01 Jan 1989

A Color-Exchange Algorithm for Exact Graph Coloring

Thomas J. Sager

Shi-Jen Lin

A COLOR-EXCHANGE ALGORITHM FOR EXACT GRAPH COLORING

Thomas J. Sager and Shi-Jen Lin

CSc-89-4

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401 (314)341-4491

A Color-Exchange Algorithm For Exact Graph Coloring

Thomas J. Sager
Department of Computer Science
University of Missouri-Rolla
Rolla, Missouri 65401 USA
email: tomsager@cs.umr.edu

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

Keywords: algorithms, branch-and-bound, chromatic number, graph-coloring, $\mathcal{NP}$-Complete, scheduling.

Abstract

DEXCH, a color-exchange exact graph coloring algorithm is presented. On many classes of graphs, *DEXCH* can, in the mean, find the chromatic number of a graph considerably faster than the *DSATUR* algorithm. The improvement over *DSATUR* stems from the ability to reorganize the subset of colored vertices and to detect 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. The mean improvement over *DSATUR* is greatest on high edge-density graphs attaining the value of 42% on random graphs of edge-density 0.7 on 64 vertices.

1 Introduction

The graph coloring problem can be stated as: *Given an undirected graph, $G = (V, E)$, with no loops or multiedges, find a function $f : V \to 1..k$, for some positive integer k, such that if $(v, w) \in E$ then $f(v) \neq f(w)$.* Such a function f is called a *coloring function*. If k is minimal over all of G's coloring functions, then f is called an *exact coloring function* and k is called the *chromatic number*. 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, heuristic graph coloring within a factor of 2 of the chromatic number is also $\mathcal{NP}$-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 have been studied by Korman [5] and Kubale and Jackowsky [6]. Both studies found that vertex sequential exact algorithms which use dynamic reordering of vertices usually give the best performance in practice.

Exact graph coloring algorithms can be used by themselves to color small graphs or as components of certain heuristic algorithms which can color large graphs. One such heuristic algorithm, *XRLF* [3], was found to outperform other known heuristic graph coloring algorithms on some classes of graphs. Thus, an improved exact graph coloring algorithm can yield improved heuristic graph coloring as well.

XRLF 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 the *DSATUR* algorithm to finish the coloring. Although originally presented by Brelaz [1] as a heuristic algorithm, a branch-and-bound version of *DSATUR* has come to represent a *de facto* standard among exact graph coloring algorithms. The branch-and-bound version, which we will refer to simply as *DSATUR*, is a vertex sequential algorithm with dynamic reordering of vertices.

In [9] we presented an exact graph coloring algorithm *DSWAP* which improved on the *DSATUR* algorithm by reorganizing the colored vertex subset according to a procedure which we called *swap*. In [10] we showed that most of the gain from *swap* comes from the portion of the algorithm which prunes the search tree and that furthermore as the size of a graph grows, the *swap* algorithm becomes more and more erratic with respect to *DSATUR*. We hypothesized the existence of a procedure for reorganizing the colored vertex set which would represent a significant improvement over *DSATUR*, but would not behave erratically as the size of the vertex set increased.

In this paper we present the *DEXCH (DSATUR COLOR-EXCHANGE)* algorithm. Unlike DSWAP, its behavior does not become erratic as the size of the vertex set increases. Also unlike *DSWAP*, on graphs of high edge-density and large vertex size, the colored vertex set reorganization component of the algorithm represents a significant part of the total improvement over *DSATUR*.

In section 2, we describe the *DEXCH* algorithm. Section 3 describes the methodology employed to compare the algorithms and the results of our comparisons.

2 The DEXCH Algorithm

In the folowing discussion, the vertices of a graph are named originally $1,2,3,\ldots$. 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. 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* from 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.

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, c be a colored vertex of G and x, y and z be vertices of G. Four procedures for transforming partially colored graphs are shown in Figure 1.

The *DEXCH* algorithm is based on the *DSATUR* algorithm but contains two additional components: a tree-pruning component and a colored vertex subset reorganization component. Pseudo-code for *DEXCH* is given in Figure 2. *DEXCH* with the colored vertex reorganization component removed will be referred to as algorithm *DPRUNE*. *DEXCH* with both the tree-pruning and colored vertex reorganization components removed is equivalent to *DSATUR*. The differences in behavior among the three algorithms are depicted in Figure 3.

The tree pruning component 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. The tree-pruning component is invoked whenever the current partially colored graph contains exactly one color fewer then the best coloring found so far. If three vertices with the above attributes are found, then the subtree rooted at the current partially colored graph is pruned since it cannot contain a completely colored graph using fewer colors than the best coloring found so far.

procedure $rename(G, vertices, y, z)$;
  $vertices(z) \leftarrow vertices(y)$; $vertices(y) \leftarrow undefined$;
  $E \leftarrow E \bigcup \{(z, x) \mid (y, x) \in E\} \setminus \{(y, x) \mid x \in V\}$; $V \leftarrow V \bigcup \{z\} \setminus \{y\}$;

procedure $newcolor(G, vertices, v)$;
  $rename(G, vertices, v, -(k+1))$; {Create a new colored vertex.}
  $E \leftarrow E \bigcup \{(c, -(k+1) \mid c \in -k.. - 1\}$; {Ensure C is still a complete subgraph.}

procedure $merge(G, vertices, v, c)$;
  $vertices(c) \leftarrow vertices(c) \bigcup vertices(v)$; $vertices(v) \leftarrow undefined$;
  $E \leftarrow E \bigcup \{(c, w) \mid (v, w) \in E\} \setminus \{(v, x) \mid x \in V\}$; $V \leftarrow V - \{v\}$;

procedure $exch(G, vertices, v, c)$;
  Let $x \notin V$; $rename(G, vertices, v, x)$;
  $rename(G, vertices, c, v)$; $rename(G, vertices, x, c)$;

Figure 1: Four operations on partially colored graphs.

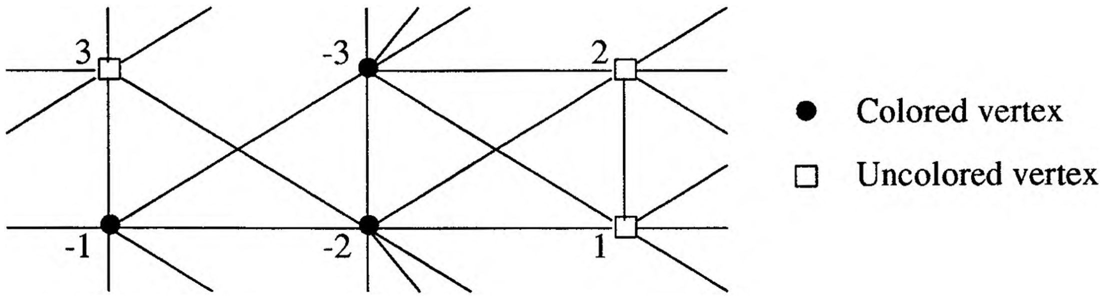
algorithm *DEXCH*;
  input: G = (V,E): graph;
  output: $\chi(G)$: positive integer; {Chromatic number}
    exactcf: **function**: $V \to 1..\chi(G)$; {Exact coloring function}

  procedure *color*($G = (V, E)$: a partially colored graph;
      vertices: **function**: $V \to \mathcal{P}(1..\infty)$);
    if G is completely colored **then** {In which case $V = C$}
      if $|V| < ncolors$ **then**
        $ncolors \leftarrow |V|$;
        $\forall j \in V, \forall i \in vertices(j), exactcf(i) \leftarrow -j$;
    else
      if $\exists v \in V \mid cadj(v) = C$ **then**
        if $|C| < ncolors - 1$ **then**
          choose $v \in V \mid cadj(v) = C$ and $wdegree(v)$ is maximal among all
            $v' \in V \mid cadj(v') = C$
          $newcolor(G, vertices, v)$; $color(G, vertices)$;
      else if $|C| = ncolors - 1$ and $\exists$ distinct v, w and $c \in V \mid (v, w) \in E$ and
          $cadj(v) = cadj(w) = C \setminus \{c\}$ **then return** **{Pruning Component}**
      else if $\exists v \in W$ and $c \in C \mid cadj(v) = C \setminus \{c\}$ and $wdegree(v) > wdegree(c)$
      then **{Reorganization Component}**
        choose $v \in V$ and $c \in C \mid cadj(v) = C \setminus \{c\}$ and $wdegree(v) - wdegree(c)$
            is maximal among all $v' \in V$ and $c' \in C \mid cadj(v') = C \setminus \{c'\}$;
        $exch(v, c)$;
      else
        choose $v \in V \mid cdegree(v)$ is maximal and $wdegree(v)$ is maximal
            among all $v' \in V \mid cdegree(v')$ is maximal;
        $\forall c \in C \mid c \notin cadj(v)$,
          if $|C| < ncolors$ **then**
            $G' \leftarrow G$; $vertices' \leftarrow vertices$;
            $merge(G', vertices', v, c)$; $color(G', vertices')$;
        if $|C| < ncolors - 1$ **then**
          $newcolor(G, vertices, v)$; $color(G, vertices)$;

$ncolors \leftarrow \infty$; $\forall v \in V, vertices(v) \leftarrow \{v\}$;
COLOR$(G, vertices)$; $\chi(G) \leftarrow ncolors$;

Figure 2: The *DEXCH* algorithm.

The colored vertex reorganization component is based on the desireability of having as many edges as possible incident to the colored vertex subset. *DSATUR* attempts to maximize this attribute by choosing at each step an uncolored vertex with maximal *wdegree* among those uncolored vertices with maximal *cdegree*. *DEXCH*, in addition, will attempt to maximize this attribute by searching for two vertices, v and c such that $cadj(v) = C \setminus \{c\}$ and $wdegree(v) > wdegree(c)$. If such a pair is found, *DEXCH* replaces c by v in the colored vertex subset.



Since $wdegree(3) > max(wdegree(1), wdegree(2))$, *DSATUR* merges vertex 3 into vertex -3.

If the best coloring so far uses 4 colors, *DPRUNE* detects the complete subgraph $\{-3, -2, 1, 2\}$ and prunes the search tree; otherwise *DPRUNE* merges vertex 3 into vertex -3.

If the best coloring so far uses more than 4 colors, *DEXCH* reorganizes the colored vertex set as $\{-2, -1, 3\}$; otherwise *DEXCH* prunes the search tree.

Figure 3: Behavior of three algorithms.

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*), increases the number of edges incident to the set of colored vertices (*exch*) or decreases the number of uncolored vertices (*merge*). *color* calls itself recursively until its argument is either completely colored or contains no fewer colored vertices than the best coloring found so far.

3 Methodology and Results

All three algorithms, *DSATUR, DPRUNE* and *DEXCH*, were programmed 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; 100 random graphs were generated using Park and Miller's minimal standard random number generator [8]. Each algorithm was executed on a PC AT computer to produce an exact coloring function for all 100 random graphs except for densities of 0.5 and 0.7 on 64 vertices where, because of the time involved, an exact coloring for only the first 30 random graphs generated and the first 10 random graphs generated respectively was produced. Execution times of the three algorithms were compared with the *paired t test*. Mean execution times and the results of the *paired t test* at the 95% confidence level are shown in Table 1.

We found that both *DEXCH* and *DPRUNE* are consistently faster than *DSATUR* at the 95% confidence level for all graphs of between 40 and 64 vertices and all densities between 0.3 and 0.9. In addition, we found that *DEXCH* performs significantly faster than *DPRUNE* on most classes of graphs of high density (0.7 and 0.9) on 40 through 64 vertices. The relative improvement of *DEXCH* over *DSATUR* reaches its maximum

		VERTICES				
		32	40	48	56	64
		D† 1.86E-1	*D*† 3.47E-1	*P*•† 5.11E-1	*D*⋆† 5.88E-1	*E* 1.02
	0.1	*P*† 1.87E-1	*P*† 3.48E-1	*D*† 5.22E-1	*P*† 6.04E-1	*P* 1.06
		E 2.07E-1	*E* 3.74E-1	*E* 5.42E-1	*E* 6.33E-1	*D* 1.14
		P•† 4.19E-1	*P*• 1.56	*E*• 9.89	*P*•† 3.22E1	*P*• 7.14E2
D	0.3	*D* 4.42E-1	*E*• 1.57	*P*• 1.03E1	*E*• 3.38E1	*E*• 7.37E2
E		*E* 4.44E-1	*D* 1.77	*D* 1.20E1	*D* 3.91E1	*D* 8.54E2
N		*P*• 1.40	*E*•⋆ 9.89	*E*• 9.93E1	*P*• 7.09E2	*E*• 3.91E3
S	0.5	*E*• 1.46	*P*• 1.08E1	*P*• 1.01E2	*E*• 7.25E2	*P*• 4.58E3
I		*D* 1.60	*D* 1.31E1	*D* 1.24E2	*D* 8.81E2	*D* 5.81E3
T		*E*•⋆ 1.70	*E*•⋆ 7.90	*E*•⋆ 1.21E2	*E*• 1.54E3	*E*•⋆ 1.29E4
Y	0.7	*P*• 1.88	*P*• 1.01E1	*P*• 1.46E2	*P*• 1.63E3	*P*• 1.72E4
		D 2.23	*D* 1.24E1	*D* 1.84E2	*D* 2.09E3	*D* 2.21E4
		P•† 3.89E-1	*E*• 9.82E-1	*E*•⋆ 5.68	*E*•⋆ 5.34E1	*E*•⋆ 3.76E2
	0.9	*D* 4.12E-1	*P*• 1.01	*P*• 6.39	*P*• 7.24E1	*P*• 4.91E2
		E 4.21E-1	*D* 1.15	*D* 7.59	*D* 8.65E1	*D* 5.88E2

D *DSATUR* algorithm. *P* *DPRUNE* algorithm.
E *DEXCH* algorithm. yEx means $y * 10^x$.

• faster than *DSATUR* algorithm at 95% confidence level.
⋆ faster than *DPRUNE* algorithm at 95% confidence level.
† faster than *DEXCH* algorithm at 95% confidence level.

Table 1: Mean execution time in seconds.

value of 42% on graphs of density 0.7 on 64 vertices, the most time-consuming class of graphs to color. On graphs of moderate density (0.3 and 0.5) on 40 through 64 vertices, both *DEXCH* and *DPRUNE* appear equally good. On most classes of low density (0.1) or small (32 vertices) graphs, *DEXCH* is outperformed by at least one of the other two algorithms. The relative performance of the three algorithms with *DSATUR* normalized at 1.0 is depicted graphically in Figure 4.

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.

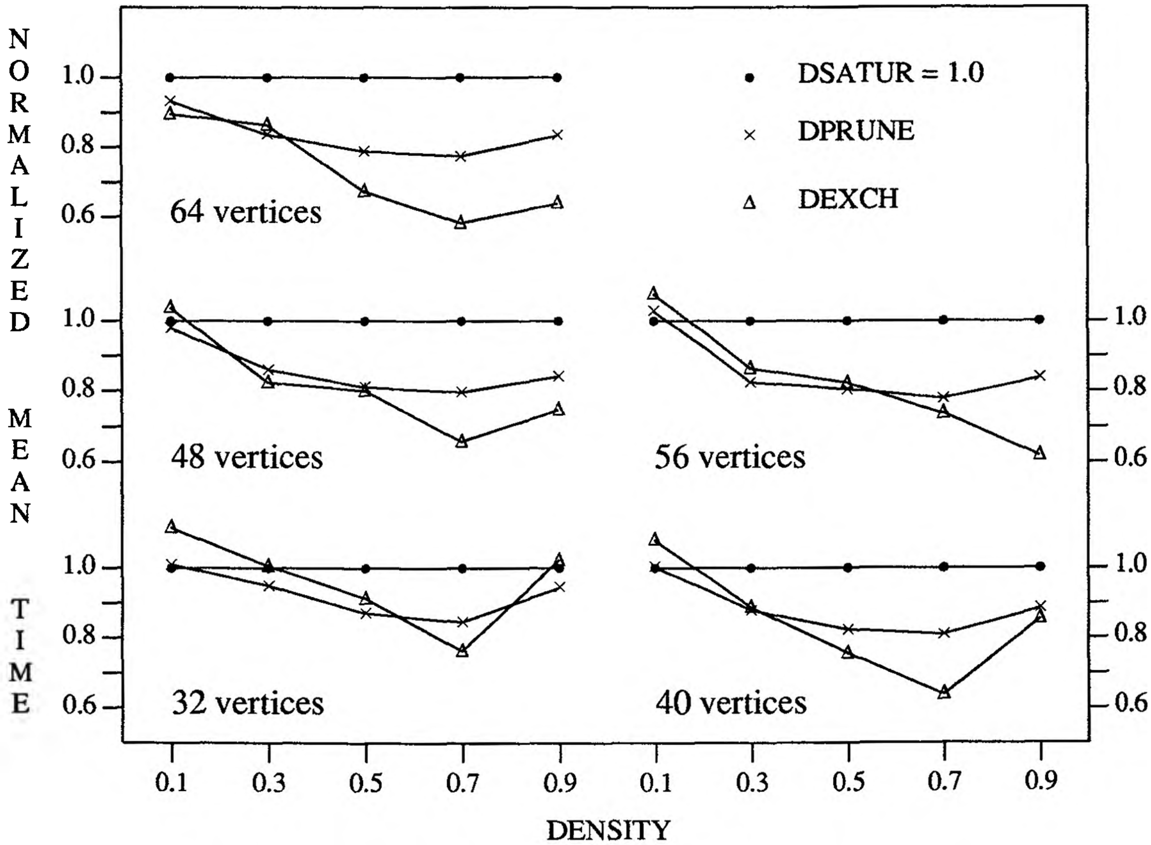


Figure 4: Normalized mean time to find chromatic number.

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

[9] Sager, T.J. and S.J. Lin: An improved exact graph coloring algorithm. *Tech. Rep. CSc-89-1*, University of Missouri-Rolla, Rolla, Missouri, April 1989.

[10] Sager, T.J. and S.J. Lin: A pruning procedure for exact graph coloring. *Tech. Rep. CSc-89-3*, University of Missouri-Rolla, Rolla, Missouri, October 1989.