Improving the Performance Guarantee for Approximate Graph Coloring

AVI WIGDERSON

Princeton University, Princeton, New Jersey

Abstract. The performance guarantee of a graph coloring algorithm is the worst case ratio between the number of colors it uses on the input graph and the chromatic number of this graph. The previous best known polynomial-time algorithm had a performance guarantee $O(n/\log n)$ for graphs on n vertices. This result stood unchallenged for eight years. This paper presents an efficient algorithm with performance guarantee of $O(n(\log\log n)^2/(\log n)^2)$.

Categories and Subject Descriptors F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms, G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Approximation algorithms, graph coloring, NP-completeness, performance guarantee

1. *Introduction*

A variety of problems in production scheduling, construction of timetables, etc. can be modeled as graph coloring problems. A (proper) r-coloring of a graph[1] 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, namely, "Given a graph G, is $\chi(G) \leq k$?" 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 and 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 closer than within a factor of two to the optimum is as hard as achieving the optimum.

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 [6, 7, 10, 11, 13]. In a 1976 paper [4], Johnson analyzed the worst case behavior of such algorithms, and we follow his notation: For a coloring algorithm A, let $A(G)$ be the maximum number of colors A

[1] Throughout the paper we deal only with simple undirected graphs.

Author's present address: Computer Science Division, Electrical Engineering and Computer Science Department, University of California at Berkeley, Berkeley, CA 94720.

might use when applied to G, $A_o(G) = A(G)/\chi(G)$, and $A_o(n) = \max\{A_o(G) \mid G$ has no more than n vertices$\}$. $A_o(n)$ is called the *performance guarantee* of algorithm A. 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 such that $A(G_m) \geq m$. This shows that the worst case behavior of these algorithms is as bad as possible: $A_o(n) \neq o(n)$, 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 $A_o(n) = O(n/\log n)$, the best bound currently known.[2]

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

In Section 2 we give a polynomial-time algorithm (Algorithm A) to color 3-colorable graphs on n vertices using at most $3\sqrt{n}$ colors. In Section 3 we show how to generalize the method to color k-colorable graphs (Algorithm B). In Section 4 we give Algorithm C, which guarantees $C_o(G) \leq 2n^{1-1/(\chi(G)-1)}$. 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. In Section 5 we show how to combine Algorithm C with that of Johnson [4], so that the hybrid algorithm (Algorithm E) guarantees $E_o(n) = O(n(\log\log n)^2/(\log n)^2)$.

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

$$N_G(v) = \text{the } \textit{neighborhood} \text{ of a vertex } v \in V = \{u \mid (v, u) \in E\},$$
$$d_G(v) = \text{the } \textit{degree} \text{ of a vertex } v \in V = |N_G(v)|,$$
$$\Delta(G) = \text{the } \textit{maximum degree} \text{ of } G = \max_{v \in V}\{d_G(V)\}.$$

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 $\Delta(G) \geq \lceil\sqrt{n}\rceil$ 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. ($\Delta(G) < \lceil\sqrt{n}\rceil$). Color $G$ with colors $i, i + 1, i + 2, \ldots$ and halt.
```

[2] All logarithms are to the base 2 unless otherwise noted

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

PROOF. To see that the coloring is proper, 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 neighbors of v, which is colored $i + 2$, were already colored with smaller colors. Since each time step 3 is executed we color at least $\lceil\sqrt{n}\rceil + 1$ vertices, it will be executed at most $\lceil\sqrt{n}\rceil$ times. Each time we use two colors, so altogether this loop uses at most $2\lceil\sqrt{n}\rceil$ colors. Step 4 is executed only once and uses at most $\lceil\sqrt{n}\rceil$ colors, which gives the total bound of $3\lceil\sqrt{n}\rceil$. The time bound follows immediately from Facts 1 and 3. $\square$

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.[3] 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. For $k = 2, 3, \ldots$ define

$$f_k(n) = n^{1-1/(k-1)}. \tag{1}$$

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 $i, i + 1, \ldots$.

Output: The number of colors used to color G.

```
1. $n \leftarrow$ the number of vertices in $G$.
2. If $k = 2$, 2-color $G$ with $i, i + 1$ and return (2).
   If $k \geq \log n$, $n$-color $G$ with $i, i + 1, \ldots, i + n - 1$ (each vertex gets a distinct color) and
   return ($n$).
3. [Recursive coloring stage]
   While $\Delta(G) \geq \lceil f_k(n)\rceil$ do:
      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 $i, i + 1, \ldots, i + j - 1$).
      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]
   ($\Delta(G) < \lceil f_k(n)\rceil$). Color $G$ with colors $i, i + 1, \ldots, i + s - 1$ and return ($s$).
   ($s \leq \lceil f_k(n)\rceil$).
```

THEOREM 3.1. *Algorithm B colors any k-colorable graph on n vertices with at most $2k\lceil f_k(n)\rceil = 2k\lceil n^{1-1/(k-1)}\rceil$ colors.*

[3] Although it gives an upper bound on the chromatic number

PROOF. We use induction on k.

$k = 2$. Bipartite graphs are colored with $2 < 4n^{1-1/(2-1)} = 4$ colors.

$k \geq \log n$. The graph was colored with $n < 2kn^{1-1/(k-1)}$.

$2 < k < \log n$. 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 $\lceil f_{k+1}(n) \rceil \leq 2(k + 1)\lceil f_{k+1}(n) \rceil$ colors. If $m > 0$, let $v_1, v_2, \ldots, v_m$ 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 let t_i be the number of vertices in H_i. By induction, the H_i's were colored properly. 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 neighbors. Finally, step 4 was executed once, again with a different set of colors, which completes the proof that the coloring is proper. 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 $x_1, x_2, \ldots, x_m$ 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 $f_k(x) = x^{1-1/(k-1)}$, which are the functions in (1) extended to the nonnegative real numbers, are concave. We also note the following: B uses no more than $2k\lceil t_i^{1-1/(k-1)} \rceil$ colors on H_i, $\sum_{i=1}^m t_i \leq n$, and since for all i, $t_i \geq n^{1-1/k}$, $m < n^{1/k}$. Combining all that information, the number of colors B uses in the recursive coloring stage is

$$\begin{aligned}
\sum_{i=1}^{m} 2k\lceil t_i^{1-1/(k-1)} \rceil &< 2k\left(m + \sum_{i=1}^{m} t_i^{1-1/(k-1)}\right) \\
&= 2km\left(1 + \frac{\sum_{i=1}^m t_i^{1-1/(k-1)}}{m}\right) \\
&< (2k + 1)m\frac{\sum_{i=1}^m t_i^{1-1/(k-1)}}{m} && \text{(footnote 4)} \\
&\leq (2k + 1)m\left(\frac{n}{m}\right)^{1-1/(k-1)} && \text{(footnote 5)} \\
&= (2k + 1)(nm^{1/(k-2)})^{1-1/(k-1)} \\
&< (2k + 1)n^{1-1/k} \\
&\leq (2k + 1)\lceil n^{1-1/k} \rceil.
\end{aligned}$$

In the brute force coloring stage we use at most $\lceil n^{1-1/k} \rceil$ colors, which gives $2(k + 1)\lceil n^{1-1/k} \rceil$ as the bound on the total number of colors used. □

[4] This step is legal when $2k$ is smaller than the quantity in brackets, which is the case here as $k < \log n$.

[5] This step uses Jensen's Inequality.

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 for 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 $d_1 \geq d_2 \geq \cdots \geq d_p$ be the degree values occurring 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 $w \in V - N(v)$ 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 then 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*

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 Algorithm C.

Algorithm C

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

1. Call $B(k, G, 1)$ with $k = 2^l$, $l = 1, 2, \ldots$ 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. $k_0 \leq \chi(G)$.

PROOF. We need only observe that for every $k \geq \chi(G)$, $B(k, G, 1)$ will answer "yes" (Theorem 3.1). Since k_0 is the smallest k for which $B(k, G, 1)$ answers "yes," $k_0 \leq \chi(G)$. □

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* $2\chi(G)\lceil n^{1-1/(\chi(G)-1)}\rceil$.

PROOF. Let k_0 be defined as above for G. By Theorem 3.1, $C(G) \leq 2k_0\lceil n^{1-1/(k_0-1)}\rceil$, since C uses the coloring of $B(k_0, G, 1)$. By Lemma 4.1, $k_0 \leq \chi(G)$, which renders $C(G) \leq 2\chi(G)\lceil n^{1-1/(\chi(G)-1)}\rceil$. □

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 4.1. *For any fixed integer k, the running time of Algorithm C on graphs G with* $\chi(G) \leq k$ *is linear, that is,* $O(|V| + |E|)$.

5. *Improving the Performance Guarantee*

The performance guarantee of the algorithm in [4] is $O(n/\log n)$ for all graphs. It is easy to check that Algorithm C improves this bound only for graphs with $\chi(G) < \log n/\log\log n$. What can we do for graphs that do not satisfy this condition? Let us recall the Greedy Independent Set algorithm for coloring [4] (here called Algorithm D).

```
Algorithm D
Input: A graph G
1. $i \leftarrow 1$.
2. Let $W$ be the set of uncolored vertices. If $W = \emptyset$, halt, otherwise $U \leftarrow W$.
3. Let $u$ be a vertex of minimum degree in the subgraph induced by $U$. Color $u$ with $i$ and set
   $U \leftarrow U - \{u\} - N(u)$.
4. If $U = \emptyset$, set $i \leftarrow i + 1$ and goto 2. Otherwise goto 3.
```

Algorithm D can be implemented to run in time $O(|V|^2)$. The number of colors it uses is given in the following.

THEOREM 5.1. *Algorithm D colors any k-colorable graph* $G(V, E)$ *with at most* $3|V|/\log_k|V|$ *colors.*

This is slightly stronger than the claim in [4], but it is essential for our purposes. The proof below follows the one in [4].

PROOF. As G is k-colorable, it must contain an independent set of size $|V|/k$, and hence there must be a vertex of degree at most $|V|(k-1)/k$ in G. Therefore, if we start at step 3 with $|U| = n$, U will not be completely emptied before we colored at least $\lfloor\log_k n\rfloor$ vertices with the current color i. Consider now the size of W. We start with $|W| = |V|$. Divide the algorithm into two phases: before and after the first time

$|W|$ gets below $|V|/\log_k|V|$. Before this point we use each color for at least $\log_k|W| > 0.5\log_k|V|$ vertices, so in this phase at most $2|V|/\log_k|V|$ colors. After this point we cannot use more colors than vertices, so this phase requires at most $|V|/\log_k|V|$ colors, which completes the proof. □

Consider now the following simple (final) algorithm.

Algorithm E

Input: A graph G

1. Color G using Algorithm C.
2. Color G using Algorithm D.
3. Produce the one of the two colorings above that uses fewer colors.

THEOREM 5.2. *The performance guarantee of Algorithm E, $E_o(n)$, satisfies $E_o(n) \le 3n(\log\log n)^2/(\log n)^2$.*

PROOF. Let G be any k-colorable graph on n vertices. From Theorems 4.1 and 5.1, respectively, we get $C_o(G) \le 2\lceil n^{1-1/(k-1)}\rceil$ and $D_o(G) \le n/k\log_k n$. The only observation needed now is that if we fix n and consider C_o and D_o as functions of k ($1 \le k \le n$), then C_o is increasing while D_o is decreasing with k. Now, to prove the theorem, it is sufficient to exhibit one value of k (for each n) for which both $C_o(G)$, $D_o(G) \le 3n(\log\log n)^2/(\log n)^2$ hold for any k-colorable graph on n vertices. It is easy to verify that these inequalities hold for $k = \lceil \log n/2\log\log n\rceil$. □

ACKNOWLEDGMENTS. I wish to thank Prof. Lipton for introducing me to the problem. I also thank the two referees for their helpful comments.

REFERENCES

(Note. References [1, 12] are not cited in the text.)

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. *J. ACM 23*, 1 (Jan. 1976), 43–49.
3. HARARY, F. *Graph Theory.* Reading, Mass., 1971.
4. JOHNSON, D.S. Worst case behaviour or graph coloring algorithms. In *Proc. 5th South-Eastern Conf. 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 *Graph Theory and Computing,* R.C. Reed, Ed., Academic Press, New York, 1972, pp. 109–122.
7. MATULA, D.W. Bounded color functions on graphs. *Networks 2* (1972), 29–44.
8. ROBERTS, A.W., AND VARBERG, D.E. *Convex Analysys.* Academic Press, New York, London, 1973, pp. 211–216.
9. STOCKMEYER, L. Planar 3-colorability is polynomial complete. *ACM SIGACT News 5*, 3 (1973), 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. *Comput. J. 10* (1967), 85–86.
11. WILLIAMS, M.R. The coloring of very large graphs. In *Combinatorial Structures and their Applications,* R. Guy, ed., Gordon and Breach, New York, 1970.
12. WIGDERSON, A. A new approximate graph coloring algorithm. In *Proc. 14th ACM Symp. on Theory of Computing* (San Francisco, Calif., May 1982), ACM, New York, pp. 325–329.
13. WOOD, D.C. A technique for coloring a graph applicable to large scale time tabling problems. *Comput. J. 12* (1969), 317–319.

RECEIVED MARCH 1982; REVISED DECEMBER 1982; ACCEPTED DECEMBER 1982