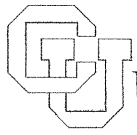
Using Euler Partitions to Edge Color Bipartite Multigraphs

Harold N. Gabow

CU-CS-082-75



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Using Euler Partitions to Edge Color Bipartite Multigraphs

By

Harold N. Gabow
Department of Computer Science
University of Colorado
Boulder, Colorado 80302

Report #CU-CS-082-75 September 1975

An algorithm for coloring the edges of a bipartite multigraph using as few colors as possible is presented. The algorithm uses $O(V^{1/2} E \log V + V)$ time and $O(E + V)$ space. It is based on a divide-and-conquer strategy, using euler partitions to divide the graph. A modification of the algorithm for matching is described. This algorithm finds a maximum matching on a regular bipartite graph with all degrees 2^n, for some n, in $O(E + V)$ time and $O(E + V)$ space.

1. Introduction

Many scheduling problems can be viewed as edge coloring problems. An example is the well-known class-teacher timetable problem [D,G]: The number of hours each class meets with each teacher is given; we must schedule the class-teacher meetings, in the fewest number of hours possible. (In this simple version of the problem, we neglect "preassignments" [D]). This problem is equivalent to finding a minimal edge coloring of a bipartite multigraph. An edge represents a certain teacher meeting a certain class; its color represents the hour the meeting takes place.

We present an algorithm that finds a minimal edge coloring of a bipartite multigraph, in $O(V^{1/2} E \log V + V)$ time and $O(E + V)$ space. A previously known algorithm requires $O(VE)$ time and $O(V^2)$ space. The algorithm is based on a "divide-and-conquer" strategy; an euler partition and a matching are used to divide the graph.

We also describe a modification of the algorithm that finds a matching that covers all maximum degree vertices, in a bipartite graph with maximum degree 2^n. For a regular bipartite graph with all degrees 2^n, the matching is maximum. The algorithm runs in $O(E + V)$ time and $O(E + V)$ space. So in this special case, the algorithm improves the best known time bound, $O(V^{1/2}(E + V))$, for bipartite matching.

Section 2 gives definitions from graph theory. Section 3 describes the edge coloring algorithm and its modification for matching.

2. Definitions

A graph G is a collection of vertices and edges; an edge (v,w) is an (unordered) set of two distinct vertices. V denotes the number of vertices in G, and E denotes the number of edges. If an edge (v,w) occurs more than once, then G is a multigraph. Edge (v,w) is incident to v and to w, and vertices v and w are adjacent.

A subgraph of G is a graph whose vertices and edges are in G. To delete edge e from G means to form the subgraph G-e, consisting of all vertices of G and all edges of G except e. To delete vertex v from G means to form the subgraph G-v, consisting of all vertices of G except v, and all edges of G except those incident to v.

A graph is bipartite with vertex sets (S_1,S_2), if S_1 and S_2 partition the vertices so each edge is incident to a vertex in S_1 and a vertex in S_2.

The degree of a vertex v is the number of edges that are incident to v. A graph is regular if all vertices have the same degree.

A path P is a sequence of edges $(v_1,v_2),(v_2,v_3),\ldots,(v_{n-1},v_n)$. The ends of P are vertices v_1 and v_n. If $v_1 \neq v_n$, P is open; otherwise, P is closed. A graph is connected if there is a path between any two distinct vertices. A connected component of a graph is a maximal connected subgraph.

A matching M on G is a set of edges, no two of which are incident to the same vertex. M covers a vertex incident to an edge in M. An edge coloring of G is an assignment of a color to each edge in G, so all edges of a given color form a matching. A minimal edge coloring uses the fewest number of colors possible.

A tree T is defined recursively as a finite set of nodes, where one node r is chosen as the root, and the remaining nodes are partitioned into disjoint trees, $T_1,\ldots,T_m$. Trees $T_1,\ldots,T_m$ are the subtrees of r. Each node s in T is the root of some subtree S contained in T. The sons of s are the roots of the subtrees of s (in tree S). The nodes of T partition into levels: the root of T is on level 0; if a node is on level i, its sons are on level (i + 1).

3. Edge Coloring Algorithm

This section describes an edge coloring algorithm EC and related algorithms. First Vizing's coloring algorithm is discussed. Next EC is presented. Finally, a modification of EC for matching is described.

We begin by citing a fundamental result.

Lemma 1: Let G be a bipartite multigraph, and let Δ be the maximum degree of a vertex. Then a minimal edge coloring of G uses exactly Δ colors.
Proof: See [B]. QED

The edge coloring algorithms described here use this result. We first describe a construction due to Vizing [Q], which gives an algorithm for edge coloring.

The edges are to be colored with Δ colors. Edges are colored one at a time. Suppose edge (v,w) is uncolored. At most Δ-1 edges (v,x) are colored, so some color a is missing in all edges (v,x); similarly, some color b is missing in all edges (w,y). Construct an "alternating (a,b) path" starting at w, as follows. The path begins with the edge (w,y) that is colored a (if it exists). Consecutive edges in the path are alternately colored a and b. The path ends at a vertex z, where the next color is missing. It is easy to see $z \neq v,w$, if the graph is bipartate.

Now interchange colors along the path: edges colored a are colored b, and edges colored b are colored a. When this is done, color a is missing at both v and w, since $z \neq v,w$. Now edge (v,w) can be colored a.

This algorithm uses O(VE) time, since to color an edge, O(V) time is needed to find the alternating path. The space is $O(E + V\Delta) = O(V^2)$, since an array of size $O(V\Delta)$ is needed. (The array specifies which edge, if any, has a given color at a given vertex. It is used to find the next edge in the alternating path.)

Now we describe an algorithm EC that improves the time and space bounds to $O(V^{1/2} E \log V + V)$ and $O(E + V)$, respectively. The algorithm uses procedures EP and MD that are based on standard graph theory constructions. We describe these procedures, and then describe EC.

Procedure EP finds an euler partition. An euler partition is a partition of the edges of a graph into open and closed paths, so each vertex of odd degree is the end of exactly one open path, and each vertex of even degree is the end no open paths. An euler partition can be found as follows [B]: Choose any vertex of odd degree; if none exists, choose a vertex of even, non-zero degree. Traverse an edge from that vertex to another; erase the edge. Continue traversing and erasing edges, until a vertex with zero degree is reached. Then choose a new start vertex, and repeat the process. Do this until no start vertex of non-zero degree remains.

Procedure EP implements this construction.

```
procedure EP;
comment EP (Euler Partition) finds an euler partition P for a multi-
        graph G.  P is a list of paths p that partition the edges;
begin
1.  make P an empty list;
2.  make S an empty queue; comment S contains all possible start vertices
    for new paths;
3.  put all vertices of odd degree in S;
4.  put all vertices of non-zero even degree in S;
5.  while S is non-empty do
        begin
6.      let s be the first vertex in S;
7.      delete s from S;
```

```
            comment vertex s may have degree 0, since edges are deleted
                    from G in line 13;
 8.         if vertex s has non-zero degree then
            begin
 9.         make a new path p empty;
10.         v: = s;
11.         while vertex v has non-zero degree do
                begin
12.             let (v,w) be an edge in G;
13.             delete (v,w) from G;
14.             put (v,w) in p;
15.             v: = w;
                end;
16.         put path p in P;
17.         if vertex s has non-zero degree then put s in S;
            end;
           end;
      end EP;
```

Lines 2-4 put all non-zero degree vertices in S, as possible start vertices of paths. Odd degree vertices are put first, so the open paths in the partition are generated first.

The loop in lines 5-17 generates each path in the partition. Lines 6-10 choose a start vertex s. The body of the loop in lines 11-15 chooses the next edge in the path. Line 13 deletes that edge from G, so it is not put in another path.

If the start vertex s has odd degree, path p is open. When line 17 is reached, s has even degree; if it has non-zero degree, it is put at the end of queue S.

If the start vertex s has even degree, every vertex in G has even degree, and p is a closed path. When line 17 is reached, s has zero degree, and is not put in S.

For the graph in Figure 1, EP finds the two paths shown in Figure 2.

Now we describe the data structure used by EP. The graph G is represented as a collection of adjacency lists. Each vertex v has a list containing all edges incident to v. So an edge is on two adjacency lists. Adjacency lists are doubly-linked. This facilitates deleting an edge from G, by removing it from the two adjacency lists.

The euler partition P is stored as a linked list of paths. A path p is a linked list of edges. The queue S is a linked list of vertices.

Lemma 2: Procedure EP finds an euler partition, in $O(E + V)$ time and $O(E + V)$ space.

Proof: The correctness of EP follows from the remarks above. Now we discuss the time bound.

Lines 1-4 use $O(V)$ time. Line 6 chooses a vertex s as the start vertex at most twice (once when the degree of s is odd, and once when it is even). So the total time in lines 6-10 is $O(V)$.

Lines 12-15 are executed once in time $O(1)$: Line 12 chooses the first edge in v's adjacency list; line 13 deletes the edge from the two adjacency lists it is on; line 14 puts it in the linked list for p. Since lines 12-15 are executed once for each edge, the total time is $O(E)$.

Lines 16-17 use a total of $O(V)$ time. So EP uses $O(E + V)$ time.

EP uses $O(E + V)$ space for G, $O(E)$ space for P, and $O(V)$ space for S. So the space is $O(E + V)$. QED.

Procedure MD finds a matching. Let G be a bipartite multigraph, with vertex sets S_1, S_2, and let Δ be the maximum degree of a vertex. A matching M, that covers every vertex of degree Δ, can be found as follows.

First for $i = 1,2$, find a matching M_i that covers each vertex of degree Δ in S_i. Note M_i exists, by Hall's Theorem [B]: Any d vertices of degree Δ in S_i are adjacent to at least d vertices, so Hall's Theorem applies.

Next construct the desired matching M, using the method of Mendelsohn-Dulmage [L]. First put all edges of $M_1 \cap M_2$ in M. Next, form $M_1 \oplus M_2$. A connected component C of $M_1 \oplus M_2$ is a path, with edges alternately in M_1 and M_2. For each component C, put all edges of $M_i \cap C$ in M, where i is chosen as follows: If C is an open path of odd length, choose i so $|M_i \cap C|$ is maximum. If C is an open path of even length, one end of C is a vertex of degree Δ; choose i so M_i covers that vertex. If C is a closed path, choose i arbitrarily. When this is done, M is a matching, that covers every vertex of degree Δ.

Procedure MD implements this construction.

```
procedure MD(Δ);
comment MD (Match Δ) finds a matching M on G that covers every vertex
        of degree Δ. Here G is a bipartite multigraph, in global
        storage. It has vertex sets S1,S2. Δ is the maximum degree
        of a vertex.
begin
1.  for i: = 1,2 do comment find a matching Mi that covers every vertex
    of degree Δ in Si;
    begin
2.  let T be the set of vertices in Si with degree <Δ;
```

```
3.    let H be the multigraph G-T;
4.    let M_i be a maximum matching on H;
      end;
      comment use the Mendelsohn Dulmage construction to combine matchings;
5.    M:= M_1 ∩ M_2;
6.    N:= M_1 ⊕ M_2;
7.    for each connected component C of N do
          begin
8.        let C be the sequence of edges e_1,...,e_r;
9.        wlog assume C starts with a vertex of degree Δ;
10.       for i: = 1 step 2 to r do
11.           put e_i in M;
          end;
      end MD;
```

Figure 3(a) shows a graph; the starred edges form the matching found by MD. Figure 3(b) shows matchings M_1 and M_2, and the components of $M_1 \oplus M_2$; edges in M_i have the label i.

Lemma 3: Procedure MD finds a matching that covers every vertex of degree Δ, in $O(V^{1/2}(E + V))$ time and $O(E + V)$ space.

Proof: The correctness of MD follows from the remarks above.

Now consider the time. Lines 1-3 use $O(V)$ time. (In line 3, it is only necessary to flag the vertices of T.) Line 4 can be done once in time $O(V^{1/2}(E + V))$ [HK]. The matchings M_i are stored so each vertex v indicates which edge of M_i contains v. So lines 5-11 require $O(V)$ time. Thus the total time is $O(V^{1/2}(E + V))$.

The space bound is obvious, since line 4 requires no extra storage. QED.

Now we describe the edge coloring algorithm EC. EC uses a divide-and-conquer approach [AHU]: Divide the multigraph G into subgraphs G_1 and G_2, where each subgraph has maximum degree $\Delta/2$. Color each subgraph with $\Delta/2$ colors. (This is possible by Lemma 1). This gives a coloring of G with Δ colors, which is minimal.

To divide G, construct an euler partition. Traverse each path of the partition, placing edges alternately in G_1 and G_2. If vertex v has degree d in G, its degree in G_1 or G_2 is $\lfloor d/2 \rfloor$ or $\lceil d/2 \rceil$.

If Δ is even, this approach produces two subgraphs with maximum degree $\Delta/2$, as desired. If Δ is odd, both subgraphs may have maximum degree $\lceil \Delta/2 \rceil$. This would give a coloring of G with $\Delta + 1$ colors, which is not minimal.

To prevent this, if Δ is odd, first construct a matching M that covers all vertices of degree Δ. Assign one color to the edges of M. Then delete these edges. This makes Δ even, so now the divide-and-conquer approach can be used.

Procedure EC implements this construction.

```
procedure EC;
comment EC (Edge Color) finds a minimal edge coloring for a bipartite
        multigraph G. For each edge e, color (e) is set to the appro-
        priate color;
begin
multigraph G; comment G is stored in global storage;
procedure REC (Δ);
comment REC (Recursive Edge Color) recursively colors a bipartite multi-
        graph G, that contains no vertices of degree 0. Δ is the
        maximum degree of a vertex;
begin
```

```
1.  if Δ is odd then
        begin
        comment make M a matching that covers every vertex of degree Δ,
                and color the edges in M;
2.      if Δ = 1 then M: = G else MD; comment MD finds a matching M;
3.      let c be a new color;
4.      for each edge e ε M do
            begin
5.          color (e): = c;
6.          delete e from G;
            end;
        end;
7.  EP; comment EP makes P an euler partition of the edges in G;
8.  if P is not empty then
        begin
9.      make L1 and L2 empty lists;
10.     for each path p in P do
            begin
11.         let p be the sequence of edges e1,...,er;
12.         for i: = 1 to r do
13.             if i is odd then put ei in L1
                else put ei in L2;
            end;
14. for i: = 1,2 do
        begin
15.     let G be the multigraph consisting of the edges in Li and all
            vertices incident to these edges;
16.     REC(⌊Δ/2⌋); comment REC colors the edges in Li;
```

```
        end;
    end;
end REC;
17.  delete all vertices of degree 0 from G;
18.  let Δ be the maximum degree of a vertex;
19.  REC(Δ);
end EC;
```

Figure 1 shows an input graph for EC. Figure 2 shows the partition found by EP. Figure 4 shows the lists of edges L_1 and L_2 constructed from Figure 2. The recursive calls to REC assign colors 1-4 to the edges, as indicated in Figure 4.

Theorem 1: Procedure EC finds a minimal edge coloring of a bipartite multigraph, in $O(V^{1/2} E \log V + V)$ time and $O(E + V)$ space.

Proof: The correctness of EC follows from the preceding remarks, using induction on Δ. Now we discuss the time bound. In EC, lines 17-18 require $O(E + V)$ time. So it suffices to show that if there are no vertices of degree 0, REC runs in $O(V^{1/2} E \log \Delta + V)$ time. We do this below. First note that since there are no degree 0 vertices, $V \leq 2E$, i.e., $V = O(E)$.

Suppose $\Delta = 1$. Then G is a matching, containing at most V edges. Lines 1-6 color the edges in $O(V)$ time, and the total time is $O(V)$.

Now suppose $\Delta > 1$. If Δ is odd, line 2 uses $O(V^{1/2} (E + V)) = O(V^{1/2}E)$ time, and lines 3-6 use $O(V)$ time. So lines 1-6 require $O(V^{1/2}E)$ time.

Lines 7-13 require $O(E + V) = O(E)$ time, as do lines 14-15. So the total time used by REC is $O(V^{1/2}E)$ plus the time for two recursive calls to REC in line 16.

Now consider a tree whose vertices represent recursive calls to REC. The root represents the original call to REC. A call REC (Δ), where $\Delta > 1$, has two sons, representing the two calls in line 16. A call REC(Δ) where $\Delta = 1$ has no sons.

The tree contains $\lfloor \log_2 \Delta \rfloor + 1$ levels. Consider the graphs for all calls on a given level. Let there be k such graphs. An edge of the original graph is in at most one of these graphs. Let V_i and E_i be the number of vertices and edges in these graphs, for $i = 1,\ldots,k$. Then

$$\sum_{i=1}^{k} V_i^{1/2} E_i \leq \sum_{i=1}^{k} V^{1/2} E_i \leq V^{1/2} E.$$

Thus a total of $O(V^{1/2}E)$ time is spent in all calls in a given level. (On the bottom level of the tree, all graphs have $\Delta = 1$, so the total time is

$$\sum_{i=1}^{k} O(V_i) = \sum_{i=1}^{k} O(E_i) = O(E) = O(V^{1/2}E)).$$

Thus REC requires $O(V^{1/2} E \log \Delta)$ time, if $\Delta > 1$. This proves the time bound for REC.

Now consider the storage space. The graph G uses $O(E + V)$ space. At a given point in the algorithm, an edge of the input graph that is not in G is in at most one list L_i; so the lists L_i require $O(E)$ space. The color array uses $O(E)$ space. The remaining space is needed for implementing the recursion. Since the depth of the recursion is at most $\lfloor \log_2 \Delta \rfloor$, at most $O(\log V)$ space is used. So a total of $O(E + V)$ space is used. QED

Now consider a special case, $\Delta = 2^n$. In the calls to REC, Δ assumes the values 2^i, for $i = n, n - 1,\ldots, 0$. So in line 2 of REC, MD is never called. Thus the following bound holds.

Corollary 1: If $\Delta = 2^n$, procedure EC finds a minimal edge coloring in $O(E \log V + V)$ time and $O(E + V)$ space.

An interesting question is whether the time for EC can be improved in other cases. When Δ is odd, it is not strictly necessary to find a matching; it suffices to partition the edges into subgraphs having maximum degrees $\lfloor \Delta/2 \rfloor$ and $\lceil \Delta/2 \rceil$. It may be possible to use a faster partitioning procedure, thereby improving the time bound.

Now we discuss an application of EC to matching. Corollary 1 implies a matching that covers all vertices of degree $\Delta = 2^n$ can be found in $O(E \log V + V)$ time. This improves algorithm MD. By rewriting EC for this special case, we can further improve the bound. The matching algorithm follows.

```
    procedure MD2;
    comment MD2 (Match Δ = 2^n) finds a matching that covers all vertices
            of degree Δ, if Δ = 2^n;
    begin
1.  delete all vertices of degree 0 from G;
2.  let Δ be the maximum degree of a vertex;
3.  while Δ > 1 do
        begin
4.      EP; comment make P an euler partition of the edges in G;
5.      for each path p in P do
            begin
6.          let p be the sequence of edges e1,...,er;
7.          for i: = 1 step 2 to r do
8.              delete ei from G;
            end;
```

```
 9.          delete all vertices of degree 0 from G;
10.          Δ: = Δ/2;
             end;
11.     M: = G;
        end MD2;
```

For the input graph of Figure 1, the matching consists of the edges labelled 4 in Figure 4.

Theorem 2: Let G be a bipartite graph with $\Delta = 2^n$. Procedure MD2 finds a matching that covers every vertex of degree Δ, in $O(E + V)$ time and $O(E + V)$ space.

Proof: The correctness of MD2 is proved similar to the correctness of EC. Now we discuss the time bound. The body of the loop in lines 3-10 is executed once in $O(E + V)$ time. Since the graph contains no degree 0 vertices (lines 1,9), this time is $O(E)$. The loop is executed for graphs containing at most $E/2^i$ edges, where $i = 0,\ldots,n-1$. So the total time in lines 3-10 is $O(E)$. Lines 1-2 and 11 require $O(E + V)$ time. So MD2 uses $O(E + V)$ time.

Since storage is needed only for the graph G, the space is $O(E + V)$. QED

The following instance Theorem 2 improves the time bound for bipartite matching [HK] in a special case:

Corollary 2: Let G be a regular bipartite graph with $\Delta = 2^n$. Procedure MD2 finds a maximum matching in $O(E + V)$ time and $O(E + V)$ space.

It is an easy exercise to modify MD2, so if Δ is arbitrary, the matching covers at least half the vertices of degree Δ.

4. Acknowledgments

The author wishes to thank Professor Eugene Lawler of the University of California at Berkeley, for his stimulating conversations on edge coloring.

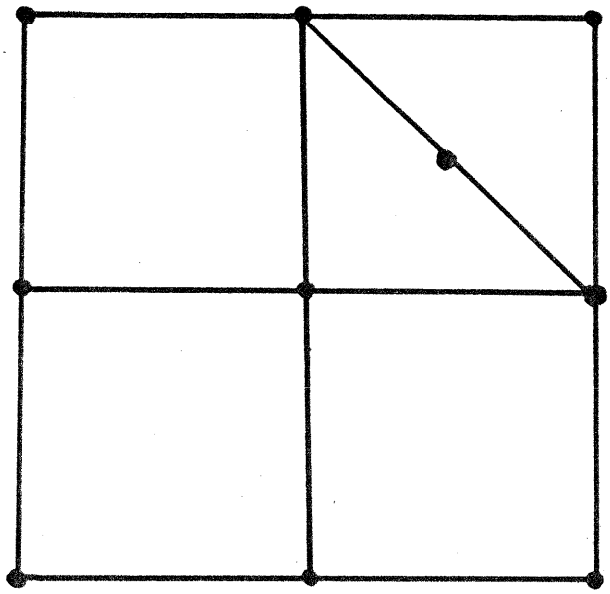


Figure 1

Input graph

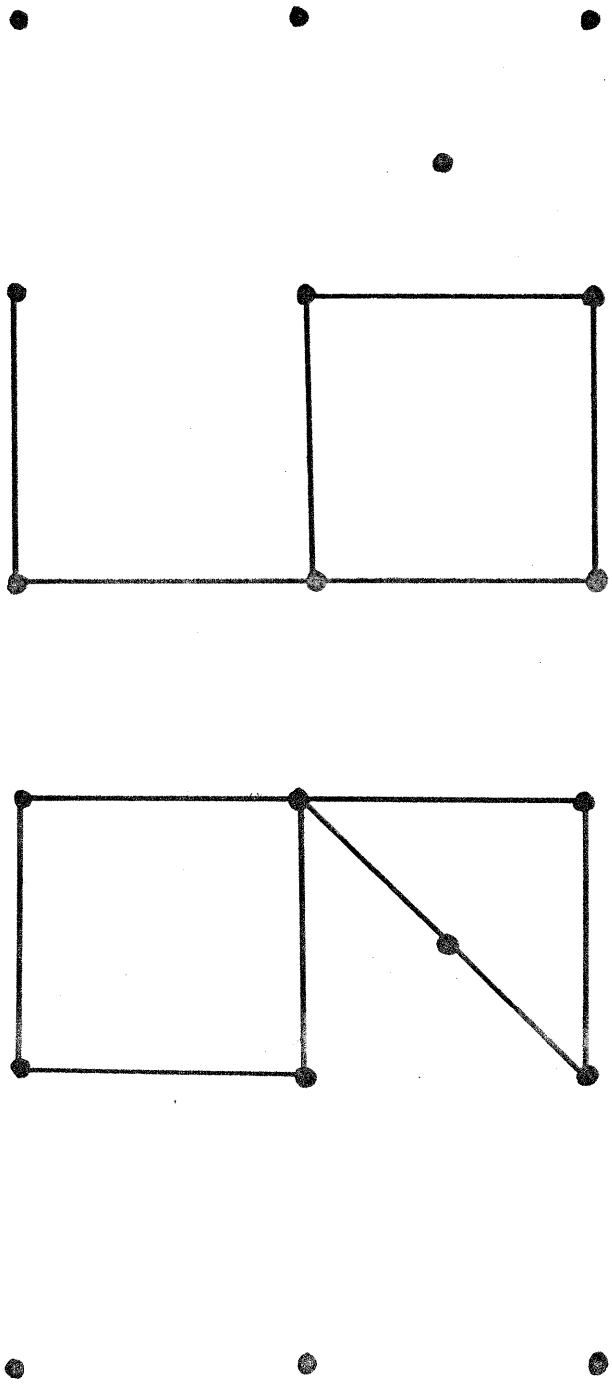
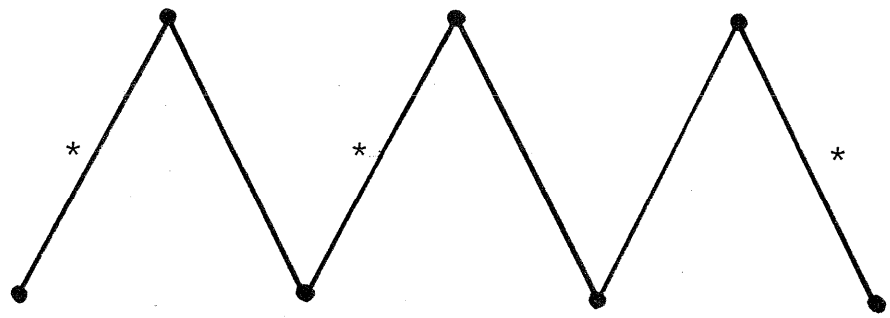
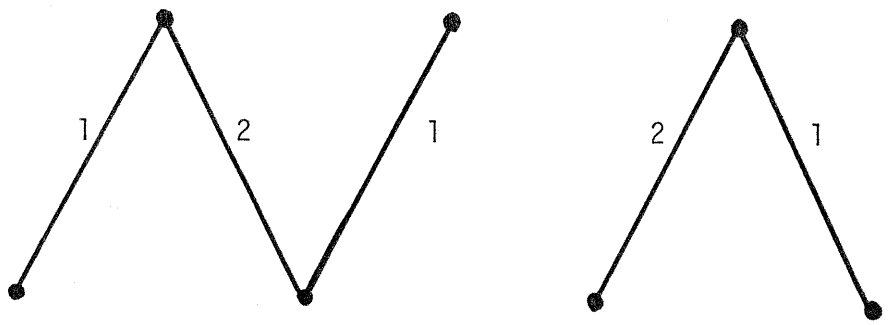


Figure 2

Euler partition



(a)



(b)

Figure 3

Constructing a matching

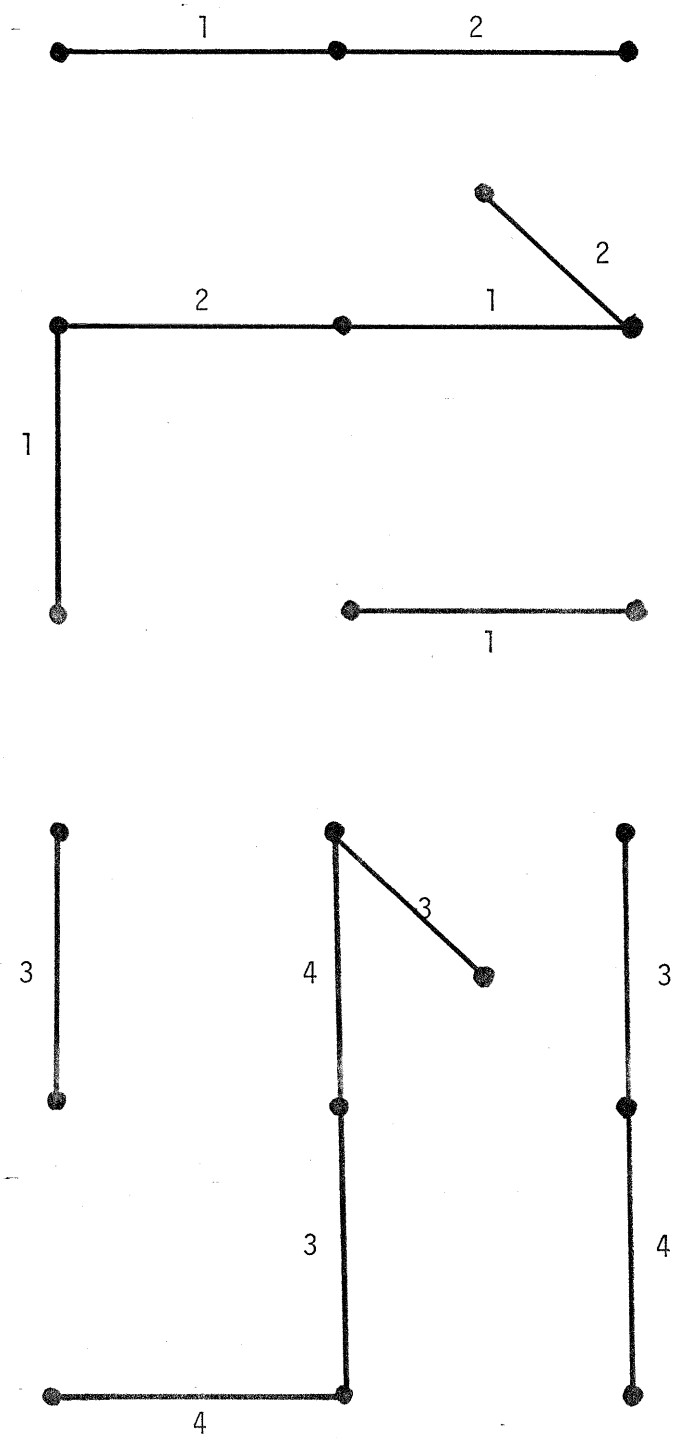


Figure 4

Edge lists L_i, and edge colors

References

[AHU] Aho, A. V., J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

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

[D] Dempster, M. A. H., "Two algorithms for the time-table problem", in Combinatorial Mathematics and Its Applications, D. J. A. Welsh, Ed., Academic Press, London, 1969, 63-85.

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

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

[L] Lawler, E. L., Combinatorial Optimization Theory, to be published.

[O] Ore, Oystein, The Four Color Problem, Academic Press, New York, 1967.