On Edge Coloring Bipartite Graphs*

R. Cole

J. Hopcroft

TR 80-443

November 1980

Department of Computer Science
Cornell University
Ithaca, New York 14853

*This research was supported in part by ONR grant N00014-76-C-0018.

1. INTRODUCTION

An algorithm for finding a minimal edge coloring of a bipartite graph in time $O(E \log V)$ is presented. Polynomial time algorithms for this problem have previously been given by Gabow in [1] and by Gabow and Kariv in [2], the best time bounds being $O(E \log^2 V)$ and $O(V^2 \log V)$.

The algorithm is based on using fast methods for finding maximal matchings in semiregular bipartite graphs; an algorithm for finding a maximal matching in a general bipartite graph was given by Hopcroft and Karp in [3]. Two algorithms for finding such a matching are given. Although the second one always has a faster running time of $O(\max\{E, V \log V \log^2 D\})$, the first one is presented for the sake of clarity.

2. NOTATION AND DEFINITIONS

Throughout this paper $G = (V,E)$ denotes a graph, V its vertex set and E its edge set. $G = (V_1,V_2,E)$ denotes a bipartite graph with V_1 and V_2 being disjoint vertex sets and $E \subseteq V_1 * V_2$ being the edge set. D denotes the maximal degree of any vertex in $V = V_1 \cup V_2$.

A graph is said to be regular if all its vertices have the same degree. A bipartite graph is said to be semiregular if all the vertices in V_1 have the same degree D, the maximal degree of any vertex in G; it is said to be high-low if there exists an integer k such that $\deg(v) \geq k$ if $v \in V_1$ and $\deg(v) \leq k$ if $v \in V_2$.

An Euler partition is a partition of the edges into open and closed paths, so that each vertex of odd degree is at the end of one open path, and each vertex of even degree is at the end of no open paths.

An Euler split of a graph $G = (V_1,V_2,E)$ is a pair of graphs $G_1 = (V_1,V_2,E_1)$ and $G_2 = (V_1,V_2,E_2)$ where E_1 and E_2 are formed from an Euler partition of E by placing alternate edges of each path into E_1 and E_2 respectively. Any vertex of even degree in G will have the same degree in both G_1 and G_2, while any vertex of odd degree in G will have degrees in G_1 and G_2 differing by one. This implies that if G is semiregular, and D, the maximal degree of any vertex in G is even, then both G_1 and G_2 are semiregular. An algorithm for finding an Euler split in time $O(E)$ is given in [1].

3. ALGORITHM 1

An $O(E \log V)$ time algorithm for finding a maximal matching in a semiregular bipartite graph is given. The algorithm works by partitioning E into sets E_1 and E_2 such that $G_1 = (V_1,V_2,E_1)$ and $G_2 = (V_1,V_2,E_2)$ are both nontrivial semiregular bipartite graphs. If the graph with smaller edge set has maximum degree one it is the required matching; otherwise the algorithm is applied recursively to that graph. Since each iteration reduces E by at least a factor of two, the algorithm eventually terminates.

The partitioning procedure to obtain the semiregular graphs G_1 and G_2 is as follows. An Euler split of G is made giving graphs G_1 and G_2. If D, the maximal degree of any vertex in G is even, then both G_1 and G_2 are semiregular. Otherwise edges are moved between G_1 and G_2 to make them semiregular. This is described more precisely below.

Let M be the set of maximum degree vertices in G. At least half of the vertices in M will have even degree in one of G_1 or G_2. Without loss of generality let G_2 be that graph in which at least half the vertices are of even degree. Then let M_1 be those vertices of M that have even degree in G_2 and M_2 be the remaining vertices of M.

Next an Euler split of G_2 is made giving graphs G_{21} and G_{22}. The vertices of M_1 have the same degree in G_{21} and G_{22}, while some of the vertices in M_2 have even degree in G_{21} and odd degree in G_{22} and the others have odd degree in G_{21} and even in G_{22}; these degrees differ by one, and one of them is the degree of the vertices of M_1 in G_{21} (and in G_{22}). Without loss of generality let G_{22} be the graph in which at least half the vertices of M_2 have even degree. Let M_{21} be the subset of vertices of M_2 that have even degree in G_{22} and let M_{22} be the remaining vertices of M_2.

Now one of the graphs G_{21} or G_{22} is combined with G_1 in such a way that vertices in M_1 will have the same degree as vertices in M_{21} in the combined graph. The new graphs are named G_1 and G_2 in such a way that vertices in M_{22} are of odd degree in G_2. M_1 and M_2 are redefined with M_2 reduced in size by at least a factor of two. The process is repeated until $M_1 = M$ when the vertices in M all have the same even degree in G_1. The partitioning procedure is shown in algol like form below.

PROCEDURE PARTITION

```
G = (V,E)
M = set of maximum degree vertices of G
BEGIN
     Let G1, G2 be an Euler split of G;
          At least half of the vertices in M have even degree in one of
          G1 or G2, Let it be G2;
     Let M1 = {v|v ∈ M, and v has even degree in G2};
     Let M2 = M - M1;
     WHILE (|M|2 ≠ 0) DO
     BEGIN
          Let G21, G22 be an Euler split of G22;
               Again at least half the vertices in M2 have even degree in
               either G21 or G22, Let it be G22;
          Let M21 = {v|v ∈ M2, and v has even degree in G22};
          Let M22 = M2 - M21;
          IF degree of a vertex in M1 in G22 is even
               then
                    G1 := G1 ∪ G21, G2 := G22
               else
                    G2 := G1 ∪ G22, G1 := G21;
          M1 := M1 ∪ M21, M2 := M22;
     END
END
```

CORRECTNESS

It is necessary to show that all the vertices in M_1 have the same degree in G_1 at any given stage of the algorithm, and likewise in G_2. The same result should be proven for vertices in M_2. It will first of all be illustrated by an example.

Consider the example in which vertices in M_1 have degree 5 in G_1 and degree 12 in G_2, while those in M_2 have degree 4 in G_1 and degree 13 in G_2.

Then vertices in M_1 have degree 6 in both G_{21} and G_{22}; vertices in M_{21} have degree 7 in G_{21} and degree 6 in G_{22}; and vertices in M_{22} have degree 6 in G_{21} and degree 7 in G_{22}. So the assignments $G_1 = G_1 \cup G_{21}$, $G_2 = G_{22}$, $M_1 = M_1 \cup M_{21}$, and $M_2 = M_{22}$ are made.

Now vertices in M_1 have degree 11 in G_1 and degree 6 in G_2, and vertices in M_2 have degree 10 in G_1 and degree 7 in G_2.

By considering respectively the cases in which the degree in G_1 of vertices in M_1 is one greater or one lesser than that of vertices from M_2 it can be proven by induction that the degree of vertices in M_1 is the same in each of G_1 and G_2 and likewise for M_2. Thus all the vertices in M have the same degree in each of G_1 and G_2 when the partitioning procedure terminates.

To show that G_1 and G_2 are both semiregular it is necessary to show that:

deg(v) in G_1, v not in $M \le$ deg(v) in G_1, $v \in M$, and similarly in G_2.

This is proven by an induction using the inductive hypothesis that:

deg(v) in G_1, v not in $M \le$ deg(v) in G_1, $v \in M_1$, and likewise in G_2.

To show that both G_1 and G_2 are nontrivial the following inductive hypothesis is proven:

deg of vertices of M_1 in $G_2 > 0$.

In fact this degree is even and so the degree of vertices from M_1 in G_{21} and G_{22} is nonzero. The induction now follows.

TIMING

Each iteration of the while loop reduces the size of M_2 by at least a factor of two. So after at most $O(\log V_1)$ iterations $|M_2| = 0$ and the procedure terminates. Each iteration of the while loop takes time O(E). So to obtain G_1 and G_2 takes time $O(E \log V_1) \le O(E \log E/D)$. Thus the time T(E) taken to find a matching is given by:

$$T(E) = O(E \log E/D) + T(E/2) = O(E \log E/D) \le O(E \log V).$$

In fact this algorithm will produce a matching covering all the vertices of maximal degree in a general bipartite graph in time $O(E \log E/D)$.

4. ALGORITHM 2

An $O(E + V \log V \log^2 D)$ time algorithm for finding a maximal matching in a semiregular bipartite graph is given.

OBSERVATION

It is known that if a network has a maximal flow, with the flow through each vertex being integral, then it has a maximal flow such that the flow through each edge is integral [4, p113].

In particular, if the edges of a bipartite graph are assigned positive weights, so that the sum of the weights at each vertex is at most one, and the weight at each vertex of V_1 is one, then the graph has a matching covering every vertex of V_1.

By shifting the weights between edges, while maintaining a constant weight at each vertex, and deleting edges of weight zero, a new smaller graph is obtained containing just as large a matching.

METHOD

Vertices of small degree are merged together so that all vertices have degrees between D/2 and D. Now $V*D = O(E)$. This simplifies the timing analysis. Every edge is given weight 1/D. Using depth first search, cycles in the graph among edges of weight 1/D are found. When a cycle is found alternate edges in the cycle are deleted; the other edges in the cycle have their weight doubled. This is continued until there are no cycles among edges of weight 1/D. Cycles are then found among edges of weight 2/D, 4/D ..in turn until no further increase in edge weights can be obtained in this way.

A graph with at most O(V log E/V) weighted edges is obtained, such that the sum of the weights at each vertex in V_1 is one. Algorithm 1 is now adapted to find this matching.

Each edge is considered to have multiplicity D times its weight . Four copies of each edge are kept, one in each of G_1, G_2, G_{21} and G_{22}. When making an Euler split, each edge added to an Euler path is made to occur as often as it can at that point in the path; the multiplicities of the copies of the edge are changed accordingly. Otherwise one proceeds just as in algorithm 1.

As with algorithm 1 this algorithm can be used to find a matching covering all the vertices of maximal degree in a general bipartite graph in the same time bound.

TIMING

One iteration of the procedure in algorithm 1 cuts the number of edges in half (counting edges according to their multiplicity), but may well not reduce the size of the structure stored, affecting the timing given with algorithm 1.

To find the graph of size O(V log E/V), O(E) time is needed. To use algorithm 1 one requires time O(V log E/V log E/D) to halve the number of edges (counted according to their multiplicity), and time O(V log E/V log E/D log D) = $O(V \log V \log^2 E/V)$ to obtain a maximal matching.

Thus the overall time taken is O(E) for $E \geq O(V\log V (\log\log^2 V))$ and

$O(V \log V \log^2 E/V)$ otherwise, which is always better than the $O(E \log V)$ of algorithm 1.

5. COLORING THE EDGES OF A BIPARTITE GRAPH

An $O(E \log V)$ time algorithm for finding a minimal edge coloring is given. It is minimal in the sense that the fewest possible colors are used. It is shown in [1] that this is D colors.

The algorithm is based on divide and conquer. When a graph has vertices of even maximal degree, using the method of Euler partitions it is split into two subgraphs of equal maximal degree which are then recursively colored. On occasion when the graph has vertices of odd maximal degree a matching M covering all the vertices of maximal degree has to be found. This is obtained by using algorithm 2.

ALGORITHM

If D is odd find a matching as described above; color it and delete it from G. Set $D = D - 1$.

Make an Euler split of G to give two bipartite graphs G_1 and G_2 each having vertices of maximal degree $D/2$.

WLOG assume G_1 has a smaller edge set than G_2 (otherwise swap the labels G_1 and G_2), and recursively color G_1. Let $2^k < D/2 = 2^{k+1} - r$. Add the r smallest sets of colored edges to G_2, delete them from the set of colored edges, and recursively color G_2.

A similar method was used in [3] and led to the current presentation of the algorithm. That exactly D colors are used can be shown by induction.

TIMING

Excluding the time taken to find the matchings, the time taken is given by:

$$T(E, D) = T(E_1, \lfloor D/2 \rfloor) + T(E_2 \cup E_3, \lfloor D/2 \rfloor + r) + O(E),$$

where $E_1 \cup E_2 = E$ and E_3 is the union of the r sets of colored edges added to G_2. For D a power of two $T(E, D) = O(E \log D)$. In all other cases as $|E_3| \leq 2r\,|E_1|/D \leq |E_2|$ one finds that $T(E, D) = O(2E \log(D + r)) = O(E \log D)$.

The time required for finding the matchings is bounded by $O(\max\{E \log D, V \log V \log^3 D\}) \leq O(\max\{E \log D, E \log V\})$ and hence the total time required is bounded by $O(E \max\{\log D, \log V\})$ which is $O(E \log V)$. For a graph without multiple edges a time bound of O(E max{log D, log V}) is obtained.

6. MATCHINGS IN HIGH LOW GRAPHS

By pruning the high-low graph a semiregular graph with D = k can be obtained, D being the maximal degree of any vertex in the semiregular graph. The matching algorithm is then applied to this graph to obtain a maximal matching for the high-low graph. So a maximal matching in a high-low graph can be found in time $O(\max\{E, V \log V \log^2 E/V\})$. High-low graphs were defined in [3] and the above method for pruning was described there.

REFERENCES

[1] Gabow - Using Euler partitions to edge colour bipartite multigraphs, IJCIS 5,4,Dec 76.

[2] Gabow and Kariv - algorithms for edge colouring bipartite graphs and multigraphs, SIGACT 78.

[3] Hopcroft and Karp - An $n^{5/2}$ algorithm for maximal matchings in bipartite graphs, SIAM 2,4,Dec 75.

[4] Lawler - Combinatorial Optimization, Networks and Matroids, Holt-Rinehart-Winston.