JOURNAL OF RESEARCH of the National Bureau of Standards
Vol. 84, No. 6, November–December 1979

A Graph Coloring Algorithm for Large Scheduling Problems*

Frank Thomson Leighton**

Center for Applied Mathematics, National Bureau of Standards Washington, DC 20234

June 6, 1979

A new graph coloring algorithm is presented and compared to a wide variety of known algorithms. The algorithm is shown to exhibit $O(n^2)$ time behavior for most sparse graphs and thus is found to be particularly well suited for use with large-scale scheduling problems. In addition, a procedure for generating large random test graphs with known chromatic number is presented and is used to evaluate heuristically the capabilities of the algorithms discussed.

Key words: Algorithm; chromatic number; color function; graph; graph coloring; heuristic; interchange; random test graphs; scheduling; time complexity.

1. Introduction

Graph coloring has considerable application to a large variety of complex problems involving optimization. In particular conflict resolution, or the optimal partitioning of mutually exclusive events, can often be accomplished by means of graph coloring. Examples of such problems include: the scheduling of exams in the smallest number of time periods such that no individual is required to participate in two exams simultaneously (see appendix A), the storage of chemicals on the minimum number of shelves such that no two mutually dangerous chemicals (i.e., dangerous when one is in the presence of the other) are stored on the same shelf, and the pairing of individuals (as in a computer dating agency) such that the maximal number of compatible persons are paired together.

In each of the above problems, the constraints are usually expressible in the form of pairs of incompatible objects (e.g., pairs of chemicals that cannot be stored on the same shelf). Such incompatibilities are usefully embodied through the structure of a graph. Each object is represented by a node and each incompatibility is represented by an edge joining the two nodes. A coloring of this graph is then simply a partitioning of the objects into blocks (or colors) such that no two incompatible objects end up in the same block. Thus, optimal solutions to such problems may be found by determining minimal colorings for the corresponding graphs. Unfortunately, this may not always be accomplishable in a reasonable amount of time.

As the graph coloring problem is known to be NP-complete [1],[1] there is no known algorithm which, for every graph, will optimally color the nodes of the graph in a time bounded by a polynomial in the number of nodes. Since exponential time algorithms [5, 6, 7, 9, 18] are prohibitively expensive for use with large-scale problems, much attention has been focused on the development of heuristic algorithms which will usually produce a good, though not necessarily optimal, coloring for any graph in a reasonable amount of time.

This paper describes a new graph coloring algorithm, the recursive largest first (RLF) coloring algorithm. In addition, a variety of existing coloring procedures are presented and their performance on a wide range of test data is compared to that of the RLF algorithm.

Also described is a procedure for generating random graphs with known chromatic number. The existence of such a procedure, heretofore lacking in the experimental literature, provides a standard method for testing the accuracy of graph coloring algorithms.

AMS-MOS 1970 Subject Classification: 05C15, 68A10, 68A20, 90B35.

* This work was done in part while the author was a staff member of the Center for Applied Mathematics of the National Bureau of Standards during the summer of 1976 and in part while the author was an undergraduate majoring in Electrical Engineering and Computer Sciences at Princeton University working under the supervision of Professor Forman S. Acton.

** Present Address: Department of Applied Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

[1] Numbers in brackets indicate literature references at the end of this paper.

2. Preliminary Definitions

Throughout this paper, the graph G with nodes V and edges E, denoted by (V, E), is assumed to contain no loops or multiple edges. The subgraph of $G = (V, E)$ *induced* by a subset U of the nodes V consists of those nodes and all the edges that directly connect them. This subgraph is represented by $< U >$ or (U, E') where $E' = \{(w_1, w_2) | (w_1, w_2) \in E, w_1 \in U, w_2 \in U\}$. The *degree* of a node $w \in G$, denoted by $d(w)$, is the number of nodes adjacent to w in G. Define $d_U(w)$ to be the number of nodes in U adjacent to w in G. This is equivalent to the degree of w in $< U \cup \{w\} >$.

A *coloring* of G is an assignment of colors to the nodes of G such that no two adjacent nodes share the same color. More formally, a *k-coloring* of G is a mapping $f: V \rightarrow \{1, 2, \ldots, k\}$ such that $f(u) = f(v)$ only if $(u, v) \notin E$. The *chromatic number* of G, denoted $\chi(G)$, is the minimal number of colors necessary to color G. An *optimal coloring* of G is one which uses exactly $\chi(G)$ colors.

3. Sequential Coloring Algorithms

One of the simplest coloring algorithms is the randomly ordered sequential (RND) graph coloring algorithm [14]. Given a graph $G = (V, E)$, the algorithm randomly orders the nodes so that $V = \{v_1, \ldots, v_n\}$ and then assigns colors to the nodes in the following manner. The first node, v_1, is assigned color number 1. Once the first i nodes have been colored $(1 \leq i \leq n - 1)$, v_{i+1} is assigned the lowest possible color number such that no previously colored node adjacent to v_{i+1} has been assigned the same color number.

Though this algorithm is locally optimal in the sense that each node is assigned the smallest possible number, the overall action is highly dependent on the initial ordering of the nodes. For any graph, there exists an ordering for which this algorithm will produce an optimal coloring [14], while a less fortuitous ordering may lead to an extremely poor coloring. Thus the problem of finding an optimal initial ordering of the nodes is equivalent to the problem of optimally coloring the graph.

This fact has led to the development of a large number of algorithms, each differing from RND only in the method of initially ordering the nodes [7, 14]. Two such algorithms are the largest first (LF) and smallest last (SL) sequential coloring algorithms.

The LF algorithm orders the nodes such that $d(v_i) \geq d(v_{i+1})$ for $1 \leq i < n$ where $V = \{v_1, \ldots, v_n\}$. The SL algorithm is similar in strategy but recursively orders the smallest degree nodes last. An SL ordering is one in which $d(v_n) = \min_{w \in V} d(w)$ and for $n - 1 \geq i \geq 1$, $d_U(v_i) = \min_{w \in U} d_U(w)$ where $U = V - \{v_n, \ldots, v_{i+1}\}$.

Note that both the LF and SL algorithms tend to order the high degree nodes before the low degree nodes. Computational experience has shown that this is generally a good strategy, whereas algorithms which color the higher degree nodes last have often been found to produce colorings worse than those produced by a random ordering.

Each of the sequential coloring algorithms presented in this section requires $O(n^2)$ time and $O(n^2)$ space to color a graph with n nodes. Quadratic time and space complexities are generally quite acceptable for use with large-scale coloring problems. If only they gave guaranteed optimal colorings, we would look no further.

4. More Sophisticated Algorithms

One successful variation of the sequential coloring algorithms involves what is known as an interchange. Given any $G = (V, E)$ and color function f such that $f(w) \in \{i, j\}$ for all $w \in V$, and (i, j)-*interchange* on G is a redefinition of f such that if $f(w) = i$ originally, $f(w)$ is now assigned j and vice versa for all $w \in V$.

Appropriate use of the interchange process has been found to yield particularly good results when used in conjunction with the LF and SL algorithms [14]. The resulting procedures are referred to as the smallest last with interchange (SLI) and largest first with interchange (LFI) coloring algorithms.

The SLI (LFI) algorithm operates just like the SL (LF) algorithm except when the latter requires the introduction of a new color. Suppose that such a situation occurs when v_m is the node to be colored and that $k = \max_{i<m} f(v_i)$. For $1 \leq i < j \leq k$, define G_{ij} to be the subgraph of G induced by the nodes of G previously colored i or j. If possible, choose i and j such that no connected component of G_{ij} contains two differently colored nodes both adjacent to v_m. If such a G_{ij} is found to exist, then perform an (i, j) − interchange on each connected component of G_{ij} which contains an i-colored node adjacent to v_m in G. It is now possible to assign

color i to v_m and thus the addition of a new color has been avoided. If no such G_{ij} exists, however, then regardless of what interchange is performed, v_m must be assigned color $k + 1$.

This version of the SLI (LFI) algorithm initially appeared in [11] and is an extension of the original version which is described in [14]. The original version allows an (i, j) − interchange only when v_m is adjacent to exactly one node colored i and one node colored j. There is little difference between the original and extended versions of the SLI and LFI algorithms in terms of colorings produced or time required. While the extended versions may be able to perform a useful interchange impossible in the original version, they will likely take slightly longer to do so. All four algorithms require $O(n^3)$ time and $O(n^2)$ space to color an n node graph. Based on a limited amount of computational experience, the extended version of the SLI algorithm (henceforth to be referred to simply as the SLI algorithm) was found to produce slightly better results than did the other interchange procedures.

All of the algorithms thus far presented are capable of producing very bad colorings, in terms of number of colors used, for certain graphs. Johnson [10, 11] has given constructions of 3-colorable graphs on $O(n)$ vertices which each of the above algorithms requires n colors to color completely. Since no more than $O(n)$ colors may be used to color an $O(n)$ node graph, such colorings are, up to a constant, the worst possible.

There is an algorithm, however, which will color any graph G with n nodes in $O\left(\frac{n}{\log n}\right)\chi(G)$ or fewer colors. While this worst-case behavior is still unacceptable in practice, the approximately maximum independent set (AMIS) algorithm is interesting because it is the only known algorithm which is known not to exhibit the worst possible worst-case behavior [11]. The algorithm proceeds as follows. Given $G = (V, E)$, select the node with minimum degree in G, say v_1, and color it 1. Once i nodes have been assigned color 1, select, if possible $v_{i+1} \in U$ such that $d_U(v_{i+1})$ is minimal for nodes in U where U is the set of uncolored nodes not adjacent to any colored node. If no such selection is possible, i.e., U is empty, then repeat the entire procedure on the subgraph of G induced by the uncolored nodes of G, using the next available color. This process is then, in turn, repeated until all the nodes of G have been colored.

Interestingly enough, while this algorithm exhibits better worst-case behavior than the other algorithms thus far discussed, computational experience has shown that, on the average, the colorings it produces are substantially inferior to those produced by the LF, SL, and SLI algorithms.

5. The Recursive Largest First (RLF) Algorithm

The RLF algorithm combines the strategy of the LF algorithm with the structure of the AMIS algorithm. Like the LF algorithm, at each step in the RLF procedure a node is selected for coloring which will, in some sense, leave the resulting uncolored nodes colorable in as few colors as possible. As with the AMIS algorithm, the RLF procedure completes the assignment of color i before commencing assignment of color $i + 1$.

The RLF graph coloring algorithm proceeds as follows. Given $G = (V, E)$, assign color 1 to the node with maximal degree in G, say v_1. Once i nodes have been assigned color 1, select, if possible, $v_{i+1} \in U_1$ such that $d_{U_2}(v_{i+1})$ is maximal for nodes in U_1 where U_1 is the set of uncolored nodes not adjacent to any colored node and U_2 is the set of uncolored nodes adjacent to at least one colored node. Ties are, if possible, broken by choosing the node that has minimal degree in $< U_1 >$. If no such selection is possible, i.e., U_1 is empty, then repeat the entire process recursively on the subgraph of G induced by the uncolored nodes of G, using the next available color. This recursion is then repeated until all of the nodes in G are colored. Several examples of this procedure are worked out in appendix A.

As was true with the SLI algorithm, the RLF algorithm, in general, requires $O(n^3)$ time and $O(n^2)$ space to color an n node graph. Unlike the SLI algorithm, however, the RLF algorithm requires only $O(n^2)$ time to color graphs for which $k \cdot e \approx n^2$ where k is the number of colors used to color the graph, e is the number of edges in the graph, and n is the number of nodes in the graph (see appendix B for proof). Such graphs, which are usually sparse, quite commonly arise in practical applications such as exam scheduling. For example, the graph associated with the 1977–8 Princeton University fall term course examinations schedule consisted of 273 nodes, 6727 edges, and required 17 colors to be colored by the RLF algorithm. Thus, for practical purposes, the RLF algorithm, if programmed properly, exhibits an $O(n^2)$ time dependence for many applications. Appendix B presents a PL-1 listing of the RLF algorithm as well as a rigorous analysis of its time complexity.

6. Generation of Test Graphs With Known Chromatic Number

A few papers have been published which compare the performance of various algorithms on large (usually 100-node) randomly generated graphs [14, 21, 23]. Unfortunately, none of these empirical studies provide the chromatic numbers of the test graphs used. Indeed, the task of closely approximating the chromatic number of a graph is NP-complete [8] and thus virtually impossible to accomplish for large graphs. Consequently, approximations of upper and lower bound results established for $\chi(G)$ have generally been crude and of little practical use [1, 7, 14, 19].

The lack of such information makes an accurate interpretation of the experimental data very difficult. For instance, if algorithm A required 22 colors to color G while algorithm B required only 20, the conclusions drawn about their relative effectiveness if $\chi(G) = 20$ might be quite different from those drawn if $\chi(G) = 4$. Further, without knowledge of $\chi(G)$, no statement can be made at all about the accuracy or closeness to optimality of either algorithm A or B. Thus there is a need for a standard procedure for generating random test graphs with known chromatic numbers. Such a procedure will now be presented.

Suppose it is desired to construct an n-node graph G with e edges and chromatic number k. For the purposes of the following argument, assume that $k|n$. This is not a significant restriction since most test or modeling uses of a large graph generator are likely to allow some flexibility in the choices of n and k. For such a graph to exist under these restrictions, e must be such that

$$\frac{k(k-1)}{2} \leq e \leq \frac{n^2(k-1)}{2k}.$$

The first step in the procedure is to choose positive integers a, c and m such that:

1. $m >> n$,
2. $(n, m) = k$,
3. $(c, m) = 1$,
4. $p|m \rightarrow p|(a-1)$ for all primes p, and
5. $4|m \rightarrow 4|(a-1)$.

Next generate a uniform sequence of random numbers $\{X_i\}$ on the interval 0 to $m-1$ by the linear congruential method described in [12]. This is accomplished by fixing X_0 and, then for each $i > 0$, setting $X_i = \text{MOD}(aX_{i-1} + c, m)$ where $\text{MOD}(X, Y) = X - \left[\frac{X}{Y}\right] * Y$. Sequences generated in such a manner exhibit two important properties [12]. First, for every i and j such that $0 \leq j \leq m-1$ and $i \geq 0$, there exists an r such that $i \leq r \leq i + m - 1$ and $X_r = j$. Second, for every $i \geq 0$, $X_i = X_{i+m}$.

Next construct the sequence $\{Y_i\}$ on the interval 0 to $n-1$ so that $Y_i = \text{MOD}(X_i, n)$. Note that unless $k = n$, $n \nmid m$ and $\{Y_i\}$ is *not* a uniform random number sequence on the interval 0 to $n-1$.

By defining $V = \{0, 1, \ldots, n-1\}$, it is possible to associate two consecutive values of $\{Y_i\}$ with edges to be added to E. Similarly, it is possible to associate h consecutive values of $\{Y_i\}$ with h-cliques to be implanted in G. For example, the subsequence $\{Y_1, Y_2, Y_3\}$ corresponds to the subset $\{(v_{Y_1}, v_{Y_2}), (v_{Y_1}, v_{Y_3}), (v_{Y_2}, v_{Y_3})\}$. By identifying certain subsequences of consecutive elements of $\{Y_i\}$ and adding the corresponding edges to E, it is possible to construct the desired graph G.

More precisely, define the $(k-1)$-vector $\mathbf{b} = (b_k, b_{k-1}, \ldots, b_2)$ so that $b_k \geq 1$ and $b_i \geq 0$ for $2 \leq i \leq k-1$. Each b_i corresponds to the number of i-cliques to be implanted in G. Specifically, given the sequence $\{Y_i\}$ and vector $\mathbf{b}$, proceed as follows. Select the first k values of $\{Y_i\}$ starting with Y_1 and add the corresponding edges to E. If $b_k > 1$, select the next k values of $\{Y_i\}$ and add the corresponding edges to E. Repeat this process until b_k k-cliques have been implanted in G. Next add, in an identical fashion, b_{k-1} $(k-1)$-cliques to G. Continue the process until b_2 2-cliques or edges have been added to E. Note that some edges may be "added" several times and thus it may not be possible to precalculate a vector $\mathbf{b}$ such that there are exactly e edges in the resulting graph. It is possible, however, to keep track of how many edges have been added at any point and to eliminate the addition of i-cliques which might result in the addition of too many edges to E. Since edges may be added one at a time, it is not difficult to show that graphs having exactly e edges may be constructed in this manner for any e such that

$$\frac{k(k-1)}{2} \leq e \leq \frac{n^2(k-1)}{2k}.$$

It now remains to be shown that $\chi(G) = k$ for any G constructed in this manner. Since $b_k \geq 1$, G contains a k-clique and thus $\chi(G) \geq k$. Before establishing that $\chi(G) \leq k$, it is useful to examine the structure of the sequence $\{Y_i'\}$ where $Y_i' = \text{MOD}(Y_i, k)$. Since $k|n$ and $k|m$,

$$\begin{aligned}
Y'_{i+1} &= \text{MOD}(Y_{i+1}, k) \\
&= \text{MOD}(\text{MOD}(X_{i+1}, n), k) \\
&= \text{MOD}(X_{i+1}, k) \\
&= \text{MOD}(\text{MOD}(aX_i + c, m), k) \\
&= \text{MOD}(aX_i + c, k) \\
&= \text{MOD}(\text{MOD}(aX_i + c, n), k) \\
&= \text{MOD}(aY_i + c, k) \\
Y'_{i+1} &= \text{MOD}(aY_i' + c, k).
\end{aligned}$$

Further,

$$\begin{aligned}
&p|k \rightarrow p|m \rightarrow p|(a-1), \\
&4|k \rightarrow 4|m \rightarrow 4|(a-1), \quad \text{and} \\
&(c, m) = 1 \rightarrow (c, k) = 1.
\end{aligned}$$

Thus $\{Y_i'\}$ is a uniform sequence of random numbers on the interval 0 to $k-1$.

This structure of the $\{Y_i\}$ modulo k allows the following coloring of G. For each i, define $f(v_{Y_i}) = \text{MOD}(i, k)$. Since for all j such that $0 \leq j < n$, there exists an $i \geq 0$ such that $Y_i = j$, it is clear that every node is assigned a color by this procedure. Since $\{Y_i\}$ is a uniform sequence of random numbers on the interval from 0 to $k-1$, we know that if $Y_i = Y_j$, then $Y_i' = Y_j'$ and $\text{MOD}(i, k) = \text{MOD}(j, k)$ and thus that $f(v_{Y_i}) = f(v_{Y_j})$. This means that f is well defined. Finally, it is easily verified that $v_{Y_i}, v_{Y_{i+1}}, \ldots, v_{Y_{i+h-1}}$ are all colored differently if $h \leq k$, for all $i \geq 0$. This means that edges occur only between differently colored nodes, and that f is a proper coloring of G. Thus $\chi(G) = k$.

It should be noted that the above result is a special case of a more general result for arbitrary k and n. That is, if k and n are such that $k \leq d$ where $d = (n, m)$, then $k \leq \chi(G) \leq k + \text{MOD}(d, k) < 2k$. The proof of the general result is not given here but is similar to that of the special case when $\text{MOD}(d, k) = 0$.

As will be demonstrated shortly, the range of graphs which can be generated by this procedure is quite large. The node degrees of such graphs may vary between 0 and $n - \frac{n}{k}$ while the average node degree may vary between $\frac{k(k-1)}{n}$ and $\frac{k-1}{k}n$. The variety of distributions of node degrees is also quite large. Most importantly, however, the procedure generates graphs which are as difficult to color as are randomly generated graphs (where the chromatic number is not known). Demonstration of this fact is provided in section 7.

Another advantage of this procedure is that the test graphs may be easily characterized. For example, only $k + 5$ values are required to generate an n-node graph with chromatic number k. These values are n, k, X_0, a, c, m, b_k, $b_{k-1}, \ldots, b_3$ and b_2. Whereas it would be infeasible to completely describe a large, randomly generated graph by conventional means in a short paper, graphs generated by this procedure are easily described. Thus, in future publications concerning the effectiveness of various graph coloring algorithms, it will be possible to specify precisely which graphs were used to test the various algorithms. There are several conceivable situations where such documentation could be valuable to the interested reader. For example, should the reader desire to compare the effectiveness of a new graph coloring algorithm to those in the literature, he would need only to regenerate the graphs used in published tests and color them with the new

algorithm. This would eliminate the necessity of developing an entirely new set of test data and of having to rerun all previous algorithms on such data. Pursuant to these goals, a complete characterization of the test graphs referred to in tables 1 and 2 is provided in appendix C.

7. Test Results

The procedure described above was used to generate 27 150-node graphs and 12 450-node graphs of varying edge density and chromatic number. In addition, 27 completely random 150-node graphs were generated with varying edge density and unknown chromatic number. The RND, LF, SL, RLF and SLI algorithms were tested on each of the 66 graphs. The resulting data are displayed in tables 1, 2, and 3, respectively.

In each table, the graphs are subdivided into groups according to chromatic number, χ, (or as in the case with the completely random graphs, to a known lower bound for χ) and average node degree, d. There are

TABLE I. *Results for 150-node test graphs generated according to the procedure detailed in section 6.*

χ	d	number of colors used					average number of excess colors used				
		RND	LF	SL	RLF	SLI	RND	LF	SL	RLF	SLI
5	11	(9, 9, 9)	(7, 8, 7)	(7, 7, 8)	(6, 6, 6)	(6, 6, 6)	4	$2\frac{1}{3}$	$2\frac{1}{3}$	1	1
	19	(11, 11, 12)	(10, 10, 9)	(9, 10, 9)	(7, 7, 8)	(9, 5, 8)	$6\frac{1}{3}$	$4\frac{2}{3}$	$4\frac{1}{3}$	$2\frac{1}{3}$	$2\frac{1}{3}$
	24	(13, 12, 13)	(10, 11, 11)	(10, 9, 10)	(7, 6, 7)	(8, 7, 6)	$7\frac{2}{3}$	$5\frac{2}{3}$	$4\frac{2}{3}$	$1\frac{2}{3}$	2
Ave. total							6	$4\frac{2}{9}$	$3\frac{7}{9}$	$1\frac{6}{9}$	$1\frac{7}{9}$
10	11	(11, 11, 12)	(10, 10, 10)	(10, 10, 10)	(10, 10, 10)	(10, 10, 10)	$1\frac{1}{3}$	0	0	0	0
	21	(15, 14, 16)	(12, 12, 12)	(12, 12, 12)	(11, 11, 11)	(11, 11, 11)	5	2	2	1	1
	29	(17, 16, 18)	(14, 14, 14)	(14, 15, 15)	(13, 13, 12)	(13, 13, 12)	7	4	$4\frac{2}{3}$	$2\frac{2}{3}$	$2\frac{2}{3}$
Ave. total							$4\frac{4}{9}$	2	$2\frac{2}{9}$	$1\frac{2}{9}$	$1\frac{2}{9}$
15	12	(15, 16, 15)	(15, 15, 15)	(15, 15, 15)	(15, 15, 15)	(15, 15, 15)	$\frac{1}{3}$	0	0	0	0
	25	(19, 18, 18)	(17, 16, 16)	(16, 16, 15)	(15, 15, 15)	(15, 15, 15)	$3\frac{1}{3}$	$1\frac{1}{3}$	$\frac{2}{3}$	0	0
	34	(19, 21, 20)	(17, 17, 18)	(17, 19, 17)	(16, 16, 16)	(16, 16, 16)	5	$2\frac{1}{3}$	$2\frac{2}{3}$	1	1
Ave. total							$2\frac{8}{9}$	$1\frac{2}{9}$	$1\frac{1}{9}$	$\frac{1}{3}$	$\frac{1}{3}$
Overall average total							$4\frac{12}{27}$	$2\frac{13}{27}$	$2\frac{10}{27}$	$1\frac{2}{27}$	$1\frac{3}{27}$
Total time (seconds)		10.2	13.3	13.5	15.3	49.2					

TABLE 2. *Results for 450-node test graphs generated according to the procedure detailed in section 6.*

χ	d	number of colors used					average number of excess colors used				
		RND	LF	SL	RLF	SLI	RND	LF	SL	RLF	SLI
5	25	(14, 13)	(11, 12)	(11, 12)	(8, 8)	(10, 9)	$8\frac{1}{2}$	$6\frac{1}{2}$	$6\frac{1}{2}$	3	$4\frac{1}{2}$
	43	(17, 18)	(12, 14)	(11, 15)	(5, 5)	(5, 5)	$12\frac{1}{2}$	8	8	0	0
Ave. total							$10\frac{1}{2}$	$7\frac{1}{4}$	$7\frac{1}{4}$	$1\frac{1}{2}$	$2\frac{1}{4}$
15	36	(22, 22)	(18, 18)	(18, 18)	(17, 16)	(16, 16)	7	3	3	$1\frac{1}{2}$	1
	74	(30, 31)	(26, 26)	(26, 26)	(23, 23)	(23, 24)	$15\frac{1}{2}$	11	11	8	$8\frac{1}{2}$
Ave. total							$11\frac{1}{4}$	7	7	$4\frac{3}{4}$	$4\frac{3}{4}$
25	37	(29, 27)	(26, 25)	(25, 25)	(25, 25)	(25, 25)	3	$\frac{1}{2}$	0	0	0
	77	(37, 35)	(29, 30)	(31, 31)	(28, 28)	(28, 29)	11	$4\frac{1}{2}$	6	3	$3\frac{1}{2}$
Ave. total							7	$2\frac{1}{2}$	3	$1\frac{1}{2}$	$1\frac{3}{4}$
Overall average total							$9\frac{7}{12}$	$5\frac{7}{12}$	$5\frac{9}{12}$	$2\frac{7}{12}$	$2\frac{11}{12}$
Total time (seconds)		32.0	42.8	44.9	80.7	308.8					

TABLE 3. *Results for random 150-node test graphs.*

Lower Bound on χ	d	RND	LF	SL	RLF	SLI
5	10	(9, 9, 8)	(7, 7, 7)	(7, 7, 7)	(6, 6, 6)	(6, 6, 6)
	18	(11, 11, 11)	(10, 10, 10)	(10, 10, 9)	(8, 8, 8)	(9, 8, 8)
	24	(13, 13, 12)	(10, 12, 11)	(11, 11, 11)	(9, 9, 9)	(10, 10, 10)
10	10	(16, 15, 16)	(15, 15, 15)	(15, 15, 15)	(15, 15, 15)	(15, 15, 15)
	18	(16, 17, 16)	(15, 15, 15)	(15, 15, 15)	(15, 15, 15)	(15, 15, 15)
	24	(17, 18, 17)	(15, 16, 16)	(15, 15, 15)	(15, 15, 15)	(15, 15, 15)
15	8	(24, 22, 21)	(24, 22, 21)	(24, 22, 21)	(24, 22, 21)	(24, 22, 21)
	16	(23, 23, 24)	(23, 23, 24)	(23, 23, 24)	(23, 23, 24)	(23, 23, 24)
	24	(24, 24, 24)	(24, 24, 24)	(24, 24, 24)	(24, 24, 24)	(24, 24, 24)
Total time (seconds)		9.8	12.9	12.9	19.0	384.3

three graphs in each 150-node group and two graphs in each 450-node group. The numbers in parentheses indicate the number of colors used by an algorithm to color the first, second, and possibly, third graph of that category. For the graphs where χ was known, the average number of excess colors used by the algorithm was computed for each group and totaled. For example, the RLF algorithm optimally colored each of the 10-colorable 150-node graphs with average degree 11 in table 1 but required, on the average, $4^3/4$ extra colors to color each of the four 15-colorable 450-node graphs in table 2.

The total run time for each algorithm is also included in each table. This figure represents execution time in seconds on an IBM 360-91. It should be noted that such figures are highly dependent on factors unrelated to the inherent efficiency of the algorithm, such as programmer skill and machine characteristics. The time complexity estimates provided earlier are much more rigorous measures of the algorithms' relative speeds. Except for the SLI time in table 3, the run times are in accordance with these theoretical estimates. It is quite possible that the graphs referenced in table 3 have chromatic numbers much higher than the minimum estimate, and that the SLI algorithm was thus induced to attempt and possibly perform a large number of time-consuming interchanges. This example points out the highly variable amounts of time required by most interchange algorithms to color various graphs (a phenomenon also observable in the data of [14]).

The random graphs of table 3 were included only for the purpose of demonstrating that the graphs generated by the technique discussed in section 6 are just as suitable for testing the relative capabilities of graph coloring algorithms as are completely randomly generated graphs. As was pointed out earlier, the data in table 3 cannot be used to draw conclusions about the accuracy of the tested algorithms. From the data in tables 1 and 2, however, we observe that, for the graphs considered, the LF and SL algorithms required about twice as many extra colors to color the graphs as did the RLF and SLI algorithms. Similarly, the RND algorithm required about twice as many extra colors as did the LF and SL algorithms. Significantly, this observation can be made for most of the graphs on an individual basis. The RND algorithm always used more colors than the LF and SL algorithms which, in turn, always used more colors than the RLF or SLI algorithms.

There is not as clear a distinction between the performance of the LF and SL algorithms or the RLF and SLI algorithms. The LF and SL algorithms required virtually the same number of colors on the average and required nearly the same amount of time. While the colorings produced by both the RLF and SLI algorithms for test graphs were, on the average, quite good, the RLF algorithm required substantially less time and used approximately 12 percent fewer excess colors on the 450-node graphs and 3 percent fewer excess colors on the 150-node graphs than did the SLI algorithm. Of the eight 450-node graphs which were not optimally colored by both the RLF and SLI algorithms, the RLF algorithm required the fewest colors for four of the graphs and the most for only one graph.

As a final note, the edge density, $\frac{d}{n}$, of the test graphs did not exceed $1/4$. This results from the fact that for most large-scale practical applications, the edge density of the graphs to be colored is generally small. For instance, the Princeton University exam scheduling graph mentioned in Section 5 had an edge density of approximately $1/6$.

8. Conclusions

From the data presented it is apparent that the RLF algorithm, when not optimal, colored large graphs with substantially fewer colors than did any of the other algorithms that did not involve interchanges. When compared with interchange algorithms, the RLF algorithm was found to produce slightly better colorings in substantially less time. While the RLF and interchange algorithms in general each require $O(n^3)$ time to color an n-node graph, the RLF procedure is unique in that it exhibits $O(n^2)$ time behavior for graphs with low edge density. Thus the RLF algorithm is particularly well suited for use with large-scale practical problems.

The method described in section 6, for generating random graphs with a known chromatic number, was found to produce test data which can be used to determine heuristically a given algorithm's accuracy as well as algorithms' relative capabilities. Previously, published comparison tests have been made only on graphs with unknown chromatic numbers, which rendered impossible any evaluation of an individual algorithm's accuracy and questionable any statement about two algorithms' relative capabilities. In addition, the procedure provides a standard method of generating test data for coloring algorithms; by its use a large graph with known chromatic number may be uniquely constructed from only a few parameters.

In addition to Professor Acton, the author would like to thank Dr. Charles Johnson, Dr. James Lawrence, and Dr. Alan Goldman for their helpful remarks.

9. References

[1] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, Reading, MA, 1974), pp. 364–404.

[2] Bondy, J. A., Bounds for the chromatic number of a graph, Journal of Combinatorial Theory, **7** (1969), pp. 96–8.

[3] Broder, S., Final examination scheduling, Communications of the ACM, Vol. **7**, No. 8 (1964), pp. 494–8.

[4] Brooks, R. L., On coloring the nodes of a network, Proceedings of the Cambridge Philosophical Society, **37** (1941), p. 194.

[5] Brown, J. R., Chromatic scheduling and the chromatic number problem, Management Science, Vol. **19**, No. 4 (1972), pp. 456–463.

[6] Christofides, N., An algorithm for the chromatic number of a graph, The Computer Journal, Vol. **14**, No. 1 (1971), pp. 38–9.

[7] Christofides, N., *Graph Theory—An Algorithmic Approach*, (Academic Press, New York, 1975), pp. 58–78.

[8] Garey, M. R., and Johnson, D. S., The Complexity of Near-optimal Graph Coloring, Journal of the ACM, Vol. **23**, No. 1 (Jan. 1976), pp. 43–9.

[9] Hall, A. D., and Acton, F. S., Scheduling University Course Examinations by Computer, Communications of the ACM, Vol. **10**, No. 4 (April 1967), pp. 235–8.

[10] Johnson, D. S., Approximation Algorithms for Combinatorial Problems, Journal of Computer and System Sciences **9** (1974), pp. 256–78.

[11] Johnson, D. S., Worst Case Behavior of Graph Coloring Algorithms, Proceedings of the 5th Southeast Conference on Combinatorics, Graph Theory, and Computing (1974), pp. 513–27.

[12] Knuth, D. E., *The Art of Computer Programming*, (Addison-Wesley, Reading, MA, 1969), Vol. **2**, pp. 9–18.

[13] Leighton, F. T., A New Solution to the Exam Scheduling Problem, unpublished paper.

[14] Matula, D. W., Marble, G., and Isaacson, J. D., Graph Coloring Algorithms, *Graph Theory and Computing*, Ronald C. Read, editor, (Academic Press, New York, 1972), pp. 109–122.

[15] Matula, D. W., Bounded Color Functions on Graphs, Networks **2** (1972), pp. 29–44.

[16] Ore, Oystein, *The Four Color Problem*, (Academic Press, New York, London, 1967).

[17] Peck, J. E. L., and Williams, M. R., Examination Scheduling, Communications of the ACM, Vol. **9**, No. 6 (1966), pp. 433–4.

[18] Pershin, O. Y., An Algorithm for Determining the Minimum Coloring of a Finite Graph, Engineering Cybernetics, Vol. **11**, No. 6 (1973), pp. 980–5.

[19] Szekeres, G., and Wilf, H. S., An Inequality for the Chromatic Number of a Graph, Journal of Combinatorial Theory, **4** (1968), pp. 1–3.

[20] Welsh, D. J. A., and Powell, M. B., An Upper Bound for the Chromatic Number of a Graph and its Applications to Timetabling Problems, The Computer Journal, **10** (1967), pp. 85–6.

[21] Williams, M. R., The Coloring of Very Large Graphs, Combinatorial Structures and Their Applications—Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications, Gordon and Breach, New York (June 1969), pp. 477–8.

[22] Wood, D. C., A System for Computing University Examination Timetables, The Computer Journal, Vol. **11**, No. 1 (May 1968), p. 41.

[23] Wood, D. C., A Technique for Coloring a Graph Applicable to Large Scale Timetabling Problems, The Computer Journal, **12** (1969), p. 317.

10. Appendix A: Application to Examination Scheduling

The examination scheduling problem is probably the best known of a large class of scheduling problems in applied mathematics and operations research. It consists of scheduling exams such that no individual is required to participate in two or more exams simultaneously. It is usually assumed desirable to schedule the exams such that the total number of time periods required for the examinations is minimized. Sometimes, additional restraints are imposed. Requiring that some examinations be given or not given in specified time periods and scheduling the exams so that a certain subset of the exams will be completed as early as is possible are examples of such restraints.

Consider the following exam scheduling problem.

Exam	Participants	Exam	Participants
1	A_1, A_2, A_3, A_6	7	A_4, A_6
2	B_1, B_2, B_3	8	B_3, B_6, B_7
3	$B_2, B_4,$	9	B_5, B_7, B_8
4	A_1, A_4, B_4	10	A_1, B_1, B_4
5	B_1, B_3	11	B_3, B_7
6	A_2, A_3, B_1	12	A_3, A_5

FIGURE 1

Time Period	Exams that may not be scheduled
1	4, 11, 12
3	3, 7
5	1, 10

FIGURE 2

In addition to the information contained in figures 1 and 2, assume that we also know that exam 2 must be scheduled in time period 1 and that the final schedule must be such that the last exam involving a participant of type A is scheduled as early as possible.

We will now proceed to solve the above scheduling problem utilizing the RLF graph coloring algorithm.

Since exam 2 must be scheduled in time period 1, we will do so and amend figure 2 so that exams incompatible (i.e., may not be scheduled concurrently) with exam 2 will not be scheduled in time period 1. This information is included in figure 3.

Time Period	Exams that may not be scheduled
1	3, 4, 5, 6, 8, 10, 11, 12
3	3, 7
5	1, 10

FIGURE 3

The restriction placed on exams involving type A participants may be satisfied, as far as is possible by heuristic means, by scheduling the exams involving type A individuals first and then, using this information, scheduling the remaining exams. The graph in figure 4 contains the information necessary for the first step.

Node E_i represents exam i and node T_j represents time period j for all i, j. There is an edge between every pair of time period nodes to insure that no two time periods are assigned the same color. An edge is inserted between node E_i and node E_j if and only if exams i and j may not be scheduled simultaneously. Finally, an edge is inserted between node E_i and node T_j if and only if exam i may not be given during time period j.

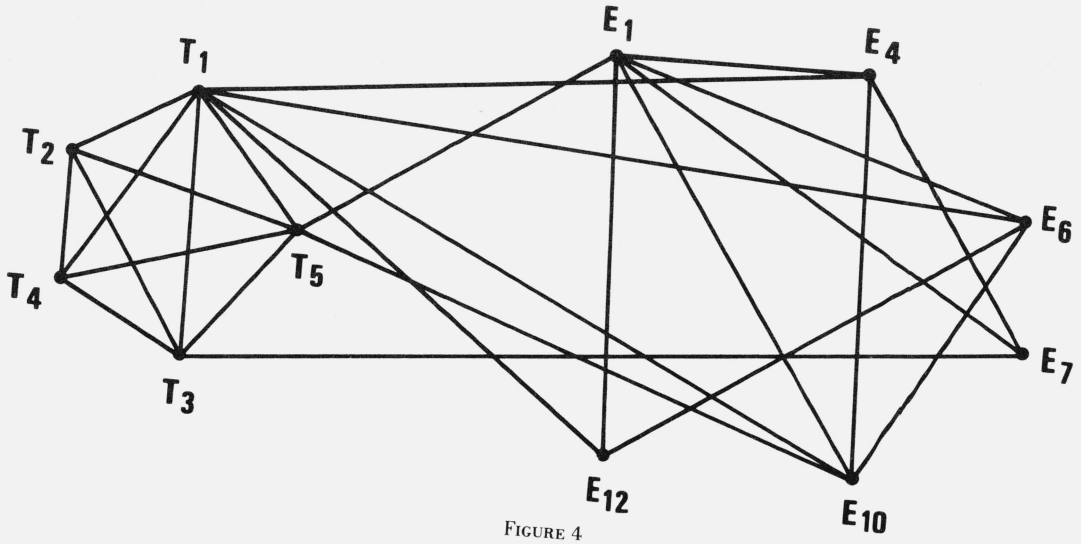

FIGURE 4

To obtain the desired schedule we will color the graph in figure 4 using a slightly modified version of the RLF algorithm. At the beginning of each recursive step, we will first color the earliest uncolored time period node instead of the uncolored node adjacent to the most uncolored nodes. This will guarantee that exams are assigned to the earliest time periods first. We will denote the set of uncolored nodes adjacent to at least one colored node, U_2, by encircling such nodes. Colors, as usual, will be denoted by a number. Uncolored nodes not adjacent to any colored node, nodes in U_1, will not be labeled.

The coloring then proceeds as follows. Node T_1 is colored 1 and nodes T_2, T_3, T_4, T_5, E_4, E_6, E_{10}, and E_{12} are circled. This is illustrated in figure 5.

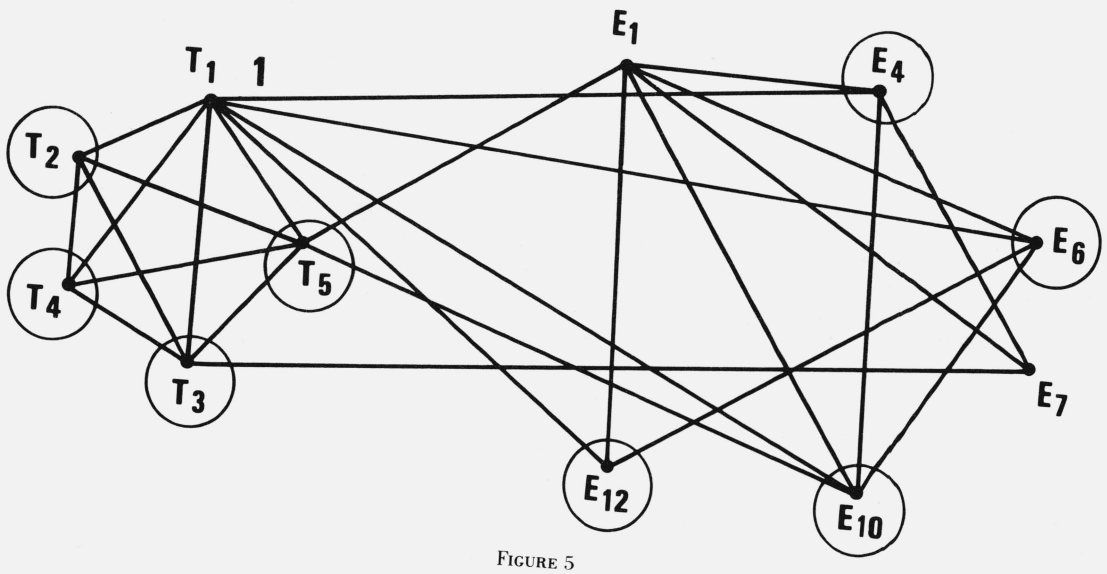

FIGURE 5

Only nodes E_1 and E_7 remain in U_1. Node E_1, is adjacent to five nodes in U_2 while node E_7 is adjacent to only two nodes in U_2. Thus node E_1 is colored 1 and node E_7 circled. This leaves every node either colored or circled (U_1 is empty) and thus we must delete the colored nodes (T_1 and E_1) from the graph and repeat the procedure on the resulting graph starting with color 2. Once T_2 has been assigned color 2 and the appropriate nodes circled, we have the graph in figure 6.

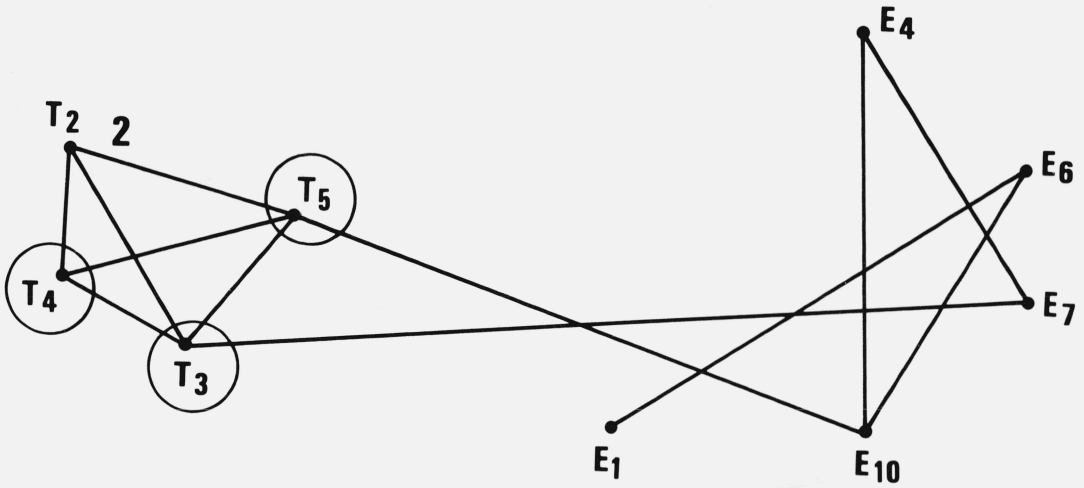

FIGURE 6

Both nodes E_7 and E_{10} are adjacent to one node in U_2 while all other nodes in U_1 are not adjacent to any node in U_2. Since E_7 is connected to only one node in U_1 while $d_{U_1}(E_{10}) = 2$, E_7 is colored 2 and E_4 is circled. This leaves the graph in figure 7.

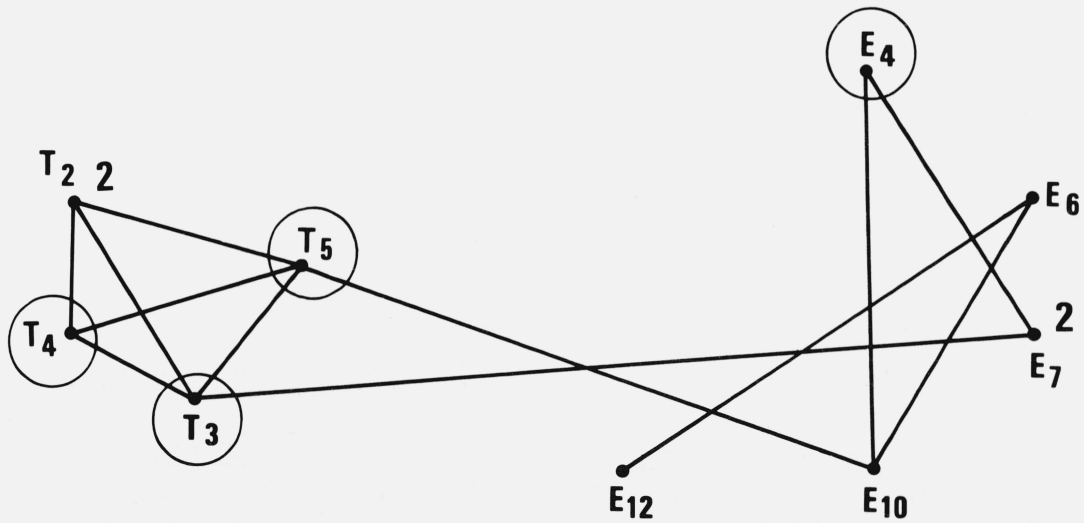

FIGURE 7

Completing the assignment of color 2, E_{10} is assigned color 2, node E_6 is circled and, finally, E_{12} is colored 2. Since U_1 is now empty, we delete the colored nodes and repeat the process on the graph in figure 8 using color 3.

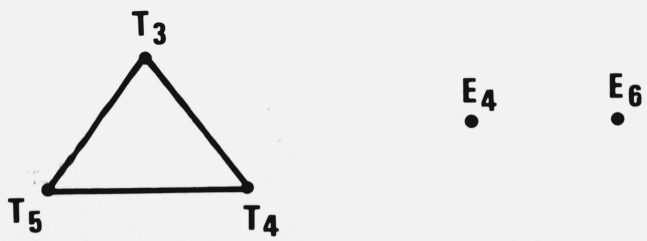

FIGURE 8

This graph is trivially colored by assigning color 3 to nodes T_3, E_4 and E_6; color 4 to T_4; and color 5 to T_5.

The schedule may now be constructed by assigning to each time period those exams which were assigned the color of that time period. Should the number of colors used exceed the number of time period nodes used, the additional colors may be arbitrarily associated with additional time periods (assuming they exist). In this example, however, the number of time period nodes, 5, exceeded the number of colors used, 3, and no such additional assignment of time periods was necessary. The resulting schedule is displayed in figure 9.

Time Period	Exams
1	1
2	7, 10, 12
3	4, 6
4	
5	

FIGURE 9

This completes the scheduling of exams involving type A individuals. We must now schedule the remaining exams taking into account the partial schedule in figure 9 and the information displayed in figure 3. This information is summarized in figure 10.

Time Period	Exams that may not be scheduled
1	3, 5, 8, 11
2	3, 5
3	3, 5

FIGURE 10

Combining this information with that in figure 1, the graph in figure 11 is readily constructed.

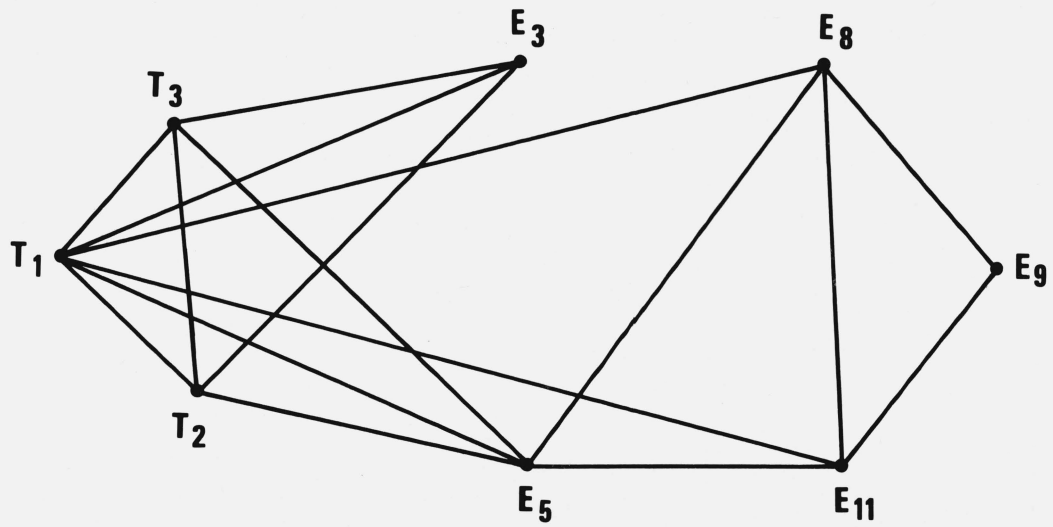

FIGURE 11

All that remains is to color the graph displayed in Figure 11. This is easily done using the RLF algorithm. The final coloring is displayed in figure 12.

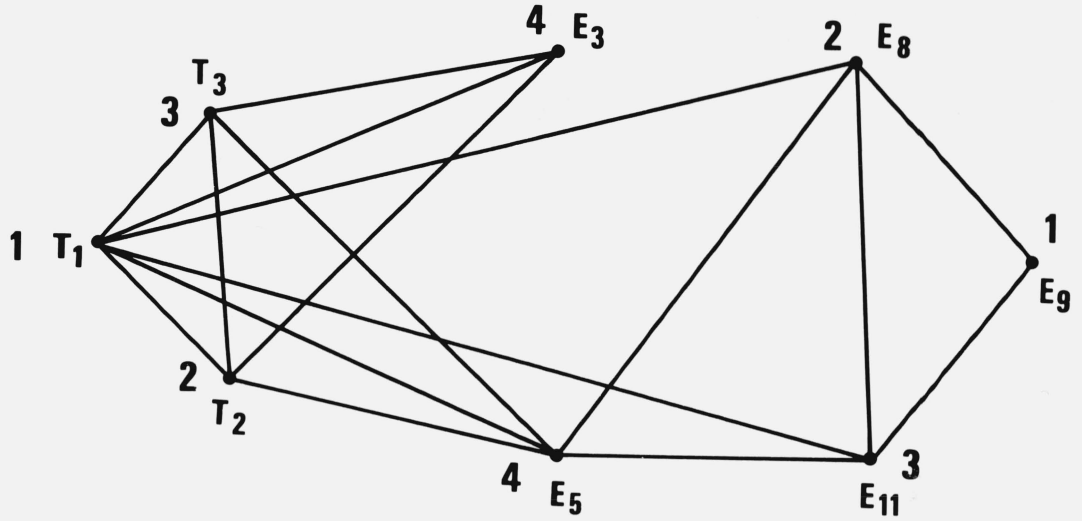

FIGURE 12

Note that, depending on the order in which the nodes were considered, E_{11} could have been assigned color 2 and E_8 color 3 since the two nodes are identical as far as the RLF algorithm is concerned. This illustrates the fact that the final colorings assigned to a graph may, to a small extent, depend on the initial ordering of the nodes (i.e., on the manner in which nodes with identical characteristics are distinguished). Finally, since 4 colors were necessary to color the graph, those exams colored 4 will be scheduled in the first available, unused time period, the fourth time period.

This results in the partial schedule in figure 13.

Time Period	Exam
1	9
2	8
3	11
4	3, 5

FIGURE 13

The completed exam schedule for all exams is displayed in figure 14. In this particular case, the schedule produced is optimal.

Time Period	Exam
1	1, 2, 9
2	7, 8, 10, 12
3	4, 6, 11
4	3, 5

FIGURE 14

11. Appendix B: Computer Implementation of the RLF Algorithm

As was demonstrated in Appendix A, the RLF algorithm can be used to color small graphs easily by hand. For large graphs, however, such a task can only be accomplished reliably by computer. In order to program the RLF algorithm efficiently, the values of d_{U_1} and d_{U_2} must be stored for each node and updated whenever U_1 or U_2 is modified. This is accomplished by defining two arrays, E and F:

$$E(w) \quad \text{is} \quad \left\{ \begin{array}{lll} <0 & \text{if} & w \text{ is colored} \\ <0 & \text{if} & w \in U_2 \\ = d_{U_1}(w) & \text{if} & w \in U_1 \end{array} \right\}.$$

and

$$F(w) \quad \text{is} \quad \left\{ \begin{array}{lll} <0 & \text{if} & w \text{ is colored} \\ = d_{U_1 \cup U_2}(w) & \text{if} & w \in U_2 \\ = d_{U_1 \cup U_2}(w) & \text{if} & w \in U_1 \end{array} \right\}.$$

Initially U_1 consists of every node in G and thus $E(w) = F(w) = d(w)$ for all $w \in V$. If node w' is selected for coloring, then we must modify E and F so that:

$E(w') \leftarrow -1$,

$E(w) \leftarrow -1$ for w adjacent to w',

$E(w) \leftarrow E(w) -$ (the number of nodes in U_1 adjacent to both w and w')

for $w \in U_1$ and nonadjacent to w',

$F(w') \leftarrow -1$, and

$F(w) \leftarrow F(w) - 1$ for w adjacent to w'.

The next node to be colored will, of the nodes in U_1, then have the maximum value of $F(w) - E(w)$ and, if a tie exists, the minimum value of $E(w)$ among those nodes that tied. When U_1 is empty (this corresponds to $E(w) < 0$ for all $w \in V$), the values of E are modified so that $E(w) \leftarrow F(w)$ for all $w \in V$. This corresponds to a reinitialization of the values of E for the subgraph of G induced by the uncolored nodes.

The above operations on E and F can be easily performed by appropriate use of the following subroutine, procedure DELETE. Given an array D and node w', DELETE performs the following operations on D:

$D(w') \leftarrow -1$

$D(w) \leftarrow D(w) - 1$ for w adjacent to w'.

Thus whenever node w' is selected for coloring we may modify E and F by simply performing DELETE on (F, w') and (E, w') as well as on (E, w) for all w in U_1 adjacent to w'. It is not difficult to verify that such a procedure maintains the desired values of E and F.

A complete PL-1 computer program listing of the RLF procedure is included at the end of this appendix. The program is written in subroutine form and assumes that values for CI and CL are provided on input. Array CI serves as an index array to CL, the node adjacency list. For example, the nodes adjacent to the ith node

are sequentially stored in $CL(CI(i-1)+1), CL(CI(i-1)+2), \ldots, CL(CI(i))$. The data are stored in this compact form to minimize the amount of storage required, a very important consideration when working with large graphs.

Each node w in the graph is processed by procedure DELETE, i.e. deleted, $f(w)$ times where $f(w)$ is the color number assigned to w. This claim is easily established by observing that for each new color i introduced, $i \leq f(w)$, w is either colored i or adjacent to a node colored i. In either case w is deleted exactly once. Once w is colored, then it may never subsequently be deleted. Thus exactly

$$\sum_{i=1}^{k} in_i \leq \sum_{i=1}^{k} kn_i = k \cdot \sum_{k=1}^{k} n_i = nk$$

deletions are performed on G, where k is the number of colors used to color G, n_i is the number of nodes colored i and n is the number of nodes in G. Since each deletion requires $O(d)$ time where d is the average degree of a node, all the deletions may be accomplished in $O(kdn)$ time. It is easily checked that all other operations may be accomplished in $O(n^2)$ time. Thus the algorithm requires $O(n^3)$ time and $O(n^2)$ space to color an arbitrary n node graph. However, for those graphs where $kd = O(n)$ or, equivalently, $ke = O(n^2)$, the RLF algorithm consumes only $O(n^2)$ time and $O(n^2)$ space in coloring the graph. As was pointed out in the text, many large-scale practical problems involve graphs for which this property holds and thus may be colored with the RLF algorithm in $O(n^2)$ time.

```
RLF: PROCEDURE (N,CI,CL,COL,C,E,F);
/*                                                                    */
/*  THIS SUBROUTINE COLORS THE N NODE GRAPH DEFINED BY CI AND CL      */
/*     USING THE RLF ALGORITHM.                                       */
/*                                                                    */
/*  ON INPUT:                                                         */
/*                                                                    */
/*     N  IS THE NUMBER OF NODES IN THE GRAPH.                        */
/*     CI  IS THE INDEX VECTOR FOR CL.                                */
/*     CL  IS THE NODE ADJACENCY LIST FOR THE INPUT GRAPH.  FOR       */
/*        EXAMPLE, CL(CI(I-1)+1), CL(CI(I-1)+2), ..., CL(CI(I))       */
/*        CONTAIN, IN SEQUENCE, THE CI(I)-CI(I-1) NODES ADJACENT      */
/*        TO NODE I.                                                  */
/*     COL, C, E AND F  ARE MEANINGLESS.                              */
/*                                                                    */
/*  ON RETURN:                                                        */
/*                                                                    */
/*     N, CI AND CL  ARE UNCHANGED.                                   */
/*     COL  IS THE NUMBER OF COLORS USED TO COLOR THE GRAPH.          */
/*     C  IS THE COLOR FUNCTION ASSIGNED TO THE GRAPH.  FOR EXAMPLE,  */
/*        C(I) IS THE COLOR ASSIGNED TO NODE I.                       */
/*     E AND F  ARE MEANINGLESS.                                      */
/*                                                                    */
   DECLARE (N,CI(*),COL,C(*),E(*),F(*),J,L,I,K) BINARY FIXED (31),
           CL(*) BINARY FIXED (15);
/*                                                                    */
/*  INITIALIZE THE COLOR FUNCTION TO ZERO.                            */
/*                                                                    */
   C=0;
   COL=0;
   J=0;
   L=1;
/*                                                                    */
/*  INITIALIZE THE F VECTOR TO THE NODE DEGREES.                      */
/*                                                                    */
   DO I=1 TO N;
      F(I)=CI(I)-CI(I-1);
   END;
/*                                                                    */
/*  IF THERE ARE ANY UNCOLORED NODES, INITIATE THE ASSIGNMENT OF      */
/*     THE NEXT COLOR.                                                */
/*                                                                    */
   DO WHILE (J<N);
      COL=COL+1;
/*                                                                    */
/*  REINITIALIZE THE E VECTOR.                                        */
/*                                                                    */
      DO I=1 TO N;
         E(I)=F(I);
      END;
/*                                                                    */
/*  SELECT THE NODE IN U1 WITH MAXIMAL DEGREE IN U1.                  */
/*                                                                    */
      DO I=1 TO N;
         IF F(I)>F(L) THEN L=I;
      END;
/*                                                                    */
/*  COLOR THE NODE JUST SELECTED AND CONTINUE TO COLOR NODES WITH     */
```

```
/*      COL UNTIL U1 IS EMPTY.                                          */
/*                                                                      */
        DO WHILE (E(L)>=0);
/*                                                                      */
/*   COLOR NODE AND MODIFY U1 AND U2 ACCORDINGLY.                       */
/*                                                                      */
           CALL DELETE (E,L);
           CALL DELETE (F,L);
           C(L)=COL;
           J=J+1;
           IF CI(L)>CI(L-1) THEN DO I=CI(L-1)+1 TO CI(L);
              IF E(CL(I))>=0 THEN CALL DELETE (E,CL(I));
           END;
/*                                                                      */
/*   FIND THE FIRST NODE IN U1, IF ANY.                                 */
/*                                                                      */
           K=0;
           DO I=1 TO N WHILE (K=0);
              IF E(I)>=0 THEN K=I;
           END;
/*                                                                      */
/*   IF U1 IS NOT EMPTY, SELECT THE NEXT NODE FOR COLORING.             */
/*                                                                      */
           IF K>0 THEN DO;
              L=K;
              DO I=K TO N;
                 IF E(I)>=0 THEN DO;
                    IF F(I)-E(I)>F(L)-E(L) THEN L=I;
                    ELSE IF F(I)-E(I)=F(L)-E(L) & E(I)<E(L) THEN L=I;
                 END;
              END;
           END;
        END;
     END;
END RLF;
/*                                                                      */
/*                                                                      */
/*                                                                      */
DELETE: PROCEDURE (H,M);
/*                                                                      */
/*   THIS SUBROUTINE DECREMENTS THE VALUE OF H FOR M AND NODES          */
/*      ADJACENT TO M.                                                  */
/*                                                                      */
/*   ON INPUT:                                                          */
/*                                                                      */
/*      H   IS EITHER THE E OR F VECTOR FOR THE GRAPH.                  */
/*      M   IS THE NODE TO BE DELETED.                                  */
/*                                                                      */
/*   ON RETURN:                                                         */
/*                                                                      */
/*      H   IS APPROPRIATELY UPDATED.                                   */
/*      M   IS UNCHANGED.                                               */
/*                                                                      */
     DECLARE (H(*),M,P) BINARY FIXED (31);
     H(M)=-1;
     IF CI(M)>CI(M-1) THEN DO P=CI(M-1)+1 TO CI(M);
        H(CL(P))=H(CL(P))-1;
     END;
END DELETE;
```

12. Appendix C: Characterization of Test Graphs

The data provided in tables 4 and 5 of this appendix provide the necessary information to regenerate each of the test graphs used in the preparation of the data included in tables 1 and 2, respectively. Each graph is referenced by its chromatic number, average node degree, and order in which the colorings (or values of X_0) are given. For example, the parameters for the third 150-node graph with chromatic number 5 and average node degree 11 (i.e., the graph that SL 8-colored) are: $n = 150$, $k = 5$, $a = 8401$, $c = 6859$, $m = 84035$, $b_5 = 19$, $b_4 = 60$, $b_3 = 97$, $b_2 = 210$, and $X_0 = 22093$. Using these parameters, it is possible to regenerate the graph by using the procedure described in section 6.

TABLE 4. *Generation Parameters for the Test Graph Used in Table I.*

χ	d	n	k	a	c	m	b	3 values of X_0
5	11	150	5	8401	6859	84035	(19, 60, 97, 210)	0,33289,22093
	19	150	5	8401	6859	84035	(39, 120, 195, 420)	21047,55697,74912
	24	150	5	8401	6859	84035	(58, 180, 292, 630)	78692,83491,52870
10	11	150	10	8401	6859	168070	(10, 0, 0, 12, 0, 0, 25, 0, 2, 4)	8879,64827,78005
	21	150	10	8401	6859	168070	(21, 0, 0, 24, 0, 0, 51, 0, 48)	5293,59845,102567
	29	150	10	8401	6859	168070	(31, 0, 0, 36, 0, 0, 76, 0, 72)	107489,118239,101759
15	12	150	15	8401	6859	252105	(4, 0, 0, 0, 0, 7, 0, 0, 0, 0, 12, 0, 0, 148)	80589,60363,94632
	25	150	15	8401	6859	252105	(9, 0, 0, 0, 0, 15, 0, 0, 0, 0, 24, 0, 0, 297)	220881,67107,198723
	34	150	15	8401	6859	252105	(13, 0, 0, 0, 0, 22, 0, 0, 0, 0, 36, 0, 0, 445)	66684,189309,9534

TABLE 5. *Generation Parameters for the Test Graphs Used in Table 2.*

χ	d avg.	n	k	a	c	m	b	2 values of X_0
5	25	450	5	8401	6859	84035	(175, 540, 877, 1890)	0,41794
	43	450	5	8401	6859	84035	(409, 1260, 2047, 4410)	35428,47927
15	36	450	15	8401	6859	252105	(40, 0, 0, 0, 0, 67, 0, 0, 0, 0, 108, 0, 0, 1336)	36276,213549
	74	450	15	8401	6859	252105	(94, 0, 0, 0, 0, 157, 0, 0, 0, 0, 252, 0, 0, 3118)	161712,160056
25	37	450	25	8401	6859	420175	(13, 0, 0, 0, 0, 0, 0, 0, 0, 27, 0, 0, 0, 0, 0, 0, 0, 0, 40, 0, 0, 0, 0, 1336)	192625,358531
	77	450	25	8401	6859	420175	(31, 0, 0, 0, 0, 0, 0, 0, 0, 63, 0, 0, 0, 0, 0, 0, 0, 0, 94, 0, 0, 0, 0, 3118)	247337,274955