Spring 5-1-2000

4-Edge-Coloring Graphs of Maximum Degree 3 in Linear Time ; CU-CS-910-00

San Skulrattanakulchai
University of Colorado Boulder

Abstract

We present a linear time algorithm to properly color the edges of any graph of maximum degree 3 using 4 colors. Our algorithm uses a greedy approach and utilizes a new structure theorem for such graphs.

4-edge-coloring graphs of maximum degree 3 in linear time

San Skulrattanakulchai
Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309 USA

1 Introduction

Graph coloring is one of the most fertile and well-studied subjects in Graph Theory. An evidence of this fact may be found by browsing through the list of solved and unsolved problems in a comprehensive book [8] on graph coloring problems. The most general problem in the field is vertex coloring, since many coloring problems can be reduced to it. In the vertex coloring problem, we want to use the least number of colors to color the vertices of a graph, one color per vertex, in such a way that no two adjacent vertices are assigned the same color. This least number of colors is called the chromatic number χ of the graph. We are interested in a related edge coloring problem. In this problem, we want to use the least number of colors to color the edges of a graph, one color per edge, in such a way that no two adjacent edges are assigned the same color. This least number of colors is called the chromatic index χ' of the graph. A theorem of Vizing states that the chromatic index of a simple graph is at most one larger than its maximum degree Δ [15, 11]. Therefore the chromatic index of any simple graph is either Δ or $\Delta + 1$. Determining the true value proves to be an NP-complete problem, as shown by Holyer[7]. In fact, [7] shows that the problem remains NP-complete even when restricted to *cubic* graphs, those graphs whose every vertex is incident with exactly three edges. As observed in the survey paper [6] on cubic graphs, *cubic graphs often seem to be the simplest class of graphs for which a problem remains as difficult to solve as on a general graph.* By studying the problem when restricted to cubic graphs we may gain insight into why the problem is difficult. Actually, we will work with cubic graphs and more. The class of graphs of interest to us properly contains the class of cubic graphs.

We will be concerned with graphs of maximum degree 3 from now on. By [7], the problem of determining the chromatic index of any such graph is NP-complete. How about a polynomial time algorithm to edge-color them using 4 colors? The proof of Vizing's Theorem [3, 11, 1, 10] gives an $O(|V||E|)$ time algorithm to edge-color any arbitrary simple graph $G = (V, E)$ using $\Delta + 1$

colors. When specialized to graphs of maximum degree 3, we get an $O(|V|^2)$ algorithm to edge-color them using 4 colors. Now edge-coloring a graph G is equivalent to vertex-coloring its line graph $L(G)$. In the vertex coloring problem, Brooks' Theorem [2] states that a connected graph that is neither a complete graph nor an odd cycle has chromatic number no bigger than its maximum degree. Suppose $G = (V, E)$ has maximum degree 3. Then $L(G)$ has maximum degree at most 4 and it has at most $(3/2)|V|$ vertices. By Brooks' Theorem $L(G)$ can be vertex-colored with 4 colors. Thus G can be 4-edge-colored. The proof of Brooks' theorem when specialized to graphs of maximum degree 4 gives an $O(|V|)$ time algorithm for 4-vertex-coloring them. However, the algorithm resulting from the proof [1, 10] has several drawbacks. It is complicated; it requires computation of 2-connectivity [13, 4] and triconnected components [14]; and it requires extra housekeeping. In this paper we develop a direct $O(|V|)$ time algorithm to 4-edge-color any graph of maximum degree 3. Our algorithm is very simple. It uses depth-first-search [13, 4] and the greedy approach. Its simplicity rests on the fact that we may decompose any such graph into two distinct parts: a forest and a collection of vertex-disjoint cycles. It does not depend on the truth of either Brooks' or Vizing's Theorem. It works equally well on both simple graphs and graphs that have multiple edges (but no self-loops). Thus it algorithmically shows that any (multi)graph of maximum degree 3 can be 4-edge-colored. In an early paper [9] Johnson gives a proof that any cubic graph can be 4-edge-colored. The algorithm that results from his proof requires dynamic maintenance of 2-edge-connectivity under contraction of vertices and addition of edges. This latter problem is not known to have a linear time algorithm. Another proof that 4 colors suffice to properly color the edges of planar cubic graphs is given by Golovina & Yaglom [5, 12]. Their inductive proof can be turned into a linear time algorithm. The disadvantages of the resulting algorithm are that it is complex and requires recoloring.

The rest of this paper is organized as follows. Section 2 defines relevant terms to be used. Section 3 concerns the decomposition theorem. Section 4 describes the coloring algorithm. Section 5 concludes our paper. Appendix A gives detailed pseudocode for a decomposition algorithm. Appendix B gives detailed pseudocode for the coloring algorithm.

2 Terminology

Definition 1. Let $G = (V, E)$ be a finite graph with vertex set V and edge set E. We allow G to have multiple edges but no self-loops. We use n to denote $|V|$ and m to denote $|E|$. A graph $H = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A *cycle* is a connected graph whose every vertex has degree 2. Two graphs are *edge-disjoint* if they have no common edge; they are *vertex-disjoint* if they have no common vertex. If v is a vertex, we write $d(v)$ for its degree. A *forest* is a graph without cycles. A *tree* is a connected forest. A *node* is a vertex in a forest. A *leaf* is a node of degree 1. A cycle is *even* if it has an even number of edges. If k is a positive integer let $\{1, 2, \ldots, k\}$ be a set

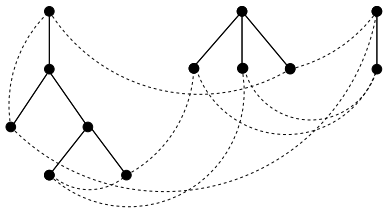

Figure 1: Example decomposition of a cubic graph into a forest and cycles

of colors. A *(proper) k-edge-coloring* of G is an assignment of some color from the color set $\{1, 2, \ldots, k\}$ to each edge of G in such a way that no two adjacent edges are assigned the same color.

Any graph-theoretic terms we use before defining is standard and its meaning can be found in [1, 11, 10].

3 Decomposition Theorem

Theorem 1. *Let G be a graph of maximum degree 3. Then G can be decomposed into 2 edge-disjoint subgraphs $\mathcal{C}$ and F, where $\mathcal{C}$ is a collection of vertex-disjoint cycles and F is a forest of maximum degree no bigger than 3.*

Proof. Suppose G contains a cycle C_1. Remove all edges of C_1 from G. If the resulting graph still contains cycles, keep removing them. Say we have removed the cycles $\mathcal{C} = \{C_1, C_2, \ldots, C_p\}$ until the resulting graph F contains no cycles (i.e., F is a forest). Any two distinct cycles C_i, C_j must be vertex-disjoint. To see this, suppose otherwise that v is a vertex on both C_i and C_j. Since C_i and C_j are edge-disjoint, v is incident with 2 edges of C_i and 2 edges of C_j. Thus $d(v) \geq 4$. This contradicts the assumption $d(v) \leq 3$. Being a subgraph of G, the maximum degree of F cannot exceed 3. □

Note that decomposition of G into forest and cycles is not unique. The theorem implies that the edges of G can be partitioned into two kinds: *cycle edges* and *tree edges*. Figure 1 shows a decomposed cubic graph. Dashed arcs are cycle edges; solid lines are tree edges. Appendix A describes a decomposition algorithm in detail.

4 Coloring Algorithm

Let $G = (V, E)$ be an input graph of maximum degree 3. Wlog assume G is connected. The coloring algorithm works in two stages. The first stage is

to decompose G into forest F and cycles $C_1, \ldots, C_p$. The actual coloring is done in the second stage. All the trees are colored first; the cycles are colored later. The coloring is done using a greedy approach. It systematically picks an edge and assigns the least available color from the color set $\{1, 2, 3, 4\}$ to that edge. For each tree T in F, it does a depth-first search starting from some arbitrary vertex in T. Tree edges are assigned colors in the order that they are discovered by the search. For each cycle C_i the algorithm tours C_i and assigns colors to the edges in the order that they are discovered, taking care to start the tour from an appropriate vertex and go around C_i in an appropriate direction. It will be convenient to define some more terms. A vertex v on C_i is *pure* if its degree in G is 2. A vertex v on C_i is *tainted* with c if the tree edge incident with v has been assigned color c. We also say that C_i is *d-tainted* if $|\{c \in \{1, 2, 3, 4\} : \text{some } v \text{ on } C_i \text{ is tainted with } c\}| = d$. The algorithm first determines if C_i contains any pure vertex. If a pure vertex w is found, let wv be a cycle edge. It starts the tour from v but away from w. If no vertex on C_i is pure, it counts the number of colors that C_i is tainted with. Three cases are distinguished whose proof will be given later.

Case 1: C_i is 1-tainted. It starts the tour from any vertex and can go around C_i in any direction.

Case 2: C_i is 2-tainted. It tries to find two consecutive vertices w, v that are both tainted with the same color.

Subcase 2.1: Such a pair of vertices exists. It tours C_i starting from v but away from w.

Subcase 2.2: Such a pair of vertices does not exist. It tours C_i starting from any vertex and can go around C_i in any direction.

Case 3: C_i is 3-tainted. It finds three consecutive vertices w, u, v no two of which are tainted with the same color. It then starts the tour from u and can go around C_i in any direction. See Appendix B for detailed pseudocode.

Example 1. Figure 2 shows how the algorithm works on non-pure cycles of each type. A number at each vertex is the color it is tainted with. A number on each edge is the color assigned by the algorithm. In the first three examples the algorithm tours the cycle starting with edge vx and ending with edge wv. So vx is the first and wv the last edge to be assigned color. In the fourth example the algorithm tours the cycle starting with edge vu and ending with edge wv. So vu is the first and wv the last edge to be assigned color.

The first cycle is tainted with one color 3. The second cycle is tainted with two colors $\{1, 2\}$, and it has consecutive vertices w, v both tainted with color 1. The third cycle is tainted with two colors $\{2, 3\}$ and every edge on it joins a vertex tainted with 2 to a vertex tainted with 3. The fourth cycle is tainted with three colors $\{1, 2, 3\}$. Vertices w, v and u are consecutive and no two of them are tainted with the same color.

Proof of Correctness. Depth-first search on the trees using greedy color choice on entrance works since each vertex of the tree has at most 3 incident edges, so our algorithm never gets stuck since it has 4 potential colors available. In fact, it never uses color 4. So the leaves are tainted with 1, 2, or 3 only.

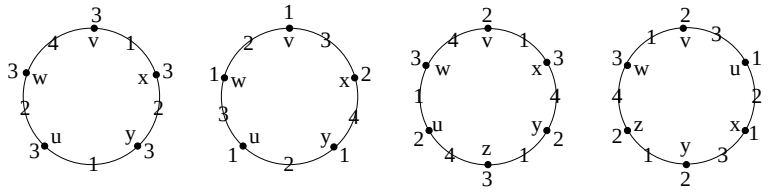

Figure 2: Examples showing how the algorithm works on each type of cycles

Note that the only time that greedy color choice on a cycle can fail is on the very last edge. So it suffices to show that when our algorithm is about to assign color to the last edge e in its tour of a cycle C_i, no more than 3 colors had been used to color the (at most) 4 edges adjacent to e. This clearly holds if C_i contains a pure vertex. Otherwise, consider each case separately.

Case 1: C_i is 1-tainted. Suppose wv is the last edge in the tour of C_i. Since w and v are tainted with the same color, when our algorithm is about to color edge wv, no more than 3 colors had been used to color the 4 edges adjacent to wv

Case 2: C_i is 2-tainted. First suppose C_i contains consecutive vertices w, v tainted with the same color. Edge wv is the last edge to be assigned color but at most 3 colors had been used to color the 4 edges adjacent to wv. Next suppose every 2 consecutive vertices of C_i are tainted with different colors. Then C_i is even; and thus the first edge and the penultimate edge in the tour of C_i had been assigned the same color. Therefore, exactly 3 colors had been used to color the 4 edges incident to wv.

Case 3: C_i is 3-tainted with $\{1, 2, 3\}$. Then we can find consecutive vertices w, v, u such that no two of them are tainted with the same color; for otherwise C_i would be at most 2-tainted. If our algorithm picks color c for the first edge vu, then c is the same color that w is tainted with! So when it is about to color the last edge wv, no more than 3 colors had been used to color the 4 edges adjacent to wv. □

5 Conclusion

We have presented a simple method to linearly 4-edge-color any graph of maximum degree 3. Our approach relies on a new decomposition theorem for such a graph. It is entertaining to ask whether this approach can be extended to attack similar problems.

Acknowledgments

I am indebted to Hal Gabow for general help, useful discussions, and for pointing me in the right direction. My thank also goes to Andrzej Ehrenfeucht for his encouragement and for introducing me to edge coloring.

6 Appendix

A Pseudocode for a Decomposition Algorithm

The proof of the decomposition theorem suggests the following high-level algorithm to decompose G into cycles $\mathcal{C}$ and forest F. This algorithm uses path-based depth-first search approach [4]. It maintains a graph F that is G with some cycles deleted. It also maintains a path P in F. Initially F is the given graph G and $\mathcal{C} = \emptyset$. It returns the collection $\mathcal{C}$ of disjoint cycles, and the remaining graph as forest F. Each cycle in $\mathcal{C}$ is a sequence of consecutive vertices on it. The high-level algorithm is as follows.

If all vertices of F have been discovered stop. Otherwise start a new path P by choosing a vertex v, marking v discovered, and setting $P = \langle v \rangle$. Continue growing P as follows.

To grow a path $P = \langle v_1, \ldots, v_k \rangle$ choose an edge $v_k w$ such that $w \neq v_{k-1}$ and do the followings:

- If w has not been discovered, mark w as discovered, add it to P, making it the last vertex of P. Continue growing P.
- If $w \in P$, say $w = v_i$, add a new cycle $\langle v_i, v_{i+1}, \ldots, v_k \rangle$ to $\mathcal{C}$. Delete every edge on this cycle from F, and delete every vertex on this cycle from P. If P is now nonempty continue growing P. Otherwise try to start a new path P.
- If every edge $v_k w \neq v_k v_{k-1}$ has $w \notin P$ and w already discovered, then v_k cannot be on any cycle in F. Delete v_k from P. If P is now nonempty continue growing P. Otherwise try to start a new path P.

To prove that the algorithm is correct, we need to show that each sequence of vertices added to $\mathcal{C}$ is indeed a cycle in G and that the remaining graph F is acyclic. We leave the proof to the reader.

Now we give an implementation that achieves linear time. Number the vertices of G by consecutive integers from 1 to n. We use two data structures, an array and an *array-stack*. An array-stack is one that can be accessed/updated either as a regular array or as a stack. Array-stack S is used to represent the path P. An array $I[1..n]$ is used to store stack indices. It is also used to indicate various status of a vertex. More precisely for a given vertex v at any

point in time,

$$I[v] = \begin{cases} 0 & \text{if } v \text{ has never been in } P; \\ n+1 & \text{if } v \text{ is known not to be on any cycle;} \\ n+1+p & \text{if } v \text{ is known to be a vertex on the cycle } C_p; \\ j & \text{if either } v \text{ is currently in } P \text{ and } S[j] = v; \text{ or } v \text{ is not} \\ & \text{currently in } P \text{ but } v \text{ is known to be on the same cycle} \\ & \text{as } S[j], S[j+1], \ldots, S[\text{TOP}(S)]; \end{cases}$$

We use the following operations to manipulate a stack S: PUSH(x, S) adds x to S at the new top of S. POP(S) removes the value at the top of the stack and returns that value. TOP(S) is the index of the value at the top of the stack. Hence $S[\text{TOP}(S)]$ is the value at the top of the stack.

A variable p is used to keep track of the cycle number. The algorithm consists of a main routine decompose() and a recursive procedure cycle().

```
procedure decompose(G) {
  empty stack S;
  for v ∈ V do I[v] ← 0;
  p ← 0;
  for v ∈ V do
    if I[v] = 0 then cycle(v);
}
procedure cycle(v) {
  PUSH(v, S); I[v] ←TOP(S); /* add v to end of P */
  for each edge vw do {
    if I[w] = 0 then {
      cycle(w);
      if I[w] < I[v] then { I[v] ← I[w]; I[w] ← n + 1 + p; POP(S); return; }
      else if I[w] = I[v] then { I[v] ← I[w] ← n + 1 + p; POP(S); return; }
    } else if I[w] < I[v] − 1 then {
      /* a new cycle is found */
      increase p by 1;
      C_p ← ⟨S[I[w]], S[I[w] + 1], ..., S[I[v] − 1], S[I[v]]⟩;
      I[v] ← I[w]; POP(S); return; } }
  I[v] ← n + 1; POP(S);
}
```

Proving that the above pseudocode correctly implements the high-level algorithm is again left to the reader. Our input graph has $m \leq (3/2)n$. Therefore, the above implementation takes $O(n)$ time since it spends $O(1)$ time on each vertex and each edge.

B Pseudocode for the Coloring Algorithm

We now give detailed pseudocode for the coloring algorithm greedy().

```
procedure greedy(G = (V, E)) {
  for each e ∈ E do
    color[e]← 0;
  decompose G into cycles {C_1, ..., C_p} and forest F = (V, E');
  for each v ∈ V do
    discovered[v] ← false;
  for each v ∈ V do
    if not discovered[v] then dfs(v);
  for i ← 1 to p do
    if C_i contains a pure vertex w then {
      let w, v, x be consecutive vertices on C_i;
      tour(v, x, C_i);
    } else if C_i is 1-tainted then {
      let v, x be consecutive vertices on C_i;
      tour(v, x, C_i);
    } else if C_i is 2-tainted then {
      let w, v be consecutive vertices on C_i that are both
        tainted with the same color if such a pair w,v exists,
        otherwise let w, v be any consecutive vertices on C_i;
      let x be such that vx is an edge on C_i and x ≠ w;
      tour(v, x, C_i);
    } else { /* C_i is 3-tainted */
      let w, v, u be consecutive vertices on C_i such that
        no two of them are tainted with the same color;
      tour(v, u, C_i);
    } }
procedure dfs(v) {
  discovered[v] ← true;
  for each edge vw ∈ E' do
    if not discovered[w] then {
      color[vw]← least_color(v, w);
      dfs(w); } }
procedure tour(v, w, C) {
  s ← v;
  color[vw] ← least_color(v, w);
  while w ≠ s do {
    x ← vertex on C adjacent to w but distinct from v;
    color[wx] ← least_color(w, x);
    v ← w; w ← x; } }
function least_color(v, w) {
  for c ← 1 to 4 do
    if c ≠ color[e] for all 4 edges e ∈ E adjacent to vw then
      return c; }
```

Our algorithm takes $O(n)$ time to run since it spends $O(1)$ time on each vertex and each edge.

References

[1] B. Bollobás, *Modern Graph Theory* (Springer-Verlag, New York, 1998).

[2] R.L. Brooks, On colouring the nodes of a network, *Proc. Cambridge Phil. Soc.* **37** (1941) 194–197.

[3] A. Ehrenfeucht, V. Faber and H.A. Kierstead, A new method of proving theorems on chromatic index, Los Alamos National Laboratory preprint LA-UR-82-661, (1982).

[4] H.N. Gabow, Path-based depth-first search for strong and bi-connected components, *Information Processing Letters* **74** (2000) 107–114.

[5] L.I. Golovina and I.M. Yaglom, *Induction in Geometry* (D.C. Heath, Boston, 1963).

[6] R. Greenlaw and R. Petreschi, Cubic graphs, *ACM Computing Surveys* **27, 4** (1995) 471–495.

[7] I.J. Holyer, The NP-completeness of edge-coloring, *SIAM J. Comp.* **10** (1981) 718–720.

[8] T.R. Jensen and B. Toft, *Graph Coloring Problems* (John Wiley & Sons, New York, 1995).

[9] E.L. Johnson, A proof of 4-coloring the edges of a cubic graph, *Amer. Math. Monthly* **73** (1966) 52–55.

[10] D. Jungnickel, *Graphs, Networks and Algorithms* (Springer-Verlag, Berlin Heidelberg, 1999).

[11] L. Lovász and M.D. Plummer, *Matching Theory* (North-Holland, Amsterdam, 1986).

[12] T.L. Saaty and P.C. Kainen, *The Four-Color Problem, Assaults and Conquest* (McGraw-Hill, 1977).

[13] R.E. Tarjan, Depth-first search and linear graph algorithms, *SIAM. J. Comp.* **1** (2) (1972) 146–160.

[14] J.E. Hopcroft and R.E. Tarjan, Dividing a graph into triconnected components, *SIAM. J. Comp.* **2** (3) (1973) 135–158.

[15] V.G. Vizing, On an estimate of the chromatic class of a p-graph (in Russian), *Metody Diskret. Analiz.* **29** (1964) 25–30.