:

Claim:

For every two different colours a and b, the component C(a,b) contains all vertices of the graph.

The proof of the claim is by induction on the number n of vertices. The smallest Halin graph with Δ=3 has four vertices and it is a wheel with three vertices on the skirt. It is easy to see that the claim holds for such a graph.

Assume now that n>4 and that the claim is true for Halin graphs with n-1 vertices. Let G be an n-vertex Halin graph with Δ=3 which is properly coloured. There is an internal vertex v≠root such that both sons v1 and v2 of v lie on the skirt. Let us contract the triangle [v,v1,v2] into a single vertex u. The new vertex is adjacent to all vertices adjacent to any of v, v1, v2 and the colours of the edges are preserved. The resulting graph G' is a Halin graph with n-1 vertices. The colouring of G gives also a proper colouring of G', because the three edges connecting the triangle [v,v1,v2] to other vertices of G are coloured by three different colours. By an inductive argument all vertices of G' (including the vertex u) are in C'(a,b), where C'(a,b) is the set of all nodes reachable from the root in G' by edges coloured a or b. This proves that all vertices, except perhaps v, v1 and v2 are in C(a,b). It is enough to prove that the vertices v, v1 and v2 are also in C(a,b). However one of the vertices v, v1, v2 is in C(a,b) because u is in C'(a,b). It is easy to see that if any vertex of a triangle is in C(a,b) then all vertices of this triangle are in C(a,b). Hence all vertices of G are in C(a,b). This completes the proof of the claim.

Let us take all edges coloured by 0 or 1. It is easy to see that each component of the resulting subgraph is a simple cycle (a vertex can appear only once in a given cycle). However C(0,1) contains all vertices of G. Hence we have only one big cycle C=C(0,1) containing all vertices. Clearly C is a Hamiltonian cycle. This completes the proof of the theorem. □

Corollary

For $\Delta=3$ the algorithm EDGECOLOUR provides every Hamiltonian cycle of the Halin graph.

Proof

Let G be a Halin graph with $\Delta=3$. A straightforward inductive argument (similar to that employed in the proof of Theorem 3) shows that the edge-colouring of G found by the algorithm EDGECOLOUR is (within a renaming of the colours) the only legal edge-colouring of G. this provides three Hamiltonian cycles C(0,1), C(0,2) and C(1,3). These are the only Hamiltonian cycles of G. Suppose that to the contrary that there exists another Hamiltonian cycle H. This would provide an optimal edge-colouring of G different from that provided by the algorithm EDGECOLOUR and so we would have a contradiction. The colouring provided by H is obtained by alternately colouring edges along H with 0 and 1. The remaining edges of G are then coloured with 2. □

References

[1] K.Diks. A Fast Parallel Algorithm for Six Colouring of Planar Graphs. Math. Foundations of Computer Science, Bratislava, 1986. 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 and 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. Computing 11, 2 (May 1982).

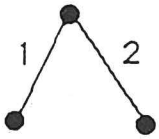
[9] G.F.Lev, N.Pippenger and L.G.Valiant. A Fast Parallel Algorithm for Routing in Permutation Networks. IEEE Tran. on Computers, C-30 (1981), 93-100.

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

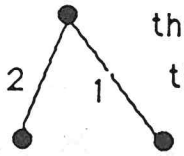
[11] M.Syslo. NP-Complete Problems on Some Tree-Structured Graphs: A Review, Proc. WG'83 International Workshop on Graph Theoretic Concepts in Computer Science, (M.Nagl and J.Perl Editors), 342-353 (1983).

[12] M.Syslo and A.Proskurowski. On Halin Graphs, in M.Borowiecki, J.w.Kennedy, M.syslo (Editors). Graph Theory-Lagow 1981, LN in Maths, Springer-Verlag, Berlin-Heidelberg, 1983.

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



the distance from the root is even



the distance from the root is odd

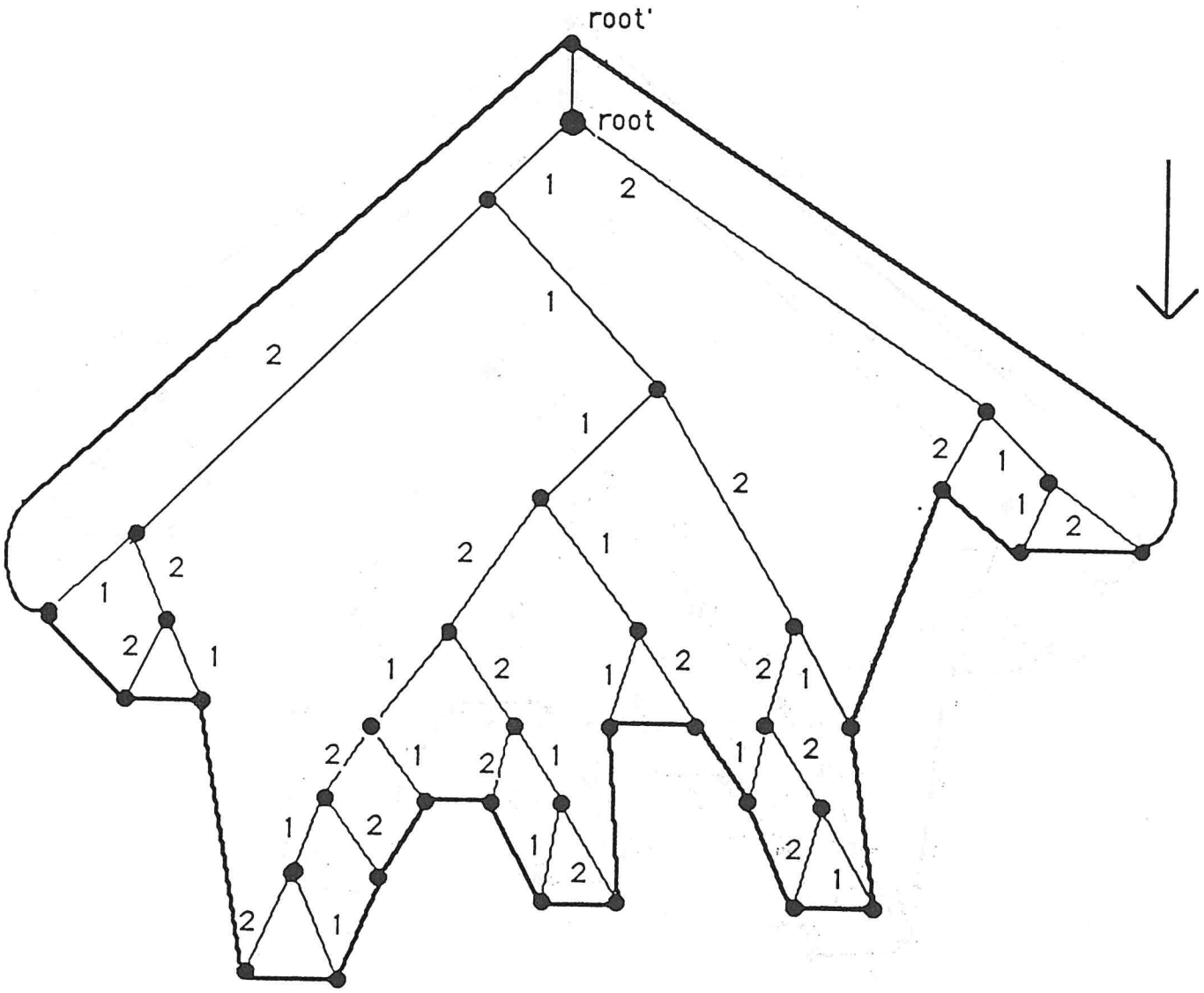


Fig.1

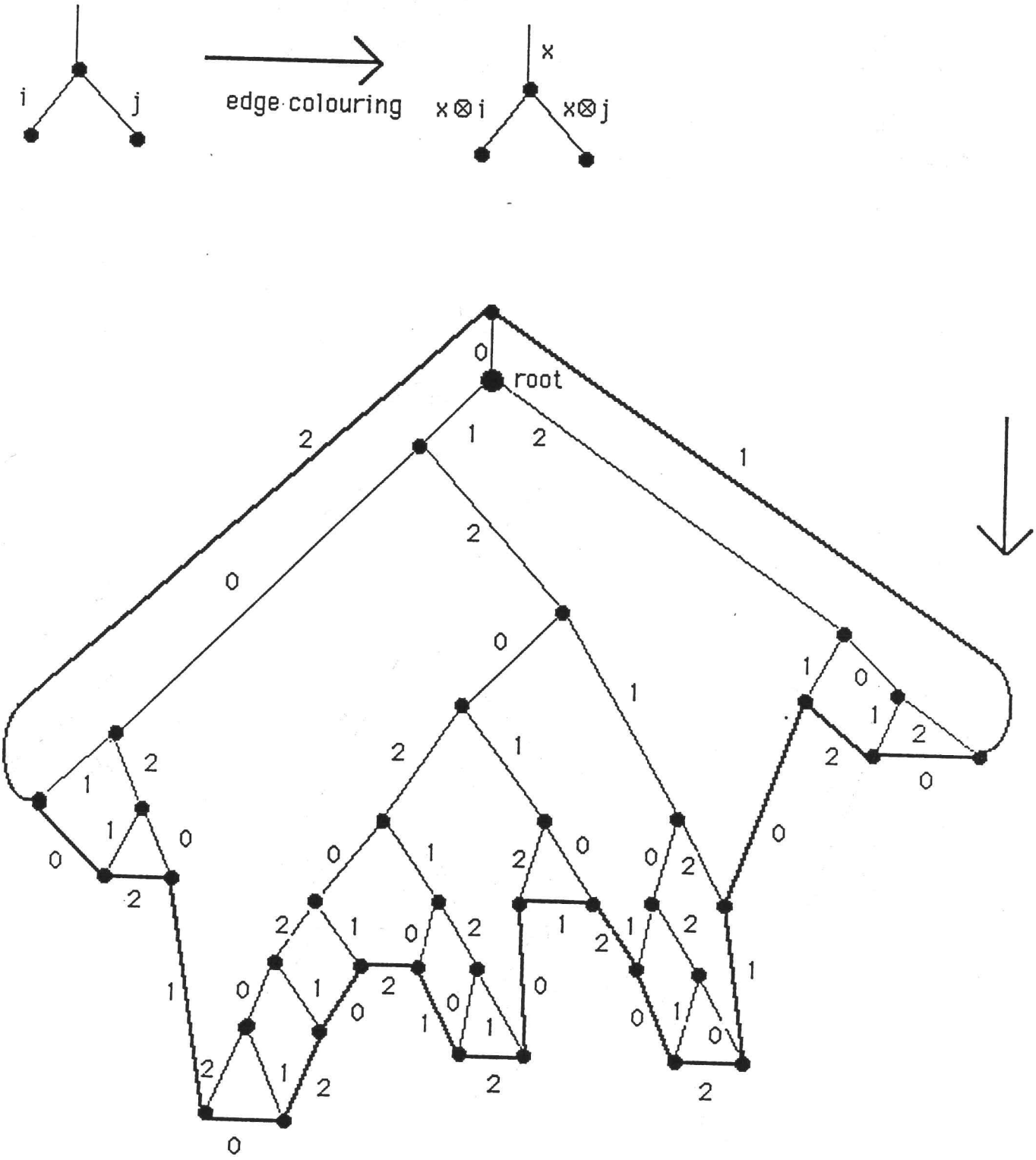


Fig.2

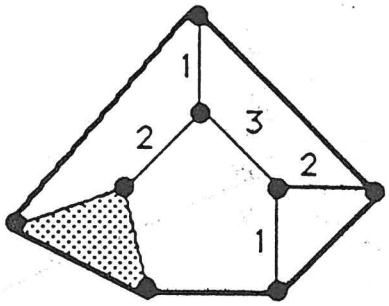


Fig. 3. The edge-colouring of the tree (except shaded triangle) cannot be extended to the skirt.

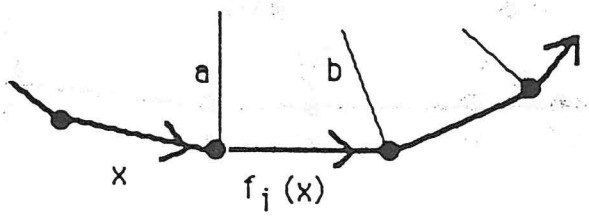


Fig. 4. $f_i(x)$ is a colour associated with the i-th skirt edge such that the colouring a,x, $f_i(x)$,b is locally legal

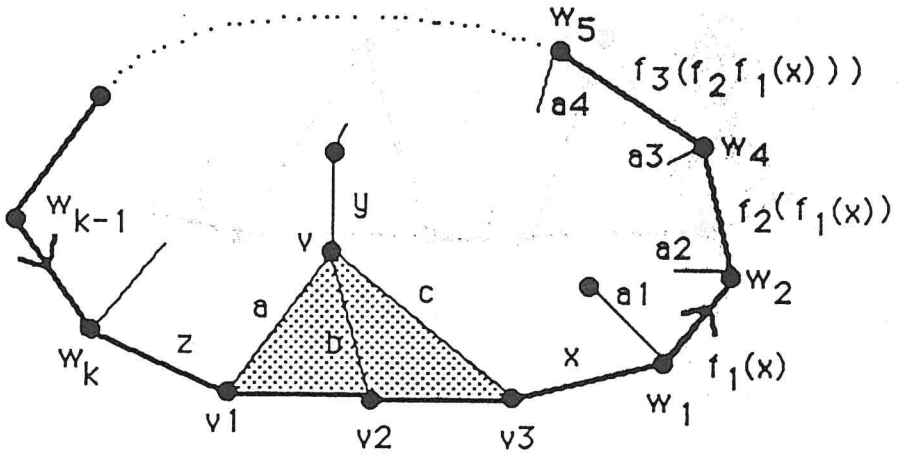


Fig.5. For every three colours x,y,z such that $x \neq y$ there are colours a, b and c giving a legal colouring in case $\Delta=4$. The colour z is determined by x and functions f.

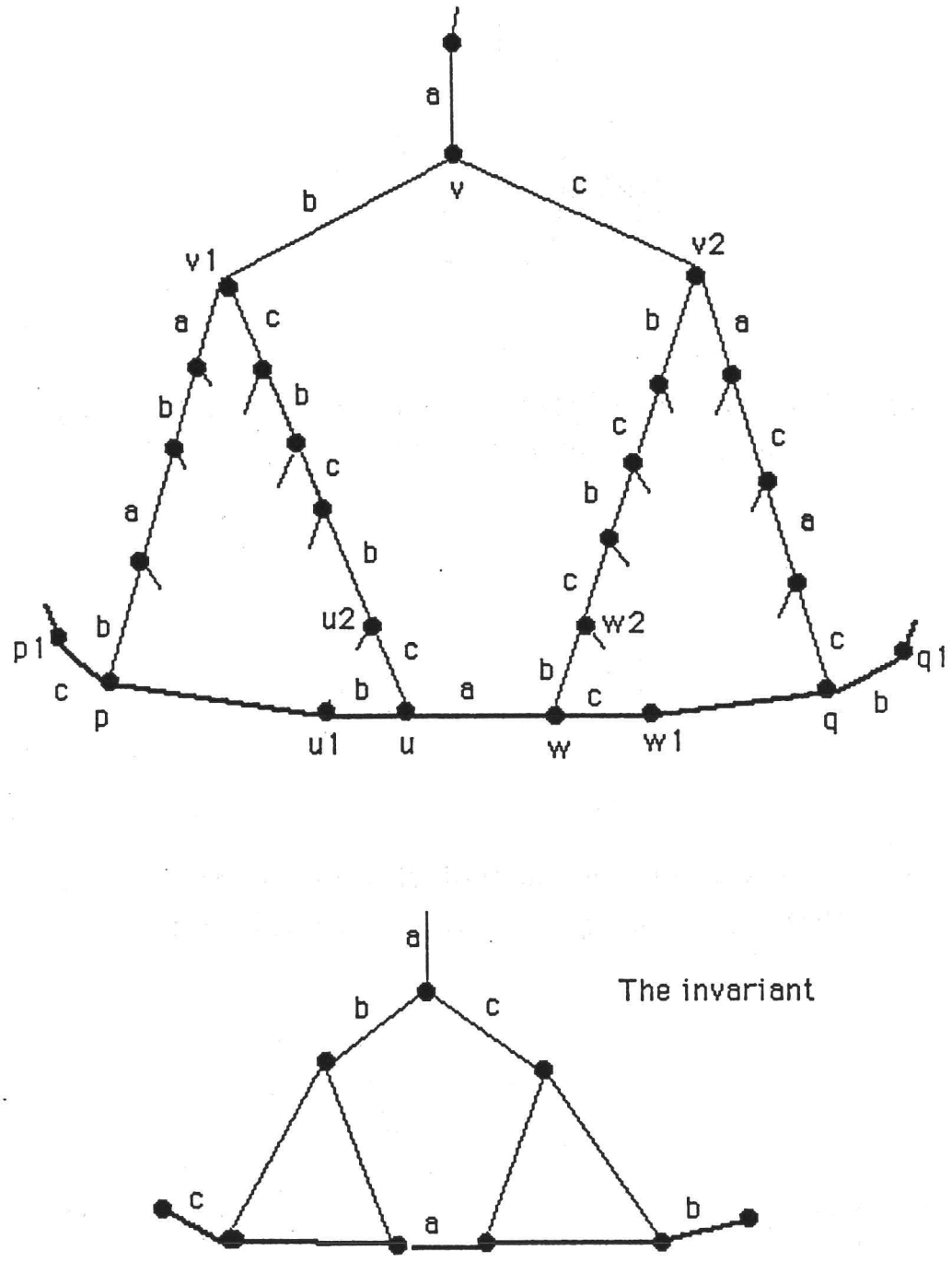


Fig.6.The invariant: the colour of (u,w) is the same as that of the edge entering the vertex v = LCA(u,w). There is no colour a on the branches from v to u and w. The colour of (p;p1) is c and the colour of (q,q1) is b.

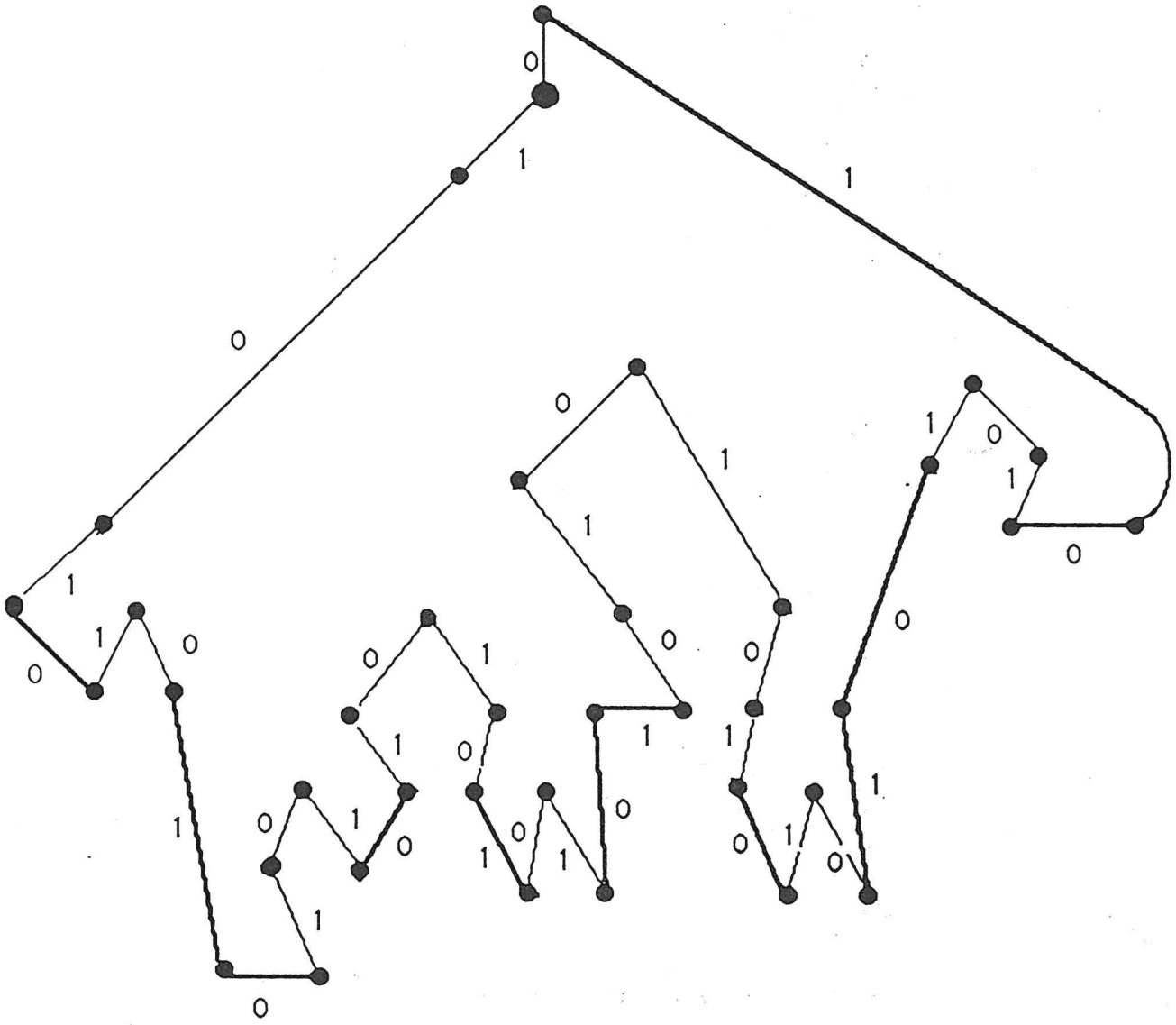


Fig.7. The edges coloured by 0 or 1 give
a Hamiltonian cycle.