An Algorithm for Coloring
the Nodes of a Graph

by

Robert A. Wagner
An Algorithm for Coloring the Nodes of a Graph

by

Robert A. Wagner*

Abstract: We study the problem of coloring the nodes of a graph such that two nodes joined by an arc are assigned different colors. An algorithm is presented which requires $\sim n^3$ time, where the graph contains $n$ nodes. This algorithm yields near-minimal colorings. The algorithm is based on the "coalescence" and "free coalescence" of nodes to yield simpler graphs. Based on these same operations, we derive an exhaustive search which examines at most $2^m$ cases, where $m$ arcs appear in the complement of the graph to be colored.

Key words and phrases: Graph coloring, graph node coloring, graph vertex coloring.

CR categories: 5.32

* Computer Science Department
Upson Hall
Cornell University
Ithaca, New York 14850
Introduction

The problem of deriving minimal colorings for the nodes of a graph is a classical problem of graph theory. Acceptable colorings assign integers (colors) to the nodes of a graph in such a way that no two nodes joined by an arc are colored alike. A minimal coloring for a graph is an acceptable coloring which requires fewest distinct colors.

Classically, graph coloring is an abstraction and generalization of a map-makers problem, where nodes of the graph correspond to regions of the map, and arcs connect nodes which correspond to countries which share a common border [6]. Recently, applications for this problem have appeared in the field of computer science. Graph coloring is required in minimizing the storage needed for scalar variables [1,2,3]. Graph coloring can also be useful in scheduling meetings to reduce total time while limiting conflicts [4,5]. These applications suggest that an efficient algorithm for coloring the nodes of a graph would be valuable.

The mathematical literature contains numerous algorithms for producing minimal colorings. For example, Berge [6] gives three such algorithms. However, these algorithms all have the unfortunate property that their time requirements grow exponentially or factorially with one of the size parameters of the graph (usually, the number of nodes). For even moderately large graphs (> 10 nodes or so), these algorithms are too time-consuming to be used.
This paper presents an algorithm for producing near-minimal colorings in time proportional to \( n^3 \), where the graph contains \( n \) nodes. In a sense, the colorings it produces are "locally" optimum. The algorithm can be generalized, by adding an exhaustive search mechanism, to produce "globally" minimum colorings. However, the generalization again seems to require time approximately proportional to \( 2^m \), in the worst case, where the complement of the graph to be colored contains \( m \) arcs.

Our purpose in presenting this approximate solution to the classical problem is primarily to spread hope among those people whose problems reduce to that of graph coloring. We suspect that many problems have been shelved on the grounds that no practical algorithm for graph coloring was available. Furthermore, we hope to stimulate efforts to find faster, or more nearly optimum, coloring algorithms.

We will first describe the algorithm for producing "locally-minimal" colorings. We will briefly analyze its time requirements, deriving an upper bound for this quantity. We will then add a heuristic rule, which increases the time required, while producing better colorings. Next, we will show that the heuristic really doesn't produce truly minimal colorings. Finally, we will discuss a generalization of the algorithm into one which produces minimal colorings.
Locally minimal colorings

Definitions:


A coloring of a graph $G$ is an assignment of integers to the nodes of $G$ such that the integers assigned to nodes $i$ and $j$ differ whenever $(i,j) \in A[G]$.

A minimal coloring of $G$ is a coloring of $G$ which requires the fewest number of distinct integers.

The complement of a graph $G$, written $G'$, is defined so that


Here $P(N)$ = the set of all unordered pairs drawn from $N$.

Thus, the graph $G'$ consists of the nodes of $G$, with an arc joining $i$ to $j$ in $G'$ just when no arc joins $i$ to $j$ in $G$.

Let $S(n,A) = \{ m | (n,m) \in A \}$

$S(n,A)$ is the set of all nodes $m$ which are joined to $n$ along an arc in $A$.

Let $R(n,A) = \{ (i,j) | (i,j) \in A \text{ and } (i=n \text{ or } j=n) \}$

$R(n,A)$ is the set of all arcs in $A$ which touch node $n$.

Let the set operation $\cdot$ be defined as:

$$A \cdot B \equiv A \cap \neg B.$$
The algorithm

The heart of the graph coloring algorithm is the notion of "coalescence" of two nodes. Given a graph $G$, certain pairs of nodes in $G$ may be colored alike. If we choose such a pair, $(i,j)$, we can derive a new graph, $H$, such that, for every valid coloring of $H$, there is a valid coloring for $G$ in which $i$ and $j$ are assigned the same color. Now consider the complement graphs $G'$ and $H'$. Since $H'$ has fewer arcs and nodes than $G'$, it is easier to color. The operation which derives $H'$ from $G'$ is termed coalescence. We define

$C_{ij}(G')$, the coalescence of arc $(i,j)$ of $G'$, as follows:

If $(i,j)\in A[G']$, then

$N[C_{ij}(G')] = N[G'] - \{j\}$


where $J(i,j,A) = \bigcup_{k \in S(i,A) \cap \neg S(j,A)} \{(i,k),(k,i)\}$

otherwise, let

$C_{ij}(G') = G'$.

$J(i,jA[G'])$ is the set of arcs of $A[G']$ which join $i$ to any node $m$ of $G'$ which isn't also joined to $j$ by an arc in $A[G']$.

Thus, in other words, $C_{ij}(G')$ is a graph identical to $G'$, as are all arcs joining $i$ and $k$, where $k$ is any node of $G'$ not joined to $j$ by an arc in $A[G']$. 
Coalescence is best visualized as a derivation of one complement graph from another.

Examples:

\[
\begin{array}{ccc}
G & G' & C_{12}(G') \\
\begin{graph}
1 & 2 \\
\end{graph} & \\
\begin{graph}
1 & 2 \\
3 & 4 \\
\end{graph} & \\
\begin{graph}
1 & 2 \\
3 & 4 \\
\end{graph} & \\
\begin{graph}
1 & 2 \\
3 & 4 \\
\end{graph} & \\
\begin{graph}
1 & 2 \\
3 & 4 \\
\end{graph} & \\
\begin{graph}
1 & 2 \\
3 & 4 \\
\end{graph} & \\
\begin{graph}
1 & 2 \\
3 & 4 \\
\end{graph} & \\
\begin{graph}
1 & 2 \\
3 & 4 \\
\end{graph} & \\
\begin{graph}
1 & 2 \\
3 & 4 \\
\end{graph} & \\
\end{array}
\]

Definition: If $Q(G)$ is a coloring for graph $G$, then let $Q(G,k)$ be the color $Q(G)$ assigns to $k$, for each $k \in \mathbb{N}[G]$.
Theorem 1

Let $Q(C_{ij}(G'))$ be a valid coloring for $C_{ij}(G')$

Then $P(G)$ is a valid coloring for $G$, where:

$$P(G,k) = Q(C_{ij}(G'),k) \text{ for all } k \in N[C_{ij}(G')]$$

and

$$P(G,j) = P(G,i) = Q(C_{ij}(G'),i) \text{ if } j \notin N[C_{ij}(G')] .$$

**proof:** If $C_{ij}(G') = G'$, there is nothing to prove.

Therefore, assume that $(i,j) \in A[G']$, and that $Q(C_{ij}(G'))$ is a valid coloring for $C_{ij}(G')$.

If $P(G)$ is not a valid coloring for $G$, it must be that two nodes $\ell$ and $m$ of $G$ are colored alike by $P(G)$, but $(\ell,m) \notin A[G]$.

**Case:** Suppose $\ell \neq j$ and $m \neq j$.

Then $\ell, m \in N[C_{ij}(G')]$ and $(\ell,m) \in A[G]$.

But then $(\ell,m) \notin A[G'] \supset A[C_{ij}(G')]$.

Hence $(\ell,m) \notin A[C_{ij}(G')]$.

$(\ell,m) \in A[C_{ij}(G')]$. Thus, $P(G)$ cannot color $\ell$ and $m$ alike since $Q(C_{ij}(G'))$ cannot.

**Case:** Suppose $\ell = j$.

Then, by assumption, $(i,j) \notin A[G]$, so that $m \neq i$. Suppose $(j,m) \in A[G]$. Then we will show that $(i,m) \notin A[C_{ij}(G')]$, and hence that $P(G,j) = P(G,i) = Q(C_{ij}(G'),i) \neq P(G,m)$. 

We have: \( m \in \neg S(j, A[G']) \), since \((j, m) \in A[G]\).

Therefore, if \( m \in S(i, A[G']) \),

\((i, m)\) and \((m, i) \in J(i, j, A[G'])\)

Thus \((i, m) \notin A[C_{ij}(G')]\), and

\((i, m) \in A(C'_{ij}(G')).\) QED
The local optimization algorithm

Using these definitions, we can proceed to describe the local-minimum coloring algorithm.

Step 1. Determine the maximal connected subgraphs of G. Step 1, which requires \( \sim n^2 \) operations, results in subgraphs \( G_1, \ldots, G_k \), where \( G_i \) and \( G_j \) have no nodes in common, and where no arc extends between a node of \( G_i \) and another node in \( G_j \) when \( i \neq j \). It should be easy to see that each maximal connected subgraph can be colored independently of all the others.

Step 2. Choose a maximal connected subgraph \( G_i \) of G.

Step 3. Choose an arc \((j,k) \in A[G'_i] = \text{the arcs of the complement graph of } G_i \).

Step 4. Replace \( G_i' \) with \( C_{jk}(G'_i) \).

Step 5. Repeat steps 3 and 4 until no arc remains in \( G_i' \). Then go on to step 6.

Step 6. Color each node in the final graph, \( H' \), with a distinct color.

Step 7. If \( H' \) was derived in step 3 as \( H' = C_{jk}(I') \), color all nodes of \( I' \) with the same colors they have in \( H' \). Color node \( k \) the same as node \( j \).
Step 8. Repeat step 7 until the original graph $G_i'$
is recovered and colored.

Step 9. Repeat steps 2-8 for each $G_i$.

This algorithm is incompletely specified. Step 3, in
particular, involves a choice, and criteria for this choice
has not been presented. Regardless of how the choice in
step 3 is made, the algorithm will yield a coloring which is
not necessarily minimal, but is difficult to improve by local
changes in the colors assigned to a few nodes.

"Free" coalescence

For all choices of $i$ and $j$, a valid coloring of $C_{ij}'(G')$
yields a valid coloring for $G$. However, for most choices of
$i$ and $j$, even a minimal coloring of $C_{ij}'(G')$ does not yield
a minimal coloring for $G$. For certain nodes $i$ and $j$,
minimal colorings of $C_{ij}'(G')$ do yield minimal colorings for $G$,
and such a coalescence operation will be termed "free."

**Theorem:** Suppose $S(j,A[G']) \cup \{j\} \supseteq S(i,A[G']) \cup \{i\}$
and $(i,j) \in A[G'].$

Then if $Q(C_{ij}'(G'))$ is a minimal coloring for
$C_{ij}'(G')$, $P(G)$ is a minimal coloring for $G$,
where $P(G,k) = Q(C_{ij}'(G'), k)$ for all $k \neq j$

$P(G,j) = P(G,i)$. 
proof:
Consider the graph $H'$ derived by deleting from $G'$ node $j$ and all arcs which touch node $j$.

$H' = C_{ij}(G')$, since

$$S(j,A[G']) \cup \{j\} \supseteq S(i,A[G']) \cup \{i\},$$

and $(i,j) \in A[G']$.

Therefore, $Q(C_{ij}'(G'))$ is a minimal coloring for $H$.

But $G$ consists of $H$, with another node, $j$, adjoined.

Adjoining an extra node to a graph can never reduce the number of colors that graph requires. Therefore, since $P(G)$ requires the same number of colors as $Q(H)$, $P(G)$ must be minimal. QED

Free coalescence can be exploited, by renumbering step 3 to be substep 3.3, and adding as sub-steps preceding substep 3.3:

3.1 Find $i, j$ and $k$ such that

$$(j, k) \in A[G'_i] \quad \text{and} \quad S(k,A[G'_i]) \cup \{k\} \supseteq S(j,A[G'_i]) \cup \{j\}.$$ 

Replace $G'_i$ by $C_{jk}(G'_i)$.

3.2 Repeat substep 3.1 until no such $j$ and $k$ exist in $G'_i$.

(Step 3 now comprises substeps 3.1, 3.2, and 3.3.)
In implementing this algorithm, it is convenient to represent graphs by their incidence matrices. The incidence matrix $M_G$ of graph $G$ is defined so that

$$M_G[i,j] = 1 \text{ if } (i,j) \in A[G]$$
$$= 0 \text{ else.}$$

By convention, $M_G[i,i] = 0$, and $M_G'[i,i] = 1$.

Then row $i$ of $M_G' = M_G'[i,*]$ corresponds to

$$S(i,A[G']) \cup \{i\},$$
where

$$M_G'[i,j] = 1 \text{ if and only if } j \in S(i,A[G']) \cup \{i\}.$$ 

Therefore, if $M_G'[i,j] = 1$ and if

$$M_G'[i,k] \ast M_G'[j,k] = M_G'[i,k] \text{ for all } k,$$

then $C_{ij}(G')$ is "free."

The operation $M_G'[i,k] \ast M_G'[j,k]$ can be performed on many computers in a single "AND" instruction, when $M_G'[i,*]$ is stored as a requestional string of bits. On the IBM 360, up to 256 nodes can be allowed before more than one instruction is needed.

**Plausible heuristics**

The algorithm given effectively creates a sequence of graphs by applying steps 3.1 and 4 repeatedly. The longer that sequence, the fewer colors needed for the original graph. To prove this statement, we observe that each coalescence step reduces the number of nodes in the graph by 1. Furthermore, if the final graph of the sequence contains $\ell$ nodes, $\ell$ colors
will be used in the coloring.

It seems plausible that the more arcs left in the graph, the longer the coalescence sequence which stems from that graph. Therefore, we can suggest a method for specifying the choice of arc in step 3.3:

Choose an arc \((j,k) \in A[G'_i]\) which satisfies

\[
|C_{jk}(G'_i)| = \max_{(\ell,m) \in G'_i} |C_{\ell m}(G'_i)|,
\]

where \(|G| = |A[G]| = \) the number of arcs of \(G\).

Equivalently, one can choose \((j,k)\) to minimize the number of arcs of \(G'_i\) deleted in forming \(C_{jk}(G'_i)\).

This heuristic is plausible, and indeed in many cases yields minimal colorings. However, there are graphs for which it does not yield such minimal colorings. Figure 1 shows the complement graph of an 8-node, 17-arc graph, \(G\). Arc \((1,2)\) of \(G'\) removes 3 arcs from \(G'\) if coalesced on; all other arcs of \(G'\) remove at least 4 arcs from \(G'\) when coalesced on. Nonetheless, if arc \((1,2)\) is chosen in step 3.3, the coloring which results requires 5 colors, while \(G\) can be colored in only 4 colors if arc \((1,2)\) is broken, rather than coalesced upon.

**Exhaustive search**

Instead of attempting a priori to pick one specific arc for coalescence in step 3.3, we can use backtrack programming \([7]\) to examine all possibilities. Several bases for exhaustive search exist.
One reasonably promising method involves the notion that every arc must either be broken, or be coalesced on by the time the algorithm terminates. Thus, if we choose any arc, and examine two derived graphs, one derived by coalescing on the arc, and the other by breaking the arc, all possibilities will be examined. Whichever derived graph requires fewest colors yields a minimal coloring for the original. Figure 2 gives a derivation-tree for the graph of figure 1. In figure 2, the chosen arc of the graph is circled, and arrows labeled B and C lead to graphs derived respectively by breaking and coalescing on the circled arc. Where free coalescences are possible, arcs are circled, and an arrow labelled F leads to the graph derived by free-coalescing on the circled arcs. The numbers associated with each derived graph give the minimum number of colors required for that graph.

An exhaustive search can take advantage of the coloring properties of non-connected graphs profitably. Our basic algorithm already exploits one such property, by dividing the original graph $G$ into disconnected subgraphs $G_i$ each of which is internally connected. Exhaustive search can make choices as to which $G_i$ to attack at each stage. One such choice strategy would employ most effort in coloring those $G_i$ which yield the largest number of colors. For example, one could employ the heuristic algorithm on each $G_i$, then use exhaustive search only on those $G_i$ whose heuristic coloring was largest. Suppose we determine
that \( C \) colors are needed for some \( G_i \) for a minimal coloring. Then, if some heuristic algorithm has demonstrated that another \( G_j \neq G_i \) needs no more than \( C \) colors, \( G_j \) need not be examined by exhaustive search at all.

Another connectedness property of graphs can be exploited to reduce the effort expended by an exhaustive search for a minimal coloring. We have defined the sub-graphs \( G_i \) to be connected subgraphs. However, their complements, \( G'_i \) need not be connected. Furthermore, as the algorithm proceeds deeper into the tree of possibilities, more arcs are deleted from \( G'_i \), thus increasing the chance that \( G'_i \) is not connected. We note that, if \( G'_i \) consists of the maximal connected subgraphs \( \{ G'_{ij} \} \), and if

\[
M(G) = \text{the minimal number of colors needed to color graph } G,
\]

then

\[
M(G_i) = \sum_j M(G'_{ij})
\]

(and \( M(G) = \max_i M(G_i) \)).

This suggests that each \( G'_{ij} \) be treated independently by the choice mechanism in the exhaustive search algorithm. In other words, the algorithm should avoid trying all possible colorings in \( G'_i \) in combination with each possible coloring of \( G_{i2} \).

Figures 3 and 4 illustrate the advantage of splitting subgraphs \( G'_i \) which are disconnected. In these figures, lines labelled \( S \) connect a graph to be split with drawings of its component subgraphs. The gains can be significantly greater,
if the component pieces individually require more effort to
color. The following brief derivation indicates the source of
this reduction of effort.

Let \( E(G) \) = the number of cases which must be
examined to compute the minimal coloring
for graph \( G \).

Suppose \( G_i' = \bigcup_j G_{ij} \) where no arc connects
\( G_{ij}' \) to \( G_{ik}' \) wherever \( j \neq k \).

Then the effort required to produce a minimal coloring
of \( G_i \) is:

\[
\sum_j E(G_{ij}) \quad \text{if } G_i \text{ is split into its component}
\text{ subgraphs, each of which is colored independently, and}
\]

\[
\sum_j \prod_{k=1}^{J-1} A(G_{ik}) \quad \text{if } G_i \text{ is not so split},
\]

where \( A(G_{ik}) = \) the number of leaves of the
search-tree for a coloring of \( G_{ik} \).

**Time required for coloring a connected graph \( G \)**

**A. Heuristic rule**

Suppose the heuristic rule discovers a coloring of \( d \) colors.

If \( G \) contained \( n \) nodes, \((N-d)\) coalescences took place. Each
derived graph must be scanned, to discover the arc to be coalesced
on next. If there are \( i \) nodes in the derived graph, this scan
requires \( \frac{i(i-1)}{2} \) operations. We have

\[
\text{Time (n)} \sim \sum_{i=d}^{n} \frac{i(i-1)}{2} < \frac{n^3}{6}
\]
B. **Exhaustive search**

Suppose the complement graph $G'$ contains $m$ arcs. Then at most

$$2^m$$

graphs need be examined.

In practice, it appears that significantly fewer graphs are actually examined, because coalescence normally deletes more than one arc, and because arc breaking often allows a sequence of free coalescences.
Acknowledgement:

We are indebted to Professor Robert W. Floyd of Carnegie-Mellon University, who suggested the operation we have termed "free coalescence."
REFERENCES


