Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 98 (2006) 101–106
Information Processing Letters
www.elsevier.com/locate/ipl

Δ-List vertex coloring in linear time ☆

San Skulrattanakulchai

Department of Mathematics and Computer Science, Gustavus Adolphus College, Saint Peter, MN 56082-1498, USA

Received 25 January 2005; received in revised form 23 November 2005; accepted 15 December 2005

Available online 17 January 2006

Communicated by F. Meyer auf der Heide

Abstract

We give a new proof of a theorem of Erdős, Rubin, and Taylor. Our proof yields the first linear time algorithm to Δ-list-color any graph containing no $(\Delta+1)$-clique, and containing no odd cycle if $\Delta=2$. Without change, our algorithm can also be used to Δ-color such graphs. It has the same resource bound as, but is simpler than, the current known algorithm of Lovász for Δ-coloring such graphs.

Keywords: Algorithms; Graph algorithms; Graph coloring; Vertex coloring; List coloring; Brooks' Theorem

1. Introduction

In the *list vertex coloring problem* [1], we are given a graph every vertex of which has been assigned its own list (set) of colors as part of the input. We are to assign to every vertex a color chosen from its color list so that adjacent vertices receive distinct colors. A graph is *list-colorable* if the task is possible. To *list-color* the graph is to find such a color assignment. The list vertex coloring problem is a generalization of the usual *vertex coloring problem* [1]. In the usual problem, we are given a graph and a constant number k of colors, and we are to assign to every vertex a color chosen from the given k colors so that adjacent vertices receive distinct colors. A graph is *k-colorable* if the task is possible. To find such a color assignment is to *k-color* the graph. A graph is *k-list-colorable* if it is list-colorable given any list assignment so long as the size of each list is at least k. To find such a color assignment, for a given set of k-lists, is to *k-list-color* the graph. The *chromatic number* of a graph is the least k for which it is k-colorable. The *list chromatic number* (or *choice number*) of a graph is the least k for which it is k-list-colorable. More generally, for any list size assignment $f : V \to \mathbb{N}$, we say that a graph is *f-choosable* if it is list-colorable given any list assignment so long as the list size of each vertex v is at least $f(v)$. We use d to refer to the degree function of a graph, i.e., for any vertex v, the *degree $d(v)$ of v* is the number of vertices adjacent to v. Thus a *d-choosable* graph is one that is list-colorable whenever every vertex's color list contains at least as many colors as its degree. For all k, *k-choosable* is synonymous to k-list-colorable. Observe that a k-choosable graph is always k-colorable since any algorithm for k-list-coloring can be used for k-coloring: simply assign the same list of k colors to every vertex of the input graph and run the k-list-coloring algorithm on it. Note also that a graph is list-colorable if and only if each of its connected

☆ A conference version of this paper appeared in *Proceedings of the Eighth Scandinavian Workshop on Algorithm Theory.*

E-mail address: sskulrat@gustavus.edu (S. Skulrattanakulchai).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2005.12.007

components is. Therefore, one needs only consider list-colorability of connected graphs.

A *Brooks graph* is a connected graph that is neither complete nor an odd cycle. Let Δ denote the maximum degree of a graph. Any complete graph or odd cycle is trivially $(\Delta+1)$-choosable but not Δ-choosable. Erdős et al. [2] prove that a connected graph is d-choosable if and only if at least one of its biconnected components is a Brooks graph. Consequently, they obtain the list version of Brooks' Theorem: *the list chromatic number* (*or choice number*) *of a Brooks graph does not exceed its maximum degree.* In other words, a Brooks graph is Δ-choosable. In their proof they check biconnectivity of the current graph at every inductive step following a sequence of vertex and edge deletions. Let m (n) be the number of edges (vertices) of an input Brooks graph. If we were to implement their proof in linear time, we would need an algorithm that is able to (1) perform vertex deletion, (2) perform edge deletion, and (3) answer biconnectivity query, all in O(1) amortized time per operation. We do not know of such an algorithm. We will give a new proof of the theorem of Erdős et al. and also derive the first algorithm to Δ-list-color any Brooks graph in $O(m+n)$ time. Both their and our proofs rely on existence of two special types of induced subgraphs in any biconnected Brooks graph. By observing that there is another type of graphs that is easy to find and that always contains one of these two types of unavoidable graphs as induced subgraphs, we are able obtain a simpler proof and a more efficient algorithm.

Our algorithm naturally specializes to solve the problem of finding a coloring in the original Brooks' Theorem [3]: *the chromatic number of a Brooks graph does not exceed its maximum degree.* In other words, a Brooks graph is Δ-colorable. The current known efficient algorithm for Δ-coloring Brooks graphs is due to Lovász [4]. (See also Problem 9.13, p. 67 in [5].) By using the results of Section 3, it is possible to implement Lovász's algorithm and make it run in $O(m+n)$ time. Lovász's algorithm computes a separating pair of vertices [6]. In contrast, our algorithm explores the graph in search of an occurrence of any one of a few simple unavoidable patterns, whose existence gives a straightforward coloring algorithm for the whole graph. Both Lovász's and our algorithms call upon a subroutine to compute biconnected components. This is fine since there exist many simple biconnected components algorithms in the literature. (See [7,8] for examples.) Again, our algorithm is simpler than Lovász's in this respect. All it needs from the biconnectivity computation is one biconnected component. In contrast, Lovász's algorithm has to construct the so-called block-cutpoint tree from all the biconnected components. Lovász's algorithm colors each biconnected component separately, then puts these colorings together by appropriately "permuting colors" in each component. Our algorithm never recolors.

1.1. Terminology

For any positive integer n, the set of all positive integers not greater than n is denoted $[n]$. We follow the terminology of [9] and consider only undirected graphs without parallel edges. By default, $G=(V,E)$ denotes the graph under consideration with $|V|=n$ and $|E|=m$. The *degree of a vertex* v is usually denoted $d(v)$, and also as $d_G(v)$ whenever G needs to be distinguished from some other graph also under consideration. The *minimum* (*maximum*) *degree* of the graph is denoted δ (Δ). A *regular* graph has $\delta=\Delta$. By a *path* we mean one with no repeated vertices. A cycle or path is *even* (*odd*) if it has an even (odd) number of edges. A *k-cycle* is a cycle on k vertices and is denoted $\langle v_1,\ldots,v_k\rangle$ if it is labeled. A *triangle* is a 3-cycle. A *wheel* W_k consists of a k-cycle and a *hub* vertex w that is not on the cycle but is adjacent to every vertex on the cycle. The cycle vertices are the *rim*; the k edges joining the hub w to the rim are called *spokes*. A *whel* is a wheel W_k with ℓ spokes missing, where $1\leqslant\ell<k-1$. In other words, a whel can be obtained from a wheel by removing at least one spoke while keeping at least two spokes. A *theta graph* $\theta(a,b,c)$ consists of 3 internally disjoint paths P_{a+1}, P_{b+1}, and P_{c+1} connecting two end vertices, where the path lengths a, b, and c satisfy $0<a\leqslant b\leqslant c$. A *diamond* is the complete graph K_4 with one edge missing. It is at the same time the smallest whel and the theta graph $\theta(1,2,2)$. Let H be a subgraph of G. A path in G *avoids* H if it uses no edge of H. A vertex w in G is called a *neighbor* of H if w is adjacent in G to some vertex of H but is itself not a vertex of H. To *contract edge* uv of a graph G means to delete vertices u and v from G, then add a new vertex x, and finally add an edge xw for every vertex $w\notin\{u,v\}$ such that either uw or vw was an edge in G. If H is a subgraph of G, then *contracting* H means contracting all edges of H. A *cutpoint* of a graph is a vertex whose removal results in a graph with more connected components. A *block* or *biconnected component* is a maximal connected subgraph without cutpoints. A *block-cutpoint graph* $BC(G)$ of a graph G is a bipartite graph in which one partite set consists of the cutpoints of G, and the other partite set has a vertex b_i for each block B_i of G. In $BC(G)$ vertex c is adjacent to vertex b_i if and only if in G vertex c is a cutpoint and block B_i contains c.

A block-cutpoint graph is always a tree. We always use the term "*induced*" to mean "induced by its vertex set". We use the term *available color* at various places in the description of our algorithm. During the execution of the algorithm, a *color c is available for a vertex v* if it is in the color list of v and no vertex adjacent to v is already colored c. We use the term *linear* to mean $O(m+n)$.

2. Structural properties

Lemma 1. *Every theta graph is either a $\theta(1,b,c)$ or contains an induced even cycle.*

Proof. Let $\theta(a,b,c)$ be a theta graph with paths P_{a+1}, P_{b+1} and P_{c+1} connecting the two end vertices. We are done if $a=1$. So suppose that $a>1$. Among the three integers a,b,c we can find two of them that are both odd or both even. Let us call these two integers having the same parity x and y. The graph $P_{x+1}\cup P_{y+1}$ is a cycle; and it has $x+y$, an even number of, vertices. Since $c\geqslant b\geqslant a>1$, this cycle is induced. □

Lemma 2. *Every whel W contains either an induced $\theta(1,b,c)$ or an induced even cycle.*

Proof. By Lemma 1, it is enough to show W contains an induced theta graph. Let W have hub w and k rim vertices. Let us say that if it is possible to label the rim vertices as $v_1,v_2,\ldots,v_k$ so that for some i and j we have wv_i and wv_j as spokes in W but no wv_ℓ is a spoke in W if $i<\ell<j$, then wv_i and wv_j are said to be *successive spokes*. There are three cases to consider.

Case 1: W contains exactly 2 spokes. Then W itself is a theta graph.

Case 2: W contains exactly 3 spokes. Since W is not a wheel, it contains successive spokes wv_i and wv_j such that $j-i>1$. The vertex-deletion graph $W-\{v_{i+1},v_{i+2},\ldots,v_{j-2},v_{j-1}\}$ is an induced theta graph.

Case 3: W contains at least 4 spokes. Let wv_i, wv_j, and wv_ℓ be three spokes in W such that wv_i and wv_j are successive, and wv_j and wv_ℓ are successive. Then vertices $w,v_i,v_{i+1},\ldots,v_j,v_{j+1},\ldots,v_\ell$ induce a theta graph. □

We now give our proof of a structure theorem that is due to Rubin [2]. The material in this proof is used heavily in Section 4 on algorithm.

Lemma 3. (Rubin.) *A biconnected Brooks graph contains either a $\theta(1,b,c)$ or an even cycle as an induced subgraph.*

Proof. Let G be a biconnected Brooks graph. By Lemmas 1 and 2 it suffices to show that G contains one of (i) an even cycle, (ii) a whel, or (iii) a theta graph, as an induced subgraph. Since G is biconnected, we see that $\delta\geqslant 2$ and G contains some cycle, and thus contains some induced cycle C. We are done if C is even. So from now on assume C is odd.

First suppose that C is a triangle. Let K be a maximal clique containing C. Clique K is necessarily a proper subgraph of G since G is not complete. Thus K has some neighbor. Say that $v_1,\ldots,v_k$ are the vertices of K. No neighbor w of K can be adjacent to all the v_i's since otherwise w and all the v_i's would form a clique containing K, contradicting maximality of K. Now if any neighbor w is adjacent to v_i, v_j but not to v_ℓ, where i,j,ℓ are all distinct, then we get an induced diamond on the vertices w,v_i,v_j,v_ℓ, and we are done. So assume that every neighbor is adjacent to exactly one vertex in K. Pick a vertex v_i adjacent to some neighbor of K. Since G is biconnected, there is a shortest path P avoiding K and connecting v_i to some other vertex v_j. Let ℓ be such that $1\leqslant\ell\leqslant k$ and $\ell\neq i,j$. Vertex v_ℓ together with the vertices on P induce a whel with v_ℓ as its hub.

Next suppose that $C=\langle v_1,\ldots,v_k\rangle$ is not a triangle. Cycle C cannot be the whole of G since G is not an odd cycle. So C has some neighbor. If some neighbor w of C is adjacent to every v_i, then $\{w,v_1,v_2,v_3\}$ induces a diamond, and we are done. If some neighbor w of C is adjacent to more than one but not all vertices of C, then w and the vertices of C induce a whel, and we are done. So assume that every neighbor of C is adjacent to exactly one vertex of C. Pick a vertex v_i adjacent to some neighbor of C. Since G is biconnected, there is a shortest path P avoiding C and connecting v_i to some other vertex v_j. The vertices of $C\cup P$ induce a theta graph. □

Lemma 4. *If G is a regular Brooks graph, then some of its blocks is a Brooks graph.*

Proof. Let H be a block of G that corresponds to a leaf (a vertex of degree 1) in the block-cutpoint tree of G. If $H=G$ then H is a Brooks graph by assumption. If $H\neq G$, let v be the only vertex in H that is also a cutpoint in G. Since v is in H and is also a cutpoint in the regular graph G, it follows that $d_H(v)<d_H(w)$ for any vertex w in H distinct from v. So H is not regular; and

is thus neither a clique nor an odd cycle. Therefore, H is a Brooks graph. □

3. Choosability properties

Lemma 5. *A connected graph G is d_G-choosable if it contains a connected, induced subgraph I that is d_I-choosable.*

Proof. Contract I to get a new graph G'. Let i be the vertex in G' that results from contracting I. Do a depth-first search [10] on G' starting from vertex i. Color all vertices of G' except i in order of increasing finish time, using any color available for each. This is possible since each vertex v (except i) has $d_G(v)$ colors in its list, $d_{G'}(v) \leqslant d_G(v)$, and v's parent is still uncolored at v's finish time; so some color is available for v.

Assign to all the vertices of G except those vertices of I the same colors assigned by the above algorithm. This partial coloring of G is clearly proper. Moreover, each vertex of I now has at least as many available colors as its degree (in I). So coloring of I can be completed since I is an induced subgraph of G and I is d_I-choosable. □

Lemma 6 appears in [11] and it implies Lemma 7.

Lemma 6. *If every vertex of a cycle C has a list of at least 2 colors, then C is list-colorable unless C is odd and every vertex has the same list of 2 colors.*

Lemma 7. *An even cycle is d-choosable.*

Lemma 8. *A theta graph is d-choosable.*

Proof. Let $\theta(a, b, c)$ be a theta graph with paths P_{a+1}, P_{b+1} and P_{c+1} connecting end vertices s and t. Let $P_{a+1} = (s, u_1, \ldots, u_{a-1}, t)$, $P_{b+1} = (s, v_1, \ldots, v_{b-1}, t)$, and $P_{c+1} = (s, w_1, \ldots, w_{c-1}, t)$. First color s by a color from its list that is different from any color in the list of v_1. This is possible since s has 3 colors in its list while v_1 has only 2. Then color $w_1, w_2 \ldots, w_{c-1}$, $u_1, u_2, \ldots, u_{a-1}, t, v_{b-1}, v_{b-2}, \ldots, v_1$ in that order, using any color available for each. It is easy to check that every vertex has some available color at the time the algorithm is about to color it. □

Lemma 9. *A whel is d-choosable.*

Proof. This follows from Lemmas 2, 5, 7, and 8. Alternatively, let w be the hub. First color w, but make sure it results in some rim vertex having more than 2 available colors or some rim vertices having different lists of available colors. This is possible since $d(w) \geqslant 2$ and some rim vertex is not adjacent to w. The rim vertices can then be colored using their available colors by Lemma 6. □

Lemma 10. *Let G be a connected graph that is not regular, and let v_1 be a vertex with $d(v_1) < \Delta$. Define a function $d' : V \to \mathbb{N}$ by $d'(v_1) = d(v_1) + 1$ and $d'(w) = d(w)$ for $w \neq v_1$. Then G is d'-choosable.*

Proof. For $j \leftarrow 2$ to n set v_j to be any vertex in $V \setminus \{v_1, \ldots, v_{j-1}\}$ that is adjacent to some vertex in $\{v_1, \ldots, v_{j-1}\}$. Color $v_n, \ldots, v_1$ in that order, using any color available for each. This procedure never fails since for $j = n, \ldots, 2$ vertex v_j is adjacent to some uncolored v_i, where $i < j$, and v_j has $d(v_j)$ colors in its list. Also, v_1 is colorable because v_1 has $d(v_1)$ colored neighbors but it has $d(v_1) + 1$ colors in its list. □

Theorem 11. (Erdős, Rubin, and Taylor.) *A Brooks graph is Δ-choosable.*

Proof. Let G be a Brooks graph. If G is not regular, then it is Δ-choosable by Lemma 10 and we are done. So assume that G is regular. By Lemma 4, some block H of G is a Brooks graph. By Lemma 3, H contains an induced subgraph I that is either a $\theta(1, b, c)$ or an even cycle. By Lemmas 8 and 7, I is d_I-choosable. Thus G is Δ-choosable by Lemma 5 since it is connected and contains the induced subgraph I that is d_I-choosable. □

4. Algorithm

We assume the following of our input. The input Brooks graph $G = (V, E)$ is represented as adjacency lists (see [10]). Each vertex has exactly Δ colors in its color list. (If not, we can throw away all colors in excess of Δ without affecting Δ-list-colorability of G.) Vertices, edges, and colors are numbered using consecutive integers starting from 1. So $V = [n]$, $E = [m]$, and the highest possible color is $n\Delta$. Each color list L_i of vertex i ($i \in [n]$) is represented as a linked list.

The proof of Theorem 11 suggests the following high-level algorithm for Δ-list-coloring G. If G is not regular, apply the algorithm in the proof of Lemma 10 and we are done. If G is regular, do the following. Execute some biconnected components algorithm [7,8] on G but stop the algorithm as soon as a biconnected component corresponding to a leaf in the block-cutpoint tree of G is identified. So we can stop the algorithm

of [8] or [7] as soon as it discovers the first biconnected component. Call this component H. Apply the algorithm in the proof of Lemma 3 to component H to extract an induced even cycle, or an induced whel, or an induced theta graph. Now apply the algorithms from Lemma 5, and Lemma 7 or 9 or 8, as appropriate.

The biconnected components algorithms [7,8] are known to take $O(m+n)$ time. Except for Lemmas 3, 5, and 10, the algorithms in the proofs of all other lemmas clearly take linear time. We now describe how to accomplish these two tasks.

Task A. To implement the algorithm in the proof of Lemma 3 so that it takes $O(m+n)$ time.

Task B. To implement the algorithms in the proof of Lemmas 10 and 5 so that they take $O(m+n)$ time.

Task A. To find an induced cycle, use a modified depth-first search (DFS) to explore the graph. Put in the language of the tree-based view of DFS [10], we make sure any back edge is explored before any tree edge. This can be done by going through the adjacency list of the currently active vertex v once. If any back edge is found and w is the vertex on the recursion stack closest and adjacent to v via a back edge, then all the vertices on the tree path from w to v induce a cycle. If no back edges are found, then continue exploring the graph in the depth-first manner.

Here is how to find a maximal clique containing a triangle. Use a boolean array to keep track of the vertices belonging to the current clique. Also, keep a count of the number of vertices in the current clique. Use another boolean array to keep track of processed vertices. Put all neighbors of the current clique in a queue. Repeat the following until the queue is empty. Delete a vertex v from the queue. If v has already been processed, continue checking the queue. Otherwise, process v as follows. If v is adjacent to all vertices in the current clique, add v to the current clique, and add all neighbors of v not belonging to the current clique to the queue, and mark v processed.

To find a shortest path avoiding H (which is either a maximal clique or an induced non-triangle odd cycle, every neighbor of which is adjacent to exactly one vertex of it) and connecting two vertices of H, use breadth-first search (BFS). (See [10].)

This implementation clearly takes $O(m+n)$ time.

Task B. We will describe an implementation of the algorithm in Lemma 10 only since the one in Lemma 5 is similar. To order the vertices as $v_1, \ldots, v_n$, use either DFS or BFS. Once the vertices are ordered, preprocess the color lists by chopping L_{v_1} so that it has exactly $d(v_1)+1$ colors; and for all $i \leftarrow 2$ to n, chop L_{v_i} so that it has exactly $d(v_i)$ colors. After the preprocessing, we have

$$\left|\bigcup_{i=1}^{n} L_i\right| \leqslant \sum_{i=1}^{n} |L_i| = 2m+1.$$

Let k be the highest color to ever appear in any chopped color list. For each vertex i, we will use a list U_i to keep track of colors made unavailable for i owing to their having been assigned to some of i's neighbors. Allocate a boolean array $A[1..k]$ of size k but leave it uninitialized. To find an available color for a vertex i, use the array $A[\cdot]$ together with list U_i. The coloring pseudocode is as follows.

```
for i ← 1 to n do empty list U_i;
for i ← n downto 1 do {
  for each color c in L_{v_i} do A[c] ← true;
  for each color c in U_{v_i} do A[c] ← false;
  c ← first color in L_{v_i};
  while A[c] = false do c ← next color in L_{v_i};
  assign color c to v_i;
  for each vertex j on the adjacency list of v_i do
    append color c to U_j }
```

It is not hard to see that this implementation is correct and takes $O(m+n)$ time. The foregoing discussion gives the following theorem.

Theorem 12. *There exists an algorithm for Δ-list-coloring any Brooks graph in $O(m+n)$ time.*

Acknowledgements

I am most grateful to Hal Gabow for his help.

References

[1] T.R. Jensen, B. Toft, Graph Coloring Problems, John Wiley & Sons, New York, 1995.
[2] P. Erdős, A.L. Rubin, H. Taylor, Choosability in graphs, in: Proc. West-Coast Conf. on Combinatorics, Graph Theory and Computing, Arcata, California, Congr. Numer. XXVI (1979) 125–157.
[3] R.L. Brooks, On coloring the nodes of a network, in: Proc. Cambridge Philosophical Society, Math. Phys. Sci. 37 (1941) 194–197.
[4] L. Lovász, Three short proofs in graph theory, J. Combin. Theory Ser. B 19 (1975) 269–271.
[5] L. Lovász, Combinatorial Problems and Exercises, second ed., North-Holland, Amsterdam, 1993.
[6] J.E. Hopcroft, R.E. Tarjan, Dividing a graph into triconnected components, SIAM J. Comput. 2 (3) (1973) 135–158.
[7] R.E. Tarjan, Depth-first search and linear graph algorithms, SIAM J Comput. 1 (2) (1972) 146–160.

[8] H.N. Gabow, Path-based depth-first search for strong and biconnected components, Inform. Process. Lett. 74 (2000) 107–114.
[9] B. Bollobás, Modern Graph Theory, Springer-Verlag, New York, 1998.
[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., McGraw-Hill, New York, 2001.
[11] H.N. Gabow, S. Skulrattanakulchai, Coloring algorithms on subcubic graphs, Internat. J. Found. Comput. Sci. 15 (1) (2004) 21–40.