01 Jan 1989

A Pruning Procedure for Exact Graph Coloring

Thomas J. Sager

Shi-Jen Lin

A PRUNING PROCEDURE FOR EXACT GRAPH COLORING

Thomas J. Sager and Shi-Jen Lin

CSc-89-3

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401 (314)341-4491

A Pruning Procedure for Exact Graph Coloring

Thomas J. Sager
Department of Computer Science
University of Missouri-Rolla
Rolla, Mo. 65401 USA
(314) 341-4856
tomsager@cs.umr.edu

and

Shi-Jen Lin
Department of Computer Science
Chung-Yung University
Chung-Li, Taiwan

Subject classification:

Analysis of algorithms: computational complexity. Computers/computer science: software. Programming: integer: algorithms: branch-and-bound.

Other keywords:

Graph-coloring, chromatic number, $\mathcal{NP}$-Complete, scheduling.

Abstract:

The graph coloring problem can be stated: *"Given an undirected graph, using a minimal number of colors, assign each vertex a color so that if two vertices are connected by an edge then they are not assigned the same color."* Graph coloring can be used to solve scheduling problems with constraints of the form: events e and e' can not be scheduled together. Graph coloring is an $\mathcal{NP}$-Complete problem. Generally large problems are solved heuristically, although some of the better heuristic algorithms use an exact graph coloring algorithm to finish coloring a graph after first reducing it heuristically to manageable size. Exact graph coloring is important both in its own right and as a component of heuristic coloring.

We present an improved exact graph coloring algorithm which runs in the mean over 20% faster than the *DSATUR* algorithm on some classes of random graphs. The improvement over *DSATUR* stems from the ability to prune the search tree by detecting in certain instances the existence of a complete subgraph of cardinality equal to the number of colors used in the best coloring found so far.

Introduction:

The graph coloring problem can be stated as: *Given an undirected graph, $G = (V, E)$, with no loops or multiedges, find a function $f : V \rightarrow 1..n$, for some positive integer n, such that if $(v, w) \in E$ then $f(v) \neq f(w)$.* Such a function, f is called a *coloring function.* If n is minimal over all of G's coloring functions, then f is called an *exact coloring function* and n is called the *chromatic number* and is written $\chi(G)$. An algorithm which, given a graph G, guarantees an output which is an exact coloring function is called an *exact graph coloring algorithm.* An algorithm whose output is a coloring function which is not necessarily exact is called a *heuristic graph coloring algorithm.*

Exact graph coloring is known to be $\mathcal{NP}$-Complete. In fact, it has been shown that heuristic graph coloring within a factor of 2 of the chromatic number is $\mathcal{NP}$-Complete [2]. Generally, because it can be quite time-consuming to find the chromatic number of large graphs, graphs of more than 60 or 70 vertices are colored with heuristic algorithms.

Graph coloring can be applied to solve scheduling problems with constraints of the form: events e and e' can not be scheduled together. One such problem is the examination scheduling problem: *"Find the minimum number of periods in which a set of examinations can be scheduled under the constraint that examinations v and w can not be scheduled in the same period if at least one person must sit for both exams."* Here V is the set of examinations and $(v, w) \in E$ iff $h(v) \bigcap h(w) \neq \emptyset$, where $h(v)$ is the set of people who will take examination v.

Exact graph coloring algorithms generally fall into four categories: vertex sequential, color sequential, dichotomous search, and integer linear programming. A study of exact graph coloring algorithms by Korman [5] found that vertex sequential exact algorithms usually give the best performance in practice and among the vertex sequential algorithms, those that use dynamic reordering of vertices appear to be superior. Similar results were

also found by Kubale and Jackowsky [6].

The *DSATUR* algorithm was originally presented by Brelaz [1] as a heuristic algorithm. A branch-and-bound version of *DSATUR*, which is a dynamic vertex sequential exact graph coloring algorithm, has come to represent a *de facto* standard among exact graph coloring algorithms. The branch-and-bound version, which we will henceforth refer to simply as *DSATUR* is shown in Figure 2. The operations *newcolor* and *merge* which are used by *DSATUR* are shown in Figure 1.

Certain heuristic algorithms use an exact algorithm to finish coloring a graph after first reducing it to manageable size (60 or 70 vertices) by heuristic techniques. Notable among this variety of heuristic graph coloring algorithm is the *XRLF* algorithm of Johnson et. al. [3] which uses a color sequential algorithm based on the work of Leighton [7] and Johri and Matula [4] to reduce a graph to manageable size and then uses *DSATUR* to finish the coloring. *XRLF* was found to outperform other known heuristic graph coloring algorithms on certain classes of graphs such as $G_{1000,0.5}$, the class of random graphs on 1000 vertices with each edge chosen independently with probability 0.5. Thus, an improved exact graph coloring algorithm can yield improved heuristic graph coloring algorithms as well.

In the following section, we present a scheme for improving exact vertex sequential algorithms. We apply this scheme to the *DSATUR* algorithm in two different ways. First, as a checking mechanism which enables us to prune the search tree more deeply. Second, as a method of reorganizing the colored portion of the graph. We find that as a pruning mechanism its use is advantageous on all but the most quickly colorable classes of graphs. As a mechanism for reorganizing the colored portion of the graph, it is still advantageous in most cases, but not always as advantageous as when used for pruning only. This is especially true of larger graphs.

1 A Scheme for Pruning and Reorganizing Colored Vertices

In the folowing discussion, we consider that the vertices of a graph are originally named $1, 2, 3, \ldots$. As colors are created, the colored vertices are named $-1, -2, -3, \ldots$. A *completely colored graph* contains only colored vertices. A *partially colored graph* may contain both colored and uncolored vertices. The uncolored vertices have names from the positive integers and the colored vertices have names from the negative integers. We let C be the set of colored vertices and W (white) be the set of uncolored vertices. $cadj(v)$ is the set of colored vertices adjacent to v and $cdegree(v)$ is the cardinality of $cadj(v)$. Similarly, $wadj(v)$ is the set of uncolored vertices adjacent to v and $wdegree(v)$ is the cardinality of $wadj(v)$.

A *partially colored graph* always has the following properties: first, there is never more than one vertex of a particular color and second, the set of colored vertices, $C = -k..-1$, always forms a complete subgraph.

As we color a graph, we *merge* pairs of non-adjacent vertices together until we arrive at a complete graph. In order to keep track of the vertices that have been *merged* together, we introduce the function, *vertices* form V' to $\mathcal{P}(V)$ where V' is the vertex set of a partially colored graph and $\mathcal{P}(V)$ is the power set of the vertex set of the original graph before beginning the coloring process.

Originally, *vertices(v)* $= \{v\}$, for all vertices, v. On *merging* vertex v into vertex c, we assign *vertices(c)* $\leftarrow$ *vertices(c)* $\bigcup$ *vertices(v)*. In this way, we keep track of the vertices of our original graph. Thus, if we color the graph $G = (V, E)$ ending with the completely colored graph $G' = (V', E')$, where $V' = -k..-1$, the function $f : V \rightarrow 1..k$, where $f(v) = j$ iff $v \in$ *vertices*$(-j)$ (in graph G'), is a coloring function of G.

In the following discussion, let $G = (V, E)$ be a partially colored graph with the set of colored vertices, $C = -k..-1$. Also let v and w be uncolored vertices of G (positive integers),

c be a colored vertex of G (negative integer) and x, y and z be vertices of G (integers).

Our scheme for improving vertex sequential exact coloring algorithms is based on the observation that if there exists v, w and c with $(v, w) \in E$ and $cadj(v) = cadj(w) = C \setminus \{c\}$ then G contains a complete subgraph of cardinality $\mid C \mid +1$, namely $C \setminus \{c\} \bigcup \{v, w\}$. Therefore, G is not colorable with fewer than $\mid C \mid +1$ colors. In this case we say that (v, w, c) is a *swappable triple.*

In Figure 1, we define four operations, *rename*, *newcolor*, *merge* and *swap* which transform partially colored graphs. The exact (branch-and-bound) version of Brelaz' *DSATUR* algorithm [1] is shown in Figure 2. Only the operations *merge* and *color* are used. In Figure 3, we present algorithm *DCHECK (DSATUR CHECK)* which checks for the existence of *swappable triples* only in order to prune the search tree.

In Figure 4, we present algorithm *DSWAP (DSATUR SWAP)* which, in addition to using *swappable triples* to prune the search tree, also uses them to reorganize the colored vertex set as a larger complete subgraph. In *DSWAP*, we check for *swappable triples* whenever there exists no vertex $v \in V$ with $cadj(v) = C$. If there exists a swappable triple, then we perform the operation *swap* on a *swappable triple* (v, w, c) which maximizes $wdegree(v) + wdegree(w) - wdegree(c)$. Otherwise, we apply the *DSATUR* algorithm.

In this manner, *DSWAP* tries to maximize first, the size of the colored vertex set, and second, the number of edges incident to a vertex in C. This seems compatible with the *DSATUR* algorithm which tends to do the same thing by choosing an uncolored vertex with first, a maximal *cdegree*, and second, a maximal *wdegree*.

The differences among the three algorithms are depicted in Figure 5. In Figure 5, we have 3 colored vertices numbered $-1, -2$ and -3 and three uncolored vertices numbered 1, 2 and 3, each of which is adjacent to two of the three colored vertices. *DSATUR* merges

vertex 3 into vertex -2 and may not notice the complete subgraph, $T = \{-3, -2, 1, 2\}$, until much later.

If the best coloring function found so far uses four colors then *DCHECK* will discover the complete subgraph, T, and prune the portion of the search tree rooted at the current node. However, if the best coloring function so far uses more than four colors *DCHECK* behaves like *DSATUR*.

If the best coloring function found so far uses more than four colors, *DSWAP* will reorganize the colored vertices, making T the colored vertex subset and renaming the vertices as shown in parentheses. However, if the best coloring function so far uses four colors then *DSWAP* behaves like *DCHECK* and prunes the search tree.

We note that all three algorithms indeed find an exact coloring function through exhaustive search. Each instantiation of the procedure, *color*, either increases the number of colored vertices (*newcolor* and *swap*) or decreases the number of uncolored vertices (*merge*). *color* calls itself recursively until a smaller coloring function is found or it ascertains that no coloring function that is smaller than the best found so far lies on this branch of the search tree.

2 Methodology

We programmed all three algorithms, *DSATUR*, *DCHECK* and *DSWAP*, in Turbo Pascal using a similar programming style and degree of optimization. For each of the vertex sizes: 32, 40, 48, 54 and 64, and for each of the edge densities: 0.1, 0.3, 0.5, 0.7 and 0.9, we generated 100 random graphs using Park and Miller's minimal standard random number generator [8]. We ran each of the three algorithms on a PC AT computer to produce an exact coloring function on each of the 100 random graphs for each density and vertex size except densities of 0.5 and 0.7 on 64 vertices. Because of the time involved in producing

an exact coloring for members of these two classes of random graphs, we generated only 30 random graphs of density 0.5 on 64 vertices and 10 random graphs of density 0.7 on 64 vertices. For these two classes of exceptions, all three algorithms were run on a PS2 model 80 computer, but the running times given in Figure 6 and Table 1 are adjusted for comparison to the running time of the AT. The running times of the three algorithms on each class of random graphs were also compared using the *paired t test.*

Table 1 gives the mean running times of each of the three algorithms and the results of the *paired t test* at the 95% confidence level. Mean running times are also shown graphically in Figure 6. Figure 7 shows the normalized mean running time with *DSATUR* equated to 1.0. Figure 8 shows the ratio of the mean running time of *DCHECK* to that of *DSATUR* and Figure 9 shows the mean chromatic number.

3 Results

We found that *DCHECK* is consistently faster than *DSATUR* at the 95% confidence level for all densities between 0.3 and 0.9 and all vertex sizes between 32 and 64. In particular, the relative improvement is greatest on the most time-consuming graphs to color, reaching its greatest value, 0.236, on random graphs of density 0.7 on 64 vertices. The differences in running time between the two algorithms are masked by the logarithmis scale in Figure 6, but are shown clearly in Figures 7 and 8. From Figure 8, we see that with the exception of two anomalous readings at densities 0.1 and 0.3 on graphs of 56 vertices, there is a steady decrease in the ratio of the mean running time of the two algorithms as the number of vertices increases for all 5 density classes. One might hypothesize that this decrease would continue on graphs of greater than 64 vertices, but this is a subject for further research.

On the other hand, *DSWAP* performed significantly faster than both *DSATUR* and *DCHECK* on graphs of densities between 0.3 and 0.9 on 32 and 40 vertices. *DSWAP*,

however, showed less improvement on larger graphs and its behavior became rather erratic with respect to the other two algorithms on graphs of 64 vertices. This situation can be seen graphically in Figure 7. We hypothesize that our method of reorganization of the colored portion of the graph in *DSWAP* is far from optimal and that there exist better schemes of reorganization that would outperform *DCHECK* on graphs of 48 through 64 vertices. This, however, is also a subject for further research.

In addition, our results show that exact graph coloring does indeed require exponential time in practice as well as theory; but mean chromatic number appears to be practically linear in the number of vertices for all 5 density classes tested. The former is shown graphically in Figure 6 whereas the latter is depicted in Figure 9.

References

[1] Brelaz, D.: New methods to color vertices of a graph. *Comm. ACM*, **22**, 4, Apr. 1979, pp251-256.

[2] Garey, M.R. and D.S. Johnson: The complexity of near-optimal graph coloring. *J. ACM*, **23**, 1, Jan. 1976, pp43-49.

[3] Johnson, D.S., C.R. Aragon, L.A. McGeoch and C. Schevon: Optimization by simulated annealing: an experimental evaluation, Part II (graph coloring and number partitioning). *Manuscript*, 1989.

[4] Johri, A. and D.W. Matula: Probabilistic bounds and heuristic algorithms for coloring large random graphs. *Tech. Rep. 82-CSE-6*, Southern Methodist University, Dallas, Tex., June 1982.

[5] Korman, S.M.: The graph coloring problem. In *Combinatorial Optimization*, Eds. N. Christofides et al., Wiley, New York, 1979, pp211-235.

[6] Kubale, M. and B. Jackowski: A general implicit enumeration algorithm for graph coloring. *Comm. ACM*, **28**, 4, April 1985, pp412-418.

[7] Leighton, F.T.: A graph coloring algorithm for large scheduling problems. *J. Res. Nat. Bur. Standards*, **84**, 6, Nov. 1979, pp489-506.

[8] Park, S.K. and Miller, K.W.: Random number generators: good ones are hard to find. *Comm. ACM*, **31**, 10, Oct. 1988, pp1192-1201.

DENSITY	VERTICES: 32		40		48		56		64	
0.1	$D\dagger$	1.86E-1	D	3.47E-1	$C\bullet$	5.11E-1	$S\star$	5.78E-1	C	1.06
	$C\dagger$	1.87E-1	C	3.48E-1	S	5.11E-1	$D\star$	5.88E-1	S	1.10
	S	1.93E-1	S	3.63E-1	D	5.22E-1	C	6.04E-1	D	1.14
0.3	$S\bullet\star$	3.89E-1	$S\bullet$	1.46	$C\bullet$	1.03E+1	$S\bullet$	3.11E+1	$S\bullet$	6.43E+2
	$C\bullet$	4.19E-1	$C\bullet$	1.56	$S\bullet$	1.03E+1	$C\bullet$	3.22E+1	$C\bullet$	7.14E+2
	D	4.42E-1	D	1.77	D	1.20E+1	D	3.91E+1	D	8.54E+2
0.5	$S\bullet\star$	1.28	$S\bullet$	1.02E+1	$C\bullet$	1.01E+2	$S\bullet$	6.34E+2	$C\bullet$	4.61E+3
	$C\bullet$	1.40	$C\bullet$	1.08E+1	$S\bullet$	1.04E+2	$C\bullet$	7.09E+2	D	5.80E+3
	D	1.60	D	1.31E+1	D	1.24E+2	D	8.81E+2	S	5.89E+3
0.7	$S\bullet$	1.64	$S\bullet$	9.83	$S\bullet$	1.42E+2	$C\bullet$	1.63E+3	$C\bullet$	1.72E+4
	$C\bullet$	1.88	$C\bullet$	1.01E+1	$C\bullet$	1.46E+2	S	1.83E+3	S	1.88E+4
	D	2.23	D	1.24E+1	D	1.84E+2	D	2.09E+3	D	2.21E+4
0.9	$S\bullet$	3.86E-1	$S\bullet$	9.63E-1	$S\bullet\star$	5.71	S	7.00E+1	$C\bullet$	4.91E+2
	$C\bullet$	3.89E-1	$C\bullet$	1.01	$C\bullet$	6.39	$C\bullet$	7.24E+1	D	5.88E+2
	D	4.12E-1	D	1.15	D	7.59	D	8.65E+1	S	6.57E+2

D *DSATUR* algorithm.
C *DCHECK* algorithm.
S *DSWAP* algorithm.

$\bullet$ faster than *DSATUR* algorithm at 95% confidence level.
$\star$ faster than *DCHECK* algorithm at 95% confidence level.
$\dagger$ faster than *DSWAP* algorithm at 95% confidence level.

$y\mathrm{E}x$ means $y * 10^x$.

Table 1. Mean Running Time in Seconds

In the procedures below:

- $G = (V, E)$ is a partially colored graph with the set of uncolored vertices, W, numbered from the positive integers and the set of colored vertices, $C = -n.. -1$, for some non-positive integer n.
- *vertices* is a function from V to the positive integers. $vertices(v)$ is the set of vertices of the original graph that have been merged into v.
- v and $w \in W$, $c \in C$, and x, y and $z \in V$.

Four graph transforming operations.

- **procedure** $rename(G, vertices, y, z)$;
  Assert $y \in V$ and $z \notin V$.
  $vertices(z) \leftarrow vertices(y)$
  $vertices(y) \leftarrow undefined$
  $E \leftarrow E \bigcup \{(z, x) \mid (y, x) \in E\} \setminus \{(y, x) \mid (y, x) \in E\}$
  $V \rightarrow V \bigcup \{z\} \setminus \{y\}$
- **procedure** $newcolor(G, vertices, v)$;
  $rename(G, vertices, v, -(n+1))$; thereby creating a new colored vertex.
  $E \leftarrow E \bigcup \{(c, -(n+1) \mid c \in -n.. -1\}$, thereby ensuring that C is still a complete subgraph.
- **procedure** $merge(G, vertices, v, c)$;
  $vertices(c) \leftarrow vertices(c) \bigcup vertices(v)$
  $vertices(v) \leftarrow undefined$
  $E \leftarrow E \bigcup \{(c, w) \mid (v, w) \in E\} \setminus \{(v, z) \mid z \in V\}$
  $V \leftarrow V - \{v\}$
- **procedure** $swap(G, vertices, v, w, c)$;
  Let $x \notin V$
  $rename(G, vertices, v, x)$
  $rename(G, vertices, c, v)$
  $rename(G, vertices, x, c)$
  $newcolor(G, vertices, w)$

Figure 1. Four Operations on Partially Colored Graphs

```
algorithm DSATUR;
  input:    G = (V,E): graph;
  output:   χ(G): positive integer;
            exactcf: function: V → 1..χ(G);


  procedure color( G = (V, E): a partially colored graph;
                   vertices: function: V → P(1..∞));
    if G is completely colored          {in which case V is a set of negative integers}
      if | V |< ncolors
        ncolors ←| V |;
        ∀j ∈ V,     ∀i ∈ vertices(j),
          exactcf(i) ← −j;
    else
      choose an uncolored vertex, v, from V such that cdegree(v) is maximal and
          wdegree(v) is maximal among those uncolored vertices with maximal cdegree;
      feasibleset ← {c ∈ V | c is colored and c ∉ cadj(v)};
      ∀c ∈ feasibleset,
        if | C |< ncolors          {C is the colored vertex subset}
          G' ← G;
          vertices' ← vertices;
          merge(G', vertices', v, c);
          color(G', vertices');
      if| C |< ncolors − 1
        newcolor(G, vertices, v);
        color(G, vertices);


ncolors ← ∞;
∀v ∈ V, vertices(v) ← {v};
color(G, vertices);
χ(G) ← ncolors
```

Figure 2. DSATUR Algorithm

```
algorithm DCHECK;
  input:    G = (V,E): graph;
  output:   χ(G): positive integer;
            exactcf: function: V → 1..χ(G);

  procedure color( G = (V, E): a partially colored graph;
                   vertices: function: V → P(1..∞));
    if G is completely colored          {in which case V is a set of negative integers}
      if | V |< ncolors
        ncolors ←| V |;
        ∀j ∈ V,     ∀i ∈ vertices(j),
          exactcf(i) ← −j;
    else
      if | C |= ncolors − 1             {C is the colored vertex subset}
        if ∃v ∈ V, an uncolored vertex adjacent to all colored vertices
          return
        else if ∃(v, w, c), a swappable triple
          return
        choose an uncolored vertex, v, from V such that cdegree(v) is maximal and
            wdegree(v) is maximal among those uncolored vertices with maximal cdegree;
        feasibleset ← {c ∈ V | c is colored and c ∉ cadj(v)};
        ∀c ∈ feasibleset,
          if | C |< ncolors
            G' ← G;
            vertices' ← vertices;
            merge(G', vertices', v, c);
            color(G', vertices');
        if| C |< ncolors − 1
          newcolor(G, vertices, v);
          color(G, vertices);

ncolors ← ∞;
∀v ∈ V, vertices(v) ← {v};
color(G, vertices);
χ(G) ← ncolors;
```

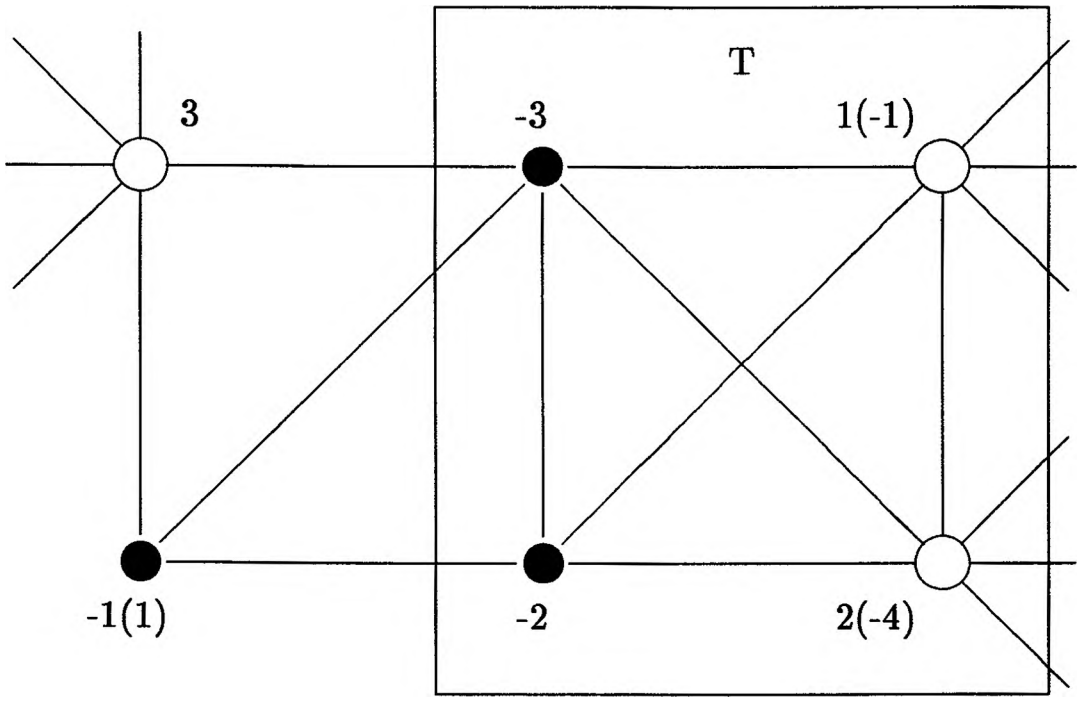
Figure 3. DCHECK Algorithm

```
algorithm DSWAP;
  input:    G = (V,E): graph;
  output:   χ(G): positive integer;
            exactcf: function: V → 1..χ(G);

  procedure color( G = (V,E): a partially colored graph;
                   vertices: function: V → P(1..∞));
    if G is completely colored          {in which case V is a set of negative integers}
      if | V |< ncolors
        ncolors ←| V |;
        ∀j ∈ V,    ∀i ∈ vertices(j),    exactcf(i) ← −j;
    else
      if ∃v ∈ V, an uncolored vertex adjacent to all colored vertices
        if | C |< ncolors − 1           {C is the colored vertex subset}
          choose an uncolored vertex, v ∈ V, adjacent to all colored vertices
               whose wdegree(v) is maximal among those uncolored vertices
               adjacent to all colored vertices;
          newcolor(G, vertices, v);                 color(G, vertices);
      else if ∃(v, w, c), a swappable triple
        if| C |< ncolors − 1
          choose a swappable triple (v,w,c) so as to maximize
               wdegree(v) + wdegree(w) − wdegree(c)
          swap(G, vertices, v, w, c);               color(G, vertices);
      else
        choose an uncolored vertex, v, from V such that cdegree(v) is maximal and
             wdegree(v) is maximal among those uncolored vertices with maximal cdegree;
        feasibleset ← {c ∈ V | c is colored and c ∉ cadj(v)};
        ∀c ∈ feasibleset,
          if | C |< ncolors
            G' ← G;                                 vertices' ← vertices;
            merge(G', vertices', v, c);             color(G', vertices');
        if| C |< ncolors − 1
          newcolor(G, vertices, v);

ncolors ← ∞;                        ∀v ∈ V, vertices(v) ← {v};
COLOR(G, vertices);                 χ(G) ← ncolors;
```

Figure 4. DSWAP Algorithm



●	Colored vertex.
o	Uncolored vertex.
x(y)	means *swap* will rename vertex x with y.
DSATUR	merges vertex 3 into vertex -2, note: $wdegree(3) > max(wdegree(1), wdegree(2))$.
DCHECK	merges vertex 3 into vertex -2 if $ncolors > 4$, prunes search tree if $ncolors = 4$.
DSWAP	reorganizes colored vertex set as T if $ncolors > 4$, prunes search tree if $ncolors = 4$.
ncolors	number of colors for best coloring found so far.

Figure 5. Behavior of Three Algorithms

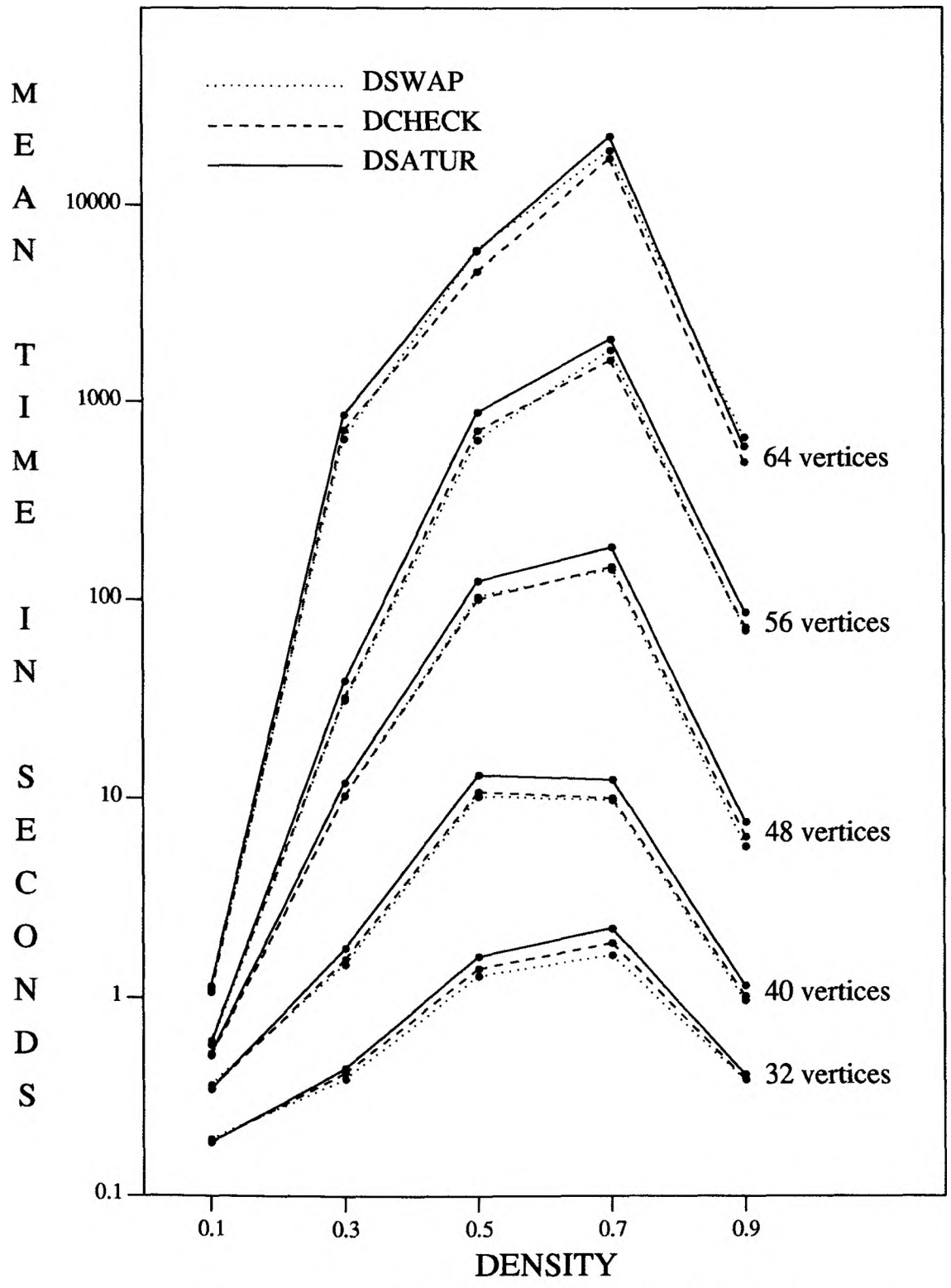


Figure 5. Mean Time to Find Minimal Coloring

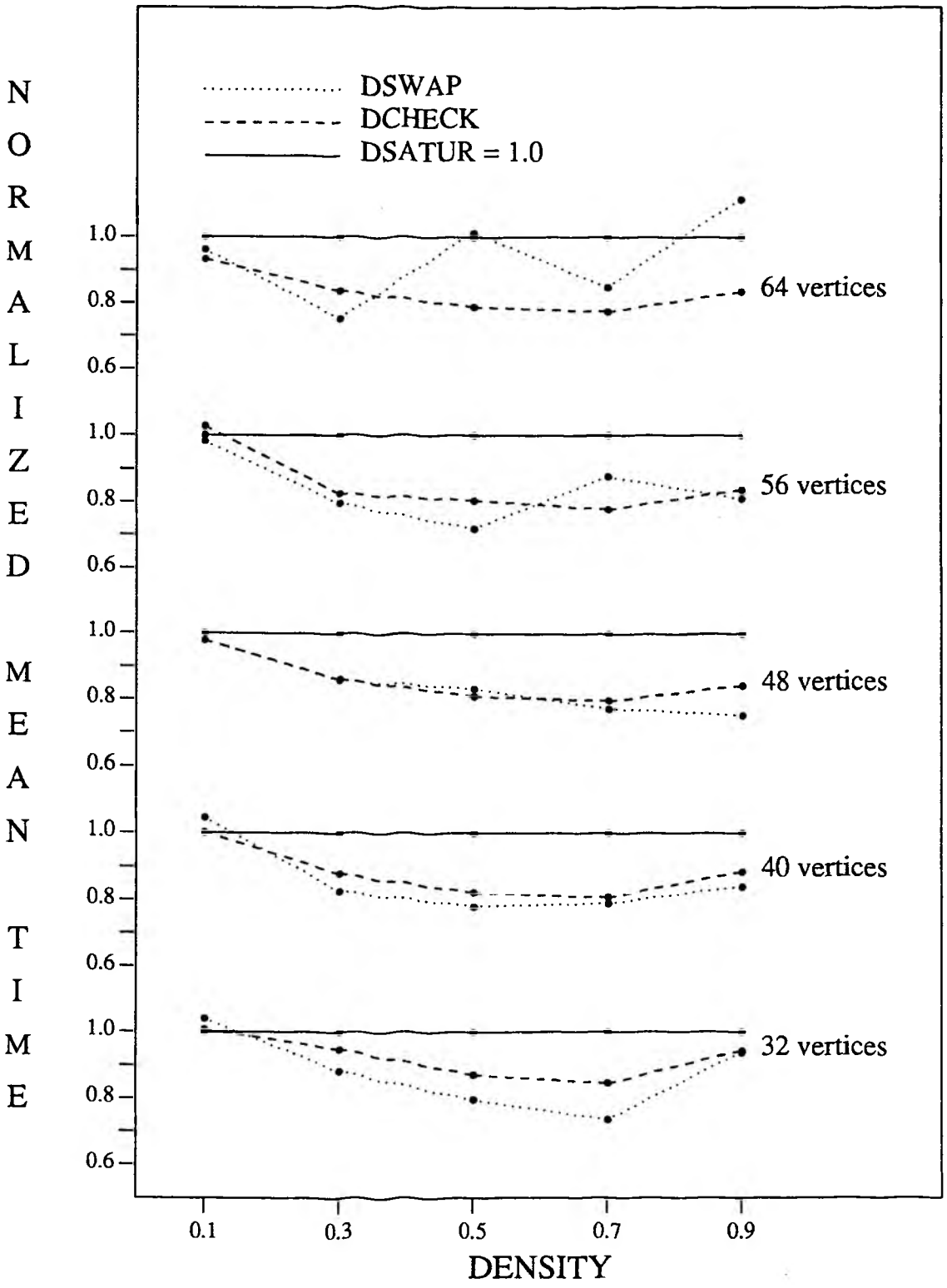


Figure 6. Normalized Mean Time to Find Minimal Coloring

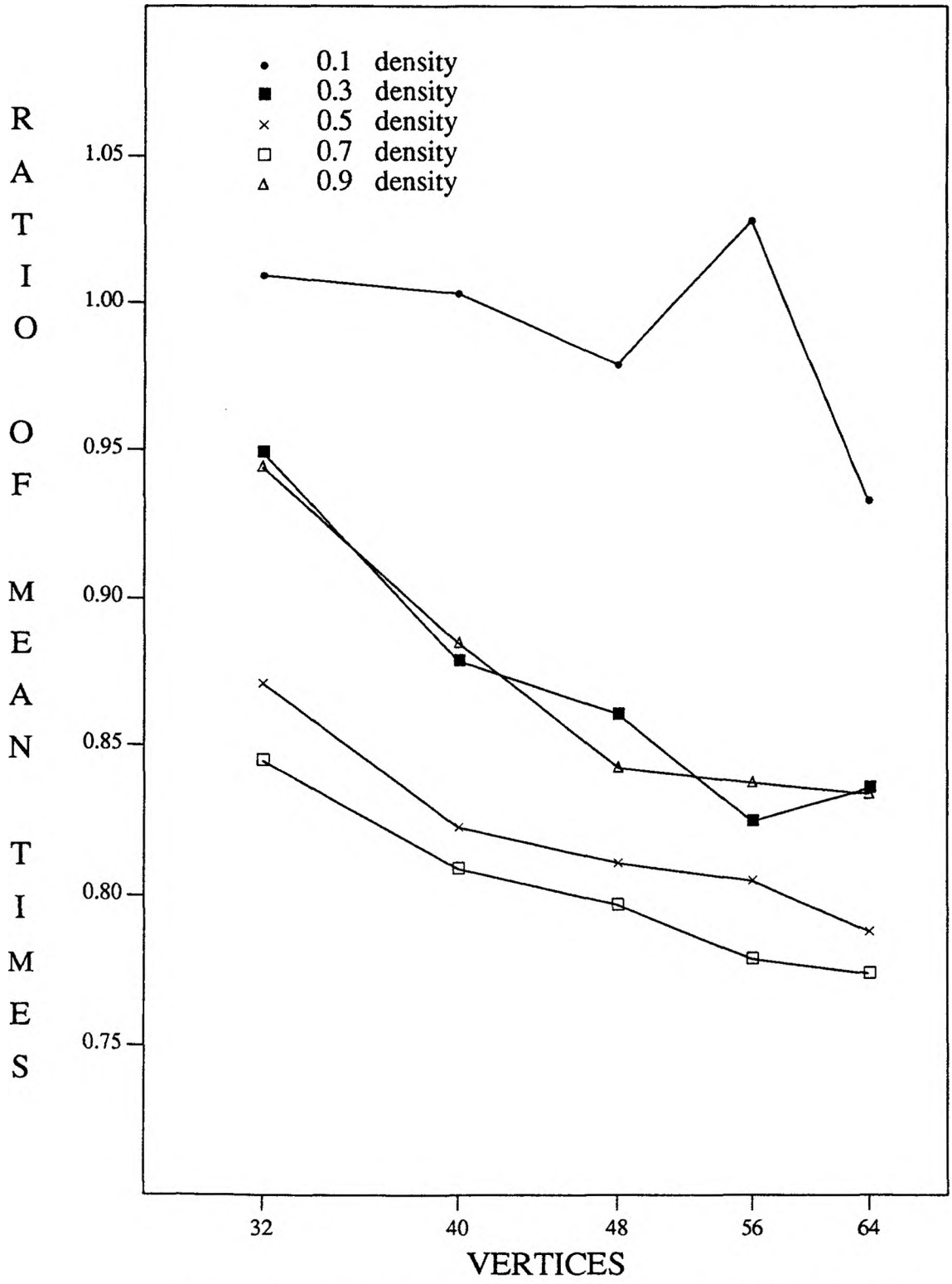


Figure 7. Ratio of Mean Times: DCHECK / DSATUR

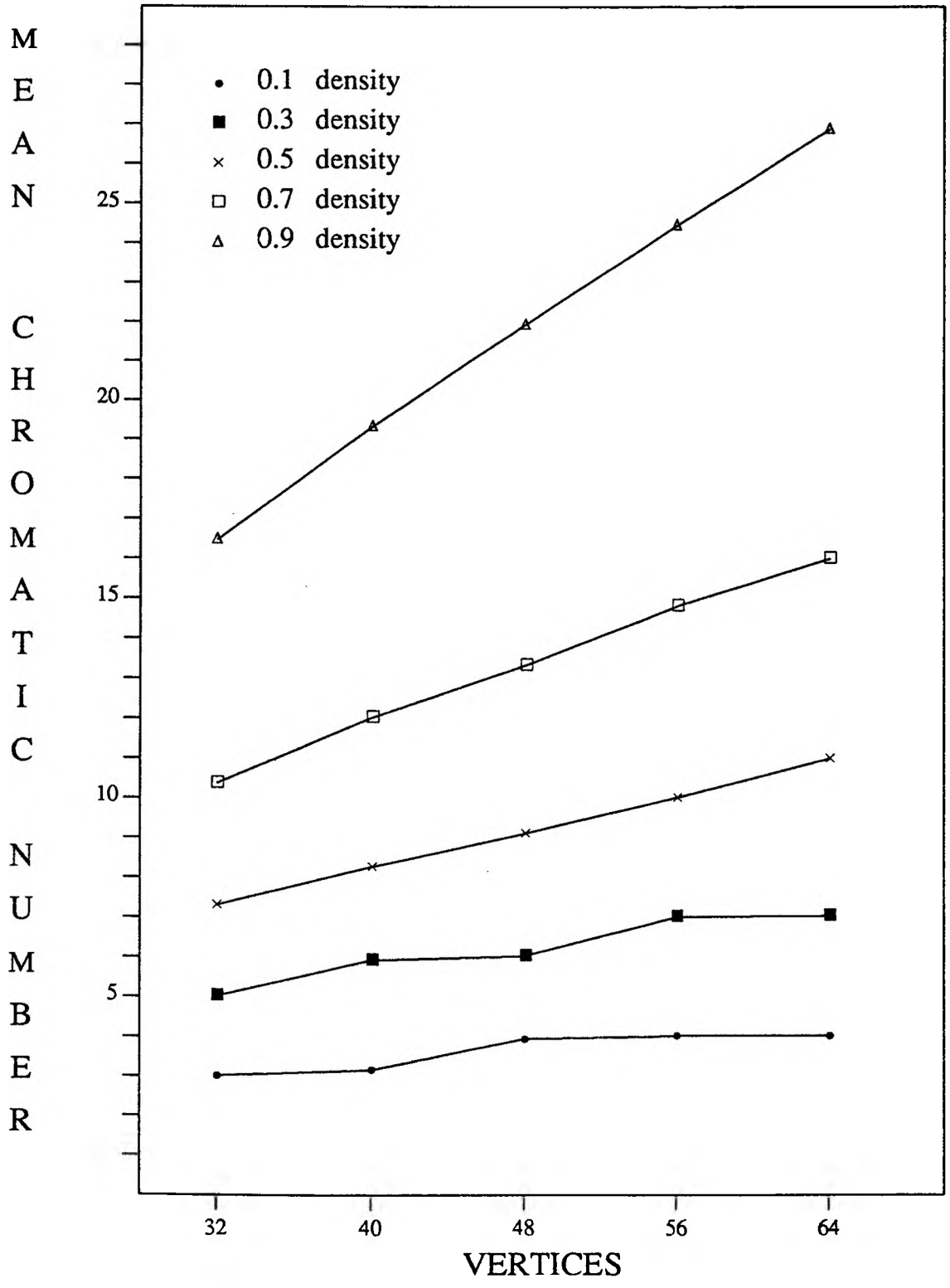


Figure 8. Mean Chromatic Number