Edge-Coloring Bipartite Multigraphs in $0(E \log D)$ Time

Richard Cole[*] Kirstin Ost[†] Stefan Schirra[‡]

September 21, 1999

Abstract

Let V, E, and D denote the cardinality of the vertex set, the cardinality of the edge set, and the maximum degree of a bipartite multigraph G. We show that a minimal edge-coloring of G can be computed in $O(E \log D)$ time.

1 Introduction

The edge-coloring problem is to find a minimal edge-coloring of a given multigraph. An *edge-coloring* of a multigraph $G = (V, E)$ with vertex set V and edge set E is a map $c : E \longrightarrow \mathbb{N}$ giving each edge $e \in E$ a *color* $c(e) \in \mathbb{N}$ such that no two adjacent edges have the same color. If edge-coloring c uses at most k colors, c is called a *k-edge-coloring*. A *color class* of an edge-coloring c of a multigraph G refers to the subset of edges having a certain color in c. An edge-coloring of a multigraph G is said to be *minimal* if there is no edge-coloring of G using fewer colors.

The number of colors used in a minimal edge-coloring of a multigraph G is called the *chromatic index* of G. It is well known that the chromatic index of a simple graph G is equal to D or $D + 1$, where D denotes the maximum degree of any vertex in G [Vi]. Upper bounds on the chromatic index of multigraphs can be found in [NZN]. In general it is an NP-complete problem to determine the chromatic index of a multigraph [Ho]. However, there are some classes of graphs for which the chromatic index always is the maximum degree. The most important is the class of bipartite multigraphs. Throughout this paper we restrict ourselves to this class.

The edge-coloring problem for bipartite multigraphs has applications in different areas. Many scheduling problems can be formulated in terms of edge-coloring bipartite multigraphs, for example the class-teacher time-table problem [Go] and the file transfer problem [NZN]. Other applications of edge-coloring bipartite multigraphs are routing permutation networks [LPV], the k-k routing problem [Si], and the simulation of a PRAM on a distributed memory machine [KLM]. Finally algorithms for edge-coloring bipartite multigraphs can be applied in matching theory.

[*]Courant Institute of Mathematical Sciences, New York University, NY 10012-1185, USA, `cole@cs.nyu.edu`. This work was supported in part by NSF grant CCR-9800085.

[†]SAP Retail Solutions, 66386 St. Ingbert, Germany, `kirstin.ost@sap-ag.de`.

[‡]Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany, `stschirr@mpi-sb.mpg.de`.

The best running times previously known for computing a D-edge-coloring were $O(E\log D+V\log V\log D)$ [Ost] and $0(ED)$ [Sch]. In Section 4, we present an algorithm which computes a minimal edge-coloring of G in time $O(E\log D)$.

This paper is organized as follows. In Section 2 we briefly report on previous related work. In Section 3 we review some basic techniques. In Section 4 we give our coloring algorithm. In Section 5 we give a variant of the algorithm with the same running time, which amounts to using the approach given in [Ost].

2 Previous Work

The first algorithms solving the edge-coloring problem for bipartite graphs were derived from classic graph theory. Splitting G into D matchings using the matching algorithm of [HK] combined with a construction of Mendelsohn and Dulmage [La] gives an algorithm running in time $O(E\sqrt{VD})$. Coloring edge by edge using augmenting paths leads to an $O(EV)$ time algorithm. In 1976 Gabow presented a solution based on a divide and conquer approach that takes $O(E\sqrt{V}\log D)$ time [Ga]. In 1978 Gabow and Kariv developed an algorithm for edge-coloring bipartite graphs with time complexity $O(\min\{E\sqrt{V\log V}, E(\log V)D, V^2\log D\})$ [GK78]. They also used a divide and conquer approach but they make the most effort in the conquer step while Gabow concentrated on the divide step. Furthermore, Gabow and Kariv took advantage of the fact that a graph with maximum degree a power of two can be edge-colored efficiently. Using this fact again in an algorithm which works by repeatedly enlarging a partial edge-coloring until all edges are colored they achieved a running time of $O(E(\log V)^2)$ in 1982 [GK82]. In the same year Cole and Hopcroft developed an algorithm which takes time $O(E\log D+V\log V(\log D)^3)=O(E\log V)$ [CH]. The algorithm combined the idea of Gabow with a faster matching algorithm and the fact that a graph with maximum degree a power of two can be edge-colored efficiently. Improving the matching algorithm led to an $O(E\log D+V\log V(\log D)^2)$ time solution in [Co]. Recently, [Sch] found an $O(E\cdot D)$ time matching algorithm, which leads to an $O(E\cdot D)$ time edge coloring algorithm. In 1995, Cole, Ost and Schirra [Ost] showed that it sufficed to find one matching; the same result was also found by Kapoor and Rizzi in 1996 [KR]. In combination with the matching algorithm in [Co], this yielded an $O(E\log D+V\log V\log D)$ time algorithm for edge coloring.

3 Preliminaries

WLOG we assume $E=V\cdot D$ (to achieve this low degree vertices are combined to give degree at least $D/2$ to all but at most one vertex; then some vertices may be added to one side of the graph to give the same number of vertices on each side; finally $O(E)$ edges are added to give every vertex degree exactly D cf. [CH]). Graphs in which every vertex has the same degree are said to be *regular*.

Nearly all previous algorithms for edge-coloring bipartite multigraphs [in principle] use a divide and conquer approach: They partition G into subgraphs that are recursively colored with different color sets. To obtain a minimal edge-coloring it is crucial that the maximum degrees of the subgraphs sum up to D. Hence the edges of G have to be divided so that each of the resulting subgraphs is regular as well. A decomposition $\mathcal{D}$ of G into t nonempty subgraphs $G_1 = (V, E_1), G_2 = (V, E_2), \ldots, G_t = (V, E_t)$, where $E_1, E_2, \ldots, E_t$ form a partition of E, is called *regular* if all subgraphs $G_i \in \mathcal{D}$, $1 \leq i \leq t$, are regular.

A *matching* in G is a subset M of E with the property that no two edges have a common endpoint. M is said to be *full* if it contains, as an endpoint, every vertex of G. Obviously each color class of a D-edge-coloring of G is a full matching in G.

Euler partitions are a helpful tool in forming regular decompositions. An *Euler partition* of a graph G is a partition of its edges of into open and closed paths, such that each vertex of odd degree is at the end of exactly one open path, and each vertex of even degree is at the end of no open path. An *Euler split* of bipartite graph G splits G into two bipartite graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ where E_1 and E_2 are formed by scanning the paths of an Euler partition of G and alternately placing one edge into E_1 and one edge into E_2. 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 regular and D is even, then all the vertices in G will have degree $D/2$ in each of G_1 and G_2 and this is the maximum degree of any vertex in G_1 or G_2. Hence the decomposition is regular.

It immediately follows that for a graph with $D = 2^i$, $i \in \mathbb{N}_0$, a minimal edge-coloring of G can be computed by splitting the graph recursively by Euler splits until all resulting subgraphs have degree one. This gives a decomposition into D matchings that can trivially be 1-colored. This algorithm runs in time $O(E \log D)$.

If G has an odd maximum degree a decomposition of G into G_1 and G_2 obtained by an Euler split is not necessarily regular. This follows from the fact that each vertex of maximum degree in G will have degree $\lceil D/2 \rceil$ in G_1 and degree $\lfloor D/2 \rfloor$ in G_2 or vice versa. Hence G_1 and G_2 might both have maximum degree $\lceil D/2 \rceil$. This means that in general using repeated Euler splits to color the edges of G yields a $(2D-2)$-edge-coloring only. Every split of a (sub)graph with an odd maximum degree might require an additional color since the sum of the maximum degrees of the resulting subgraphs might exceed the maximum degree of the split graph by one.

We can take advantage of the above edge-coloring algorithm for graphs with maximum degree a power of two in the recursion of the divide and conquer approach as follows [GK82, CH]:

Tool 1. Assume that G is partitioned into two regular subgraphs G_1 and G_2. Let D_{G_1} and D_{G_2} denote the chromatic indices of G_1 and G_2, respectively. Suppose that $D_{G_1} \geq 2^{\lceil \log D_{G_2} \rceil} - D_{G_2}$. First, a minimal edge-coloring c of G_1 is found; then $2^{\lceil \log D_{G_2} \rceil} - D_{G_2}$ color classes of c are moved from G_1 to G_2, forming graph G_2'. The maximum degree of G_2' is a power of two and a minimal edge-coloring of G_2' can easily be computed using repeated Euler splits. Together with the remaining color classes of G_1 this gives a D-coloring of G.

4 Computing a Minimal Edge-Coloring

The basic approach of the algorithm in [CH] is to attempt to follow the Euler split method even when D is not a power of 2. When D is even, using an Euler split, G is partitioned into two graphs each of degree $D/2$. The first graph is colored recursively, the second using Tool 1. When D_G is odd, a regular matching M is computed; it is colored and removed from G. The graph $G - M$ has even degree and it is colored using the method for even D.

It is not hard to see that in the worst case as many as $\log D$ full matchings may be needed (for a variety of subgraphs of G). Let T_{match} be the cost of computing such a matching. Then the Cole-Hopcroft algorithm has running time $O(E \log D + \log D \cdot T_{match})$. The first term accounts for the Euler splits of G and the colorings using Tool 1. As $T_{match} = O(V \log V \log D)$ [Co], this gives a running time of $O(E \log D + V \log V \log^2 D)$. This section gives a new matching algorithm with $T_{match} = O(E)$.

Cole, Ost and Schirra [Ost] showed that only one matching, rather than $\log D$ matchings need be computed. In section 5, for completeness, we give a brief review of their method.

The basic approach used in [Sch] and [Co] is one of weight redistribution. Each edge is given a non-negative integer weight, initially one. The weights are redistributed so that the weight incident on each vertex remains unchanged at D and the edge weights remain integers in the range $[0, D]$. The goal is to obtain one edge of weight D incident on each vertex, i.e. a matching. The method used is to repeatedly find a cycle C in the graph, then to alternately increment and decrement the weight of the edges on the cycle by Δ so as to cause at least one edge to have weight 0 or D while keeping all weights in the range $[0, D]$. Any edges with weight 0 or D are removed from the graph. This process is continued until all edges have weight 0 or D. Using DFS, this process is readily implemented in time $0(L)$ for each length L cycle found.

The key observation in [Sch] is that by choosing the reweighing to go in the "right" direction, this takes time $O(E \cdot D)$ overall. Suppose an edge e_i has weight $wt(e_i) = w_i$. Edge e_i is given potential e_i^2. Suppose WLOG that on cycle C of length $2l$, $\sum_{i=1}^{l} w_{2i} \geq \sum_{i=1}^{l} w_{2i-1}$. Suppose the weight of the even index edges is incremented by one. Then the gain in potential is

$$\sum_{i=1}^{l} (w_{2i}+1)^2 - w_{2i}^2 + \sum_{i=1}^{l} (w_{2i-1}-1)^2 - w_{2i-1}^2 = \sum_{i=1}^{l} (2w_{2i}+1) - \sum_{i=1}^{l} (2w_{2i-1}-1) \geq 2l$$

Since the final potential is VD^2, the method runs in time $O(VD^2) = O(E \cdot D)$ time. We speed this up by a $\Theta(D)$ factor.

As in [CH] our algorithms start by reducing the number of edges to $O(V \log D)$ in $O(E)$ time. This is done by creating edge sets of weight $0, 2, \cdots, 2^i, \cdots$, for $2^i \leq D$, where all the edges sets, except for that of weight 0, have no cycles. To this end, the cycle finding and reweighting is applied to the weight one edges until no cycles remain amoung weight one edges; this takes time $O(E)$. In turn, the cycle finding is applied to the weight 2^i edges, for $2 \leq 2^i \leq D$, and this takes time $O(E/2^i)$.

For the remaining edge collection of size $O(V \log D)$ the goal is to process long cycles in sublinear time. To this end, edges are formed into paths, called *chains*, of length at most s edges, s a parameter to be specified. Chains have an implicit direction. The chains will be vertex disjoint. The algorithm will need to perform the following operations on chains, each in time $O(\log s)$.

- Concatenation.

- Split.
- Reverse.
- Find the total weight of the even index edges and of the odd index edges.
- Find a minimum weight edge of even (resp. odd) index and optionally delete it.
- Find a maximum weight edge of even (resp. odd) index and optionally delete it.
- Given a vertex, find the chain it belongs to, if any.

These operations are readily implemented using a balanced tree; indeed a splay tree, with just an amortized time bound, would suffice for our application.

A cycle will comprise an alternating sequence of chains and edges; further, all but at most two chains on the cycle will have length exactly s. To process and reweight the edges on a cycle, the cycle is traversed a chain at a time to find the weight of even index edges and the weight of odd index edges; the maximum and minimum weights of edges of odd and even index are also found (four weights in all). WLOG suppose the even index edges have greater weight; their weight is incremented maximally. Each chain is split at each edge now having weight 0 or D. Then the remainder of the cycle is removed from the path. The process of growing a cycle continues from the remainder of the path, which could be a single vertex.

A path is built by DFS. Whenever possible, the path will be extended by a chain rather than a single edge. If an edge is added which is incident on a vertex v in the interior of the chain, the chain is split at v and the larger portion is added to the path. The shorter portion edge with endpoint v is removed from the chain, and the remainder of this portion forms a new shorter chain. On the path being built, by means of concatentions and splits, the path is maintained as an alternation of chains of length s and singleton edges plus possibly a chain of length less than s at the end of the path.

Because of the vertex disjointness of chains, a cycle can be formed only by the addition of an edge and not a chain to the path. But this edge may be added to a chain of length less than s and may be incident on the interior of a chain. Thus a cycle may have up to two chains of length less than s, as claimed earlier.

To help detect a cycle, for each chain a bit is kept indicating if it is on the current path. Whenever an edge or chain is added to the path, the new final vertex on the path is tested to see if it belongs to a chain already on the path.

It remains to analyze the running time of this process. It is helpful to associate a potential of $\log s \log t$ with a chain of length t edges, which is not on the current path, and a potential of $\log s$ with each edge not on a chain or the current path. Then the cost of processing a length L cycle is:

$$O((L \log^2 s)/s + \log^2 s \cdot \text{number of edges removed})$$

The reason is that each chain resulting from processing the cycle needs a potential of $O(\log^2 s)$. There may be one chain for each edge removed, plus another $O(L/s)$ chains. Also, there may be this many singleton edges, which each need $O(\log s)$ potential.

The cost of building the path is $O(\log s)$ for each edge or portion of a chain that is appended. This cost is covered by the potential associated with the edge or by reducing by $\log s$ the potential associated with the chain (recall that the longer portion of a chain being split is added to the path; the shorter portion retains the remainder of the chain's potential, which suffices as it is of the form $\log s \log t$, where t is the chain edge length).

Thus the charge for removing all but $V/2$ edges (a matching) is $O(V \log D \log^2 s)$. Since $\sum L$ over all cycles is at most VD^2, the total cost due to the term $O((L \log^2 s)/s)$ is at most $O((VD^2 \log^2 s)/s)$. Choosing $s = D^2$ limits this term to $O(V \log^2 s)$. Then the total running time is $O(V \log^3 D) = O(E)$.

5 Coloring Using a Single Matching

The hard problem is to color a graph G for D an odd integer. The Cole, Ost and Schirra [Ost] algorithm proceeds essentially as follows. First, compute a full matching M; then perform an Euler split of $G - M$, yielding a regular decomposition into graphs G_2 and G_3; set $G_1 = M$. It remains to minimally color $G_1 \cup G_2 \cup G_3$.

The correct generalization is to the following problem: to minimally edge color $G_1 \cup G_2 \cup G_3$, where G_1, G_2, G_3 are regular graphs over the same vertex set, D_{G_1} is odd and $D_{G_1} \neq D_{G_2} = D_{G_3}$, or $D_{G_1} = D_{G_2} = D_{G_3} = 1$.

Case 0. $D_{G_1} = D_{G_2} = D_{G_3} = 1$. Simply color each graph with a distinct color.

Case 1. D_{G_2} is even.
Set aside G_3; then perform an Euler split of G_2, yielding a smaller instance of the same problem, and solve it recursively. Finally, color G_3 using Tool 1.

Case 2. D_{G_2} is odd.
$G_1 \cup G_2 \cup G_3$ is partitioned into regular graphs G_1', G_2', G_3', with $D_{G_2'} = D_{G_3'}$ being even. $G_1' \cup G_2' \cup G_3'$ is colored using the method for Case 1.

The partitioning is obtained as follows. Set $H_1 = G_1 \cup G_2$ and perform an Euler split of H_1, yielding regular graphs J_1 and J_2. Assign $\bar{G_1} \leftarrow G_3$, $\bar{G_2} \leftarrow J_1$ and $\bar{G_3} \leftarrow J_2$. This is a new instance of the same problem and is solved recursively.

To see the recursion terminates, we argue as follows. Notice that $D_{\bar{G_2}} = (D_{G_1} + D_{G_2})/2$; $D_{\bar{G_1}} = D_{G_2} = D_{G_3}$. Thus $|D_{\bar{G_1}} - D_{\bar{G_2}}| = |(D_{G_1} - D_{G_2})/2|$. Eventually, this difference becomes odd and Case 1 is reached.

Analysis. Notice that each time Case 1 is performed a graph G_3 is set aside. The time to color G_3 is $O(V \cdot D_{G_3} \log D_{G_3})$, which bounds the time for splitting G_2. Thus over the whole algorithm, as the various graphs G_3 are disjoint, Case 1 uses time $O(V \cdot D \log D) = O(E \log D)$.

It remains to analyze the time due to Case 2. Consider a maximal sequence of consecutive iterations of Case 2. The more iterations there are, the closer the final $D_{G_3'}$ is to the initial $(D_{G_1} + D_{G_2} + D_{G_3})/3$. More precisely if there were i iterations of Case 2, $|D_{G_1'} - D_{G_2'}| \leq (D_{G_1} + D_{G_2} + D_{G_3})/2^i$. As $i \geq 1$, it follows that $D_{G_2'} = D_{G_3'} \geq (D_{G_1} + D_{G_2} + D_{G_3})/6$ (for otherwise $D_{G_1'} \geq 2/3(D_{G_1} + D_{G_2} + D_{G_3})$, contradicting the observation in the previous sentence). Note that graph G_3' will be set aside by the immediately following instance of Case 1. Thus the next instance of Case 2, if any, will have size at most $5/6V(D_{G_1}+D_{G_2}+D_{G_3})$.

The cost of its maximal sequence of i iterations of Case 2 is $O(V(D_{G_1}+D_{G_2}+D_{G_3})\log(D_{G_1}+D_{G_2}+D_{G_3}))$. Over all instances of Case 2 this sums to $O(\log D(E+(5/6)E+(5/6)^2E+\cdots)=O(E\log D)$. Thus aside from the cost of computing one full matching, the algorithm was uses time $O(E\log D)$.

References

[CH] R. Cole, J. Hopcroft. *On Edge Coloring Bipartite Graphs.* SIAM J. Comput. 11 (1982), 540-546.

[Co] R. Cole. *Two Problems in Graph Theory.* Ph.D.-Thesis, Cornell University, August 1982.

[Ga] H. Gabow. *Using Euler Partitions to Edge Color Bipartite Multigraphs.* Internat. J. Comput. Inform. Sci., 5 (1976), 345-355.

[GK78] H. Gabow, O. Kariv. *Algorithms for Edge Coloring Bipartite Graphs.* Proc. 10^{th} Annual ACM Symp. on Theory of Computing (STOC), San Diego, CA., 1978, 184-192.

[GK82] H. Gabow, O. Kariv. *Algorithms for Edge Coloring Bipartite Graphs.* SIAM J. Comput. 11 (1982), 117-129.

[Go] C. Gotlieb. *The Construction of Class-Teachers Time-Tables.* Proc. IFIP Congress 62, Munich (North-Holland, Amsterdam, 1963), 73-77.

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

[KR] A. Kapoor, R. Rizzi. *Edge-Coloring Bipartite Graphs.* Manuscript, 1996.

[Ho] I. Holyer. *The NP-Completeness of Edge-Coloring.* SIAM J. on Computing, Vol. 10, No. 4, 1981, 718-720.

[KLM] R. Karp, M. Luby, F. Meyer auf der Heide. *Efficient PRAM Simulation on a Distributed Memory Machine.* Proc. 24^{th} Annual ACM Symp. on Theory of Computing (STOC), Victoria, B.C., Canada, 1992, 318-326.

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

[LPV] G. Lev, N. Pippenger, L.G. Valiant. *A Fast Parallel Algorithm for Routing in Permutation Networks.* IEEE Trans. Comput. C-30 (1981), 93-110.

[NZN] Sh. Nakano, X. Zhou, T. Nishizeki. Edge-Coloring Algorithms. in *Computer Science Today: Recent Trends and Developments*, Ed. Jan van Leeuwen, LNCS 1000, 1995.

[Ost] K. Ost. *Algorithmen für das kantenfärbungsproblem.* PhD Thesis, Universität des Saarlandes, January, 1995. (In German.)

[Sch] A. Schrijver. *Bipartite Edge Coloring in* $0(\Delta m)$ *Time.* SIAM J. on Computing, Vol. 28, No. 3, 1999, 841-846.

[Si] J. Sibeyn. *Edge Coloring Bipartite Graphs Efficiently.* Proc. Computing Science in the Netherlands, SION, 1992, 269-278.

[Vi] V. Vizing. *On an estimate of the chromatic class of a p-graph. (Russian)* Diskret. Analiz. 3 (1964), 25-30.