A New Approximate Graph Coloring Algorithm

Avi Wigderson

Electrical Engineering & Computer Science Department
Princeton University
Princeton,NJ,08544

Abstract: Let A be a graph coloring algorithm. Denote by $\hat{A}(G)$ the ratio between the maximum number of colors A will use to color the graph G, and the chromatic number of G, $\chi(G)$. For most existing **polynomial** coloring algorithms, $\hat{A}(G)$ can be as bad as $O(n)$, where n is the number of vertices in G. The best currently known algorithm guarantees $\hat{A}(G)=O(n/\log n)$. In this paper we present a simple and efficient coloring algorithm which guarantees $\hat{A}(G)\leq\chi(G)n^{1-\frac{1}{\chi(G)-1}}$, a considerable improvement over the current bounds.

1. Introduction

A variety of problems in production scheduling, construction of timetables, etc. can be modeled as graph coloring problems. A (legal) r-coloring of a graph* G is an assignment of r colors to the vertices of G such that no two adjacent vertices are assigned the same color. A graph G that has an r-coloring is said to be r-colorable. The smallest integer r such that G has an r-coloring is called the chromatic number of G and is denoted by $\chi(G)$.

The graph coloring problem, i.e. "Given a graph G, is $\chi(G)=k$? " is known to be NP-complete, even if k is fixed, $k\geq 3$ [5,9]. Therefore it is unlikely that there is a polynomial time algorithm that will color every graph G with $\chi(G)$ colors. Moreover, Garey & Johnson [2] showed that a polynomial time algorithm that guarantees to color every graph G with at most $a\chi(G)+b$ colors, $a<2$, will imply a polynomial algorithm to color every graph G with $\chi(G)$ colors. In other words, getting close within a factor of two to the optimum is as hard as achieving it.

The inherent difficulty of the coloring problem caused researchers to devise efficient heuristic algorithms, hoping that the number of colors they use is near optimal. A few examples of this approach are [10,12,11,6,7]. In a 1976 paper [4], Johnson analyzed the worst case behaviour of such algorithms, and we follow his notation: for a coloring algorithm A, let $A(G)$ be the maximum number of colors A might use when applied to G, $\hat{A}(G)=A(G)/\chi(G)$ and $\hat{A}(n)=\text{Max}\{\hat{A}(G) \mid G \text{ has no more than } n \text{ vertices}\}$. For every algorithm in a list containing most of the known heuristics for coloring, Johnson constructs a sequence of 3-colorable graphs $\{G_m\}$ with $O(m)$ vertices, s.t. $A(G_m)\geq m$. This shows that the worst case behaviour of these algorithms is as bad as possible: $\hat{A}(n)\neq o(n)$, which is quite surprising in light of the intuitive nature of many of them. He concludes his paper with a polynomial time algorithm that guarantees $\hat{A}(G)=O(n/\log n)$**, the best bound currently known.

To summarize, there is a huge gap between what we know is NP-hard: $\hat{A}(G)<2$, and what we can guarantee in polynomial time: $\hat{A}(G)=O(n/\log n)$.

In the second section we give a polynomial time algorithm (Algorithm A) to color 3-colorable graphs on n vertices using at most $3\sqrt{n}$ colors. In the third section we show how to generalize the method to color k-colorable graphs (Algorithm B). In section 4 we give the final version of the algorithm (Algorithm C),

* Throughout the paper we deal only with simple undirected graphs.

** All logs are to the base 2.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1982 ACM 0-89791-067-2/82/005/0325 $00.75

which uses at most $\chi(G)n^{1-\frac{1}{\chi(G)-1}}$ colors. This is a considerable improvement on the upper bound mentioned above, for values of $\chi(G)$ which are $o(\log n / \log\log n)$. We further show that this algorithm is very practical; it can be implemented to run in linear time $(O(|V|+|E|))$ for graphs with a fixed chromatic number, and in time $O((|V|+|E|)\chi(G)\log\chi(G))$ for all graphs.

2. Algorithm A - coloring 3-colorable graphs.

First, let us introduce the graph theoretic notation we will use. For a graph $G(V,E)$ we define:

$N_G(v)$ = The *neighbourhood* of a vertex $v \in V$
$= \{u \mid (v,u) \in E\}$.

$d_G(v)$ = The *degree* of a vertex $v \in V$
$= |N_G(v)|$

$\Delta(G)$ = The *maximum degree* of G
$= \mathrm{Max}_{v \in V}\{d_G(V)\}$.

The subgraph of G *induced* by $U \subseteq V$ is the graph $H(U,F)$ where
$F = \{(u,w) \mid u \in U,\ w \in U \text{ and } (u,w) \in E\}$.

(We omit "G" from above notation when the graph at hand is clear from the context.)

We next state three simple facts on which our algorithm is based.

Fact 1: Any graph G can be colored in polynomial time with at most $1+\Delta(G)$ colors.

Fact 2: Let $G(V,E)$ be a 3-colorable graph. Then for every $v \in V$, the subgraph of G induced by $N(v)$ is bipartite (2-colorable).

Fact 3: Any bipartite graph can be 2-colored in polynomial time.

Algorithm A

Input: A 3-colorable graph $G(V,E)$.

1. $n \leftarrow |V|$
2. $i \leftarrow 1$.
3. while $\Delta(G) \geq \sqrt{n}$ do:
   let v be a vertex of maximum degree in G.
   $H \leftarrow$ the subgraph of G induced by $N_G(v)$.
   2-color H with colors $i, i+1$.
   color v with color $i+2$.
   $i \leftarrow i+2$.
   $G \leftarrow$ the subgraph of G resulting from it by deleting $N(v) \cup \{v\}$.
4. $(\Delta(G) < \sqrt{n})$. color G with colors $i, i+1, i+2, \cdots$ and halt.

Theorem 2.1: Algorithm A colors any 3-colorable graph $G(V,E)$ on n vertices with at most $3\sqrt{n}$ colors, and its running time is polynomial in n.

Proof: To see that the coloring is legal, we need only to observe that each time step 3 is executed, we use a new set of colors $\{i, i+1\}$ for $N(v)$, and all neighbours of v, which is colored $i+2$, were already colored with smaller colors.

Since each time step 3 is executed we color at least $\sqrt{n}+1$ vertices, it will be executed less than $\sqrt{n}$ times. Each time we use 2 colors, so altogether this loop uses less than $2\sqrt{n}$ colors. Step 4 is executed only once, and uses less than $\sqrt{n}+1$ colors, which gives the total bound of $3\sqrt{n}$.

The time bound follows immediately from facts 1 and 3. □

3. Algorithm B - coloring k-colorable graphs.

A natural question to ask at this point is: How can we use the ideas of the previous section to color k-colorable graphs for any k in polynomial time, and what upper bound on the number of colors can we guarantee? We look again at the three facts stated in section 2. Fact 1 is independent of the chromatic number of the graph*. Fact 2 is trivially generalized to:
Fact 2': Let $G(V,E)$ be a $(k+1)$-colorable graph. Then for every vertex $v \in V$, the subgraph of G induced by $N(v)$ is k-colorable.
Fact 3 was used in the following sense: given a polynomial time coloring algorithm for 2-colorable graphs, we obtained one for 3-colorable graphs. In general, we can recursively use the k-colorable graphs algorithm in the one for $(k+1)$-colorable graphs.

The above discussion suggests the following recursive algorithm B.

Algorithm B(k,G,i)

Input: An integer k, a k-colorable graph G, and an integer i, telling it to color G with successive colors $i, i+1, \cdots$.

* Although it gives an upper bound on the chromatic number.

Output: The number of colors used to color G.

1. If $k=2$, 2-color G with $i,i+1$ and return (2).
2. ($k>2$). $n \leftarrow$ the number of vertices in G.
3. [*Recursive coloring* stage]
   while $\Delta(G) \geq f_k(n)$ do:
   (The functions $f_k(n)$ are discussed below)
   let v be a vertex with $d_G(v)=\Delta(G)$.
   $H \leftarrow$ the subgraph of G induced by $N_G(v)$.
   $j \leftarrow B(k-1,H,i)$. (H was colored with $i,i+1,\cdots,i+j-1$).
   color v with color $i+j$.
   $i \leftarrow i+j$.
   $G \leftarrow$ the subgraph of G resulting from it by deleting $N_G(v) \cup \{v\}$.
4. [*Brute force coloring* stage]
   ($\Delta(G) < f_k(n)$). Color G with colors $i,i+1,\cdots,i+s-1$ and return (s). ($s < 1+f_k(n)$).

How do we choose the "break even point" $f_k(n)$ so as to minimize the number of colors we use? The concept of "balance" in algorithms suggests that we use about the same number of colors in each of the two stages of the algorithm. In algorithm A, this gave us the equation $f_3(n)=n/f_3(n)$, whose solution, $f_3(n)=\sqrt{n}$ was chosen to be the break even point. In general, the equation we get is:

$$f_{k+1}(n)=f_k(f_{k+1}(n))\frac{n}{f_{k+1}(n)} \qquad (1)$$

Lemma 3.1: The sequence of functions

$$f_k(n)=n^{1-\frac{1}{k-1}} \qquad (2)$$

for $k=2,3,\cdots$ is the solution of the recurence relation (1) with initial condition $f_2(n)=1$.

Proof: Simple induction on k. □

From this point on we identify the functions $f_k(n)$ mentioned in algorithm B with those defined in (2).

Theorem 3.1: Algorithm B colors any k-colorable graph on n vertices with at most $kf_k(n)=kn^{1-\frac{1}{k-1}}$ colors.

Proof: We use induction on k.
$k=2$. Coloring bipartite graphs with $2n^{1-\frac{1}{2-1}}=2$ colors.

$k>2$. Assume inductively that the claim is true for all k-colorable graphs, and that B colors a $(k+1)$-colorable graph on n vertices. Let m be the number of times step 3 was executed. If $m=0$, then G was colored with at most $f_{k+1}(n)+1 \leq (k+1)f_{k+1}(n)$ colors. If $m>0$, let $v_1,v_2,\cdots,v_m$ be the sequence of vertices chosen at each execution of step 3. Let H_i, $1 \leq i \leq m$, be the subgraphs induced by $N_G(v_i)$ in the current graph G, and t_i be the number of vertices in H_i. By induction, the H_i's were colored legally. Each was given a different set of colors, so there is no conflict in edges between different H_i's. Each v_i was assigned a bigger color than all its neighbours. Finally, step 4 was executed once, again with a different set of colors, which completes the proof that the coloring is legal. To prove the upper bound on the number of colors, we need the following:

Definition: Let S be a convex region. Then a function $f:S \rightarrow R$ is called *concave* if for every $x,y \in S$ and $0 \leq \lambda \leq 1$ we have $\lambda f(x)+(1-\lambda)f(y) \leq f(\lambda x+(1-\lambda)y)$

For concave functions one can prove the following (e.g. see [8]):

Jensen's Inequality: Let $f:S \rightarrow R$ be a concave function, and let $x_1,x_2,\cdots,x_m$ be points in S, then

$$\frac{\sum_{i=1}^{m} f(x_i)}{m} \leq f\left(\frac{\sum_{i=1}^{m} x_i}{m}\right)$$

Now we can continue the proof. It is easy to see that the functions $f_k(x)=x^{1-\frac{1}{k-1}}$, which are the functions in (2) extended to the non-negative real numbers, are concave. We also note the following: B uses no more than $kt_i^{1-\frac{1}{k-1}}$ colors on H_i, $\sum_{i=1}^{m} t_i \leq n$, and since for all i $t_i \geq n^{1-\frac{1}{k}}$, $m \leq n^{\frac{1}{k}}$. Combining all that information, the number of colors B uses in the recursive coloring stage is:

$$\sum_{i=1}^{m} kt_i^{1-\frac{1}{k-1}} = km\left(\frac{\sum_{i=1}^{m} t_i^{1-\frac{1}{k-1}}}{m}\right) \leq$$

$$\leq km\left(\frac{n}{m}\right)^{1-\frac{1}{k-1}} = k\left(nm^{\frac{1}{k-2}}\right)^{1-\frac{1}{k-1}} <$$

$$< k\left(n^{1+\frac{1}{k(k-2)}}\right)^{1-\frac{1}{k-1}} = kn^{1-\frac{1}{k}}$$

In the brute force coloring stage we use less than $1+n^{1-\frac{1}{k}}$ colors, which gives $(k+1)n^{1-\frac{1}{k}}$

as the bound on the total number of colors used. □

It is easy to show that algorithm B requires time polynomial in n. The next theorem states a stronger result, which shows that B is a practical algorithm.

Theorem 3.2: Algorithm B can be implemented to run in time $O(k(|V|+|E|))$ on k-colorable graphs $G(V,E)$.

Proof: The data structures we maintain in the implementation are given below.

The input graph is given in the data structure GRAPH. It has a doubly linked list of the vertices. Each vertex points to its adjacency list, which is also doubly linked. In addition, if $(u,w)\in E$, then w on u's list will point to u on w's list and vice versa. GRAPH allows us to delete each edge or vertex in constant time.

GRAPH can be preprocessed to construct the data structure DEGREE. Its purpose is to maintain the degree of each vertex so that we can update it in constant time with each removal of an edge, and have a constant time access to a vertex of maximum degree. Let $d_1\geqslant d_2\geqslant\cdots\geqslant d_p$ be the degree values occouring in the graph. For each d_i we keep a "bucket" D_i. These buckets are doubly linked in the above order, and we have a pointer to the first bucket, D_1. In each bucket D_i we keep all vertices of degree d_i, doubly linked in some order. Every D_i points to the first vertex in it, so we can tell when it gets empty. Each vertex in GRAPH will point to its place in the appropriate bucket in DEGREE. Clearly, we can create or delete a bucket in constant time. We can add or delete a vertex from a bucket in constant time. Also, DEGREE may be constructed from GRAPH in $O(|V|+|E|)$ time.

Finally, we keep an array COLOR of the vertices, global to all levels of the recursion.

Since we have $k-1$ levels in the recursion, to prove the time bound it is sufficient to show that each level takes time $O(|V|+|E|)$.

Assume that the input to the current level of recursion is a k-colorable graph $G(V,E)$, given in GRAPH(G). If $k=2$, it is known that G can be 2-colored in linear time. If $k>2$, we construct DEGREE(G). Let v be a vertex of maximum degree. If we are in the recursive coloring stage, we wish to construct GRAPH(H) for the graph H induced by $N_G(v)$, and update GRAPH(G) and DEGREE(G) to contain the new graph G resulting from the deletion of $N(v)\cup\{v\}$. To do that we first remove $N(v)$ from GRAPH(G), leaving these vertices linked to form GRAPH(H). Then we scan the adjacency list of each vertex $u\in N(v)$. For each vertex $w\notin N(v)$ in u's list we do the following: delete w from u's list, delete u from w's list, and decrement the degree of w by 1 in DEGREE(G). We than delete v and $N(v)$ from DEGREE(G). Note that we spend constant time on each edge and vertex before they are deleted from the graph. The brute force coloring stage can also be done in linear time using "sequential coloring" ([6]). Therefore, the time used in each level of the recursion is $O(|V|+|E|)$. □

4. Algorithm C - the final version

Until now we have assumed that the chromatic number of the input graph was also given as input. However, this is usually not the case. We overcome this problem by trying increasing values of k in algorithm B. We slightly change B to answer 'no' if it cannot find a legal coloring (i.e. the value of k was too small), and 'yes' if it can. This brings us to the final version of our algorithm - Algorithm C.

Algorithm C

Input: A graph $G(V,E)$.

1. Call $B(k,G,1)$ with $k=2^l$, $l=1,2,\cdots$ until the first l for which B answers 'yes'.
2. Using l of step 1, apply binary search to find the smallest k in $[2^{l-1},2^l]$ for which $B(k,G,1)$ answers 'yes'. Let this minimum value be k_0.
3. Color G with the coloring produced by $B(k_0,G,1)$.

Lemma 4.1: $k_0\leqslant\chi(G)$.

Proof: We need only observe that for every $k\geqslant\chi(G)$, $B(k,G,1)$ will answer 'yes' (Theorem 3.1). Since k_0 is the smallest k for which $B(k,G,1)$ answers 'yes', $k_0\leqslant\chi(G)$. □

Theorem 4.1: For any graph G on n vertices, the number of colors $C(G)$ that will be used by C, is at most $\chi(G)n^{1-\frac{1}{\chi(G)-1}}$.

Proof: Let k_0 be defined as above for G. By theorem 3.1, $C(G)\leqslant k_0 n^{1-\frac{1}{k_0-1}}$, since C uses

the coloring of $B(k_0,G,1)$. By lemma 4.1, $k_0 \leq \chi(G)$ which renders $C(G) \leq \chi(G) n^{1-\frac{1}{\chi(G)-1}}$. □

Corollary 4.1: For any graph G on n vertices, $\hat{C}(G) \leq n^{1-\frac{1}{\chi(G)-1}}$. In particular, if $\chi(G) = o(\log n / \log\log n)$, then $\hat{C}(G) = o(n/\log n)$.

Proof: The first statement is immediate from the definition of $\hat{C}(G)$. For the second, assume $\chi(G) = o(\log n / \log\log n)$. Then

$$\hat{C}(G) \leq n^{1-\frac{1}{\chi(G)-1}} < n^{1-\frac{1}{\chi(G)}} = \frac{n}{n^{\frac{1}{\chi(G)}}} = \frac{n}{2^{\frac{\log n}{\chi(G)}}} = o\left(\frac{n}{2^{\log\log n}}\right) = o\left(\frac{n}{\log n}\right)$$

Theorem 4.2: For every graph $G(V,E)$, the running time of algorithm C is $O((|V|+|E|)\chi(G)\log\chi(G))$.

Proof: The search method that is applied in C to find k_0 requires at most $2\log\chi(G)$ calls to algorithm B. By Theorem 3.2 and lemma 4.1, each call takes time $O((|V|+|E|)\chi(G))$, which proves the required bound. □

Corollary 4.2: For any fixed integer k, the running time of algorithm C on graphs G with $\chi(G) \leq k$ is linear, i.e. $O(|V|+|E|)$.

5. Further research

We conclude with three open problems:

1) Can the bound in Fact 1 be improved in polynomial time? A positive answer will immediately imply improvement to the upper bound on the number of colors our algorithm uses. It is known Δ colors are sufficient if the graph is neither a clique nor an odd cycle. Can we do with less than Δ colors whenever it is possible? In general this is unlikely since deciding if $\chi(G) \leq 3$ for graphs with $\Delta(G) \leq 4$ is NP-complete. However, it might be possible if some minimum difference or ratio between $\Delta(G)$ and $\chi(G)$ is guaranteed.

2) By lemma 4.1, algorithm C can be used to find a lower bound on the chromatic number of the input graph. How good is this bound?

3) This long standing conjecture is aimed to improve the lower bound on what can be done in polynomial time: Prove that to achieve $A(G) < r$ is NP-hard for every fixed positive number r.

Acknowledgements: I would like to thank Prof. R. Lipton for interesting me in this problem, and the National Science Foundation for supporting this work through grant MCS-8023806.

References

[1] Aho, A.V., Hopcroft, J.E. and Ullman, J.D. *The design and analysis of computer algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] Garey, M.R. and Johnson, D.S. "The complexity of near optimal graph coloring". *JACM* 23, pp. 43-49.

[3] Harary, F. *Graph Theory*. Reading, Mass., 1971.

[4] Johnson, D.S. "Worst case behaviour or graph coloring algorithms", *Proc. of the fifth South-Eastern Conference on Combinatorics, Graph Theory and Computing*, Utilitas Mathematica Publishing, Winnipeg, Canada, 1974 pp.513-528.

[5] Karp, R.M. "Reducability among combinatorial problems" in *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher, Eds. Plenum Press, New York, 1972 pp. 85-104.

[6] Matula, D.W., Marble, G. and Issacson, J.D. "Graph coloring algorithms" in R.C. Read (ed.) *Graph Theory and Computing*, Academic Press, New York, 1972.

[7] Matula, D.W. "Bounded color functions on graphs" *Networks* 2 (1972) pp. 29-44.

[8] Roberts, A.W. and Varberg, D.E. *Convex Analysys*, Academic press, New York and London, 1973. pp 211-216.

[9] Stockmeyer, L. "Planar 3-colorability is polynomial complete", *ACM SIGACT News* 5, 3(1973) pp. 19-25.

[10] Welsh, D.J.A. and Powel, M.B. "An upper bound to the chromatic number of a graph and its application to time tabling problems", *The Computer Journal*, 10 (1967) pp. 85-86.

[11] Williams, M.R. "The coloring of very large graphs" in *Combinatorial Structures and their Applications*, Gordon and Breach, New York (1970).