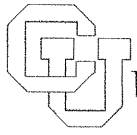
Algorithms for Edge Coloring Bipartite Graphs

Harold N. Gabow
Oded Kariv

CU-CS-123-78



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

ALGORITHMS FOR EDGE COLORING BIPARTITE GRAPHS

by

Harold N. Gabow*

and

Oded Kariv**

#CU-CS-123-78 January, 1978

Abstract

A (minimum) edge coloring of a bipartite graph is a partition of the edges into Δ matchings, where Δ is the maximum degree in the graph. Coloring algorithms are presented that use time $O(\min(|E|\Delta \log n, |E|\sqrt{n \log n}, n^2 \log \Delta))$ and space $O(n\Delta)$. This compares favorably to the previous $O(|E|\sqrt{n \log \Delta})$ time bound. A coloring algorithm finds a maximum matching on a regular graph. These algorithms compare favorably to the $O(|E|\sqrt{n})$ matching algorithm, except when $\frac{\sqrt{n}}{\log n} < \Delta < \sqrt{n} \log n$.

---

* Harold N. Gabow, Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado 80309

** Oded Kariv, Dept. of Mathematics and Computer Science Drexel University, Philadelphia, Pennsylvania 19104

1. Introduction

We investigate the problem of finding a minimum edge coloring on a bipartite graph (or multigraph). Here an (edge) coloring is an assignment of a color to each edge of the graph so the colors occurring at any given vertex are distinct; equivalently, each color forms a matching. A minimum coloring uses the fewest number of distinct colors possible.

Many scheduling problems can be formulated in terms of edge coloring. An example is the "class-teacher timetable problem". [Go] Here the bipartite multigraph contains vertices representing classes and teachers; there is an edge joining a class and a teacher for each required meeting of the two. A coloring gives a schedule, i.e., all meetings of a given color occur during the same time slot. A minimum coloring gives a schedule with the fewest number of time slots. In practice such scheduling problems incorporate additional constraints, making them quite difficult. For example, if some teachers are available only during a restricted set of hours, the timetable problem is NP-complete [E]. Other common types of constraints that are difficult to handle are "preassignments" of meetings [Go].

In contrast, the basic problem of finding a minimum edge coloring can be solved efficiently. Let n, $|E|$, and Δ represent the number of vertices, the number of edges, and the maximum degree of any vertex, respectively. (Thus $\Delta \leq n, |E| \leq n^2$). The König-Hall theorem [B] leads to an algorithm that finds a minimum coloring by finding Δ matchings; the run time is $O(|E|\Delta\sqrt{n})$. A second approach derived from classic graph theory uses augmenting paths; the time is $O(|E|n)$ [O]. A previous algorithm based on a divide-and-conquer strategy, uses time $O(|E|\sqrt{n} \log \Delta)$ and space $O(n+|E|)$ [Ga]. This paper presents several improved coloring algorithms. The first algorithm uses augmenting paths, organized in stages to prevent duplication of effort; it uses $O(|E|\Delta \log n)$ time. The second algorithm applies the first to the divide-and conquer strategy of [Ga]; it uses $O(|E|\sqrt{n \log n})$ time. The third algorithm elaborates on the divide-and-conquer approach, and uses $O((En \log \Delta)/\Delta)$, or $O(n^2 \log \Delta)$, time.

Figure 1 gives an informal comparison of these asymptotic run times, assuming all constant coefficients are 1. It shows that collectively the algorithms are strictly better than [Ga], and each of our three algorithms is best over some interval. The third algorithm has broadest range; it also gives the best estimate in terms of the single parameter n, $O(n^2 \log n)$.

Our results apply to the problem of finding a maximum cardinality matching on a regular graph. The algorithms find a maximum matching in time $O(\min(n^2 \log \Delta, n\Delta^2 \log n))$ (In terms of just n, $O(n^2 \log n)$). Except when $\frac{\sqrt{n}}{\log n} < \Delta < \sqrt{n} \log n$, this improves the $O(n\sqrt{n}\Delta)$ bound achieved by the general $O(|E|\sqrt{n})$ matching algorithm [HK].

2. Preliminaries

This Section introduces terminology and reviews previous coloring algorithms as a basis for the algorithms in Sections 3-4.

Throughout this paper, G denotes a given bipartite graph, and Δ denotes its maximum degree. A coloring of G that uses exactly k colors is a k-coloring. Recall these basic facts from graph theory [B].

Fact 1: G has a matching that covers all vertices of degree Δ.

Fact 2: A minimum coloring of G is a Δ-coloring.

Fact 1 follows from the well-known König-Hall theorem. It justifies line 1 in the following procedure, that finds one color in a coloring.

```
procedure one-color;
  begin
1.  find a matching M that covers all vertices of maximum degree;
2.  assign a new color to the edges of M;
3.  remove the edges of M from the graph;
  end;
```

To find a Δ-coloring, it suffices to call one-color Δ times. This proves Fact 2, and gives the first coloring algorithm.

In procedure one-color, line 1 can be implemented in time $O(|E|\sqrt{n})$. The approach is to find two matchings M_1, M_2: if G has vertex sets V_1, V_2, (so each edge joins V_1 to V_2), matching M_i covers all maximum degree vertices in V_i. These matchings are combined to give M, the desired matching, using a construction of Mendelsohn and Dulmage [L]. The time is dominated by finding each M_i; this is $O(|E|\sqrt{n})$ [HK]. With this implementation of one color, a minimum edge coloring is found in time $O(|E|\sqrt{n}\,\Delta)$.

A second well-known algorithm is based on augmenting paths, resembling algorithms for network flows and matchings [L]. We begin with some definitions. Suppose a subset of the edges of G are k-colored, for some $k \leq \Delta$. A color α is missing at vertex x if no edge colored α is in-

cident to x. An uncolored edge e has type $\alpha\beta$ if color α is missing at one end of e and color β is missing at the other end. (Note an edge may have more than one type.) If e is type $\alpha\beta$, an $\alpha\beta$ (augmenting) path (from e) is a path that starts at one end of e, has edges that alternate between colors α and β, and has maximal length. (Note the first edge of an $\alpha\beta$ path may have color α or β; we only require that consecutive colors alternate). An $\alpha\beta$ path starts at a vertex of e; however it never ends at a vertex of e (If it did, G would contain an odd cycle). This justifies the following procedure to color e:

```
     procedure augment (e);
       begin
1.     let e be of type αβ;
2.     let P be an αβ path from e; comment if α = β, let P be empty;
3.     interchange colors α and β on the edges of P; comment now either
       α or β is missing at both ends of e;
4.     color edge e;
     end;
```

(Note the comment in line 3 follows since P does not end at e.)

With the appropriate data structure, path P can be found in line 2 in time proportional to its length, O(n). For example, one possibility is to maintain a vertex-color incidence matrix: for vertex v and color α, the matrix specifies the edge, if any, incident to v with color α. Thus augment works in time O(n). Starting with all edges uncolored, we can use augment to color each edge e. This gives an algorithm using time $O(|E|n)$, and space $O(n\Delta)$.

A third approach [Ga] uses the method of divide-and-conquer. Recall an euler partition is a partition of the edges of G into open and closed paths, so that each vertex of odd (even) degree is the end of exactly one (zero) open path. Any graph has an euler partition, which can be found in time $O(n+|E|)$[B,Ga]. This can be used to divide G into two subgraphs, G_1 and G_2: traverse each path of the partition, placing edges alternately in G_1 and G_2. It is easy to see G_1 and G_2 both have maximum degree either $\lfloor \frac{\Delta}{2} \rfloor$ or $\lceil \frac{\Delta}{2} \rceil$. This suggests the following coloring algorithm (Note the algorithm is incomplete; insertions will be made at labels A,B,C to complete the algorithm.)

```
procedure euler-color (G);
  begin
1.  let Δ be the maximum degree in G;
2.  if Δ = 1 then color all edges in G, using a new color
    else begin
    A:
3.  divide G into subgraphs G1, G2, each with maximum degree at most ⌈Δ/2⌉
    (use an euler partition);
4.  euler-color(G1);
    C:
5.  euler-color (G2);
    B:
  end;
```

Before discussing how to complete this algorithm, we define a useful way to represent a computation of euler-color. The partition tree T for G is a binary tree. Its nodes represent subgraphs passed to euler-color as arguments. The root node represents G. A node representing a subgraph G' with maximum degree at least 2 has 2 sons in T; they represent subgraphs G_1' and G_2' of line 3. A node representing a subgraph G' with maximum degree 1 is a leaf in T.

Now consider a node on level i of the partition tree* representing subgraph G_i, with edges E_i and maximum degree Δ_i. The following relations are easy to check:

$$(1) \quad \left\lfloor \frac{|E|}{2^i} \right\rfloor \leq |E_i| \leq \left\lceil \frac{|E|}{2^i} \right\rceil$$

$$(2) \quad \left\lfloor \frac{\Delta}{2^i} \right\rfloor \leq \Delta_i \leq \left\lceil \frac{\Delta}{2^i} \right\rceil$$

Further, there are at most 2^i nodes on level i; the edges of G are partitioned among the nodes on level i, and T has at most $\lceil \log\Delta \rceil + 1$ levels.

Now we discuss how to complete euler-color. First suppose the original graph G has maximum degree a power of 2, i.e., $\Delta = 2^k$. Then euler-color finds the desired minimum coloring. This is true because all subgraphs in the partition tree (except leaves) have Δ even. Thus in line 3 both G_1 and G_2 have maximum degree $\Delta/2$; the $\Delta/2$-colorings of G_1, G_2 give a Δ-coloring of G. Further, it is easy to see the time required is $O(|E|\log\Delta)$, i.e., $O(|E|)$ on each level of the partition tree.

*By convention, a node on level i is at distance i from the root. Thus the root is on level 0, its sons are on level 1, etc.

A difficulty arises whenever Δ is odd. In this case subgraphs G_1 and G_2 of line 3 can both have degrees as large as $\lceil \frac{\Delta}{2} \rceil$. It is not sufficient to $\lceil \frac{\Delta}{2} \rceil$-color each subgraph, since this gives a $(\Delta+1)$- coloring of G. Some action must be taken to eliminate the extra color.

The approach of [Ga] eliminates this color before dividing the graph. Thus, at label A, if Δ is odd, procedure one-color (given above) is called. It assigns a new color to a matching that covers all maximum degree vertices. Then line 3 only partitions the remaining edges of G, which form a graph with Δ even. So the algorithm finds a minimum coloring on any graph. The time is $O(|E|\sqrt{n}\log\Delta)$, since in the worst case, one-color is called at each node of the partition tree, using time $O(|E|\sqrt{n})$ on each level.

Now we present our algorithms. Section 3 gives two algorithms that eliminate the extra color after dividing and coloring the graph, (at label B). Section 4 gives an algorithm that repartitions the graph, at label C, and also eliminates the extra color at B.

3. Recoloring Methods

This section presents recoloring methods that, used in procedure euler-color of Section 2, give edge coloring algorithms. The best times for these algorithms are $O(|E|\Delta\log n)$ and $O(|E|\sqrt{n \log n})$.

A recoloring method solves the following problem: Given is (i) a bipartite graph G, with maximum degree Δ; (ii) a partition of G into subgraphs G_1, G_2, each with maximum degree at most $\lceil \frac{\Delta}{2} \rceil$; (iii) $\lceil \frac{\Delta}{2} \rceil$ - colorings of G_1, G_2. The problem is to find a Δ-coloring of G. We assume Δ is odd (else there is no problem). Thus the colorings of G_1, G_2 give a $(\Delta+1)$-coloring of G. To get a Δ- coloring, the recoloring method chooses a color, whose edges are denoted by M; these edges are then recolored with the remaining Δ colors. A recoloring method that successfully accomplishes this is called valid. If R is a valid recoloring method, we get a correct edge coloring algorithm by inserting in procedure euler-color, at label B, the statement

if Δ is odd then R;

The first method simply uses augmenting paths to recolor edges one-by-one.

```
procedure sequential-recolor;
  begin
1.   let M be the color with the fewest edges;
2.   for each edge e∈M do
3.      augment (e)
  end;
```

Lemma 1: Procedure sequential-recolor is a valid recoloring method that uses time $O(|E|n/\Delta)$ and space $O(n\Delta)$.

Proof: The method is obviously valid. The time and space bounds follow from those of augment. □

Procedure euler-color, with sequential-recolor, gives an edge coloring algorithm that runs in time $O(|E|n)$. To show this, note the time spent on all subgraphs on level i of the partition tree is (by (1)-(2))

$$2^i O\left(\left\lceil \frac{|E|/2^i}{\Delta/2^i} \right\rceil n\right) = O\left(2^i \frac{|E|n}{\Delta}\right).$$

The time spent recoloring edges in levels 0 through k is

$$(3) \quad \sum_{i=0}^{k} O\left(2^i \frac{|E|n}{\Delta}\right) = O\left(2^k \frac{|E|n}{\Delta}\right).$$

Taking $k = \log \Delta$ shows the total recoloring time is $O(|E|n)$. This dominates the run time.

The $O(|E|n)$ bound does not improve known methods. However (3) is a good bound if $k \ll \log \Delta$. We will use this fact to get better bounds.

A source of inefficiency in sequential-recolor is that two $\alpha\beta$ paths can overlap. Thus an edge can change colors many times. Now we eliminate this inefficiency, by coloring all edges of one type, $\alpha\beta$, together.

To do this, define a subgraph $H_{\alpha\beta}$ containing the edges involved in $\alpha\beta$ augments: Edge e is in $H_{\alpha\beta}$ when either

(i) e is an uncolored $\alpha\beta$ edge, or

(ii) e is on an $\alpha\beta$ path starting at an uncolored $\alpha\beta$ edge.

A vertex has degree at most 2 in $H_{\alpha\beta}$, so $H_{\alpha\beta}$ is a disjoint union of paths and cycles. This leads to the following algorithm for coloring all $\alpha\beta$ edges.

```
    procedure TR(α,β);
      begin
1.    form subgraph H_αβ;
2.    for each connected component C of H_αβ do
      begin
3.    let e_1,...,e_k be the sequence of αβ edges in C; let P_i be the αβ
        path joining e_i to e_{i+1}, 1≤i<k; let P_k be the αβ path starting
        at e_k, P_k ≠ P_{k-1};
4.    for i := 1 step 2 until k do
      begin
5.      interchange colors in P_i;
6.      color e_i and e_{i+1} (if e_{i+1} exists);
    end end end;
```

In Figure 2, component C is a cycle. In Figure 2(a), edges e_1 and e_2 are uncolored. Figure 2(b) shows the recolored cycle. In Figure 3(a), C is a path, containing edges e_1, e_2, e_3. Note f is an uncolored edge, but not type $\alpha\beta$. Figure 3(b) shows the recolored graph.

In the time estimates below, we use this notation: c_α is the number of edges colored α; $u_{\alpha\beta}$ is the number of (uncolored) edges in M of type $\alpha\beta$.

Lemma2: TR(α,β) colors all $\alpha\beta$ edges in M in time $O(c_\beta+u_{\alpha\beta})$ and space $O(n\Delta)$.

Proof: First we discuss correctness. TR is clearly correct if C is a path or if C is a cycle with an even number of $\alpha\beta$ edges. Now we show C is not a cycle with an odd number of $\alpha\beta$ edges. We do this by contradiction as follows.

Suppose k (in line 3) is odd. After lines 5-6 are executed for i = 1,3,...,k-2, the edges in the sequence P_k, e_1, P_1,..., e_{k-1}, P_{k-1} form an $\alpha\beta$ path joining the ends of the $\alpha\beta$ edge e_k. This gives an odd cycle, the desired contradiction.

For the time bound, first note the time is $O(|E_{\alpha\beta}|)$, where $E_{\alpha\beta}$ is the edge set of $H_{\alpha\beta}$. This is clear for lines 2-6. For line 1,

we shall see TR is called with a list of type $\alpha\beta$ edges available to it. Using this list and the vertex-color incidence matrix, $H_{\alpha\beta}$ is formed in $O(|E_{\alpha\beta}|)$ time.

Now note $|E_{\alpha\beta}| = O(c_\beta + u_{\alpha\beta})$. For the number of α edges in $H_{\alpha\beta}$ is at most $c_\beta + u_{\alpha\beta}$. Thus $|E_{\alpha\beta}| \leq 2\ (c_\beta + u_{\alpha\beta})$.

The desired time bound now follows. The space is dominated by the vertex-color incidence matrix. □

Note TR can be appreciably better than sequential-recolor. In Figure 3, suppose sequential-recolor colors edges e_3, e_2, e_1, in that order, always choosing an augmenting path that goes clockwise. The edges between e_2 and e_3 are traversed twice, and those between e_3 and f are traversed three times. But in TR, all edges are traversed once. Extending this example to k edges e_i increases the disparity between the algorithms.

Now we use TR to get a second recoloring method. Note it is not sufficient to apply TR to every color pair $\alpha\beta$. Again consider Figure 3. Edge f is type $\beta\gamma$ in Figure 3(a), and $\alpha\gamma$ in Figure 3(b). So if TR(α,β) is executed before TR(α,γ), edge f will not get colored.

The following algorithm handles this problem correctly.

```
   procedure typed-recolor;
        begin
1.      while there are uncolored edges in M do
            begin
2.          for each color α do
3.          for each color β=α do
4.              if there is an edge of type αβ then TR(α,β);
        end end;
```

Lemma 3: Procedure typed-recolor is a valid recoloring method that uses time $O(|E|\Delta \log n)$, and space $O(n\Delta)$.

Proof: Correctness of the algorithm is obvious. Now we prove the time bound. First note the algorithm maintains lists of $\alpha\beta$ edges, for every pair of distinct colors $\alpha\beta$. These lists are used by TR(in line 1, to form $H_{\alpha\beta}$). They are also used by typed-recolor, to do the test of line 4 in $O(1)$ time. The lists are initialized when M is chosen, in time $O(\Delta^2+n)$. They are updated by TR(in line 6), when an edge f changes type (see discussion above); the updates do not change the time estimate of Lemma 2.

Now we show an execution of the loop in lines 3-4 for a fixed α, is $O(|E|)$. The time to loop through all colors β is $O(\Delta)$. A call $TR(\alpha,\beta)$ takes time $O(c_\beta+u_{\alpha\beta})$, where these quantities are calculated immediately before the call (Lemma 2). When $TR(\alpha,\beta)$ is called, the β edges have not changed since the beginning of the loop (in lines 3-4); thus $\Sigma c_\beta \leq |E|-|M|$. When TR (α,β) is called there may be new $\alpha\beta$ edges, created in previous executions of TR (e.g., when TR (α,γ) is called, a $\beta\gamma$ edge can become $\alpha\beta$). However, it is easy to see $\sum_{\beta\neq\alpha} u_{\alpha\beta} \leq |M|$. Thus the total time in lines 3-4, for a fixed α, is $\sum_{\beta\neq\alpha} O(c_\beta+u_{\alpha\beta}) = O(|E|)$.

It follows that the loop in lines 2-4 is executed once in $O(|E|\Delta)$ time. So for the desired time bound it suffices to show lines 2-4 are executed at most log n times. To do this, we show each execution of lines 2-4 colors at least half of the edges that are uncolored when the loop begins.

Suppose an uncolored edge e is not colored by the loop. It must be that e changes type, say from $\beta\gamma$ to $\alpha\gamma$, before $TR(\beta,\gamma)$ is called but after $TR(\alpha,\gamma)$. The change occurs in $TR(\alpha,\beta)$, when an $\alpha\beta$ path P_k gets recolored. Edge e is at the end of P_k, and so becomes an $\alpha\gamma$ edge At the beginning of P_k is an edge e_k, that gets colored. Now associate edge e_k with e. Then every edge that does not get colored is associated with one that does. So at least half the edges get colored.

The space is dominated by the vertex-color incidence matrix. (The lists of $\alpha\beta$ edges only require $O(\Delta^2)$ extra space for list heads, and $\Delta^2 \leq n\Delta$). □

Now we estimate the time to color a graph using this method.

Theorem 1: Procedure euler-color with typed-recolor finds a minimum coloring in time $O(|E|\Delta \log n)$, and space $O(n\Delta)$.

Proof: The total time for recoloring graphs on level i of the partition tree is $O(\frac{|E|\Delta}{2^i} \log n)$. So the time for all levels greater k is

$$(4) \qquad \sum_{i=k+1}^{\log \Delta} O(\frac{|E|\Delta}{2^i} \log n) = O(\frac{|E|\Delta}{2^k} \log n)$$

Taking k = -1 gives the desired time bound. □

Since sequential-recolor works well for large Δ and typed-recolor works well for small Δ, we can combine them to get another efficient algorithm.

```
procedure combined-recolor;
    begin
1.      if Δ ≥ √(n / log n) then sequential-recolor else typed-recolor
    end;
```

Theorem 2: Euler-color with combined-recolor finds a minimum coloring in time $O(|E|\sqrt{n \log n})$ and space $O(n\Delta)$.

Proof: Consider a graph on level i of the partition tree. If $i \leq \log (\Delta \sqrt{\frac{\log n}{n}})$, the maximum degree is at least $\left\lfloor \frac{\Delta}{2^i} \right\rfloor = \sqrt{\frac{n}{\log n}}$; thus sequential-recolor is used. Otherwise, typed-recolor is used. The bound now follows from (3) - (4). □

4. A Repartitioning Method

This Section presents a version of euler-color that uses two partitions of the graph. The run time is $O(n^2 \log\Delta)$. This is usually the best of all algorithms presented in this paper.

The main idea is to take advantage of the fact that graphs with maximum degree a power of 2, i.e., $\Delta = 2^k$, are easy to color. As noted in Section 2, the unmodified procedure euler-color works correctly, in time $O(|E|\log\Delta)$, on such graphs.

Suppose the graph G, with maximum degree Δ, is partitioned into G_i, i = 1,2, each with maximum degree $\lceil \frac{\Delta}{2} \rceil$, and G_1 is $\lceil \frac{\Delta}{2} \rceil$- colored. Subgraph G_2 can be colored in time $O(|E|\min(\Delta \log n, \sqrt{n \log n}))$, using the algorithms of Section 3. However, we can do better, by enlarging G_2! Suppose we make G_2 a graph with maximum degree a power of 2, by transferring the correct number of colors from G_1 to G_2. The enlarged G_2 can be colored in time $O(|E|\log\Delta)$ (faster than the original G_2). This coloring, plus the coloring of the remainder of G_1, can be combined to get a coloring of G. This gives the following algorithm.

```
procedure binary-color (G);
  begin
1.    let Δ be the maximum degree in G;
2.    if Δ = 1 then color all edges in G, using a new color
      else begin
```

3. divide G into subgraphs G_1 and G_2, each with maximum degree at most $\lceil \frac{\Delta}{2} \rceil$, where G_1 has no more edges than G_2 (use an euler partition);
4. binary-color (G_1);
5. C: transfer the edges of $2^i - \lceil \frac{\Delta}{2} \rceil$ colors from G_1 to G_2, where $i = \lceil \log \frac{\Delta}{2} \rceil$;
6. binary-color (G_2);
7. B: if Δ is odd then sequential-recolor;

end end;

Labels B and C show the relationship of this algorithm to euler-color.

Figure 4 illustrates binary-color. It shows the "spine" of the partition tree for a graph G with maximum degree 13. Line 3 divides G into subgraphs G_1, G_2, with maximum degree 7. G_1 is represented by the left son of the root. In line 4, recursive calls to binary-color divide G into subgraphs of degree 4, etc. Eventually G_1 is 7-colored. Then line 5 transfers one color from G_1 to G_2. The maximum degree of G_1 becomes 6, while that of G_2 becomes 8 (a power of 2). Line 6 8-colors G_2. This gives a 14-coloring of G. Finally line 7 finds a 13-coloring.

Theorem 3: Procedure binary-color finds a minimum edge coloring in time $O(|E| n \log\Delta)/\Delta) = O(n^2 \log\Delta)$ and space $O(n\Delta)$.

Proof: First we discuss the correctness of the algorithm. We begin by noting the transfer in line 5 can always be done. For subgraph G_2 has maximum degree at least $\lfloor \frac{\Delta}{2} \rfloor$, and

$$\lfloor \frac{\Delta}{2} \rfloor = \Delta - \lceil \frac{\Delta}{2} \rceil = 2^{(\log \frac{\Delta}{2})+1} - \lceil \frac{\Delta}{2} \rceil > 2^{\lceil \log \frac{\Delta}{2} \rceil} - \lceil \frac{\Delta}{2} \rceil .$$

Thus G_2 contains a sufficient number of colors to transfer.

Now we check binary-color finds a Δ- coloring. Let Δ_i be the maximum degree of subgraph G_i, $i = 1,2$. Although the transfer in line 5 changes Δ_i, the quantity $\Delta_1 + \Delta_2$ stays the same. So if Δ is even, the coloring of G after line 6 is minimum. If Δ is odd, this coloring uses one extra color, which is eliminated in line 7. So in either case, binary-color finds a Δ- coloring.

For the time bound, we refer to the spine of the partition tree illustrated in Figure 4. (Define the spine as follows: The root represents the original graph. A node representing a G_1 subgraph is a left son, and a G_2 subgraph is a right son. All descendents (in the partition tree) of G_2 subgraphs are omitted from the spine.) Now we break the time up into two parts: that spent in the G_1 subgraphs, and that spent in the G_2 subgraphs, in the spine.

Consider a G_1 subgraph on level i. We estimate the time spent in all lines of binary-color, excluding the recursive calls. Let Δ_i and E_i represent the maximum degree and the edge set of G_1, respectively. It is easy to see lines 1-3 and 5 are $O(|E_i|)$, while line 7 is $O(|E_i|n/\Delta_i)$, by Lemma 1. Since $n>\Delta_i$, line 7 dominates. Using (1)-(2), we see line 7 is $O((|E|n/\Delta)$. Summing over all log Δ levels of the partition tree gives a total time of $O((|E|n \log\Delta)/\Delta)$.

Now consider a G_2 subgraph on level i. We estimate the time spent by binary-color to color the enlarged graph, with maximum degree a power of 2. Let Δ_i and E_i represent the maximum degree and the edge set of G_2, before it is enlarged. Then after enlarging, the maximum degree is at most $2\Delta_i$, and the number of edges is at most $2|E_i|$ (see line 3). In general, the time for binary-color to color a graph with Δ a power of 2 and edges E is $O(|E|\log\Delta)$ (Lines 5 and 7 do nothing, so the algorithm works identically to those in Section 3) So using (1)-(2) of Section 2, the time to color all G_2 subgraphs is

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

Adding the times for the G_1 and G_2 subgraphs gives the desired bound.

The space is dominated by the vertex-color incidence matrix. □

5. Application to Matching

In a regular graph, each color of a minimum edge coloring is a maximum matching. So any coloring algorithm finds a maximum matching. This gives the following result.

Lemma 4: If G is regular, a maximum matching can be found in time $O(\min(n\Delta^2 \log n, n\Delta\sqrt{n \log n}, n^2\log\Delta))$ and space $O(n\Delta)$. □

When G is regular and Δ is a power of 2, euler-color can be simplified to find a maximum matching in time linear in the number of edges, $O(n\Delta)$, [Ga]. When Δ is close to a power of 2, we can also improve the above bounds:

Lemma 5: If G is regular and $\Delta = 2^k \pm c$, where c is constant, a maximum matching can be found in time $O(n^2+|E|\log\Delta)$ and space $O(n\Delta)$.

Proof: If $\Delta=2^k+c$, proceed as follows: First form a subgraph H with $\Delta\leq 2^k$, by removing c or more edges from each vertex. Then use euler-color to 2^k color H. Finally use sequential-recolor for the edges in G-H. The time is $O(|E|\log\Delta)$ for euler-color, and $O(n^2)$ to sequential-recolor at most cn edges.

If $\Delta=2^k-c$, proceed similarly: First 2^k color G; then sequential-color the extra c colors. □

References

[B] Berge, C., Graphs and Hypergraphs, North-Holland, Amsterdam, 1973.

[E] Even, S., Itai, A., and Shamir, A., "On the complexity of timetable and multicommodity flow problems," SIAM J. Comput. 5,4, Dec. 1976, pp. 691-703.

[Ga] Gabow, H., "Using euler partitions to edge color bipartite multigraphs," International Journal of Computer and Information Sciences 5, 4 Dec. 1976, pp. 345-355.

[Go] Gotlieb, C.C. "The construction of class-teacher time-tables," Proc. IFIP Congress 62, Munich, North-Holland, Amsterdam, 1963, pp. 73-77.

[HK] Hopcroft, J.E. and Karp, R. "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," SIAM J. Comput. 2, 4, Dec. 1973, pp. 225-231.

[L] Lawler, E.L., Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[O] Ore, O., Theory of Graphs, Amer. Math. Soc. Colloq. Publ. 38, Providence, R.I., 1962.

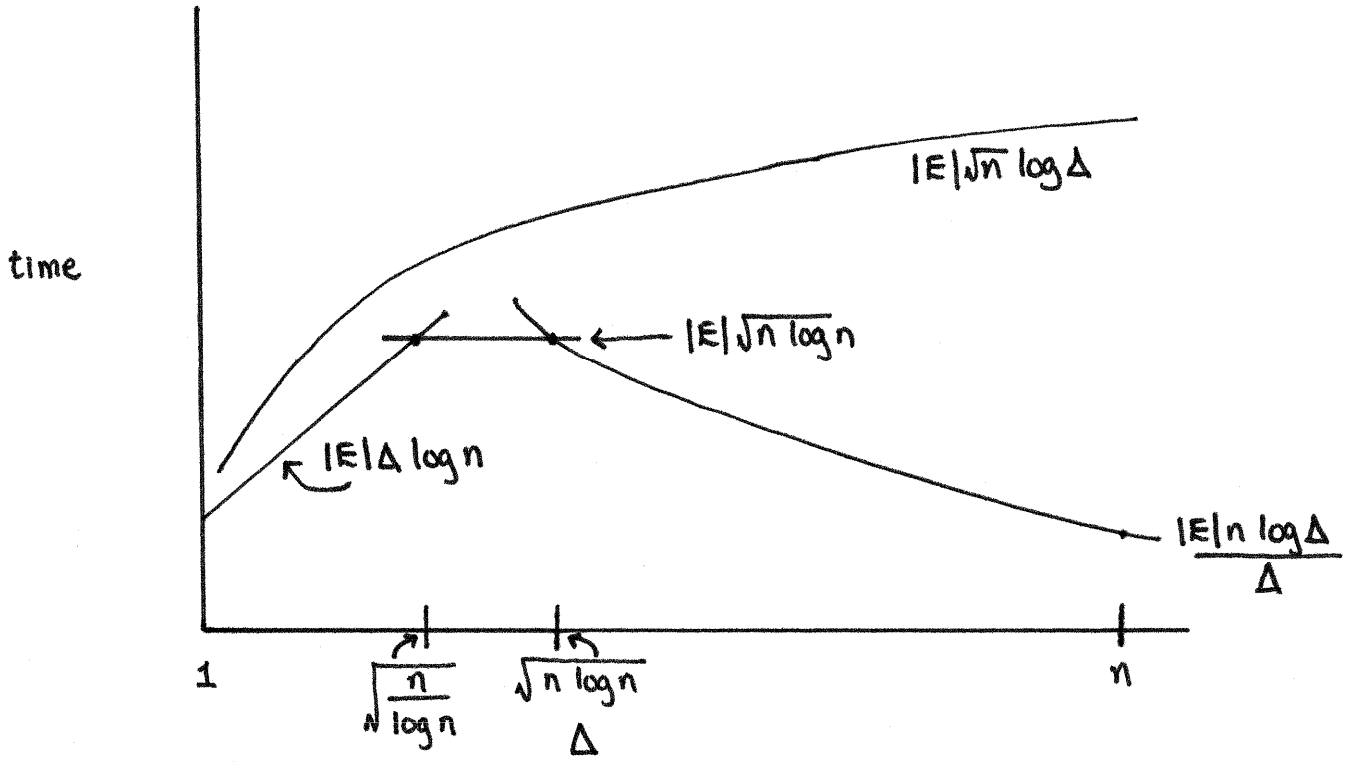


Fig. 1 Asymptotic times of coloring algorithms

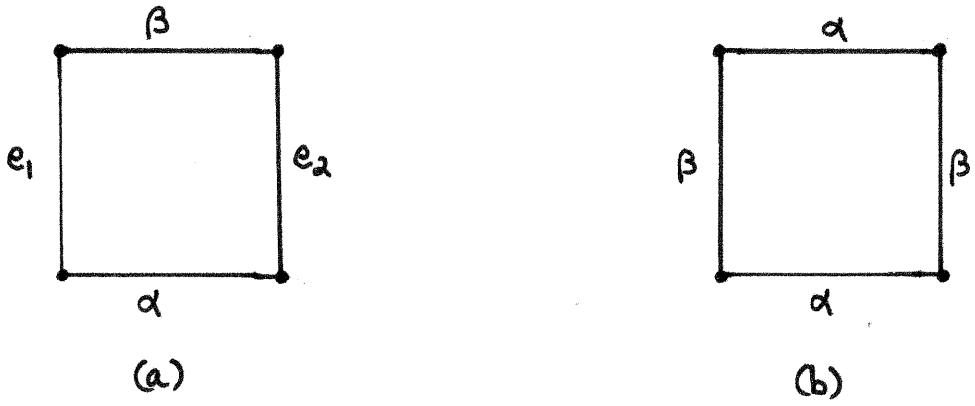


Fig. 2 First recoloring example

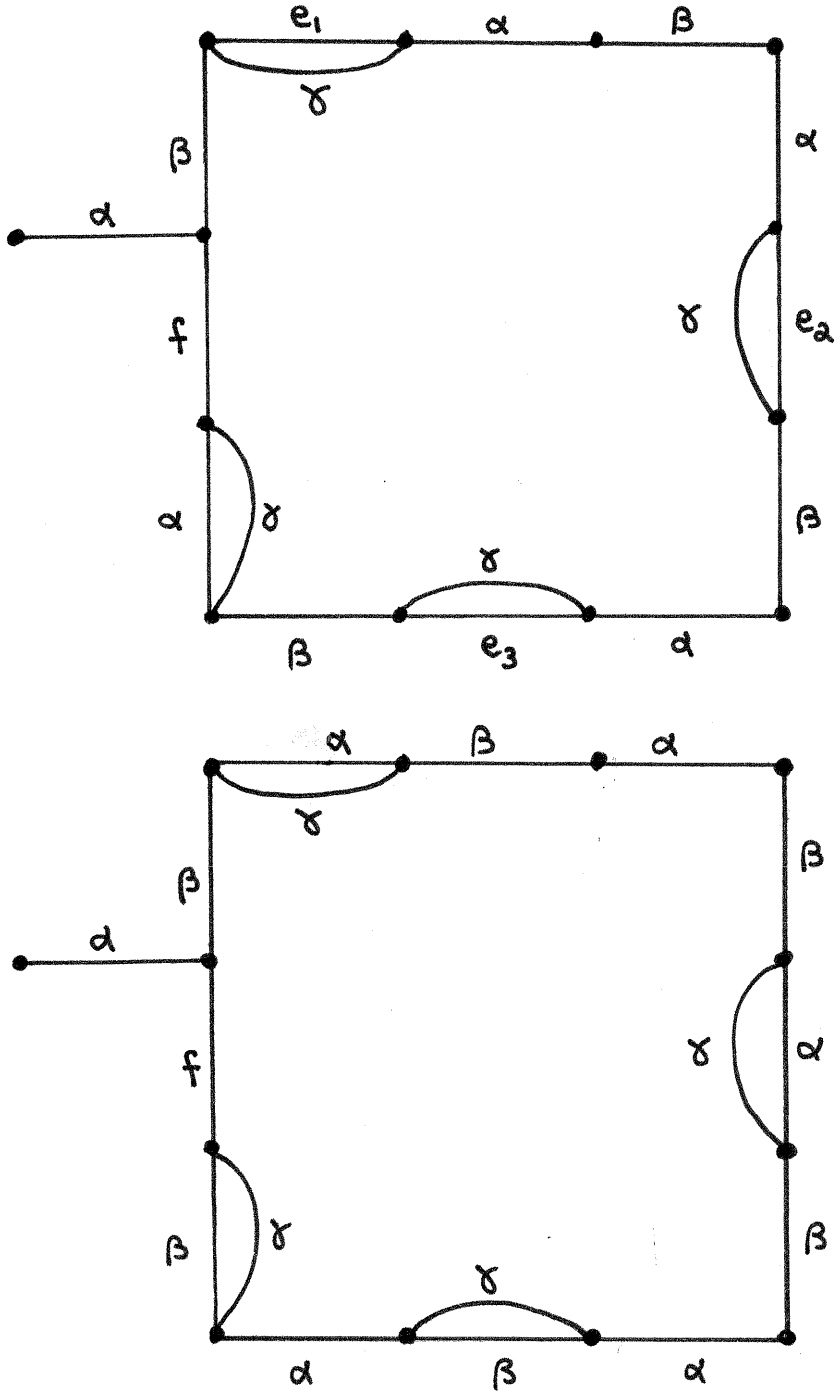


Fig. 3 Second recoloring example

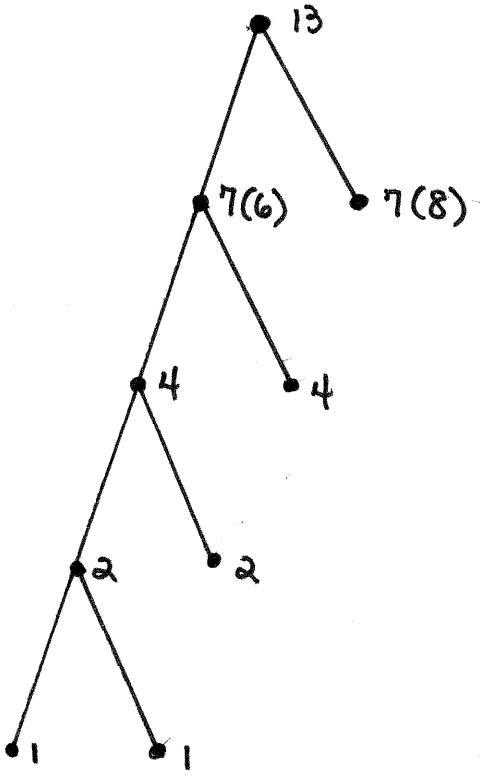


Fig. 4 Partition tree spine