AN EFFICIENT ALGORITHM FOR COLOURING THE EDGES OF A GRAPH WITH $\Delta + 1$ COLOURS*

ESHRAT ARJOMANDI

Department of Computer Science, York University, Downsview, Ontario

Abstract

The edge colouring problem has received considerable attention from mathematicians and computer scientists. The edges of a simple graph G can be coloured with Δ or $\Delta + 1$ colours, where Δ is the maximum degree in G. Holyer has recently shown that Δ-edge-colourability is NP-complete. In this paper we present a $(O(\min\{|E|\cdot|V|, |V|\cdot\Delta + |E|\cdot\sqrt{|V|\cdot\log|V|}\})$ edge colouring algorithm for general graphs which uses at most $\Delta + 1$ colours.

Résumé

Le problème de colorer les arcs d'un graphe a été très étudié par les mathématiciens et les informaticiens. Les arcs d'un graphe simple G peuvent être colorés avec Δ où $\Delta + 1$ couleurs, où Δ est le degré maximum de G. Holyer a montré récemment que le problème de la colorabilité-Δ des arcs est NP-complet. Dans cet article nous présentons un algorithme pour colorer les arcs en temps $O(\min\{|E| \cdot |V|, |V| \cdot \Delta + |E| \cdot \sqrt{|V| \cdot \log|V|}\})$ pour un graphe quelconque utilisant au plus $\Delta + 1$ couleurs.

1 Introduction

An edge colouring of a graph $G = (V, E)$, where V is the vertex set and E is the edge set, is an assignment of colours to the edges of G such that no two incident edges have the same colour. The edge chromatic number $\chi'(G)$ of a graph is the minimum number of colours that can be used in colouring the edges of G. Clearly $\chi'(G) \geqslant \Delta$, where Δ is the maximum degree in G. Vizing[17] and Gupta[13] independently proved the following theorem.

Theorem 1.1
If G is simple, then $\Delta \leqslant \chi'(G) \leqslant \Delta + 1$.

A proof of the above theorem may be found in Bondy and Murty.[3]

Many scheduling problems may be formulated as an edge colouring problem. An example is the class-teacher timetable problem. Teacher T_i, $1 \leqslant i \leqslant m$, teaches class C_j, $1 \leqslant j \leqslant k$, for P_{ij} periods. We would like to schedule a timetable with minimum number of time periods. This problem can be represented by a bipartite graph G with bipartition

*Received 4 March 1980; revised 31 July 1980.

(T, C), where $T = \{T_1, ..., T_m\}$, $C = \{C_1, ..., C_k\}$ and vertices T_i and C_j are joined by P_{ij} edges. The problem of scheduling a timetable with minimum number of time periods is equivalent to colouring the edges of a bipartite multigraph G constructed as above with a minimum number of colours (for more information on applications, the reader is referred to Berge[2] and Bondy and Murty[3]).

Mathematicians have shown a great deal of interest in finding necessary and/or sufficient conditions for a graph to be Δ or $\Delta + 1$ edge colourable.[2,3,6] In general the question remains unanswered. However, the question is answered for many families of graphs.[2,3] For example, it is known that bipartite graphs, complete graphs with even number of nodes, and planar cubic graphs with edge connectivity $\geqslant 2$ are Δ-colourable, whereas the edge chromatic number of regular graphs with an odd number of nodes is $\Delta + 1$.

The problem of Δ-edge-colourability has received considerable attention from a computational complexity point of view. Many combinatorial problems for which no polynomial time algorithms are known, have proved to be either NP-complete or NP-hard.[4,9,10,16] Holyer[15] has recently proved that cubic 3-edge-colourability is NP-complete. From his result it immediately follows that general graph Δ-edge-colourability is also NP-complete. Since it is unlikely to solve the NP-complete problems in polynomial time, a common approach is to design polynomial-time heuristic algorithms that generate approximate solutions to these problems. For the node colouring problem it is shown[9] that coming close to $\chi(G)$ with a fast algorithm is hard, where $\chi(G)$ is the chromatic number of G. Namely it is shown that, if for a constant $r < 2$ and a constant d, there exists a polynomial time algorithm which guarantees to use at most $r \cdot \chi(G) + d$ colours, then there also exists a polynomial time algorithm which guarantees to use exactly $\chi(G)$ colours.

In this paper we shall show that the edges of a graph can be coloured efficiently using $\Delta + 1$ colours. Many algorithms have appeared in the literature for the minimum edge colouring of bipartite graphs.[5,7,8,11,12] The best known bound for bipartite edge colouring is

$$O(\min\{|V| \cdot |E|, |E| \cdot \Delta \cdot \log |V|, \Delta \cdot |V| + |E| \cdot \sqrt{|V| \cdot \log |V|}, |V|^2 \log \Delta\}).^{(8)}$$

A straightforward implementation of the proof of Vizing's theorem yields an $O(|E| \cdot |V|)$ algorithm for the general edge colouring problem using $\Delta + 1$ colours. In Section 2 of this paper we present an $O(\min\{|V| \cdot |E|, \Delta \cdot |V| + |E| \cdot \sqrt{|V| \cdot \log |V|}\})$ general edge colouring algorithm which uses at most $\Delta + 1$ colours.

All graphs considered in this paper are simple. Definitions not given here may be found in Harary.[14]

2 An Edge Colouring Algorithm

In this section an $O(\min\{|E|\cdot|V|, |V|\cdot\Delta + |E|\cdot\sqrt{|V|\cdot\log|V|}\})$ general edge colouring algorithm which uses at most $\Delta + 1$ colours is presented. A straightforward implementation of Vizing's theorem yields an $O(|E|\cdot|V|)$ algorithm.

We first present an $O(|E|\cdot|V|)$ algorithm based on Vizing's theorem which uses at most $\Delta + 1$ colours. Parts of this algorithm will be used to develop the $O(|V|\cdot\Delta + |E|\cdot\sqrt{|V|\cdot\log|V|})$ algorithm. Consider an uncoloured edge $e = uv_0$ in G. Since the degree of v_0 is at most Δ, there is a colour c_1 missing at v_0; colour e with c_1. If c_1 is not present at u, we have managed to colour e without introducing a new colour. Now assume there is an edge incident on u coloured c_1. We now have two edges incident on u coloured c_1. Procedure "Paint" either recolours a subset of the edges incident on u such that every edge incident on u has a distinct colour or finds a sequence of vertices $v_0, v_1, v_2, \ldots, v_k$ and a sequence of colours $c_1, c_2, \ldots, c_{k+1}$ and an integer t, $1 \leqslant t \leqslant k$, such that:

(i) v_i, $0 \leqslant i \leqslant k$, is adjacent to u,
(ii) uv_0 has colour c_1,
(iii) uv_j has colour c_j, $1 \leqslant j \leqslant k$,
(iv) colour c_{j+1} is not present at v_j, $1 \leqslant j \leqslant k$,
(v) $c_{k+1} = c_t$, $1 \leqslant t < k$.

If $t > 1$, colour c_t is missing at both v_k and v_{t-1}. If $t = 1$, colour c_t is only missing at v_k. Procedure "Augment" is then used to recolour appropriate parts of G such that every edge incident on u has a distinct colour.

Before presenting the edge colouring algorithm let us introduce some terminology. An edge $e = uv$ is of type $\alpha\beta$ if colour α is missing at vertex u and u is adjacent to two distinct vertices $w \neq v$ and $w' \neq v$ such that either β is missing only at w and the edges $e = uv$ and uw' are coloured β or β is missing at both w and w'. P is called an *$\alpha\beta$-path* if the length of P is maximal and the colours of edges on P alternate between α and β. An *Euler partition* is a partitioning of the edges of a graph G into open and closed *walks* such that every node of odd (even) degree is at the end of exactly one (zero) open walk.

We now present the procedures "Paint" and "Augment."

Procedure Paint (e)
begin
1 $t \leftarrow 0$; **comment**: t will stay zero if "paint" successfully colours e.
comment: let A_v, $v \in V$, be the list of missing colours at v. These lists are constructed before "paint" is called.
2 let $e = uv_0$; let β be the first colour in A_u;

3 let c_1 be the first colour in A_{v0}; delete c_1 from A_{v0}; colour e with c_1;
comment: to avoid searching A_v, $v \in V$, we always select the first colour in A_v.
4 if c_1 is not present at u **then begin**
delete c_1 from A_u; return;
end
5 let uv_1 be the edge coloured c_1; let $k \leftarrow 1$, $T \leftarrow$ false;
6 **while** (T = false) *do*
if β is missing at v_k
then begin
colour uv_k with β; *if* $k > 1$ *then* colour uv_i, $1 \leqslant i \leqslant k - 1$, with c_{i+1};
update the lists of missing colours A_{vi}, $1 \leqslant i \leqslant k$, as follows: (a) delete β from A_u and A_{vk}
(b) delete c_{i+1}, $1 \leqslant i \leqslant k - 1$, from A_{vi}
(c) add c_i, $1 \leqslant i \leqslant k$, to A_{vi}
comment: from now on we will not mention the details of how the lists of missing colours are updated.
return;
end
let c_{k+1} be the first colour in A_{vk};
if c_{k+1} is not present at u
then begin
recolour uv_j, $1 \leqslant j \leqslant k$, with c_{j+1};
update the lists of missing colour A_{vj}, $1 \leqslant j < k$, accordingly; return;
end
if $c_{k+1} = c_t$, $1 \leqslant t \leqslant k$, **then** $T \leftarrow$ true;
else begin
$k \leftarrow k + 1$; let uv_k be the edge coloured c_k;
end
end
comment: e is of type βc_t. We now have a sequence $v_0, v_1, \ldots, v_k$ of vertices and a sequence $c_1, c_2, \ldots, c_{k+1}$ of colours and an integer t, $1 \leqslant t \leqslant k$, such that properties ($i$) – ($v$) hold.
end paint;

The result of procedure "paint" is illustrated in figure 1. The numbers on the edges are the colours assigned to the edges.

Theorem 2.1
Procedure "Paint" uses time $O(\Delta)$.

Proof

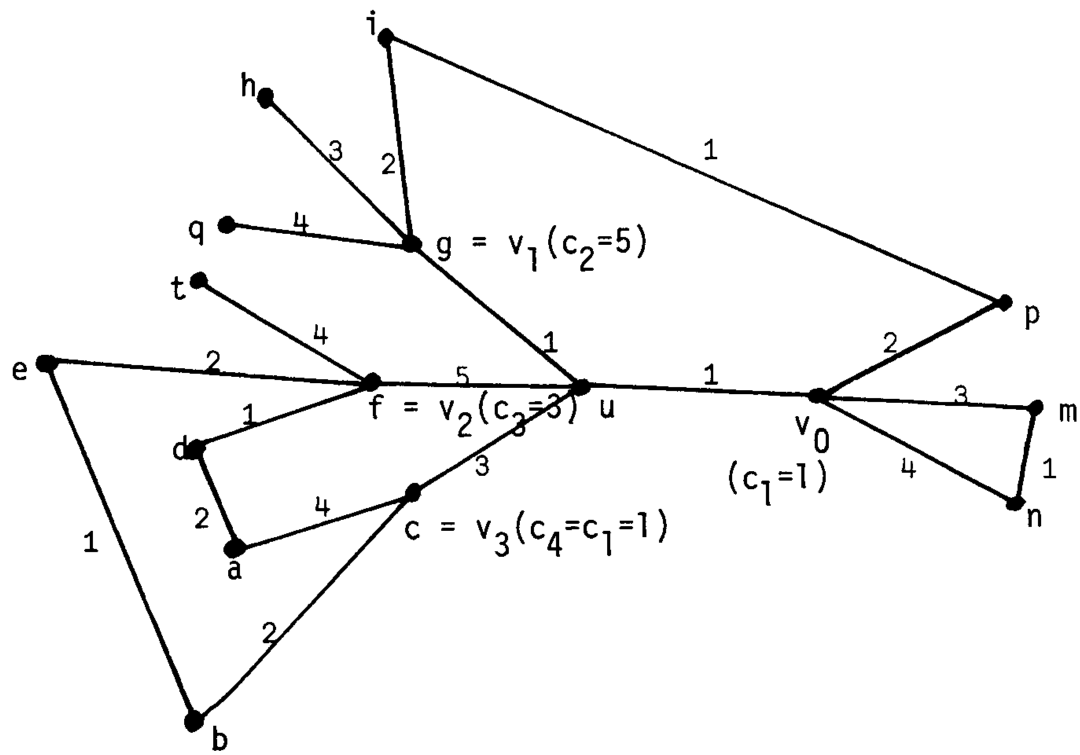


FIG. 1.

Let us first introduce some data structures necessary for the efficient implementation of "Paint." For each node $v \in V$ we maintain a list A_v of missing colours at v. To allow efficient deletions and additions, linked storage allocation is used to represent $A_v, v \in V$. Since we are using $\Delta + 1$ colours to colour G and the maximum degree in G is Δ, $A_v, v \in V$, is non-empty. Also, in order to be able to check the presence of a colour at a vertex, we maintain an n by $\Delta + 1$ vertex-colour incidence matrix N–C such that (v, α)i s the edge, if any, incident on v coloured α. Although throughout this section we have suppressed explicit reference to the maintenance of N–C, it is obvious that N–C should be updated every time an edge changes colour. The updating of N–C does not change the time bounds. In order to be able to perform the test in step 6.4 in constant time, we construct a vector I of size $\Delta + 1$ as follows:

$$I(j) = \begin{cases} v_i & \text{where } v_i \text{ is the vertex in the sequence of vertices generated} \\ & \text{by "Paint" and } uv_i \text{ is coloured } j. \\ 0 & \text{there is no node } v_i,\ 0 \leqslant i \leqslant k, \text{ in the sequence of vertices} \\ & \text{generated by "Paint" such that } uv_i \text{ is coloured } j. \end{cases}$$

By using the above data structures it is easy to see that the procedure "Paint" uses time $0(\Delta)$. (For more details of the proof of timing, the reader is referred to Arjomandi[(1)].) Q.E.D.

If procedure "Paint" does not succeed in colouring $e = uv_0$ such that every edge incident on u has a distinct colour, procedure "Augment" is then called. The procedure "Augment" uses the sequences of vertices and colours generated by "Paint" and recolours appropriate parts of G such that every edge has a distinct colour. This leads us to procedure "Augment."

Procedure Augment (e)

begin

1 let $e = uv_0$;

2 if $t > 1$

  comment: t is a global variable and its value is determined in the procedure "Paint."

  then begin

2.1 let P_k and P_{t-1} be two βc_t-paths from v_k and v_{t-1} respectively;

2.2 let $P_m = P_k$ or P_{t-1}, where $m = k$ or $t - 1$, be the path that does not end in u (if P_k and P_{t-1} both do not end in u, let $P_m = P_{t-1}$, $m = t - 1$);

  end

3 **else begin**

3.1 let P_0 and P_1 be two $c_1\beta$-paths from u;

3.2 let P_k be a βc_1-path from v_k;

  comment: two of the paths P_0, P_1, and P_k may be identical, in which case the third path will be vertex disjoint from the two identical ones and will not end in u. If P_k is identical to either P_0 or P_1, then one of P_0 or P_1 (i.e. the one that is identical to P_k) ends in v_k and the other one in a vertex different from u.

3.3 **if** P_0 is identical to P_1 **then** let $P_m = P_k$, $m = k$;

  else let $P_m = P_0$, $m = 0$;

  end

4 let P_m end in vertex w; interchange the colours β and c_t along P_m;

5 **if** $m > 1$

  then begin

5.1 recolour uv_m with β; recolour uv_j, $1 \leqslant j \leqslant m - 1$, with c_{j+1};

  end

6 update the lists of missing colours at u, w, and v_1, $1 \leqslant i \leqslant m$, accordingly;

end Augment;

Consider the graph of figure 1. In this example $t = 1$, $P_0 = P_1$, and

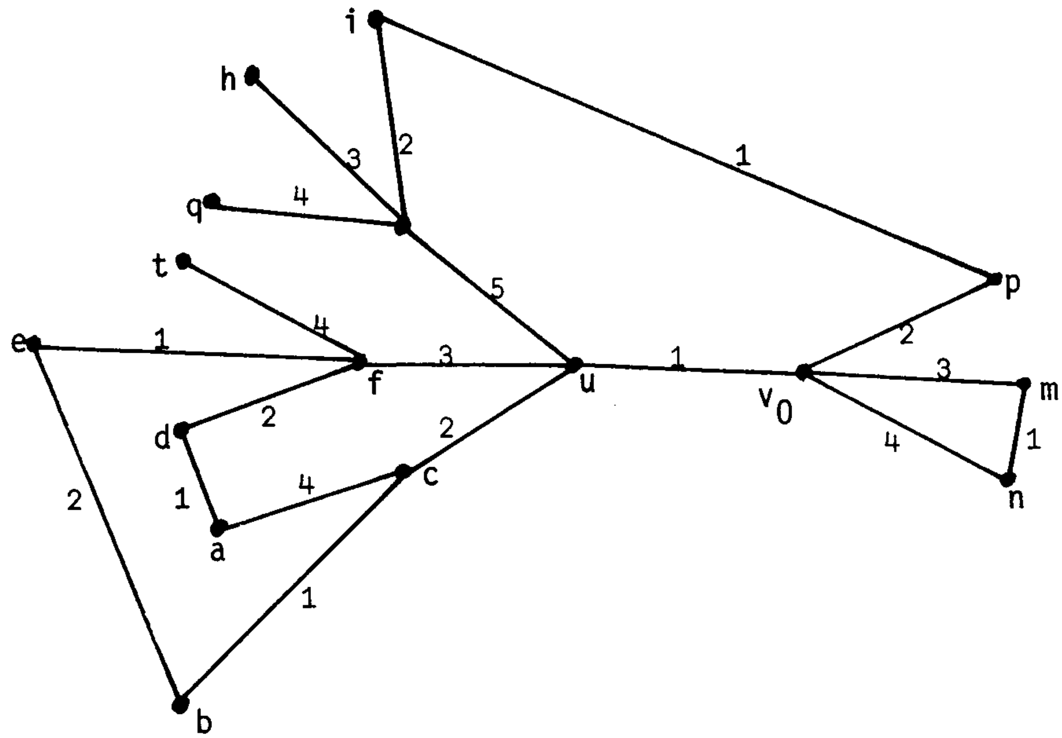


FIG. 2.

the path P_3: $(v_3 - b - e - f - d - a)$ does not end in u. Figure 2 illustrates the result of procedure "Augment" for the graph of figure 1.

Theorem 2.2
Procedure "Augment" uses $O(|V|)$ time.

Proof
Procedure "Augment" constructs at most three paths. Since each path is at most of length $O(|V|)$, steps 1–4 are $O(|V|)$. Steps 5 and 6 are $O(\Delta)$; thus the overall complexity of "Augment" is $0(|V|)$. Q.E.D.

We now consider a procedure that uses "Paint" and "Augment" and colours the edges of a graph G using at most $\Delta + 1$ colours.

Procedure edge-colour (G)
begin
1 let colours used for colouring G be represented by $\{1, 2, ..., \Delta + 1\}$.
2 Initialization: $A_v, v \in V$, the list of missing colours at vertex v is initialized to contain $\{1, 2, ..., \Delta + 1\}$.
3 **while** there is an uncoloured edge $e = uv$ **do**
    paint (e);
    if $t \neq 0$ *then* Augment (e);
  end
end Edge-Colour;

Theorem 2.3
The procedure "Edge-Colour" uses $O(|E| \cdot |V|)$ time.

Proof
Steps 1 and 2 require $O(|V| \cdot \Delta)$. The loop at step 3 is executed $|E|$ times and the body of the loop is at most $O(|V|)$, hence the complexity of "Edge-Colour" is dominated by $O(|E| \cdot |V|)$. Q.E.D.

We now present an $O(|V| \cdot \Delta + |E| \sqrt{|V|} \log |V|)$ algorithm based on a divide-and-conquer technique. This approach is similar to the approach used in Gabow and Kariv[8] for colouring the edges of a bipartite graph. The Euler partition is used to divide a graph G into two edge disjoint subgraphs G_1 and G_2. Now to obtain an edge colouring for G all we have to do is colour G_1 and G_2. This process can be repeated until a graph with maximum degree 1 is encountered.

An Euler partition may be found as follows. Select a start vertex of odd degree. If no odd degree vertex exists, then select an even non-zero degree vertex. Construct a walk from the start vertex by walking through the graph from one vertex to another and deleting the edges as they are traversed. Continue walking through the graph and deleting edges until a vertex of degree zero is reached. When a vertex of degree zero is reached, a walk of the Euler partition is completed. Now select a new start vertex and repeat the process. Gabow[7] presents an $O(|E| + |V|)$ algorithm for generating an Euler partition of a graph.

Consider an Euler partitioning of the edges of G. The following algorithm divides G into G_1 and G_2 by traversing each walk of the partition, placing edges alternately in G_1 and G_2.

Procedure Euler-divide(G);
begin
1 make P an empty queue;
2 **for** each vertex $v \in V$ **do**
2.1 place all the odd closed Euler walks that begin and end in v in P
2.2 place the odd open walk (if it exists) that begins in v in P;
comment: in the Euler partition algorithm, each time a vertex is selected as the start vertex of a new Euler walk. The start vertex is the vertex that the walk begins in it. If the walk is open, it will end in a vertex different from the start vertex. Hence if the open walk ends in vertex v it will not be included in P when the Euler walks for v are being included in P
2.3 place all the even closed paths that begin and end in v in P;
end
3 set $i \leftarrow 2, j \leftarrow 1$;
4 **while** $P \neq \Phi$ **do**

4.1 consider $p \in P$; delete p from P;
4.2 traverse p and place edges alternately in G_i and G_j;
4.3 set $k \leftarrow i$, $i \leftarrow j$, $j \leftarrow k$;
end
end Euler-divide;

It is easy to see that with proper data structures, the procedure Euler-divide can be implemented in $O(|E| + |V|)$. The following theorem determines the maximum degree in G_1 and G_2.

Theorem 2.4
The maximum degree in G_1 and G_2 is at most $\lceil \Delta/2 \rceil + 1$.

Proof
We prove the theorem in two cases:
1 Δ is even.

Consider a vertex v of degree Δ. As was mentioned earlier, in an Euler partition, even degree vertices are at the end of zero open walks. When an even closed walk going through vertex v is traversed, the edges of this path incident on v are distributed evenly between G_1 and G_2 (note that the even closed path may contain many odd cycles that loop around v). However, when an odd closed walk is traversed, if the first edge is included in $G_1[G_2]$, then the last edge will also be included in $G_1[G_2]$. Let α be the number of edges incident on v from this Euler walk. Then $G_1[G_2]$ will get $\alpha/2 + 1$ edges and $G_2[G_1]$ will get $\alpha/2 - 1$ edges. Therefore, if the number of odd closed walks going through v is odd, $G_1[G_2]$ will get $\Delta/2 + 1$ edges incident on v and $G_2[G_1]$ will get $\Delta/2 - 1$ edges. Note that, since in placing the first edge of each walk in one of the subgraphs we alternate between G_1 and G_2, no subgraph will get more than $\Delta/2 + 1$ edges.

2 Δ is odd.

Consider a node v of degree Δ. Odd degree vertices are at the end of exactly one open walk. Once again when an even closed walk that begins and ends in v is traversed, the edges of this walk incident on v are distributed evenly between G_1 and G_2. Let the number of odd closed walks beginning and ending in v be α. Now consider two cases:

(i) α is even.

In this case the edges incident on v from these walks are distributed evenly between G_1 and G_2. Hence when the open walk beginning or ending in v is traversed $G_1[G_2]$ will get at most two edges more than $G_2[G_1]$. (Note: the open walk may contain odd cycles that loop around v.)

(ii) α is odd.

In this case, if β is the number of edges incident on v from these α odd closed walks, $\beta/2 + 1$ edges are included in one of the subgraphs and

$\beta/2 - 1$ edges in the other one. Let $G_1[G_2]$ be the subgraph that gets $\beta/2 + 1$ edges. Let the number of edges incident on v from the open walk beginning or ending in v by γ (note: γ is odd). Of these γ edges, $\lceil \gamma/2 \rceil$ edges may be included in $G_1[G_2]$. Thus the maximum degree in $G_1[G_2]$ may be at most $\lceil \Delta/2 \rceil + 1$ and at least $\lfloor \Delta/2 \rfloor - 1$. Q.E.D.

We now present an edge colouring algorithm based on a divide and conquer technique using the Euler partition.

Procedure Euler-colour(G);

begin

Level# ← Level# + 1;

comment: Level# is initialized to -1 before "Euler-colour" is called. The role of Level# will be explained later. Delta is the maximum degree in the graph passed to "Euler-colour." When "Euler-colour" is called for the first time Delta = Δ.

if Delta = 1

then do

1 colour all the edges in G using a new colour c;

2 **for** each $v \in V$ **do**

if $\deg(v) = 1$

then $A_v^G \leftarrow$ empty; **comment**: A_v^G is the list of missing colours at v in G;

else $A_v^G \leftarrow c$;

end

end

else do

3 divide G into two edge-disjoint subgraphs G_1 and G_2 using a Euler partition and the Euler-divide procedure;

4 Euler-colour (G_1); Euler-colour (G_2);

5 **for** each v in G **do**

$A_v^G \leftarrow A_v^{G_1} \cup A_v^{G_2}$;

end

comment: G_1 and G_2 have no colour in common, hence $A_v^{G_1} \cap A_v^{G_2} = \Phi$. Since linked storage allocation is used in representing the lists of missing colours, the union operation is performed in constant time.

6 let $q \leftarrow$ the number of different colours used in colouring the edges of G;

7 **while** $q >$ Delta + 1 **do**

if Level# $\leqslant \log \lceil (\Delta/\sqrt{|V|/\log |V|}) \rceil$

then Recolour-One(G); **else** Recolour-two(G);

$q \leftarrow q - 1$;

end

comment: the significance of the test

Level# $\leqslant \left\lceil \log \left(\Delta \Big/ \sqrt{\frac{|V|}{\log |V|}} \right) \right\rceil$ will become evident after we have discussed procedures "Recolour-one" and "Recolour-two."

end

end Euler-colour;

Before discussing the procedures "Recolour-one" and "Recolour-two" we study a computation of the "Euler-colour" procedure. Consider the partition tree T of a graph G. The vertices of T are subgraphs of G passed to Euler-colour and G is the root of T. The leaves are subgraphs with maximum degree 1. Every internal vertex is a subgraph with maximum degree at least 2 and has two children in T. In theorem 2.4 we showed that the maximum degree in G_1 and G_2 is $\lceil \Delta/2 \rceil + 1$. Consider a subgraph G_i at level i(*) of the partition tree T. The following inequalities hold.

$$\left\lfloor \frac{|E|}{2^i} \right\rfloor \leqslant |E_i| \leqslant \left\lceil \frac{|E|}{2^i} \right\rceil \tag{1}$$

$$\lceil \Delta/2^i \rceil - 2 < \Delta_i < \lceil \Delta/2^i \rceil + 2. \tag{2}$$

Level i of the partition tree has at most 2^i vertices and T has at most $\lceil \log \Delta \rceil + 2$ levels. The variable Level# in procedure "Euler-colour" is the distance of a subgraph G_i from the root in the partition tree of a Graph G.

Let G_i be a subgraph of maximum degree Δ_i at level i. Its two children G_{i1} and G_{i2} at level $i + 1$ may have maximum degree $\Delta_{i+1} = \lceil \Delta_i/2 \rceil + 1$. The edge colouring algorithm will colour the edges of G_{i1} and G_{i2} using at most $\Delta_i + 5$ colours. Procedures "Recolour-one" and "Recolour-two" are used to eliminate the redundant colours. To motivate the necessity for "Recolouring-two", let us assume that at step 7 of "Euler-colour" we keep calling "Recolour-one" to eliminate the redundant colours. Namely the body of the loop at step 7 is replaced by (Recolour-one (G); $q \leftarrow q - 1$).

Procedure Recolour-One(G);

1 Let the colour with the fewest edges have edge set M;

2 uncolour all the edges in M;

3 **for** each edge $e \in M$ **do**

  Paint (e);

  if $t \neq 0$;

  then Augment (e);

*A vertex is at level i if it is at distance i from the root. The root is at level O.

end;
end Recolour-one;

Theorem 2.5
Procedure "Recolour-one" uses $O(|E|/\Delta \cdot |V|)$ time.

Proof
It is easy to see that the cardinality of the set M is at most $|E|/q$, where q is the number of different colours used in colouring G. As was mentioned before, $\Delta + 1 \leqslant q \leqslant \Delta + 5$. Procedures "Paint" and "Augment" require $O(|V|)$ time, hence procedure "Recolour-one "is $O(|E|/\Delta \cdot |V|)$.

Q.E.D.

Theorem 2.6
Procedure "Euler-colour," with "Recolour-one" alone, gives an $O(|E| \cdot |V|)$ algorithm.

Proof
Consider the partition tree T of G. Procedure Euler-colour spends

$$2^i \cdot O\left(\frac{|E|/2^i}{\Delta/2^i} \cdot |V|\right)$$

to recolour the subgraphs at level i of T. Hence the overall complexity of Euler-colour, only using procedure "Recolour-one," is

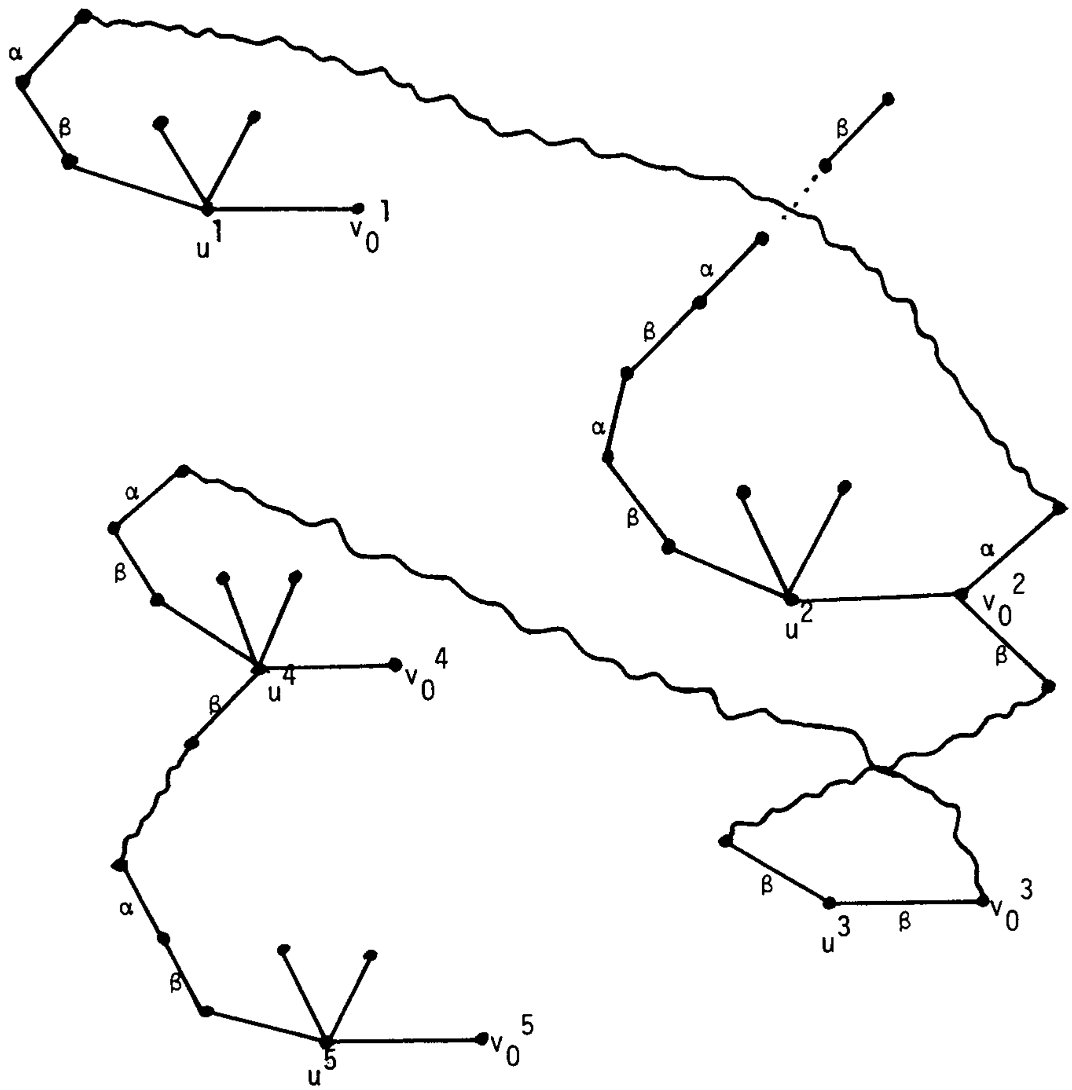
$$\sum_{i=0}^{\lceil \log\Delta \rceil + 2} 2^i \left(\frac{|E|}{\Delta} \cdot |V|\right) = O(\Delta) \cdot O(|V| \cdot |E|/\Delta)$$
$$= O(|E| \cdot |V|)$$

Q.E.D.

One reason for inefficiency in "Recolour-one" is that the edges in M are coloured one at a time and since two $\alpha\beta$-paths may overlap, an edge may unnecessarily change colour many times. To eliminate this inefficiency we like to be able to colour all the $\alpha\beta$-type edges together. Let us assume that we are given a set of $\alpha\beta$-type edges (in procedure "Recolour-two" we shall see how a set of $\alpha\beta$-type edges is determined). In order to be able to colour the $\alpha\beta$-type edges together, we construct a subgraph $G_{\alpha\beta}$ as follows:*

(1) all the edges $e = uv_0$ of type $\alpha\beta$ belong to $G_{\alpha\beta}$;
(2) for each $e = uv_0$ of type $\alpha\beta$, procedure "Paint" finds a sequence of vertices $v_0, v_1, \ldots, v_k$ and a sequence of colours $c_1, c_2, \ldots, c_{k+1} = c_t = \beta$, and an integer t, $1 \leqslant t < k$. If $t > 1$, let P_m be an $\alpha\beta$-path starting at v_m, $m = k$ or $t - 1$, that does not end in u. If both

*Gabow [8] uses a similar technique for colouring all the $\alpha\beta$-type edges together.



A typical $G_{\alpha\beta}$

FIG. 3.

paths from v_k and v_{t-1} do not end in u, let P_m be the one starting at v_{t-1}, $m = t - 1$. If $t = 1$, let P_0 and P_1 be two $\beta\alpha$ paths from u. Let P_k be an $\alpha\beta$-path from v_k. If P_0 is identical with P_1, let $P_m = P_k$, $m = k$, otherwise let $P_m = P_0$, $m = 0$. The edges on P_m belong to $G_{\alpha\beta}$. (Note: P_m does not end in u.)

(3) if $m > 1$, then the edges uv_j, $1 \leqslant j \leqslant m$, also belong to $G_{\alpha\beta}$;

(4) if P_m ends in v_j', where v_j' is a node in the sequence of vertices $v_0', v_1', \ldots, v_j', \ldots, v_k'$ generated for an $\alpha\beta$-type edge $e' = u'v_0'$, then the edges $u'v_i'$, $1 \leqslant i \leqslant j$, belong to $G_{\alpha\beta}$.

A typical $G_{\alpha\beta}$ is shown in figure 3. Let us first evaluate the cost of constructing $G_{\alpha\beta}$. Let $E_\beta(E_\alpha)$ be the number of edges coloured $\beta(\alpha)$ on all the $\alpha\beta$-paths considered during the construction of $G_{\alpha\beta}$. Let $E_{\alpha\beta}$ be the number of $\alpha\beta$-type edges. If an $\alpha\beta$-path is of odd length, then the number of edges coloured α on this path is one more than the edges coloured β. Therefore $E_\alpha \leqslant E_\beta + E_{\alpha\beta}$. For each edge $e = uv_0$ of type $\alpha\beta$, $G_{\alpha\beta}$ may contain at most Δ edges incident on u. Hence the cost of constructing $G_{\alpha\beta}$ is $O(\Delta \cdot E_{\alpha\beta} + E_\beta)$. Note that procedure "Recolour-two" is called for subgraphs at higher levels in the computation tree. These subgraphs have smaller maximum degrees, hence there are more missing colours at each node and consequently for a particular pair of colours (α, β), it is likely to have more $\alpha\beta$-type edges.

We now present the procedure "Recolour-two."

```
Procedure Recolour-two(G);
begin
1 let the colour with the fewest edges have edge set M;
2 uncolour all the edges in M; p ← |M|;
3 while p ≠ 0 do
4         for each colour α do
               for each edge e in M do
                   if α is missing at one end vertex of e
                   then begin
5.1                          let e = uv₀, where α is missing at u;
5.2                          Paint (e);
5.3                          if t = 0
                             then begin
                                       comment: "Paint" has succeeded
                                    to colour e. p ← p − 1; M ← M − {e},
                                    end
                         end
               end
6              for each colour β ≠ α do
                   if there is an edge of type αβ
                   then begin
6.1                          construct a G_αβ subgraph; colour-all (G_αβ);
                         end
               end
          end
    end
end Recolour-two;
```

Before discussing the complexity of "Recolour-two," let us introduce the procedure "colour-all." Procedure "colour-all" attempts to colour as many $\alpha\beta$-type edges in $G_{\alpha\beta}$ as possible.

Procedure colour-all($G_{\alpha\beta}$);

begin

1 let N be the set of $\alpha\beta$-type edges in $G_{\alpha\beta}$;

2 **for** each $e = uv_0$ in N **do**

2.1 consider the $\alpha\beta$-path P_m for e generated during the construction of $G_{\alpha\beta}$;

2.2 let v be the end vertex of P_m and wv the last edge on P_m;

2.3 **if** $(v = u')$, where $e' = u'v_0'$ is a deleted edge from N **then do**

comment: an edge $e' = u'v_0'$ is deleted from N for two reasons:

(a) an $\alpha\beta$-path ended in u' and e' changed type, hence e' is still incident on two edges coloured the same;

(b) a subset of edges incident on u' are re-coloured and every edge incident on u' has a distinct colour. In this case colours α and β are both present at u'.

Note that exactly one $\alpha\beta$-path may end in vertex u and change the type of the edge $e' = u'v_0'$ (see step 2.9 below). Hence if P_m ends in u and the edge $e' = u'v_0'$ is deleted from N, it must be that case (b) above is true. Therefore we can not interchange colours along P_m and e should not be considered in this call of "colour-all."

$N \leftarrow N - \{e\}$; go to 2.11;

end

2.4 interchange colours along P_m;

2.5 **if** $m > 1$ **then** recolour uv_i, $1 \leqslant i \leqslant m - 1$, with $c_i + 1$;

2.6 update the lists of missing colours at u, v, and v_i, $1 \leqslant i \leqslant m$, accordingly;

2.7 $p \leftarrow p - 1$; $N \leftarrow N - \{e\}$; $M \leftarrow M - \{e\}$;

comment: M is the set of all uncoloured edges in G. M is constructed in the procedure "recolour-two" before "colour-all" is called. p is the cardinality of the set M.

2.8 **if** $(v = u')$ and $(w = v_0'$ or $v_1')$, where $e' = u'v_0'$ is an edge in N

then begin

comment: v_0' and v_1' are the first two nodes on the sequence of vertices $v_0', v_1', \ldots, v_k'$, generated for $e' = u'v_0'$. Note that since e' is of type $\alpha\beta$, wv must have had colour β before we interchanged colours along P_m (recall properties (i)–(v) listed earlier in this section). After colours α and β are interchanged along P_m in step 2.4, every edge incident on u' will have a distinct colour. v and w are defined in step 2.2 above.

$N \leftarrow N - \{e'\}$; $M \leftarrow M - \{e'\}$;

$p \leftarrow p - 1$; go to 2.11;

end

2.9 **if** $(v = u')$ and $(w \neq v_0'$ or $v_1')$, where $e' = u'v_0'$ is an edge
then begin

comment: An $\alpha\beta$ path ends in u', hence after colours are interchanged along this path, α will no longer be missing at u. Thus e' changes type and the edges $u'v_0'$ and $u'v_1'$ are still coloured the same. Therefore e' can not be considered in this call of "colour-all."

$N \leftarrow N - \{e'\}$; go to 2.11;

end

2.10 **if** $(v = v_j')$ and (v_j' is adjacent to node u', where $e' = u'v_0'$ is an edge in N) and (v_j' is a node in the sequence of nodes$v_0', v_1', \ldots, v_k'$, generated for e').

then begin

recolour $u\ v_j$ with α;

comment: procedure "Paint" guarantees that α is present at all the vertices in the sequence of $v_0', v_1', \ldots, v_k'$, generated for e'. Therefore when P_m ends in v_j', it must end in an edge coloured α, hence after colours are interchanged along P_m, α will now be missing at v_j'.

recolour $u'v_i'$, $1 \leqslant i \leqslant j-1$, with c_{i+1}; $p \leftarrow p - 1$; $M \leftarrow M - \{e'\}$; $N \leftarrow N - \{e\}$; update the lists of missing colours at u' and v_1', $1 \leqslant i \leqslant j$, accordingly;

end

2.11 **end**

end colour-all;

Theorem 2.7

Procedure "colour-all" uses $O(E_\beta + \Delta \cdot E_{\alpha\beta})$ time.

Proof

To help comprehend the algorithm, we make the following remarks. Consider an edge $e = uv_0$ in N and the $\alpha\beta$-path associated with it in $G_{\alpha\beta}$. Note that the edges uv_0 and uv_1 are coloured the same. Hence if the colour of uv_0 and uv_1 is β, then two $\alpha\beta$-paths may end in u. Let the two $\alpha\beta$-paths that end in u be the paths associated with two $\alpha\beta$-type edges e_1 and e_2. Assume $e_1(e_2)$ is selected before e at step 2 of "colour-all." When the colours along the $\alpha\beta$-path associated with $e_1(e_2)$ are interchanged, one of the β-coloured edges incident on u changes colour to α. Thus every edge incident on u now has a distinct colour, hence e is removed from N. When the path associated with $e_2(e_1)$ is considered, we can not interchange colours along this path, since by doing so u will be incident with two edges coloured the same.

Now assume the colour of uv_0 and uv_1 is not β, then only one $\alpha\beta$-path may end in u. If that path is considered before the path associated with e, then e changes type and can not be considered in this call of "colour-all."

For the proof of timing, note that throughout the procedure "colour-all" every edge of $G_{\alpha\beta}$ is considered exactly once; hence the complexity of "colour-all" is dominated by $O(\Delta \cdot E_{\alpha\beta} + E_\beta)$. Q.E.D.

We now prove the complexity of the procedure "Recolour-two."

Theorem 2.8

Procedure "Recolour-two" uses time $O(|E| \cdot \Delta \cdot \log |V|)$.

Proof

To prove the time bound we first mention the data structures necessary for the efficient implementation of "Recolour-two." The algorithm maintains a list of missing colours for each vertex. We also maintain a vertex-colour incidence matric N–C. This matrix is described in the proof of theorem 2.1. For each pair of distinct colours α and β, where α is fixed, a list of $\alpha\beta$-type edges is constructed. These lists may be constructed by adding a statement after step 5.3 and are also used in step 6.1 for the construction of $G_{\alpha\beta}$. The construction of these lists does not change the time bound.

We now prove that the loops at steps 5 and 6 are $O(|E|)$ (it is easy to see that steps 1–2 are $O(|M|)$. The body of the loop at step 5 is $O(\Delta)$ and $|M|$ is at most $O(|E|/\Delta)$, hence the loop at step 5 is $O(|E|)$. The complexity of the loop at step 6 is

$$\sum_\beta O(E_\beta + \Delta \cdot E_{\alpha\beta}).$$

Procedure "colour-all" is called for each colour $\beta \neq \alpha$ and the edges coloured β have not changed colour since the beginning of the loop at step 6. Thus $\sum_\beta E_{\alpha\beta} \leqslant |M|$. Hence:

$$\sum_\beta O(E_\beta + \Delta \cdot E_{\alpha\beta}) \leqslant O(|E| + \Delta \cdot |M|) \leqslant O(|E|).$$

Therefore the total time for the loops at steps 5 and 6 is $O(|E|)$. The loop at step 4 is executed $O(\Delta)$ times, therefore the body of the loop at step 3 is $O(\Delta \cdot |E|)$.

All that remains to be proved is that the loop at step 3 is executed $O(\log|V|)$ times. To prove this we show that each execution of the loop at step 3 eliminates half of the edges in M.

Assume during an execution of the loop at step 3, edge $e = uv_0$ remains in M. Edge e may remain in M for the following reasons: (1) e was assigned a type $\alpha\beta$ and an $\alpha\beta$-path ended in u; (2) colour γ was missing at one end of e and a $\gamma\beta$-path ended in u after the execution of the loop at step 4 for β and before its execution for γ. In either case we can associate with each edge which remains in M, an edge that is deleted from M.

Q.E.D.

Theorem 3.7
Procedure "Euler-colour" runs in time $0(\Delta \cdot |V| + |E|\sqrt{|V| \log |V|})$.

Proof
Consider the partition tree T of G. Let $h = \lceil \log \Delta \rceil + 2$. The complexity of steps 1–3 and 5–6 is determined by:

$$\sum_{i=0}^{h} 2^i O(|E_i| + |V|) = O(|E| \cdot \log \Delta + \Delta \cdot |V|).$$

As was mentioned earlier, the edges of a subgraph G_i at level i with maximum degree Δ_i may be coloured with as many as $\Delta_i + 5$ colours. Therefore for any subgraph in the partition tree, the loop at step 7 is executed at most 4 times. Now let us ascertain the complexity of the body of the loop at step 7.

Consider a subgraph G_i with $|E_i|$ edges and maximum degree Δ_i at level i. $|E_i|$ and Δ_i satisfy relations (1) and (2) presented earlier. If

$$i \leqslant \left\lceil \log \left(\Delta \Big/ \sqrt{\frac{|V|}{\log |V|}}\right) \right\rceil$$

then procedure "Recolour-one" is called. Let

$$k = \left\lceil \log \left(\Delta \Big/ \sqrt{\frac{|V|}{\log |V|}}\right) \right\rceil .$$

The overall complexity of "Euler-colour" for levels i, $i \leqslant k$ is

$$\sum_{i=0}^{k} 2^i \cdot 0\left(\frac{|E_i| \cdot |V|}{\Delta_i}\right) = 0(|E|\sqrt{|V| \log |V|}).$$

If

$$i > \left\lceil \log \left(\Delta \Big/ \sqrt{\frac{|V|}{\log |V|}} \right) \right\rceil$$

then procedure "Recolour-two" is called. The overall complexity for levels i, $k < i \leqslant h$ is

$$\sum_{i=k+1}^{h} \mathrm{O}(|E_i|\Delta_i \log |V|) = |E| \log |V| \sum_{i=k+1}^{h} \mathrm{O}\left(\frac{\Delta}{2^{2i}}\right).$$

From

$$i > \log \left\lceil \left(\Delta \Big/ \sqrt{\frac{|V|}{\log |V|}} \right) \right\rceil$$

immediately follows that

$$\left\lfloor \frac{\Delta}{2^i} \right\rfloor < \left\lceil \sqrt{\frac{|V|}{\log |V|}} \right\rceil .$$

By using this inequality

$$|E| \log |V| \sum_{i=k+1}^{h} \mathrm{O}\left(\frac{\Delta}{2^{2i}}\right) < |E| \log |V| \sqrt{\frac{|V|}{\log |V|}} \sum_{i=k+1}^{h} \mathrm{O}\left(\frac{0}{2^{i}}\right)$$

$$= 0(|E|\sqrt{|V| \log |V|}).$$

Thus the overall complexity of the algorithm is

$$\mathrm{O}(\Delta \cdot |V| + |E|\sqrt{|V| \cdot \log |V|}).$$

4 Concluding Remarks

Holyer[15] has recently shown that Δ-edge-colourability is NP-complete. Hence it is unlikely to design a polynomial time algorithm which guarantees to use optimal number of colours. In this paper we presented a general $\mathrm{O}(|V| \cdot \Delta + |E|\sqrt{|V| \log |V|})$ edge colouring algorithm which uses at most $\Delta + 1$ colours.

References

(1) E. Arjomandi, "An efficient algorithm for colouring the edges of a graph with $\Delta + 1$ Colours," Dept. of Computer Science, York University, Tech. Rep. 1, 1980.
(2) C. Berge, *Graphs and hypergraphs*. Amsterdam: North Holland Press, 1976.
(3) J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*. Macmillan, 1976.
(4) S.A. Cook, The complexity of theorem-proving procedures," Proc. Third Ann. ACM Symp. on the Theory of Computing, 1970, 151–158.
(5) D. de Werra, "On some combinatorial problems arising in Scheduling," INFOR, vol. 8, 1970, 165–175.
(6) P. Erdös and R.J. Wilson, "On the chromatic index of almost all graphs," J. Comb. Theory, Series B, 1977, no. 23, 255–257.
(7) H.N. Gabow, "Using Euler partitions to edge colour bipartite multigraphs," Int. J. Comput. Infor. Sci., vol. 5, no. 4, 1976, 345–355.

(8) H.N. Gabow and O. Kariv, "Algorithms for edge colouring bipartite graphs," Proc. Tenth Ann. ACM Symp. Theory of Computing, 1978, 184–192.
(9) M.R. Garey and D.S. Johnson, "The complexity of near-optimal graph colouring," JACM, vol. 23, no. 1, 1976, 43–49.
(10) M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-Completeness*. San Francisco, CA: Freeman, 1978.
(11) T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," J. ACM, vol. 23, no. 4, 1976, 665–679.
(12) C.C. Gotlieb, "The construction of class-teacher time-tables," Proc. IFIP Congress 62, Munich. Amsterdam: North-Holland, 1963, 73–77.
(13) R.P. Gupta, "The chromatic index and the degree of a graph," Notices Am. Math. Soc., 1966, no. 13, abstract 66T-429.
(14) F. Harary, *Graph theory*. Reading, MA: Addison-Wesley, 1969.
(15) I. Holyer, "Cubic 3-edge-colourability is NP-complete." Private communication.
(16) R.M. Karp, "On the computational complexity of combinatorial problems," Networks, vol. 5, 1975, 45–68.
(17) V.G. Vizing, "On an estimate of the chromatic class of a p-graph," (Russian) Diskret. Analiz., 1964, no. 3, 25–30.