01 May 1990

Algorithms and Probabilistic Bounds for the Chromatic Number of Random Composite Graphs

Jack L. Oakes

Billy E. Gillett
Missouri University of Science and Technology

ALGORITHMS AND PROBABILISTIC BOUNDS FOR THE CHROMATIC NUMBER OF RANDOM COMPOSITE GRAPHS

J. L. Oakes* and B. E. Gillett

CSc-90-6

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401 (314)341-4491

*This report is substantially the Ph.D. dissertation of the first author, completed May, 1990.

ABSTRACT

The composite graph coloring problem (CGCP) is a generalization of the standard graph coloring problem (SGCP). Associated with each vertex is a positive integer called its chromaticity. The chromaticity of a vertex specifies the number of consecutive colors which must be assigned to it.

An exact algorithm for solving the CGCP is presented. The algorithm is a generalization of the vertex-sequential with dynamic reordering approach for the SGCP. It is shown that the method is as effective on composite graphs as its counterpart is on standard graphs. Let $\overline{\chi}(CG_{n,p})$ and $\overline{\chi}(SG_{n,p})$ denote, respectively, the mean chromatic number of a sample of random composite and standard graphs of order n and edge density p. It is demonstrated that the ratio $\overline{\chi}(CG_{n,p})/\overline{\chi}(SG_{n,p})$, depends on p, but, for fixed p, is essentially constant, over the range of values of n for which the algorithms were applied.

Several new heuristic methods for efficiently approximating $\chi(CG_{n,p})$ for large values of n are presented. Of these, the CDsatur and CDsaturI1 algorithms, which are generalizations of the well known Dsatur algorithm, are shown to be very competitive with previously tested procedures.

A known method for calculating probabilistic lower bounds for $\overline{\chi}(SG_{n,p})$ is generalized to produce such bounds for $\overline{\chi}(CG_{n,p})$. Also, a method for estimating the value of $\chi(SG_{n,p})$, is shown to produce probabilistic upper bounds for $\overline{\chi}(SG_{n,p})$. This procedure is then generalized to a method for calculating probabilistic upper bounds for $\overline{\chi}(CG_{n,p})$. The resulting bounds are used to evaluate the actual effectiveness of several heuristic algorithms. It is shown that, for fixed p, although the mean absolute error of the heuristic procedures appears to increase as n is varied from 100 to 1000, the mean relative error remains reasonably constant.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my sincerest appreciation to several individuals who have helped make this accomplishment possible. Dr. Julio S. Leon, Dr. Floyd E. Belk, Mr. James K. Maupin, Mr. J. Steve Earney, and Dr. John M. Cragin, all members of the Missouri Southern State College Administration, supported my request for a two year leave of absence and financial assistance. Dr. Billy E. Gillett, chairman of my committee, has been invaluable with his guidance and encouragement. The suggestions and expressions of support from the other members of my committee, Dr. John B. Prater, Dr. A. Kellam Rigler, Dr. Chung You Ho, and Dr. Leonard F. Koederitz, have been very helpful. Finally, the Computer Science Faculty made the courses I took at UMR informative, challenging, and enjoyable.

I. INTRODUCTION

This document deals with the coloring of the vertices of finite undirected graphs which do not contain multiple edges or loops. This involves assigning colors, which are represented by positive integers, to each vertex so that no two adjacent vertices are assigned the same color. The *chromatic number* of a graph is the smallest possible number of distinct colors which can be used to color the graph. Identifying the chromatic number of a graph is known as the *graph coloring problem (GCP).*

In the past, research in the area of graph coloring has focused primarily on problem instances which require that a single color be assigned to each vertex. This will be referred to as the *standard graph coloring problem (SGCP).*

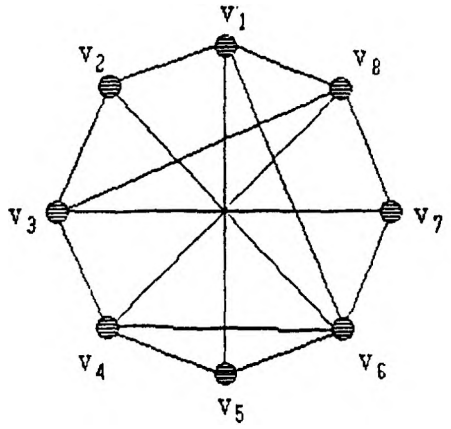
Example 1

Consider the graph in Figure 1. If $f(v_i)$ denotes the color assigned to vertex v_i, the associated table identifies an optimal coloring of the graph. Thus, the graph has a chromatic number of 3.

The number of consecutive colors assigned to each vertex is called its *chromaticity.* The SGCP requires the chromaticity of each vertex to be one and the associated graphs are called *standard graphs.* Graphs which may have vertex chromaticities that are greater than one are called *composite graphs.* The *composite graph coloring problem (CGCP)* is the problem of finding the chromatic number of a composite graph. Obviously, the SGCP can be viewed as a special case of the CGCP.

Example 2

Consider the graph in Figure 2. Let $c(v_i)$ denote the chromaticity of vertex



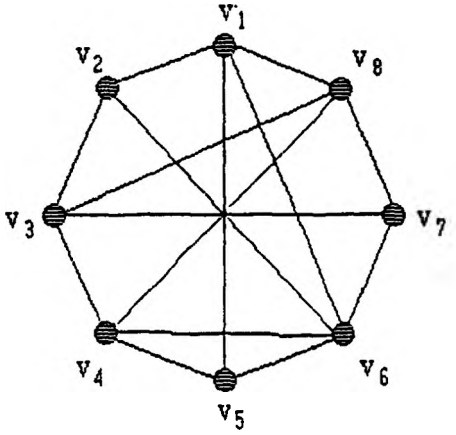
i	$f(v_i)$
1	1
2	2
3	3
4	1
5	2
6	3
7	1
8	2

Figure 1. Standard Graph Coloring Problem

v_i and suppose we require the vertex chromaticities which are specified in the associated table. If we let $F(v_i)$ represent the set of consecutive colors assigned to vertex v_i, then the assignments indicated in the table represent an optimal coloring of the graph. Thus, the graph has a chromatic number of 7.

Applications of the SGCP include the scheduling of exams in the smallest number of time periods such that no individual is required to participate in two exams simultaneously, the storage of chemicals on the minimum number of shelves such that no two mutually dangerous chemicals are on the same shelf, the storage of parse tables, the scheduling of computer processes such that no two which require exclusive access to the same resource are scheduled together, and the assignment of frequencies for radio stations [*Ha68,Le79,Me81,Pe86*]. These problems belong to a class of scheduling and allocation problems in which the tasks to be scheduled are of equal duration and some tasks can not be scheduled concurrently.

The SGCP formulation is not suitable for modeling tasks which have unequal duration. The CGCP formulation allows this to be done. Composite graphs provide an appropriate model for a number of allocation problems including school time



i	$c(v_i)$	$F(v_i)$
1	2	{1,2}
2	2	{5,6}
3	3	{1,2,3}
4	1	{7}
5	2	{5,6}
6	2	{3,4}
7	1	{7}
8	3	{4,5,6}

Figure 2. Composite Graph Coloring Problem

tabling in which lessons may require multiple periods and computer storage economy problems in which one seeks to minimize the memory requirements of a program by deciding which variables can occupy the same locations [*CE83*].

The SGCP is known to be NP-complete [*Ka72*]. Since the CGCP is a generalization of the SGCP, it is also NP-complete. Hence, the existence of a polynomial time exact algorithm for solving all instances of either problem is extremely unlikely. As a result, a number of polynomial time heuristic algorithms for coloring standard graphs have been developed [*Br79,Ch75,LS86,MM72,Wo69*]. Recently, much of this work has been generalized to instances of the CGCP [*CE83,Ro87*]. But, in the case of the SGCP, it has been shown that worst case situations exist for which these heuristic approaches perform rather poorly [*Jo74,Mi76*]. One would expect that such cases also exist for the CGCP.

Although, it is very unlikely that an exact polynomial time algorithm for solving the SGCP exists, algorithms which can solve instances of the problem with up to 75 vertices in less than fifteen minutes of CPU time have been demonstrated [*Br72,Br79,Ch71,Ch75,KJ85,Ko79,SL90*]. Work in the area of exact procedures has some very important potential benefits. These algorithms can be used to supply data

for either supporting or disproving theoretical propositions. Any real world problems which can be modeled as SGCP instances with no more than 70 to 80 vertices may be solved optimally. Past and current trends suggest that advances in technology will continue to dramatically increase CPU speeds. In conjunction with this, exact algorithms will be capable of solving larger problem instances. To date, exact methods for solving the SGCP have not been generalized to the CGCP. Obviously, the potential benefits of such methods are the same as outlined for the SGCP.

This research effort attempted to accomplish three major goals.

1. To design and test exact algorithms for solving the CGCP.
2. To design and test heuristic algorithms for the CGCP which are demonstratively more effective than those which currently exist.
3. To establish a methodology for evaluating the average effectiveness of heuristic algorithms.

The basic approach taken in each case was to build or generalize on the extensive knowledge base which has been established for the SGCP.

Although the reader is assumed to be familiar with graph theoretic terminology and properties, some special color related concepts will be defined and discussed in the next chapter.

Chapter III deals with exact algorithms. It contains a description of several different categories of algorithms for the SGCP. Generalizations of these algorithms are formulated for the CGCP. The most efficient category of exact algorithms for the SGCP is the vertex-sequential with dynamic vertex reordering approach [*Ko79*]. In addition to describing a vertex-sequential with dynamic vertex reordering algorithm for the CGCP, results of implementations for both the SGCP and CGCP are presented and analyzed.

The focus of Chapter IV is heuristic algorithms. A review of the literature is made, several new algorithms for the CGCP are described, and the results of implementations of selected algorithms for both the SGCP and CGCP are presented.

It is known that the computation time required by exact algorithms will be unacceptable for all but relatively small problem instances. Heuristic algorithms are

designed to be efficient but, typically, at the expense of not being able to effectively approximate the optimal solution. They can easily be compared among themselves; however, since the chromatic number of a large graph is usually not known, it is difficult to measure the actual effectiveness of a given heuristic. Chapter V describes a methodology, which is based on the application of probability theory, for calculating both an upper and lower bound for the mean chromatic number of a sample of either SGCP or CGCP instances. It is shown that the validity of this methodology is supported by the data base produced with the exact algorithms of Chapter III. The effectiveness of the heuristic algorithms presented in Chapter IV are then analyzed.

The goal of Chapter VI is to summarize the material presented in Chapters III, IV, and V, and to draw final conclusions.

Finally, Chapter VII describes some areas for further research.

II. DEFINITIONS

A. GRAPH THEORETIC DEFINITIONS

A *graph*, $G=(V,E)$, is a finite set of vertices, $V=\{v_1,v_2..,v_n\}$, together with a set of edges, $E=\{e_1,e_2,..,e_m\}$. An *undirected graph* is a graph for which the edges are not directionally oriented. The notation, $e_k=<v_i,v_j>$, will denote that e_k connects v_i and v_j. Since graph coloring deals primarily with undirected graphs which contain no loops and no more than one edge between any pair of vertices, the term *graph* will be used for such graphs.

Two vertices are said to be *adjacent* if they are connected by an edge. The *degree* of a vertex, denoted $\deg(v_i)$, is the number of edges which contain v_i as an endpoint. The *order* of a graph, denoted $|G|$, is the number of vertices in V.

An *independent set* of a graph, $G=(V,E)$, is a set of vertices of G no two of which are adjacent. An independent set $U\subset V$ is a *maximal independent set*, denoted MIS, if there is no independent set S such that $U\subseteq S$. The *independence number* of G, denoted $\alpha(G)$, is the number of vertices in the largest MIS of G.

A *completely connected set* of $G=(V,E)$ is a set of vertices of G, each pair of which are adjacent. A maximally completely connected set is said to be a *clique*. The *clique number*, denoted $\mu(G)$, is the number of vertices in the largest clique of G.

A *subgraph* of $G=(V,E)$ is a graph $G'=(V',E')$ such that $V'\subset V$, $E'\subseteq E$, and for all $<v_i,v_j>\in E'$, $v_i,v_j\in V'$. G' is said to be an *induced subgraph* of G if $E'=\{<v_i,v_j>\mid <v_i,v_j>\in E\wedge v_i,v_j\in V'\}$.

B. GRAPH COLORING DEFINITIONS

A *coloring* of a graph, G, is an assignment of colors to all the vertices of G so that no adjacent vertices are assigned the same color. A coloring is defined by a coloring function, f for the SGCP and F for the CGCP, on the vertices of G.

$f(v_i)=j$, where j is a positive integer, if and only if vertex v_i is colored with color j. $F(v_i)=\{j_1,j_2,..,j_k\}$, where $j_r-j_{r-1}=1$, if and only if vertex v_i is assigned the set of colors $\{j_1,j_2,..,j_k\}$.

A graph is said to be *k-colorable* if it can be colored with k colors. The *chromatic number* of a graph $G=(V,E)$, denoted by $\chi(G)$, is the minimum number of colors required to color G. If $\chi(G)=q$ then G is said to be *q-chromatic*.

Associated with every k-coloring of a graph $G=(V,E)$ is a partitioning of the elements of V into a set of sets $S=\{S_1,S_2,..,S_k\}$ where $S_j=\{v_i|f(v_i)=j\}$, if G is a standard graph, and $S_j=\{v_i|j=\min F(v_i)\}$, if G is a composite graph. Two k-colorings are said to be *redundant* if their respective partitionings are equivalent sets.

The *chromaticity* of a vertex v_i, denoted $c(v_i)$, is the number of consecutive colors which must be assigned to that vertex. In the SGCP, the chromaticity of each vertex defaults to 1. The *total vertex chromaticity* of a graph G, denoted c(G), is the sum of the chromaticities of all the vertices in G. The maximum vertex chromaticity of all the vertices in G will be denoted MaxChrom(G).

The *adjacent chromatic degree* of a vertex v_i, denoted AdjChromDeg(v_i), is the sum of the chromaticities of all vertices adjacent to v_i. The *chromatic degree* of a vertex v_i, denoted ChromDeg(v_i), is the sum of $c(v_i)$ and AdjChromDeg(v_i). The *chromatic degree* of a graph G, denoted ChromDeg(G), is the maximum vertex chromatic degree.

A *random graph*, $G_{n,p}=(V,E)$, is a graph of order n with the property that for all $v_i,v_j \in V$, Prob$\{<v_i,v_j> \in E\}=p$, where $0 \le p \le 1$. The probability, p, is said to be the *edge density* of the graph. Obviously, the number of edges, m, in $G_{n,p}$ is a binomially distributed random variable with

$$E(m)=p\frac{n(n-1)}{2} \quad and \quad Var(m)=p(1-p)\frac{n(n-1)}{2} \qquad (1)$$

A *standard random graph*, $SG_{n,p}=(V,E)$, is a random graph of order n and edge density p with the property that for all $v_i \in V$, $c(v_i)=1$. A *type 1 composite random graph*, $CG_{n,p,1}=(V,E)$, is a random graph of order n and edge density p with the property that the vertex chromaticities are given by n independent truncated Poisson random variables, with parameter $q=1$. Thus, for all $v_i \in V$,

$$Prob\{c(v_i)=k\}=\frac{1}{(e-1)k!} \quad k=1,2,... \tag{2}$$

A *type 2 composite random graph*, $CG_{n,p,2}=(V,E)$, is a random graph of order n and density p with the property that for all $v_i \in V$,

$$\begin{aligned} Prob\{c(v_i)=1\}&=0.75 \\ Prob\{c(v_i)=2\}&=0.25 \end{aligned} \tag{3}$$

III. EXACT GCP FORMULATIONS

A. (0,1) INTEGER PROGRAMMING FORMULATION FOR THE SGCP

The SGCP can be formulated as a (0,1) integer programming problem [*Ch75*]. Let q be an upper bound on the chromatic number of a standard graph G. Let n equal the number of vertices and m the number of edges in G. Let $A=[a_{ij}]$ be the adjacency matrix of the graph with diagonal elements set to zero. Let $B=[b_{ij}]$ be an $n \times q$ matrix which allocates vertices to colors as follows.

$$b_{ij} = \begin{cases} 1 & \text{if } f(v_i)=j \\ 0 & \text{otherwise} \end{cases} \quad i=1,2,..,n \quad j=1,2,..,q \tag{4}$$

For the SGCP two constraints must be modeled. First, each vertex must be assigned a single color. This can be represented as follows.

$$\sum_{j=1}^{q} b_{ij} = 1 \quad i=1,2,..,n \tag{5}$$

Second, no two adjacent vertices may be assigned the same color. The following condition assures this.

$$L(1-b_{ij}) - \sum_{k=1}^{n} a_{ik} b_{kj} \geq 0 \quad i=1,2,..,n \quad j=1,2,..,q \tag{6}$$

where L is any positive integer greater than or equal to n. If vertex v_i is colored with color j (i.e. $b_{ij} = 1$) then the first term of the condition is zero. This requires the second term to also be zero in order to satisfy the inequality; i.e., no adjacent vertex may have the same color. If vertex v_i is not colored j ($b_{ij} = 0$) then the first term becomes L and, since the second term is for sure less than L, the inequality will be satisfied.

The objective is to use the minimum number of colors. This is modeled by associating with each color j a penalty p_j defined by

$$\begin{cases} p_1 = 1 \\ p_j > h * p_{j-1}, \quad j = 2,3,..,q \end{cases} \tag{7}$$

where h is an upper bound on the largest number of vertices which can be colored with any one color; i.e., h is any number greater than the independence number $\alpha(G)$. In the absence of a better bound, h can be set equal to n at no extra computational cost.

A SGCP formulation can be expressed as follows.

$$\begin{aligned} &Min \quad z = \sum_{j=1}^{q} p_j \sum_{i=1}^{n} b_{ij} \\ &s.\ t. \quad \sum_{j=1}^{q} b_{ij} = 1 \quad i = 1,2,..,n \\ &\qquad L(1 - b_{ij}) - \sum_{k=1}^{n} a_{ik} b_{kj} \geq 0 \quad i = 1,2,..,n \quad j = 1,2,..,q \\ &\qquad b_{ij} \in (0,1) \quad i = 1,2,..,n \quad j = 1,2,..,q \end{aligned} \tag{8}$$

The minimization of the objective function ensures that a color j + 1 is never used for coloring vertices, if colors 1 to j are sufficient for a feasible coloring. The vertex to colors allocation matrix $[b_{ij}]^{opt}$ which is the solution to the above (0,1) integer programming problem can be used to identify the optimal coloring and the number of colors used is the chromatic number of the graph.

This formulation requires nq variables and n(q + 1) constraints. Thus, a 100 vertex graph with an upper bound on the chromatic number of 25 would need 2500 variables and 2600 constraints.

B. (0,1) INTEGER PROGRAMMING FORMULATION FOR THE CGCP

The previous (0,1) integer programming formulation can be generalized to the CGCP. All the notation and associated definitions introduced in that formulation are valid for the generalization with the following exception.

$$b_{ij} = \begin{cases} 1 & \text{if } j = \min F(v_i) \\ 0 & \text{otherwise} \end{cases} \quad i = 1,2,..,n \quad j = 1,2,..,q \tag{9}$$

The CGCP formulation can be expressed as

$$
\begin{aligned}
Min \quad & z-\sum_{\substack{j-1 \\ q-c(v_i)+1}}^{q} p_j \sum_{i-1}^{n} b_{ij} \\
s.\ t. \quad & \sum_{j-1} b_{ij}-1 \quad i-1,2,..,n \\
& L(1-b_{ij})-\sum_{k-1}^{n} a_{ik} \sum_{l-j-c(v_j)+1}^{j+c(v_j)-1} b_{kl} \geq 0 \quad i-1,2,..,n \quad j-1,2,..,q-c(v_i)+1 \\
& b_{ij} \varepsilon (0,1) \quad i-1,2,..,n \quad j-1,2,..,q
\end{aligned}
\tag{10}
$$

where L is any positive integer greater than or equal to 2n*MaxChrom(G).

This formulation requires n(q+1)-c(G) variables and n(q+2)-c(G) constraints.

Example 3

Consider the composite graph in Figure 3. Assume that a known upper bound on the chromatic number of the graph is 5.

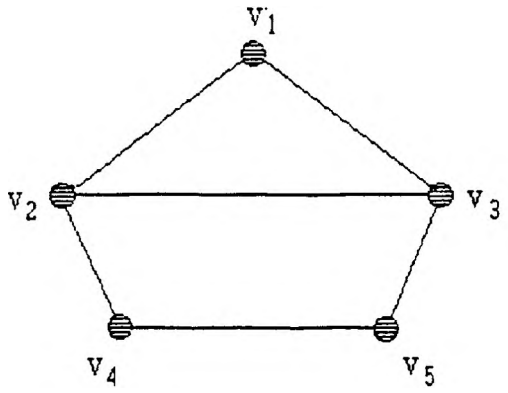
$$
A-\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \qquad B-\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} & 0 \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & 0 & 0 \\ b_{51} & b_{52} & b_{53} & b_{54} & 0 \end{bmatrix} \tag{11}
$$

A (0,1) integer programming formulation is displayed below the graph.

C. MIXED (0,1) INTEGER PROGRAMMING FORMULATION FOR THE CGCP

This formulation was developed by Roberts [*Ro87*] specifically for the CGCP. It can easily be adapted to the SGCP by assigning all vertex chromaticity parameters the value 1.

In the definition of graph coloring, the lowest integer which represents a color



i	$c(v_i)$
1	2
2	2
3	1
4	3
5	2

$$
\begin{aligned}
\textit{Min } z = \quad & (b_{11}+b_{21}+b_{31}+b_{41}+b_{51}) + \\
& 3(b_{12}+b_{22}+b_{32}+b_{42}+b_{52}) + \\
& 9(b_{13}+b_{23}+b_{33}+b_{43}+b_{53}) + \\
& 27(b_{14}+b_{24}+b_{34}+b_{54}) + \\
& 81(b_{35})
\end{aligned}
$$

$$
\begin{aligned}
\textit{s. t.} \quad & b_{11}+b_{12}+b_{13}+b_{14} = 1 \\
& b_{21}+b_{22}+b_{23}+b_{24} = 1 \\
& b_{31}+b_{32}+b_{33}+b_{34}+b_{35} = 1 \\
& b_{41}+b_{42}+b_{43} = 1 \\
& b_{51}+b_{52}+b_{53}+b_{54} = 1
\end{aligned}
$$

$$
\begin{aligned}
& 30(1-b_{11})-(b_{21}+b_{22})-(b_{31}+b_{32}) \geq 0 \\
& 30(1-b_{12})-(b_{21}+b_{22}+b_{23})-(b_{32}+b_{33}) \geq 0 \\
& 30(1-b_{13})-(b_{22}+b_{23}+b_{24})-(b_{33}+b_{34}) \geq 0 \\
& 30(1-b_{14})-(b_{23}+b_{24})-(b_{34}+b_{35}) \geq 0
\end{aligned}
$$

etc.

Figure 3. Composite Graph Coloring Problem

is 1. In this formulation the lowest possible color is represented by 0. The following parameters and variables are used.

n = the number of vertices
m = the number of edges
$J_i = \{j | <v_i,v_j> \in E \wedge j>i\}$, $i=1,2,..,n$
$y_i = \max \{k | k \in F(v_i)\}$, $i=1,2,..,n$
$y_0 = \max \{y_i | i=1,2,..,n\}$
$K > \chi(G)$

A CGCP formulation can be expressed as follows.

$$\begin{aligned} &\textit{Min} \quad z-y_0 \\ &\textit{s. t.} \quad y_i \le y_0 \quad i=1,2,..,n \\ &\quad \begin{cases} y_i-y_j \ge c_i \\ \vee \\ y_i-y_j \le -c_j \end{cases} \quad i=1,2,..,n \quad j \in J_i \\ &\quad y_i \ge 0 \wedge \textit{integer} \quad i=1,2,..,n \end{aligned} \tag{13}$$

The either/or expressions can be eliminated by introducing (0,1) variables, δ_{ij}, for each i and j. The formulation is then stated as follows.

$$\begin{aligned} &\textit{Min} \quad z-y_0 \\ &\textit{s. t.} \quad y_i \le y_0 \quad i=1,2,..,n \\ &\quad y_i-y_j+K\delta_{ij} \ge c_i \quad i=1,2,..,n \quad j \in J_i \\ &\quad y_i-y_j+K\delta_{ij} \le K-c_j \quad i=1,2,..,n \quad j \in J_i \\ &\quad y_i \ge 0 \wedge \textit{integer} \quad i=1,2,..,n \\ &\quad \delta_{ij} \in (0,1) \quad i=1,2,..,n \quad j \in J_i \end{aligned} \tag{14}$$

This formulation contains 2m constraints, m (0,1) variables, and n integer variables. But, because of the nature of the constraints and their interaction with the objective function, the n integer variables y_i can be treated as continuous variables and the resulting optimal solution will still contain only integer values. By modeling them as continuous variables, the solution process should be simplified. The major drawback with this formulation is not its size, but that the (0,1) variables δ_{ij} are not represented in the objective function. The standard implicit enumeration procedures for solving mixed (0,1) integer programs use the coefficients of the (0,1) variables

in the objective function to predict the effect of changes in those variables and, thus, for fathoming. Hence, in order for this formulation to be useful, a way for measuring the effect of changes in these variables needs to be developed.

D. BASIC VERTEX-SEQUENTIAL ALGORITHM FOR THE SGCP

This formulation of the SGCP is due to Korman [*Ko79*]. Assume that the vertices of a standard graph, G = (V,E), have been ordered $v_1,v_2,..,v_n$. An initial feasible coloring of G can be produced by sequentially assigning the smallest possible color to each vertex in the following manner. Vertex v_1 is assigned color 1. If vertices $v_1,v_2,..,v_k$ are assigned colors $f(v_1),f(v_2),..,f(v_k)$ then

$$f(v_{k+1}) = \min\{j : \forall i \le k,\ <v_{k+1},v_i> \in E \Rightarrow j \neq f(v_i)\} \tag{15}$$

This initial feasible coloring will require q_1 colors for some integer value of q_1. It is very unlikely that this coloring will be optimal; but, q_1 can serve as an upper bound for the next stage of the algorithm.

Stage 2 involves backtracking in an attempt to find a feasible solution which requires $q_2 < q_1$ colors. Backtracking proceeds to vertex v_k, where v_{k+1} is the vertex of lowest index which has been assigned color q_1. An attempt is then made to color v_k with the smallest alternative color which is greater than its current color, $f(v_k)$. Obviously, no color larger than q_1-1 needs to be considered. Randall Brown [*Br72*] proved that regardless of the current feasible coloring, no color which exceeds p+1 where p is the number of colors used on $v_1,v_2,..,v_{k-1}$ needs to be considered. Thus, an upper bound on the colors which must be considered is given by

$$\min\{q_1\text{-}1,p+1\}$$

If no alternative color exists, backtracking proceeds to v_{k-1}. If a feasible alternative color is found, v_k is recolored and, as before, each succeeding vertex is then colored with its smallest feasible color until either v_n is colored or a vertex v_r is encountered which requires the use of color q_1.

In the former case, a feasible coloring using $q_2 < q_1$ colors has been found. Hence, q_1 is replaced with q_2 in the above procedure and backtracking is initiated

again in an attempt to find a coloring using $q_3 < q_2$ colors. In the latter case, backtracking from v_k occurs and then the search proceeds forward as before. When backtracking reaches v_1, the algorithm terminates with $\chi(G)$ being the smallest integer for which a feasible coloring of G has been obtained. The current values of f represent an optimal coloring of the graph.

Example 4

Consider the standard graph in Figure 4. Application of the Basic Vertex-Sequential Algorithm produces the associated search tree. Since no further backtracking is possible, $\chi(G) = 3$ and an optimal coloring is as is indicated.

E. BASIC VERTEX-SEQUENTIAL ALGORITHM FOR THE CGCP

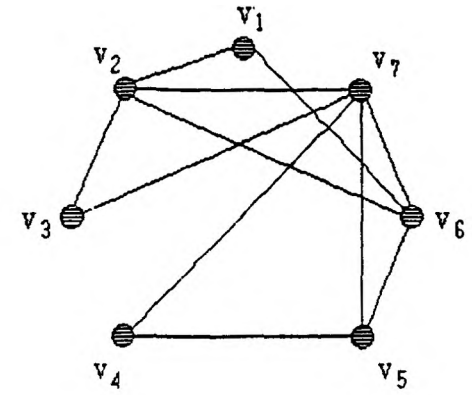
Stage 1 of this algorithm is identical to the algorithm for the SGCP with the exception that a set of consecutive colors is assigned to each vertex. The order of the set must equal the chromaticity of the vertex and the smallest color must be no larger than necessary such the intersection of the set with sets previously assigned to adjacent vertices is empty. Stage 2 can proceed in a manner which is also very similar except that sets of colors are reassigned instead of single colors. Also, when backtracking, an upper bound on the smallest color which needs to be considered for assignment to vertex v_k is given by

$$\min\{q_1 - c(v_k), p + \text{MaxChrom}(G)\}$$

where q_1 is the current upper bound for $\chi(G)$ and p is the number of colors used to color $v_1, v_2, .. v_{k-1}$.

Example 5

Consider the composite graph in Figure 5. Suppose the indicated vertex chromaticities are required. Application of the Basic Vertex-Sequential Algorithm produces the associated search tree. Although further backtracking is possible at level 3, $\chi(G) = 7$. Therefore, it is not shown. Note that an optimal coloring is also specified.



i	$f(v_i)$
1	1
2	2
3	3
4	3
5	2
6	3
7	1

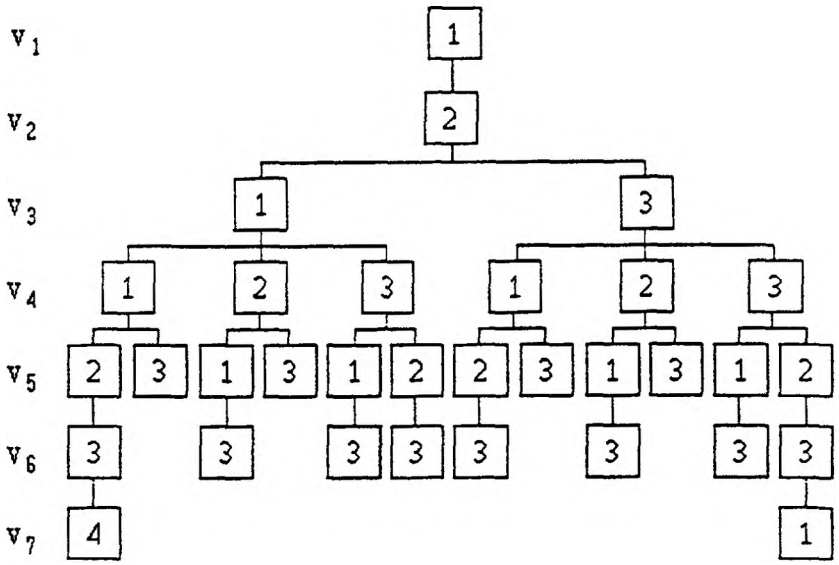
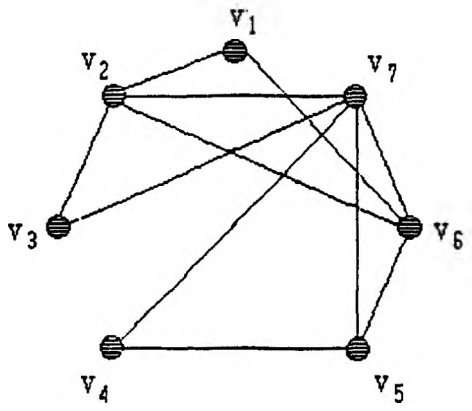


Figure 4. Basic Vertex-Sequential Algorithm for the SGCP

Before proceeding, consider further the above upper bound for the beginning color of any feasible color set for vertex v_k. The fact that q_1-$c(v_k)$ is an upper bound is obvious. It will now be shown that p+MaxChrom(G) is also an upper bound. The proof is a simple application of the following theorem which was proved by Randall Brown [*Br72*].

Let $X=\{x_1,x_2,..,x_n\}$ be a set of n variables associated with n objects $v_1,v_2,..,v_n$.



i	$c(v_i)$	$F(v_i)$
1	2	{1,2}
2	2	{3,4}
3	1	{1}
4	2	{4,5}
5	3	{1,2,3}
6	2	{5,6}
7	1	{7}

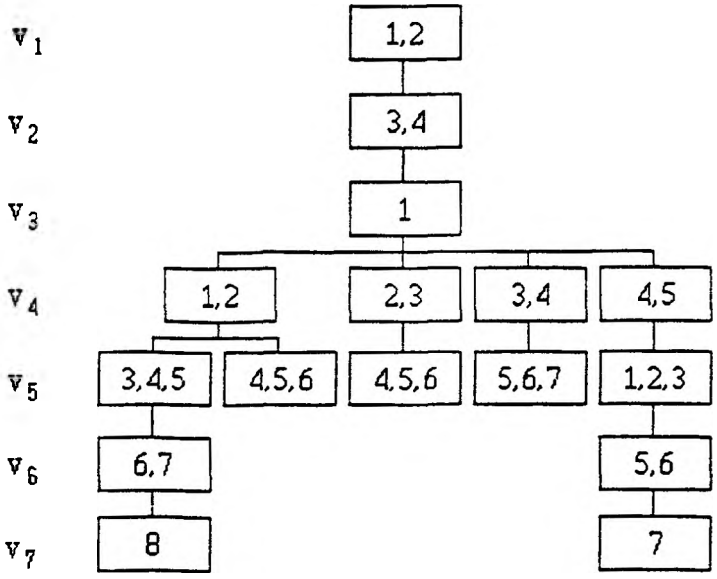


Figure 5. Basic Vertex-Sequential Algorithm for the CGCP

Define a solution to be a set of sets $S_X=\{X_1,X_2,..,X_m\}$ where $X_i \subseteq X$, $X_i \cap X_j = \varnothing$ for $i \neq j$, and $X_1 \cup X_2 \cup .. \cup X_m = X$ together with a set of attributes $A=\{a_1,a_2,..,a_m\}$ where a_j is assigned to each member of X_j. The set A is said to be an attribute set for S_X. Suppose $B=\{b_1,b_2,..,b_m\}$ is another attribute set for S_X. If A=B and the members of A are considered to be indistinguishable then A and B are said to be *redundant*. A partial solution is defined to be the assignment of attributes $a_1,a_2,..,a_p$ to subsets of variables of $\{x_1,x_2,..,x_{k-1}\} \subseteq X$. Two partial solutions are said to be redundant if one

can be derived from the other by interchanging two or more indistinguishable attributes.

Theorem 1. Let $A=\{a_1,a_2,..,a_m\}$ be a set of attributes that are indistinguishable among themselves. When a partial solution $\{x_1,x_2,..,x_{k-1}\}\subseteq X$ and associated attribute set $A_p=\{a_1,a_2,..,a_p\}\subseteq A$ is being extended to produce new partial solutions for $\{x_1,x_2,..,x_{k-1},x_k\}$, redundant solutions may be prevented by considering the addition of only a_{p+1} to A_p.

Since colors are indistinguishable attributes, this theorem can be applied to the SGCP to justify the following corollary.

Corollary 1.1. Let $f(v_1),f(v_2),..,f(v_{k-1})$ be a partial coloring of G which uses p colors. If no redundant coloring is to be generated, then the color assigned to v_k can be no larger than $p+1$.

In order to apply Theorem 1 to the CGCP, the concept of *indistinguishable* colors must be more clearly established. Assume that $<v_k,v_{k+1}>\in E$. The color $p+1$ is indistinguishable from the color $p+2$ because coloring v_k with $p+1$ would require in the worst case the use of $p+2$ to color v_{k+1}. On the other hand, if we first colored v_k with $p+2$, then in the worst case v_{k+1} could be colored with $p+1$. Thus, the same two new colors will have been used. But, if $c(v_k)=2$ and $c(v_{k+1})=3$ then $\{p+1,p+2\}$ is not indistinguishable from $\{p+2,p+3\}$ because coloring v_k with $\{p+1,p+2\}$ would in the worst case require using $\{p+3,p+4,p+5\}$ to color v_{k+1}. However, coloring v_k with $\{p+2,p+3\}$ could require the use of $\{p+4,p+5,p+6\}$ to color v_{k+1}. Obviously, the same new colors are not used. Similarly, $\{p+1,p+2\}$ is not indistinguishable from $\{p+3,p+4\}$. But, if v_k is assigned $\{p+4,p+5\}$ then in the worst case v_{k+1} would require the use of $\{p+1,p+2,p+3\}$. Thus, $\{p+1,p+2\}$ is indistinguishable from $\{p+4,p+5\}$. This illustration can easily be generalized to justify the following corollary.

Corollary 1.2. Let $F(v_1),F(v_2),..,F(v_{k-1})$ be a partial coloring of G which uses p colors. If no redundant coloring is to be generated, then the color set assigned to v_k can begin with a color which is no larger than p+MaxChrom(G).

The following example illustrates the preceding corollaries. More importantly, it demonstrates that the CGCP is a much more difficult problem than the SGCP.

Example 6

Consider Figure 6 and Figure 7. Although both problems have the same base graph, the fact that the composite graph has a maximum vertex chromaticity of 3 increases the size of the associated search space exponentially.

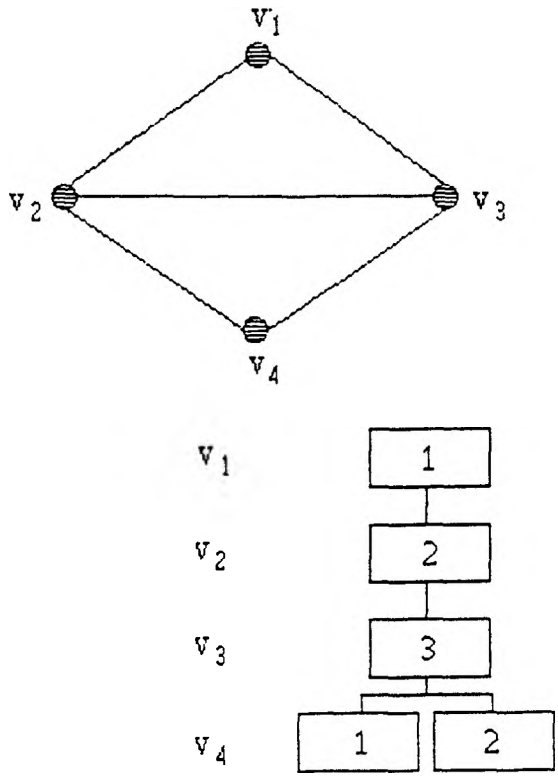
F. VERTEX-SEQUENTIAL WITH DYNAMIC REORDERING FOR THE SGCP

This formulation is primarily due to Korman [*Ko79*]. In the basic algorithm, it was shown that, when backtracking, vertex v_k need never be assigned a color which is smaller than its current color or larger than

$$\min\{q_1-1,p+1\}$$

where q_1 is a current upper bound for $\chi(G)$ and p is the number of colors used to color $v_1,v_2,..,v_{k-1}$.

If this property is considered when creating the ordering for Stage 1, it is possible to increase the level of fathoming. Specifically, v_1 need never be colored any color other than 1. If v_2 is adjacent to v_1, it need never be colored any color other than 2. Repeating this logic, if vertices $v_1,v_2,..,v_{k-1}$ constitute a clique, there is only one possible color which could be assigned to each of them during the execution of the procedure. Thus, the algorithm can terminate when all backtracking from vertex v_k is completed. Hence, an effective way to order the vertices for Stage 1 would be to arrange the first $\mu(G)$ vertices so that they constitute a maximal clique of the graph. Carrying the idea forward, remaining vertices can be ordered so that for any vertex v_k, it is adjacent to more of the vertices $v_1,v_2,..,v_{k-1}$ than any other



i	$f(v_i)$
1	1
2	2
3	3
4	1

Figure 6. Corollary 1.1

currently unordered vertex with ties being broken by choosing the vertex of largest degree. Since this approach will usually generate a near maximal clique anyway, it is reasonable to use it from the beginning instead of identifying a maximal clique.

It is also possible to significantly improve the Basic Vertex-Sequential Algorithm by incorporating the concept of a dynamic reordering of the as yet uncolored vertices as the algorithm progresses. The vertex to be colored next is selected to be the one which has the smallest possible number of feasible color assignments available to it; i.e., is adjacent to the largest number of differently colored vertices. Thus, forward branchings which will not lead to an improved solution will terminate as soon as possible. If ties occur Korman suggested breaking

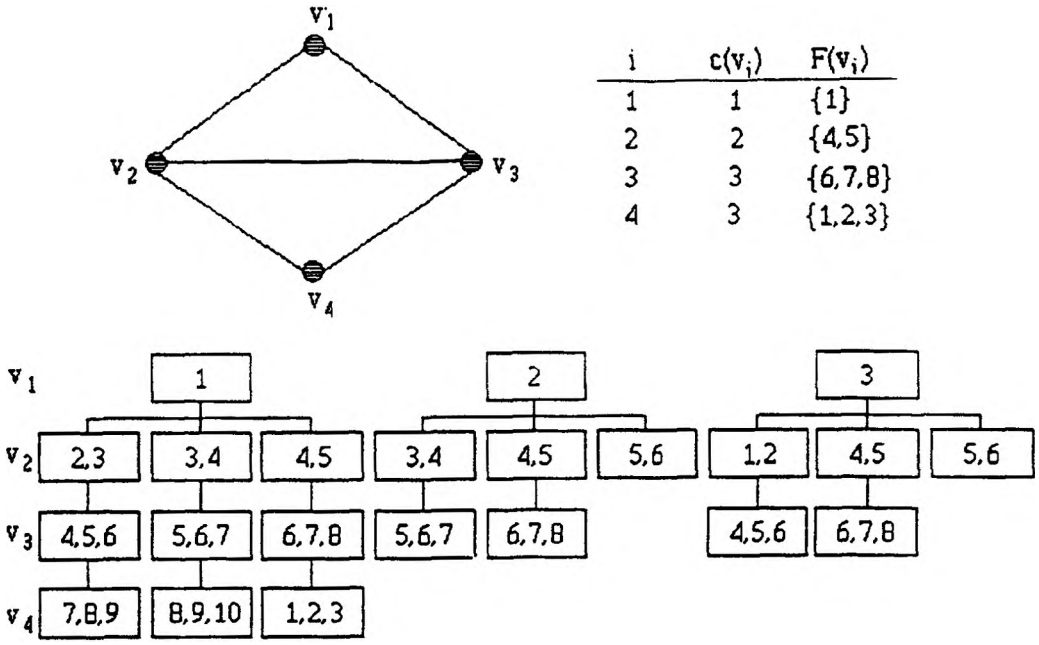


Figure 7. Corollary 1.2

them by selecting the vertex of largest degree. Lin [*Li89*] showed that a more effective approach is to select the vertex which is adjacent to the largest number of as yet uncolored vertices.

If $f(v_1),f(v_2),..,f(v_{k-1})$ is a partial coloring of $G=(V,E)$, the *colored degree* of vertex v_j, $j=k,k+1,..,n$, denoted $cdeg(v_j)$, is defined to be the number of differently colored vertices which are adjacent to v_j. The *uncolored degree* of v_j, denoted $ucdeg(v_j)$, is defined to be the number of as yet uncolored vertices adjacent to v_j.

Example 7

Consider again the SGCP in Figure 4. Table I illustrates an application of the

TABLE I

VERTEX-SEQUENTIAL WITH DYNAMIC REORDERING FOR THE SGCP

Level	Assignment	v_1	v_2	v_3	v_4	v_5	v_6	v_7
0		(0,2)	(0,4)	(0,2)	(0,2)	(0,3)	(0,4)	(0,5)
1	$f(v_7)=1$	(0,2)	(1,3)	(1,1)	(1,1)	(1,2)	(1,3)	-
2	$f(v_2)=2$	(1,1)	-	(2,0)	(1,1)	(1,2)	(2,2)	-
3	$f(v_6)=3$	(2,0)	-	(2,0)	(1,1)	(2,1)	-	-
4	$f(v_5)=2$	(2,0)	-	(2,0)	(2,0)	-	-	-
5	$f(v_1)=1$	-	-	(2,0)	(2,0)	-	-	-
6	$f(v_3)=3$	-	-	-	(2,0)	-	-	-
7	$f(v_4)=3$	-	-	-	-	-	-	-

Vertex-Sequential with Dynamic Reordering Algorithm to the problem. For each level of the search tree and each vertex, it displays the value of the ordered pair $(cdeg(v_i),ucdeg(v_i))$. The vertex with the maximum colored degree was selected for coloring. Ties were broken by selecting the vertex which had the largest uncolored degree. If ties still existed, the vertex with the lowest index was selected. Easily, no backtracking possibilities exist. Thus, an optimal solution was attained. Note that this search tree contains 7 nodes as compared to 32 for the Basic Vertex-Sequential Algorithm.

G. VERTEX-SEQUENTIAL WITH DYNAMIC REORDERING FOR THE CGCP

Several adjustments must be made in order to generalize the previous concepts to the CGCP. First, as was proven in section E, when backtracking a lower bound on the smallest color in any set of colors assigned to vertex v_k is its currently smallest assigned color and an upper bound is given by

$$\min\{q_1\text{-}c(v_k),p+\text{MaxChrom}(G)\}$$

where q_1 is the current upper bound for $\chi(G)$ and p is the number of colors used to color $v_1,v_2,..v_{k-1}$.

The *uncolored adjacent chromatic degree* of vertex v_k, denoted $\text{ucAdjChromDeg}(v_k)$, is defined to be the sum of the chromaticities of all the as yet

uncolored vertices which are adjacent to v_k. Vertex reordering is based on the colored degree, chromaticity, uncolored adjacent chromatic degree, and uncolored degree of each vertex, in that order of precedence.

Example 8

Reconsider the graph in Figure 5. Table II illustrates the concepts of dynamic vertex reordering to find $\chi(G)$. For each level of the search tree, it displays the value of $(cdeg(v_i),c(v_i),ucAdjChromDeg(v_i),ucdeg(v_i))$. The previously described criteria for vertex selection was used to choose the vertex to be colored next. If ties occurred, the vertex of lowest index was selected. Although backtracking possibilities exist, the displayed solution is optimal.

H. COLOR-SEQUENTIAL ALGORITHM FOR THE SGCP

Recall that a vertex-sequential algorithm colors vertices sequentially with the smallest possible color which is not already assigned to an adjacent vertex. Another common approach is to concurrently color all the vertices which are to be assigned a particular color before assigning the next color. Specifically, all vertices which are to be assigned color 1 are colored first. Next, all vertices which are to be assigned color 2 are colored and the process continues until all vertices are colored. Thus, sets of vertices can be considered to be assigned to colors in a sequential fashion, hence the name color-sequential. An exact color-sequential algorithm was described in the Korman paper [*Ko79*].

Before presenting that procedure, the concept of an induced subgraph will be reviewed. A *subgraph* of $G=(V,E)$ is a graph $G'=(V',E')$ such that $V' \subset V$, $E' \subseteq E$, and for all $<v_i,v_j> \in E'$, $v_i,v_j \in V'$. G' is said to be an *induced subgraph* of G on V' if $E'=\{<v_i,v_j> \mid <v_i,v_j> \in E \wedge v_i,v_j \in V'\}$. The notation G-U will denote the induced subgraph of G on the set V-U. The basis for the color-sequential algorithm is the following theorem [*Ch71*].

TABLE II

VERTEX-SEQUENTIAL WITH DYNAMIC REORDERING FOR THE CGCP

Level	Assignment	v_1	v_2	v_3	v_4	v_5	v_6	v_7
0		(0,2,4,2)	(0,2,6,4)	(0,1,3,2)	(0,2,4,2)	(0,3,5,3)	(0,2,8,4)	(0,1,10,5)
1	$F(v_5)=\{1,2,3\}$	(0,2,4,2)	(0,2,6,4)	(0,1,3,2)	(3,2,1,1)	-	(3,2,5,3)	(3,1, 7,4)
2	$F(v_6)=\{4,5\}$	(2,2,2,1)	(2,2,4,3)	(0,1,3,2)	(3,2,1,1)	-	-	(5,1, 5,3)
3	$F(v_7)=\{6\}$	(2,2,2,1)	(3,2,3,2)	(1,1,2,1)	(4,2,0,0)	-	-	-
4	$F(v_4)=\{4,5\}$	(2,2,1,1)	(3,2,3,2)	(1,1,2,1)	-	-	-	-
5	$F(v_2)=\{1,2\}$	(4,2,2,1)	-	(3,1,0,0)	-	-	-	-
6	$F(v_1)=\{6,7\}$	-	-	(3,1,0,0)	-	-	-	-
7	$F(v_3)=\{3\}$	-	-	-	-	-	-	-

Theorem 2. Any graph $G=(V,E)$ can be optimally colored by coloring with color 1 some maximal independent set M_1 of G, then coloring with color 2 some maximal independent set M_2 of $G_1=G-M_1$, and so on until all vertices have been colored, where

$$G_k=\begin{cases} G-\bigcup_{i=1}^{k} M_i & ,k\geq 1 \\ G & ,k=0 \end{cases} \tag{16}$$

It should be emphasized that the theorem uses the phrase "some independent set". Finding the optimal sequence of independent sets requires a tree search.

This procedure requires that a list of the maximal independent sets of $G_0,G_1,G_2,...$ be available; but, this is not a major problem since efficient algorithms exist for finding all the maximal independent sets of a graph [*Br73*]. Furthermore, if $M_{k1},M_{k2},...,M_{ks}$ are the maximal independent sets of G_k, the maximal independent sets of G_{k+1} can be obtained by generating $M_{k1}-M_{kr}, M_{k2}-M_{kr},..,M_{ks}-M_{kr}$, where M_{kr} is the set selected to assign color $k+1$, and then eliminating any sets which are subsets of other sets.

Obviously, a depth first search can be developed based on these concepts. But, using only these ideas, level 1 of the tree would contain as many nodes as there are maximal independent sets in the graph. Level 2 would contain as many nodes

as there are maximal independent sets in all the possible subgraphs of G induced by removing one of its maximal independent sets, etc. A significant reduction in the size of the tree can be made by implementing the following fathoming technique.

Given an optimal coloring of G for which vertex v_i is colored $f(v_i)$, it is easy to obtain an equivalent optimal coloring in which v_i is assigned any desired specific color. Thus, at any level of coloring it can be stipulated that any one of the as yet uncolored vertices must be assigned the next available color. Hence, at level 1, it can be specified that vertex v_i, is to be colored with color 1. The only subgraphs of G which must be processed are those which are induced by removing maximal independent sets which contain v_i. By choosing a vertex which is contained in the least number of sets a maximal amount of fathoming can be obtained. This approach is continued at each succeeding level.

Example 9

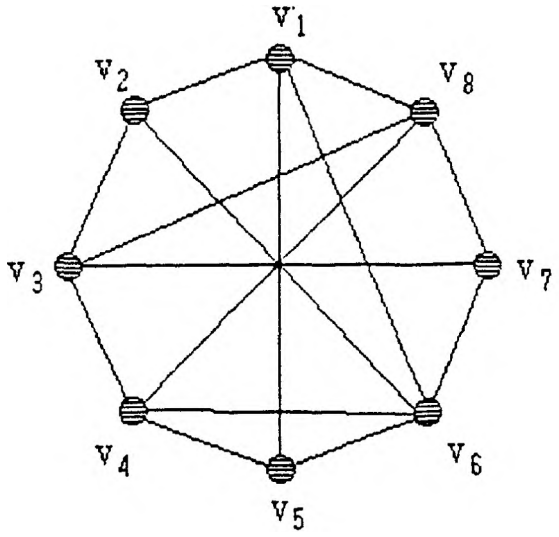
Consider the graph in Figure 8. It contains the following independent sets.

$M_{01} = \{v_3,v_5\}$ $M_{02} = \{v_1,v_3\}$ $M_{03} = \{v_2,v_4,v_7\}$
$M_{04} = \{v_2,v_5,v_7\}$ $M_{05} = \{v_2,v_5,v_8\}$ $M_{06} = \{v_1,v_4,v_7\}$
$M_{07} = \{v_3,v_6\}$ $M_{08} = \{v_6,v_8\}$

Since v_1 is contained in only 2 sets, it would be wise to color it first. The only subgraphs which have to be processed are those induced on $G\text{-}M_{02}$ and $G\text{-}M_{06}$. The maximal independent sets of $G\text{-}M_{02}$ can be obtained by finding $M_{01}\text{-}M_{02}$, $M_{03}\text{-}M_{02}$, $M_{04}\text{-}M_{02}$, $M_{05}\text{-}M_{02}$, $M_{06}\text{-}M_{02}$, $M_{07}\text{-}M_{02}$, and $M_{08}\text{-}M_{02}$ and then removing any sets which are subsets of other sets. This results in the following sets.

$M_{11} = \{v_2,v_4,v_7\}$ $M_{12} = \{v_2,v_5,v_7\}$
$M_{13} = \{v_2,v_5,v_8\}$ $M_{14} = \{v_6,v_8\}$

At this stage a good choice for the vertex to color next would be v_4 since it is contained in only one set. The entire search tree is displayed. A depth first search would visit either 7 or 8 nodes, depending on which node at level 1 is selected as the first node to visit.



i	$f(v_i)$
1	1
2	2
3	3
4	1
5	2
6	3
7	1
8	2

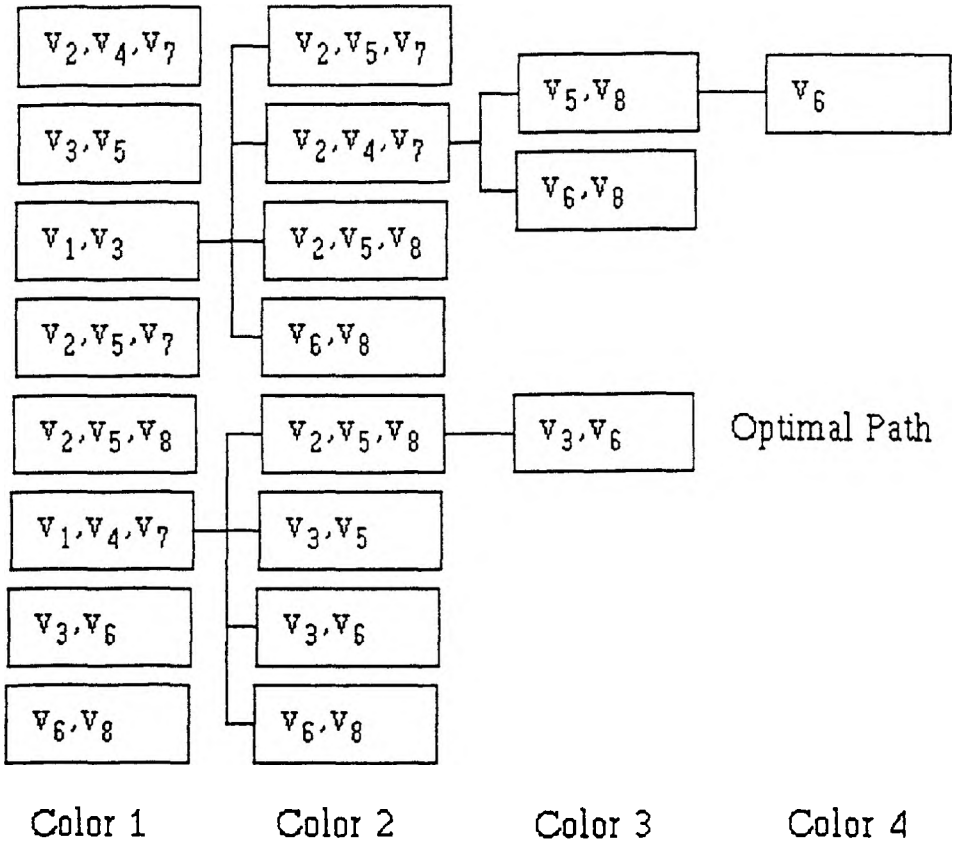


Figure 8. Color-Sequential Algorithm for the SGCP

I. COLOR-SEQUENTIAL ALGORITHM FOR THE CGCP

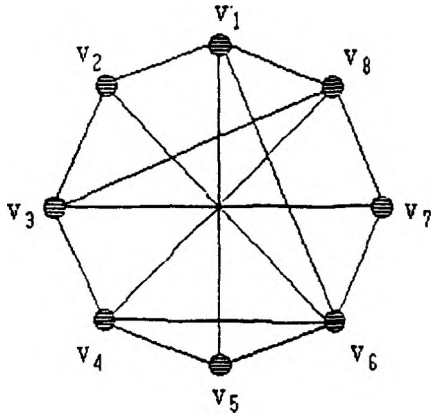
The Color-Sequential Algorithm for the SGCP can be generalized to include the CGCP by assigning a sequence of maximal independent sets to multiple colors instead of single colors. The number of colors a set is assigned to is the minimum chromaticity of the vertices in the set. As a result some vertices may be only partially colored and they must be left in the induced subgraph at the next level with a reduced chromaticity. Specifically, assume $M_{k1},M_{k2},..,M_{ks}$ are the maximal independent sets at level $k+1$ and suppose M_{kr} is the set selected to assign to a sequence of colors. The number of colors is given by $x=\min\{c(v_i)|v_i\epsilon M_{kr}\}$. Let $X=\{v_i|v_i\epsilon M_{kr}\Lambda c(v_i)=x\}$. The maximal independent sets at level $k+2$ are obtained by generating M_{k1}-X,M_{k2}-X,..,M_{ks}-X and then eliminating sets which are subsets of other sets. At level $k+2$ it is necessary to update the chromaticities of those vertices which were only partially colored at level $k+1$ and then select a maximal independent set which will result in their continued coloring. As long as these restrictions are followed, all of the other concepts for set selection and fathoming which apply to the SGCP algorithm apply to this generalization.

Example 10

Consider the graph in Figure 9. The Color-Sequential Algorithm for the CGCP generates the search tree in Figure 10. Superscripts of the form [1,2] indicate assigned colors.

J. EXPERIMENTS -- DESCRIPTION AND RESULTS

Of the four exact methods which have been described, Korman [*Ko79*] showed that the Vertex-Sequential with Dynamic Vertex Reordering is the most efficient method for finding the chromatic number of a random standard graph. Hence, it was decided that implementation would be limited to the generalization of that algorithm for the CGCP. Also, every composite graph has an associated standard graph with the same vertex and edge sets. It was hypothesized that there could be a functional relationship between the chromatic number of a composite



i	$c(v_i)$	$F(v_i)$
1	2	{1,2}
2	2	{5,6}
3	3	{1,2,3}
4	1	{7}
5	2	{5,6}
6	2	{3,4}
7	1	{7}
8	3	{4,5,6}

Figure 9. Color-Sequential Algorithm for the CGCP

graph and that of its related standard graph. In an effort to identify any such correlation, it was decided that the Vertex-Sequential with Dynamic Vertex Reordering Algorithm for the SGCP would be implemented.

The algorithm implementations were programmed in Turbo Pascal version 5.5 and executed on a NEC PowerMate 386/20 with a math coprocessor. All program listings are provided the Appendix.

1. Problem Generation Methodology.

A sample *random graph*, $G_{n,p}=(V,E)$, can be generated by utilizing a pseudo random number generator which produces random numbers between 0 and 1. For each of the n(n-1)/2 possible edges, a pseudo random number, r, is obtained. The edge is included in the graph if and only if $0 \le r \le p$.

Remember that a *type 1 composite random graph*, $CG_{n,p,1}=(V,E)$, is a random graph of order n and density p with the property that the vertex chromaticities are given by n independent truncated Poisson random variables, with parameter $q=1$. Hence, for all $v_i \in V$,

$$Prob\{c(v_i)=k\}=\frac{1}{(e-1)k!} \quad k=1,2,... \tag{17}$$

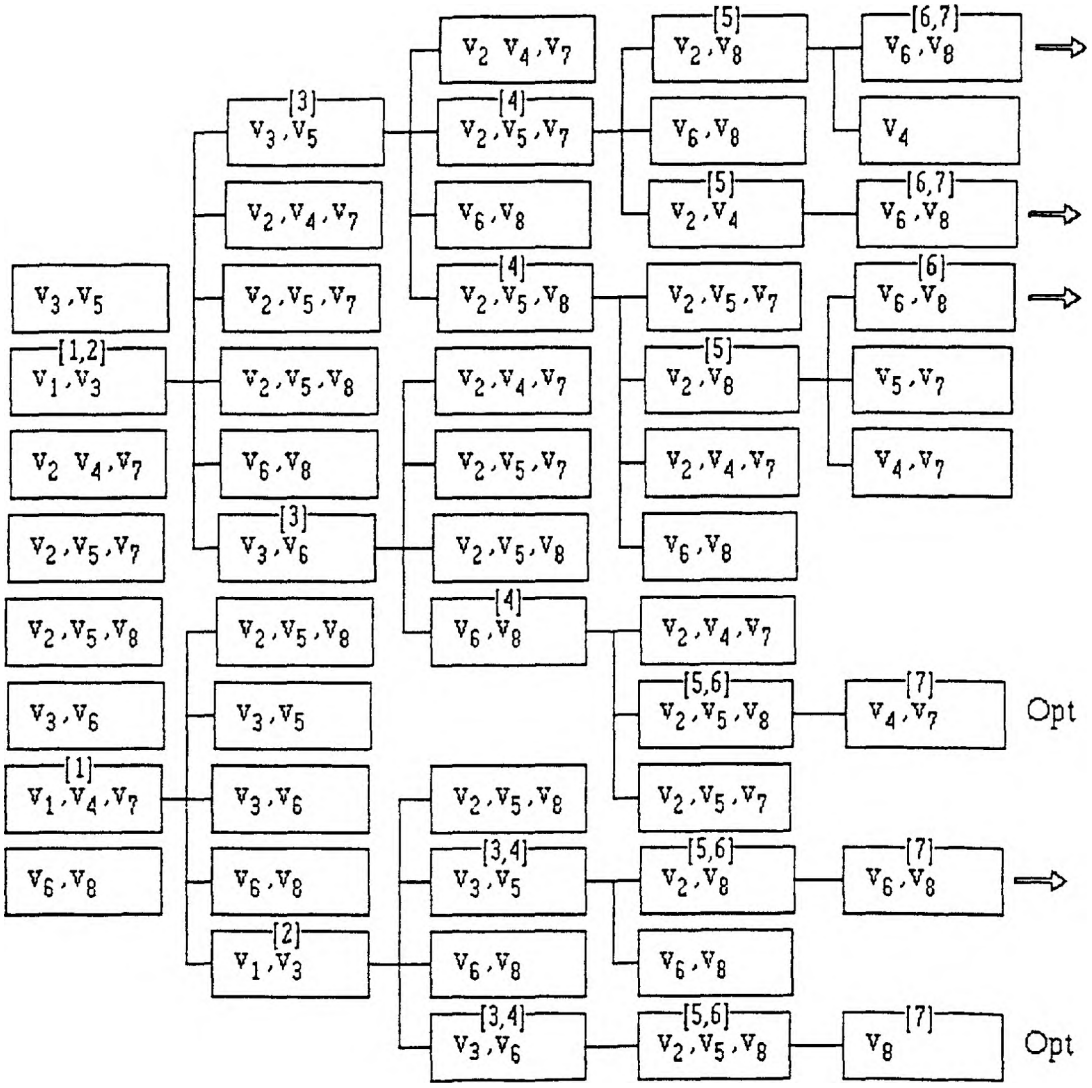


Figure 10. Color-Sequential Algorithm for the CGCP

Also, a *type 2 composite random graph*, $CG_{n,p,2}=(V,E)$, is a random graph of order n and density p with the property that for all $v_i \in V$,

$$\begin{aligned} Prob\{c(v_i)=1\} &= 0.75 \\ Prob\{c(v_i)=2\} &= 0.25 \end{aligned} \tag{18}$$

Easily, a pseudo random number generator can be used to construct instances of either type 1 or type 2 composite random graphs. For each such graph, an associated standard random graph can be created by assigning all vertex chromaticities the value 1.

The program which was used to produce all problem instances is provided as *Listing 1. GenGraph.Pas* in the Appendix.

2. Algorithm Implementations and Experiments.

The Vertex-Sequential with Dynamic Vertex Reordering Algorithms for the SGCP and CGCP were described in detail in sections III.F and III.G, respectively. These algorithms were applied to random samples of size 25 for problem instances with all the possible combinations of the specified values for the following characteristics.

1. Composite graph type: 1, 2
2. Density: 0.25, 0.50
3. Order: 10,12,..,MaxPossible

MaxPossible was determined by the size for which an upper limit of 250,000 forward moves at the current smallest upper bound resulted in fewer than 25 of 30 sample instances being solved. The SGCP algorithm was applied to the same problem instances as the CGCP algorithm except that all vertex chromaticities were set to 1. As was expected, MaxPossible for the CGCP was smaller than for the SGCP. Hence, at the point at which it was reached, random samples of larger order were processed; but, only the SGCP algorithm was applied. The associated program listings are provided in the Appendix as *Listing 2. SDynVSE.Pas* and *Listing 3. CDynVSE.Pas.*

TABLE III

GRAPH COLORING RESULTS
TYPE 2 RANDOM GRAPHS
p=0.25

Num Of Vert	******** Standard Graph *********					******** Composite Graph *******				
	Mean	Min	Max	SD	CI HW	Mean	Min	Max	SD	CI HW
10	2.840	2	4	0.472	0.264	3.840	2	5	0.800	0.447
12	3.120	2	4	0.440	0.246	4.400	3	6	0.816	0.457
14	3.280	3	4	0.459	0.257	4.520	4	7	0.714	0.400
16	3.560	3	4	0.506	0.283	4.960	3	6	0.840	0.470
18	3.640	3	4	0.489	0.274	5.080	4	7	0.862	0.482
20	3.800	3	4	0.408	0.228	5.440	4	7	0.870	0.486
22	4.000	4	4	0.000	0.000	5.480	5	7	0.654	0.366
24	4.080	4	5	0.278	0.155	5.800	5	7	0.577	0.323
26	4.200	4	5	0.408	0.228	5.880	5	6	0.726	0.406
28	4.200	4	5	0.408	0.228	6.000	5	8	0.764	0.427
30	4.320	4	5	0.477	0.267	6.320	5	8	0.802	0.449
32	4.800	4	5	0.408	0.228	6.520	5	8	0.771	0.431
34	4.880	4	5	0.332	0.186	6.560	5	9	0.870	0.486
36	4.960	4	5	0.199	0.111	6.720	5	8	0.792	0.443
38	5.000	5	5	0.000	0.000	6.920	6	8	0.760	0.425
40	5.080	5	6	0.278	0.155	6.840	6	8	0.624	0.349
42	5.040	5	6	0.199	0.111	7.240	6	9	0.779	0.436
44	5.120	5	6	0.332	0.186	7.200	6	8	0.645	0.361
46	5.440	5	6	0.506	0.283	7.520	7	9	0.586	0.328
48	5.760	5	6	0.435	0.244	7.760	7	10	0.723	0.405
50	6.000	6	6	0.000	0.000	8.040	7	9	0.611	0.342
52	5.960	5	6	0.199	0.111	8.280	7	9	0.614	0.343
54	6.000	6	6	0.000	0.000	8.120	7	9	0.600	0.336
56	6.040	6	7	0.199	0.111	8.320	7	9	0.557	0.312
58	6.080	6	7	0.278	0.155	8.400	7	9	0.577	0.323
60	6.120	6	7	0.332	0.186					
62	6.320	6	7	0.477	0.267					
64	6.600	6	7	0.500	0.280					
66	6.840	6	7	0.374	0.209					
68	7.000	7	7	0.000	0.000					
70	7.000	7	7	0.000	0.000					
72	7.000	7	7	0.000	0.000					
74	7.000	7	7	0.000	0.000					
76	7.040	7	8	0.199	0.111					
78	7.080	7	8	0.278	0.155					

TABLE IV

GRAPH COLORING RESULTS
TYPE 2 RANDOM GRAPHS
p=0.50

Num Of Vert	******** Standard Graph ********* Mean	Min	Max	SD	CI HW	******** Composite Graph ******** Mean	Min	Max	SD	CI HW
10	3.960	3	5	0.675	0.378	5.360	4	8	1.075	0.601
12	4.440	3	6	0.650	0.364	6.040	4	8	1.020	0.570
14	4.80	4	5	0.408	0.228	6.440	5	9	0.961	0.537
16	5.080	4	6	0.494	0.276	7.080	5	9	0.909	0.509
18	5.280	4	6	0.542	0.303	6.800	5	8	0.866	0.484
20	5.720	5	7	0.542	0.303	7.840	6	10	1.179	0.659
22	5.880	5	6	0.332	0.186	8.160	7	10	0.850	0.476
24	6.240	6	7	0.435	0.244	8.800	7	11	1.000	0.559
26	6.440	6	7	0.506	0.283	8.560	7	11	1.121	0.627
28	7.000	6	8	0.408	0.228	9.360	7	12	1.186	0.663
30	7.000	6	8	0.289	0.161	9.840	8	12	1.247	0.698
32	7.240	6	8	0.522	0.292	9.880	7	12	1.130	0.632
34	7.400	7	8	0.500	0.280	10.120	9	12	1.093	0.611
36	7.880	7	8	0.332	0.186	10.680	9	13	0.988	0.553
38	8.040	7	9	0.351	0.196	10.920	9	12	0.813	0.455
40	8.080	8	9	0.278	0.155	10.920	10	13	0.909	0.509
42	8.720	8	9	0.459	0.257	11.600	10	14	1.155	0.646
44	8.920	8	9	0.278	0.155	11.560	10	13	1.044	0.584
46	9.040	9	10	0.199	0.111					
48	9.280	9	10	0.459	0.257					
50	9.560	9	10	0.506	0.283					
52	9.880	9	10	0.332	0.186					
54	9.920	9	10	0.278	0.155					
56	10.040	9	11	0.351	0.196					
58	10.200	10	11	0.408	0.228					

3. Results and Conclusions.

Table III displays a summary of the results of applying the algorithms to composite type 2 graphs and their associated standard graphs with edge densities of 0.25. Table IV presents the results for type 2 graphs with edge densities of 0.50 and Tables V and VI show the outcomes for type 1 graphs with densities of 0.25 and 0.50, respectively. In each table the column whose heading is Mean contains values for the arithmetic mean of the chromatic numbers of the sample graphs. The

TABLE V

GRAPH COLORING RESULTS
TYPE 1 RANDOM GRAPHS
p = 0.25

Num Of Vert	******** Standard Graph *********					******** Composite Graph *******				
	Mean	Min	Max	SD	CI HW	Mean	Min	Max	SD	CI HW
10	2.880	2	4	0.600	0.336	5.120	3	8	1.269	0.710
12	3.040	3	4	0.199	0.111	5.120	4	7	0.928	0.519
14	3.240	3	4	0.435	0.244	6.120	4	9	1.202	0.672
16	3.480	3	4	0.510	0.285	6.600	5	10	1.323	0.740
18	3.560	3	4	0.506	0.283	6.800	4	10	1.354	0.757
20	3.840	3	4	0.374	0.209	6.920	5	10	1.188	0.664
22	4.040	4	5	0.199	0.111	7.120	5	9	0.882	0.493
24	4.040	4	5	0.199	0.111	7.600	5	11	1.414	0.791
26	4.080	4	5	0.278	0.155	7.920	6	10	1.115	0.624
28	4.240	4	5	0.435	0.244	7.800	6	10	1.041	0.582
30	4.560	4	5	0.506	0.283	8.200	6	11	1.118	0.625
32	4.680	4	5	0.477	0.267	8.640	6	10	0.952	0.533
34	4.960	4	5	0.199	0.111	8.960	7	12	1.136	0.635
36	5.000	5	5	0.000	0.000	8.760	7	10	0.830	0.465
38	5.000	5	5	0.000	0.000	8.800	7	11	1.041	0.582
40	5.040	5	6	0.199	0.111	9.280	8	12	0.980	0.548
42	5.200	5	6	0.408	0.228	9.640	8	11	0.907	0.507
44	5.280	5	6	0.459	0.257	9.680	8	11	0.802	0.449
46	5.440	5	6	0.506	0.283					
48	5.760	5	6	0.435	0.244					
50	6.000	6	6	0.000	0.000					
52	5.960	5	6	0.199	0.111					
54	6.000	6	6	0.000	0.000					
56	6.040	6	7	0.199	0.111					
58	6.080	6	7	0.278	0.155					
60	6.120	6	7	0.332	0.186					
62	6.320	6	7	0.477	0.267					
64	6.600	6	7	0.500	0.280					
66	6.840	6	7	0.374	0.209					
68	7.000	7	7	0.000	0.000					
70	7.000	7	7	0.000	0.000					
72	7.000	7	7	0.000	0.000					
74	7.000	7	7	0.000	0.000					
76	7.080	7	8	0.199	0.111					
78	7.120	7	8	0.278	0.155					

TABLE VI

GRAPH COLORING RESULTS
TYPE 1 RANDOM GRAPHS
p=0.50

Num Of Vert	******** Standard Graph *********					******** Composite Graph *******				
	Mean	Min	Max	SD	CI HW	Mean	Min	Max	SD	CI HW
10	3.960	3	5	0.538	0.301	7.040	5	10	1.369	0.766
12	4.360	4	5	0.489	0.274	7.440	5	11	1.416	0.792
14	4.680	4	5	0.477	0.267	8.200	6	12	1.581	0.884
16	5.000	4	6	0.500	0.280	9.400	7	12	1.384	0.774
18	5.400	4	6	0.577	0.323	9.480	7	12	1.295	0.724
20	5.600	5	6	0.500	0.280	9.920	7	12	1.256	0.702
22	6.040	5	7	0.454	0.254	10.480	8	12	1.159	0.648
24	6.400	6	7	0.500	0.280	11.320	9	15	1.701	0.952
26	6.440	6	7	0.506	0.283	11.720	9	16	1.458	0.816
28	6.880	6	7	0.332	0.186	12.080	9	14	1.222	0.684
30	6.960	6	8	0.454	0.254	12.320	9	18	1.773	0.992
32	7.240	7	8	0.435	0.244	13.040	10	15	1.207	0.675
34	7.400	7	8	0.500	0.280					
36	7.880	7	8	0.332	0.186					
38	8.040	7	9	0.351	0.196					
40	8.080	8	9	0.278	0.155					
42	8.720	8	9	0.459	0.257					
44	8.920	8	9	0.278	0.155					
46	9.040	9	10	0.199	0.111					
48	9.280	9	10	0.459	0.257					
50	9.560	9	10	0.506	0.283					
52	9.880	9	10	0.332	0.186					
54	9.920	9	10	0.278	0.155					
56	10.040	9	11	0.351	0.196					
58	10.200	10	11	0.408	0.228					

columns headed Max and Min contain the value of the maximum and minimum chromatic numbers, respectively. The SD columns contain the values of the sample standard deviation for the chromatic numbers. The columns whose heading is CI/HW contain values for the half interval width of a 99% confidence interval for the mean chromatic number of the associated population of graphs. These values were calculated from the following formula.

TABLE VII

VERTEX CHROMATICITY PROBABILITIES

k	$Pr\{c(v_i)=k\}$ Type 1	Type 2
1	0.582	0.750
2	0.291	0.250
3	0.097	0.000
4	0.024	0.000
5	0.005	0.000
6	0.001	0.000
7	0.000	0.000
Mean	1.582	1.250
Variance	0.661	0.188
Std Dev	0.813	0.433

$$Half\ Width = \frac{t_{n-1,1-\alpha/2}\, s}{\sqrt{n}} \tag{19}$$

where n=25 is the sample size, $(1-\alpha)=0.99$ is the confidence coefficient, s is the sample standard deviation, and $t_{n-1,1-\alpha/2}=2.7969$ is a critical value from the student's t-distribution [*Me65*].

As was shown in sections III.E-III.G, the tables further illustrate that the CGCP is a much more difficult problem to solve than the SGCP. Specifically, MaxPossible for standard graphs with edge density 0.25 was 78. But, as Table III displays, for type 2 composite graphs it was only 58; and, as is shown in Table V, for type 1 composite graphs with edge density 0.25, MaxPossible was only 44. Likewise, MaxPossible for standard graphs with edge density 0.50 was 58. However, for type 2 composite graphs, Table IV indicates that MaxPossible was 44; and, for type 1 graphs, Table VI exhibits that it was only 32.

Tables V and VI when compared to Tables III and IV demonstrate that type 1 graphs are more difficult to solve then type 2 graphs. This is also justified by the theory which was presented in sections III.E-III.G. It was shown that the search

space for a particular problem increased exponentially with the maximum vertex chromaticity. As is displayed in Table VII, type 1 graphs have a larger expected vertex chromaticity then do type 2 graphs.

The experimental results displayed in Tables III-VI also indicate that there is a larger variation in chromatic numbers for composite graphs when compared to standard graphs and for type 1 composite graphs when compared to type 2 composite graphs. The following theory explains these findings, as well as, several other observations which will be made later.

Suppose $CG_{n,p}=(V,E)$ is a random composite graph with $V=\{v_1,v_2,..,v_n\}$. Define its *projection* to be the standard graph, $PG_{n',p'}=(V',E')$, where

$$V' = \bigcup_{i=1}^{n} \{v_{i1},v_{i2},..,v_{ic(v_i)}\} \tag{20}$$

and

$$\begin{aligned} E' = {} & \{<v_{ik},v_{il}> \mid 1\le i\le n \wedge 1\le k<l\le c(v_i)\} \cup \\ & \{<v_{ik},v_{jl}> \mid <v_i,v_j>\in E \wedge 1\le k\le c(v_i) \wedge 1\le l\le c(v_j)\} \end{aligned} \tag{21}$$

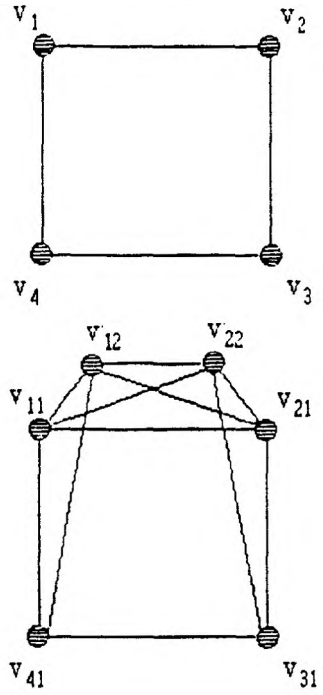
Example 11

Consider Figure 11. The upper graph is a composite graph with vertex chromaticities specified in the associated table. The lower graph represents its projection.

Two vertices, $v_{ij},v_{ik}\in V'$ where $1\le i\le n$, $1\le j\le c(v_i)$, and $1\le k\le c(v_i)$, are defined to be *images* of $v_i\in V$. The vertex v_i will be called the *source* of v_{ij} and v_{ik}. The vertices v_{ij} and v_{ik} are said to be *isometric* to each other, denoted $v_{ij}\simeq v_{ik}$. Let $n_{iso}(v_{ij})$ designate the number of vertices which are isometric to v_{ij}.

Easily, $n'=c(CG_{n,p})$ is a normal random variable with

$$\mu(n') = \begin{cases} 1.582n, & \text{if } CG_{n,p} \text{ is Type 1} \\ 1.250n, & \text{if } CG_{n,p} \text{ is Type 2} \end{cases} \tag{22}$$



i	c(v_i)
1	2
2	2
3	1
4	1

Figure 11. The Projection of a Composite Graph

and

$$\sigma(n') = \begin{cases} \sqrt{0.661n}, & \text{if } CG_{n,p} \text{ is Type 1} \\ \sqrt{0.188n}, & \text{if } CG_{n,p} \text{ is Type 2} \end{cases} \tag{23}$$

The value of p′ is established by the following theorem. In order to simplify the notation, single subscripts will be used to distinguish vertices in $PG_{n',p'}$.

Theorem 3. If $CG_{n,p}$ is a random composite graph, then the edge density of its projection, $PG_{n',p'}$, is given by

$$p' = \begin{cases} p\,\dfrac{1.582n-2+1/p}{1.582n-1}, & \text{if } CG_{n,p} \text{ is Type 1} \\ p\,\dfrac{25n-28+8/p}{25n-20}, & \text{if } CG_{n,p} \text{ is Type 2} \end{cases} \tag{24}$$

Proof. Suppose $CG_{n,p}$ is a type 1 random composite graph. Let v_i and v_j be two vertices which are selected at random from the vertex set of its projection, $PG_{n',p'}$. Then

$$\begin{aligned} Pr\{v_i \sim v_j\} = {} & Pr\{v_i \sim v_j \mid n_{iso}(v_i)=1\} \cdot Pr\{n_{iso}(v_i)=1\} + \\ & Pr\{v_i \sim v_j \mid n_{iso}(v_i)=2\} \cdot Pr\{n_{iso}(v_i)=2\} + \\ & Pr\{v_i \sim v_j \mid n_{iso}(v_i)=3\} \cdot Pr\{n_{iso}(v_i)=3\} + \\ & Pr\{v_i \sim v_j \mid n_{iso}(v_i)=4\} \cdot Pr\{n_{iso}(v_i)=4\} + \\ & Pr\{v_i \sim v_j \mid n_{iso}(v_i)=5\} \cdot Pr\{n_{iso}(v_i)=5\} + \\ & Pr\{v_i \sim v_j \mid n_{iso}(v_i)=6\} \cdot Pr\{n_{iso}(v_i)=6\} \end{aligned} \tag{25}$$

Substitution yields,

$$\begin{aligned} Pr\{v_i \sim v_j\} = {} & \frac{1}{1.582n-1} \cdot \frac{2(0.291)n}{1.582n} + \frac{2}{1.582n-1} \cdot \frac{3(0.097)n}{1.582n} + \\ & \frac{3}{1.582n-1} \cdot \frac{4(0.024)n}{1.582n} + \frac{4}{1.582n-1} \cdot \frac{5(0.005)n}{1.582n} + \\ & \frac{5}{1.582n-1} \cdot \frac{6(0.001)n}{1.582n} + \frac{6}{1.582n-1} \cdot \frac{7(0.000)n}{1.582n} \\ = {} & \frac{1}{1.582n-1} \end{aligned} \tag{26}$$

Hence,

$$\begin{aligned} p' &= Pr\{<v_i,v_j> \in E'\} \\ &= Pr\{<v_i,v_j> \in E' \mid v_i \sim v_j\} \cdot Pr\{v_i \sim v_j\} + Pr\{<v_i,v_j> \in E' \mid v_i \not\sim v_j\} \cdot Pr\{v_i \not\sim v_j\} \\ &= 1 \cdot \frac{1}{1.582n-1} + p \cdot \frac{1.582n-2}{1.582n-1} = p\,\frac{1.582n-2+1/p}{1.582n-1} \end{aligned} \tag{27}$$

The result for type 2 composite graphs can be proved in a similar manner.

Table VIII displays values of $\mu(n')$, $\mu(n')+\sigma(n')$ and p' for $p=0.25$ and $p=0.50$ and selected values of n. Since n' is a normal random variable,

$$Pr\{n' \geq \mu(n')+\sigma(n')\} = 0.1587 \tag{28}$$

TABLE VIII

SAMPLE STATISTICS FOR $PG_{n',p'}$

	**** $\mu(n')$ *****		** $\mu(n')+\sigma(n')$ **		********** p′ **********			
					p = 0.25		p = 0.50	
n	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2
10	15.82	12.50	18.39	13.87	0.301	0.276	0.534	0.517
20	31.64	25.00	35.28	26.94	0.274	0.263	0.516	0.508
30	47.46	37.50	51.91	39.87	0.266	0.258	0.511	0.505
32	50.62	40.00	55.22	42.45	0.265	0.258	0.510	0.505
40	63.28	50.00	68.42	52.74	0.262	0.256	0.508	0.504
44	69.61	55.00	75.00	57.88	0.261	0.256	0.507	0.504
50	79.10	62.50	84.85	65.57	0.260	0.255	0.506	0.503
58	91.76	72.50	97.95	75.80	0.258	0.254	0.506	0.503
60	94.92	75.00	101.22	78.36	0.258	0.254	0.505	0.503
70	110.74	87.50	117.54	91.13	0.257	0.253	0.505	0.502
80	126.56	100.00	133.83	103.88	0.256	0.253	0.504	0.502
90	142.38	112.50	150.09	116.61	0.255	0.253	0.504	0.502
100	158.20	125.00	166.33	129.34	0.255	0.252	0.503	0.502
200	316.40	250.00	327.90	256.13	0.252	0.251	0.502	0.501
300	474.60	375.00	488.68	382.51	0.252	0.251	0.501	0.501
400	632.80	500.00	649.06	508.67	0.251	0.251	0.501	0.500
500	791.00	625.00	809.18	634.70	0.251	0.250	0.501	0.500
1000	1582.00	1250.00	1607.71	1263.71	0.250	0.250	0.500	0.500

Thus, as Table VIII indicates, a 58 node type 2 composite graph with edge density 0.25 would produce a projection with an edge density of 0.254 and, in about 16% of the cases, the projection would have at least 75.80 nodes. Note that for both type 1 and type 2 composite graphs, as n gets large p′ approaches p as a limit; and, as Table VIII illustrates, even for relatively small values of n, p′ is approximately equal to p.

Obviously, if the projection, $PG_{n',p'}$, of a composite graph, $CG_{n,p}$, was processed as a standard graph then any optimal coloring of $CG_{n,p}$ would be feasible for $PG_{n',p'}$. In order to make $PG_{n',p'}$ equivalent to $CG_{n,p}$, with respect to coloring, a constraint that any feasible coloring must assign *consecutive* colors to *isometric* vertices would have to be added. This, together with the fact that $p' > p$, means that

TABLE IX

STANDARD GRAPH COLORING
IMPLEMENTATION EFFECTIVENESS EVALUATION

	SDynVSE		Sager & Lin	
n	Mean Time	Mean $\chi(n)$	Mean Time	Mean $\chi(n)$
32	0.63	7.24	0.58	7.30
40	4.58	8.08	4.87	8.28
48	39.29	9.28	48.70	9.12
56	224.62	10.04	312.00	10.02

$CG_{n,p}$ can be expected to be *at least* as difficult to color as $PG_{n',p'}$, processed simply as a standard graph.

Before proceeding, it is appropriate to discuss the effectiveness of the implementation of the Vertex-Sequential with Dynamic Vertex Reordering Algorithm for the SGCP. Sager and Lin developed two improved versions of the algorithm and tested their effectiveness, as compared to the basic version, using implementations written in Turbo Pascal 5.5 and executed on a 20 megahertz IBM PS2 model 80 [*SL90*]. Since, the same language and comparable hardware were used for the implementation described in this paper, it was felt that a reasonably fair comparison could be made. Table IX displays the mean chromatic numbers and mean execution times (in seconds) which were observed by this author along with those reported by Sager and Lin for standard random graphs with $p=0.50$ and $n=32,40,48,56$. For $n=32$ and 40, the times were similar. But, for $n=48$ and 56, SDynVSE required significantly less processing time. One possible explanation for the improved execution time is that Sager and Lin used only an adjacency matrix structure to represent the graphs in main memory. In SDynVSE, graphs are stored redundantly using both an adjacency matrix structure and an adjacency list structure. The time needed to identify if two given vertices are adjacent is minimized by the adjacency matrix. But, the time needed to identify all vertices adjacent to a given vertex is minimized by the adjacency list representation. Thus, by having both

representations available, it should be possible to significantly improve execution time.

The preceding discussion, along with Tables III through IX, support the following four conclusions.

1. The fact that composite graph coloring problems are more difficult to solve than standard graph coloring problems of the same order is further justified. For example, as previously indicated, with probability 0.1587 a 58 node type 2 random composite graph with an edge density of $p=0.25$ will be more difficult to solve than a 75.80 node random standard graph with edge density 0.254. Likewise, applying the properties of normal random variables to n', we find that, with probability 0.9898, such a graph will be more difficult to solve than a 64.8 node random standard graph.

2. The larger variation in chromatic numbers for composite graphs when compared to standard graphs and for type 1 composite graphs when compared to type 2 is explained by the standard deviation formulas for n' in Equation 23.

3. The implementation of the Vertex-Sequential with Dynamic Vertex Reordering Algorithm for the SGCP in SDynVSE.Pas is reasonably efficient.

4. The Vertex-Sequential with Dynamic Vertex Reordering Algorithm for the CGCP appears to be as effective as the Vertex-Sequential with Dynamic Vertex Reordering Algorithm for the SGCP. For example, as indicated in Table III, the value of MaxPossible for the SGCP with $p=0.25$ was $n=78$. The same table shows that MaxPossible for type 2 random composite graphs was $n=58$. But, the above theory and, specifically, Table VIII suggests that, in about 16% of the cases, such problems would be more difficult to solve than standard graphs with $n=75.80$. Table V lists MaxPossible for type 1 composite graphs with $p=0.25$ as $n=44$. Table VIII indicates that we can expect at least 16% of these problems to be harder to solve than standard graphs with $n=75.00$. For $p=0.50$, Table IV identifies that MaxPossible for standard graphs was $n=58$. According to the same table, for type 2 composite graphs it was $n=44$. However, Table VIII documents that, in 16% of the cases, a 44 node type 2 composite graph with $p=0.50$ would be more difficult to solve than a 57.88 node standard graph with $p=0.504$. Table VI specifies that

TABLE X

GRAPH COLORING RESULTS
TYPE 2 RANDOM GRAPHS
p=0.25

n	χ(SG)	χ(CG)	χ(CG)/χ(SG)	p	n′	c(v)	p′
10	2.840	3.840	1.357	0.246	12.440	1.244	0.266
12	3.120	4.400	1.430	0.245	14.960	1.247	0.267
14	3.280	4.520	1.387	0.247	17.440	1.246	0.266
16	3.560	4.960	1.407	0.249	20.280	1.268	0.261
18	3.640	5.080	1.417	0.243	22.000	1.222	0.256
20	3.800	5.440	1.440	0.246	25.200	1.260	0.255
22	4.000	5.480	1.370	0.250	27.640	1.256	0.259
24	4.080	5.800	1.424	0.249	30.680	1.278	0.256
26	4.200	5.880	1.410	0.247	32.240	1.240	0.253
28	4.200	6.000	1.436	0.248	34.720	1.240	0.259
30	4.320	6.320	1.472	0.248	38.160	1.272	0.255
32	4.800	6.520	1.372	0.246	40.720	1.273	0.256
34	4.880	6.560	1.350	0.247	42.720	1.256	0.254
36	4.960	6.720	1.356	0.246	45.640	1.268	0.249
38	5.000	6.920	1.384	0.248	47.760	1.257	0.254
40	5.080	6.840	1.349	0.247	49.160	1.229	0.252
42	5.040	7.240	1.437	0.248	53.360	1.270	0.251
44	5.120	7.200	1.408	0.247	54.680	1.243	0.254
46	5.440	7.520	1.393	0.249	57.440	1.249	0.255
48	5.760	7.760	1.353	0.251	61.080	1.272	0.255
50	6.000	8.040	1.340	0.252	63.160	1.263	0.257
52	5.960	8.280	1.389	0.252	65.960	1.268	0.257
54	6.000	8.120	1.353	0.250	67.560	1.251	0.256
56	6.040	8.320	1.379	0.252	70.400	1.257	0.256
58	6.080	8.400	1.384	0.252	72.680	1.253	0.257
Sample Mean			1.392	0.248		1.255	0.257
Sample SD			0.035	0.002		0.014	0.004
99% CI LBound			1.372	0.247		1.247	0.254
99% CI UBound			1.412	0.249		1.263	0.259

MaxPossible for type I composite graphs with edge density 0.50 was only n=32. But, according to Table VIII, with probability greater than 0.1587, such a graph would be more difficult to color than a 55.22 node random standard graph with density 0.510. Thus, in all tested cases, the conclusion is supported.

Tables X-XIII document the actual observed mean values for such attributes as edge density (p), vertex chromaticity ($c(v_i)$), projection order (n′), and projection

TABLE XI

GRAPH COLORING RESULTS
TYPE 2 RANDOM GRAPHS
p=0.50

n	χ(SG)	χ(CG)	χ(CG)/χ(SG)	p	n′	c(v)	p′
10	3.960	5.360	1.363	0.504	12.440	1.244	0.513
12	4.440	6.040	1.365	0.505	14.960	1.247	0.517
14	4.800	6.440	1.342	0.492	17.440	1.246	0.507
16	5.080	7.080	1.401	0.494	20.280	1.268	0.500
18	5.280	6.800	1.291	0.487	22.000	1.222	0.496
20	5.720	7.840	1.375	0.493	25.200	1.260	0.500
22	5.880	8.160	1.392	0.494	27.640	1.256	0.504
24	6.240	8.800	1.413	0.499	30.680	1.278	0.509
26	6.440	8.560	1.335	0.494	32.240	1.240	0.496
28	7.000	9.360	1.337	0.499	34.720	1.240	0.508
30	7.000	9.840	1.405	0.499	38.160	1.272	0.507
32	7.240	9.880	1.367	0.495	40.720	1.273	0.501
34	7.400	10.120	1.371	0.497	42.720	1.256	0.500
36	7.880	10.680	1.356	0.495	45.640	1.268	0.498
38	8.040	10.920	1.359	0.497	47.720	1.256	0.501
40	8.080	10.920	1.352	0.493	49.160	1.229	0.495
42	8.720	11.600	1.333	0.498	53.360	1.270	0.498
44	8.920	11.560	1.296	0.496	54.880	1.247	0.498
Sample Mean			1.359	0.496		1.254	0.503
Sample SD			0.033	0.004		0.016	0.006
99% CI LBound			1.340	0.494		1.245	0.499
99% CI UBound			1.377	0.498		1.263	0.506

edge density (p′). In all cases, the data is as would be expected according to the preceding theory.

In addition, these tables contain mean values for the chromatic numbers of the sample of composite graphs and their associated standard graphs and mean values for the ratios of composite chromatic number to corresponding standard chromatic number. Based on these results we can make the following conclusions.

1. As displayed in Table XII, for type 1 graphs with p=0.25, the sample mean ratio of composite chromatic number to corresponding standard chromatic number is 1.836 with a sample standard deviation of 0.067. Applying the theory associated with Equation 19, it follows that a 99% confidence interval for the population mean

TABLE XII

GRAPH COLORING RESULTS
TYPE 1 RANDOM GRAPHS
p=0.25

n	χ(SG)	χ(CG)	χ(CG)/χ(SG)	p	n′	c(v)	p′
10	2.880	5.120	1.813	0.231	16.080	1.608	0.259
12	3.040	5.120	1.687	0.245	18.400	1.533	0.272
14	3.240	6.120	1.917	0.244	22.280	1.591	0.273
16	3.480	6.600	1.920	0.248	25.920	1.620	0.274
18	3.560	6.800	1.923	0.244	29.000	1.611	0.264
20	3.840	6.920	1.817	0.248	32.000	1.600	0.259
22	4.040	7.120	1.762	0.246	34.680	1.576	0.262
24	4.040	7.600	1.878	0.246	38.200	1.592	0.261
26	4.080	7.920	1.942	0.248	40.880	1.572	0.268
28	4.240	7.800	1.848	0.246	43.920	1.569	0.255
30	4.560	8.200	1.816	0.248	47.360	1.579	0.259
32	4.680	8.640	1.862	0.250	51.480	1.609	0.262
34	4.960	8.960	1.810	0.247	53.360	1.569	0.257
36	5.000	8.760	1.752	0.246	55.880	1.552	0.255
38	5.000	8.800	1.760	0.246	59.480	1.565	0.253
40	5.040	9.280	1.844	0.252	63.360	1.584	0.261
42	5.200	9.640	1.861	0.251	66.360	1.580	0.260
44	5.280	9.680	1.844	0.251	69.520	1.580	0.258
Sample Mean			1.836	0.246		1.583	0.262
Sample SD			0.067	0.004		0.022	0.006
99% CI LBound			1.799	0.244		1.571	0.258
99% CI UBound			1.874	0.249		1.595	0.265

ratio is [1.799,1.874]. For p=0.50, Table XIII indicates that a 99% confidence interval for the mean composite to standard chromatic number ratio is [1.755,1.806].

2. For type 2 graphs, Tables X and XI exhibit that 99% confidence intervals for the mean ratio of composite chromatic number to standard chromatic number are [1.372,1.412] and [1.340,1.377] for edge densities of p=0.25 and p=0.50, respectively.

For both types, it appears that a composite graph requires a number of colors which is a consistent percentage above that required by its associated standard graph. This percentage varies with p; but, it seems to be independent of graph order n.

TABLE XIII

GRAPH COLORING RESULTS
TYPE 1 RANDOM GRAPHS
p=0.50

n	χ(SG)	χ(CG)	χ(CG)/χ(SG)	p	n′	c(v)	p′
10	3.960	7.040	1.794	0.508	16.080	1.608	0.527
12	4.360	7.440	1.704	0.497	18.400	1.533	0.509
14	4.680	8.200	1.766	0.488	22.280	1.591	0.501
16	5.000	9.400	1.892	0.494	25.920	1.620	0.509
18	5.400	9.480	1.758	0.492	29.000	1.611	0.497
20	5.600	9.920	1.777	0.492	32.000	1.600	0.495
22	6.040	10.480	1.740	0.495	34.680	1.576	0.505
24	6.400	11.320	1.773	0.495	38.200	1.592	0.506
26	6.440	11.720	1.824	0.497	40.880	1.572	0.513
28	6.880	12.080	1.758	0.499	43.920	1.569	0.504
30	6.960	12.320	1.773	0.494	47.280	1.576	0.502
32	7.240	13.040	1.804	0.498	51.720	1.616	0.506
Sample Mean			1.780	0.496		1.589	0.506
Sample SD			0.046	0.005		0.024	0.008
99% CI LBound			1.755	0.493		1.575	0.502
99% CI UBound			1.806	0.498		1.602	0.511

IV. HEURISTIC GCP ALGORITHMS

As was illustrated in the previous chapter, exact algorithms for finding an optimal coloring of either a standard or composite graph will most likely be computationally intractable if the graph contains on the order of 100 vertices or more and has an edge density which is something other than 0 or 1. This was expected since both problems are known to be NP-complete.

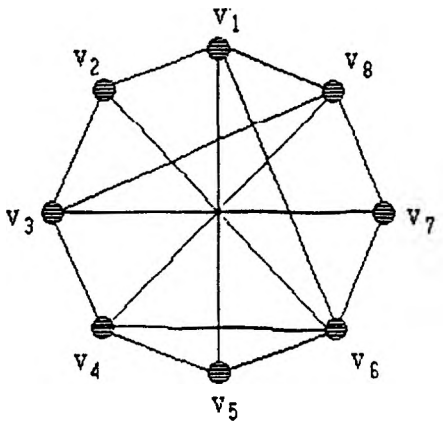
A common approach for dealing with large instances of NP-complete problems is to find heuristic algorithms which can be computed efficiently and produce an acceptable approximation to an optimal solution. This chapter consists of a description of the design, implementation, and results of testing ten such algorithms for both the standard and composite graph coloring problems.

A. BASIC VERTEX-SEQUENTIAL ALGORITHMS FOR THE SGCP

Given a standard graph, SG = (V,E), basic vertex-sequential algorithms use an *ordering rule* for controlling the sequence in which the vertices are processed. Suppose that applying this rule results in the vertex order $v_1,v_2,..,v_n$. In a manner which is identical to the way the exact vertex-sequential algorithms generate an initial feasible coloring, basic vertex-sequential heuristic methods produce an approximate solution in the following way. Vertex v_1 is assigned color 1. If vertices $v_1,v_2,..,v_{k-1}$ are assigned colors $f(v_1),f(v_2),..,f(v_{k-1})$ then

$$f(v_k) = \min\{j: \forall i \le k-1, \langle v_k, v_i \rangle \in E \Rightarrow j \neq f(v_i)\} \tag{29}$$

Two well known heuristic ordering rules are *largest first* and *smallest last* [*MB83,MM72*]. The largest first rule requires that $\deg(v_i) \ge \deg(v_{i+1})$ for $1 \le i \le n$. Using the smallest last rule, v_n is selected to be a vertex with minimum degree in G. For $i=n-1,n-2,..,1$, v_i is selected to be a vertex which has minimum degree in the induced subgraph of G on $V-\{v_n,v_{n-1},..,v_{i+1}\}$. The process of applying the largest first ordering rule and then using the vertex-sequential coloring method will be referred



i	deg(v_i)	$f_{LF}(v_i)$	$f_{SL}(v_i)$
1	4	2	2
2	3	3	4
3	4	1	3
4	4	2	2
5	3	3	3
6	5	1	1
7	3	2	2
8	4	3	1

SLF Order: $v_6, v_1, v_3, v_4, v_8, v_2, v_5, v_7$

SSL Order: $v_8, v_7, v_3, v_4, v_6, v_5, v_1, v_2$

Figure 12. SGCP Instance Solved with SLF and SSL Algorithms

to as the *Standard Largest First Algorithm*, denoted SLF. Similarly, the heuristic vertex-sequential coloring procedure which orders the vertices of a standard graph according to the smallest last rule will be termed the *Standard Smallest Last Algorithm*, designated SSL.

Example 12

Consider the standard graph in Figure 12. Using the SLF and SSL ordering rules, where ties are broken by selecting the vertex with smallest subscript, produces the indicated vertex processing order and results in the specified coloring. Note that the SLF algorithm uses 3 colors and the SSL method requires 4.

A third heuristic vertex-sequential coloring method which has received much attention is often referred to as the *Dsatur* algorithm [*Br79*]. This algorithm applies criteria which are identical to those used by the exact Vertex-Sequential with Dynamic Reordering Algorithm to generate an initial feasible solution. Specifically, recall that if $f(v_1),f(v_2),..,f(v_{k-1})$ is a partial coloring of SG=(V,E), the *colored degree*

TABLE XIV

SDSATUR HEURISTIC ALGORITHM
FOR THE SGCP

Iter	Assignment	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
0		(0,4)	(0,3)	(0,4)	(0,4)	(0,3)	(0,5)	(0,3)	(0,4)
1	$f(v_6)=1$	(1,3)	(1,2)	(0,4)	(1,3)	(1,2)	-	(1,2)	(0,4)
2	$f(v_1)=2$	-	(2,1)	(0,4)	(1,3)	(2,1)	-	(1,2)	(1,3)
3	$f(v_2)=3$	-	-	(1,3)	(1,3)	(2,1)	-	(1,2)	(1,3)
4	$f(v_5)=3$	-	-	(1,3)	(2,2)	-	-	(1,2)	(1,3)
5	$f(v_4)=2$	-	-	(2,2)	-	-	-	(1,2)	(2,2)
6	$f(v_3)=1$	-	-	-	-	-	-	(2,1)	(3,1)
7	$f(v_8)=3$	-	-	-	-	-	-	(3,0)	-
8	$f(v_7)=2$	-	-	-	-	-	-	-	-

of vertex v_j, $j=k,k+1,..,n$, denoted $cdeg(v_j)$, is defined to be the number of differently colored vertices which are adjacent to v_j. The *uncolored degree* of v_j, denoted $ucdeg(v_j)$, is defined to be the number of as yet uncolored vertices adjacent to v_j. Instead of considering the ordering of the vertices as static or predetermined, both algorithms dynamically select v_k to be the vertex which has the largest colored degree. If ties occur then the vertex with largest uncolored degree is chosen. The heuristic vertex-sequential algorithm which employees the Dsatur ordering rule will be termed the *Standard Dsatur Algorithm*, denoted SDsatur.

Example 13

Consider again the problem in Figure 12. Table XIV illustrates the application of the SDsatur algorithm to the problem. For each iteration, it displays the selected vertex, its assigned color, and the values of the ordered pair $(cdeg(v_i),ucdeg(v_i))$. If ties occurred, the vertex with of lowest index was selected. Notice that the procedure produced a 3-coloring.

B. BASIC VERTEX-SEQUENTIAL ALGORITHMS FOR THE CGCP

Basic vertex-sequential algorithms for the CGCP also use an ordering rule for

controlling the sequence in which the vertices are colored. Assuming that this ordering is represented by $v_1,v_2,..,v_n$, such an algorithm will process the vertices in the following way. Vertex v_1 is assigned colors $\{1,2,..,c(v_1)\}$. If vertices $v_1,v_2,..,v_{k-1}$ are assigned color sets $F(v_1),F(v_2),..,F(v_{k-1})$ then

$$F(v_k) = \{r, r+1,.., r+c(v_k)-1\} \text{ where } r = \min\{j: \forall i \le k-1, <v_k,v_i> \in E \Rightarrow \{j,j+1,..,j+c(v_k)-1\} \cap F(v_i) = \phi\} \quad (30)$$

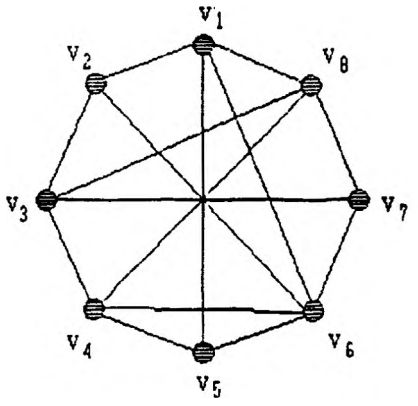
Clementson and Elphick developed and tested two generalizations of the largest first ordering rule for the SGCP which they called LF1 and LF2 [*CE83*]. The LF1 rule specifies that the vertices are processed in decreasing chromaticity order with ties broken by selecting the vertex with largest chromatic degree. If the LF2 rule is used then the vertices are colored in decreasing chromatic degree order with ties broken by choosing the vertex with largest chromaticity.

Roberts described two other generalizations of the LF ordering approach which were denoted LFCD and LFPH [*Ro87*]. The LFCD method processes the vertices in decreasing order by the value of the product of vertex chromaticity and degree. The LFPH rule sequences the vertices in decreasing order by their static pigeon hole measure. The *static pigeon hole* measure of vertex v_i, i = 1,2,..,n, denoted $SPH(v_i)$, is defined by

$$SPH(v_i) = ChromDeg(v_i) + (c(v_i) - 1)\, deg(v_i) - 1 \quad (31)$$

The static pigeon hole measure can be interpreted as an indicator of how hard a vertex will be to color. Thus, hard vertices are processed first.

Clementson and Elphick, as well as Roberts, showed that LF1 is significantly superior to LF2 with respect to mean number of colors used. The results reported by Roberts indicated that when the Truncated Poisson distribution was used to generate vertex chromaticities, then the difference in the average number of colors used by LF1, LFCD, and LFPH was always less than 1 and typically less than 0.50. Hence, it was decided that all vertex-sequential algorithms in this study would assign primary selection priority to vertex chromaticity. Also, it is easy to justify that if the



i	$c(v_i)$	$ad(v_i)$	$d(v_i)$	$F(v_i)$
1	2	8	4	{4,5}
2	1	7	3	{3}
3	3	6	4	{4,5,6}
4	1	10	4	{8}
5	2	5	3	{6,7}
6	2	7	5	{1,2}
7	1	8	3	{7}
8	3	7	4	{1,2,3}

CLF Order: $v_8, v_3, v_1, v_6, v_5, v_4, v_7, v_2$

CSL Order: $v_8, v_3, v_1, v_6, v_5, v_4, v_7, v_2$

Figure 13. CGCP Instance Solved with CLF and CSL Algorithms

adjacent chromatic degree of a vertex had been used instead of its chromatic degree, the LF1 algorithm would have produced identical results. Specifically, suppose that $c(v_i) = c(v_j)$ and $AdjChromDeg(v_i) > AdjChromDeg(v_j)$. Then, it follows that $c(v_i) + AdjChromDeg(v_i) > c(v_j) + AdjChromDeg(v_j)$. Hence, by definition, it must be that $ChromDeg(v_i) > ChromDeg(v_j)$. In this study, all composite vertex-sequential algorithms utilize the adjacent chromatic degree of a vertex for secondary ordering purposes.

Finally, Clementson and Elphick do not specify how to select from a group of vertices which have the same chromaticity and chromatic degree. In such cases, the vertex with the largest degree will be picked. The heuristic coloring method which orders vertices in decreasing order with first priority given to vertex chromaticity, second to adjacent chromatic degree, and third to degree will be called the *Composite Largest First Algorithm*, designated CLF.

For comparison purposes the SSL algorithm was also generalized into a composite graph coloring heuristic. Accordingly, v_n is selected to be a vertex with minimum chromaticity. If more than one vertex qualifies, then the one with the smallest adjacent chromatic degree is chosen. In the case that a tie is still present, the vertex with lowest degree is picked. For $i = n-1, n-2, .., 1$, the same criteria are

applied to select v_i from the induced subgraph of G on $V-\{v_n,v_{n-1},..,v_{i+1}\}$. This procedure was named the *Composite Smallest Last Algorithm*, abbreviated CSL.

Example 14

Figure 13 displays an instance of the CGCP. The included table documents for each vertex its chromaticity $c(v_i)$, adjacent chromatic degree $ad(v_i)$, degree $d(v_i)$, and assigned color set. Using both the CLF and CSL ordering rules, with ties broken by selecting the vertex with smallest index, produced identical vertex processing sequences. Note that 8 colors were required.

The generalization of the SDsatur algorithm will be denoted the *CDsatur* algorithm. It implements a rule which orders the vertices dynamically by maximum chromaticity, colored degree, uncolored adjacent chromatic degree, and finally uncolored degree. Note that the CDsatur heuristic places a higher priority on vertex chromaticity than colored degree. Recall that the exact Vertex-Sequential with Dynamic Vertex Reordering Algorithm reversed the criteria. Experimentation showed that, although the initial feasible solution tends to use fewer colors when chromaticity is given precedence over colored degree, the mean total search time of the exact algorithm is reduced if colored degree is given first priority.

Example 15

For the CGCP problem instance in Figure 13, Table XV illustrates the use of the CDsatur algorithm. For each iteration, it displays the selected vertex, its assigned color set, and the values of the ordered pairs $(c(v_i),cdeg(v_i))$ and $(ucAdjChromDeg(v_i),ucdeg(v_i))$. If ties occurred, the vertex with of lowest index was selected. Notice that the procedure produced an 8-coloring.

Roberts developed and tested two algorithms that are similar to CDsatur in that the order of the vertices is determined dynamically. That is, the identity of v_k depends on the colors which have been assigned to $v_1,v_2,..,v_{k-1}$ and, thus, it can not

TABLE XV

CDSATUR HEURISTIC ALGORITHM
FOR THE CGCP

Iter	Assignment	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
0		(2,0)	(1,0)	(3,0)	(1,0)	(2,0)	(2,0)	(1,0)	(3,0)
		(8,4)	(7,3)	(6,4)	(10,4)	(5,3)	(7,5)	(8,3)	(7,4)
1	$F(v_8)=\{1,2,3\}$	(2,3)	(1,0)	(3,3)	(1,3)	(2,0)	(2,0)	(1,3)	-
		(5,3)	(7,3)	(3,3)	(7,3)	(5,3)	(7,5)	(5,2)	
2	$F(v_3)=\{4,5,6\}$	(2,3)	(1,3)	-	(1,6)	(2,0)	(2,0)	(1,6)	-
		(5,3)	(4,2)		(4,2)	(5,3)	(7,5)	(2,1)	
3	$F(v_1)=\{4,5\}$	-	(1,3)	-	(1,6)	(2,2)	(2,2)	(1,6)	-
			(2,1)		(4,2)	(3,2)	(5,4)	(2,1)	
4	$F(v_6)=\{1,2\}$	-	(1,5)	-	(1,6)	(2,4)	-	(1,6)	-
			(0,0)		(2,1)	(1,1)		(0,1)	
5	$F(v_5)=\{6,7\}$	-	(1,5)	-	(1,7)	-	-	(1,6)	-
			(0,0)		(0,0)			(0,1)	
6	$F(v_4)=\{8\}$	-	(1,5)	-	-	-	-	(1,6)	-
			(0,0)					(0,1)	
7	$F(v_7)=\{7\}$	-	(1,5)	-	-	-	-	-	-
			(0,0)						
8	$F(v_2)=\{3\}$	-	-	-	-	-	-	-	-

be determined until they have been colored. The two methods were called DYNPH and DYNFPH. The DYNPH algorithm selects as v_k the uncolored vertex with the largest pigeonhole measure and the DYNFPH picks the one with maximum floating point pigeonhole measure. The *pigeonhole measure* of a vertex v_k, $1 \le k \le n$, is defined by

$$PH(v_k) = (c(v_k) - 1)\,(NG(v_k) + ucdeg(v_k)) + cdeg(v_k) + ucAdjChromDeg(v_k) - M \tag{32}$$

where M is an upper bound on the number of colors the algorithm will use and $NG(v_k)$ is the number of gaps in an adjacent color bit array which the method associates with v_k. The *floating point pigeonhole measure* of v_k is defined by

$$FPH(v_k) = \frac{PH(v_k)}{NG(v_k) + ucdeg(v_k)} \tag{33}$$

The computational results reported in the Roberts paper placed the DYNPH and DYNFPH algorithms into the same effectiveness category as LF1, LFPH, and LFCD. As previously indicated, it was determined that for this study LF1 would be used to represent this group.

C. VERTEX-SEQUENTIAL WITH INTERCHANGE FOR THE SGCP

Matula, Marble, and Issacson [*MM72*] and Johnson [*Jo74*] described *interchange* techniques which can be integrated into any standard vertex-sequential algorithm. They demonstrated that in the case of SLF and SSL the resulting algorithm significantly reduces the average number of colors used.

A *(c_1,c_2)-interchange* is defined to be the process of assigning color c_1 to all vertices currently colored c_2 and vice versa. Given any standard graph SG=(V,E), assume that vertices $v_1,v_2,..,v_{k-1}$ have been assigned colors $f(v_1),f(v_2),...,f(v_{k-1})$ using the distinct colors 1,2,..,MaxColor. A vertex-sequential with interchange algorithm will first attempt to find a color c, where $c \leq$ MaxColor, which can be assigned to v_k. If this is not possible, an attempt is made to find c_1 and c_2, where $1 \leq c_1 < c_2 \leq$ MaxColor, such that a (c_1,c_2)-interchange will allow v_k to be assigned either c_1 or c_2. The method for identifying c_1 and c_2 distinguish the Matula, Marble, and Issacson version from the Johnson approach. These two different schemes will be referred to as *I1* and *I2*, respectively.

Consider first the I1 method. Let C be the set of colors defined by

$$C = \{c : \exists! i \; s.t. \; f(v_i) = c \wedge \langle v_i, v_k \rangle \in E\}$$

That is, C is the set of all colors such that there is *exactly one* vertex adjacent to v_k which is currently assigned that color. If there exist $c_1,c_2 \in C$ such that a (c_1,c_2) connected component in the induced subgraph on $\{v_1,v_2,..,v_k\}$ has only *one* vertex adjacent to v_k, then a (c_1,c_2)-interchange on just that component will result in making either c_1 or c_2 a feasible assignment for v_k. If this is not possible, then v_k is assigned MaxColor+1 and the algorithm proceeds to v_{k+1}. The SLF, SSL, and SDsatur

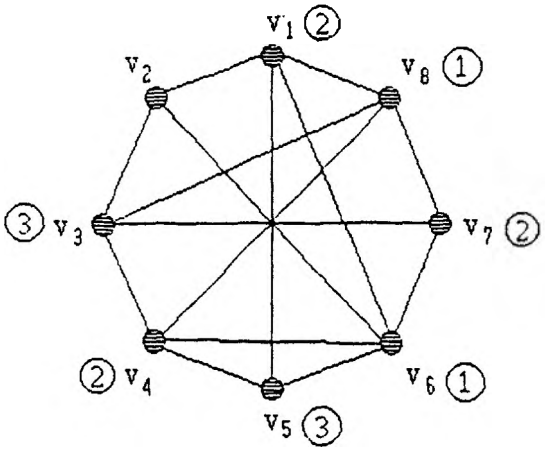


Figure 14. The I1 and I2 Interchange Techniques for the SGCP

algorithms with the I1 interchange technique incorporated will be designated *SLFI1*, *SSLI1*, and *SDsaturI1*, respectively.

Example 16

The SSL algorithm produced a 4-coloring of the graph pictured in Figure 12. Figure 14 depicts the situation at the point at which v_2 is being processed and SSL would be forced to introduce color 4. The SSLI1 algorithm would calculate that $C=\{1,2,3\}$. There are two (1,3) connected components in the graph, (v_3,v_8) and (v_5,v_6). Since (v_3,v_8) has only v_3 adjacent to v_2, a (1,3)-interchange on that component would allow v_2 to be assigned color 3. Therefore, the SSLI1 algorithm would produce a 3-coloring of the graph.

For each pair of values for c_1 and c_2, where $1 \le c_1 < c_2 \le \text{MaxColor}$, the I2 technique attempts to identify an acceptable interchange by finding *all* (c_1,c_2) connected components in the subgraph induced on $\{v_1,v_2,..,v_k\}$. If none of these components contain 2 differently colored vertices adjacent to v_k, then a (c_1,c_2)-interchange is performed on only those components which have c_1 adjacent to v_k.

This will now allow v_k to be assigned color c_1. The SLF, SSL, and SDsatur algorithms with the I2 interchange technique incorporated will be denoted *SLFI2*, *SSLI2*, and *SDsaturI2*, respectively.

Example 17

Reconsider the situation pictured in Figure 14. The list of (1,3) connected components consists of (v_3,v_8) and (v_5,v_6). Each one of these components has either a vertex colored 1 or a vertex colored 3 adjacent to v_2, but, not both. Hence, if a (1,3)-interchange is performed on all those components with 1 adjacent to v_2, namely (v_5,v_6), then v_2 can be colored with color 1. Thus, the SSLI2 algorithm would produce a 3-coloring of the graph.

D. VERTEX-SEQUENTIAL WITH INTERCHANGE FOR THE CGCP

Clementson and Elphick described an interchange technique which can be used in conjunction with any heuristic vertex-sequential method for the CGCP [*CE83*]. They also presented experimental results which demonstrated that, for $n=100$ and $p=0.2$, 0.3, and 0.4, this approach, when integrated into CLF, significantly reduced the mean number of colors used. This interchange scheme will be denoted *I1*. A second interchange method, which will be referred to as *I2*, was designed for this study.

Given a composite graph $CG=(V,E)$, assume that $v_1,v_2,..,v_{k-1}$ have been assigned color sets $F(v_1),F(v_2),..,F(v_{k-1})$, respectively, using colors 1,2,..,MaxColor. In general, a vertex-sequential with interchange algorithm will first identify the smallest color c such that $C=\{c,c+1,..,c+c(v_k)-1\}$ is a feasible assignment to v_k. If $c+c(v_k)-1 \leq$ MaxColor, then C is assigned to v_k and the procedure moves on to v_{k+1}. Otherwise, an attempt is made to identify one or more vertices whose current color assignments can be changed so as to allow v_k to be colored and at the same time minimize the required net increase in the value of MaxColor. The way in which this is done distinguishes I1 from I2.

The I1 approach first calculates the value of a color set P defined by

$$P = \{p_1, p_2, .., p_r\} \text{ where } p_i \le c-1,\ 1 \le i \le r, \text{ and}$$
$$\exists!\ v_j,\ 1 \le j \le k-1 \text{ s.t. } \langle v_j, v_k \rangle \in E \wedge F(v_i) \cap \{p_i, p_i+1, .., p_i+c(v_k)-1\} \neq \phi \quad (35)$$

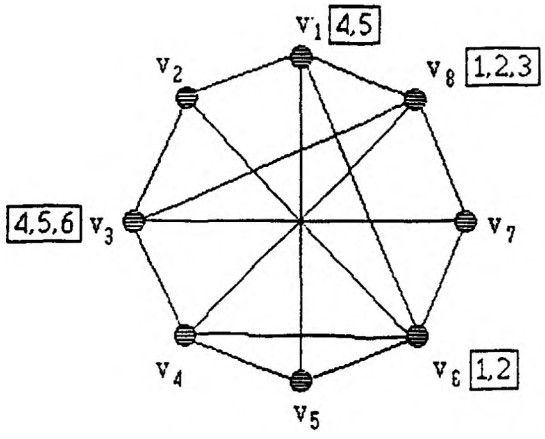
If P is empty, then v_k is assigned the color set C and the algorithm proceeds to process v_{k+1}. Otherwise, I1 calculates a vertex set $W=\{w_1,w_2,..,w_r\}$ which is the set of unique vertices associated with the elements of P. A second color set $Q=\{q_1,q_2,..,q_r\}$ is then generated, where q_i is the smallest first color which could be assigned to w_i assuming v_k were assigned color set $C_i=\{p_i,p_i+1,..,p_i+c(v_k)-1\}$, $1 \le i \le r$. Finally, a third color set R is produced where R is defined by

$$R = \{p_i \in P : q_i + c(v_i) - 1 < c + c(v_k) - 1\} \quad (36)$$

If R is empty, the procedure assigns v_k the color set C and moves on to v_{k+1}. But, if R is not empty, then there exits at least one w_i which can be recolored with $\{q_i,q_i+1,..,q_i+c(w_i)-1\}$ to allow v_k to be colored $\{p_i,p_i+1,..,p_i+c(v_k)-1\}$ that will avoid increasing MaxColor to $c+c(v_k)-1$. The vertex w_i is selected which will minimize the maximum value of the pair $q_i+c(w_i)-1$ and $p_i+c(v_k)-1$. The CLF, CSL, and CDsatur algorithms with the I1 interchange technique incorporated will be referred to as *CLFI1*, *CSLI1*, and *CDsatur11*, respectively.

Example 18

Recall that the CLF algorithm when applied to the CGCP instance displayed in Figure 13 produced an 8-coloring of the graph. Figure 15 depicts the situation at the point at which v_5 is being processed and CLF would be forced to assign it the color set {6,7}. But, this would increase MaxColor from 6 to 7. However, applying I1 produces $P=\{1,2,3,4,5\}$, $W=\{v_6,v_6,v_1,v_1,v_1\}$, $Q=\{6,6,5,6,7\}$, and $R=\{3\}$. Therefore, v_1 can be reassigned {5,6} to allow v_5 to be colored {3,4} and avoid increasing MaxColor. Moving on, v_4 can be assigned {7}. Although MaxColor is increased, this can not be avoided since $\{v_3,v_4,v_8\}$ is a clique. Vertex v_7 can now be colored with {7} and finally v_2 can be assigned {3}. Thus, CLFI1 will color the graph with 7 colors instead of the 8 used by CLF.



i	$c(v_i)$	$ad(v_i)$	$d(v_i)$
1	2	8	4
2	1	7	3
3	3	6	4
4	1	10	4
5	2	5	3
6	2	7	5
7	1	8	3
8	3	7	4

Figure 15. The I1 and I2 Interchange Techniques for the CGCP

The I2 interchange method for the CGCP is a generalization of the I2 technique for the SGCP. Given color sets C_1 and C_2 where $n(C_1)=n(C_2)$ and $C_1 \cap C_2 = \phi$, a $C_1 \cup C_2$ connected component is said to be *pure* if and only if each color set in the component is either C_1, C_2, or a proper subset of C_1 or C_2. Let $C_1=\{c_{11},c_{12},..,c_{1r}\}$ and $C_2=\{c_{21},c_{22},..,c_{2r}\}$. A *$(C_1,C_2)$-interchange* involves making $(c_{11},c_{21}),(c_{12},c_{22}),..,(c_{1r},c_{2r})$-interchanges. If v_k requires interchange processing, then the I2 method considers any pair of color sets which contain the same number of consecutive colors, do not intersect, and have orders that are at least as large as $c(v_k)$ to be potential interchange sets. For each potential pair of color sets C_1 and C_2, the I2 approach attempts to identify an acceptable interchange by finding $C_1 \cup C_2$ connected components in the subgraph induced on $\{v_1,v_2,..,v_k\}$. If these components are pure and none of them contain 2 vertices adjacent to v_k that have been assigned subsets of both C_1 and C_2, then a (C_1,C_2)-interchange is performed on only those components which have subsets of C_1 adjacent to v_k. This will now allow v_k to be assigned a subset of C_1. The CLF, CSL, and CDsatur algorithms with the I2 interchange technique incorporated will be designated *CLFI2*, *CSLI2*, and *CDsatur12*, respectively.

Example 19

In Figure 15, suppose I2 were applied in an attempt to color v_5 without having to increase MaxColor to 7. The only potential interchange color sets which involve only pure components are $C_1 = \{1,2,3\}$ and $C_2 = \{4,5,6\}$. There is exactly one $C_1 \cup C_2$ connected component, (v_3,v_8,v_1,v_6). But, this component has both v_1 and v_6 adjacent to v_5 with v_1 assigned a subset of C_2 and v_6 assigned a subset of C_1. Therefore, the I2 method does not produce an interchange and MaxColor must be increased.

E. COLOR-SEQUENTIAL ALGORITHMS FOR THE SGCP

Heuristic color-sequential algorithms are similar to exact color-sequential algorithms in that all vertices which are to be assigned a given color are processed before assigning any vertex the next color. Also, each color is assigned to as many vertices as possible before the next color is introduced.

Leighton described a heuristic color-sequential algorithm named *RLF* which, according to reported computational experience, uses a smaller mean number of colors than other known one pass color-sequential methods [*Le79*]. Suppose $SG=(V,E)$ is a standard graph which is to be colored using the RLF algorithm. Assume that colors 1,2,..,c-1 have been completely assigned. Let U_1 be the set of currently uncolored vertices. The first vertex that is assigned color c is the one which has the largest degree in U_1. Succeeding vertices are identified by repetitively assigning to U_1 all uncolored vertices that are *not* adjacent to any vertex which has been allocated c and U_2 all uncolored vertices that are adjacent to such a vertex. If U_1 is not empty, the next vertex which is assigned color c is the one with largest degree in U_2. Ties are broken by picking the vertex which has smallest degree in U_1. Otherwise, the algorithm moves on to c+1 unless all vertices have been colored, in which case, it terminates. In this paper, the RLF algorithm for the SGCP is denoted *SRLF*.

Example 20

Applying the SRLF algorithm to the SGCP problem instance in Figure 12

would result in a 3-coloring. The following outline summarizes the various iterations the algorithm would generate.

$c=1$
$U_1=\{v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8\} \rightarrow f(v_6)=1$
$U_1=\{v_3,v_8\},\ U_2=\{v_1,v_2,v_4,v_5,v_7\} \rightarrow f(v_3)=1$
$U_1=\phi,\ U_2=\{v_1,v_2,v_4,v_5,v_7,v_8\}$
$c=2$
$U_1=\{v_1,v_2,v_4,v_5,v_7,v_8\} \rightarrow f(v_1)=2$
$U_1=\{v_4,v_7\},\ U_2=\{v_2,v_5,v_8\} \rightarrow f(v_4)=2$
$U_1=\{v_7\},\ U_2=\{v_2,v_5,v_8\} \rightarrow f(v_7)=2$
$W_1=\phi,\ W_2=\{v_2,v_5,v_8\}$
$c=3$
$U_1=\{v_2,v_5,v_8\} \rightarrow f(v_2)=3$
$U_1=\{v_5,v_8\},\ U_2=\phi \rightarrow f(v_5)=3$
$U_1=\{v_8\},\ U_2=\phi \rightarrow f(v_8)=3$
$U_1=\phi,\ U_2=\phi$

F. COLOR-SEQUENTIAL ALGORITHMS FOR THE CGCP

Heuristic color-sequential algorithms for the CGCP process all vertices that are to be assigned a color set beginning with color c before any vertex which is to be assigned a color set beginning with color c+1. Roberts developed two such algorithms. They were named RLF1 and RLFD1, since both are generalizations of SRLF [*Ro87*].

Consider first RLF1. Assume that all color sets which begin with 1,2,..,c-1 have been completely assigned. For each uncolored vertex, define its *minimum color* to be the smallest color that could be allocated to it given the current color assignments. Let U_1 be the set of uncolored vertices whose minimum color is c. The first vertex that is assigned a color set beginning with c is the U_1 vertex which has the largest chromaticity. Ties are broken by selecting the vertex with largest chromatic degree in U_1. Succeeding vertices are identified by repetitively assigning to U_1 all uncolored vertices that are *not* adjacent to any vertex which has been allocated a color set beginning with c and to U_2 all uncolored vertices that are adjacent to such a vertex. If U_1 is not empty, the next vertex which is assigned a color set beginning with c is selected according to maximum chromaticity, maximum

U_2 adjacent chromatic degree, and minimum U_1 chromatic degree in that order of precedence. Otherwise, the algorithm moves on to c + 1 unless all vertices have been colored, in which case, it terminates.

RLFD1 is identical to RLF1 except that vertex selection is based on degree instead of chromatic degree or adjacent chromatic degree. The computational results reported in the Robert's paper suggested little difference in the mean number of colors used by RLF1 and RLFD1. Hence, it was decided that for the experiments conducted in this study only RLF1 would be implemented. Actually, RLF1 and RLFD1 were integrated in the following way. When the first vertex is selected from U_1, the selection criteria, in their order of precedence, are maximum chromaticity, maximum chromatic degree in U_1, and maximum degree in U_1. Succeeding U_1 vertices are picked according maximum chromaticity, maximum U_2 adjacent chromatic degree, maximum U_2 degree, minimum U_1 chromatic degree, and minimum U_1 degree. All future references to this algorithm will be denoted *CRLF*.

Example 21

The CRLF algorithm was used to color the problem pictured in Figure 13 and a 7-coloring was produced. The following outline summarizes the various iterations of the algorithm.

c=1
$U_1=\{v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8\}$, $U_2=\phi \rightarrow F(v_8)=\{1,2,3\}$
$U_1=\{v_2,v_5,v_6\}$, $U_2=\{v_1,v_3,v_4,v_7\} \rightarrow F(v_6)=\{1,2\}$
$U_1=\phi$, $U_2=\{v_1,v_2,v_3,v_4,v_5,v_7\}$

c=2
$U_1=\phi$, $U_2=\{v_1,v_2,v_3,v_4,v_5,v_7\}$

c=3
$U_1=\{v_2,v_5\}$, $U_2=\{v_1,v_3,v_4,v_7\} \rightarrow F(v_5)=\{3,4\}$
$U_1=\{v_2\}$, $U_2=\{v_1,v_3,v_4,v_7\} \rightarrow F(v_2)=\{3\}$
$U_1=\phi$, $U_2=\{v_1,v_3,v_4,v_7\}$

c=4
$U_1=\{v_3,v_7\}$, $U_2=\{v_1,v_4\} \rightarrow F(v_3)=\{4,5,6\}$
$U_1=\phi$, $U_2=\{v_1,v_4,v_7\}$

c=5
$U_1=\{v_1\}$, $U_2=\{v_4,v_7\} \rightarrow F(v_1)=\{5,6\}$
$U_1=\phi$, $U_2=\{v_4,v_7\}$

$c=6$
$U_1=\phi$, $U_2=\{v_4,v_7\}$
$c=7$
$U_1=\{v_4,v_7\}$, $U_2=\phi \rightarrow F(v_4)=\{7\}$
$U_1=\{v_7\}$, $U_2=\phi \rightarrow F(v_7)=\{7\}$
$U_1=\phi$, $U_2=\phi$

G. EXPERIMENTS -- DESCRIPTION AND RESULTS

1. Introductory Comments.

Roberts implemented LF1, LF2, LFPH, LFCD, DYNPH, DYNFPH, LF1I1, LF2I1, LFPHI1, LFCDI1, RLF1, and RLFD1. Each of these algorithms were used to color 25 instances of the CGCP for each possible combination of $n=100$ and $p=0.10,0.15,0.20,0.30,0.40,0.50$ and for $n=200,300,400,500$ and $p=0.10,0.15,0.20$. Also, in addition to the Truncated Poisson distribution, four other probability distributions were used to produce vertex chromaticities. An analysis of the results of these experiments resulted in the twelve algorithms being placed into the following three effectiveness categories.

Category 1: LF1I1, LFPHI1, LFCDI1, RLF1, RLFD1
Category 2: LF1, LFPH, LFCD, LF2I1, DYNPH, DYNFPH
Category 3: LF2

With respect to mean number of colors used the methods listed in category 1 were more effective than those in category 2. Likewise, those in category 2 consistently used a smaller average number of colors than did LF2. Of the methods listed in category 1, the vertex sequential with interchange method tended to be more effective for graphs of small order and the recursively largest first methods did better for graphs of larger order.

Example 22

Figure 16 displays a graph of the mean number of colors reportedly used by each of the methods in category 1 for graphs with $p=0.20$ and $n=100,200,300,400,500$ where the vertex chromaticities were generated according to the Truncated Poisson

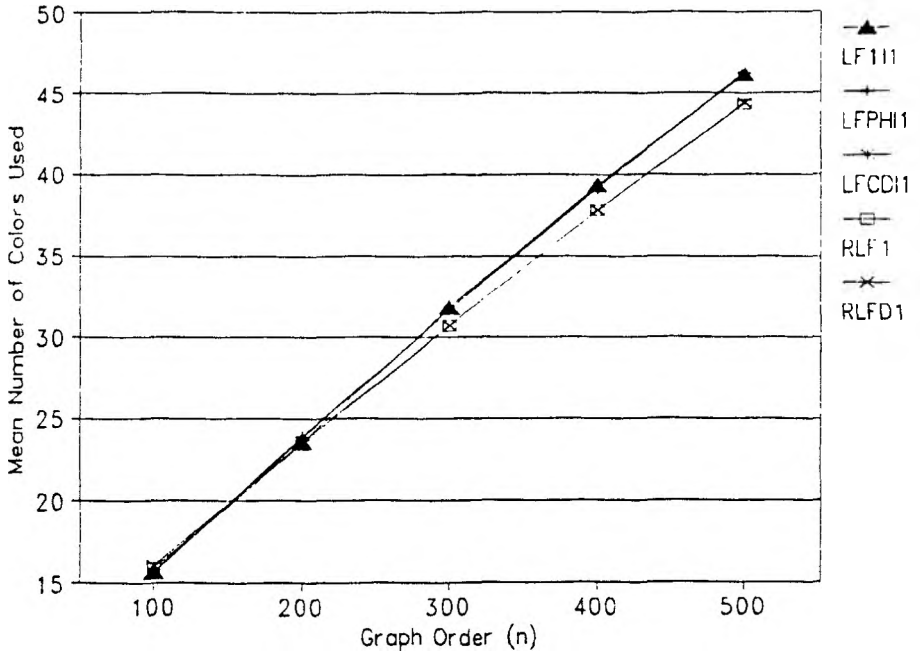


Figure 16. Category 1 Algorithms (p=0.20)

TABLE XVI

CATEGORY 1 ALGORITHMS
MEAN NUMBER OF COLORS USED
p=0.20

n	LF1I1	LFPHI1	LFCDI1	RLF1	RLFD1
100	15.68	15.52	15.72	15.88	16.04
200	23.56	23.84	23.88	23.40	23.56
300	31.84	31.64	31.68	30.68	30.68
400	39.40	39.16	39.24	37.84	37.80
500	46.20	46.24	46.28	44.32	44.36

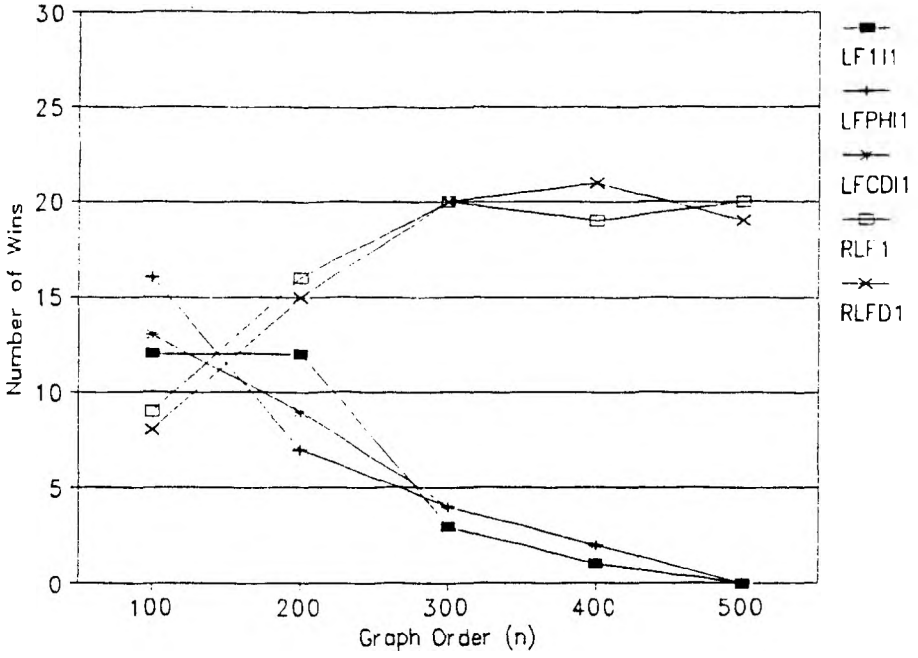


Figure 17. Category 1 Algorithms (p=0.20)

TABLE XVII

CATEGORY 1 ALGORITHMS
NUMBER OF WINS
p=0.20

n	LF1I1	LFPHI1	LFCDI1	RLF1	RLFD1
100	12	16	13	9	8
200	12	7	9	16	15
300	3	4	4	20	20
400	1	2	2	19	21
500	0	0	0	20	19

distribution. Table XVI lists the corresponding data. Figure 17 contains a graph of the number of times each method used no more colors than any of the other methods. This will be called the *number of wins*. Table XVII lists these values. Although the distinction for $n = 100$ is not large, the interchange methods did slightly better. But, as the order of graph gets larger, the RLF algorithms become increasingly more effective when compared to the interchange approaches. It is also apparent that there is not a significant difference among the interchange methods or between the two RLF algorithms.

The reason for the previous comments is to provide justification for the conclusion that implementing LF1, LFI1, and RLF1 as CLF, CLFI1, and CRLF would provide a basis for relating the results of this study to those reported by Roberts.

2. Algorithm Implementations and Experiments.

The decisions as to the algorithms to implement and experiments to conduct were based on the following considerations.

1. It should be possible to compare the results with those of the Roberts paper. Hence, at least some of the algorithms he tested would have to be implemented and some of the same graph definition parameters would have to be used.
2. The largest graph order tested by Roberts for $p = 0.50$ was $n = 100$. The typical test case for standard graph coloring heuristics seems to be $n = 1000$ and $p = 0.50$. Therefore, it was felt that graph orders for $p = 0.50$ should range up to 1,000 or at least as large as could be computed in a reasonable time period.
3. The SDsatur algorithm was reported to perform significantly better than SLF and SSL [*Br79,JA89*]. Thus, it was hypothesized that there was a good chance that its generalization would be relatively effective for the CGCP.
4. The general goal of this study was to compare properties of the SGCP with those of the CGCP. Hence, it would be necessary to implement and test

companion pairs of algorithms for both problem types.

The preceding considerations resulted in the following algorithm types being tested. The names which appear in parenthesis are those which were usually encountered in the literature.

SLF (LF)	CLF (LF1)
SSL (SL)	CSL
SDsatur (Dsatur)	CDsatur
SRLF (RLF)	CRLF (RLF1)
SLFI1 (LFI)	CLFI1 (LF1I)
SSLI1 (SLI)	CSLI1
SDsaturI1 (DsaturI)	CDsaturI1
SLFI2 (LFI)	CLFI2
SSLI2 (SLI)	CSLI2
SDsaturI2 (DsaturI)	CDsaturI2

The Appendix contains the implementations of these algorithms in Listings 4-13.

Each procedure was used to color 25 random graphs for every possible combination of p=0.20,0.50 and n=50,100,200,300,400,500 where, in the case of composite graphs, vertex chromaticities were generated according to the Truncated Poisson probability distribution. The SLF, SSL, SDsatur, SRLF, CLF, CSL, CDsatur, and CRLF algorithms were also tested on 10 graphs for each combination of p=0.20,0.50 and n=550,600,..,1000.

3. SGCP Heuristic Algorithm Results.

Table XVIII shows the results produced with the SGCP algorithms for p=0.20 and n=50,100,200,300,400,500. For each algorithm and graph order, it displays the observed mean values for the associated estimate of $\chi(SG_{n,p})$ and execution time in seconds. For each graph order, the minimum mean estimate for $\chi(SG_{n,p})$ is emphasized by being printed in bold. The table also displays the number of wins credited to each procedure, with maximum values displayed in bold. For example, for n=500, the SRLF algorithm used an average of 25.20 colors, required on the average 33.09 seconds of computing, and produced the minimum coloring for all 25 sample graphs. Table XIX contains the same categories of information for p=0.50. Figures 18 and 19 contain graphs of the mean number of colors used by SRLF,

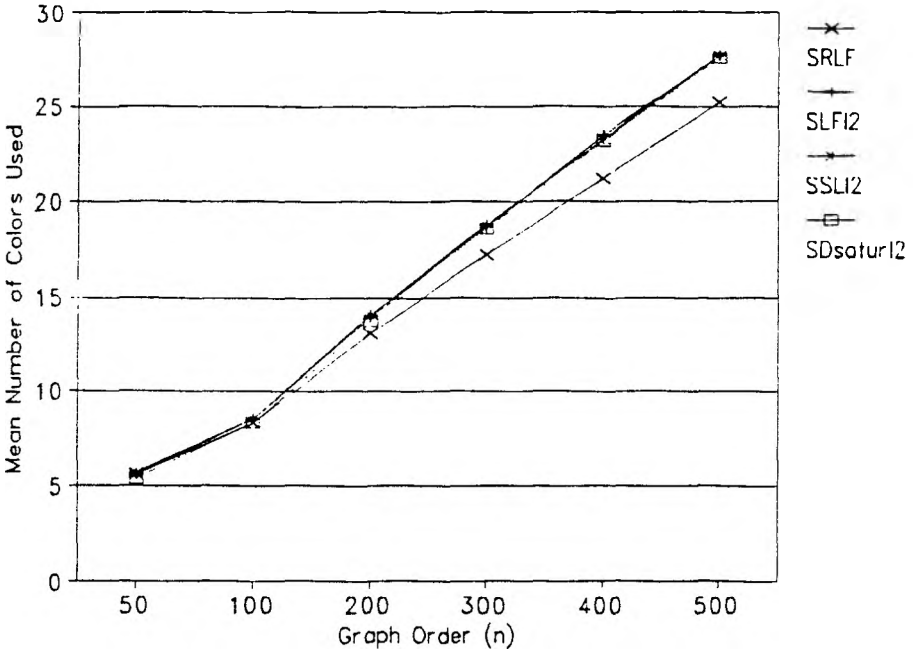


Figure 18. SGCP Heuristic Algorithms (p=0.20)

TABLE XVIII

STANDARD GRAPH COLORING
HEURISTIC ALGORITHM RESULTS
p=0.20

n	SLF	SSL	SDsatur	SRLF	SLFI1	SSLI1	SDstrI1	SLFI2	SSLI2	SDstrI2
50	6.48	6.36	5.68	5.60	5.92	5.84	5.64	5.60	5.72	**5.40**
	0.05	0.09	0.11	0.16	0.19	0.19	0.20	0.13	0.14	0.15
	3	4	15	17	9	11	16	17	14	22
100	9.68	9.96	8.68	**8.32**	8.92	9.08	8.48	8.56	8.56	**8.32**
	0.19	0.31	0.42	0.69	0.77	0.81	0.79	0.89	0.94	1.78
	0	0	9	**18**	4	2	14	12	12	**18**
200	15.88	16.04	13.80	**13.08**	14.28	14.12	13.64	13.92	14.08	13.76
	0.75	1.22	1.58	3.44	3.86	4.20	5.61	6.34	6.93	28.85
	0	0	7	**24**	1	4	10	4	2	7
300	20.80	21.20	18.48	**17.24**	19.00	18.84	18.36	18.68	18.76	18.56
	1.69	2.73	3.53	9.11	11.72	12.48	25.23	21.65	22.89	143.55
	0	0	4	**25**	1	1	3	0	1	4
400	25.60	26.28	22.92	**21.24**	23.64	23.32	22.88	23.44	23.28	23.16
	2.95	4.81	6.22	18.63	27.24	30.56	86.40	52.32	55.27	400.90
	0	0	1	**25**	0	0	1	0	0	0
500	30.20	30.84	26.92	**25.20**	27.68	27.72	27.40	27.68	27.68	27.52
	4.58	7.49	9.65	33.09	56.85	66.07	292.60	103.25	103.25	831.13
	0	0	0	**25**	0	0	0	0	0	0

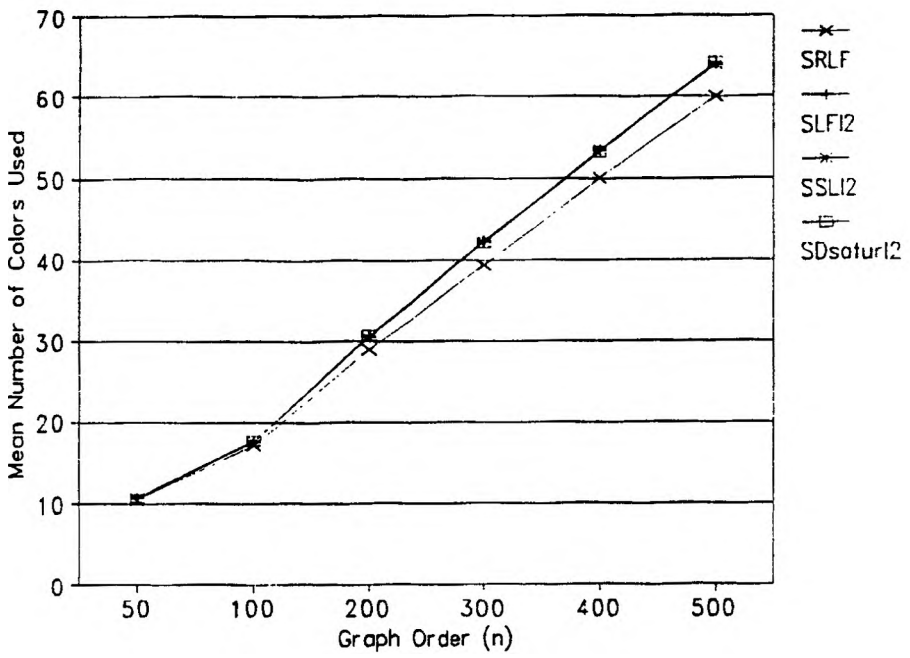


Figure 19. SGCP Heuristic Algorithms (p=0.50)

TABLE XIX

STANDARD GRAPH COLORING
HEURISTIC ALGORITHM RESULTS
p=0.50

n	SLF	SSL	SDsatur	SRLF	SLFI1	SSLI1	SDstrI1	SLFI2	SSLI2	SDstrI2
50	12.04	12.16	10.80	**10.40**	10.96	10.88	10.80	10.60	10.64	10.52
	0.11	0.14	0.15	0.17	0.59	0.56	0.56	0.45	0.48	0.51
	0	0	7	**16**	5	5	6	11	11	13
100	19.68	20.32	18.40	**17.08**	18.08	18.04	18.00	17.72	17.72	17.68
	0.42	0.54	0.57	0.94	3.69	3.57	5.42	3.38	3.72	7.64
	0	0	3	**19**	3	4	5	9	8	7
200	33.68	34.32	31.00	**28.96**	30.72	30.92	30.80	30.72	30.48	30.64
	1.66	2.13	2.22	5.70	31.29	33.61	83.72	27.99	30.94	82.51
	0	0	0	**24**	1	2	1	1	2	3
300	46.36	47.12	42.96	**39.36**	42.36	42.24	42.56	42.32	42.08	42.12
	3.77	4.83	4.92	16.66	134.31	137.11	445.60	97.01	110.12	310.25
	0	0	1	**25**	1	0	1	0	0	0
400	58.16	59.28	54.16	**50.12**	53.68	53.60	53.84	53.44	53.52	53.20
	6.74	8.63	8.70	36.80	387.77	409.06	1485.89	253.56	274.47	801.32
	0	0	0	**25**	0	0	0	0	0	0
500	69.80	70.64	64.76	**60.08**	64.20	63.80	64.38	64.06	63.80	64.20
	11.80	14.85	13.53	67.67	767.91	901.32	2906.16	603.17	637.54	1470.32
	0	0	0	**25**	0	0	0	0	0	0

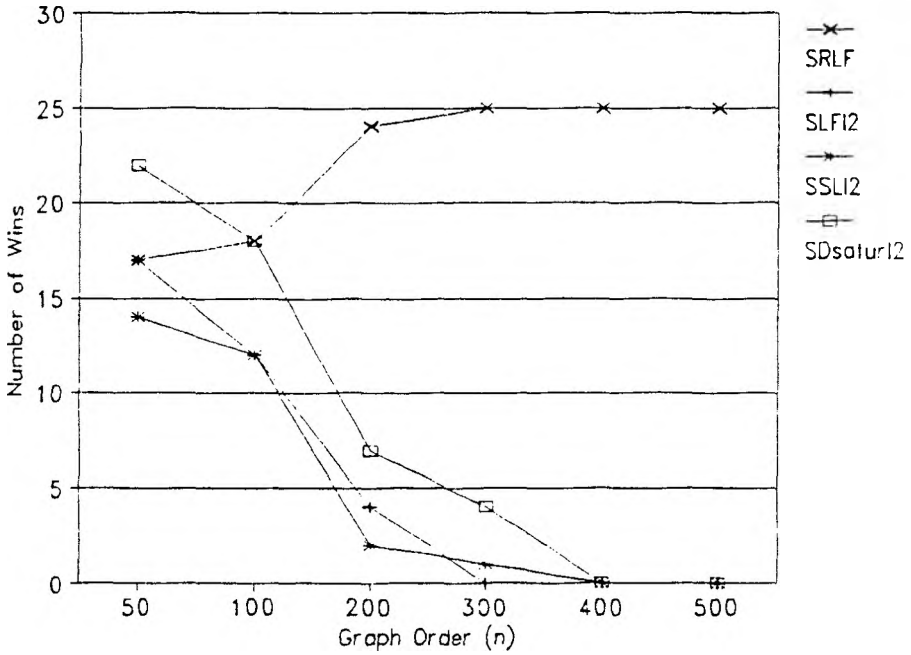


Figure 20. SGCP Heuristic Algorithms (p=0.20)

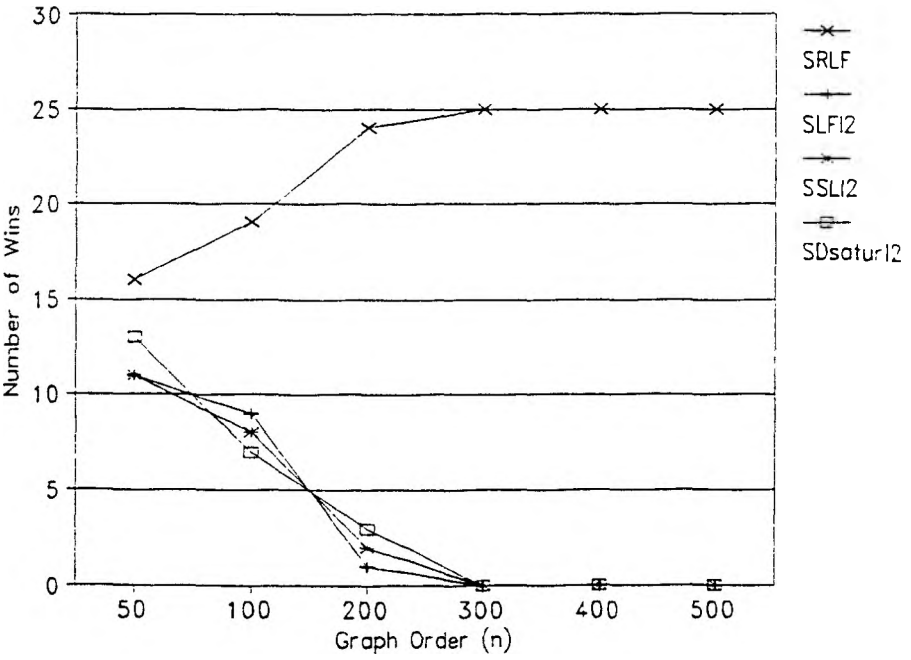


Figure 21. SGCP Heuristic Algorithms (p=0.50)

TABLE XX

STANDARD GRAPH COLORING
HEURISTIC ALGORITHM RESULTS

	p=0.20				p=0.50			
n	SLF	SSL	SDsatur	SRLF	SLF	SSL	SDsatur	SRLF
50	6.48	6.36	5.68	5.60	12.04	12.16	10.80	10.40
100	9.68	9.96	8.68	8.32	19.68	20.32	18.40	17.08
150	12.72	12.88	11.20	10.80	27.08	27.60	24.72	23.16
200	15.88	16.04	13.80	13.08	33.68	34.32	31.00	28.96
250	18.40	19.00	16.32	15.12	40.12	41.16	36.96	34.44
300	20.80	21.20	18.48	17.24	46.36	47.12	42.96	39.36
350	23.08	24.04	20.64	19.16	52.24	53.12	47.96	45.04
400	25.60	26.28	22.92	21.24	58.16	59.28	54.16	50.12
450	27.96	28.48	25.04	23.16	64.08	65.08	59.64	55.08
500	30.20	30.84	26.92	25.20	69.80	70.64	64.76	60.08
550	32.30	32.90	29.40	27.00	75.10	77.00	69.80	64.80
600	34.40	35.60	31.00	29.10	81.10	82.20	74.90	69.60
650	36.70	37.20	33.10	30.80	86.20	87.60	80.10	74.60
700	38.60	39.80	35.60	32.70	92.30	92.80	85.60	79.30
750	40.90	41.80	37.40	34.50	97.10	98.50	90.70	84.30
800	43.20	44.00	39.20	36.30	101.70	103.50	95.90	88.90
850	45.20	46.00	41.40	38.00	106.80	109.40	100.90	93.00
900	47.50	47.80	43.10	39.90	112.80	114.00	105.20	97.70
950	49.30	50.20	45.10	41.30	117.10	119.00	110.50	102.50
1000	51.30	52.40	47.10	43.10	122.00	124.70	115.00	106.80

SLFI2, SSLI2, and SDsaturI2 for p=0.20 and p=0.50, respectively. Figures 20 and 21 illustrate the number of wins generated by each of the four methods. The purpose of the graphs is to illustrate the relative effectiveness of the vertex-sequential with interchange methods as compared the SRLF algorithm. The I2 interchange method was selected because the data suggests that it is slightly superior to the I1 scheme.

Table XX contains a listing of the mean number of colors used by each of the three basic vertex-sequential methods, as well as SRLF, for p=0.20,0.50 and n=50,100,150,..,1000. Figures 22 and 23 depict this information graphically for p=0.20 and p=0.50, respectively.

This information supports several conclusions.

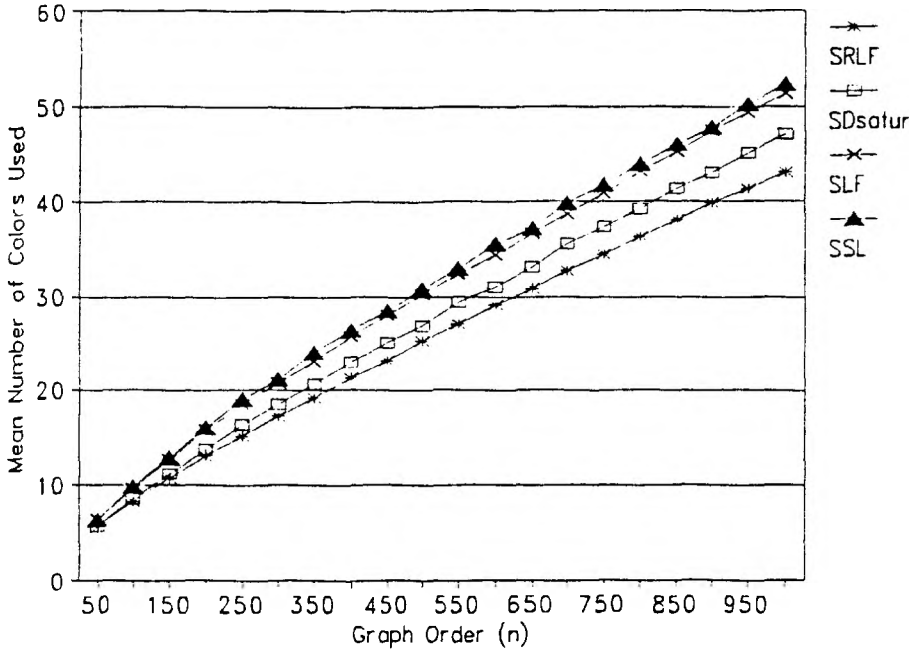


Figure 22. SGCP Heuristic Algorithms (p=0.20)

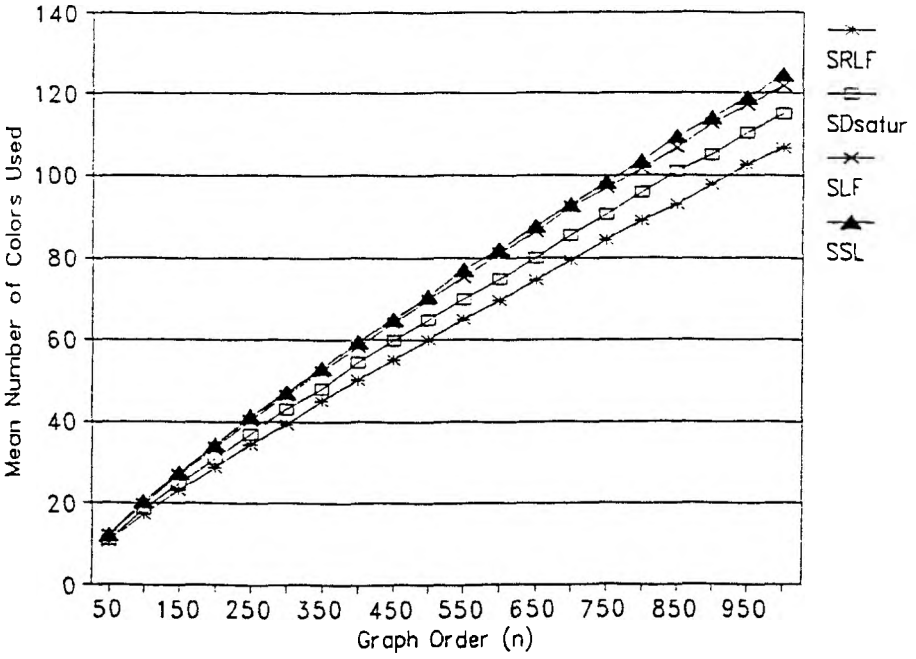


Figure 23. SGCP Heuristic Algorithms (p=0.50)

1. For both $p=0.20$ and $p=0.50$, there is little difference between the capabilities of the vertex-sequential with interchange methods and the color-sequential method when $n \leq 100$.

2. As n is increased above 100, the SRLF method becomes the dominant method with respect to all three measures of effectiveness. For example, when $p=0.20$ and $n=500$, SRLF used an average of 25.20 colors while the least corresponding value used by the interchange methods was 27.40. For $p=0.50$ and $n=500$, SRLF required only 60.08 colors on the average while the interchange algorithms used at least 63.80. The SRLF algorithm was credited with producing a minimum coloring on all 25 sample graphs for $p=0.20,0.50$ and $n=300,400,500$. Finally, the processing time for the interchange algorithms grows much more rapidly than for the SRLF method.

3. Clearly, there is very little difference in the vertex-sequential with interchange algorithms with respect to the measures of effectiveness. The graphs in Figures 18 and 19 illustrate this with the mean colors used curves being virtually indistinguishable. One interesting phenomena can be seen by comparing Figures 22 and 23 to Figures 18 and 19. As n grows, the SDsatur algorithm produces colorings that are increasingly superior to SLF and SSL. One would expect that SDsaturI1 and SDsaturI2 should be capable of performing significantly better than SLFI1, SSLI1, SLFI2, and SSLI2; but, clearly they do not.

4. For $n>500$, SRLF is obviously better that SLF, SSL, and SDsatur as far as mean number of colors used. However, the execution time statistics show that this superiority is not without cost. For example, when $p=0.50$ and $n=100$, SRLF required an average of 482.08 sec while SLF, SSL, and SDsatur used an average of 68.53 sec, 81.14 sec, and 54.83 sec, respectively.

It should be noted that most of the properties listed in the preceding conclusions do not represent new knowledge. Although the details would have been difficult to glean, the general content of the information could have been pieced together from the literature. But, recall that one goal of this study is to construct a basis for comparing properties of the CGCP with those of the SGCP. It was felt that

in order to establish these relationships, it would be necessary to identify the specific properties of the SGCP algorithms with respect to the standard versions of the problems that were solved with the CGCP methods.

4. CGCP Heuristic Algorithm Results.

Table XXI shows the results produced with the CGCP algorithms for p=0.20 and n=50,100,200,300,400,500. For each algorithm and graph order, it displays the observed mean values for the associated estimate of $\chi(CG_{n,p})$ and execution time, with minimum values for the estimate of the mean of $\chi(CG_{n,p})$ printed in bold. It also displays the number of wins credited to each procedure, with maximum values printed in bold. For example, for n=500, the CRLF algorithm used an average of 44.12 colors, required on the average 53.98 seconds of computing, and produced the minimum coloring for 21 of the 25 sample graphs. Table XXII contains the same categories of information for p=0.50. Figures 24 and 25 display graphs of the mean number of colors used by CRLF, CLFI1, CSLI1, and CDsaturI1 for p=0.20 and p=0.50, respectively. Figures 26 and 27 depict the number of wins generated by each of the four methods.

Table XXIII provides a listing of the mean number of colors used by each of the three basic vertex-sequential methods, as well as CRLF, for p=0.20,0.50 and n=50,100,150,..,1000. Figures 28 and 29 show this data graphically for p=0.20 and p=0.50, respectively.

This information supports the following conclusions.

1. For n=50 and p=0.20,0.50, CLFI1 and CDsaturI1 produced slightly superior results with respect to mean number of colors used and number of wins.

2. For n=100 and both edge densities, the CDsaturI1 method had the minimum average number of colors used statistic and was dominant in terms of number of wins.

3. As n was increased above 100, CDsaturI1 and CRLF became dominant with respect to mean number of colors used and number of wins. CRLF did somewhat better for p=0.20 and CDsaturI1 was only slightly superior for p=0.50.

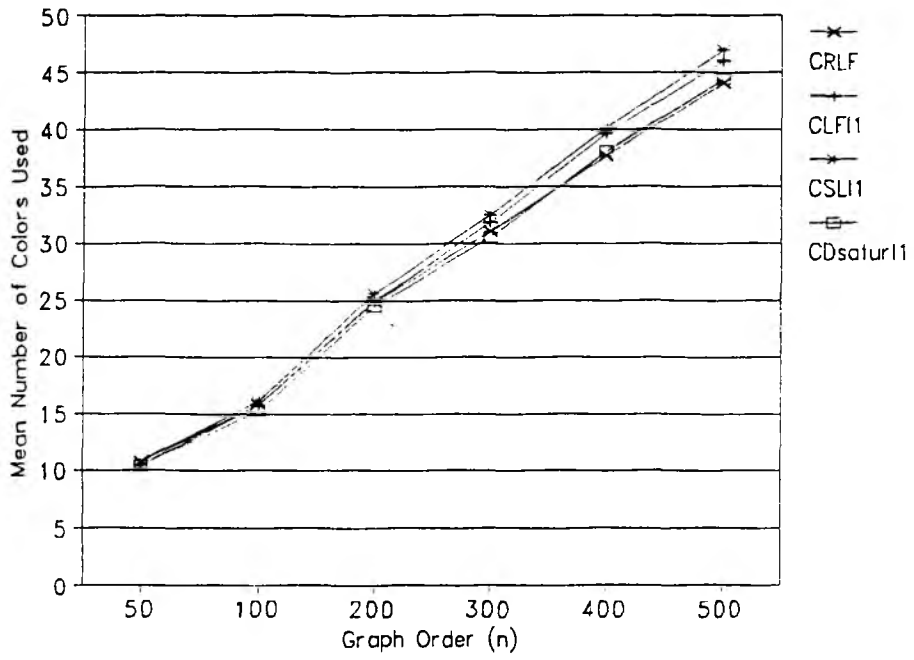


Figure 24. CGCP Heuristic Algorithms (p=0.20)

TABLE XXI

COMPOSITE GRAPH COLORING
HEURISTIC ALGORITHM RESULTS
p=0.20

n	CLF	CSL	CDsatur	CRLF	CLFI1	CSLI1	CDstrI1	CLFI2	CSLI2	CDstrI2
50	11.20	11.76	10.84	10.72	**10.40**	10.80	10.44	10.88	11.40	10.64
	0.10	0.13	0.19	0.16	0.60	0.76	0.77	0.29	0.35	0.36
	6	3	9	10	17	9	16	8	4	12
100	17.28	18.16	16.08	15.80	15.76	16.16	**15.24**	16.40	17.52	15.88
	0.37	0.49	0.67	0.82	3.22	3.88	3.91	1.67	1.88	2.01
	1	0	8	13	11	4	23	3	0	10
200	26.88	28.20	25.16	24.88	24.96	25.64	**24.44**	25.96	27.40	25.00
	1.38	1.90	2.47	4.64	24.30	30.48	32.41	14.80	15.16	17.44
	0	0	6	11	8	3	18	1	0	9
300	34.28	35.16	31.28	31.04	31.88	32.56	**30.52**	33.00	33.84	31.28
	2.89	4.02	5.30	13.86	77.17	103.29	117.77	57.27	62.60	83.69
	0	0	7	10	3	2	19	0	0	8
400	41.48	43.00	38.48	**37.68**	39.64	40.16	38.12	40.08	41.36	38.64
	4.99	7.07	9.31	29.44	204.54	253.90	341.51	160.12	157.80	246.72
	0	0	5	19	0	0	12	0	0	2
500	49.16	50.00	45.08	**44.12**	46.04	46.96	44.44	47.00	48.20	45.28
	7.64	10.86	14.42	53.98	426.67	532.55	762.45	361.73	330.02	560.11
	0	0	6	21	0	0	14	0	0	6

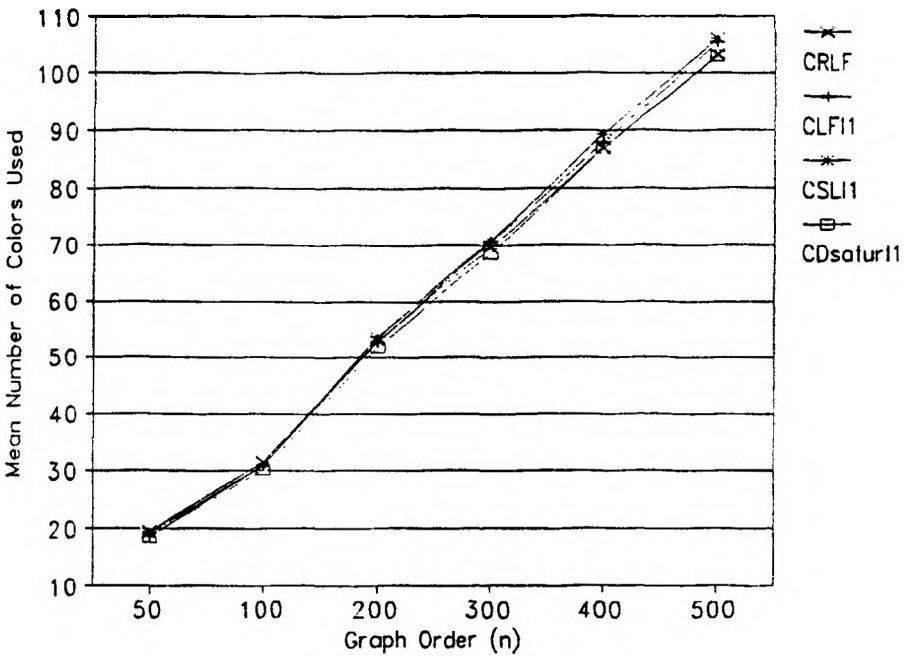


Figure 25. CGCP Heuristic Algorithms (p=0.50)

TABLE XXII

COMPOSITE GRAPH COLORING
HEURISTIC ALGORITHM RESULTS
p=0.50

n	CLF	CSL	CDsatur	CRLF	CLFI1	CSLI1	CDstrI1	CLFI2	CSLI2	CDstrI2
50	20.36	20.76	19.28	19.40	18.68	19.08	18.40	19.40	20.00	18.88
	0.19	0.23	0.30	0.28	3.54	4.05	3.89	3.27	3.52	3.33
	1	0	3	4	12	9	18	3	1	11
100	33.28	33.92	31.12	31.52	30.80	30.80	30.20	31.92	32.52	30.84
	0.70	0.85	1.05	1.66	27.74	27.74	31.70	30.47	31.73	28.74
	0	0	5	3	10	7	18	4	1	7
200	56.68	56.96	53.04	53.00	52.64	53.72	51.80	54.12	54.60	52.00
	2.75	3.31	3.82	10.92	268.68	358.36	430.35	326.68	316.07	315.45
	0	0	4	4	7	0	16	0	1	15
300	74.56	75.64	70.04	69.48	70.52	70.72	68.40	70.88	72.40	69.08
	5.87	7.14	8.29	33.46	1106.63	1321.18	1916.04	1267.99	1336.82	1396.48
	0	0	3	6	2	2	18	1	0	9
400	93.80	94.92	88.48	86.92	87.92	89.28	86.84	89.44	90.40	88.00
	10.46	12.69	14.86	80.87	2946.56	3713.02	5688.99	3121.40	3208.58	3681.95
	0	0	4	14	7	0	15	0	0	6
500	110.68	112.40	105.36	103.20	104.40	105.00	102.96	106.20	107.44	103.84
	14.87	18.37	21.49	148.26	6960.58	8617.09	12987.45	6783.49	7114.74	8350.14
	0	0	4	11	0	0	18	0	0	7

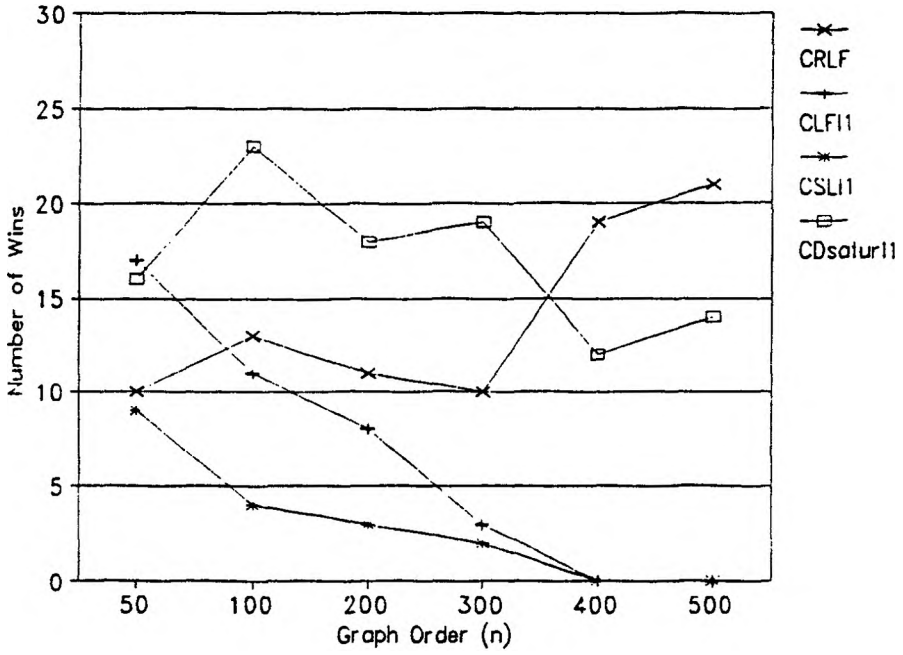


Figure 26. CGCP Heuristic Algorithms (p=0.20)

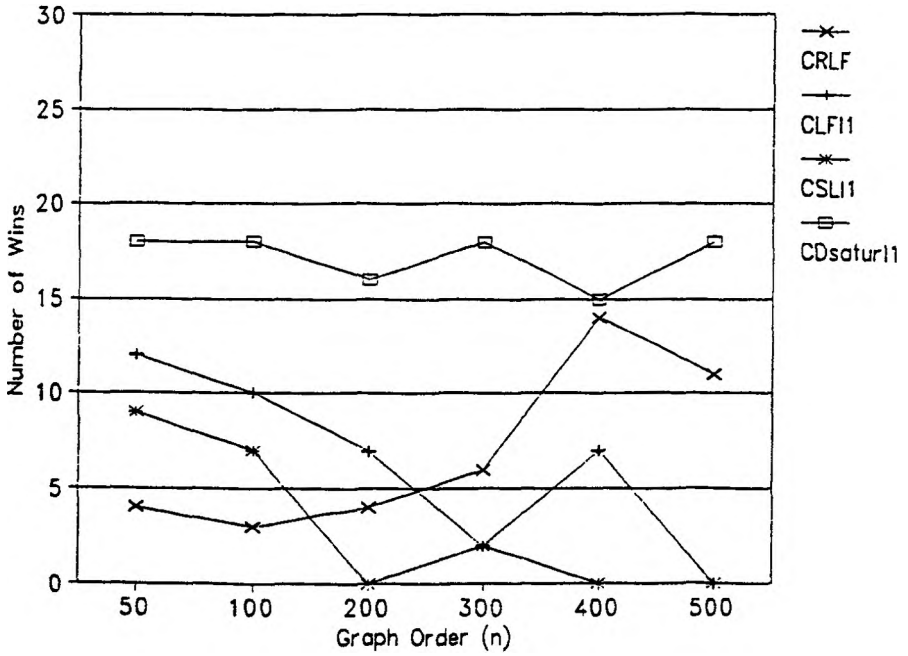


Figure 27. CGCP Heuristic Algorithms (p=0.50)

TABLE XXIII

COMPOSITE GRAPH COLORING
HEURISTIC ALGORITHM RESULTS

	p=0.20				p=0.50			
n	CLF	CSL	CDsatur	CRLF	CLF	CSL	CDsatur	CRLF
50	11.20	11.76	10.84	10.72	20.36	20.76	19.28	19.40
100	17.28	18.16	16.08	15.80	33.28	33.92	31.12	31.52
150	21.72	22.60	20.24	20.16	44.70	45.80	41.90	42.10
200	26.88	28.20	25.16	24.88	56.68	56.96	53.04	53.00
250	30.48	31.76	28.04	28.12	66.50	67.40	62.70	62.20
300	34.28	35.16	31.28	31.04	74.75	75.64	70.04	69.48
350	38.20	39.68	35.32	34.84	84.50	85.70	80.00	78.40
400	41.48	43.00	38.48	37.68	93.80	94.92	88.48	86.92
450	45.12	46.60	41.84	41.00	102.90	104.90	98.40	96.50
500	49.16	50.08	45.08	44.12	110.68	112.40	105.36	103.20
550	52.90	54.70	49.00	48.30	121.20	122.70	114.10	112.80
600	56.00	57.50	51.30	50.70	128.80	129.70	121.40	120.00
650	60.50	61.40	55.40	53.90	138.00	139.60	131.00	129.30
700	62.50	64.00	58.30	56.70	145.70	147.10	138.40	136.20
750	66.30	67.90	61.90	60.30	155.00	156.80	148.70	145.40
800	70.40	71.80	65.40	63.70	163.90	167.20	156.50	152.60
850	72.80	74.20	67.70	65.90	169.80	172.20	162.20	159.00
900	76.30	77.00	70.30	68.80	178.20	181.80	171.00	167.10
950	79.10	80.90	74.20	71.10	187.20	190.50	178.40	174.50
1000	82.50	84.00	76.20	74.50	195.70	197.10	186.20	182.50

This and the previous two properties are compatible with the results reported by Roberts. Recall that the vertex-sequential with interchange methods did better for small graphs; but, as the graph order was increased, at some point the color-sequential methods became preferable. However, that study did not test the variations of CDsatur.

4. The I2 interchange scheme appears to be only marginally effective.

5. The obvious shortcoming of the interchange methods in general is the significant rise in computing expense associated with increasing values of n. For example, when n=500 and p=0.50, the average processing time required by CDsaturI1 for each of the sample graphs was 3.61 hours. On the other hand, CRLF needed only 148.26 seconds.

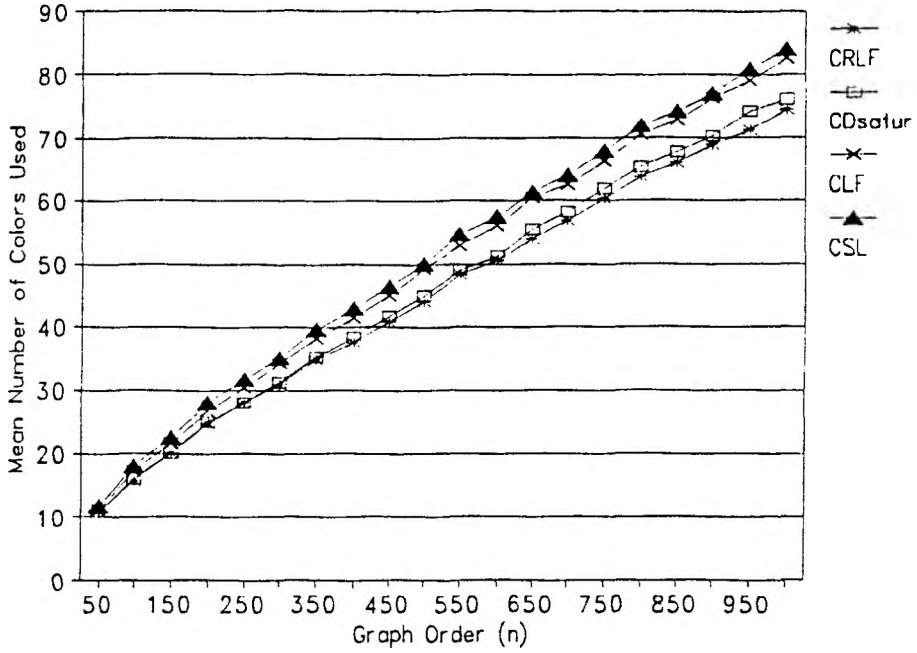


Figure 28. CGCP Heuristic Algorithms (p=0.20)

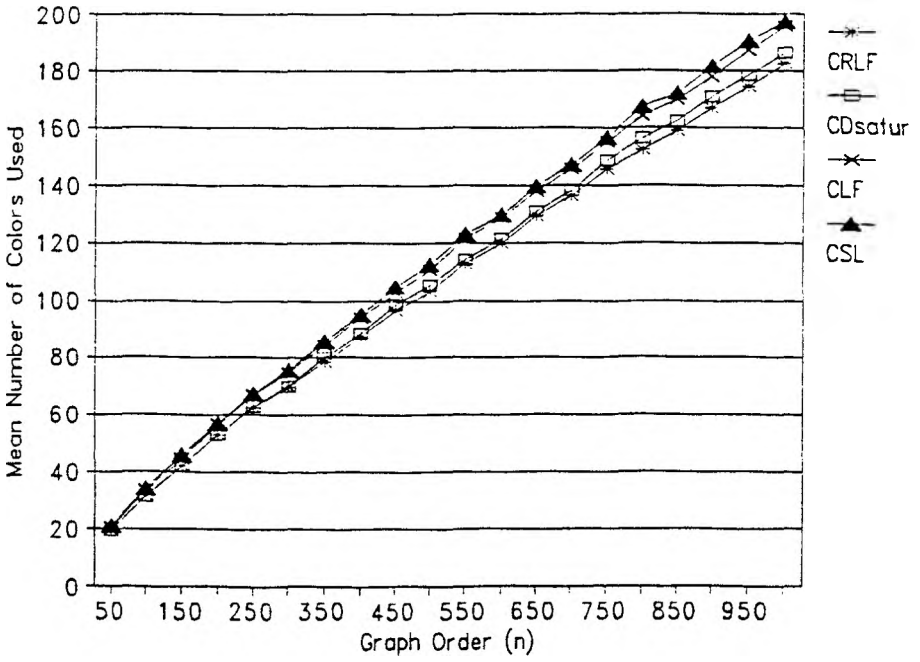


Figure 29. CGCP Heuristic Algorithms (p=0.50)

6. Table XXIII and Figures 28 and 29 show that CDsatur is marginally competitive with CRLF for large graphs and significantly better than CLF and CSL. The major advantage of CDsatur is its speed. For example, when $n=1000$ and $p=0.50$, CDsatur used an average of 89.40 seconds to compute each graph while CRLF required 1155.84 seconds.

Before terminating this chapter, a few comments, with respect to a comparison of the results for the SGCP heuristics with the CGCP methods, seem in order. For random standard graphs of medium to large order, the color-sequential approach is apparently very superior to the vertex-sequential methods. But, for composite graphs, the picture is not nearly as clear. Likely, part of the reason is the fact that a primary criteria used for determining the order in which the vertices are colored is the same for all methods, namely, vertex chromaticity. This seems to mask the differences in the algorithms. Thus, it requires larger problems in order for the superiority of the RLF approach to show itself.

V. PROBABILISTIC BOUNDS

Since graph coloring is an NP-complete problem, the computation time required by exact algorithms will almost certainly be unacceptable for all but relatively small problem instances. The heuristic algorithms presented in Chapter IV are designed to be time efficient but, very likely, at the expense of not being able to effectively approximate optimal solutions. These algorithms were compared among themselves; however, a much more useful capability would be the ability to measure the actual effectiveness of a given method.

Some initial results with respect to this goal can be drawn from Tables XXIV and XXV. Let $\overline{\chi}(SG_{n,p})$ and $\overline{\chi}(CG_{n,p})$ denote, respectively, the mean values for $\chi(SG_{n,p})$ and $\chi(CG_{n,p})$ of a sample of standard and composite graphs. Table XXIV contains the values for $\overline{\chi}(SG_{n,p})$ and $\overline{\chi}(CG_{n,p})$ that were generated by the exact Vertex-Sequential with Dynamic Vertex Reordering Algorithms of Chapter III for random samples of size 25 of standard and type 1 composite graphs with $p=0.25$ and $n=10,12,..,78$ and $n=10,12,..44$, respectively. The column headed $\overline{\chi}(CG_{n,p})$ also includes, in italics, projected estimates for $n=46,48,..,78$. These values were calculated by multiplying the corresponding values of $\overline{\chi}(SG_{n,p})$ by 1.836 which was the middle of the 99% confidence interval for $\overline{\chi}(CG_{n,p})/\overline{\chi}(SG_{n,p})$ described in Section J of Chapter III. The column headed Min contains estimates for $\overline{\chi}(SG_{n,p})$ and $\overline{\chi}(CG_{n,p})$ that are guaranteed to be optimal with respect to the 10 heuristic algorithms tested in Chapter IV. Specifically, for a given graph, the Min heuristic method uses the best coloring produced by the other ten methods. Table XXV contains similar information for $p=0.50$.

Both tables suggest that the Min heuristic is very accurate for small graphs. However, it becomes less capable as the size of the problem increases. Unfortunately, this limited amount of data is not sufficient for making an evaluation of how effective, or ineffective, Min or any other heuristic approach might be for graphs with orders that range up to 1,000 vertices or more.

TABLE XXIV

GRAPH COLORING
HEURISTIC EVALUATION FOR VERY SMALL GRAPHS
p=0.25

n	$\overline{\chi}(SG_{n,p})$	Min	Min-$\overline{\chi}(SG_{n,p})$	$\overline{\chi}(CG_{n,p})$	Min	Min-$\overline{\chi}(CG_{n,p})$
10	2.88	2.88	0.00	5.12	5.12	0.00
12	3.04	3.04	0.00	5.12	5.12	0.00
14	3.24	3.24	0.00	6.12	6.12	0.00
16	3.48	3.48	0.00	6.60	6.60	0.00
18	3.56	3.56	0.00	6.80	6.80	0.00
20	3.84	3.84	0.00	6.92	7.00	0.08
22	4.04	4.04	0.00	7.12	7.20	0.08
24	4.04	4.04	0.00	7.60	7.88	0.28
26	4.08	4.12	0.04	7.92	8.04	0.12
28	4.24	4.32	0.08	7.80	8.04	0.24
30	4.56	4.60	0.04	8.20	8.64	0.44
32	4.68	4.76	0.08	8.64	9.16	0.52
34	4.92	4.92	0.00	8.96	9.32	0.36
36	5.00	5.00	0.00	8.76	9.08	0.32
38	5.00	5.08	0.08	8.80	9.28	0.48
40	5.04	5.20	0.16	9.28	9.64	0.36
42	5.20	5.40	0.20	9.64	10.36	0.72
44	5.28	5.64	0.36	9.68	10.68	1.00
46	5.44	5.92	0.48	*9.99*	10.80	0.81
48	5.76	5.88	0.12	*10.58*	11.12	0.54
50	6.00	6.04	0.04	*11.02*	11.32	0.30
52	5.96	6.04	0.08	*10.94*	11.68	0.74
54	6.00	6.32	0.32	*11.02*	11.44	0.42
56	6.04	6.40	0.36	*11.09*	12.08	0.99
58	6.08	6.64	0.56	*11.17*	11.96	0.79
60	6.12	6.92	0.80	*11.24*	12.32	1.08
62	6.32	6.96	0.64	*11.61*	12.80	1.19
64	6.60	7.04	0.44	*12.12*	13.36	1.24
66	6.84	7.04	0.20	*12.56*	13.56	1.00
68	7.00	7.20	0.20	*12.85*	13.80	0.95
70	7.00	7.56	0.56	*12.85*	13.72	0.87
72	7.00	7.72	0.72	*12.85*	13.96	1.11
74	7.00	7.84	0.84	*12.85*	14.04	1.19
76	7.08	8.00	0.92	*13.00*	14.24	1.24
78	7.12	8.00	0.88	*13.08*	14.84	1.76

TABLE XXV

GRAPH COLORING
HEURISTIC EVALUATION FOR VERY SMALL GRAPHS
p=0.50

n	$\overline{\chi}(SG_{n,p})$	Min	Min-$\overline{\chi}(SG_{n,p})$	$\overline{\chi}(CG_{n,p})$	Min	Min-$\overline{\chi}(CG_{n,p})$
10	3.96	3.96	0.00	7.04	7.08	0.04
12	4.36	4.36	0.00	7.44	7.48	0.04
14	4.68	4.68	0.00	8.20	8.24	0.04
16	5.00	5.00	0.00	9.40	9.52	0.12
18	5.40	5.40	0.00	9.48	9.52	0.04
20	5.60	5.60	0.00	9.92	10.28	0.36
22	6.04	6.04	0.00	10.48	10.56	0.08
24	6.40	6.48	0.08	11.32	11.64	0.32
26	6.44	6.56	0.12	11.72	11.96	0.24
28	6.88	7.08	0.20	12.08	12.48	0.40
30	6.96	7.28	0.32	12.32	13.36	1.04
32	7.24	7.44	0.20	13.04	13.64	0.60
34	7.40	7.84	0.44	*13.17*	14.48	1.31
36	7.88	8.04	0.16	*14.03*	14.12	0.09
38	8.04	8.48	0.44	*14.31*	15.04	0.73
40	8.08	8.68	0.60	*14.38*	15.56	1.18
42	8.72	9.08	0.36	*15.52*	16.36	0.84
44	8.92	9.24	0.32	*15.88*	17.08	1.20
46	9.04	9.68	0.64	*16.09*	17.52	1.43
48	9.28	9.80	0.52	*16.52*	17.36	0.84
50	9.56	10.04	0.48	*17.02*	18.04	1.02
52	9.88	10.40	0.52	*17.59*	18.68	1.09
54	9.92	10.76	0.84	*17.66*	18.98	1.32
56	10.04	10.92	0.88	*17.87*	19.44	1.57
58	10.20	11.36	1.16	*18.16*	19.72	1.56

Since exact algorithms can not be used to produce comparison data for graphs of large order, some alternative approach must be found. The purpose of this chapter is to describe methods, which are based on the application of probability theory, for calculating both lower and upper bounds on $\overline{\chi}(SG_{n,p})$ and $\overline{\chi}(CG_{n,p})$. It will be shown that the data base which was produced with the exact algorithms supports the validity of this theory. The effectiveness of the heuristic algorithms presented in Chapter IV are then analyzed.

A. PROBABILISTIC LOWER BOUND FOR THE SGCP

Let K denote the random variable which is the number of distinguishable k-colorings of a given graph. Bollobas and Thomason [*Bo85,BT85*] state the following result without proof.

Theorem 4. Given an instance of a random standard graph $SG_{n,p}=(V,E)$, the expected number of different k-colorings of $SG_{n,p}$ is given by

$$E(K) = \sum_{(a_i)} \frac{n!}{\prod_{i=1}^{k} a_i! \prod_{j=1}^{n} n_j!} q^{\sum_{i=1}^{k} \binom{a_i}{2}} \tag{37}$$

where $n=|V|$, $q=1-p$, the sum is over all combinations (a_i) of length k whose sum is n, and n_j is the number of a_i which equal j.

Proof. Let $P_k=\{S_1,S_2,..,S_k\}$ represent an arbitrary partitioning of V into sets of size $a_1,a_2,..,a_k$, respectively, where $a_1+a_2+..+a_k=n$. Let $Pr\{P_k\}$ denote the probability that the function f, defined by $f(v_j)=i$ iff $v_j \in S_i$, is a k-coloring of $SG_{n,p}$. In order for f to be feasible, it must be the case that no pair of vertices in S_i are connected, $i=1,2,..k$. Since $|S_i|=a_i$, it follows that there are

$$\binom{a_i}{2} = \frac{a_i(a_i-1)}{2} \tag{38}$$

possible edges involving vertices in S_i. Now, $q=1-p$ represents the probability that a random pair of vertices are not connected. Thus,

$$Pr\{P_k\} = q^{\binom{a_1}{2}} q^{\binom{a_2}{2}} \cdots q^{\binom{a_k}{2}} = q^{\sum_{i=1}^{k} \binom{a_i}{2}} \tag{39}$$

Fundamental principles of counting [*Me65*] state that the number of distinguishable ways of partitioning n distinguishable objects into subsets of size $a_1,a_2,..,a_k$ is given by

$$\frac{n!}{\prod_{i=1}^{k} a_i! \prod_{j=1}^{n} n_j!} \tag{40}$$

where n_j is the number of a_i which equal j. Thus, the expected number of acceptable partitions whose member sets are of size $a_1,a_2,..,a_k$ is given by

$$\frac{n!}{\prod_{i=1}^{k} a_i! \prod_{j=1}^{n} n_j!} q^{\sum_{i=1}^{k} \binom{a_i}{2}} \tag{41}$$

Since permutations of the sets in P_k would produce indistinguishable colorings, we can find E(K) by considering all possible combinations, (a_i), such that $a_1 + a_2 + .. + a_k = n$. Thus,

$$E(K) = \sum_{(a_i)} \frac{n!}{\prod_{i=1}^{k} a_i! \prod_{j=1}^{n} n_j!} q^{\sum_{i=1}^{k} \binom{a_i}{2}} \tag{42}$$

Consider the significance of Theorem 4. For specified values of n and p, let $k_{n,p}$ denote the largest value of k such that $E(K) < 1$. With large probability, it can be concluded that it will require *at least* $k_{n,p} + 1$ colors to color any random standard graph of order n and edge density p. Thus, $k_{n,p} + 1$ is a probabilistic *lower bound* for $\chi(SG_{n,p})$.

In order to calculate E(K), for specified values of n, p, and k, an algorithm was designed for generating all possible sequences, $a_1,a_2,..,a_k$, where $a_1 + a_2 + .. + a_k = n$ and $a_1 \geq a_2 \geq .. \geq a_k \geq 1$. The following theory forms a basis for this algorithm.

Lemma 1. If $a_1,a_2,..,a_k,n$ are natural numbers, $a_1 + a_2 + .. + a_k = n$, and $a_1 \geq a_2 \geq .. \geq a_k \geq 1$ then, for any given value of i, $1 \leq i \leq k$, the maximum value of a_i, denoted $MAX(a_i)$, is given by

$$MAX(a_i) = \min\{(n - \sum_{j=1}^{i-1} a_j) - (k-i),\ a_{i-1}\} \tag{43}$$

Proof. Easily, $a_i \le a_{i-1}$. Consider an arbitrary value of i such that $1 \le i \le k$. Assume that $a_1, a_2, .., a_{i-1}$ have been specified, where $a_1 \ge a_2 \ge .. \ge a_{i-1} \ge 1$ and $a_1 + a_2 + .. + a_{i-1} < n$. Easily,

$$a_i = n - \sum_{j=1}^{i-1} a_j - \sum_{j=i+1}^{n} a_j \tag{44}$$

But, the smallest that a_j could be for $j = (i+1), (i+2), .., k$ is 1. Thus, it must be that

$$a_i \le (n - \sum_{j=1}^{i-1} a_j) - \sum_{j=i+1}^{k} 1 = (n - \sum_{j=1}^{i-1} a_j) - (k-i) \tag{45}$$

Lemma 2. If $a_1, a_2, .., a_k, n$ are natural numbers, $a_1 + a_2 + .. + a_k = n$, and $a_1 \ge a_2 \ge .. \ge a_k \ge 1$ then, for any given value of i, $1 \le i \le k$, the minimum value of a_i, denoted $MIN(a_i)$, is given by

$$MIN(a_i) = \left\lceil \frac{n - \sum_{j=1}^{i-1} a_j}{k-(i-1)} \right\rceil \tag{46}$$

Proof. Consider an arbitrary value of i such that $1 \le i \le k$. Assume that $a_1, a_2, .., a_{i-1}$ have been specified, where $a_1 \ge a_2 \ge .. \ge a_{i-1} \ge 1$ and $a_1 + a_2 + .. + a_{i-1} < n$. Since $a_i \ge a_{i+1} \ge .. \ge a_k \ge 1$, it follows that

$$n - \sum_{j=1}^{i-1} a_j - \sum_{j=i}^{k} a_j \le \sum_{j=i}^{k} a_i = [k-(i-1)]a_i \quad or$$
$$a_i \ge \frac{n - \sum_{j=1}^{i-1} a_j}{k-(i-1)} \tag{47}$$

```
procedure GenerateSequences (I,SumA)
 if (I=K) then
   A[K] ← N - SumA
 else
   calculate MinA,MaxA
   for A[I] ← MinA to MaxA do
     GenerateSequences (I+1,SumA+A[I])
```

Figure 30. Algorithm *GenerateSequences*

Since a_i is a natural number, the conclusion of the lemma follows.

Together, Lemma 1 and Lemma 2 justify the algorithm *GenerateSequences*, which is displayed in Figure 30 and is designed to generate all possible sequences (a_i), such that $a_1+a_2+..+a_k=n$ and $a_1 \geq a_2 \geq .. \geq a_k \geq 1$.

Example 23

Suppose n=10 and k=4. Using GenerateSequences, the 9 sequences listed in Table XXVI were generated.

GenerateSequences can easily be extended to produce a procedure for calculating E[K] for any given combination of values for n, k, and p. The result appears in Figure 31 as *CalcExactEOfK*. The implementation of this procedure is presented in the Appendix as *Listing 14. S_LB1.Pas.* For any given pair of values for n and p, the program was designed to identify $k_{n,p}$ and, thus, the probabilisitic lower bound for $\chi(SG_{n,p})$, $k_{n,p}+1$.

Results of executing this program for p=0.5, n=20,22,..,80, are provided in Table XXVII. Note that for n=80, if k=11 then $E(K)<10^{-3}$ and if k=12 then $E(K)>10^{5}$. Hence, $k_{80,0.5}=11$ and there is a very high probability that $\chi(80) \geq 12$. Similarly, the table can be used to state a probabilistic lower bound for the other values of n.

Evidence of the validity of this approach is also provided by the exact Vertex-

TABLE XXVI

STANDARD GRAPH COLORING
POSSIBLE COLOR CLASS SIZES
n=10 k=4

a_1	a_2	a_3	a_4
3	3	2	2
3	3	3	1
4	2	2	2
4	3	2	1
4	4	1	1
5	2	2	1
5	3	1	1
6	2	1	1
7	1	1	1

Sequential with Dynamic Vertex Reordering algorithm presented in Chapter III. In the columns headed $\chi_{min}(SG_{n,p})$, Table XXVIII contains the minimum observed values of the chromatic number for 25 instances of the SGCP at each specified value of n and p. In the columns headed $k_{n,p}+1$, probabilistic lower bounds, which were produced with S_LB1.Pas, are displayed. In all cases, the probabilistic lower bounds are no larger then the observed minimum values.

The obvious problem with this method is illustrated in the two *Sequences* columns of Table XXVII. These columns contain counts for the number of different sequences, $a_1,a_2,..,a_k$, which were processed. Obviously, this number grows exponentially with increasing values of n. A method for dealing with this problem was developed by observing that not all sequences contribute equally to the expectation. In fact, relatively few sequences account for the vast majority of the expected value. For example, consider Table XXIX which shows the first 40 of the 64 different sequences for n=20, p=0.5, and $k_{n,p}=4$. The most likely sequence is $a_1=6$, $a_2=5$, $a_3=5$, and $a_4=4$ which accounts for over 52% of E(K). The first 8 sequences account for 98.9% for E(K) and the first 27 sequences account for 99.999% of E(K). Based on these observations, it was hypothesized that an efficient implicit enumeration algorithm could be developed for closely approximating E(K).

```
procedure CalcExactEOfK (Level, SumA)
  if (Level=K) then
    A[K] ← N - SumA
    save NJ[A[K]], ExpTerm
    update NJ[A[K]], ExpTerm
    update ExpNumOfKColorings, NumOfSequences
    restore NJ[A[K]], ExpTerm
  else
    calculate MinA ,MaxA
    for A[Level] ← MinA to MaxA
      save NJ[A[Level]], ExpTerm
      update NJ[A[Level]], ExpTerm
      CalcExactEOfK (Level+1,SumA+A[Level])
      restore NJ[A[Level]], ExpTerm
```

Figure 31. Algorithm *CalcExactEOfK*

The resulting algorithm was named *CalcEstimatedEOfK* and is listed in Figure 32.

In both CalcExactEOfK and CalcEstimatedEOfK, the value of the variable named *ExpTerm* represents the contribution of the sequence currently being processed. ExpTerm is initialized to n! and then adjusted as the a_i, i=1,2,..,k, are generated. Specifically, as each new term, a_i, of the sequence is produced, the value of the variable is adjusted by dividing it by $a_i!$ and multiplying by q raised to the $a_i(a_i-1)/2$ power. When this adjustment has been applied to *ExpTerm* for a_k, a final adjustment for the $n_j!$ denominators must be made.

The difference in the two algorithms is that CalcExactEOfK processes every sequence, while the approach taken by CalcEstimatedEOfK is to identify and abandon as soon as possible those sequences whose contribution to E(K) would be considered insignificant. This determination is made by estimating the final value of ExpTerm as each a_i is processed. The approximation is named *ExpTermApprox* and is calculated by assigning $a_{i+1},a_{i+2},..,a_k$ the minimum value they were assigned on the previous forward pass. As soon as ExpTermApprox < SF*ExpNumOfKColorings, where SF denotes a *selection factor* and ExpNumOfKColorings denotes the currently accumulated value for E(K), backtracking is invoked. In order to keep thrashing from occurring, the array MaxClassSize is used to disallow more than one unsuccessful forward pass from a level beyond the last successful one. Hence, the

TABLE XXVII

STANDARD GRAPH COLORING
EXACT EXPECTED NUMBER OF K COLORINGS
$k_{n,p} = \max\{k \mid E(K) < 1\}$
$p = 0.50$

n	$k_{n,p}$	E(K)	Sequences	$k_{n,p}+1$	E(K)	Sequences
20	4	4.2630762E-03	64	5	4.9546188E+01	84
22	4	8.4354971E-05	84	5	6.3764818E+00	119
24	5	4.7571208E-01	164	6	3.2703712E+03	199
26	5	2.0549958E-02	221	6	6.3042827E+02	282
28	5	5.1353005E-04	291	6	7.7187730E+01	391
30	5	7.4175744E-06	377	6	5.9975545E+00	532
32	6	2.9552304E-01	709	7	1.7849023E+04	860
34	6	9.2282794E-03	931	7	2.2120906E+03	1175
36	6	1.8252479E-04	1206	7	1.8562259E+02	1579
38	6	2.2855249E-06	1540	7	1.0540751E+01	2093
40	7	4.0487892E-01	2738	8	1.3525384E+05	3319
42	7	1.0515143E-02	3539	8	1.2462259E+04	4417
44	7	1.8458027E-04	4526	8	8.1588994E+02	5812
46	7	2.1892559E-06	5731	8	3.7940558E+01	7564
48	7	1.7539806E-08	7190	8	1.2527900E+00	9749
50	8	2.9365266E-02	12450	9	1.2395336E+05	15224
52	8	4.8849621E-04	15765	9	6.8770557E+03	19720
54	8	5.7658176E-06	19805	9	2.8143957E+02	25331
56	8	4.8277293E-08	24699	9	8.4939400E+00	32278
58	9	1.8901064E-01	40831	10	2.2338633E+06	49037
60	9	3.1005261E-03	51294	10	1.1098853E+05	62740
62	9	3.7487140E-05	64015	10	4.1926146E+03	79725
64	9	3.3400934E-07	79403	10	1.2039380E+02	100654
66	9	2.1928117E-09	97922	10	2.6276554E+00	126299
68	10	4.3582837E-02	157564	11	3.3548661E+06	188556
70	10	5.4927229E-04	195491	11	1.2086631E+05	237489
72	10	5.2593407E-06	241279	11	3.3934043E+03	297495
74	10	3.8255728E-08	296320	11	7.4235778E+01	370733
76	10	2.1136676E-10	362198	11	1.2652864E+00	459718
78	11	1.6800272E-02	567377	12	6.4809089E+06	674585
80	11	1.7376101E-04	697097	12	1.8129431E+05	839286

TABLE XXVIII

STANDARD GRAPH COLORING
EXACT VERSUS PROBABILISTIC MINIMUMS

n	p=0.25		p=0.50	
	$\chi_{min}(SG_{n,p})$	$k_{n,p}+1$	$\chi_{min}(SG_{n,p})$	$k_{n,p}+1$
20	3	3	5	5
22	4	3	5	5
24	4	4	6	6
26	4	4	6	6
28	4	4	6	6
30	4	4	6	6
32	4	4	6	6
34	4	4	7	7
36	4	4	7	7
38	5	4	7	7
40	5	5	8	8
42	5	5	8	8
44	5	5	8	8
46	5	5	9	8
48	5	5	9	8
50	6	5	9	9
52	5	5	9	9
54	6	5	9	9
56	6	5	9	9
58	6	6	10	10
60	6	6		
62	6	6		
64	6	6		
66	6	6		
68	7	6		
70	7	6		
72	7	6		
74	7	6		
76	7	7		
78	7	7		

selection factor SF, in conjunction with the array MaxClassSize, can be used to control the granularity of the sampling.

The implementation of CalcEstimatedEOfK is presented in the Appendix as *Listing 15. S_LB2.Pas.* Results of executing this program for p=0.50 and n=20,22,..,80, with SF=e^{-23}, are provided in Table XXX. Table XXXI and Table

TABLE XXIX

STANDARD GRAPH COLORING
CONTRIBUTION TO E(K) BY SEQUENCE
n = 20 p = 0.50 $k_{n,p} = 4$

Seq Num	a_1	a_2	a_3	a_4	Sequence Exp Value	Cumulative Exp Value	Percent of E(K)
1	5	5	5	5	4.4461967E-04	4.4461967E-04	10.4296%
2	6	5	5	4	2.2230981E-03	2.6677178E-03	52.1477%
3	6	6	4	4	4.6314541E-04	3.1308632E-03	10.8641%
4	6	6	5	3	3.7051632E-04	3.5013795E-03	8.6913%
5	6	6	6	2	7.7190898E-06	3.5090986E-03	0.1811%
6	7	5	4	4	3.9698178E-04	3.9060804E-03	9.3121%
7	7	5	5	3	1.5879271E-04	4.0648731E-03	3.7248%
8	7	6	4	3	1.3232725E-04	4.1972004E-03	3.1040%
9	7	6	5	2	1.9849088E-05	4.2170494E-03	0.4656%
10	7	6	6	1	2.0676133E-07	4.2172562E-03	0.0049%
11	7	7	3	3	2.3629863E-06	4.2196192E-03	0.0554%
12	7	7	4	2	1.7722399E-06	4.2213914E-03	0.0416%
13	7	7	5	1	8.8611996E-08	4.2214800E-03	0.0021%
14	8	4	4	4	1.0338067E-05	4.2318181E-03	0.2425%
15	8	5	4	3	2.4811361E-05	4.2566295E-03	0.5820%
16	8	5	5	2	1.8608521E-06	4.2584903E-03	0.0437%
17	8	6	3	3	2.0676132E-06	4.2605579E-03	0.0485%
18	8	6	4	2	1.5507101E-06	4.2621087E-03	0.0364%
19	8	6	5	1	7.7535506E-08	4.2621862E-03	0.0018%
20	8	7	3	2	1.1076499E-07	4.2622970E-03	0.0026%
21	8	7	4	1	1.3845625E-08	4.2623108E-03	0.0003%
22	8	8	2	2	3.2450682E-10	4.2623111E-03	0.0000%
23	8	8	3	1	2.1633788E-10	4.2623113E-03	0.0000%
24	9	4	4	3	4.3075278E-07	4.2627421E-03	0.0101%
25	9	5	3	3	1.7230110E-07	4.2629144E-03	0.0040%
26	9	5	4	2	1.2922584E-07	4.2630436E-03	0.0030%
27	9	5	5	1	3.2306460E-09	4.2630468E-03	0.0001%
28	9	6	3	2	2.1537638E-08	4.2630684E-03	0.0005%
29	9	6	4	1	2.6922050E-09	4.2630711E-03	0.0001%
30	9	7	2	2	2.8845049E-10	4.2630714E-03	0.0000%
31	9	7	3	1	1.9230033E-10	4.2630716E-03	0.0000%
32	9	8	2	1	2.2535196E-12	4.2630716E-03	0.0000%
33	9	9	1	1	9.7809008E-16	4.2630716E-03	0.0000%
34	10	4	3	3	2.6922047E-09	4.2630743E-03	0.0001%
35	10	4	4	2	1.0095768E-09	4.2630753E-03	0.0000%
36	10	5	3	2	8.0766144E-10	4.2630761E-03	0.0000%
37	10	5	4	1	1.0095769E-10	4.2630762E-03	0.0000%
38	10	6	3	1	1.6826280E-11	4.4630762E-03	0.0000%
39	10	7	2	1	4.5070391E-13	4.2630762E-03	0.0000%
40	10	8	1	1	8.8028110E-13	4.2630762E-03	0.0000%

```
procedure CalcEstimatedEOfK (Level, SumA)
  if (Level=K) then
    A[K] ← N - SumA
    save NJ[A[K]], ExpTerm
    update NJArray[A[K]], ExpTerm
    if ExpTerm>SF*ExpNumOfKColorings then
      update ExpNumOfKColorings, NumOfSequences, MaxClassSizeArray
      Backtrack ← false
    else
      Backtrack ← true
    restore NJ[A[K]], ExpTerm
  else
    calculate MinA, MaxA
    A[Level] ← MinA
    while A[Level]≤MaxA and not Backtrack
      save NJ[A[Level]], ExpTerm
      update NJ[A[Level]], ExpTerm
      calculate ExpTermApprox
      if ExpTermApprox>SF*ExpNumOfKColorings then
        CalcEstimatedEOfK (Level+1, SumA+A[Level])
      else
        Backtrack ← true
      restore NJ[A[Level]], ExpTerm
      if A[Level]=MaxClassSizeArray[Level] and Backtrack then
        Backtrack ← false
        MaxClassSizeArray[Level] ← 0
      incr A[Level]
    A[Level] ← MinA
```

Figure 32. Algorithm *CalcEstimatedEOfK*

XXXII present a comparison of the results of using CalcExactEOfK versus CalcEstimatedEOfK. In all cases, the implicit enumeration approximation is accurate to at least 6 significant digits. Also, while the total number of possible sequences increases exponentially with n, the number actually processed by CalcEstimatedEOfK is growing much more slowly.

CalcEstimatedEOfK is a sampling approach whose sample size is controlled by the value of the selection factor SF. Bollobas and Thomason [BT85] also designed an algorithm for estimating E(K). Although this method was unavailable to this author, it was provided to David Johnson at AT&T Bell Labs. Through a private communication, he agreed to make available results of executing the algorithm for p=0.20,0.50 and n=100,150,..,1000. Tables XXXIII and XXXIV

TABLE XXX

STANDARD GRAPH COLORING
ESTIMATED EXPECTED NUMBER OF K COLORINGS
$k_{n,p} = \max\{k \mid E(K) < 1\}$
$p = 0.50 \quad SF = e^{-23}$

n	$k_{n,p}$	E(K)	Sequences	$k_{n,p}+1$	E(K)	Sequences
20	4	4.2630762E-03	38	5	4.9546188E+01	54
22	4	8.4354971E-05	40	5	6.3764818E+00	70
24	5	4.7571209E-01	76	6	3.2703712E+03	98
26	5	2.0549958E-02	84	6	6.3042827E+02	129
28	5	5.1353005E-04	94	6	7.7187731E+01	146
30	5	7.4175743E-06	94	6	5.9975544E+00	146
32	6	2.9552298E-01	161	7	1.7849024E+04	232
34	6	9.2282793E-03	180	7	2.2120905E+03	247
36	6	1.8252479E-04	176	7	1.8562259E+02	275
38	6	2.2855249E-06	183	7	1.0540751E+01	293
40	7	4.0487891E-01	300	8	1.3525384E+05	412
42	7	1.0515143E-02	300	8	1.2462259E+04	438
44	7	1.8458026E-04	308	8	8.1588991E+02	455
46	7	2.1892558E-06	312	8	3.7940555E+01	461
48	7	1.7539804E-08	296	8	1.2527899E+00	474
50	8	2.9365263E-02	485	9	1.2395335E+05	658
52	8	4.8849617E-04	483	9	6.8770547E+03	679
54	8	5.7658168E-06	485	9	2.8143951E+02	686
56	8	4.8277283E-08	469	9	8.4939381E+00	681
58	9	1.8901059E-01	692	10	2.2338627E+06	945
60	9	3.1005254E-03	703	10	1.1098846E+05	956
62	9	3.7487125E-05	694	10	4.1926133E+03	982
64	9	3.3400919E-07	676	10	1.2039374E+02	950
66	9	2.1928109E-09	686	10	2.6276542E+00	960
68	10	4.3582802E-02	932	11	3.3548643E+06	1327
70	10	5.4927180E-04	930	11	1.2086624E+05	1325
72	10	5.2593362E-06	927	11	3.3934019E+03	1310
74	10	3.8255691E-08	919	11	7.4235718E+01	1296
76	10	2.1136657E-10	923	11	1.2652843E+00	1244
78	11	1.6800241E-02	1248	12	6.4809046E+06	1783
80	11	1.7376079E-04	1229	12	1.8129413E+05	1722

TABLE XXXI

STANDARD GRAPH COLORING
ESTIMATED VERSUS EXACT CALCULATIONS OF E(K)
$k_{n,p} = \max\{k \mid E(K) < 1.0\}$
$p = 0.50 \quad SF = e^{-23}$

n	$k_{n,p}$	Exact E(K)	Estimated E(K)	Relative Error	Exact Seq	Est Seq	Seq Ratio
20	4	4.2630762E-03	4.2630762E-03	0.0000%	64	38	59.38%
22	4	8.4354971E-05	8.4354971E-05	0.0000%	84	40	47.62%
24	5	4.7571209E-01	4.7571209E-01	0.0000%	164	76	46.34%
26	5	2.0549958E-02	2.0549958E-02	0.0000%	221	84	38.01%
28	5	5.1353005E-04	5.1353005E-04	0.0000%	291	94	32.30%
30	5	7.4175745E-06	7.4175743E-06	0.0000%	377	94	24.93%
32	6	2.9552304E-01	2.9552298E-01	0.0000%	709	161	22.71%
34	6	9.2282795E-03	9.2282793E-03	0.0000%	931	180	19.33%
36	6	1.8252479E-04	1.8252479E-04	0.0000%	1206	176	14.59%
38	6	2.2855250E-06	2.2855249E-06	0.0000%	1540	183	11.88%
40	7	4.0487893E-01	4.0487891E-01	0.0000%	2738	300	10.96%
42	7	1.0515143E-02	1.0515143E-02	0.0000%	3539	300	8.48%
44	7	1.8458027E-04	1.8458026E-04	0.0000%	4526	308	6.81%
46	7	2.1892559E-06	2.1892558E-06	0.0000%	5731	312	5.44%
48	7	1.7539807E-08	1.7539804E-08	0.0000%	7190	296	4.12%
50	8	2.9365266E-02	2.9365263E-02	0.0000%	12450	485	3.90%
52	8	4.8849622E-04	4.8849617E-04	0.0000%	15765	483	3.06%
54	8	5.7658177E-06	5.7658168E-06	0.0000%	19805	485	2.45%
56	8	4.8277294E-08	4.8277283E-08	0.0000%	24699	469	1.90%
58	9	1.8901065E-01	1.8901059E-01	0.0000%	40831	692	1.69%
60	9	3.1005261E-03	3.1005254E-03	0.0000%	51294	703	1.37%
62	9	3.7487140E-05	3.7487125E-05	0.0000%	64015	694	1.08%
64	9	3.3400935E-07	3.3400919E-07	0.0000%	79403	676	0.85%
66	9	2.1928117E-09	2.1928109E-09	0.0000%	97922	686	0.70%
68	10	4.3582838E-02	4.3582802E-02	0.0001%	157564	932	0.59%
70	10	5.4927229E-04	5.4927180E-04	0.0001%	195491	930	0.48%
72	10	5.2593407E-06	5.2593362E-06	0.0001%	241279	927	0.38%
74	10	3.8255730E-08	3.8255691E-08	0.0001%	296320	919	0.31%
76	10	2.1136680E-10	2.1136657E-10	0.0001%	362198	923	0.25%
78	11	1.6800270E-02	1.6800241E-02	0.0002%	567377	1248	0.22%
80	11	1.7376100E-04	1.7376079E-04	0.0001%	697097	1229	0.18%

TABLE XXXII

STANDARD GRAPH COLORING
ESTIMATED VERSUS EXACT CALCULATIONS OF E(K)
$k_{n,p}+1=\min\{k \mid E(K) \geq 1.0\}$
$p=0.50 \quad SF=c^{23}$

n	$k_{n,p}+1$	Exact E(K)	Estimated E(K)	Relative Error	Exact Seq	Est Seq	Seq Ratio
20	5	4.9546188E+01	4.9546188E+01	0.0000%	84	54	64.29%
22	5	6.3764818E+00	6.3764818E+00	0.0000%	119	70	58.82%
24	6	3.2703713E+03	3.2703712E+03	0.0000%	199	98	49.25%
26	6	6.3042828E+02	6.3042827E+02	0.0000%	282	129	45.74%
28	6	7.7187731E+01	7.7187731E+01	0.0000%	391	146	37.34%
30	6	5.9975546E+00	5.9975544E+00	0.0000%	532	146	27.44%
32	7	1.7849024E+04	1.7849024E+04	0.0000%	860	232	26.98%
34	7	2.2120906E+03	2.2120905E+03	0.0000%	1175	247	21.02%
36	7	1.8562259E+02	1.8562259E+02	0.0000%	1579	275	17.42%
38	7	1.0540751E+01	1.0540751E+01	0.0000%	2093	293	14.00%
40	8	1.3525385E+05	1.3525384E+05	0.0000%	3319	412	12.41%
42	8	1.2462260E+04	1.2462259E+04	0.0000%	4417	438	9.92%
44	8	8.1588994E+02	8.1588991E+02	0.0000%	5812	455	7.83%
46	8	3.7940558E+01	3.7940555E+01	0.0000%	7564	461	6.09%
48	8	1.2527900E+00	1.2527899E+00	0.0000%	9749	474	4.86%
50	9	1.2395336E+05	1.2395335E+05	0.0000%	15224	658	4.32%
52	9	6.8770558E+03	6.8770547E+03	0.0000%	19720	679	3.44%
54	9	2.8143957E+02	2.8143951E+02	0.0000%	25331	686	2.71%
56	9	8.4939400E+00	8.4939381E+00	0.0000%	32278	681	2.11%
58	10	2.2338634E+06	2.2338627E+06	0.0000%	49037	945	1.93%
60	10	1.1098853E+05	1.1098846E+05	0.0001%	62740	956	1.52%
62	10	4.1926146E+03	4.1926133E+03	0.0000%	79725	982	1.23%
64	10	1.2039381E+02	1.2039374E+02	0.0001%	100654	950	0.94%
66	10	2.6276555E+00	2.6276542E+00	0.0000%	126299	960	0.76%
68	11	3.3548661E+06	3.3548643E+06	0.0001%	188556	1327	0.70%
70	11	1.2086631E+05	1.2086624E+05	0.0001%	237489	1325	0.56%
72	11	3.3934040E+03	3.3934019E+03	0.0001%	297495	1310	0.44%
74	11	7.4235780E+01	7.4235718E+01	0.0001%	370733	1296	0.35%
76	11	1.2652860E+00	1.2652843E+00	0.0001%	459718	1244	0.27%
78	12	6.4809090E+06	6.4809046E+06	0.0001%	674585	1783	0.26%
80	12	1.8129430E+05	1.8129413E+05	0.0001%	839286	1722	0.21%

present a comparison of some of that data with results obtained by executing S_LB2.Pas, the implementation of CalcEstimatedEOfK. The criteria for choosing the selection factor was to make it small enough so as to ascertain that the algorithm was converging to the results obtained by the Bollobas algorithm.

The results indicate that, for $p=0.50$ and $n \geq 600$, CalcEstimatedEOfK requires a substantial sample size to attain a high degree of accuracy. But, the primary goal is not evaluating E(K). It is to establish the value of $k_{n,p}$. Because of the large variation in the magnitudes of E(K) for $k_{n,p}$ versus $k_{n,p}+1$, it was hypothesized that a sample size on the order of 10^5 would be not only economical to calculate, but also, more than adequate for establishing $k_{n,p}$. Tables XXXV and XXXVI present the results of executing S_LB2.Pas for $n=100,150,...,1000$ and $p=0.20$ with a requirement that SF be small enough to either match the Bollobas values or produce a sample size of at least 10^5. Tables XXXVII and XXXVIII display the results for $p=0.50$ and $n=100,150,...,1000$. In all cases, although the values of E(K) are smaller than those obtained with the Bollabas algorithm, the values of $k_{n,p}$ are in complete agreement. From Tables XXXV and XXXVI, for $n=1000$ and $p=0.20$, both algorithms indicate than a probabilistic lower bound for $\chi(SG_{n,p})$ is 33. If $n=1000$ and $p=0.50$ then Tables XXXVII and XXXVIII indicate that it is very likely that $\chi(SG_{n,p}) \geq 80$.

TABLE XXXIII

STANDARD GRAPH COLORING
COMPARISON OF ESTIMATES FOR E(K)
p=0.50

n	$k_{n,p}$	Bollobas	S_LB2.Pas	Sequences	ln(SF)
100	13	1.055181E-04	1.054398E-04	718	-18
200	22	5.616607E-06	5.615740E-06	8246	-24
300	30	1.612662E-09	1.611042E-09	20429	-26
400	38	3.662054E-04	3.655172E-04	149678	-36
500	45	6.562584E-12	6.292827E-12	588334	-46
600	52	3.394517E-15	3.389024E-15	1254883	-46
700	59	4.608306E-14	4.597338E-14	1117948	-42
800	66	1.000627E-08	7.200000E-09	2256770	-62
900	72	7.187329E-26	6.156652E-26	2041198	-52
1000	79	1.443428E-13	1.365219E-13	2539325	-48

TABLE XXXIV

STANDARD GRAPH COLORING
COMPARISON OF ESTIMATES FOR E(K)
p=0.50

n	$k_{n,p}+1$	Bollobas	S_LB2.Pas	Sequences	ln(SF)
100	14	1.365914E+06	1.364021E+06	609	-16
200	23	7.832963E+08	7.830309E+08	12301	-26
300	31	1.935598E+08	1.927456E+08	25864	-28
400	39	2.882704E+15	2.864776E+15	249119	-40
500	46	6.627035E+09	6.626559E+09	685622	-40
600	53	1.383705E+08	1.369675E+08	1364753	-50
700	60	3.423734E+10	3.399460E+10	1666492	-46
800	67	7.677895E+16	7.477870E+16	1848364	-48
900	73	1.894050E+01	1.459357E+01	4386402	-60
1000	80	1.697616E+14	1.387420E+14	3171158	-54

TABLE XXXV

STANDARD GRAPH COLORING
PROBABILISTIC LOWER BOUND CALCULATIONS
$k_{n,p} = \max\{k \mid E(K) < 1\}$
$p = 0.20$

n	$k_{n,p}$	Bollobas	S_LB2.Pas	Sequences	ln(SF)
100	6	2.256047E-03	2.256047E-03	6041	-42
150	8	2.170367E-01	2.170367E-01	50631	-44
200	9	3.270752E-24	3.270752E-24	116121	-44
250	11	4.179253E-15	4.179251E-15	133186	-34
300	13	5.272471E-02	5.272296E-02	106921	-28
350	14	7.107221E-23	7.104805E-23	124889	-28
400	16	1.020337E-03	1.012292E-03	104064	-26
450	17	1.101352E-23	1.086366E-23	183394	-28
500	18	0.000000E+00	5.544035E-45	140899	-28
550	20	6.659359E-18	5.263072E-18	132350	-28
600	21	2.205376E-37	1.614320E-37	159568	-30
650	23	3.881332E-06	1.192537E-06	136895	-32
700	24	7.859691E-24	2.588954E-24	169880	-32
750	25	0.000000E+00	3.162392E-42	145989	-28
800	27	5.938567E-05	2.257581E-05	258917	-32
850	28	2.682607E-21	3.272169E-22	122062	-34
900	29	6.511443E-38	5.593544E-38	196850	-36
950	30	0.000000E+00	1.676561E-55	162009	-32
1000	32	9.233367E-11	2.277247E-12	188724	-38

TABLE XXXVI

STANDARD GRAPH COLORING
PROBABILISTIC LOWER BOUND CALCULATIONS
$k_{n,p}+1 = \min\{k \mid E(K) \geq 1\}$
$p = 0.20$

n	$k_{n,p}+1$	Bollobas	S_LB2.Pas	Sequences	ln(SF)
100	7	3.604564E+14	3.604563E+14	18769	-44
150	9	1.003795E+21	1.003795E+21	112980	-44
200	10	9.602781E+05	9.602778E+05	101340	-36
250	12	5.139931E+16	5.139923E+16	115911	-30
300	14	9.348999E+30	9.348404E+30	154795	-28
350	15	1.565169E+15	1.562810E+15	104538	-26
400	17	3.802722E+34	3.571805E+34	101500	-26
450	18	6.049697E+18	5.796066E+18	111273	-26
500	19	2.527773E+01	1.932806E+01	116340	-28
550	21	6.425750E+27	6.397172E+27	221272	-32
600	22	4.244035E+11	3.132206E+11	407073	-32
650	24	1.701412E+38	4.261931E+41	129702	-30
700	25	1.674889E+27	1.301000E+27	399657	-30
750	26	5.575587E+11	2.241433E+11	102878	-28
800	28	1.701412E+38	1.443677E+47	176589	-32
850	29	5.932030E+33	4.755668E+32	119118	-34
900	30	2.729703E+19	1.977596E+19	357214	-34
950	31	6.411129E+04	8.469554E+03	163896	-32
1000	33	1.701412E+38	4.387010E+45	128962	-36

TABLE XXXVII

STANDARD GRAPH COLORING
PROBABILISTIC LOWER BOUND CALCULATIONS
$k_{n,p} = \max\{k \mid E(K) < 1\}$
$p = 0.50$

n	$k_{n,p}$	Bollobas	S_LB2.Pas	Sequences	ln(SF)
100	13	1.055181E-04	1.055181E-04	106848	-64
150	18	6.374861E-01	6.374860E-01	107555	-44
200	22	5.616607E-06	5.616632E-06	101002	-38
250	26	6.750317E-09	6.750305E-09	104127	-36
300	30	1.612662E-09	1.612657E-09	112948	-34
350	34	6.723286E-08	6.689219E-08	105308	-36
400	38	3.662054E-04	3.648462E-04	102964	-34
450	41	5.890727E-19	5.878491E-19	100172	-32
500	45	6.562584E-12	6.292826E-12	476665	-44
550	49	4.482894E-03	4.037614E-03	272264	-44
600	52	3.394517E-15	2.840439E-15	102078	-36
650	56	1.980857E-03	1.733102E-03	145657	-36
700	59	4.608306E-14	4.429541E-14	203294	-36
750	62	2.136744E-24	8.035380E-25	621535	-54
800	66	1.000627E-08	2.577990E-10	127652	-56
850	69	1.704750E-17	2.376355E-18	122458	-44
900	72	7.187329E-26	2.510296E-26	122386	-38
950	76	1.039766E-06	2.851100E-07	117651	-38
1000	79	1.443428E-13	9.813984E-14	245640	-38

TABLE XXXVIII

STANDARD GRAPH COLORING
PROBABILISTIC LOWER BOUND-CALCULATIONS
$k_{n,p}+1 = \min\{k \mid E(K) \geq 1\}$
$p = 0.50$

n	$k_{n,p}+1$	Bollobas	S_LB2.Pas	Sequences	ln(SF)
100	14	1.365914E+06	1.365913E+06	110927	-60
150	19	5.124492E+11	5.124491E+11	104308	-42
200	23	7.832963E+08	7.832966E+08	117491	-38
250	27	4.310423E+07	4.310420E+07	136389	-36
300	31	1.935598E+08	1.935511E+08	102064	-34
350	35	8.167174E+10	8.166478E+10	102781	-32
400	39	2.882704E+15	2.862078E+15	186674	-38
450	42	1.880749E+02	1.868319E+02	128518	-34
500	46	6.627035E+09	6.568329E+09	112359	-32
550	50	1.193803E+19	6.810682E+18	102321	-40
600	53	1.383705E+08	7.529002E+07	145969	-42
650	57	1.476338E+20	9.229788E+19	129570	-40
700	60	3.423734E+10	2.895457E+10	122899	-34
750	63	1.318310E+01	7.366557E+00	104442	-38
800	67	7.677895E+16	3.898671E+16	182199	-42
850	70	8.176354E+08	1.037597E+07	118258	-54
900	73	1.894050E+01	2.249441E+00	124929	-44
950	77	2.712409E+20	2.679358E+19	124360	-44
1000	80	1.697616E+14	3.757129E+13	113861	-38

B. PROBABILISTIC LOWER BOUND FOR THE CGCP

Recall that in Section III.J.3 the *projection* of a random composite graph $CG_{n,p}=(V,E)$, was defined to be the standard graph $PG_{n',p'}=(V',E')$, where

$$\begin{aligned} V' &= \bigcup_{i=1}^{n} \{v_{i1},v_{i2},..,v_{ic(v_i)}\} \text{ and} \\ E' &= \{<v_{ik},v_{il}> \mid 1\le i\le n \wedge 1\le k<l\le c(v_i)\} \cup \\ &\quad \{<v_{ik},v_{jl}> \mid <v_i,v_j>\in E \wedge 1\le k\le c(v_i) \wedge 1\le l\le c(v_j)\} \end{aligned} \tag{48}$$

If the Truncated Poisson probability distribution is used to generate vertex chromaticities, then

$$\begin{aligned} &Pr\{c(v_i)=1\}=0.582,\ Pr\{c(v_i)=2\}=0.291,\ Pr\{c(v_i)=3\}=0.097, \\ &Pr\{c(v_i)=4\}=0.024,\ Pr\{c(v_i)=5\}=0.005,\ Pr\{c(v_i)=6\}=0.001, \end{aligned} \tag{49}$$

and $n'=c(CG_{n,p})$ is a normal random variable with

$$\mu(n')=1.582n \text{ and } \sigma(n')=\sqrt{0.661n} \tag{50}$$

Also,

$$p'=p\frac{1.582n-2+1/p}{1.582n-1} \tag{51}$$

By definition any two vertices, $v_{ij},v_{ik}\in V'$ where $1\le i\le n$, $1\le j\le c(v_i)$, and $1\le k\le c(v_i)$, are said to be *isometric*, denoted $v_{ij}\Leftrightarrow v_{ik}$; and, $n_{iso}(v_{ij})$ denotes the number of vertices in $PG_{n',p'}$ which are isometric to v_{ij}. If v_{ij} and v_{ik} are images of a vertex whose chromaticity is m, then they will be said to be *order m isometric*, designated $v_{ij}\Leftrightarrow^{m}v_{ik}$.

In order to simplify the notation in the following material, single subscripts will be used to distinguish vertices in $PG_{n',p'}$. Also, assume that a *feasible* coloring of $PG_{n',p'}$ must assign *consecutive* colors to sets of isometric vertices. Thus, any optimal coloring of $PG_{n',p'}$ will be optimal for $CG_{n,p}$; and, conversely.

Theorem 5. Given an instance of a random composite graph $CG_{n,p,1}=(V,E)$, whose vertex chromaticities are defined by the Truncated Poisson probability distribution, the expected number of different k-colorings of $CG_{n,p,1}$ is given by

$$E(K)=\sum_{(a_i)} \frac{[1.582n]!}{\prod_{i=1}^{k} a_i! \prod_{j=1}^{[1.582n]} n_j! \prod_{i=1}^{[0.291n]} 2! \prod_{i=1}^{[0.097n]} 3! \prod_{i=1}^{[0.024n]} 4! \prod_{i=1}^{[0.005n]} 5! \prod_{i=1}^{[0.001n]} 6!} q_1^{\sum_{i=1}^{k}\binom{a_i}{2}} \sum_{(b_j)} q_2^{\sum_{i=1}^{k-2}\sum_{j=i+2}^{k} b_i b_j} q_3^{\sum_{i=1}^{k-3}\sum_{j=i+3}^{k} b_i b_j} q_4^{\sum_{i=1}^{k-4}\sum_{j=i+4}^{k} b_i b_j} q_5^{\sum_{i=1}^{k-5}\sum_{j=i+5}^{k} b_i b_j} q_6^{\sum_{i=1}^{k-6}\sum_{j=i+6}^{k} b_i b_j} \tag{52}$$

where

$$\begin{aligned} q_1 &= 1-p\frac{1.582n-2+1/p}{1.582n-1}, & q_2 &= 1-\frac{0.368}{1.582n-1}, \\ q_3 &= 1-\frac{0.368}{1.582n-1}, & q_4 &= 1-\frac{0.182}{1.582n-1}, \\ q_5 &= 1-\frac{0.063}{1.582n-1}, & q_6 &= 1-\frac{0.019}{1.582n-1}, \end{aligned} \tag{53}$$

$n=|V|$, the outer sum is over all sequences (a_i) of length k whose sum is $[1.582n]$ and are such that $a_1 \geq a_2 \geq .. \geq a_k \geq 1$, n_j is the number of a_i which equal j, and the inner sum is over all sequences (b_i) that are *distinguishable* permutations of the sequence (a_i).

Proof. Let $PG_{n',p'}$ denote be the projection of $CG_{n,p,1}$. From Equation 51, it follows that the probability that two arbitrary vertices are *not* connected is given by

$$q_1 = 1-p\frac{1.582n-2+1/p}{1.582n-1} \tag{54}$$

Given two arbitrary vertices $v_i, v_j \in V'$, the following probability statements can be made.

$$Pr\{v_i \not\sim^2 v_j\} = Pr\{v_i \not\sim^2 v_j \mid n_{iso}(v_i)=1\} \cdot Pr\{n_{iso}(v_i)=1\} = \frac{1}{1.582n-1} \cdot \frac{2(0.291n)}{1.582n} = \frac{0.368}{1.582n-1} \tag{55}$$

$$Pr\{v_i \not\sim^3 v_j\} = Pr\{v_i \not\sim^3 v_j \mid n_{iso}(v_i)=2\} \cdot Pr\{n_{iso}(v_i)=2\} = \frac{2}{1.582n-1} \cdot \frac{3(0.097n)}{1.582n} = \frac{0.368}{1.582n-1} \tag{56}$$

$$Pr\{v_i \not\sim^4 v_j\} = Pr\{v_i \not\sim^4 v_j \mid n_{iso}(v_i)=3\} \cdot Pr\{n_{iso}(v_i)=3\} = \frac{3}{1.582n-1} \cdot \frac{4(0.024n)}{1.582n} = \frac{0.182}{1.582n-1} \tag{57}$$

$$Pr\{v_i \not\sim^5 v_j\} = Pr\{v_i \not\sim^5 v_j \mid n_{iso}(v_i)=4\} \cdot Pr\{n_{iso}(v_i)=4\} = \frac{4}{1.582n-1} \cdot \frac{5(0.005n)}{1.582n} = \frac{0.063}{1.582n-1} \tag{58}$$

$$Pr\{v_i \not\sim^6 v_j\} = Pr\{v_i \not\sim^6 v_j \mid n_{iso}(v_i)=5\} \cdot Pr\{n_{iso}(v_i)=5\} = \frac{5}{1.582n-1} \cdot \frac{6(0.001n)}{1.582n} = \frac{0.019}{1.582n-1} \tag{59}$$

Thus, if we let q_m denote the probability that two arbitrary vertices are *not* order m isometric, it follows that

$$q_2 = 1-\frac{0.368}{1.582n-1},\ q_3 = 1-\frac{0.368}{1.582n-1},\ q_4 = 1-\frac{0.182}{1.582n-1},\ q_5 = 1-\frac{0.063}{1.582n-1},\ q_6 = 1-\frac{0.019}{1.582n-1} \tag{60}$$

Now, suppose we have an arbitrary k-partition, $A=\{A_1,A_2,..,A_k\}$, where $|A_i|=a_i$, $i=1,2,..,k$, and $a_1 \geq a_2 \geq .. \geq a_k \geq 1$. In order for the function f, where $f(v_j)=i$ iff $v_j \in A_i$, to be a feasible coloring function for $PG_{n',p'}$, it must be the case that no two elements of A_i, $i=1,2,..,k$, can have an edge between them. The probability of this occurring is given by

$$q_1^{\sum_{i=1}^{k}\binom{a_i}{2}} \tag{61}$$

It must also be the case that no vertex in A_i is order 2 isometric to a vertex in A_j, where $1 \le i \le k\text{-}2$ and $i+2 \le j \le k$. The associated probability is given by

$$q_2^{\sum_{i=1}^{k-2}\sum_{j=i+2}^{k} a_i a_j} \tag{62}$$

Likewise, it must be true that no vertex in A_i is order 3 isometric to a vertex in A_j, where $1 \le i \le k\text{-}3$ and $i+3 \le j \le k$. The associated probability is given by

$$q_3^{\sum_{i=1}^{k-3}\sum_{j=i+3}^{k} a_i a_j} \tag{63}$$

Extending this requirement to order 4, 5, and 6 isometric vertices, we find that the probability that f is a feasible coloring function is given by

$$q_1^{\sum_{i=1}^{k}\binom{a_i}{2}}\, q_2^{\sum_{i=1}^{k-2}\sum_{j=i+2}^{k} a_i a_j}\, q_3^{\sum_{i=1}^{k-3}\sum_{j=i+3}^{k} a_i a_j}\, q_4^{\sum_{i=1}^{k-4}\sum_{j=i+4}^{k} a_i a_j}\, q_5^{\sum_{i=1}^{k-5}\sum_{j=i+5}^{k} a_i a_j}\, q_6^{\sum_{i=1}^{k-6}\sum_{j=i+6}^{k} a_i a_j} \tag{64}$$

where a factor is assigned the value 1 if the associated summation is undefined.

Now, in order to calculate the expected number of k colorings associated with partition sets of size $a_1, a_2, .., a_k$, we must be able to count the number of *distinguishable* ways of generating the sets $A_1, A_2, .., A_k$ from $V' = \{v_1, v_2, .., v_{[1.582n]}\}$. Since brothers are *not* distinguishable with respect to coloring, the factor

$$\frac{[1.582n]!}{\prod_{i=1}^{k} a_i!\ \prod_{j=1}^{[1.582n]} n_j!\ \prod_{i=1}^{[0.291n]} 2!\ \prod_{i=1}^{[0.097n]} 3!\ \prod_{i=1}^{[0.024n]} 4!\ \prod_{i=1}^{[0.005n]} 5!\ \prod_{i=1}^{[0.001n]} 6!} \tag{65}$$

represents this value. Thus, the percentage or expected number which will satisfy the coloring constraints is given by

$$\frac{[1.582n]!}{\prod_{i=1}^{k} a_i!\ \prod_{j=1}^{[1.582n]} n_j!\ \prod_{i=1}^{[0.291n]} 2!\ \prod_{i=1}^{[0.097n]} 3!\ \prod_{i=1}^{[0.024n]} 4!\ \prod_{i=1}^{[0.005n]} 5!\ \prod_{i=1}^{[0.001n]} 6!} \cdot q_1^{\sum_{i=1}^{k}\binom{a_i}{2}}\, q_2^{\sum_{i=1}^{k-2}\sum_{j=i+2}^{k} a_i a_j}\, q_3^{\sum_{i=1}^{k-3}\sum_{j=i+3}^{k} a_i a_j}\, q_4^{\sum_{i=1}^{k-4}\sum_{j=i+4}^{k} a_i a_j}\, q_5^{\sum_{i=1}^{k-5}\sum_{j=i+5}^{k} a_i a_j}\, q_6^{\sum_{i=1}^{k-6}\sum_{j=i+6}^{k} a_i a_j} \tag{66}$$

For the SGCP, the order of $A_1,A_2,..,A_k$ could be changed and the resulting sequence would also represent a feasible k-coloring. However, the requirement that sets of isometric vertices be assigned consecutive colors means that this is not necessarily the case for the projection of an instance of the CGCP. Thus, for each sequence, $a_1,a_2,..,a_k$, we must consider all distinguishable permutations.

Combining the above concepts and applying simple algebra, Equation 52 easily follows.

Consider the significance of Theorem 5. Using the notation defined in the previous section, for specified values of n and p, let $k_{n,p,1}$ denote $\max\{k|E(K)<1\}$. Also, recall that $\overline{\chi}(CG_{n,p,1})$ represents the mean value of $\chi(CG_{n,p,1})$ for a random sample of random composite graphs of order n and edge density p. With large probability it can be concluded that $k_{n,p,1}+1 \leq \overline{\chi}(CG_{n,p,1})$. Thus, $k_{n,p,1}+1$ is a probabilistic *lower bound* for $\overline{\chi}(CG_{n,p,1})$.

Unfortunately, even if the implicit enumeration techniques which were used in the previous section are applied, Equation 52 would still be intractable to compute for large values of n. The problem is that the number of distinguishable permutations of each sequence $a_1,a_2,..,a_k$ is given by

$$\frac{k!}{2\prod_{j=1}^{[1.582n]} n_j!} \tag{67}$$

The divisor 2 is required because permutations which are reversals of each other are indistinguishable with respect to coloring. A method for dealing with this problem was developed by utilizing the following theory.

Lemma 3a. Given any set of whole numbers $A=\{a_1,a_2,..,a_k\}$, where $a_1 \geq a_2 \geq .. \geq a_k \geq 1$, let $b_1,b_2,..,b_k$ be an arbitrary permutation of the elements in A, and let

$$S_1 = \sum_{i=1}^{k-1} b_i b_{i+1}, \quad S_2 = \sum_{i=1}^{k-2} b_i b_{i+2}, \quad S_3 = \sum_{i=1}^{k-3} b_i b_{i+3},$$
$$S_4 = \sum_{i=1}^{k-4} b_i b_{i+4}, \quad S_5 = \sum_{i=1}^{k-5} b_i b_{i+5} \tag{68}$$

Upper bounds for S_1, S_2, S_3, S_4, and S_5 are given, respectively, by

$$\hat{S}_1 = a_1 \sum_{i=1}^{k-1} a_i, \quad \hat{S}_2 = a_1 \sum_{i=1}^{k-2} a_i, \quad \hat{S}_3 = a_1 \sum_{i=1}^{k-3} a_i,$$
$$\hat{S}_4 = a_1 \sum_{i=1}^{k-4} a_i, \quad \hat{S}_5 = a_1 \sum_{i=1}^{k-5} a_i \tag{69}$$

Proof. Consider S_1. Since $b_1,b_2,..,b_k$ is some permutation of $a_1,a_2,..,a_k$, it follows that $a_1 \geq b_i$ and $a_k \leq b_i$, $i = 1,2,..,k$. Therefore,

$$S_1 = \sum_{i=1}^{k-1} b_i b_{i+1} \leq \sum_{i=1}^{k-1} b_i a_1 = a_1 \sum_{i=1}^{k-1} b_i$$
$$= a_1 \left(\sum_{i=1}^{k} b_i - b_k \right) \leq a_1 \sum_{i=1}^{k-1} a_i = \hat{S}_1 \tag{70}$$

The upper bound formulas for S_2, S_3, S_4, and S_5 follow similarly.

Lemma 3b. Given any set of whole numbers $A = \{a_1,a_2,..,a_k\}$, where $a_1 \geq a_2 \geq .. \geq a_k \geq 1$, let $b_1,b_2,..,b_k$ be an arbitrary permutation of the elements in A, and let

$$T_2 = \sum_{i=1}^{k-2} \sum_{j=i+2}^{k} b_i b_j, \quad T_3 = \sum_{i=1}^{k-3} \sum_{j=i+3}^{k} b_i b_j, \quad T_4 = \sum_{i=1}^{k-4} \sum_{j=i+4}^{k} b_i b_j,$$
$$T_5 = \sum_{i=1}^{k-5} \sum_{j=i+5}^{k} b_i b_j, \quad T_6 = \sum_{i=1}^{k-6} \sum_{j=i+6}^{k} b_i b_j \tag{71}$$

Lower bounds for T_2, T_3, T_4, T_5, and T_6 are given, respectively, by

$$\begin{aligned} &T-\hat{S}_1,\ T-\left(\hat{S}_1+\hat{S}_2\right),\ T-\left(\hat{S}_1+\hat{S}_2+\hat{S}_3\right), \\ &T-\left(\hat{S}_1+\hat{S}_2+\hat{S}_3+\hat{S}_4\right),\ T-\left(\hat{S}_1+\hat{S}_2+\hat{S}_3+\hat{S}_4+\hat{S}_5\right) \end{aligned} \tag{72}$$

where

$$T-\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} a_i a_j \tag{73}$$

Proof. Since T is simply the sum of all possible products of terms in A, it follows that for all permutations $b_1,b_2,..,b_k$,

$$\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} b_i b_j - T \tag{74}$$

Consider T_2. Easily, Equation 74 and Lemma 3a can be applied to justify

$$T_2-\sum_{i=1}^{k-2}\sum_{j=i+2}^{k} b_i b_j-\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} b_i b_j-\sum_{i=1}^{k-1} b_i b_{i+1}-T-S_1 \geq T-\hat{S}_1 \tag{75}$$

Now consider T_3. Equation 74 and Lemma 3a can also be used to justify

$$\begin{aligned} T_3-\sum_{i=1}^{k-3}\sum_{j=i+3}^{k} b_i b_j-\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} b_i b_j-\left(\sum_{i=1}^{k-1} b_i b_{i+1}+\sum_{i=1}^{k-2} b_i b_{i+2}\right) \\ -T-\left(S_1+S_2\right) \geq T-\left(\hat{S}_1+\hat{S}_2\right) \end{aligned} \tag{76}$$

In a similar fashion, the results for T_4, T_5, and T_6 follow.

Lemma 4a. Given any set of whole numbers $A=\{a_1,a_2,..,a_k\}$, where $a_1 \geq a_2 \geq .. \geq a_k \geq 1$, let $b_1,b_2,..,b_k$ be an arbitrary permutation of the elements in A, and let

$$\begin{aligned} &S_1-\sum_{i=1}^{k-1} b_i b_{i+1},\quad S_2-\sum_{i=1}^{k-2} b_i b_{i+2},\quad S_3-\sum_{i=1}^{k-3} b_i b_{i+3}, \\ &S_4-\sum_{i=1}^{k-4} b_i b_{i+4},\quad S_5-\sum_{i=1}^{k-5} b_i b_{i+5} \end{aligned} \tag{77}$$

Lower bounds for S_1, S_2, S_3, S_4, and S_5 are given, respectively, by

$$\begin{aligned} &\check{S}_1 = a_k \sum_{i=2}^{k} a_i, \quad \check{S}_2 = a_k \sum_{i=3}^{k} a_i, \quad \check{S}_3 = a_k \sum_{i=4}^{k} a_i, \\ &\check{S}_4 = a_k \sum_{i=5}^{k} a_i, \quad \check{S}_5 = a_k \sum_{i=6}^{k} a_i \end{aligned} \tag{78}$$

Proof. Consider S_1. Since $b_1,b_2,..,b_k$ is some permutation of $a_1,a_2,..,a_k$, it follows that $a_1 \geq b_i$ and $a_k \leq b_i$, $i = 1,2,..,k$. Therefore,

$$\begin{aligned} S_1 &= \sum_{i=1}^{k-1} b_i b_{i+1} \geq \sum_{i=1}^{k-1} b_i a_k - a_k \sum_{i=1}^{k-1} b_i \\ &= a_k \left(\sum_{i=1}^{k} b_i - b_k \right) \geq a_k \sum_{i=2}^{k} a_i = \check{S}_1 \end{aligned} \tag{79}$$

The lower bound formulas for S_2, S_3, S_4, and S_5 follow similarly.

Lemma 4b. Given any set of whole numbers $A = \{a_1,a_2,..,a_k\}$, where $a_1 \geq a_2 \geq .. \geq a_k \geq 1$, let $b_1,b_2,..,b_k$ be an arbitrary permutation of the elements in A, and let

$$\begin{aligned} &T_2 = \sum_{i=1}^{k-2} \sum_{j=i+2}^{k} b_i b_j, \quad T_3 = \sum_{i=1}^{k-3} \sum_{j=i+3}^{k} b_i b_j, \quad T_4 = \sum_{i=1}^{k-4} \sum_{j=i+4}^{k} b_i b_j, \\ &T_5 = \sum_{i=1}^{k-5} \sum_{j=i+5}^{k} b_i b_j, \quad T_6 = \sum_{i=1}^{k-6} \sum_{j=i+6}^{k} b_i b_j \end{aligned} \tag{80}$$

Upper bounds for T_2, T_3, T_4, T_5, and T_6 are given, respectively, by

$$\begin{aligned} &T - \check{S}_1, \quad T - (\check{S}_1 + \check{S}_2), \quad T - (\check{S}_1 + \check{S}_2 + \check{S}_3), \\ &T - (\check{S}_1 + \check{S}_2 + \check{S}_3 + \check{S}_4), \quad T - (\check{S}_1 + \check{S}_2 + \check{S}_3 + \check{S}_4 + \check{S}_5) \end{aligned} \tag{81}$$

Proof. The proof for Lemma 4b is very similar to the one of Lemma 3b.

Together, Theorem 5, Equation 67, Lemma 3b, and Lemma 4b can be used to justify the following theorem.

Theorem 6. Given an instance of a random composite graph, $CG_{n,p,1}=(V,E)$, an *upper bound*, $E(K)_{UB}$, on the expected number of different k-colorings of $CG_{n,p,1}$ is given by

$$\sum_{(a_i)} \frac{[1.582n]!}{\prod_{i=1}^{k} a_i! \prod_{j=1}^{[1.582n]} n_j! \prod_{i=1}^{[0.291n]} 2! \prod_{i=1}^{[0.097n]} 3! \prod_{i=1}^{[0.024n]} 4! \prod_{i=1}^{[0.005n]} 5! \prod_{i=1}^{[0.001n]} 6!} \cdot \frac{k!}{2\prod_{j=1}^{n} n_j!} \quad (82)$$
$$q_1^{\sum_{i=1}^{k}\binom{a_i}{2}} q_2^{T-\hat{S}_1} q_3^{T-(\hat{S}_1+\hat{S}_2)} q_4^{T-(\hat{S}_1+\hat{S}_2+\hat{S}_3)} q_5^{T-(\hat{S}_1+\hat{S}_2+\hat{S}_3+\hat{S}_4)} q_6^{T-(\hat{S}_1+\hat{S}_2+\hat{S}_3+\hat{S}_4+\hat{S}_5)}$$

and a *lower bound*, $E(K)_{LB}$, is given by

$$\sum_{(a_i)} \frac{[1.582n]!}{\prod_{i=1}^{k} a_i! \prod_{j=1}^{[1.582n]} n_j! \prod_{i=1}^{[0.291n]} 2! \prod_{i=1}^{[0.097n]} 3! \prod_{i=1}^{[0.024n]} 4! \prod_{i=1}^{[0.005n]} 5! \prod_{i=1}^{[0.001n]} 6!} \cdot \frac{k!}{2\prod_{j=1}^{n} n_j!} \quad (83)$$
$$q_1^{\sum_{i=1}^{k}\binom{a_i}{2}} q_2^{T-\check{S}_1} q_3^{T-(\check{S}_1+\check{S}_2)} q_4^{T-(\check{S}_1+\check{S}_2+\check{S}_3)} q_5^{T-(\check{S}_1+\check{S}_2+\check{S}_3+\check{S}_4)} q_6^{T-(\check{S}_1+\check{S}_2+\check{S}_3+\check{S}_4+\check{S}_5)}$$

where

$$\begin{aligned} q_1 &= 1-p\frac{1.582n-2+1/p}{1.582n-1}, & q_2 &= 1-\frac{0.368}{1.582n-1}, \\ q_3 &= 1-\frac{0.368}{1.582n-1}, & q_4 &= 1-\frac{0.182}{1.582n-1}, \\ q_5 &= 1-\frac{0.063}{1.582n-1}, & q_6 &= 1-\frac{0.019}{1.582n-1}, \end{aligned} \quad (84)$$

TABLE XXXIX

COMPOSITE GRAPH COLORING
EXPECTED NUMBER OF K COLORINGS
UPPER BOUND FORMULA
$\check{k}_{n,p,1} = \max\{k | E(K)_{UB} < 1\}$
p=0.50

n	$\check{k}_{n,p,1}$	$E(K)_{UB}$	Sequences	$\check{k}_{n,p,1}+1$	$E(K)_{UB}$	Sequences
20	6	2.2434318E-05	709	7	2.9444236E+00	860
22	7	2.0772256E-03	1367	8	7.6942159E+01	1527
24	7	6.9477141E-06	2093	8	1.4560992E+00	2462
26	8	2.0807151E-03	3828	9	1.3680535E+02	4206
28	8	1.6678149E-05	5812	9	5.0236051E+00	6615
30	9	4.6034302E-02	10156	10	4.7606517E+03	10936
32	9	6.2984838E-05	17354	10	4.0333118E+01	19466
34	10	3.3435173E-01	29292	11	7.3143010E+04	31316
36	10	2.9593365E-03	43214	11	2.3612397E+03	47420
38	10	1.1620557E-06	62740	11	3.6679377E+00	70515
40	11	1.9056366E-02	103226	12	1.8426554E+04	111036
42	11	1.1282204E-04	148847	12	3.8352076E+02	163540
44	12	3.5773262E-01	268079	13	5.9286955E+05	285373
46	12	2.2220838E-03	382562	13	1.1614183E+04	414624
48	12	6.9786247E-07	539672	13	1.1738220E+01	595056
50	13	4.5111645E-02	844504	14	2.3944404E+05	899427

$n = |V|$, the outer sum is over all sequences (a_i) of length k whose sum is n and are such that $a_1 \geq a_2 \geq .. \geq a_k \geq 1$, and n_j is the number of a_i which equal j.

If $\check{k}_{n,p,1} = \max\{k | E(K)_{UB} < 1\}$ and $\hat{k}_{n,p,1} = \max\{k | E(K)_{LB} < 1\}$, then it follows that $\check{k}_{n,p,1} \leq k_{n,p,1} \leq \hat{k}_{n,p,1}$ and $\check{k}_{n,p,1} + 1 \leq k_{n,p,1} + 1 \leq \hat{k}_{n,p,1} + 1$. Thus, an algorithm for calculating values of Equations 82 and 83 can be used to establish an interval within which a probabilistic lower bound for $\overline{\chi}(CG_{n,p,1})$ must be located.

Easily, the algorithm CalcExactEOfK, which was designed to calculate exact values for Equation 37, can also be used to produce exact values of Equations 82 and 83, if appropriate implementations of the *update* procedures are provided. Such implementations were developed and are included in the Appendix as *Listing 16.*

TABLE XL

COMPOSITE GRAPH COLORING
EXPECTED NUMBER OF K COLORINGS
LOWER BOUND FORMULA
$\hat{k}_{n,p,1} = \max\{k \mid E(K)_{LB} < 1\}$
p=0.50

n	$\hat{k}_{n,p,1}$	$E(K)_{LB}$	Sequences	$\hat{k}_{n,p,1}+1$	$E(K)_{LB}$	Sequences
20	7	2.4125073E-03	860	8	1.3548253E+01	919
22	8	4.3948104E-02	1527	9	1.2940165E+02	1549
24	8	8.1888970E-04	2462	9	9.8140756E+00	2592
26	9	5.5256041E-02	4206	10	3.5289005E+02	4242
28	9	2.0701837E-03	6615	10	4.4608878E+01	6912
30	9	2.0272181E-05	10156	10	1.4391442E+00	10936
32	10	1.0959604E-02	19466	11	5.2949935E+02	20298
34	10	9.0183826E-05	29292	11	1.4702311E+01	31316
36	11	5.0461012E-01	47420	12	3.6132614E+04	48849
38	11	7.7195788E-04	70515	12	1.7252977E+02	74287
40	11	3.8287240E-06	103226	12	3.1366571E+00	111036
42	12	6.4883209E-02	163540	13	1.8823822E+04	169919
44	12	5.0108702E-05	268079	13	7.1550268E+01	285373
46	12	3.1776261E-07	382562	13	1.4023700E+00	414624
48	13	1.3677882E-03	595056	14	2.1894043E+03	624188
50	13	5.2051544E-06	844504	14	2.4550599E+01	899427

CTP_LB1.Pas and *Listing 17. CTP_LB2.Pas*, respectively. For any given pair of values for n and p, CTP_LB1.Pas is designed to identify $\check{k}_{n,p,1}$ and CTP_LB2.Pas will calculate the value of $\hat{k}_{n,p,1}$. Thus, together, the two programs can be used to obtain an interval within which the probabilisitic lower bound $k_{n,p,1}+1$ must lie.

The results of executing these programs for p=0.50 and n=20,22,..,50, are provided in Tables XXXIX and XL. Note that for n=48, Table XXXIX indicates that $\check{k}_{n,p,1}+1=13$ and Table XL documents that $\hat{k}_{n,p,1}+1=14$. Thus, a probabilistic lower bound interval for $\overline{\chi}(CG_{n,p,1})$ is [13,14]. It is also worth noting that in all cases the difference between $\hat{k}_{n,p,1}$ and $\check{k}_{n,p,1}$ is either 0 or 1.

Evidence of the validity of this approach is provided by the exact Vertex-Sequential with Dynamic Reordering Algorithm for the CGCP presented in Chapter III. In the columns headed $\overline{\chi}(CG_{n,p,1})$, Table XLI contains the mean values of the

TABLE XLI

COMPOSITE GRAPH COLORING
EXACT VERSUS PROBABILISTIC MINIMUMS

n	p=0.25 $\check{k}_{n,p,1}+1$	p=0.25 $\hat{k}_{n,p,1}+1$	p=0.25 $\bar{\chi}(CG_{n,p,1})$	p=0.50 $\check{k}_{n,p,1}+1$	p=0.50 $\hat{k}_{n,p,1}+1$	p=0.50 $\bar{\chi}(CG_{n,p,1})$
20	5	5	6.92	7	8	9.92
22	5	6	7.12	8	9	10.48
24	5	6	7.60	8	9	11.32
26	6	6	7.92	9	10	11.72
28	6	6	7.80	9	10	12.08
30	6	6	8.20	10	10	12.32
32	6	7	8.64	10	11	13.04
34	6	7	8.96			
36	6	7	8.76			
38	7	7	8.80			
40	7	7	9.28			
42	7	8	9.64			
44	7	8	9.68			

chromatic numbers for 25 instances of the CGCP at each specified value of n and p. In the columns headed $\check{k}_{n,p,1}+1$ and $\hat{k}_{n,p,1}+1$, the lower and upper limits of the associated probabilistic lower bound intervals are displayed. In all cases, *both* interval limits are smaller then the sample mean values. In fact, at least for the range of graph orders involved, they are very conservative.

As with the SGCP, the *Sequences* columns of Tables XXXIX and XL illustrate that CalcExactEOfK becomes intractable to compute for relatively small values of n. Also, as with the SGCP, it was hypothesized that the implicit enumeration algorithm CalcEstimatedEOfK, listed in Figure 32, could be used to produce approximations for $E(K)_{UB}$ and $E(K)_{LB}$ which would be adequate for identifying $\check{k}_{n,p,1}$ and $\hat{k}_{n,p,1}$. The resulting implementations are provided in the Appendix as *Listing 18. CTP_LB3.Pas* and *Listing 19. CTP_LB4.Pas*. Results of executing these programs for p=0.50 and n=20,22,..,50 using a selection factor of $SF=e^{-30}$ are provided in Tables XLII and XLIII, respectively. In all cases, the implicit enumeration approximation is accurate to at least 5 significant digits. Also,

TABLE XLII

COMPOSITE GRAPH COLORING
ESTIMATED EXPECTED NUMBER OF K COLORINGS
UPPER BOUND FORMULA
$k_{n,p,1} = \max\{k \mid E(K)_{UB} < 1\}$
$p = 0.50 \quad SF = e^{-30}$

n	$k_{n,p,1}$	$E(K)_{UB}$	Scquences	$k_{n,p,1}+1$	$E(K)_{UB}$	Scquences
20	6	2.2434318E-05	393	7	2.9444236E+00	504
22	7	2.0772256E-03	650	8	7.6942159E+01	768
24	7	6.9477141E-06	762	8	1.4560992E+00	954
26	8	2.0807151E-03	1244	9	1.3680535E+02	1408
28	8	1.6678149E-05	1426	9	5.0236051E+00	1760
30	9	4.6034302E-02	2115	10	4.7606517E+03	2393
32	9	6.2984838E-05	2512	10	4.0333118E+01	3017
34	10	3.3435172E-01	3487	11	7.3143010E+04	4026
36	10	2.9593365E-03	3657	11	2.3612397E+03	4504
38	10	1.1620557E-06	4166	11	3.6679369E+00	4658
40	11	1.9056366E-02	5688	12	1.8426547E+04	5953
42	11	1.1282202E-04	5913	12	3.8352074E+02	7169
44	12	3.5773258E-01	7997	13	5.9286839E+05	7846
46	12	2.2220833E-03	8138	13	1.1614130E+04	9308
48	12	6.9786196E-07	7973	13	1.1737729E+01	8200
50	13	4.5111267E-02	10331	14	2.3944247E+05	11704

while the total number of possible sequences increases exponentially with n, the number actually processed grows much more slowly.

Although the number of sequences which need to be processed by CalcEstimatedEOfK are very small in comparison to the total of all possible sequences, as was shown in the case of the SGCP, in order to maintain a high degree of accuracy, this number borders on being intractable for graphs of order 600 and above. Since $n' = [1.582n]$, the problem would be even worse for the CGCP. But, remember that the goal is to identify a probabilistic lower bound, and, because of the large variation in both $E(K)_{UB}$ and $E(K)_{LB}$ for k and $k+1$, a high degree of accuracy is not necessary. In the previous section, it was shown that using a value of the selection factor SF which processed at least 10^5 sequences produced values for $k_{n,p}$ which were in complete agreement with the more accurate Bollobas algorithm.

TABLE XLIII

COMPOSITE GRAPH COLORING
ESTIMATED EXPECTED NUMBER OF K COLORINGS
LOWER BOUND FORMULA
$\hat{k}_{n,p,1}=\max\{k|E(K)_{LB}<1\}$
$p=0.50 \quad SF=e^{30}$

n	$\hat{k}_{n,p,1}$	$E(K)_{LB}$	Sequences	$\hat{k}_{n,p,1}+1$	$E(K)_{LB}$	Sequences
20	7	2.4125073E-03	320	8	1.3548253E+01	380
22	8	4.3948104E-02	478	9	1.2940165E+02	544
24	8	8.1888970E-04	590	9	9.8140756E+00	680
26	9	5.5256041E-02	843	10	3.5289005E+02	957
28	9	2.0701837E-03	973	10	4.4608878E+01	1149
30	9	2.0272181E-05	1060	10	1.4391442E+00	1275
32	10	1.0959604E-02	1509	11	5.2949934E+02	1757
34	10	9.0183826E-05	1673	11	1.4702310E+01	2019
36	11	5.0461012E-01	2263	12	3.6132612E+04	2585
38	11	7.7195788E-04	2485	12	1.7252977E+02	3063
40	11	3.8287240E-06	2546	12	3.1366571E+00	3123
42	12	6.4883209E-02	3457	13	1.8823821E+04	3890
44	12	5.0108701E-05	3531	13	7.1550267E+01	4178
46	12	3.1776260E-07	3658	13	1.4023700E+00	4401
48	13	1.3677882E-03	4688	14	2.1894041E+03	5290
50	13	5.2051539E-06	4564	14	2.4550599E+01	5975

Also, the formula for $E(K)_{UB}$ produces values which are upper bounds on the actual value of E(K). Finally, Table XLI suggests that the interval $[\check{k}_{n,p,1},\hat{k}_{n,p,1}]$ is conservative. Hence, it was concluded that, if CP_LB3.Pas and CP_LB4.Pas were executed with a value of SF which processed a minimum of $5*10^5$ sequences, with a cutoff at $7.5*10^5$, then accurate values for $\check{k}_{n,p,1}$ and $\hat{k}_{n,p,1}$ would result.

Tables XLIV and XLV display the results which were obtained for p=0.20 and n=50,100,..,1000. Tables XLVI and XLVII contain the results for p=0.50 and n=50,100,..,1000. The absence of a value for ln(SF) indicates that the associated value of E(K) is exact. Notice that, for all values of n and p, the difference in $\hat{k}_{n,p,1}+1$ and $\check{k}_{n,p,1}+1$ is either 0 or 1.

TABLE XLIV

COMPOSITE GRAPH COLORING
ESTIMATED EXPECTED NUMBER OF K COLORINGS
UPPER BOUND FORMULA
$k_{n,p,1} = \max\{k \mid E(K)_{UB} < 1\}$
$p = 0.20$

n	$k_{n,p,1}$	$E(K)_{UB}$	Seq	ln(SF)	$k_{n,p,1}+1$	$E(K)_{UB}$	Seq	ln(SF)
50	6	2.313156E-04	46461		7	8.437298E+07	108869	
100	9	5.186185E-19	617969	-52	10	1.150245E+01	704391	-44
150	13	4.106826E-08	750000	-36	14	6.282920E+13	750000	-36
200	16	1.418089E-07	750000	-40	17	1.821427E+17	750000	-44
250	19	5.171093E-04	649656	-40	20	7.889629E+23	505016	-40
300	21	4.858828E-29	506431	-44	22	2.331271E+03	750000	-48
350	24	9.005070E-19	750000	-52	25	7.092489E+14	750000	-56
400	27	6.572302E-03	750000	-60	28	6.452119E+31	564314	-52
450	29	9.127721E-21	694912	-56	30	3.132389E+17	750000	-52
500	31	5.965263E-38	750000	-60	32	9.479811E+02	648666	-52
550	34	3.170248E-17	552843	-56	35	6.478617E+23	593014	-52
600	36	3.487866E-33	581050	-64	37	1.953163E+11	562928	-56
650	39	8.841915E-05	541569	-68	40	4.390481E+40	539996	-56
700	41	8.841915E-05	541569	-68	42	1.339006E+31	750000	-72
750	43	1.267510E-27	750000	-64	44	9.297787E+18	750000	-68
800	45	4.971057E-15	750000	-84	46	5.553089E+08	750000	-68
850	45	1.132847E-03	750000	-68	46	5.508799E+27	750000	-84
900	46	2.511314E-01	750000	-72	47	8.292169E+31	501112	-80
950	47	1.702686E-30	750000	-76	48	2.828076E+08	538877	-80
1000	49	1.769488E-22	693280	-84	50	1.951640E+44	750000	-72

TABLE XLV

COMPOSITE GRAPH COLORING
ESTIMATED EXPECTED NUMBER OF K COLORINGS
LOWER BOUND FORMULA
$\hat{k}_{n,p,1} = \max\{k \mid E(K)_{LB} < 1\}$
$p = 0.20$

n	$\hat{k}_{n,p,1}$	$E(K)_{LB}$	Seq	ln(SF)	$\hat{k}_{n,p,1}+1$	$E(K)_{LB}$	Seq	ln(SF)
50	6	4.041179E-09	46461		7	6.581340E+02	108869	
100	10	8.307269E-06	531159	-64	11	1.120415E+11	516735	-54
150	13	6.204790E-15	643236	-44	14	8.059067E+06	585270	-40
200	16	1.414970E-14	750000	-40	17	6.088217E+10	750000	-40
250	19	2.924157E-11	750000	-40	20	6.318226E+16	692092	-40
300	22	4.398352E-04	717446	-44	23	5.272367E+25	750000	-48
350	24	5.968765E-24	750000	-52	25	1.507244E+09	549716	-52
400	27	3.060420E-09	750000	-56	28	2.971260E+25	750000	-52
450	29	7.822259E-27	750000	-54	30	1.825521E+11	635115	-50
500	32	2.950627E-03	749004	-52	33	1.519162E+34	750000	-56
550	34	3.621125E-23	728713	-56	35	2.849087E+18	625007	-50
600	36	1.277771E-38	639065	-64	37	1.626771E+06	750000	-56
650	39	3.409199E-10	742893	-68	40	6.009369E+35	659566	-56
700	41	1.024328E-22	628562	-64	42	1.167697E+23	552974	-64
750	43	1.606782E-33	698283	-60	44	2.260641E+14	750000	-68
800	45	2.793626E-22	750000	-80	46	1.284992E+05	750000	-66
850	45	3.375922E-06	682048	-56	46	9.331059E+22	750000	-80
900	46	1.664199E-05	750000	-68	47	5.971288E+31	750000	-80
950	47	5.443529E-34	750000	-72	48	9.525240E+05	750000	-80
1000	49	3.020930E-22	750000	-80	50	3.822631E+39	680570	-64

TABLE XLVI

COMPOSITE GRAPH COLORING
ESTIMATED EXPECTED NUMBER OF K COLORINGS
UPPER BOUND FORMULA
$k_{n,p,1} = \max\{k | E(K)_{UB} < 1\}$
$p = 0.50$

n	$k_{n,p,1}$	$E(K)_{UB}$	Seq	ln(SF)	$k_{n,p,1}+1$	$E(K)_{UB}$	Seq	ln(SF)
50	13	4.511164E-02	844504		14	2.394440E+05	899427	
100	21	3.341281E-09	750000	-52	22	5.033594E+01	750000	-52
150	29	3.156413E-11	563652	-52	30	3.324453E+01	723194	-48
200	37	9.171237E-01	750000	-64	38	6.255187E+12	649294	-64
250	43	1.751183E-16	550568	-76	44	1.404062E+00	750000	-64
300	50	2.084249E-13	750000	-68	51	4.890729E+02	750000	-80
350	57	1.025990E-03	750000	-60	58	3.430729E+12	619624	-64
400	63	1.285678E-12	750000	-92	64	1.142446E+09	750000	-64
450	70	1.151367E-02	589676	-104	71	3.613183E+22	522727	-68
500	75	2.199801E-16	657049	-68	76	1.329378E+02	750000	-80
550	81	8.977141E-19	543124	-66	82	2.447622E+01	750000	-68
600	87	2.006900E-15	651875	-64	88	2.165000E+05	531574	-64
650	93	8.784054E-29	509733	-112	94	1.189634E+10	521413	-60
700	100	5.175709E-04	563573	-112	101	1.191281E+34	750000	-72
750	105	7.919912E-14	613283	-96	106	2.580932E+04	662771	-112
800	111	1.424872E-01	750000	-92	112	2.786964E+16	740425	-100
850	116	1.317299E-06	631351	-76	117	2.532382E+13	558466	-82
900	121	3.435168E-15	728442	-68	122	1.015628E+07	687414	-72
950	127	1.247502E-09	750000	-80	128	7.073050E+19	547270	-66
1000	132	4.766458E-28	503072	-92	133	3.229948E+05	750000	-80

TABLE XLVII

COMPOSITE GRAPH COLORING
ESTIMATED EXPECTED NUMBER OF K COLORINGS
LOWER BOUND FORMULA
$\hat{k}_{n,p,1} = \max\{k \mid E(K)_{LB} < 1\}$
$p = 0.50$

n	$\hat{k}_{n,p,1}$	$E(K)_{LB}$	Seq	ln(SF)	$\hat{k}_{n,p,1}+1$	$E(K)_{LB}$	Seq	ln(SF)
50	13	5.205154E-06	844504		14	2.455060E+01	899427	
100	22	1.186310E-03	584649	-56	23	2.451039E+06	590909	-56
150	30	3.192371E-04	605258	-50	31	4.758210E+07	598232	-52
200	37	7.401009E-06	586399	-56	38	6.561470E+07	656393	-68
250	44	1.277651E-05	750000	-64	45	3.711670E+09	750000	-52
300	51	8.684103E-03	596573	-80	52	5.571157E+12	750000	-88
350	57	8.430897E-09	750000	-60	58	7.625361E+07	512864	-64
400	63	1.052399E-16	750000	-92	64	6.426333E+03	612024	-56
450	70	1.215713E-04	590980	-100	71	1.536203E+18	637433	-72
500	76	4.867934E-03	745697	-80	77	1.906610E+15	608158	-52
550	82	3.275120E-04	504564	-66	83	1.879875E+14	585520	-74
600	87	1.104706E-19	587805	-64	88	2.039995E+01	501945	-64
650	93	4.891966E-29	750000	-118	94	8.857987E+05	504973	-60
700	100	3.179570E-07	518621	-116	101	3.389515E+30	750000	-72
750	105	1.229864E-15	703556	-98	106	3.391140E+00	508824	-110
800	111	1.152960E-03	736475	-92	112	3.037713E+14	617752	-100
850	116	1.990264E-10	547810	-74	117	1.627323E+10	616050	-82
900	121	1.649178E-19	502785	-68	122	3.212594E+03	709532	-72
950	127	5.579835E-09	750000	-76	128	5.877833E+16	547690	-66
1000	132	1.482999E-31	552331	-100	133	4.225397E+02	750000	-80

C. PROBABILISTIC UPPER BOUND FOR THE SGCP

In this section, an algorithm will be described and tested which can be used to generate a probabilistic upper bound for $\overline{\chi}(SG_{n,p})$. The algorithm was presented in a paper by Johri and Matula [*JM82*]. The theory on which it is based was reported in a paper by Matula [*Ma70*].

Given an arbitrary instance of $SG_{n,p}$, let $A_{n,p,j}$ be the random variable which represents the number of independent sets of size j in that instance. Matula states and proves the following theorem.

Theorem 7. Given a random standard graph $SG_{n,p}$, the expected value of the random variable $A_{n,p,j}$ is given by

$$E(A_{n,p,j}) = \binom{n}{j} q^{\binom{j}{2}} \tag{85}$$

where q = 1-p.

Recall that $\alpha(SG_{n,p})$ denotes the number of vertices in the largest maximal independent set of the random standard graph $SG_{n,p}$. Given an arbitrary instance of $SG_{n,p}$, let $\alpha_{n,p}$ designate the random variable which represents the value of $\alpha(SG_{n,p})$. Matula derived the following *bounds* on $\alpha_{n,p}$.

$$Prob\{\alpha_{n,p} \geq j\} \leq E(A_{n,p,j}) = \binom{n}{j} q^{\binom{j}{2}} \tag{86}$$

and

$$Prob\{\alpha_{n,p} \geq j\} \geq \left[\sum_{i=\max\{0,2j-n\}}^{j} \frac{\binom{n-j}{j-i}\binom{j}{i}}{\binom{n}{j}} q^{-\binom{j}{2}} \right]^{-1} \tag{87}$$

TABLE XLVIII

STANDARD GRAPH COLORING
UPPER AND LOWER BOUNDS ON $\Pr\{\alpha_{n,p} \geq j\}$
$p = 0.50$

	n = 250		n = 500	
j	$E(A_{n,p,j})$	$LB_{n,p,j}$	$E(A_{n,p,j})$	$LB_{n,p,j}$
1	250.0000	1.0000	500.0000	1.0000
2	15562.5000	1.0000	62375.0000	1.0000
3	321625.0000	0.9997	2588562.5000	0.9999
4	2482542.9688	0.9988	40203611.3281	0.9997
5	7633819.6289	0.9966	249262390.2344	0.9992
6	9741071.9223	0.9922	642629599.8230	0.9981
7	5305405.2434	0.9838	708613889.0905	0.9962
8	1258997.5334	0.9693	341158835.2750	0.9931
9	132238.4562	0.9442	72851626.2827	0.9881
10	6224.5055	0.8979	6986357.1298	0.9804
11	132.6244	0.7726	303916.4590	0.9686
12	1.2898	0.2180	6047.1659	0.9483
13	0.0058	0.0027	55.4202	0.8520
14	0.0000	0.0000	0.2353	0.0915
15	0.0000	0.0000	0.0005	0.0003
16	0.0000	0.0000	0.0000	0.0000
17	0.0000	0.0000	0.0000	0.0000
18	0.0000	0.0000	0.0000	0.0000
19	0.0000	0.0000	0.0000	0.0000
20	0.0000	0.0000	0.0000	0.0000

Example 24

Let $LB_{n,p,j}$ denote the lower bound on $\Pr\{\alpha_{n,p} \geq j\}$ stated in Equation 87. Table XLVIII displays values of $E(A_{n,p,j})$, which is the upper bound on $\Pr\{\alpha_{n,p} \geq j\}$, and $LB_{n,p,j}$ for $p = 0.50$, $n = 250$ and 500, and $j = 1,2,..,20$. Table XLIX lists the values of $E(A_{n,p,j})$ and $LB_{n,p,j}$ for $p = 0.50$, $n = 750$ and 1000, and $j = 1,2,..,20$. From this information it easily follows that

$$\max\{j \mid E(A_{1000,0.50,j}) \geq 1\} - 15$$

TABLE XLIX

STANDARD GRAPH COLORING
UPPER AND LOWER BOUNDS ON $Pr\{\alpha_{n,p} \geq j\}$
$p = 0.50$

	n = 750		n = 1000	
j	$E(A_{n,p,j})$	$LB_{n,p,j}$	$E(A_{n,p,j})$	$LB_{n,p,j}$
1	750.0000	1.0000	1000.0000	1.0000
2	140437.5000	1.0000	249750.0000	1.0000
3	8753937.5000	1.0000	20770875.0000	1.0000
4	204349728.5156	0.9999	647142574.2188	0.9999
5	1905561218.4082	0.9996	8056925049.0235	0.9998
6	7393974519.3444	0.9992	41753335540.5121	0.9995
7	12279279112.4826	0.9984	92640213230.5112	0.9991
8	8909672246.6548	0.9970	89835675525.2907	0.9984
9	2869347572.4904	0.9950	38679249184.5002	0.9972
10	415270810.7843	0.9919	7486549988.6407	0.9956
11	27281640.6233	0.9874	657997557.5954	0.9932
12	820358.5783	0.9809	26479475.2792	0.9899
13	11369.9037	0.9707	491318.3890	0.9853
14	73.0645	0.9141	4228.2649	0.9776
15	0.2188	0.0997	16.9640	0.8490
16	0.0003	0.0002	0.0319	0.0194
17	0.0000	0.0000	0.0000	0.0000
18	0.0000	0.0000	0.0000	0.0000
19	0.0000	0.0000	0.0000	0.0000
20	0.0000	0.0000	0.0000	0.0000

and

$$Prob\{\alpha_{1000,0.50} = 15\} = Prob\{\alpha_{1000,0.50} \geq 15\} - Prob\{\alpha_{1000,0.50} \geq 16\}$$
$$\geq 0.8490 - 0.0319 - 0.8171$$

In a similar fashion, it can be computed that

$$\max\{j \mid E(A_{750,0.50,j}) \geq 1\} = 14 \text{ and } Prob\{\alpha_{750,0.50} = 13 \vee 14\} \geq 0.7519,$$
$$\max\{j \mid E(A_{500,0.50,j}) \geq 1\} = 13 \text{ and } Prob\{\alpha_{500,0.50} = 12 \vee 13\} \geq 0.7130,$$
$$\max\{j \mid E(A_{250,0.50,j}) \geq 1\} = 12 \text{ and } Prob\{\alpha_{250,0.50} = 11 \vee 12\} \geq 0.7668$$

Thus, at least for n=250,500,750, and 1000 and p=0.50, the size of the largest independent set in a random instance of $SG_{n,p}$ is accurately estimated by computing the maximum value of j such that the expected number of independent sets of size j is at least 1. For p=0.20, computations similar to the preceding give a much smaller bound on the probability that $\alpha_{n,p}$ has most of its density near $\max\{j|E(A_{n,p,j})>=1\}$. But, as Matula emphasizes, the upper and lower bounds stated in Equations 86 and 87 become much weaker as p decreases. Based on numerical data which was presented in that paper, it is apparent that the $\text{Prob}\{\alpha_{n,p}=j\}$, where $j=\max\{j|E(A_{n,p,j})>=1\}$, is substantial.

Johri and Matula used the above theory to develop an algorithm for estimating $\chi(SG_{n,p})$ [*JM82*]. Let $\tilde{\alpha}(n,p)$ denote the above estimate for $\alpha_{n,p}$. Let $\tilde{\chi}(n,p)$ represent an estimate for $\chi(SG_{n,p})$. $\tilde{\chi}(n,p)$ can be determined recursively by using the following equations.

$$\tilde{\chi}(n,p) = 1 + \tilde{\chi}(n - \tilde{\alpha}(n,p),p) \tag{91}$$

where

$$\tilde{\alpha}(n,p) = \max\{j|\binom{n}{j} q^{\binom{j}{2}} \geq 1\} \tag{92}$$

Table L tabulates the values of $\tilde{\alpha}(n,p)$ for p=0.20 and 0.50 and n=1,2,..,1000. Tables LI, LII, and LIII display values of $\tilde{\chi}(n,p)$ for n=10,11,..,1000 and p=0.20, 0.25, and 0.50, respectively. For example, from Table L, if n=1000 and p=0.20, then $\tilde{\alpha}(n,p)=38$. Table LI indicates that $\tilde{\chi}(1000,0.20)=37$ and Table LIII documents that $\tilde{\chi}(1000,0.50)=85$. The implementation of this methodology is included in the Appendix as *Listing 20. MIS_Est.Pas* and *Listing 21. S_UB.Pas*.

Johri and Matula refer to the values of $\tilde{\chi}(n,p)$ as estimates for $\chi(SG_{n,p})$. It will now be argued that a case can be made for considering these values to be *upper bounds* for $\overline{\chi}(SG_{n,p})$, the mean value of $\chi(SG_{n,p})$ for a random sample of $SG_{n,p}$. This argument is based on three observations.

TABLE L

STANDARD GRAPH COLORING PROBABILISTIC ESTIMATES FOR $\alpha_{n,p}$

$\bar{\alpha}(n,p) = \max\{j \mid E(A_{n,p,j}) \geq 1\}$

p=0.20			p=0.50		
n_{LB}	n_{UB}	$\bar{\alpha}(n,p)$	n_{LB}	n_{UB}	$\bar{\alpha}(n,p)$
1	2	1	1	2	1
3	3	2	3	4	2
4	4	3	5	7	3
5	6	4	8	12	4
7	8	5	13	19	5
9	9	6	20	30	6
10	12	7	31	46	7
13	14	8	47	70	8
15	17	9	71	107	9
18	20	10	108	162	10
21	23	11	163	244	11
24	27	12	245	368	12
28	32	13	369	553	13
33	37	14	554	829	14
38	44	15	830	1000	15
45	51	16			
52	59	17			
60	68	18			
69	78	19			
79	91	20			
92	104	21			
105	120	22			
121	139	23			
140	160	24			
161	184	25			
185	211	26			
212	243	27			
244	279	28			
280	320	29			
321	368	30			
369	457	31			
458	483	32			
484	554	33			
555	634	34			
635	726	35			
727	831	36			
832	951	37			
952	1000	38			

TABLE LI

STANDARD GRAPH COLORING
PROBABILISTIC ESTIMATES FOR $\chi(SG_{n,p})$
p=0.20

n_{LB}	n_{UB}	$\tilde{\chi}(n,p)$	n_{LB}	n_{UB}	$\tilde{\chi}(n,p)$
10	13	3	430	461	21
14	25	4	462	494	22
26	40	5	495	527	23
41	57	6	528	561	24
58	76	7	562	595	25
77	97	8	596	629	26
98	119	9	630	664	27
120	143	10	665	699	28
144	168	11	700	735	29
169	194	12	736	771	30
195	221	13	772	807	31
222	249	14	808	844	32
250	277	15	845	881	33
278	306	16	882	918	34
307	336	17	919	956	35
337	366	18	957	994	36
367	397	19	995	1000	37
398	429	20			

1. Table LIV contains a comparison of values $\overline{\chi}(SG_{n,p})$ to corresponding values of $\tilde{\chi}(n,p)$ over the range of values of n for which the Vertex-Sequential with Dynamic Vertex Reordering Algorithm of Chapter III was able to solve exactly. It includes data for both p=0.25 and 0.50. The values of $\tilde{\chi}(n,p)$ were altered slightly from those contained in Tables LII and LIII by applying linear interpolation over the range of values of n for which $\tilde{\chi}(n,p)$ was constant. Notice that in all cases $\overline{\chi}(SG_{n,p}) \leq \tilde{\chi}(n,p)$.

2. Table LV displays a comparison of the Min heuristic, described in the introduction to this chapter, with corresponding values of $\tilde{\chi}(n,p)$ for p=0.25 and n=10,12,..,78,80,90,100,200 and for p=0.50 and n=10,12,..,58,60,70,80,90,100,200. In addition, it contains a copy of the Min-$\overline{\chi}(SG_{n,p})$ columns which were presented in Tables XXIV and XXV. For p=0.25, the Min heuristic generated estimates for

TABLE LII

STANDARD GRAPH COLORING
PROBABILISTIC ESTIMATES FOR $\chi(SG_{n,p})$
p = 0.25

n_{LB}	n_{UB}	$\tilde{\chi}(n,p)$	n_{LB}	n_{UB}	$\tilde{\chi}(n,p)$
10	10	3	407	432	24
11	19	4	433	458	25
20	30	5	459	485	26
31	43	6	486	512	27
44	58	7	513	539	28
59	74	8	540	567	29
75	91	9	568	595	30
92	109	10	596	623	31
110	128	11	624	651	32
129	148	12	652	680	33
149	169	13	681	709	34
170	190	14	710	738	35
191	212	15	739	767	36
213	235	16	768	797	37
236	258	17	798	827	38
259	282	18	828	857	39
283	306	19	858	887	40
307	330	20	888	917	41
331	355	21	918	948	42
356	380	22	949	979	43
381	406	23	980	1000	44

$\overline{\chi}(SG_{n,p})$ which were smaller than $\tilde{\chi}(n,p)$ for all values of n except n = 200. Thus, it is certain that $\overline{\chi}(SG_{n,p}) \leq \tilde{\chi}(n,p)$ for n = 10,12,..,78,80,90, and 100. For n = 200, the Min estimate was 15.44 and $\tilde{\chi}(200,0.25) = 15.41$. But, considering the values of Min-$\overline{\chi}(SG_{n,p})$, it is virtually certain $\overline{\chi}(SG_{200,0.25}) \leq 15.41$. For n = 0.50, the Min heuristic produced estimates for $\overline{\chi}(SG_{n,p})$ which were smaller than $\tilde{\chi}(n,p)$ for all values of n except n = 90,100, and 200. However, again the Min-$\overline{\chi}(SG_{n,p})$ values strongly suggest that $\overline{\chi}(SG_{n,p}) \leq \tilde{\chi}(n,p)$ for these values as well.

3. Johnson, Aragon, McGeoch, and Schevon designed an extended searching heuristic, named XRLF, for estimating the chromatic number of a random standard graph [*JA89*]. This algorithm was based on ideas suggested by Johri and Matula, augmented with a final exact coloring phase using the exact Vertex-Sequential with

TABLE LIII

STANDARD GRAPH COLORING
PROBABILISTIC ESTIMATES FOR $\chi(SG_{n,p})$
$p = 0.50$

n_{LB}	n_{UB}	$\bar{\chi}(n,p)$	n_{LB}	n_{UB}	$\bar{\chi}(n,p)$
10	10	4	424	436	45
11	15	5	437	449	46
16	21	6	450	462	47
22	27	7	463	475	48
28	34	8	476	488	49
35	41	9	489	501	50
42	49	10	502	514	51
50	57	11	515	527	52
58	65	12	528	540	53
66	74	13	541	554	54
75	83	14	555	568	55
84	92	15	569	582	56
93	101	16	583	596	57
102	111	17	597	610	58
112	121	18	611	624	59
122	131	19	625	638	60
132	141	20	639	652	61
142	151	21	653	666	62
152	161	22	667	680	63
162	172	23	681	694	64
173	183	24	695	708	65
184	194	25	709	722	66
195	205	26	723	736	67
206	216	27	737	750	68
217	227	28	751	764	69
228	238	29	765	778	70
239	250	30	779	792	71
251	262	31	793	806	72
263	274	32	807	820	73
275	286	33	821	835	74
287	298	34	836	850	75
299	310	35	851	865	76
311	322	36	866	880	77
323	334	37	881	895	78
335	346	38	896	910	79
347	358	39	911	925	80
359	371	40	926	940	81
372	384	41	941	955	82
385	397	42	956	970	83
398	410	43	971	985	84
411	423	44	986	1000	85

TABLE LIV

STANDARD GRAPH COLORING
COMPARISON OF $\overline{\chi}(SG_{n,p})$ AND $\bar{\chi}(n,p)$

n	p = 0.25		p = 0.50	
	$\overline{\chi}(SG_{n,p})$	$\bar{\chi}(n,p)$	$\overline{\chi}(SG_{n,p})$	$\bar{\chi}(n,p)$
10	2.88	3.83	3.96	4.75
12	3.04	4.11	4.36	5.20
14	3.24	4.33	4.68	5.60
16	3.48	4.56	5.00	6.00
18	3.56	4.78	5.40	6.33
20	3.84	5.00	5.60	6.67
22	4.04	5.18	6.04	7.00
24	4.04	5.36	6.40	7.33
26	4.08	5.55	6.44	7.67
28	4.24	5.73	6.88	8.00
30	4.56	5.91	6.96	8.29
32	4.68	6.08	7.24	8.57
34	4.96	6.23	7.40	8.86
36	5.00	6.38	7.88	9.14
38	5.00	6.54	8.04	9.43
40	5.04	6.69	8.08	9.71
42	5.20	6.85	8.72	10.00
44	5.28	7.00	8.92	10.25
46	5.44	7.13	9.04	10.50
48	5.76	7.27	9.28	10.75
50	6.00	7.40	9.56	11.00
52	5.96	7.53	9.88	11.25
54	6.00	7.67	9.92	11.50
56	6.04	7.80	10.04	11.75
58	6.08	7.93	10.20	12.00
60	6.12	8.06		
62	6.32	8.19		
64	6.60	8.31		
66	6.84	8.44		
68	7.00	8.56		
70	7.00	8.69		
72	7.00	8.81		
74	7.00	8.94		
76	7.08	9.06		
78	7.12	9.18		

TABLE LV

STANDARD GRAPH COLORING
COMPARISON OF $\tilde{\chi}(n,p)$ TO THE MIN HEURISTIC

n	p=0.25 Min	p=0.25 $\tilde{\chi}(n,p)$	p=0.25 Min-$\overline{\chi}(SG_{n,p})$	p=0.50 Min	p=0.50 $\tilde{\chi}(n,p)$	p=0.50 Min-$\overline{\chi}(SG_{n,p})$
10	2.88	3.83	0.00	3.96	4.75	0.00
12	3.04	4.11	0.00	4.36	5.20	0.00
14	3.24	4.33	0.00	4.68	5.60	0.00
16	3.48	4.56	0.00	5.00	6.00	0.00
18	3.56	4.78	0.00	5.40	6.33	0.00
20	3.84	5.00	0.00	5.60	6.67	0.00
22	4.04	5.18	0.00	6.04	7.00	0.00
24	4.04	5.36	0.00	6.48	7.33	0.08
26	4.12	5.55	0.04	6.56	7.67	0.12
28	4.32	5.73	0.08	7.08	8.00	0.20
30	4.60	5.91	0.04	7.28	8.29	0.32
32	4.76	6.08	0.08	7.44	8.57	0.20
34	4.92	6.23	0.00	7.84	8.86	0.44
36	5.00	6.38	0.00	8.04	9.14	0.16
38	5.08	6.54	0.08	8.48	9.43	0.44
40	5.20	6.69	0.16	8.68	9.71	0.60
42	5.40	6.85	0.20	9.08	10.00	0.36
44	5.64	7.00	0.36	9.24	10.25	0.32
46	5.92	7.13	0.48	9.68	10.50	0.64
48	5.88	7.27	0.12	9.80	10.75	0.52
50	6.04	7.40	0.04	10.04	11.00	0.48
52	6.04	7.53	0.08	10.40	11.25	0.52
54	6.32	7.67	0.32	10.76	11.50	0.84
56	6.40	7.80	0.36	10.92	11.75	0.88
58	6.64	7.93	0.56	11.36	12.00	1.16
60	6.92	8.06	0.80	11.80	12.25	
62	6.96	8.19	0.64			
64	7.04	8.31	0.44			
66	7.04	8.44	0.20			
68	7.20	8.56	0.20			
70	7.56	8.69	0.56	12.96	13.44	
72	7.72	8.81	0.72			
74	7.84	8.94	0.84			
76	8.00	9.06	0.92			
78	8.00	9.18	0.88			
80	8.08	9.29		14.44	14.56	
90	8.96	9.88		15.68	15.67	
100	9.40	10.44		16.84	16.78	
200	15.44	15.41		28.92	26.45	

Dynamic Vertex Reordering Algorithm when the set of remaining vertices became sufficiently small. After 6.4 hours of running time, XRLF found a 17 coloring for a random standard graph with $n=125$ and $p=0.50$. Note that $\tilde{\chi}(125,0.50)=19.3>17$. Using 2.2 hours, it was able to locate a feasible 29 coloring for a graph with $n=250$ and $p=0.50$. Table LIII indicates that $\tilde{\chi}(250,0.50)=30.92>29$. For a sample graph with $n=500$ and $p=0.50$, it found a 49 coloring after 73.8 hours of computing time. $\tilde{\chi}(500,0.50)=50.85>49$. Finally, with $n=1000$ and $p=0.50$, it found an 86 coloring after 68.3 hours of searching. $\tilde{\chi}(1000,0.50)=85.0$. Since the distribution of the random variable $\chi(SG_{n,p})$ is known to be highly peaked around its mean [*Ma70*], in each of the above cases, it is very probable that $\overline{\chi}(SG_{n,p})\leq\tilde{\chi}(n,p)$.

Based on the previous three observations, it was concluded that, at least for all values of n between 10 and 1000, it is almost certain that $\tilde{\chi}(n,p)$ is an upper bound for $\overline{\chi}(SG_{n,p})$.

D. PROBABILISTIC UPPER BOUND FOR THE CGCP

In order to use the concepts described in the previous section as a basis for developing a method of calculating probabilistic upper bounds for $\overline{\chi}(CG_{n,p})$, two problems must be solved.

The Johri and Matula algorithm is a color-sequential simulation. It estimates the size of the largest independent set, assumes such a set is identified, assigns the same color to all vertices in that set, and then recursively repeats this procedure on the subgraph induced by removing all those vertices from the graph. As with the color-sequential algorithm for the CGCP described in Chapter III, any attempt to extend the Johri and Matula method would have to take into consideration the chromaticities of the vertices in the independent set being colored. Any vertex whose chromaticity count had not been satisfied could not be eliminated from the induced subgraph. Also, it would be necessary that such a vertex be an element in the independent set which was subsequently selected for coloring. Thus, one problem is how to calculate an estimate of the size of the largest independent set in a random instance of $CG_{n,p}$ which contains some specified independent subset.

The second problem is related to the first. Processing the independent sets requires a mathematically sound methodology for identifying vertex chromaticities and, hence, those vertices which must be left in the induced subgraph.

Let $\alpha_{n,p,r}$ denote the random variable which represents the size of the largest independent set in $CG_{n,p}$ which contains some given independent subset of size r. This known independent subset will be called a *residue*. Also, let $A_{n,p,j,r}$ designate the number of independent sets of size j that contain a particular residue of size r.

Theorem 8. Given a random composite graph $CG_{n,p}$, the expected value of the random variable $A_{n,p,j,r}$ is given by

$$E(A_{n,p,j,r}) = \binom{n-r}{j-r} q^{\binom{j}{2}-\binom{r}{2}} \tag{93}$$

Proof. Consider an arbitrary set of j vertices which contains some specified residue of size r, where $j \geq r$. In order for this set to be an independent set, none of the possible ${}_jC_2$ edges between vertices in this set can exist. Since r of the vertices are known to be independent, this accounts for ${}_rC_2$ of these edges. This leaves ${}_jC_2$-${}_rC_2$ edges which could exist. The probability that they are not present is given by

$$q^{\binom{j}{2}-\binom{r}{2}} \tag{94}$$

Since there are ${}_{n-r}C_{j-r}$ such sets, Equation 93 follows.

Thus, if $\tilde{\alpha}(n,p|r)$ represents an estimate for $\alpha_{n,p,r}$, then $\tilde{\alpha}(n,p|r)$ can be obtained by evaluating the following formula.

$$\tilde{\alpha}(n,p|r) = \max\left\{j \,\middle|\, \binom{n-r}{j-r} q^{\binom{j}{2}-\binom{r}{2}} \geq 1\right\} \tag{95}$$

TABLE LVI

COMPOSITE GRAPH COLORING
PROBABILISTIC ESTIMATES FOR $\alpha_{n,p,r}$
$\tilde{\alpha}(n,p|r) = \max\{j|E(A_{n,p,j,r}) \geq 1\}$
$p = 0.50$

$\tilde{\alpha}(n,p\|r)$	r=0 n Range		r=1 n Range		r=2 n Range		r=3 n Range		r=4 n Range	
1	1	2	1	2						
2	3	4	3	5	2	5				
3	5	7	6	9	6	10	3	10		
4	8	12	10	15	11	17	11	19	4	19
5	13	19	16	23	18	28	20	33	20	36
6	20	30	24	37	29	45	34	54	37	63
7	31	46	38	58	46	72	55	88	64	105
8	47	70	59	89	73	113	89	140	106	172
9	71	107	90	137	114	175	141	222	173	277
10	108	162	138	210	176	271	223	347	278	439
11	163	244	211	320	272	416	348	537	440	688
12	245	368	321	485	417	635	538	827	689	1000
13	369	553	486	732	636	964	828	1000		
14	554	829	733	1000	965	1000				
15	830	1000								

Table LVI illustrates the results of calculating values of $\tilde{\alpha}(n,p|r)$ for $p=0.50$, $n=1,2,..,1000$ and $r=0,1,2,3,4$.

The formula for calculating $\tilde{\alpha}(n,p|r)$ solves the first problem which was described above. The basis for an algorithm for estimating the values of $\overline{\chi}(CG_{n,p})$ can now be specified in the form of the following recursive definition.

$$\tilde{\chi}(n,p) = 1 + \tilde{\chi}(n - \tilde{\alpha}(n,p|r),p) \tag{96}$$

where r is the *residue* of vertices whose chromaticity have not yet been satisfied.

Initially, $r=0$. But, establishing succeeding values of r is exactly the second problem which was initially posed. It was decided that a sound solution to this problem would be to use a random number generator along with the definition of the Truncated Poisson probability distribution to simulate the assignment a vertex

chromaticity to each of the $\tilde{\alpha}(n,p|r)$-r new vertices. These are the vertices which have not been partially colored. Initially, $\tilde{\alpha}(n,p|0)$ vertex chromaticities are generated. All chromaticities larger than 1 identify vertices which have to be a part of the residue of succeeding maximal independent sets. With some simple bookkeeping, this approach can be easily integrated into the implementation of Equation 96. Since the vertex chromaticities are random variables, the evaluation of $\tilde{\chi}(n,p)$ is repeated 1000 times and $\tilde{\chi}(n,p)$ is assigned the value of the mean of these repetitions. This value of $\tilde{\chi}(n,p)$ is the proposed *upper bound* for $\overline{\chi}(CG_{n,p})$.

The implementation of this methodology is included in the Appendix as *Listing 22. CTP_UB.Pas.* Table LVII contains a comparison of the values $\overline{\chi}(CG_{n,p})$ and corresponding values of $\tilde{\chi}(n,p)$ over the range of values of n for which the Vertex-Sequential with Dynamic Vertex Reordering algorithm of Chapter III was able to solve exactly. It includes data for both p=0.25 and 0.50. In all cases $\overline{\chi}(CG_{n,p}) \leq \tilde{\chi}(n,p)$. Note also that the table documents the average vertex chromaticity which was produced by the random number generator during the process of calculating the values of $\tilde{\chi}(n,p)$. Also, recall that those values of $\overline{\chi}(CG_{n,p})$ that appear in italics are actually projections from corresponding values of $\overline{\chi}(SG_{n,p})$.

Table LVIII displays a comparison of the mean values produced by the Min heuristic with corresponding values of $\tilde{\chi}(n,p)$ for p=0.25 and n=10,12,..,78,80,90,100, 200 and for p=0.50 and n=10,12,..,58,60,70,80,90,100,200. In addition, it contains a copy of the Min-$\overline{\chi}(CG_{n,p})$ columns which were presented in Tables XXIV and XXV. For p=0.25, the Min heuristic generated estimates for $\overline{\chi}(CG_{n,p})$ which were smaller than $\tilde{\chi}(n,p)$ for all values of n except n=100 and 200. Thus, it is certain that $\overline{\chi}(CG_{n,p}) \leq \tilde{\chi}(n,p)$ for n=10,12,..,78,80, and 90. For n=100, the Min estimate was 17.40 and $\tilde{\chi}(100,0.25)=17.16$. But, considering the values of Min-$\overline{\chi}(CG_{n,p})$, it is virtually certain $\overline{\chi}(CG_{100,0.25}) \leq 17.16$. The Min estimate for n=200 was 28.12 and $\tilde{\chi}(200,0.25)=26.25$. Again, the values of Min-$\overline{\chi}(CG_{n,p})$ strongly indicate that $\overline{\chi}(CG_{200,0.25}) \leq 26.25$. For p=0.50, the Min heuristic produced estimates for $\overline{\chi}(CG_{n,p})$ which were smaller than $\tilde{\chi}(n,p)$ for all values of n except n=90,100, and 200. However, the Min-$\overline{\chi}(CG_{n,p})$ values obviously support the conclusion that

TABLE LVII

COMPOSITE GRAPH COLORING
COMPARISON OF $\bar{\chi}(CG_{n,p})$ AND $\bar{\chi}(n,p)$

n	p=0.25 $\bar{\chi}(CG_{n,p})$	p=0.25 $\bar{\chi}(n,p)$	p=0.25 $\bar{c}(v)$	p=0.50 $\bar{\chi}(CG_{n,p})$	p=0.50 $\bar{\chi}(n,p)$	p=0.50 $\bar{c}(v)$
10	5.12	5.66	1.5916	7.04	7.26	1.5916
12	5.12	6.00	1.5773	7.44	8.12	1.5773
14	6.12	6.55	1.5730	8.20	8.56	1.5730
16	6.60	6.86	1.5821	9.40	9.43	1.5821
18	6.80	7.15	1.5802	9.48	10.16	1.5802
20	6.92	7.58	1.5830	9.92	10.66	1.5830
22	7.12	7.83	1.5734	10.48	11.18	1.5734
24	7.60	8.20	1.5811	11.32	11.85	1.5811
26	7.92	8.47	1.5786	11.72	12.47	1.5786
28	7.80	8.77	1.5768	12.08	13.07	1.5768
30	8.20	9.20	1.5838	12.32	13.73	1.5838
32	8.64	9.32	1.5840	13.04	14.03	1.5840
34	8.96	9.67	1.5834	*13.17*	14.65	1.5834
36	8.76	9.93	1.5753	*14.03*	15.14	1.5753
38	8.80	10.30	1.5828	*14.31*	15.71	1.5828
40	9.28	10.46	1.5835	*14.38*	16.19	1.5835
42	9.64	10.75	1.5790	*15.52*	16.73	1.5790
44	9.68	11.08	1.5820	*15.88*	17.25	1.5820
46	*9.99*	11.19	1.5844	*16.09*	17.72	1.5844
48	*10.58*	11.44	1.5812	*16.52*	18.04	1.5812
50	*11.02*	11.75	1.5842	*17.02*	18.59	1.5842
52	*10.94*	11.99	1.5797	*17.59*	19.07	1.5797
54	*11.02*	12.34	1.5882	*17.66*	19.59	1.5882
56	*11.09*	12.34	1.5811	*17.87*	19.92	1.5811
58	*11.17*	12.63	1.5795	*18.16*	20.44	1.5795
60	*11.24*	12.92	1.5840			
62	*11.61*	13.18	1.5831			
64	*12.12*	13.38	1.5824			
66	*12.56*	13.63	1.5795			
68	*12.85*	13.79	1.5845			
70	*12.85*	13.96	1.5785			
72	*12.85*	14.26	1.5819			
74	*12.85*	14.53	1.5812			
76	*13.00*	14.78	1.5819			
78	*13.08*	14.99	1.5812			

TABLE LVIII

COMPOSITE GRAPH COLORING
COMPARISON OF $\bar{\chi}(n,p)$ TO THE MIN HEURISTIC

	p = 0.25			p = 0.50		
n	Min	$\bar{\chi}(n,p)$	Min-$\bar{\chi}(CG_{n,p})$	Min	$\bar{\chi}(n,p)$	Min-$\bar{\chi}(CG_{n,p})$
10	5.12	5.66	0.00	7.08	7.26	0.04
12	5.12	6.00	0.00	7.48	8.12	0.04
14	6.12	6.55	0.00	8.24	8.56	0.04
16	6.60	6.86	0.00	9.52	9.43	0.12
18	6.80	7.15	0.00	9.52	10.16	0.04
20	7.00	7.58	0.08	10.28	10.66	0.36
22	7.20	7.83	0.08	10.56	11.18	0.08
24	7.88	8.20	0.28	11.64	11.85	0.32
26	8.04	8.47	0.12	11.96	12.47	0.24
28	8.04	8.77	0.24	12.48	13.07	0.40
30	8.64	9.20	0.44	13.36	13.73	1.04
32	9.16	9.32	0.52	13.64	14.03	0.60
34	9.32	9.67	0.36	14.48	14.65	1.31
36	9.08	9.93	0.32	14.12	15.14	0.09
38	9.28	10.30	0.48	15.04	15.71	0.73
40	9.64	10.46	0.36	15.56	16.19	1.18
42	10.36	10.75	0.72	16.36	16.73	0.84
44	10.68	11.08	1.00	17.08	17.25	1.20
46	10.80	11.19	0.81	17.52	17.72	1.43
48	11.12	11.44	0.54	17.36	18.04	0.84
50	11.32	11.75	0.30	18.04	18.59	1.02
52	11.68	11.99	0.74	18.68	19.07	1.09
54	11.44	12.34	0.42	18.98	19.59	1.32
56	12.08	12.34	0.99	19.44	19.92	1.57
58	11.96	12.63	0.79	19.72	20.44	1.56
60	12.32	12.92	1.08	20.12	20.95	
62	12.80	13.18	1.19			
64	13.36	13.38	1.24			
66	13.56	13.63	1.00			
68	13.80	13.79	0.95			
70	13.72	13.96	0.87	22.92	23.16	
72	13.96	14.26	1.11			
74	14.04	14.53	1.19			
76	14.24	14.78	1.24			
78	14.84	14.99	1.76			
80	14.96	15.13		25.08	25.18	
90	15.96	16.17		27.44	27.32	
100	17.40	17.16		29.68	29.29	
200	28.12	26.25		51.32	47.49	

TABLE LIX

COMPOSITE GRAPH COLORING
PROBABILISTIC UPPER BOUNDS FOR $\bar{\chi}(CG_{n,p})$
p=0.20

n	$\bar{\chi}(n,p)$	$\bar{c}(v)$	n	$\bar{\chi}(n,p)$	$\bar{c}(v)$
50	10.47	1.5842	550	43.73	1.5802
100	14.99	1.5800	600	46.51	1.5812
150	18.84	1.5798	650	49.26	1.5829
200	22.40	1.5806	700	52.02	1.5840
250	25.88	1.5810	750	54.65	1.5837
300	29.10	1.5820	800	57.27	1.5823
350	32.11	1.5831	850	59.77	1.5818
400	35.16	1.5821	900	62.33	1.5819
450	38.12	1.5820	950	64.94	1.5817
500	41.01	1.5819	1000	67.39	1.5815

TABLE LX

COMPOSITE GRAPH COLORING
PROBABILISTIC UPPER BOUNDS FOR $\bar{\chi}(CG_{n,p})$
p=0.25

n	$\bar{\chi}(n,p)$	$\bar{c}(v)$	n	$\bar{\chi}(n,p)$	$\bar{c}(v)$
50	11.75	1.5842	550	52.43	1.5814
100	17.16	1.5800	600	55.92	1.5819
150	21.88	1.5804	650	59.22	1.5808
200	26.25	1.5806	700	62.52	1.5827
250	30.46	1.5831	750	65.91	1.5836
300	34.37	1.5819	800	69.03	1.5819
350	38.18	1.5820	850	72.17	1.5815
400	41.85	1.5804	900	75.39	1.5812
450	45.51	1.5822	950	78.53	1.5817
500	49.04	1.5839	1000	81.62	1.5816

TABLE LXI

COMPOSITE GRAPH COLORING
PROBABILISTIC UPPER BOUNDS FOR $\overline{\chi}(CG_{n,p})$
p=0.50

n	$\tilde{\chi}(n,p)$	$\overline{c}(v)$	n	$\tilde{\chi}(n,p)$	$\overline{c}(v)$
50	18.59	1.5842	550	101.33	1.5802
100	29.29	1.5791	600	108.40	1.5812
150	38.74	1.5798	650	115.56	1.5829
200	47.49	1.5806	700	122.65	1.5840
250	55.94	1.5812	750	129.39	1.5837
300	64.09	1.5820	800	136.06	1.5823
350	71.95	1.5831	850	142.64	1.5818
400	79.46	1.5821	900	149.24	1.5819
450	87.01	1.5820	950	155.78	1.5822
500	94.31	1.5819	1000	162.34	1.5815

$\overline{\chi}(CG_{n,p}) \leq \tilde{\chi}(n,p)$ for these values as well.

Based on these observations, as well as, the repeated demonstration of the fact that the CGCP has properties that are very similar to the SGCP, it was concluded that $\tilde{\chi}(n,p)$ is a probabilistic upper bound for $\overline{\chi}(CG_{n,p})$. Tables LIX, LX, and LXI document the values of $\tilde{\chi}(n,p)$ and $\overline{c}(v)$ for p=0.20,0.25 and 0.50, respectively, and n=50,100,..,1000.

E. EVALUATION OF THE HEURISTICS USING PROBABILISTIC BOUNDS

The upper and lower bounds which were developed in the preceding four sections will now be used as a basis for evaluating the actual effectiveness of some of the heuristic algorithms presented in Chapter IV.

1. Evaluation of SGCP Heuristic Algorithms.

First, consider the SGCP. Tables LXII and LXIII display the values of the probabilistic lower bound $k_{n,p}+1$, the probabilistic upper bound $\tilde{\chi}(n,p)$, and the estimates for $\overline{\chi}(SG_{n,p})$ that were generated by the heuristic algorithms SRLF, SDsatur, SLF, and SSL, for n=50,100,..,1000 and p=0.20 and 0.50, respectively.

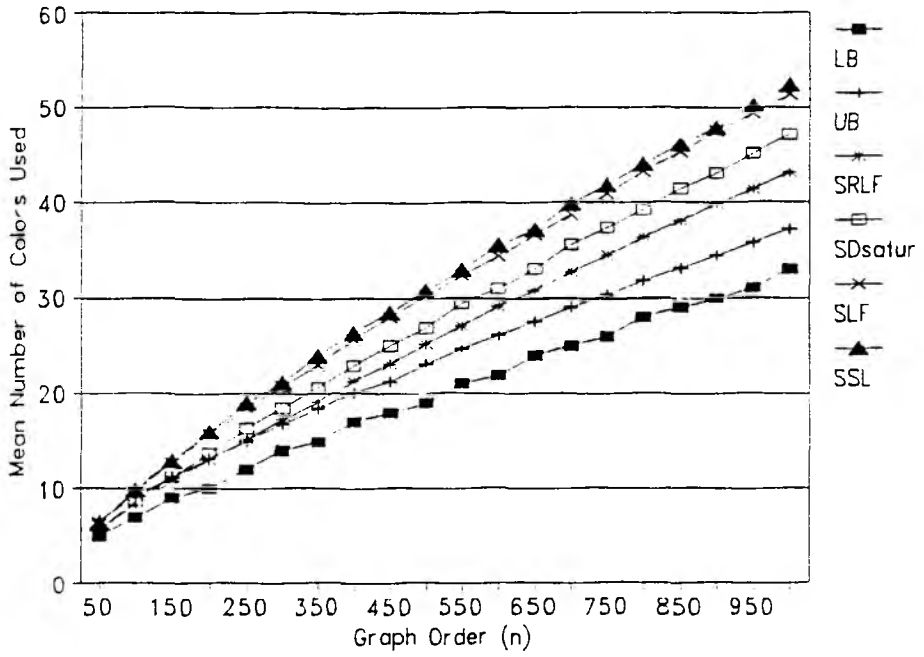


Figure 33. SGCP Heuristic Algorithm Evaluation (p=0.20)

TABLE LXII

STANDARD GRAPH COLORING
HEURISTIC ESTIMATES VERSUS PROBABILISTIC BOUNDS
p=0.20

n	$k_{a,p}+1$	$\bar{\chi}(n,p)$	SRLF	SDsatur	SLF	SSL
50	5.0	6.6	5.6	5.7	6.5	6.4
100	7.0	9.1	8.3	8.7	9.7	10.0
150	9.0	11.2	10.8	11.2	12.7	12.9
200	10.0	13.2	13.1	13.8	15.9	16.0
250	12.0	15.0	15.1	16.3	18.4	19.0
300	14.0	16.8	17.2	18.5	20.8	21.2
350	15.0	18.4	19.2	20.6	23.1	24.0
400	17.0	20.1	21.2	22.9	25.6	26.3
450	18.0	21.3	23.2	25.0	28.0	28.5
500	19.0	23.2	25.2	26.9	30.2	30.8
550	21.0	24.7	27.0	29.4	32.3	32.9
600	22.0	26.1	29.1	31.0	34.4	35.6
650	24.0	27.6	30.8	33.1	36.7	37.2
700	25.0	29.0	32.7	35.6	38.6	39.8
750	26.0	30.4	34.5	37.4	40.9	41.8
800	28.0	31.8	36.3	39.2	43.2	44.0
850	29.0	33.1	38.0	41.4	45.2	46.0
900	30.0	34.5	39.9	43.1	47.5	47.8
950	31.0	35.8	41.3	45.1	49.3	50.2
1000	33.0	37.1	43.1	47.1	51.3	52.4

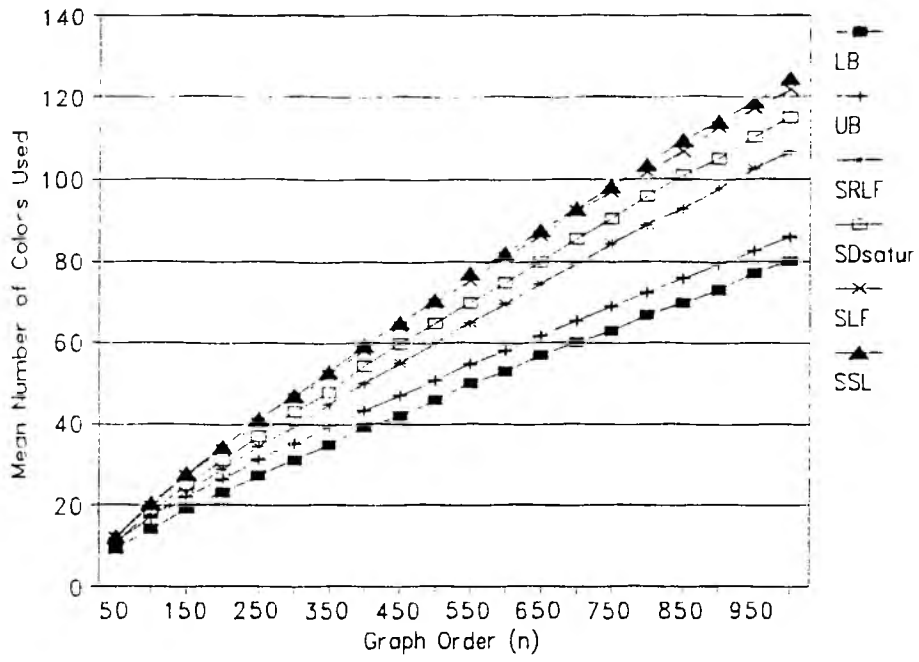


Figure 34. SGCP Heuristic Algorithm Evaluation (p=0.50)

TABLE LXIII

STANDARD GRAPH COLORING
HEURISTIC ESTIMATES VERSUS PROBABILISTIC BOUNDS
p=0.50

n	$k_{n,p}+1$	$\bar{\chi}(n,p)$	SRLF	SDsatur	SLF	SSL
50	9.0	11.0	10.4	10.8	12.0	12.2
100	14.0	16.8	17.1	18.4	19.7	20.3
150	19.0	21.8	23.2	24.7	27.1	27.6
200	23.0	26.5	29.0	31.0	33.7	34.3
250	27.0	30.9	34.4	37.0	40.1	41.2
300	31.0	35.1	39.4	43.0	46.4	47.1
350	35.0	39.3	45.0	48.0	52.2	53.1
400	39.0	43.2	50.1	54.2	58.2	59.3
450	42.0	47.0	55.1	59.6	64.1	65.1
500	46.0	50.9	60.1	64.8	69.8	70.6
550	50.0	54.6	64.8	69.8	75.1	77.0
600	53.0	58.2	69.6	74.9	81.1	82.2
650	57.0	61.8	74.6	80.1	86.2	87.6
700	60.0	65.4	79.3	85.6	92.3	92.8
750	63.0	68.9	84.3	90.7	97.1	98.5
800	67.0	72.5	88.9	95.9	101.7	103.5
850	70.0	75.9	93.0	100.9	106.8	109.4
900	73.0	79.3	97.7	105.2	112.8	114.0
950	77.0	82.6	102.5	110.5	117.1	119.0
1000	80.0	85.9	106.8	115.0	122.0	124.7

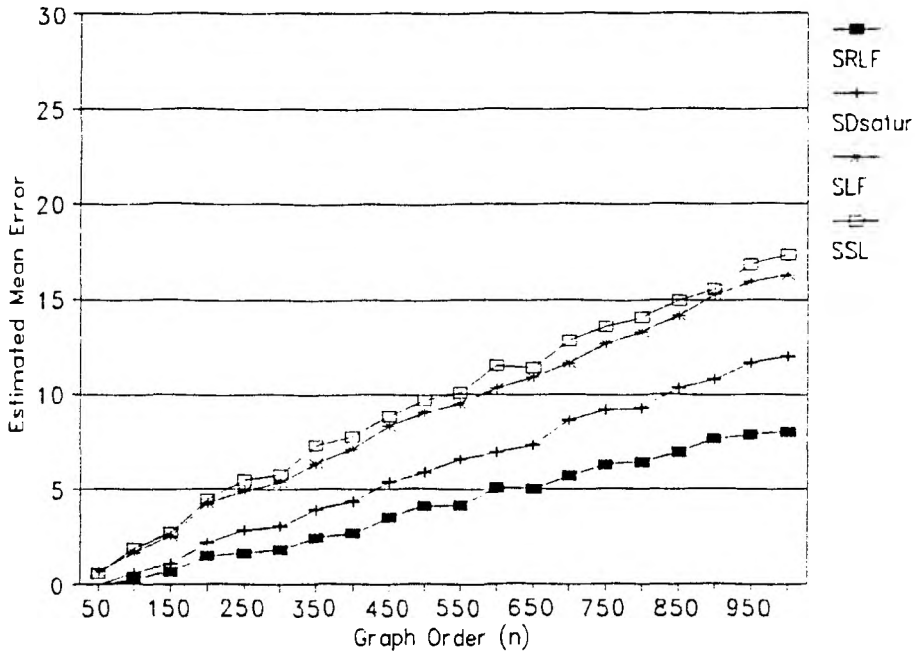


Figure 35. SGCP Heuristic Algorithm Evaluation (p=0.20)

TABLE LXIV

STANDARD GRAPH COLORING
ESTIMATED MEAN ERRORS OF HEURISTIC ALGORITHMS
p=0.20

n	SRLF	SDsatur	SLF	SSL
50	-0.2	-0.1	0.7	0.5
100	0.3	0.6	1.6	1.9
150	0.7	1.1	2.6	2.8
200	1.5	2.2	4.3	4.4
250	1.6	2.8	4.9	5.5
300	1.9	3.1	5.4	5.8
350	2.4	3.9	6.4	7.3
400	2.7	4.4	7.1	7.8
450	3.5	5.4	8.3	8.8
500	4.1	5.8	9.1	9.8
550	4.2	6.6	9.5	10.1
600	5.0	6.9	10.3	11.5
650	5.0	7.3	10.9	11.4
700	5.7	8.6	11.6	12.8
750	6.3	9.2	12.7	13.6
800	6.4	9.3	13.3	14.1
850	6.9	10.3	14.1	14.9
900	7.7	10.9	15.3	15.6
950	7.9	11.7	15.9	16.8
1000	8.0	12.0	16.2	17.3

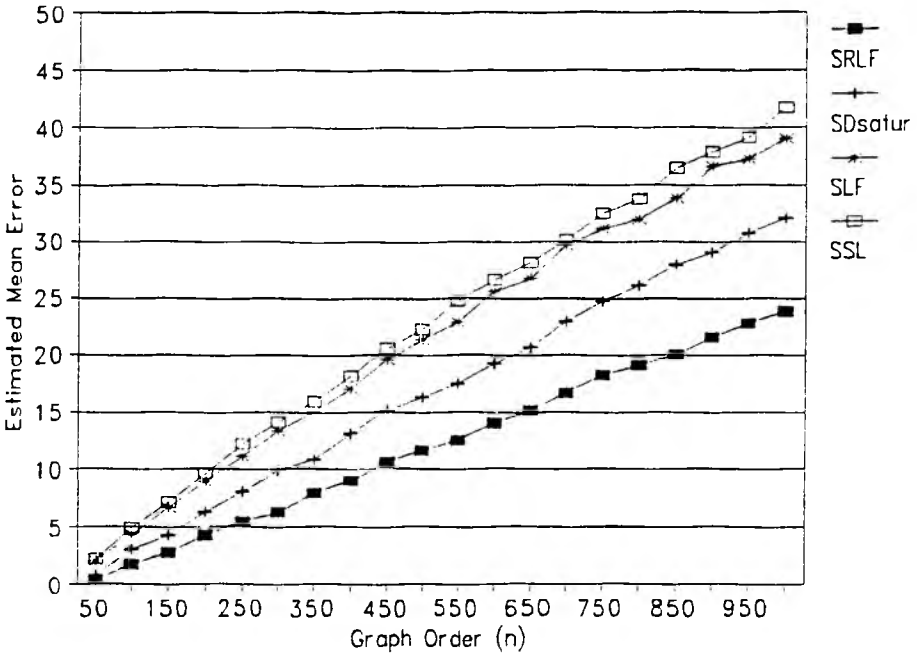


Figure 36. SGCP Heuristic Algorithm Evaluation (p=0.50)

TABLE LXV

STANDARD GRAPH COLORING
ESTIMATED MEAN ERRORS OF HEURISTIC ALGORITHMS
p=0.50

n	SRLF	SDsatur	SLF	SSL
50	0.4	0.8	2.0	2.2
100	1.7	3.0	4.3	4.9
150	2.8	4.3	6.7	7.2
200	4.2	6.3	9.0	9.6
250	5.5	8.0	11.2	12.2
300	6.3	9.9	13.3	14.1
350	7.9	10.8	15.1	16.0
400	9.0	13.1	17.1	18.2
450	10.6	15.1	19.6	20.6
500	11.7	16.3	21.4	22.2
550	12.5	17.5	22.8	24.7
600	14.0	19.3	25.5	26.6
650	15.2	20.7	26.8	28.2
700	16.6	22.9	29.6	30.1
750	18.3	24.7	31.1	32.5
800	19.2	26.2	32.0	33.8
850	20.0	27.9	33.8	36.4
900	21.6	29.1	36.7	37.9
950	22.7	30.7	37.3	39.2
1000	23.8	32.0	39.0	41.7

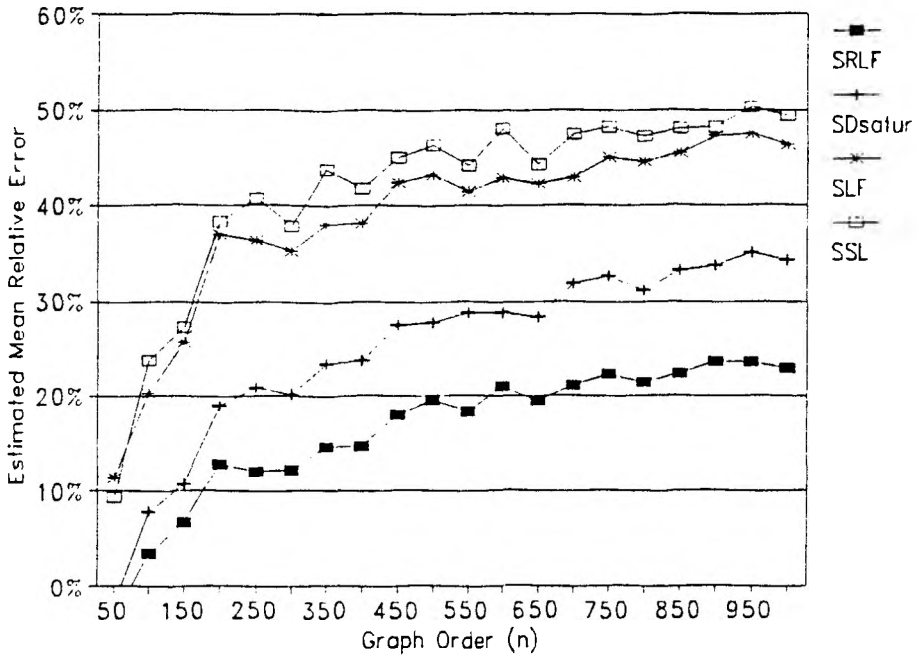


Figure 37. SGCP Heuristic Algorithm Evaluation (p=0.20)

TABLE LXVI

STANDARD GRAPH COLORING
ESTIMATED MEAN RELATIVE ERRORS
p=0.20

n	SRLF	SDsatur	SLF	SSL
50	-3.7%	-2.3%	11.4%	9.4%
100	3.4%	7.9%	20.3%	23.8%
150	6.7%	10.7%	25.7%	27.3%
200	12.8%	19.0%	37.0%	38.3%
250	12.0%	20.9%	36.3%	40.7%
300	12.1%	20.2%	35.2%	37.8%
350	14.6%	23.5%	38.1%	43.8%
400	14.6%	23.7%	38.2%	41.8%
450	18.0%	27.6%	42.4%	45.1%
500	19.6%	27.7%	43.3%	46.3%
550	18.3%	28.8%	41.5%	44.1%
600	20.9%	28.8%	43.0%	48.0%
650	19.4%	28.4%	42.3%	44.3%
700	21.1%	31.9%	43.0%	47.4%
750	22.4%	32.6%	45.1%	48.3%
800	21.4%	31.1%	44.5%	47.2%
850	22.3%	33.2%	45.5%	48.1%
900	23.7%	33.7%	47.3%	48.2%
950	23.6%	35.0%	47.6%	50.3%
1000	22.9%	34.3%	46.3%	49.4%

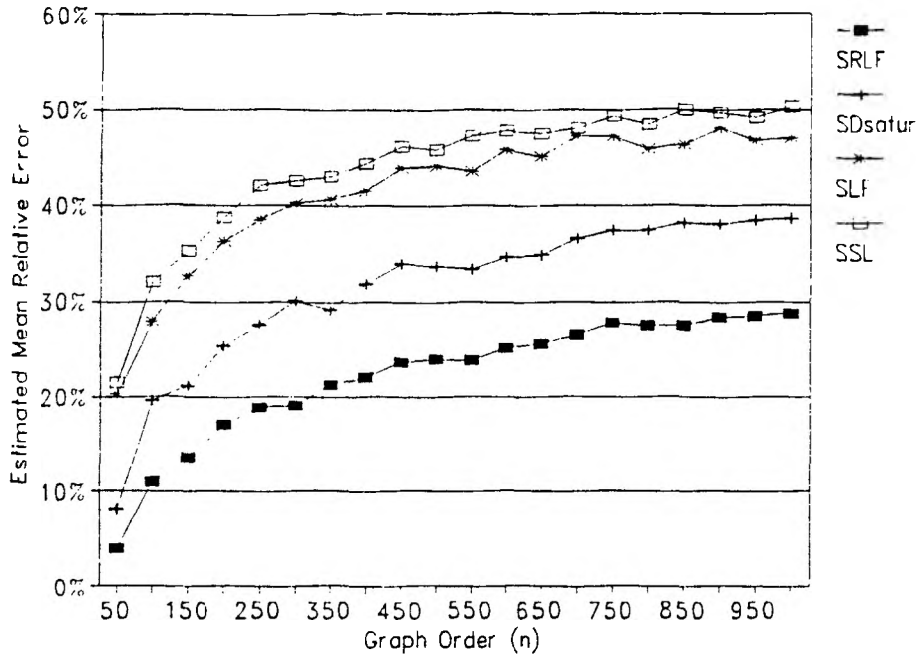


Figure 38. SGCP Heuristic Algorithm Evaluation (p=0.50)

TABLE LXVII

STANDARD GRAPH COLORING
ESTIMATED MEAN RELATIVE ERRORS
p=0.50

n	SRLF	SDsatur	SLF	SSL
50	4.0%	8.0%	20.4%	21.6%
100	11.0%	19.6%	27.9%	32.0%
150	13.5%	21.2%	32.7%	35.3%
200	17.1%	25.4%	36.2%	38.8%
250	18.9%	27.6%	38.5%	42.1%
300	19.1%	30.0%	40.3%	42.6%
350	21.3%	29.2%	40.7%	43.1%
400	22.0%	31.9%	41.6%	44.3%
450	23.8%	34.0%	44.0%	46.2%
500	24.1%	33.7%	44.1%	45.9%
550	23.9%	33.4%	43.5%	47.2%
600	25.2%	34.7%	45.9%	47.8%
650	25.6%	34.9%	45.1%	47.5%
700	26.5%	36.6%	47.3%	48.1%
750	27.8%	37.5%	47.2%	49.3%
800	27.5%	37.5%	45.8%	48.4%
850	27.5%	38.3%	46.4%	49.9%
900	28.3%	38.2%	48.2%	49.7%
950	28.4%	38.5%	46.7%	49.1%
1000	28.7%	38.6%	47.0%	50.3%

Figures 33 and 34 contain graphs of the same information. Tables LXIV and LXV list data which represent estimates for the error in the mean value produced by each heuristic method. These numbers were calculated by using the following formula.

$$\tilde{Err} = h - \frac{\tilde{\chi}(n,p) - (k_{n,p} + 1)}{2} \tag{97}$$

where h is the value of the heuristic. Figures 35 and 36 provide graphs of these numbers. Estimates for the relative error in the mean values of the heuristics were calculated by employing the following formula.

$$\tilde{RelErr} = \frac{2 * \tilde{Err}}{\tilde{\chi}(n,p) - (k_{n,p} + 1)} \tag{98}$$

Tables LXVI and LXVII show the results of these calculations for n=50,100,..,1000 and p=0.20 and p=0.50, respectively. Graphs of the same results are displayed in Figures 37 and 38.

An analysis of this information supports the following conclusions.

1. The $[k_{n,p}+1, \tilde{\chi}(n,p)]$ intervals can be used to make probabilistic statements about $\overline{\chi}(SG_{n,p})$. For example, based on the theory presented in this chapter, it is very likely that $\overline{\chi}(SG_{1000,0.50}) = 83 \pm 3$.

2. As suspected, for each of the heuristic methods, the error in its estimate for $\overline{\chi}(SG_{n,p})$ increases with as n grows.

3. However, the results for the mean relative error calculations were somewhat surprising. Although these values increase rapidly as n grows from 50 to 200, from this point on the slope of the curve quickly begins to level off. For example, for p=0.50, the relative error of SRLF, when n=750, is about 28%±6% and, for n=1000, it is 29%±5%. Thus, it appears that the algorithms reach a point at which their effectiveness, when measured by relative error, becomes virtually independent of problem size.

2. Evaluation of CGCP Heuristic Algorithms.

Tables LXVIII through LXXIII, along with Figures 39 through 44, contain the same categories of information for the CGCP as were presented in the previous six tables and figures for the SGCP. Based on this data the following conclusions can be made.

1. The $[k_{n,p}+1,\bar{\chi}(n,p)]$ intervals can be used to make probabilistic statements about $\overline{\chi}(CG_{n,p})$. For example, it is very likely that $\overline{\chi}(CG_{1000,0.50}) = 148 \pm 15$.

2. Almost without exception, the estimated mean errors in the heuristic results increase as n grows.

3. The relative errors of the algorithms appear to vary within a surprisingly small interval throughout the range in values of n. For example, for $p=0.50$, the relative errors vary from a low of about 19% at $n=50$ to a high of 25% at $n=250$. For $n=1000$, it is estimated to be about 24%. However, because of the large size of the $[k_{n,p}+1,\bar{\chi}(n,p)]$ intervals, there is room for a lot of variation at any given value of n.

4. For both the SGCP and the CGCP, $p=0.20$ and 0.50, and all values of n, except $n=50$ in the case of the SGCP, the heuristic algorithms produce results that leave a lot of room for improvement.

Some interesting data, that lends some more credibility to the theory, is provided in Table LXXIV. Recall that, from Chapter III, a 99% confidence interval for the value of $\overline{\chi}(CG_{n,0.50})/\overline{\chi}(SG_{n,0.50})$, where $n=10,12,..,32$, was calculated to be [1.755,1.806]. The table displays values for the midpoint of $[k_{n,p}+1,\bar{\chi}(n,p)]$, labeled *SMidpt*, the midpoint of $[k_{n,p}+1,\bar{\chi}(n,p)]$, denoted *CMidpt*, the product of 1.755 and the midpoint of $[k_{n,p}+1,\bar{\chi}(n,p)]$, designated *CI_LB*, and the product of 1.806 and the midpoint of $[k_{n,p}+1,\bar{\chi}(n,p)]$, denoted *CI_UB*. The two products represent a projected confidence interval for $\overline{\chi}(CG_{n,0.50})$. There is no sound basis for interpreting this interval as a true confidence interval; but, the results do provide some more support for the plausibility of the probabilistic bounds. Figure 45 shows the same data graphically. Notice that the CMidpt values are either between the associated CI_LB and CI_UB numbers or within 2 of CI_LB.

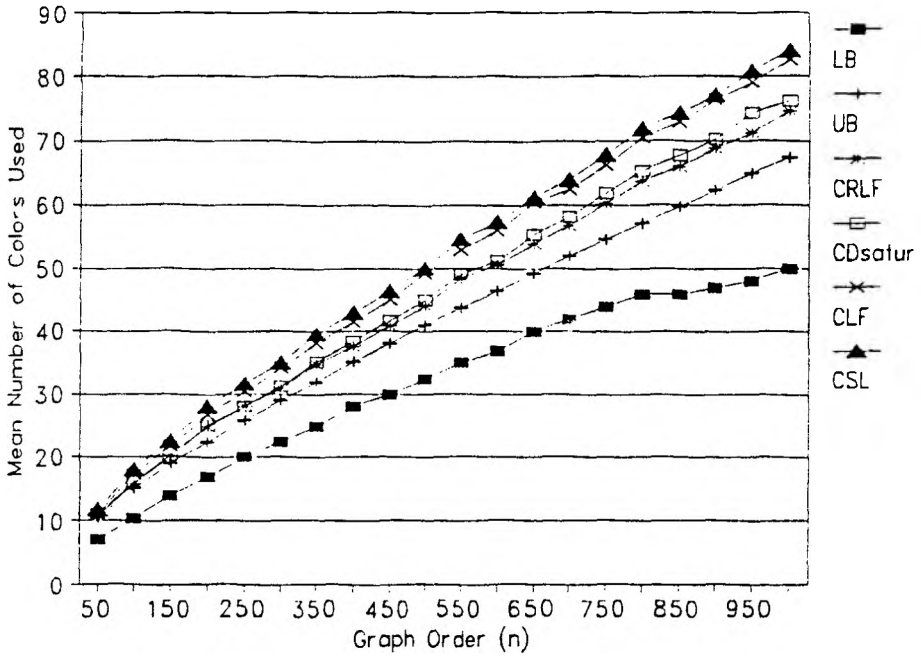


Figure 39. CGCP Heuristic Algorithm Evaluation (p=0.20)

TABLE LXVIII

COMPOSITE GRAPH COLORING
HEURISTIC ESTIMATES VERSUS PROBABILISTIC BOUNDS
p=0.20

n	$\tilde{k}_{n,p}+1$	$\bar{\chi}(n,p)$	CRLF	CDsatur	CLF	CSL
50	7.0	10.5	10.7	10.8	11.2	11.8
100	10.0	15.0	15.8	16.1	17.3	18.2
150	14.0	18.8	20.2	20.2	21.7	22.6
200	17.0	22.4	24.9	25.2	26.9	28.2
250	20.0	25.9	28.1	28.0	30.5	31.8
300	22.0	29.1	31.0	31.3	34.3	35.2
350	25.0	32.1	34.8	35.3	38.2	39.7
400	28.0	35.2	37.7	38.5	41.5	43.0
450	30.0	38.1	41.0	41.8	45.1	46.6
500	32.0	41.0	44.1	45.1	49.2	50.1
550	35.0	43.7	48.3	49.0	52.9	54.7
600	37.0	46.5	50.7	51.3	56.0	57.5
650	40.0	49.3	53.9	55.4	60.5	61.4
700	42.0	52.0	56.7	58.3	62.5	64.0
750	44.0	54.7	60.3	61.9	66.3	67.9
800	46.0	57.3	63.7	65.4	70.4	71.8
850	46.0	59.8	65.9	67.7	72.8	74.2
900	47.0	62.3	68.8	70.3	76.3	77.0
950	48.0	64.9	71.1	74.2	79.1	80.9
1000	50.0	67.4	74.5	76.2	82.5	84.0

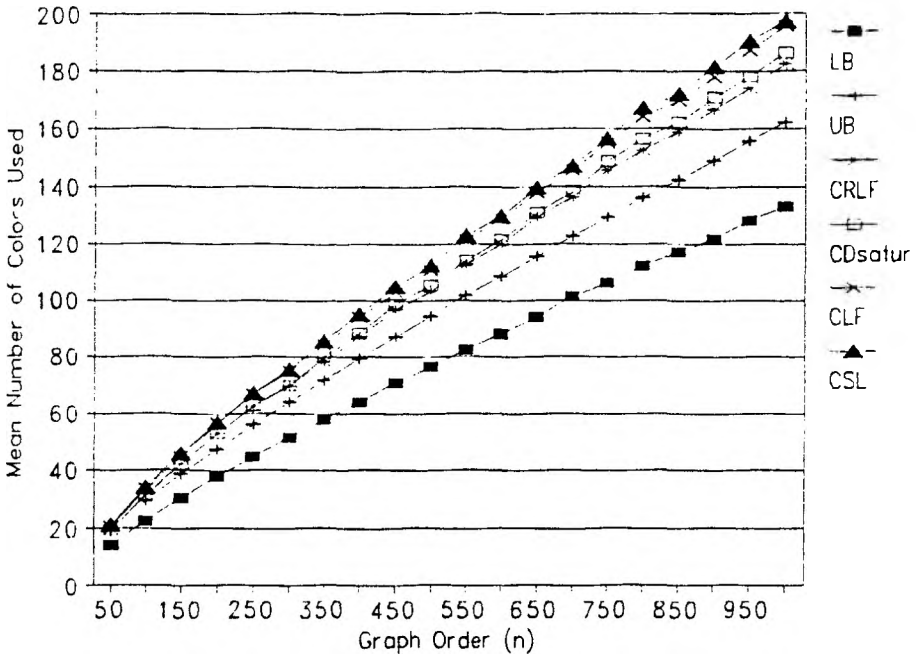


Figure 40. CGCP Heuristic Algorithm Evaluation (p=0.50)

TABLE LXIX

COMPOSITE GRAPH COLORING
HEURISTIC ESTIMATES VERSUS PROBABILISTIC BOUNDS
p=0.50

n	$\tilde{k}_{n,p}+1$	$\tilde{\chi}(n,p)$	CRLF	CDsatur	CLF	CSL
50	14.0	18.6	19.4	19.3	20.4	20.8
100	22.0	29.3	31.5	31.1	33.3	33.9
150	30.0	38.7	42.1	41.9	44.7	45.8
200	38.0	47.5	53.0	53.0	56.7	57.0
250	44.0	55.9	62.2	62.7	66.5	67.4
300	51.0	64.1	69.5	70.0	74.8	75.6
350	58.0	72.0	78.4	80.0	84.5	85.7
400	64.0	79.5	86.9	88.5	93.8	94.9
450	71.0	87.0	96.5	98.4	102.9	104.9
500	76.0	94.3	103.2	105.4	110.7	112.4
550	82.0	101.3	112.8	114.1	121.2	122.7
600	88.0	108.4	120.0	121.4	128.8	129.7
650	94.0	115.6	129.3	131.0	138.0	139.6
700	101.0	122.7	136.2	138.4	145.7	147.1
750	106.0	129.4	145.4	148.7	155.0	156.8
800	112.0	136.1	152.6	156.5	163.9	167.2
850	117.0	142.6	159.0	162.2	169.8	172.2
900	122.0	149.2	167.1	171.0	178.2	181.8
950	128.0	155.8	174.5	178.4	187.2	190.5
1000	133.0	162.3	182.5	186.2	195.7	197.1

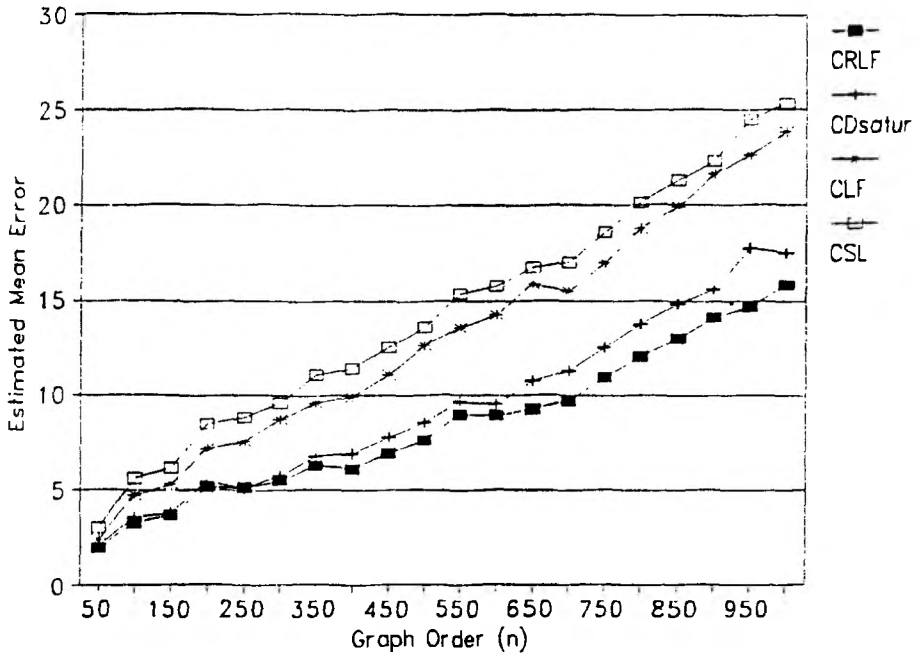


Figure 41. CGCP Heuristic Algorithm Evaluation (p=0.20)

TABLE LXX

COMPOSITE GRAPH COLORING
ESTIMATED MEAN ERRORS OF HEURISTIC ALGORITHMS
p=0.20

n	CRLF	CDsatur	CLF	CSL
50	2.0	2.1	2.5	3.0
100	3.3	3.6	4.8	5.7
150	3.7	3.8	5.3	6.2
200	5.2	5.5	7.2	8.5
250	5.2	5.1	7.5	8.8
300	5.5	5.7	8.7	9.6
350	6.3	6.8	9.6	11.1
400	6.1	6.9	9.9	11.4
450	6.9	7.8	11.1	12.5
500	7.6	8.6	12.7	13.6
550	8.9	9.6	13.5	15.3
600	8.9	9.5	14.2	15.7
650	9.3	10.8	15.9	16.8
700	9.7	11.3	15.5	17.0
750	11.0	12.6	17.0	18.6
800	12.1	13.8	18.8	20.2
850	13.0	14.8	19.9	21.3
900	14.1	15.6	21.6	22.3
950	14.6	17.7	22.6	24.4
1000	15.8	17.5	23.8	25.3

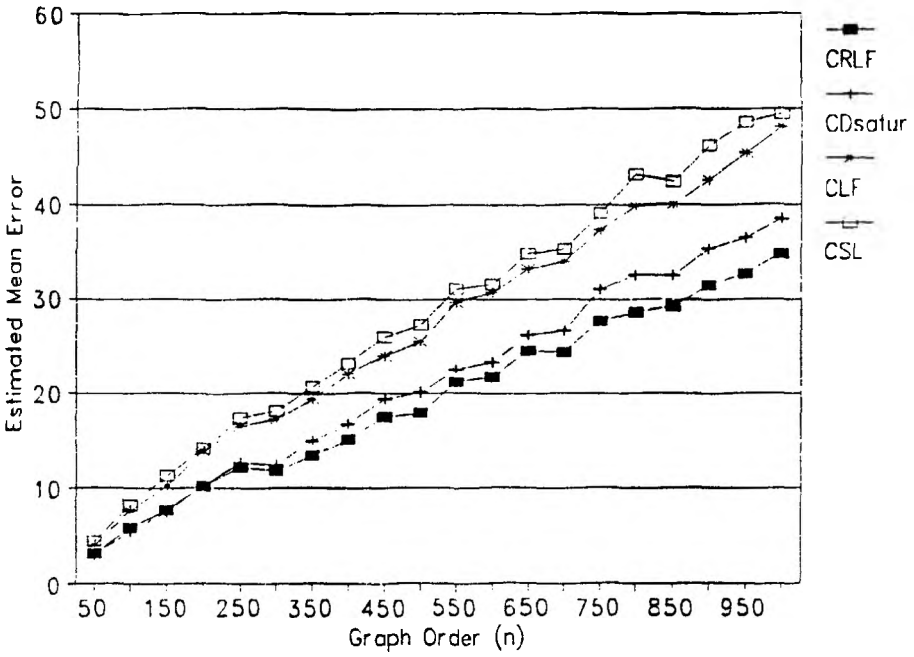


Figure 42. CGCP Heuristic Algorithm Evaluation (p=0.50)

TABLE LXXI

COMPOSITE GRAPH COLORING
ESTIMATED MEAN ERRORS OF HEURISTIC ALGORITHMS
p=0.50

n	CRLF	CDsatur	CLF	CSL
50	3.1	3.0	4.1	4.5
100	5.9	5.5	7.6	8.3
150	7.7	7.5	10.3	11.4
200	10.3	10.3	13.9	14.2
250	12.2	12.7	16.5	17.4
300	11.9	12.5	17.2	18.1
350	13.4	15.0	19.5	20.7
400	15.2	16.8	22.1	23.2
450	17.5	19.4	23.9	25.9
500	18.0	20.2	25.5	27.2
550	21.1	22.4	29.5	31.0
600	21.8	23.2	30.6	31.5
650	24.5	26.2	33.2	34.8
700	24.4	26.6	33.9	35.3
750	27.7	31.0	37.3	39.1
800	28.6	32.5	39.9	43.2
850	29.2	32.4	40.0	42.4
900	31.5	35.4	42.6	46.2
950	32.6	36.5	45.3	48.6
1000	34.8	38.5	48.0	49.4

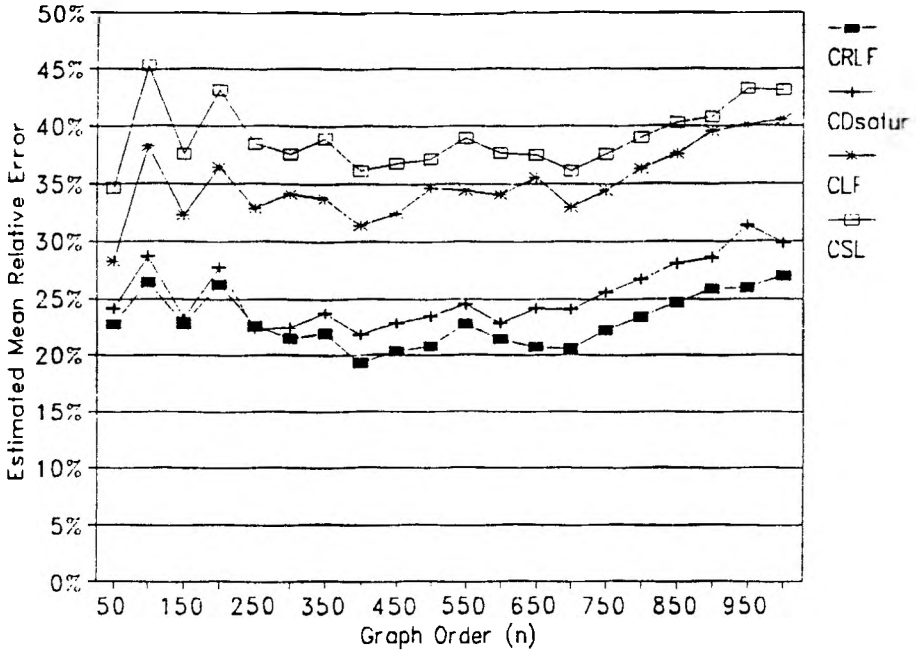


Figure 43. CGCP Heuristic Algorithm Evaluation (p=0.20)

TABLE LXXII

COMPOSITE GRAPH COLORING
ESTIMATED MEAN RELATIVE ERRORS
p=0.20

n	CRLF	CDsatur	CLF	CSL
50	22.7%	24.1%	28.2%	34.6%
100	26.5%	28.7%	38.3%	45.3%
150	22.8%	23.3%	32.3%	37.6%
200	26.3%	27.7%	36.4%	43.1%
250	22.6%	22.2%	32.9%	38.4%
300	21.5%	22.4%	34.2%	37.6%
350	22.0%	23.7%	33.8%	39.0%
400	19.3%	21.8%	31.3%	36.2%
450	20.4%	22.8%	32.5%	36.8%
500	20.9%	23.5%	34.7%	37.2%
550	22.7%	24.5%	34.4%	39.0%
600	21.4%	22.9%	34.1%	37.7%
650	20.8%	24.1%	35.6%	37.6%
700	20.6%	24.0%	33.0%	36.1%
750	22.3%	25.5%	34.4%	37.7%
800	23.4%	26.7%	36.3%	39.1%
850	24.6%	28.0%	37.7%	40.3%
900	25.9%	28.6%	39.6%	40.9%
950	25.9%	31.4%	40.1%	43.3%
1000	26.9%	29.8%	40.6%	43.1%

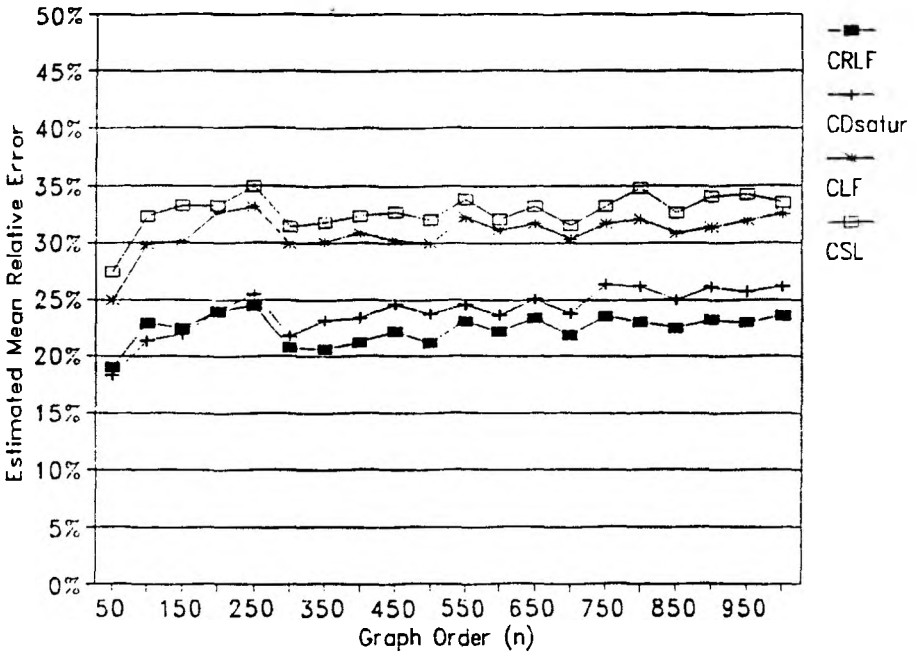


Figure 44. CGCP Heuristic Algorithm Evaluation (p=0.50)

TABLE LXXIII

COMPOSITE GRAPH COLORING
ESTIMATED MEAN RELATIVE ERRORS
p=0.50

n	CRLF	CDsatur	CLF	CSL
50	19.1%	18.3%	24.9%	27.4%
100	22.9%	21.3%	29.8%	32.3%
150	22.5%	21.9%	30.1%	33.3%
200	24.0%	24.1%	32.6%	33.3%
250	24.5%	25.5%	33.1%	34.9%
300	20.7%	21.7%	29.9%	31.4%
350	20.7%	23.1%	30.1%	31.9%
400	21.2%	23.4%	30.8%	32.3%
450	22.1%	24.5%	30.2%	32.8%
500	21.2%	23.7%	30.0%	32.0%
550	23.1%	24.5%	32.2%	33.9%
600	22.2%	23.6%	31.2%	32.1%
650	23.4%	25.0%	31.7%	33.2%
700	21.8%	23.8%	30.3%	31.5%
750	23.5%	26.3%	31.7%	33.2%
800	23.0%	26.2%	32.1%	34.8%
850	22.5%	24.9%	30.8%	32.6%
900	23.2%	26.1%	31.4%	34.1%
950	23.0%	25.7%	31.9%	34.3%
1000	23.6%	26.1%	32.5%	33.5%

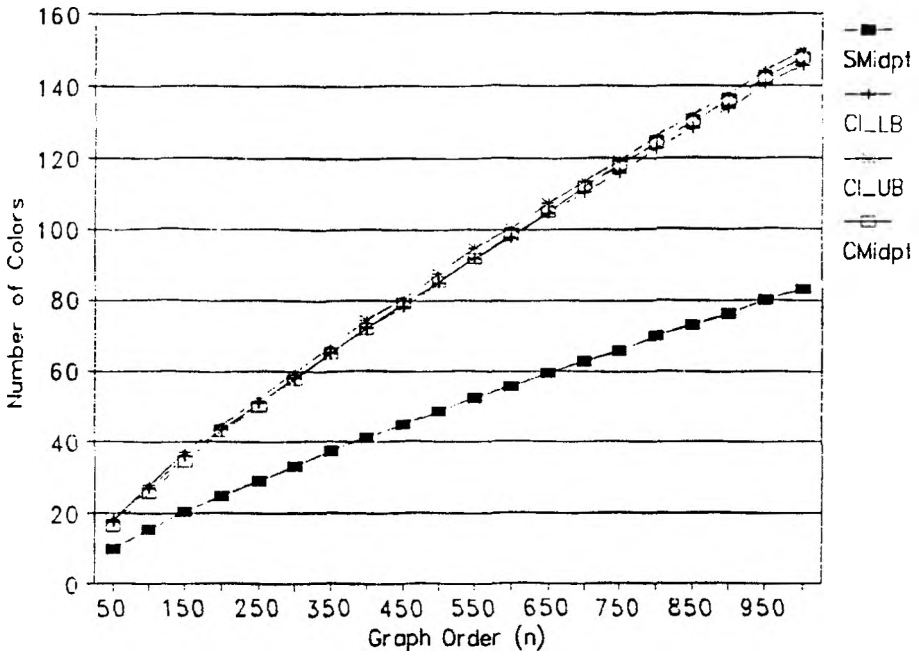


Figure 45. Analysis of Probabilistic Bounds

TABLE LXXIV

COMPOSITE GRAPH COLORING
PROJECTED CONFIDENCE INTERVAL BOUNDS VERSUS
MIDPOINT OF PROBABILISTIC BOUNDS INTERVAL

n	SMidpt	CI_LB	CI_UB	CMidpt
50	10.0	17.6	18.1	16.3
100	15.4	27.0	27.8	25.6
150	20.4	35.8	36.8	34.4
200	24.7	43.4	44.7	42.7
250	29.0	50.8	52.3	50.0
300	33.0	58.0	59.7	57.5
350	37.1	65.2	67.0	65.0
400	41.1	72.1	74.2	71.7
450	44.5	78.1	80.4	79.0
500	48.4	85.0	87.5	85.2
550	52.3	91.8	94.5	91.7
600	55.6	97.6	100.4	98.2
650	59.4	104.2	107.3	104.8
700	62.7	110.0	113.2	111.8
750	66.0	115.8	119.1	117.7
800	69.8	122.4	126.0	124.0
850	73.0	128.1	131.8	129.8
900	76.1	133.6	137.5	135.6
950	79.8	140.0	144.1	141.9
1000	83.0	145.6	149.8	147.7

VI. SUMMARY

In Chapter I, it was stated that this study was initiated with three goals.

1. To design and test algorithms for finding the chromatic number of random composite graphs.
2. To design and test heuristic algorithms for estimating the chromatic number of random composite graphs.
3. To establish a methodology for evaluating the effectiveness of heuristic algorithms, as measured by how well they estimate $\overline{\chi}(CG_{n,p})$.

Since the CGCP is a generalization of the SGCP, it was hypothesized that a productive approach would be to build upon the extensive knowledge base which has been established for the SGCP.

With respect to the first goal, an exact vertex-sequential with dynamic vertex reordering algorithm for solving instances of the SGCP, designed by Korman, was converted into an exact method for solving composite graphs. In Chapter III, it was demonstrated that this procedure is as efficient as the Korman algorithm.

Chapter IV contained a description of ten heuristic algorithms for estimating an optimal coloring of random composite graphs. Of the ten, seven were new methods designed specifically for this study. The three new methods which were based on the well known Dsatur algorithm for the SGCP proved to be very competitive when compared to the methods which had previously been shown to be most effective, namely, the LF1I algorithm, devised by Clementson and Elphick, and the RLF1 procedure, conceived by Roberts.

Finally, Chapter V presented a methodology, which was based on the application of probability theory to the definition of random graphs, for establishing probabilistic lower and upper bounds on the values of $\overline{\chi}(SG_{n,p})$ and $\overline{\chi}(CG_{n,p})$. These bounds were then used to evaluate the effectiveness of the basic vertex-sequential and color-sequential heuristic algorithms. It was shown that these methods use a number of colors that are typically on the order of 20% to 45% above the optimal.

VII. SUGGESTED TOPICS FOR FURTHER RESEARCH

The exact vertex-sequential algorithms described in Chapter III are designed to find an optimal solution of the SGCP and CGCP by either explicitly or implicitly processing every feasible solution. Since the feasible solution space grows exponentially with n and the fathoming capability of the algorithms is not sufficiently effective, they become intractable to compute for graph orders of 100 or more.

The heuristic methods presented in Chapter IV are greedy algorithms which use some vertex selection and color assignment criteria to build a single feasible solution. This feasible solution is the algorithm's approximation for an optimal solution. Such procedures execute in polynomial time; but, as was demonstrated in Chapter V, they produce estimates that are, on the average, rather poor.

The two approaches just described seem to be at opposite ends of the spectrum of all possible methods. In the one case, every feasible solution is considered, at least implicitly, and, in the other case, only one. Recently, some *intermediate* strategies for coloring standard graphs have been conceived and tested. The term intermediate was selected because these techniques typically require much more time to compute than the traditional greedy heuristics; but, they are by no means exhaustive search methods, so the time required may be large, but tractable.

The XRLF algorithm, which was referred to in Chapter V, is one example of such a search tactic [*JA89*]. XRLF is parameter controlled generalization of SRLF that is based on ideas first suggested by Johri and Matula [*JM82*], augmented with an exact coloring phase when the set of vertices remaining to be processed is sufficiently small. Both SRLF and XRLF are color-sequential approaches which attempt to construct the current color class by finding a near optimal solution to the following NP-hard problem: Find an independent set $U_2 \subseteq U_1$, where U_1 is the current set of uncolored vertices, such that $|\{<u_1,u_2> \in E: u_1 \in U_1 - U_2 \wedge u_2 \in U_2\}|$ is maximized. Thus, the number of edges in the induced uncolored subgraph will be minimized. SRLF uses the simple criteria described in Chapter IV for constructing U_2, which

results in relatively fast execution, but, compromises the quality of the solution. The name RLF* has been adopted for the algorithm that would solve the above problem optimally. XRLF is designed to provide for the calculation of a subset of approximations to RLF* through the values assigned to four parameters which control the granularity of the search space sampling.

Recall that, in Chapter IV, it was recorded that SRLF used an average of 106.8 colors on a random sample of $SG_{100,0.5}$. The potential effectiveness of XRLF is illustrated by the fact that, on a graph which SRLF used 108 colors, it found a 97 coloring in 0.2 hours, a 93 coloring in 0.5 hours, a 92 coloring in 0.6 hours, a 90 coloring in 4.7 hours, an 89 coloring in 8.0 hours, an 88 coloring in 9.6 hours, an 87 coloring in 18.3 hours, and an 86 coloring in 68.3 hours.

Another category of long running, but tractable, heuristics is called *simulated annealing* and is based on a general technique for improving on local optimization algorithms proposed by Kirkpatrick, Gelatt, and Vecchi [*KG83*]. Local optimization starts with an initial feasible solution and makes only neighborhood changes which produce feasible solutions that represent improvements in the objective function. The final solution is guaranteed to be a local minimum; but, it typically is not a global minimum. The objective of simulated annealing is to avoid being trapped into deterministically searching the neighborhood of a local minimum. To accomplish this, it always moves to the next candidate solution, if that solution produces an improvement in the objective function. However, with probability $e^{-\Delta/T}$, where Δ is the magnitude of the regression and T is an internal parameter called the *temperature*, it will elect to move to a candidate that has a negative effect on the objective function. The temperature controls the percentage of the sample space that is selected by determining the frequency and magnitude of uphill moves. Initially, it is set to some large value causing a large area of the search space to be processed. But, as execution proceeds, it is gradually decreased to force the search to converge to a local optimum that, hopefully, closely approximates the global optimum.

Three simulated annealing algorithms for coloring standard graphs were

encountered in the literature under the names Kempe Chain Annealing, Penalty Function Annealing, and Fixed K Annealing [*MS86,VA87,JA89*]. The third reference reported on tests which compared these three implementations to XRLF; and, the results indicated that the best approach depended on the order of the graph and its edge density.

Thus, for random standard graphs, it has been shown that XRLF, as well as the simulated annealing methods, are capable of producing significantly better colorings than basic greedy approaches such as SDsatur and SRLF, assuming large amounts of computing time are available. To date, no such formulation for the CGCP has been designed and tested. The potential for constructing similarly capable algorithms for coloring random composite graphs seems plausible, again, assuming that run times on the order of several hours are practical.

APPENDIX

PROGRAM LISTINGS

The program listings, pages 156-369, are contained in a separately bound volume.

PROGRAM LISTINGS

FOR

ALGORITHMS AND PROBABILISTIC BOUNDS FOR THE CHROMATIC NUMBER OF RANDOM COMPOSITE GRAPHS

BY

JACK L. OAKES, 1949-

APPENDIX

PROGRAM LISTINGS

1. GenGraph.Pas

```
{$N+,E-,M 65520,0,655360}
program GenGraph;
uses CRT;
const
  MaxNumOfVert = 1000;
type
  String11 = string[11];  VertexType = 1..MaxNumOfVert;
  VertexSetType = set of 1..250;  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  VertSetArray : array[1..4] of VertexSetType;
                end;
var
  GraphType : char;  Count,NumOfVert,EdgeCnt : word;
  NumOfEdges : longint;  Seed : longint;
  Param,EdgeDensity : double;  FileName,Extension : String11;
  Vert, Chrom : array [1..MaxNumOfVert] of word;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  DataOut,Out : text;

{*** Generate power with integer exponent ***}
function Power (Base : double; Exponent : integer) : double;
  var
    I : word;  Result : double;
  begin
    Result := 1.0;
    for I := 1 to Exponent do
      Result := Result * Base;
    Power := Result;
  end;

{*** Generate N Factorial ***}
function Fact (N : integer) : integer;
  var
    I,Result : word;
  begin
    Result := 1;
```

```
    for I := 1 to N do
      Result := Result * I;
    Fact := Result;
  end;

{*** Random number generator ***}
function Random : double;
  const
    A : longint = 16807;     M : longint = 2147483647;
    Q : longint = 127773;    R : longint = 2836;
  var
    Low,High,Test : longint;
  begin
    High := Seed div Q;    Low := Seed mod Q;
    Test := A*Low - R*High;
    if (Test>0) then
      Seed := Test
    else
      Seed := Test + M;
    Random := Seed/M;
  end;

{*** Initialize variables ***}
procedure Initialize;
  var
    Vert : VertexType;    AdjListPtr : AdjListPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin
        new(AdjListPtr);
        AdjVertArray[Vert] := AdjListPtr;
      end;
  end;

{*** Generate vertex chromaticities ***}
procedure GenChrom;
  var
    I,J : word;    CumPDF : double;
  begin
    for I := 1 to NumOfVert do
      begin
        CumPDF := Param/(Exp(Param)-1);
        J := 1;
        while (CumPDF<Random+0.06) do
```

```
            begin
              J := J + 1;
              CumPDF := CumPDF + Power(Param,J)/((Exp(Param)-1)*Fact(J));
            end;
          Chrom[I] := J;
        end;
    end;

{*** Add vertex to adjacency set structure ***}
procedure AddNewAdjVert (Vert1,Vert2 : VertexType);
  var
     I : word;    Vert : VertexType;
  begin
    case Vert2 of
      1..250    : begin
                    I := 1;    Vert := Vert2;
                  end;
      251..500  : begin
                    I := 2;    Vert := Vert2 - 250;
                  end;
      501..750  : begin
                    I := 3;    Vert := Vert2 - 500;
                  end;
      751..1000 : begin
                    I := 4;    Vert := Vert2 - 750;
                  end;
    end;
    AdjVertArray[Vert1]^.VertSetArray[I] :=
      AdjVertArray[Vert1]^.VertSetArray[I] + [Vert];
  end;

{*** Generate graph edges ***}
procedure GenEdges;
  var
    Vert,Vert1,Vert2 : word;    Rand : double;
  begin
    for Vert := 1 to NumOfVert do
      begin
        AdjVertArray[Vert]^.VertSetArray[1] := [];
        AdjVertArray[Vert]^.VertSetArray[2] := [];
        AdjVertArray[Vert]^.VertSetArray[3] := [];
        AdjVertArray[Vert]^.VertSetArray[4] := [];
      end;
    NumOfEdges := 0;    Vert1 := 1;
```

```
    while (Vert1<NumOfVert) do
      begin
        Vert2 := Vert1 + 1;
        while (Vert2<=NumOfVert) do
          begin
            if (EdgeDensity>=Random) then
              begin
                NumOfEdges := NumOfEdges + 1;
                AddNewAdjVert (Vert1,Vert2);
              end;
            Vert2 := Vert2 + 1;
          end;
        Vert1 := Vert1 + 1;
      end;
  end;

{*** Check if Vert1 is adjacent to Vert2 ***}
function IsIn (Vert1,Vert2 : VertexType) : boolean;
  var
    I : word;
    Vert : VertexType;
  begin
    case Vert1 of
      1..250    : begin
                    I := 1;   Vert := Vert1;
                  end;
      251..500  : begin
                    I := 2;   Vert := Vert1 - 250;
                  end;
      501..750  : begin
                    I := 3;   Vert := Vert1 - 500;
                  end;
      751..1000 : begin
                    I := 4;   Vert := Vert1 - 750;
                  end;
    end;
    IsIn := Vert in AdjVertArray[Vert2]^.VertSetArray[I];
  end;

{*** Output file defining random graph ***}
procedure OutputGraph;
  var
    Vert1,Vert2,J : word;
  begin
```

```
assign (DataOut, 'C:\THESIS\GRAPHS\' + FileName);
rewrite (DataOut);
writeln (DataOut,'{*********************************}');
writeln (DataOut,'{*                               *}');
writeln (DataOut,'{* Data File For GRPH_CLR.PAS *}');
writeln (DataOut,'{*                               *}');
writeln (DataOut,'{*********************************}');
writeln (DataOut,'{*** Number of Vertices ***}');
writeln (DataOut,NumOfVert);
writeln (DataOut,'{*** Number of Edges ***}');
writeln (DataOut,NumOfEdges);
writeln (DataOut,'{*** Vertex Definitions (Vertex Chromaticity) ***}');
Vert1 := 1;
while (Vert1 <= NumOfVert) do
  begin
    J := 1;
    while (Vert1 <= NumOfVert) and (J <= 10) do
      begin
        write (DataOut,Vert1:2,' ',Chrom[Vert1]:2,' ');
        Vert1 := Vert1 + 1;    J := J + 1;
      end;
    writeln (DataOut);
  end;
writeln (DataOut,'{*** Edge Definitions (Vertex1 Vertex2) ***}');
J := 1;    Vert1 := 1;
while (Vert1 <= NumOfVert-1) do
  begin
    Vert2 := Vert1 + 1;
    while (Vert2 <= NumOfVert) and (J <= 10) do
      begin
        if (IsIn(Vert2,Vert1)) then
          begin
            write (DataOut,Vert1:2,' ',Vert2:2,' ');
            J := J + 1;
            if (J > 10) then
              begin
                writeln (DataOut);
                J := 1;
              end;
          end;
        Vert2 := Vert2 + 1;
      end;
    Vert1 := Vert1 + 1;
  end;
```

```
    close (DataOut);
  end;

begin
  NumOfVert := 500;
  EdgeDensity := 0.50;
  NumOfEdges := round(EdgeDensity*NumOfVert*(NumOfVert-1)/2.0);
  ClrScr;
  writeln;
  writeln ('NumOfEdges = ',NumOfEdges:5);
  Param := 1.0;
  Seed := 493544361;
  Initialize;
  for Count := 1 to 25 do
    begin
      writeln ('Working on File ',Count);
      if (Count<10) then
        begin
          FileName := 'G500P50.00';
          str(Count:1,Extension);
        end
      else
        begin
          FileName := 'G500P50.0';
          str(Count:2,Extension);
        end;
      FileName := FileName + Extension;
      GenChrom;
      GenEdges;
    end;
end.
```

2. SDynVSE.Pas

```
program SDynVSE;
uses DOS,CRT;
const
  MaxVert = 100;  {Max no. of vertices}
  MaxColor = 15;  {Upper bound for chromatic number}
  MaxCurUBoundAccesses = 250000;
type
  VertexType = 1..MaxVert;
  VertexSetType = set of VertexType;
  ColorType = 0..MaxColor+1;
```

```
  ColorSetType = set of ColorType;
  String11 = string[11];
  ActivityType = record
                   UBound : ColorType;
                   Accesses : longint;
                 end;
  WVertListPtrType = ^WVertListType;
  WVertListType = record
                    Vert : VertexType;
                    NextPtr : WVertListPtrType;
                  end;
  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  Vert : VertexType;
                  NextPtr : AdjListPtrType;
                end;
  CandidateListPtrType = ^CandidateListType;
  CandidateListType = record
                        Vert : VertexType;
                        NextPtr : CandidateListPtrType;
                      end;

var
  FileName,FileNamePrefix,Extension : String11;
  FoundOne : boolean;    Code,GraphType : char;
  Hour1,Min1,Sec1,Frac1 : word;    Hour2,Min2,Sec2,Frac2 : word;
  StartGraphNum,EndGraphNum,NumOfGraphs,ActArrPos : word;
  Pos,CRCount,Count,NumOfVert,NumOfEdges : word;
  Accesses,CurUBoundAccesses : longint;
  ElapsedTime,AvgElapsedTime,AvgChrom,AvgNumOfAccesses : real;
  UBound,Chromaticity : word;
  WVertListHeadPtr,WVertListRearPtr : WVertListPtrType;
  CandidateListHeadPtr,CandidateListRearPtr : CandidateListPtrType;
  ActivityArray : array[1..30] of ActivityType;
  AssgnArray : array[VertexType] of ColorType;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  AdjColorArray : array[VertexType] of ColorSetType;
  ChromArray,CDegArray,WDegArray : array[VertexType] of integer;
  Prn,DataIn,Out : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word;    ConvertedName : String11;
```

```
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  begin
    ClrScr;   writeln ('Application:  Standard Graph Coloring');
    writeln;  writeln ('Program:  SDynVSE.Pas -- Exact');
    writeln;  write ('Enter File Name Prefix: ');
    readln (FileNamePrefix);
    writeln;  write ('Enter Number of Vertices: ');
    readln (NumOfVert);
    writeln;  write ('Enter Start Graph Number: ');
    readln (StartGraphNum);
    write ('Enter End Graph Number: ');
    readln (EndGraphNum);
    NumOfGraphs := EndGraphNum - StartGraphNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Count : byte;    Vert : VertexType;
    AdjListHeadPtr : AdjListPtrType;
  begin
    new (WVertListHeadPtr);
    WVertListHeadPtr^.NextPtr := nil;
    WVertListRearPtr := WVertListHeadPtr;
    new(CandidateListHeadPtr);
    CandidateListHeadPtr^.NextPtr := nil;
    CandidateListRearPtr := CandidateListHeadPtr;
    for Vert := 1 to NumOfVert do
      begin
        new(AdjListHeadPtr);
        AdjListHeadPtr^.NextPtr := nil;
        AdjVertArray[Vert] := AdjListHeadPtr;
      end;
    AvgChrom := 0.0;
    AvgNumOfAccesses := 0.0;
    AvgElapsedTime := 0.0;
```

```
    writeln (Out,'Application: Standard Graph Coloring');
    writeln (Out,'Program:     SDynVSE.Pas -- Exact');
    writeln (Out,'Method:      Vertex Sequential with Dynamic Vertex Reording');
    writeln (Out,'Ordering:    Colored Deg, White Deg');
    writeln (Out);
    writeln (Out,'              Chromatic  Num Of  Processing');
    writeln (Out,' File Name    Number  Accesses    Time          ',
          'Search Summary':47);
    write  (Out,'------------ --------- -------- ----------       ');
    for Count := 1 to 80 do
      write (Out,'-');
    writeln (Out);
  end;

{*** Perform all initializations for global variables. ***}
procedure Initialize;
  var
    Vert : VertexType;
  begin
    Chromaticity := 999;       Accesses := 0;
    CurUBoundAccesses := 0;    ActArrPos := 0;
    for Vert := 1 to MaxVert do
      begin
        AdjColorArray[Vert] := [];    CDegArray[Vert] := 0;
        WDegArray[Vert] := 0;         AssgnArray[Vert] := 0;
      end;
  end;

{***  Input problem definition and update graph definition structures. ***}
procedure ReadData;
  var
    I,Vertex,Vertex1,Vertex2,Chrom : integer;
    AdjListHeadPtr : AdjListPtrType;
    WVertListPtr : WVertListPtrType;
  procedure AddNewAdjVertex (Vert1,Vert2 : VertexType);
    var
      Ptr,RearPtr : AdjListPtrType;
    begin
      Ptr := AdjVertArray[Vert1];
      while (Ptr^.NextPtr<>nil) do
        Ptr := Ptr^.NextPtr;
      RearPtr := Ptr;       new (Ptr);
      Ptr^.Vert := Vert2;   Ptr^.NextPtr:= nil;
      RearPtr^.NextPtr := Ptr;
```

```
    end;
  begin
    for I := 1 to 6 do
      readln (DataIn);
    readln (DataIn,NumOfVert);
    readln (DataIn);  readln (DataIn,NumOfEdges);
    readln (DataIn);
    for I := 1 to NumOfVert do
      begin
        read (DataIn,Vertex,Chrom);
        new(WVertListPtr);
        WVertListPtr^.Vert := Vertex;
        WVertListPtr^.NextPtr := nil;
        WVertListRearPtr^.NextPtr := WVertListPtr;
        WVertListRearPtr := WVertListPtr;
      end;
    readln(DataIn);  readln(DataIn);
    for I := 1 to NumOfEdges do
      begin
        read (DataIn,Vertex1,Vertex2);
        AddNewAdjVertex (Vertex1,Vertex2);
        AddNewAdjVertex (Vertex2,Vertex1);
        WDegArray[Vertex1] := WDegArray[Vertex1] + 1;
        WDegArray[Vertex2] := WDegArray[Vertex2] + 1;
      end;
  end;

{*** Release all stack space which is no longer needed ***}
procedure ReclaimStackSpace;
  var
    Vert : VertexType;
    WVertListPtr,PrevPtr1 : WVertListPtrType;
    AdjListPtr,PrevPtr2 : AdjListPtrType;
  begin
    WVertListPtr := WVertListHeadPtr^.NextPtr;
    WVertListHeadPtr^.NextPtr := nil;
    WVertListRearPtr := WVertListHeadPtr;
    while (WVertListPtr<>nil) do
      begin
        PrevPtr1 := WVertListPtr;
        WVertListPtr := WVertListPtr^.NextPtr;
        dispose(PrevPtr1);
      end;
    for Vert := 1 to NumOfVert do
```

```
      begin
        AdjListPtr := AdjVertArray[Vert]^.NextPtr;
        AdjVertArray[Vert]^.NextPtr := nil;
        while (AdjListPtr<>nil) do
          begin
            PrevPtr2 := AdjListPtr;
            AdjListPtr := AdjListPtr^.NextPtr;
            dispose(PrevPtr2);
          end;
      end;
  end;

{*** Select the vert to prcess next.  First priority is given to the  ***}
{    vert with the largest cdeg and then largest wdeg.                  }
procedure SelectVertex (var Vertex : VertexType);
  var
    FoundIt : boolean;
    MaxCDeg,MaxWDeg : integer;
    Vert : VertexType;
    WVertListPtr : WVertListPtrType;
    CandidateListPtr,PrevPtr : CandidateListPtrType;
  begin
    MaxCDeg := 0;
    WVertListPtr := WVertListHeadPtr^.NextPtr;
    while (WVertListPtr<>nil) do
      begin
        Vert := WVertListPtr^.Vert;
        if (CDegArray[Vert]>MaxCDeg) then
          MaxCDeg := CDegArray[Vert];
        WVertListPtr := WVertListPtr^.NextPtr;
      end;
    WVertListPtr := WVertListHeadPtr^.NextPtr;
    while (WVertListPtr<>nil) do
      begin
        Vert := WVertListPtr^.Vert;
        if (CDegArray[Vert]=MaxCDeg) then
          begin
            new (CandidateListPtr);
            CandidateListPtr^.Vert := Vert;
            CandidateListPtr^.NextPtr := nil;
            CandidateListRearPtr^.NextPtr := CandidateListPtr;
            CandidateListRearPtr := CandidateListPtr;
          end;
        WVertListPtr := WVertListPtr^.NextPtr;
```

```
      end;
    MaxWDeg := 0;
    CandidateListPtr := CandidateListHeadPtr^.NextPtr;
    while (CandidateListPtr <>nil) do
      begin
        Vert := CandidateListPtr^.Vert;
        if (WDegArray[Vert]>MaxWDeg) then
          MaxWDeg := WDegArray[Vert];
        CandidateListPtr := CandidateListPtr^.NextPtr;
      end;
    FoundIt := false;
    CandidateListPtr := CandidateListHeadPtr^.NextPtr;
    CandidateListHeadPtr^.NextPtr := nil;
    CandidateListRearPtr := CandidateListHeadPtr;
    while (CandidateListPtr <>nil) do
      begin
        Vert := CandidateListPtr^.Vert;
        if (WDegArray[Vert]=MaxWDeg) and (not FoundIt) then
          begin
            Vertex := Vert;
            FoundIt := true;
          end;
        PrevPtr := CandidateListPtr;
        CandidateListPtr := CandidateListPtr^.NextPtr;
        dispose(PrevPtr);
      end;
  end;

{*** Generate the set of feasible colors for a vertex. ***}
procedure GenFeasibleSet (Vertex:VertexType; ColorsUsed:ColorType; var
FeasibleSet:ColorSetType);
  var
    Color,MaxColor,Limit1,Limit2 : ColorType;
  begin
    Limit1 := ColorsUsed + 1;
    Limit2 := UBound - 1;
    if (Limit1 <=Limit2) then
      MaxColor := Limit1
    else
      MaxColor := Limit2;
    FeasibleSet := [];
    for Color := 1 to MaxColor do
      FeasibleSet := FeasibleSet + [Color];
    FeasibleSet := FeasibleSet - AdjColorArray[Vertex];
```

```
  end;

{*** Maintains all coloring def sets and arrays.  It is invoked  ***}
{    for backward moves, as well as, forward moves.                }
procedure PerformUpdates (Vertex : VertexType; Color : ColorType;
                          Control : integer);
  var
    StillIn,Again : boolean;
    Vert1,Vert2 : VertexType;
    AdjListPtr1,AdjListPtr2 : AdjListPtrType;
    WVertListPtr,PrevPtr : WVertListPtrType;
  begin
    if (Control=1) then        {Process updates for a forward move}
      begin
        AssgnArray[Vertex] := Color;
        WVertListPtr := WVertListHeadPtr^.NextPtr;
        PrevPtr := WVertListHeadPtr;
        while (WVertListPtr^.Vert<>Vertex) do
          begin
            PrevPtr := WVertListPtr;
            WVertListPtr := WVertListPtr^.NextPtr;
          end;
        PrevPtr^.NextPtr := WVertListPtr^.NextPtr;
        if (WVertListRearPtr=WVertListPtr) then
          WVertListRearPtr := PrevPtr;
        dispose(WVertListPtr);
        AdjListPtr1 := AdjVertArray[Vertex]^.NextPtr;
        while (AdjListPtr1<>nil) do
         begin
           Vert1 := AdjListPtr1^.Vert;
           if (AssgnArray[Vert1]=0) then
              begin
                 WDegArray[Vert1] := WDegArray[Vert1] - 1;
                 if (not (Color in AdjColorArray[Vert1])) then
                    begin
                       AdjColorArray[Vert1] := AdjColorArray[Vert1] + [Color];
                       CDegArray[Vert1] := CDegArray[Vert1] + 1;
                    end;
              end;
           AdjListPtr1 := AdjListPtr1^.NextPtr;
         end;
      end
    else           {Process updates for a backward/horizontal move}
      begin
```

```
            AssgnArray[Vertex] := 0;
            new (WVertListPtr);
            WVertListPtr^.Vert := Vertex;
            WVertListPtr^.NextPtr := nil;
            WVertListRearPtr^.NextPtr := WVertListPtr;
            WVertListRearPtr := WVertListPtr;
            AdjListPtr1 := AdjVertArray[Vertex]^.NextPtr;
            while (AdjListPtr1<>nil) do
              begin
                Vert1 := AdjListPtr1^.Vert;
                if (AssgnArray[Vert1]=0) then
                  begin
                    WDegArray[Vert1] := WDegArray[Vert1] + 1;
                    StillIn := false; {Is color still in adj. color set?}
                    Again := true;
                    AdjListPtr2 := AdjVertArray[Vert1]^.NextPtr;
                    while (AdjListPtr2<>nil) and (Again) do
                      begin
                        Vert2 := AdjListPtr2^.Vert;
                        if (AssgnArray[Vert2]=Color) then
                          begin
                            StillIn := true;
                            Again := false;
                          end
                        else
                          AdjListPtr2 := AdjListPtr2^.NextPtr;
                      end;
                    if (not StillIn) then
                      begin
                        AdjColorArray[Vert1] := AdjColorArray[Vert1] - [Color];
                        CDegArray[Vert1] := CDegArray[Vert1] - 1;
                      end;
                  end;
                AdjListPtr1 := AdjListPtr1^.NextPtr;
              end;
          end;
      end;

{*** This is the recursive procedure which generates an optimal coloring ***}
procedure GenColors (Level : VertexType; ColorsUsed : ColorType);
  var
    Vert,Vertex : VertexType;    Clr,Color : ColorType;
    FeasibleSet : ColorSetType;
  begin
```

```
    Accesses := Accesses + 1;
    CurUBoundAccesses := CurUBoundAccesses + 1;
    if (Level>NumOfVert) then
      begin
        FoundOne := true;
        Chromaticity := ColorsUsed;
        UBound := ColorsUsed;
        ActArrPos := ActArrPos + 1;
        ActivityArray[ActArrPos].UBound := UBound;
        ActivityArray[ActArrPos].Accesses := CurUBoundAccesses;
        CurUBoundAccesses := 0;
      end
    else
      begin
        SelectVertex (Vertex);
        GenFeasibleSet (Vertex,ColorsUsed,FeasibleSet);
        Color := 1;
        while (Color<UBound) and (not FoundOne) and
            (CurUBoundAccesses<MaxCurUBoundAccesses) do
          begin
            if (Color in FeasibleSet) then
              begin
                PerformUpdates (Vertex,Color,1);
                if (ColorsUsed>=Color) then
                  GenColors (Level+1,ColorsUsed)
                else
                  GenColors (Level+1,Color);
                PerformUpdates (Vertex,Color,-1);
              end;
            Color := Color + 1;
          end;
      end;
  end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : real;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
```

```
  end;

begin
  assign (Prn,'Prn');
  rewrite (Prn);
  assign (Out, 'REPORT.PRN');
  rewrite (Out);
  ObtainUserInfo;
  PrintReportHeadingsAndInitTotals;
  for Count := StartGraphNum to EndGraphNum do
    begin
      if (Count<10) then
        begin
          FileName := FileNamePrefix + '.00';
          str(Count:1,Extension);
        end
      else
        begin
          FileName := FileNamePrefix + '.0';
          str(Count:2,Extension);
        end;
      FileName := FileName + Extension;
      assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName);
      reset (DataIn);
      Initialize;          writeln; writeln ('Processing ',FileName);
      ReadData;            UBound := MaxColor + 1;
      FoundOne := true;    GetTime (Hour1,Min1,Sec1,Frac1);
      while (FoundOne) do
        begin
          FoundOne := false;    GenColors (1,0);
        end;
      GetTime (Hour2,Min2,Sec2,Frac2);
      CalcElapsedTime;         close (DataIn);
      UBound := UBound - 1;   ActArrPos := ActArrPos + 1;
      ActivityArray[ActArrPos].UBound := UBound;
      ActivityArray[ActArrPos].Accesses := CurUBoundAccesses;
      if (CurUBoundAccesses>=MaxCurUBoundAccesses) then
        Code := '*'
      else
        Code := ' ';
      writeln (UpperCase(FileName):11,Chromaticity:7,Code:1,
              Accesses:9,ElapsedTime:11:2,' ':9);
      write (Out,UpperCase(FileName):11,Chromaticity:7,Code:1,
            Accesses:9,ElapsedTime:11:2,' ':9);
```

```
        CRCount := 0;
        for Pos := 1 to ActArrPos do
          begin
            if (CRCount=7) then
              begin
                writeln (Out); write (Out,' ':48);
                CRCount := 0;
              end;
            write (Out,' (',ActivityArray[Pos].UBound:2,
                  ActivityArray[Pos].Accesses:6,')');
            CRCount := CRCount + 1;
          end;
        writeln (Out);
        AvgChrom := AvgChrom + Chromaticity;
        AvgNumOfAccesses := AvgNumOfAccesses + Accesses;
        AvgElapsedTime := AvgElapsedTime + ElapsedTime;
        ReclaimStackSpace;
      end;
    AvgChrom := AvgChrom/NumOfGraphs;
    AvgNumOfAccesses := AvgNumOfAccesses/NumOfGraphs;
    AvgElapsedTime := AvgElapsedTime/NumOfGraphs;
    writeln (Out,'              --------- -------- ----------');
    writeln (Out,'  Averages',AvgChrom:8:2,AvgNumOfAccesses:11:1,
          AvgElapsedTime:9:2);
    close (Out);
end.
```

3. CDynVSE.Pas

```
program CDynVSE;
uses DOS,CRT;
const
  MaxVert = 100;  {Max no. of vertices}
  MaxColor = 25;  {Upper bound for chromatic number}
  MaxCurUBoundAccesses = 250000;
type
  VertexType = 1..MaxVert;
  VertexSetType = set of VertexType;
  ColorType = 0..MaxColor+1;
  ColorSetType = set of ColorType;
  String11 = string[11];
  ActivityType = record
                   UBound : ColorType;    Accesses : longint;
                 end;
```

```
  WVertListPtrType = ^WVertListType;
  WVertListType = record
                    Vert : VertexType;    NextPtr : WVertListPtrType;
                  end;
  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  Vert : VertexType;      NextPtr : AdjListPtrType;
                end;
  CandidateListPtrType = ^CandidateListType;
  CandidateListType = record
                        Vert : VertexType;    NextPtr : CandidateListPtrType;
                      end;

var
  FileName,FileNamePrefix,Extension : String11;
  FoundOne : boolean;
  GraphType,Code : char;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2,BegFileNum,EndFileNum,UBound
: word;

CRCount,MaxVertexChrom,NumOfGraphs,NumOfOptSolut,ActArrPos,Pos,Chro
maticity : word;
  Count,NumOfVert,NumOfEdges : integer;
  Accesses,CurUBoundAccesses : longint;
  ElapsedTime,AvgElapsedTime,AvgChrom,AvgNumOfAccesses : real;
  WVertListHeadPtr,WVertListRearPtr : WVertListPtrType;
  CandidateListHeadPtr,CandidateListRearPtr : CandidateListPtrType;
  ActivityArray : array[1..30] of ActivityType;
  AssgnArray : array[VertexType] of ColorSetType;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  AdjColorArray : array[VertexType] of ColorSetType;
  ChromArray,CDegArray : array[VertexType] of ColorType;
  WChromDegArray,WDegArray : array[VertexType] of word;
  Prn,DataIn,Out : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word;    ConvertedName : String11;
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;
```

```
{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  begin
    ClrScr;
    writeln ('Application:  Composite Graph Coloring'); writeln;
    writeln ('Program: CDynVSE.Pas -- Exact'); writeln;
    writeln ('Ordering: Colored Deg, Chromaticity, White Chromatic Deg,
          White Deg');
    writeln; write ('Enter File Name Prefix: ');
    readln (FileNamePrefix);
    writeln; write ('Enter Number of Vertices: ');
    readln (NumOfVert);
    writeln; write ('Enter Begin File Number: '); readln (BegFileNum);
    write ('Enter End File Number: '); readln (EndFileNum);
    NumOfGraphs := EndFileNum - BegFileNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Count : byte;    Vert : VertexType;    AdjListHeadPtr : AdjListPtrType;
  begin
    new(WVertListHeadPtr); WVertListHeadPtr^.NextPtr := nil;
    WVertListRearPtr := WVertListHeadPtr; new(CandidateListHeadPtr);
    CandidateListHeadPtr^.NextPtr := nil;
    CandidateListRearPtr := CandidateListHeadPtr;
    for Vert := 1 to NumOfVert do
      begin
        new(AdjListHeadPtr);
        AdjListHeadPtr^.NextPtr := nil;
        AdjVertArray[Vert] := AdjListHeadPtr;
      end;
    AvgChrom := 0.0; AvgNumOfAccesses := 0.0; AvgElapsedTime := 0.0;
    write (Out,chr(27),'(s16.67H');
    writeln (Out,'Application: Composite Graph Coloring');
    writeln (Out,'Program:     CDynVSE.Pas -- Exact');
    writeln (Out,'Method:      Vertex Sequential with Dynamic Vertex Reording');
    writeln (Out,'Ordering:    Colored Deg, Chromaticity, ',
          'White Chromatic Deg, White Deg');
    writeln (Out);
    writeln (Out,'              Chromatic  Num Of  Processing');
    writeln (Out,' File Name    Number  Accesses   Time   Search Summary':55);
    write  (Out,'---------- --------- -------- ----------',' ':8);
    for Count := 1 to 80 do
```

```
        write (Out,'-');
      writeln (Out);
  end;

{*** Perform all initializations for global variables. ***}
procedure Initialize;
  var
    Vert : VertexType;
  begin
    Chromaticity := 999; Accesses := 0; CurUBoundAccesses := 0;
    ActArrPos := 0;
    for Vert := 1 to MaxVert do
      begin
        AdjColorArray[Vert] := []; CDegArray[Vert] := 0;
        WChromDegArray[Vert] := 0;
        WDegArray[Vert] := 0; AssgnArray[Vert] := [];
      end;
  end;

{*** Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I,Vertex,Vertex1,Vertex2,Chrom : integer;
    AdjListHeadPtr : AdjListPtrType;
    WVertListPtr : WVertListPtrType;
  procedure AddNewAdjVertex (Vert1,Vert2 : VertexType);
    var
      Ptr,RearPtr : AdjListPtrType;
    begin
      Ptr := AdjVertArray[Vert1];
      while (Ptr^.NextPtr < > nil) do
        Ptr := Ptr^.NextPtr;
      RearPtr := Ptr; new (Ptr); Ptr^.Vert := Vert2;
      Ptr^.NextPtr := nil; RearPtr^.NextPtr := Ptr;
    end;
  begin
    for I := 1 to 6 do
      readln (DataIn);
    readln (DataIn,NumOfVert); readln (DataIn);
    readln (DataIn,NumOfEdges); readln (DataIn);
    MaxVertexChrom := 0;
    for I := 1 to NumOfVert do
      begin
        read (DataIn,Vertex,Chrom); ChromArray[Vertex] := Chrom;
```

```
          if (Chrom > MaxVertexChrom) then
            MaxVertexChrom := Chrom;
          new (WVertListPtr); WVertListPtr^.Vert := Vertex;
          WVertListPtr^.NextPtr:= nil;WVertListRearPtr^.NextPtr:= WVertListPtr;
          WVertListRearPtr := WVertListPtr;
        end;
      readln(DataIn);  readln(DataIn);
      for I := 1 to NumOfEdges do
        begin
          read (DataIn,Vertex1,Vertex2);
          AddNewAdjVertex(Vertex1,Vertex2); AddNewAdjVertex(Vertex2,Vertex1);
          WDegArray[Vertex1] := WDegArray[Vertex1] + 1;
          WDegArray[Vertex2] := WDegArray[Vertex2] + 1;
          WChromDegArray[Vertex1] := WChromDegArray[Vertex1] +
                                     ChromArray[Vertex2];
          WChromDegArray[Vertex2] := WChromDegArray[Vertex2] +
                                     ChromArray[Vertex1];
        end;
    end;

{*** Reclease the stack space which is no longer needed. ***}
procedure ReclaimStackSpace;
  var
    Vert : VertexType;
    WVertListPtr,PrevPtr1 : WVertListPtrType;
    AdjListPtr,PrevPtr2 : AdjListPtrType;
  begin
    WVertListPtr := WVertListHeadPtr^.NextPtr;
    WVertListHeadPtr^.NextPtr := nil; WVertListRearPtr := WVertListHeadPtr;
    while (WVertListPtr < > nil) do
      begin
        PrevPtr1 := WVertListPtr;
        WVertListPtr := WVertListPtr^.NextPtr; dispose (PrevPtr1);
      end;
    for Vert := 1 to NumOfVert do
      begin
        AdjListPtr := AdjVertArray[Vert]^.NextPtr;
        AdjVertArray[Vert]^.NextPtr := nil;
        while (AdjListPtr < > nil) do
          begin
            PrevPtr2 := AdjListPtr;
            AdjListPtr := AdjListPtr^.NextPtr; dispose (PrevPtr2);
          end;
      end;
```

```
  end;

{*** Select the vertex to prcess next ***}
procedure SelectVertex (var Vertex : VertexType);
  var
    FoundIt : boolean;
    MaxCDeg,MaxChrom,MaxWChromDeg,MaxWDeg : word;
    Vert : VertexType;
    WVertListPtr : WVertListPtrType;
    CandidateListPtr,PrevPtr : CandidateListPtrType;
  begin
    MaxCDeg := 0; WVertListPtr := WVertListHeadPtr^.NextPtr;
    while (WVertListPtr < > nil) do
      begin
        Vert := WVertListPtr^.Vert;
        if (CDegArray[Vert] > MaxCDeg) then
          MaxCDeg := CDegArray[Vert];
        WVertListPtr := WVertListPtr^.NextPtr;
      end;
    WVertListPtr := WVertListHeadPtr^.NextPtr;
    while (WVertListPtr < > nil) do
      begin
        Vert := WVertListPtr^.Vert;
        if (CDegArray[Vert] = MaxCDeg) then
          begin
            new (CandidateListPtr); CandidateListPtr^.Vert := Vert;
            CandidateListPtr^.NextPtr := nil;
            CandidateListRearPtr^.NextPtr := CandidateListPtr;
            CandidateListRearPtr := CandidateListPtr;
          end;
        WVertListPtr := WVertListPtr^.NextPtr;
      end;
    MaxChrom := 0; CandidateListPtr := CandidateListHeadPtr^.NextPtr;
    while (CandidateListPtr < > nil) do
      begin
        Vert := CandidateListPtr^.Vert;
        if (ChromArray[Vert] > MaxChrom) then
          MaxChrom := ChromArray[Vert];
        CandidateListPtr := CandidateListPtr^.NextPtr;
      end;
    PrevPtr := CandidateListHeadPtr; CandidateListPtr := PrevPtr^.NextPtr;
    while (CandidateListPtr < > nil) do
      begin
        Vert := CandidateListPtr^.Vert;
```

```
      if (ChromArray[Vert]<MaxChrom) then
        begin
          PrevPtr^.NextPtr := CandidateListPtr^.NextPtr;
          dispose (CandidateListPtr);
        end
      else
        PrevPtr := CandidateListPtr;
      CandidateListPtr := PrevPtr^.NextPtr;
    end;
MaxWChromDeg := 0; CandidateListPtr := CandidateListHeadPtr^.NextPtr;
while (CandidateListPtr<>nil) do
  begin
    Vert := CandidateListPtr^.Vert;
    if (WChromDegArray[Vert]>MaxWChromDeg) then
      MaxWChromDeg := WChromDegArray[Vert];
    CandidateListPtr := CandidateListPtr^.NextPtr;
  end;
PrevPtr := CandidateListHeadPtr; CandidateListPtr := PrevPtr^.NextPtr;
while (CandidateListPtr<>nil) do
  begin
    Vert := CandidateListPtr^.Vert;
    if (WChromDegArray[Vert]<MaxWChromDeg) then
      begin
        PrevPtr^.NextPtr := CandidateListPtr^.NextPtr;
        dispose (CandidateListPtr);
      end
    else
      PrevPtr := CandidateListPtr;
    CandidateListPtr := PrevPtr^.NextPtr;
  end;
MaxWDeg := 0; CandidateListPtr := CandidateListHeadPtr^.NextPtr;
while (CandidateListPtr<>nil) do
  begin
    Vert := CandidateListPtr^.Vert;
    if (WDegArray[Vert]>MaxWDeg) then
      MaxWDeg := WDegArray[Vert];
    CandidateListPtr := CandidateListPtr^.NextPtr;
  end;
FoundIt := false; CandidateListPtr := CandidateListHeadPtr^.NextPtr;
CandidateListHeadPtr^.NextPtr := nil;
CandidateListRearPtr := CandidateListHeadPtr;
while (CandidateListPtr<>nil) do
  begin
    Vert := CandidateListPtr^.Vert;
```

```
        if (WDegArray[Vert]=MaxWDeg) and (not FoundIt) then
          begin
            Vertex := Vert; FoundIt := true;
          end;
        PrevPtr := CandidateListPtr;
        CandidateListPtr := PrevPtr^.NextPtr; dispose (PrevPtr);
      end;
  end;

{*** Generate the set of feasible colors for a vertex. ***}
procedure GenFeasibleSet(Vertex:VertexType; ColorsUsed:ColorType;
                    var FeasibleSet:ColorSetType);
  var
    ColorIsFeasible : boolean;
    Color,Clr,MaxColor,Limit1,Limit2 : ColorType;
  begin
    Limit1 := ColorsUsed + MaxVertexChrom;
    Limit2 := UBound - ChromArray[Vertex];
    if (Limit1 <=Limit2) then
      MaxColor := Limit1
    else
      MaxColor := Limit2;
    FeasibleSet := [];
    for Color := 1 to MaxColor do
      begin
        ColorIsFeasible := true;
        for Clr := Color to Color+ChromArray[Vertex]-1 do
          if (Clr in AdjColorArray[Vertex]) then
            ColorIsFeasible := false;
        if ColorIsFeasible then
          FeasibleSet := FeasibleSet + [Color];
      end;
  end;

{*** Maintains all coloring definition sets and arrays.  It is invoked for ***}
{    backward moves, as well as, forward moves.                              }
procedure PerformUpdates (Vertex : VertexType; Color : ColorType;
                    LargestAssgnColor : ColorType; Control : integer);
  var
    StillIn,Again : boolean;    Vert1,Vert2 : VertexType;
    Clr : ColorType;            AdjListPtr1,AdjListPtr2 : AdjListPtrType;
    WVertListPtr,PrevPtr : WVertListPtrType;
  begin
    if (Control=1) then         {Process updates for a forward move}
```

```
    begin
      for Clr := Color to LargestAssgnColor do
        AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
      WVertListPtr := WVertListHeadPtr^.NextPtr;
      PrevPtr := WVertListHeadPtr;
      while (WVertListPtr^.Vert <> Vertex) do
        begin
          PrevPtr := WVertListPtr; WVertListPtr := WVertListPtr^.NextPtr;
        end;
      PrevPtr^.NextPtr := WVertListPtr^.NextPtr;
      if (WVertListRearPtr = WVertListPtr) then
        WVertListRearPtr := PrevPtr;
      dispose (WVertListPtr);
      AdjListPtr1 := AdjVertArray[Vertex]^.NextPtr;
      while (AdjListPtr1 <> nil) do
        begin
          Vert1 := AdjListPtr1^.Vert;
          if (AssgnArray[Vert1] = []) then
            begin
              WDegArray[Vert1] := WDegArray[Vert1] - 1;
              WChromDegArray[Vert1] := WChromDegArray[Vert1] -
                                       ChromArray[Vertex];
              for Clr := Color to LargestAssgnColor do
                if (not (Clr in AdjColorArray[Vert1])) then
                  begin
                    AdjColorArray[Vert1] := AdjColorArray[Vert1] + [Clr];
                    CDegArray[Vert1] := CDegArray[Vert1] + 1;
                  end;
            end;
          AdjListPtr1 := AdjListPtr1^.NextPtr;
        end;
    end
  else                {Process updates for a backward/horizontal move}
    begin
      AssgnArray[Vertex] := []; new (WVertListPtr);
      WVertListPtr^.Vert := Vertex; WVertListPtr^.NextPtr := nil;
      WVertListRearPtr^.NextPtr := WVertListPtr;
      WVertListRearPtr := WVertListPtr;
      AdjListPtr1 :=  AdjVertArray[Vertex]^.NextPtr;
      while (AdjListPtr1 <> nil) do
        begin
          Vert1 := AdjListPtr1^.Vert;
          if (AssgnArray[Vert1] = []) then
            begin
```

```
                    WDegArray[Vert1] := WDegArray[Vert1] + 1;
                    WChromDegArray[Vert1] := WChromDegArray[Vert1] +
                                             ChromArray[Vertex];
                    for Clr := Color to LargestAssgnColor do
                      begin
                        StillIn := false; Again := true;
                        AdjListPtr2 := AdjVertArray[Vert1]^.NextPtr;
                        while (AdjListPtr2<>nil) and (Again) do
                          begin
                            Vert2 := AdjListPtr2^.Vert;
                            if (Clr in AssgnArray[Vert2]) then
                              begin
                                StillIn := true; Again := false;
                              end
                            else
                              AdjListPtr2 := AdjListPtr2^.NextPtr;
                          end;
                        if (not StillIn) then
                          begin
                            AdjColorArray[Vert1] := AdjColorArray[Vert1] - [Clr];
                            CDegArray[Vert1] := CDegArray[Vert1] - 1;
                          end;
                      end;
                  end;
                AdjListPtr1 := AdjListPtr1^.NextPtr;
              end;
          end;
      end;

{*** This is the recursive procedure which generates an optimal coloring. ***}
procedure GenColors (Level : VertexType; ColorsUsed : ColorType);
  var
    Vert,Vertex : VertexType;
    Clr,Color,LargestAssgnColor : ColorType;
    FeasibleSet : ColorSetType;
  begin
    Accesses := Accesses + 1; CurUBoundAccesses := CurUBoundAccesses + 1;
    if (Level>NumOfVert) then
       begin
         FoundOne := true; Chromaticity := ColorsUsed;
         UBound := Chromaticity; ActArrPos := ActArrPos + 1;
         ActivityArray[ActArrPos].UBound := UBound;
         ActivityArray[ActArrPos].Accesses := CurUBoundAccesses;
         CurUBoundAccesses := 0;
```

```
        end
      else
        begin
          SelectVertex (Vertex); GenFeasibleSet (Vertex,ColorsUsed,FeasibleSet);
          Color := 1;
          while (Color+ChromArray[Vertex]-1<UBound) and (not FoundOne)
              and (CurUBoundAccesses<MaxCurUBoundAccesses) do
            begin
              if (Color in FeasibleSet) then
                begin
                  LargestAssgnColor := Color + ChromArray[Vertex] - 1;
                  PerformUpdates (Vertex,Color,LargestAssgnColor,1);
                  if (ColorsUsed>=LargestAssgnColor) then
                    GenColors (Level+1,ColorsUsed)
                  else
                    GenColors (Level+1,LargestAssgnColor);
                  PerformUpdates (Vertex,Color,LargestAssgnColor,-1);
                end;
              Color := Color + 1;
            end;
        end;
    end;

{*** Calculate time between two check points ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : real;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Prn,'Prn'); rewrite (Prn); assign (Out, 'REPORT.PRN'); rewrite (Out);
  ObtainUserInfo; PrintReportHeadingsAndInitTotals;
  for Count := BegFileNum to EndFileNum do
    begin
      if (Count<10) then
        begin
          FileName := FileNamePrefix + '.00'; str(Count:1,Extension);
        end
```

```
      else
        begin
          FileName := FileNamePrefix + '.0'; str(Count:2,Extension);
        end;
      FileName := FileName + Extension;
      assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName); reset (DataIn);
      Initialize; writeln ('Processing ',FileName);
      ReadData; UBound := MaxColor + 1; FoundOne := true;
      GetTime (Hour1,Min1,Sec1,Frac1);
      while (FoundOne) do
        begin
          FoundOne := false; GenColors (1,0);
        end;
      GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime; close (DataIn);
      UBound := UBound - 1; ActArrPos := ActArrPos + 1;
      ActivityArray[ActArrPos].UBound := UBound;
      ActivityArray[ActArrPos].Accesses := CurUBoundAccesses;
      if (CurUBoundAccesses> =MaxCurUBoundAccesses) then
        Code := '*'
      else
        Code := ' ';
      writeln(UpperCase(FileName):11,Chromaticity:7,Code:1,
            Accesses:9,ElapsedTime:11:2,' ':9);
      write (Out,UpperCase(FileName):11,Chromaticity:7,Code:1,Accesses:9,
            ElapsedTime:11:2,' ':9);
      CRCount := 0;
      for Pos := 1 to ActArrPos do
        begin
          if (CRCount=7) then
            begin
              writeln (Out); write (Out,' ':48); CRCount := 0;
            end;
          write (Out,' (',ActivityArray[Pos].UBound:2,
                ActivityArray[Pos].Accesses:6,')');
          CRCount := CRCount + 1;
        end;
      writeln (Out); AvgChrom := AvgChrom + Chromaticity;
      AvgNumOfAccesses := AvgNumOfAccesses + Accesses;
      AvgElapsedTime := AvgElapsedTime + ElapsedTime;
      ReclaimStackSpace;
    end;
  AvgChrom := AvgChrom/NumOfGraphs;
  AvgNumOfAccesses := AvgNumOfAccesses/NumOfGraphs;
  AvgElapsedTime := AvgElapsedTime/NumOfGraphs;
```

```
  writeln (Out,'            --------- -------- ----------');
  writeln (Out,'  Averages',AvgChrom:8:2,AvgNumOfAccesses:11:1,
       AvgElapsedTime:9:2);
  close (Out);
end.
```

4. SRLF.Pas -- Implementation of SRLF

```
{$M 65520,0,655360}
program SRLF;
uses DOS,CRT;
const
  MaxVert = 1000;  {Max no. of vertices}
  MaxColor = 150;  {Upper bound for chromatic number}
type
  String2 = string[2];    VertexType = 1..MaxVert;
  VertexSetType = set of 1..250;    ColorType = 0..MaxColor;
  ColorSetType = set of ColorType;  String11 = string[11];
  DegArrayType = array[VertexType] of word;
  VertexStatsArrayType = record
                           Vert  : VertexType;    UDeg  : word;
                           U1Deg : word;          U2Deg : word;
                         end;
  AltAdjListPtrType = ^AltAdjListType;
  AltAdjListType = record
                     VertSetArray : array[1..4] of VertexSetType;
                   end;
  VertListPtrType = ^VertListType;
  VertListType = record
                   Vert : VertexType;            UNextPtr : VertListPtrType;
                   U1NextPtr : VertListPtrType;
                 end;
  CandidateArrayPtrType = ^CandidateArrayType;
  CandidateArrayType = record
                         NumOfCandidates : word;
                         CandidateArray : array[1..700] of word;
                       end;
var
  Feasible : boolean; GraphType : char;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2,NumOfVert : word;
  NumOfUVert,NumOfU1Vert : word;
  I,Count,NumOfGraphs,BegGraphNum,EndGraphNum,RLFColors : word;
  NumOfEdges : longint;
  RLFTime,ElapsedTime,AvgElapsedTime,AvgRLFColors,AvgRLFTime : real;
```

```
  FileName,FileNamePrefix,Extension : String11;
  VertStatsArray : array[1..MaxVert] of VertexStatsArrayType;
  AssgnArray : array[VertexType] of ColorType;
  AltAdjVertArray : array[VertexType] of AltAdjListPtrType;
  AdjColorArray,FeasibleSetArray : array[VertexType] of ColorSetType;
  VertListHeadPtr : VertListPtrType;
  CandidateArrayPtr : CandidateArrayPtrType;
  DataIn,Out,Prn : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word; ConvertedName : String11;
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  var
    Code : integer;
  begin
    ClrScr; writeln ('Application:  Standard Graph Coloring'); writeln;
    writeln ('Program:  S_RLF.PAS -- Heuristic'); writeln;
    write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
    write ('Enter Num of Vertices -- '); readln (NumOfVert); writeln;
    write ('Enter Begin Graph Number:  '); readln (BegGraphNum); writeln;
    write ('Enter End Graph Number:  '); readln (EndGraphNum);
    BegGraphNum := 1; EndGraphNum := 25;
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Count : byte; Vert : VertexType;   AltAdjListPtr : AltAdjListPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin
        new(AltAdjListPtr); AltAdjVertArray[Vert] := AltAdjListPtr;
      end;
```

```
    new (VertListHeadPtr); VertListHeadPtr^.UNextPtr := nil;
    VertListHeadPtr^.U1NextPtr := nil; new (CandidateArrayPtr);
    AvgRLFColors := 0.0; AvgRLFTime := 0.0;
    writeln (Out,'Application: Standard Graph Coloring');
    writeln (Out,'Program:     S_RLF.PAS -- Heuristic');
    writeln (Out,'Method:      Recursively Largest First'); writeln (Out);
    writeln (Out,' File Name       RLF  ');
    writeln (Out,'-----------   --------');
  end;

{*** Initialize variables ***}
procedure Initialize;
  var
    I : VertexType; VertListPtr,VertListRearPtr : VertListPtrType;
  begin
    VertListRearPtr := VertListHeadPtr;
    for I := 1 to NumOfVert do
      begin
        new(VertListPtr); VertListPtr^.Vert := I; VertListPtr^.UNextPtr := nil;
        VertListPtr^.U1NextPtr := nil; VertListRearPtr^.UNextPtr := VertListPtr;
        VertListRearPtr := VertListPtr;
      end;
    for I := 1 to NumOfVert do
      AssgnArray[I] := 0;
  end;

{***  Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I : longint; Vert,Vertex,Vertex1,Vertex2,Chrom : word;
  procedure AddNewAdjVert2 (Vert1,Vert2 : VertexType);
    var
       I : word; Vert : VertexType;
    begin
      case Vert2 of
        1..250    : begin
                      I := 1; Vert := Vert2;
                    end;
        251..500  : begin
                      I := 2; Vert := Vert2 - 250;
                    end;
        501..750  : begin
                      I := 3; Vert := Vert2 - 500;
                    end;
```

```
            751..1000 : begin
                          I := 4; Vert := Vert2 - 750;
                        end;
          end;
          AltAdjVertArray[Vert1]^.VertSetArray[I] :=
            AltAdjVertArray[Vert1]^.VertSetArray[I] + [Vert];
        end;
      begin
        RLFColors := 999;
        for Vert := 1 to NumOfVert do
          begin
            AltAdjVertArray[Vert]^.VertSetArray[1] := [];
            AltAdjVertArray[Vert]^.VertSetArray[2] := [];
            AltAdjVertArray[Vert]^.VertSetArray[3] := [];
            AltAdjVertArray[Vert]^.VertSetArray[4] := [];
            VertStatsArray[Vert].Vert := Vert; VertStatsArray[Vert].UDeg := 0;
          end;

        for I := 1 to 6 do
          readln (DataIn);
        readln (DataIn,NumOfVert); readln (DataIn);
        readln (DataIn,NumOfEdges); readln (DataIn);
        for I := 1 to NumOfVert do
          read (DataIn,Vertex,Chrom);
        readln(DataIn);  readln(DataIn);
        for I := 1 to NumOfEdges do
          begin
            read (DataIn,Vertex1,Vertex2);
            AddNewAdjVert2 (Vertex1,Vertex2); AddNewAdjVert2 (Vertex2,Vertex1);
            VertStatsArray[Vertex1].UDeg := VertStatsArray[Vertex1].UDeg + 1;
            VertStatsArray[Vertex2].UDeg := VertStatsArray[Vertex2].UDeg + 1;
          end;
      end;

{*** Based on selection criteria, pick vertex to color next ***}
procedure SelectVertToColor1 (var Vertex : VertexType);
  var
    Vert : VertexType; MaxU1Deg : integer; VertListPtr : VertListPtrType;
  begin
    MaxU1Deg := -999; VertListPtr := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (VertStatsArray[Vert].U1Deg>MaxU1Deg) then
```

```
            begin
              Vertex := Vert; MaxU1Deg := VertStatsArray[Vert].U1Deg;
            end;
          VertListPtr := VertListPtr^.U1NextPtr;
        end;
    end;

procedure SelectVertToColor2 (var Vertex : VertexType);
  var
    Vert : VertexType; I,MaxU2Deg,MinU1Deg : integer; VertListPtr :
VertListPtrType;
  begin
    MaxU2Deg := -999;
    VertListPtr := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr < > nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (VertStatsArray[Vert].U2Deg > MaxU2Deg) then
          MaxU2Deg := VertStatsArray[Vert].U2Deg;
        VertListPtr := VertListPtr^.U1NextPtr;
      end;
    CandidateArrayPtr^.NumOfCandidates := 0;
    VertListPtr := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr < > nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (VertStatsArray[Vert].U2Deg = MaxU2Deg) then
          begin
            CandidateArrayPtr^.NumOfCandidates :=
              CandidateArrayPtr^.NumOfCandidates + 1;
            I := CandidateArrayPtr^.NumOfCandidates;
            CandidateArrayPtr^.CandidateArray[I] := Vert;
          end;
        VertListPtr := VertListPtr^.U1NextPtr;
      end;
    MinU1Deg := 999;
    for I := 1 to CandidateArrayPtr^.NumOfCandidates do
      begin
        Vert := CandidateArrayPtr^.CandidateArray[I];
        if (VertStatsArray[Vert].U1Deg < MinU1Deg) then
          begin
            Vertex := Vert; MinU1Deg := VertStatsArray[Vert].U1Deg;
          end;
      end;
```

```
  end;

{*** Check if Vert1 is adjacent to Vert2 ***}
function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
  var
    I : word; Vert : VertexType;
  begin
    case Vert1 of
      1..250   : begin
                   I := 1; Vert := Vert1;
                 end;
      251..500 : begin
                   I := 2; Vert := Vert1 - 250;
                 end;
      501..750 : begin
                   I := 3; Vert := Vert1 - 500;
                 end;
      751..1000 : begin
                   I := 4; Vert := Vert1 - 750;
                 end;
    end;
    IsIn2 := Vert in AltAdjVertArray[Vert2]^.VertSetArray[I];
  end;

{*** Delete Vertex from U list ***}
procedure DeleteUNode (Vertex : VertexType);
  var
    Vert : VertexType; FoundIt : boolean; VertListPtr,PrevPtr : VertListPtrType;
  begin
    FoundIt := false; PrevPtr := VertListHeadPtr;
    VertListPtr := VertListHeadPtr^.UNextPtr;
    NumOfUVert := NumOfUVert - 1;
    while (not FoundIt) do
      if (Vertex=VertListPtr^.Vert) then
        FoundIt := true
      else
        begin
          PrevPtr := VertListPtr; VertListPtr := VertListPtr^.UNextPtr;
        end;
    PrevPtr^.UNextPtr := VertListPtr^.UNextPtr; dispose (VertListPtr);
    VertListPtr := VertListHeadPtr^.UNextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
```

```
          if (IsIn2(Vert,Vertex)) then
            VertStatsArray[Vert].UDeg := VertStatsArray[Vert].UDeg - 1;
          VertListPtr := VertListPtr^.UNextPtr;
        end;
    end;

{*** Delete Vertex for U1 list **}
procedure DeleteU1Node (Vertex : VertexType);
  var
    Vert : VertexType; FoundIt : boolean; VertListPtr,PrevPtr : VertListPtrType;
  begin
    FoundIt := false; PrevPtr := VertListHeadPtr;
    VertListPtr := VertListHeadPtr^.U1NextPtr;
    NumOfU1Vert := NumOfU1Vert - 1;
    while (not FoundIt) do
      if (Vertex=VertListPtr^.Vert) then
        FoundIt := true
      else
        begin
          PrevPtr := VertListPtr; VertListPtr := VertListPtr^.U1NextPtr;
        end;
    PrevPtr^.U1NextPtr := VertListPtr^.U1NextPtr;
  end;

{*** Perform all data structure updates ***}
procedure DoUpdates (Vertex : VertexType);
  var
    Vert1,Vert2 : VertexType; VertListPtr1,VertListPtr2,PrevPtr : VertListPtrType;
  begin
    DeleteUNode (Vertex); DeleteU1Node (Vertex);
    PrevPtr := VertListHeadPtr; VertListPtr1 := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr1<>nil) do
      begin
        Vert1 := VertListPtr1^.Vert;
        if (IsIn2(Vert1,Vertex)) then
          begin
            NumOfU1Vert := NumOfU1Vert - 1;
            VertStatsArray[Vert1].U1Deg := VertStatsArray[Vert1].U1Deg - 1;
            PrevPtr^.U1NextPtr := VertListPtr1^.U1NextPtr;
            VertListPtr2 := VertListHeadPtr^.U1NextPtr;
            while (VertListPtr2<>nil) do
              begin
                Vert2 := VertListPtr2^.Vert;
                if (IsIn2(Vert2,Vert1)) then
```

```
                    begin
                      VertStatsArray[Vert2].U1Deg :=
                        VertStatsArray[Vert2].U1Deg - 1;
                      VertStatsArray[Vert2].U2Deg :=
                        VertStatsArray[Vert2].U2Deg + 1;
                    end;
                  VertListPtr2 := VertListPtr2^.U1NextPtr;
                end;
            end
          else
            PrevPtr := VertListPtr1;
          VertListPtr1 := VertListPtr1^.U1NextPtr;
        end;
    end;

{*** Initialize U1 list ***}
procedure InitializeU1List;
  var
    Vert : VertexType; VertListPtr : VertListPtrType;
  begin
    NumOfU1Vert := NumOfUVert; VertListPtr := VertListHeadPtr;
    VertListPtr^.U1NextPtr := VertListPtr^.UNextPtr;
    VertListPtr := VertListHeadPtr^.UNextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        VertStatsArray[Vert].U1Deg := VertStatsArray[Vert].UDeg;
        VertStatsArray[Vert].U2Deg := 0;
        VertListPtr^.U1NextPtr := VertListPtr^.UNextPtr;
        VertListPtr := VertListPtr^.UNextPtr;
      end;
  end;

{*** RLF algorithm implementation ***}
procedure DoRLFColoring;
  var
    Vertex : VertexType;
  begin
    RLFColors := 0; NumOfUVert := NumOfVert;
    while (NumOfUVert>0) do
      begin
        InitializeU1List; RLFColors := RLFColors + 1;
        SelectVertToColor1 (Vertex); AssgnArray [Vertex] := RLFColors;
        DoUpDates (Vertex);
```

```
        while (NumOfU1Vert>0) do
          begin
            SelectVertToColor2 (Vertex); AssgnArray[Vertex] := RLFColors;
            DoUpdates (Vertex);
          end;
      end;
  end;


{*** Check AssgnArray for feasibility ***}
procedure CheckAssgnArray (ColorNum : word);
  var
    NumOfAdjVert,Clr : word; Vert1,Vert2 : VertexType;
  begin
    Feasible := true; Vert1 := 1;
    while (Vert1<=NumOfVert) and (Feasible) do
      begin
        if (AssgnArray[Vert1]<=0) or (AssgnArray[Vert1]>ColorNum) then
          begin
            writeln ('AssgnArray for Vert ',Vert1,' is invalid.'); Feasible := false;
          end;
        for Clr := 1 to ColorNum do
          if (Clr=AssgnArray[Vert1]) then
            begin
              Vert2 := 1;
              while (Vert2<=NumOfVert) and (Feasible) do
                begin
                  if (IsIn2(Vert2,Vert1)) and
                       (Clr=AssgnArray[Vert2]) then
                    begin
                      Feasible := false;
                      writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                    end
                  else
                    Vert2 := Vert2 + 1;
                end;
            end;
        Vert1 := Vert1 + 1;
      end;
  end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
  var
```

```
    BegTime,EndTime : real;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Prn,'con'); rewrite (Prn); assign (Out, 'REP1.PRN'); rewrite (Out);
  ObtainUserInfo; PrintReportHeadingsAndInitTotals;
  for Count := BegGraphNum to EndGraphNum do
    begin
      if (Count<10) then
        begin
          FileName := FileNamePrefix + '.00'; str(Count:1,Extension);
        end
      else
        begin
          FileName := FileNamePrefix + '.0'; str(Count:2,Extension);
        end;
      FileName := FileName + Extension;
      assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName);
      reset (DataIn); writeln; writeln ('Processing ',FileName);
      ReadData; close (DataIn);
      Initialize; writeln('Working on RLF');
      GetTime (Hour1,Min1,Sec1,Frac1);
      DoRLFColoring;
      GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
      RLFTime := ElapsedTime; AvgRLFTime := AvgRLFTime + RLFTime;
      AvgRLFColors := AvgRLFColors + RLFColors;
      CheckAssgnArray(RLFColors);
      writeln (UpperCase(FileName):11,RLFColors:8);
      writeln (RLFTime:22:2); writeln (' ':10,Feasible:11);
      writeln (Out,UpperCase(FileName):11,RLFColors:8);
      writeln (Out,RLFTime:22:2); writeln (Out,' ':10,Feasible:11);
    end;
  AvgRLFColors := AvgRLFColors/NumOfGraphs; AvgRLFTime :=
    AvgRLFTime/NumOfGraphs;
  writeln (Out,'---------- --------'); writeln (Out,'  Averages',AvgRLFColors:11:2);
  writeln (Out,AvgRLFTime:22:2); close (Out);
end.
```

5. SVSI1.Pas -- Implementation of SLFI1 and SSLI1

```
{$M 65520,0,655360}
{*** Implements SLFI1 and SSLI1 ***}
program SVSI1;
uses DOS,CRT;
const
  MaxVert = 500;  {Max no. of vertices}
  MaxColor = 80;  {Upper bound for chromatic number}
type
  VertexType = 1..MaxVert;
  VertexSetType = set of 1..250;
  ColorType = 0..MaxColor;
  ColorSetType = set of ColorType;
  String11 = string[11];
  VertexOrderArrayType = record
                           Vert : VertexType;
                           Deg : word;
                         end;

  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  NumOfAdjVert : word;
                  VertArray : array[1..300] of VertexType;
                end;

  AltAdjListPtrType = ^AltAdjListType;
  AltAdjListType = record
                     VertSetArray : array[1..2] of VertexSetType;
                   end;
var
  FileName,FileNamePrefix,Extension : String11;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2,NumOfVert,NumOfEdges:word;
  I,Count,NumOfGraphs,BegGraphNum,EndGraphNum : word;
  LFI1Colors,SLI1Colors : word;
  LFI1Time,SLI1Time,ElapsedTime,AvgElapsedTime : real;
  AvgLFI1Colors,AvgSLI1Colors,AvgLFI1Time,AvgSLI1Time : real;
  SaveVertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
  VertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
  AssgnArray,SaveAssgnArray : array[VertexType] of ColorType;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  AltAdjVertArray : array[VertexType] of AltAdjListPtrType;
  AdjColorArray,SaveAdjColorArray : array[VertexType] of ColorSetType;
  DegArray : array[VertexType] of word;
```

```
  FeasibleSetArray,SaveFeasibleSetArray : array[VertexType] of ColorSetType;
  FeasibleArray : array[1..5] of boolean;
  DataIn,Out,Prn : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word; ConvertedName : String11;
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  var
    Code : integer;
  begin
    ClrScr; writeln (MemAvail,' bytes available.');
    writeln ('Largest free block is ',MaxAvail,' bytes'); writeln;
    writeln ('Application:  Standard Graph Coloring'); writeln;
    writeln ('Program:  SVSI1.PAS -- Heuristic'); writeln;
    write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
    write ('Enter Num of Vertices -- '); readln (NumOfVert); writeln;
    write ('Enter Begin Graph Number:  '); readln (BegGraphNum); writeln;
    write ('Enter End Graph Number:  '); readln (EndGraphNum);
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Vert : VertexType; AdjListPtr : AdjListPtrType;
    AltAdjListPtr : AltAdjListPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin
        new(AdjListPtr); AdjVertArray[Vert] := AdjListPtr;
      end;
    for Vert := 1 to NumOfVert do
      begin
        new(AltAdjListPtr); AltAdjVertArray[Vert] := AltAdjListPtr;
```

```
      end;
    AvgLFI1Colors := 0.0; AvgSLI1Colors := 0.0; AvgLFI1Time := 0.0;
    AvgSLI1Time := 0.0;
    writeln (Out,'Application: Standard Graph Coloring');
    writeln (Out,'Program:     SVSI1.PAS -- Heuristic');
    writeln (Out,'Method:      Vertex Sequential with Interchange'); writeln (Out);
    writeln (Out,' File Name      LFI1       SLI1  ');
    writeln (Out,'----------   --------   --------');
  end;

{*** Initialize variables ***}
procedure Initialize;
  var
    Vert : VertexType;
  begin
    for Vert := 1 to MaxVert do
      begin
        AssgnArray[Vert] := 0; AdjColorArray[Vert] := [];
        FeasibleSetArray[Vert] := [];
      end;
  end;

{***  Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I,Vert,Vertex,Vertex1,Vertex2,Chrom : word;
  procedure AddNewAdjVertex1 (Vert1,Vert2 : VertexType);
    var
      NumOfAdjVert : word;
    begin
      AdjVertArray[Vert1]^.NumOfAdjVert :=
        AdjVertArray[Vert1]^.NumOfAdjVert + 1;
      NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
      if (NumOfAdjVert>300) then
        writeln ('Vertex = ',Vert1,' NumOfAdjVert = ',NumOfAdjVert);
      AdjVertArray[Vert1]^.VertArray[NumOfAdjVert] := Vert2;
    end;

  procedure AddNewAdjVertex2 (Vert1,Vert2 : VertexType);
    var
       I : word; Vert : VertexType;
    begin
      if (Vert2< =250) then
        begin
```

```
            I := 1; Vert := Vert2;
          end
        else
          begin
            I := 2; Vert := Vert2 - 250;
          end;
        AltAdjVertArray[Vert1]^.VertSetArray[I] :=
          AltAdjVertArray[Vert1]^.VertSetArray[I]+[Vert];
      end;
    begin
      LFI1Colors := 999; SLI1Colors := 999;
      for Vert := 1 to NumOfVert do
        begin
          AdjVertArray[Vert]^.NumOfAdjVert := 0;
          AltAdjVertArray[Vert]^.VertSetArray[1] := [];
          AltAdjVertArray[Vert]^.VertSetArray[2] := [];
          DegArray[Vert] := 0;
        end;
      for I := 1 to 6 do
        readln (DataIn);
      readln (DataIn,NumOfVert); readln (DataIn); readln (DataIn,NumOfEdges);
      readln (DataIn);
      for I := 1 to NumOfVert do
        begin
          read (DataIn,Vertex,Chrom);
        end;
      readln(DataIn); readln(DataIn);
      for I := 1 to NumOfEdges do
        begin
          read (DataIn,Vertex1,Vertex2);
          AddNewAdjVertex1(Vertex1,Vertex2);
          AddNewAdjVertex1(Vertex2,Vertex1);
          AddNewAdjVertex2(Vertex1,Vertex2);
          AddNewAdjVertex2(Vertex2,Vertex1);
          DegArray[Vertex1] := DegArray[Vertex1] + 1;
          DegArray[Vertex2] := DegArray[Vertex2] + 1;
        end;
    end;

{*** Display coloring and coloring number. ***}
procedure DisplayResults (ColorNumber : ColorType);
  var
    Clr : ColorType;
    Vert : VertexType;
```

```
  begin
    writeln (Prn,'File Name = ',FileName); writeln (Prn);
    for Vert := 1 to NumOfVert do
      writeln (Prn,'Assign Vertex ',Vert,' Color ',AssgnArray[Vert]);
    writeln(Prn); writeln (Prn,'Coloring Number = ',ColorNumber); writeln (Prn);
  end;

{*** Check if Vert is in adjacency list ***}
function IsIn1 (Vert : VertexType; AdjListPtr : AdjListPtrType) : boolean;
  var
    I,NumOfAdjVert : word; FoundIt : boolean;
  begin
    NumOfAdjVert := AdjListPtr^.NumOfAdjVert; FoundIt := false; I := 1;
    while (not FoundIt) and (I<=NumOfAdjVert) do
      begin
        if (Vert=AdjListPtr^.VertArray[I]) then
          FoundIt := true;
        I := I + 1;
      end;
    IsIn1 := FoundIt;
  end;

{*** Check if Vert1 is adjacent to Vert2 ***}
function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
  var
    I : word; Vert : VertexType;
  begin
    if (Vert1<=250) then
      begin
        I := 1; Vert := Vert1;
      end
    else
      begin
        I := 2; Vert := Vert1 - 250;
      end;
    IsIn2 := Vert in AltAdjVertArray[Vert2]^.VertSetArray[I];
  end;

{*** Gen an array containg the vertex indices in largest first order ***}
procedure GenLFVertOrderArray;
  var
    I,J,K : word; Temp : VertexOrderArrayType;
  begin
    for I := 1 to NumOfVert do
```

```
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].Deg := DegArray[I];
      end;
    for I := 1 to (NumOfVert-1) do
      begin
        K := I;
        for J := (I+1) to NumOfVert do
          if (VertOrderArray[J].Deg>VertOrderArray[K].Deg) then
            K := J;
        if (K<>I) then
          begin
            Temp := VertOrderArray[I];
            VertOrderArray[I] := VertOrderArray[K];
            VertOrderArray[K] := Temp;
          end;
      end;
  end;

{*** Generate an array containg the vertex indices in smallest last order ***}
procedure GenSLVertOrderArray;
  var
    I,J,K : word; Temp : VertexOrderArrayType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].Deg := DegArray[I];
      end;
    for I := 1 to (NumOfVert-1) do
      begin
        K := I;
        for J := (I+1) to NumOfVert do
          if (VertOrderArray[J].Deg<VertOrderArray[K].Deg) then
            K := J;
        if (K<>I) then
          begin
            Temp := VertOrderArray[I];
            VertOrderArray[I] := VertOrderArray[K];
            VertOrderArray[K] := Temp;
          end;
        for J := (I+1) to NumOfVert do
          if (IsIn2(VertOrderArray[I].Vert,
            VertOrderArray[J].Vert)) then
            VertOrderArray[J].Deg := VertOrderArray[J].Deg - 1;
      end;
```

```
    for I := 1 to (NumOfVert div 2) do
      begin
        Temp := VertOrderArray[I];
        VertOrderArray[I] := VertOrderArray[NumOfVert-I+1];
        VertOrderArray[NumOfVert-I+1] := Temp;
      end;
  end;

{*** Update feasible set structure ***}
procedure UpdateFeasibleSets1 (MaxClr : ColorType);
  var
    Vert : VertexType; Color : ColorType;
  begin
    for Vert := 1 to NumOfVert do
      FeasibleSetArray[Vert] := [];
    for Vert := 1 to NumOfVert do
      for Color := 1 to MaxClr do
        if (not (Color in AdjColorArray[Vert])) then
          FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Color];
  end;

procedure UpdateFeasibleSets2 (MaxClr : ColorType);
  var
    ColorIsFeasible : boolean; Vert : VertexType; Color : ColorType;
  begin
    for Vert := 1 to NumOfVert do
        if (not (MaxClr in AdjColorArray[Vert])) then
          FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [MaxClr];
  end;

procedure UpdateFeasibleSets3 (Vertex : VertexType; Color : ColorType);
  var
    Vert : VertexType; I,NumOfAdjVert : word;
  begin
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        if (Color in FeasibleSetArray[Vert]) then
          FeasibleSetArray[Vert] := FeasibleSetArray[Vert] - [Color];
      end;
  end;

procedure UpdateFeasibleSets4 (Vertex : VertexType; Color : ColorType);
```

```
  var
    Vert : VertexType; I,NumOfAdjVert : word;
  begin
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        if (not (Color in AdjColorArray[Vert])) then
          FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Color];
      end;
  end;

procedure GenSmallestFeasibleColor (MaxClr:ColorType; Vertex:VertexType; var
Color:ColorType);
  var
    FoundIt : boolean; Clr : ColorType;
  begin
    FoundIt := false;
    Clr := 1;
    while (Clr<=MaxClr) and (not FoundIt) do
      if (Clr in FeasibleSetArray[Vertex]) then
        begin
          Color := Clr; FoundIt := true;
        end
      else
        Clr := Clr + 1;
  end;

{*** Update data structures ***}
procedure DoUpdate1 (Vert1 : VertexType; Color : ColorType);
  var
    StillIn : boolean; Vert2,Vert3 : VertexType;
    I1,NumOfAdjVert1,I2,NumOfAdjVert2 : word;
  begin
    NumOfAdjVert1 := AdjVertArray[Vert1]^.NumOfAdjVert;
    for I1 := 1 to NumOfAdjVert1 do
      begin
        Vert2 := AdjVertArray[Vert1]^.VertArray[I1]; StillIn := false;
        NumOfAdjVert2 := AdjVertArray[Vert2]^.NumOfAdjVert; I2 := 1;
        while (I2<=NumOfAdjVert2) and (not StillIn) do
          begin
            Vert3 := AdjVertArray[Vert2]^.VertArray[I2];
            if (AssgnArray[Vert3]=Color) then
              StillIn := true
```

```
            else
              I2 := I2 + 1;
          end;
        if (not StillIn) then
            AdjColorArray[Vert2] := AdjColorArray[Vert2] - [Color];
      end;
  end;

procedure DoUpdate2 (Vert1 : VertexType; Color : ColorType);
  var
    Vert2 : VertexType; I,NumOfAdjVert : word;
  begin
    NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert2 := AdjVertArray[Vert1]^.VertArray[I];
        AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Color];
      end;
  end;

procedure AttemptAnInterchange1 (MaxClr : ColorType; UpLimit : word;
                                 Vertex : VertexType; var Color : ColorType;
                                 var FoundOne : boolean);
  var
    Changed,TryedOne : boolean; Vert1 : VertexType; Clr1,Vert1Clr : ColorType;
  begin
    SaveFeasibleSetArray := FeasibleSetArray;
    SaveAdjColorArray := AdjColorArray;
    SaveAssgnArray := AssgnArray; FoundOne := false; Changed := false; I := 1;
    while (not FoundOne) and (I <= UpLimit) do
      begin
        if (Changed) then
          begin
            FeasibleSetArray := SaveFeasibleSetArray;
            AdjColorArray := SaveAdjColorArray;
          end;
        Changed := false;
        Vert1 := VertOrderArray[I].Vert;
        if (IsIn2(Vert1,Vertex)) and (FeasibleSetArray[Vert1] <> []) then
          begin
            Vert1Clr := AssgnArray[Vert1]; TryedOne := false; Clr1 := 1;
            while (Clr1 <= MaxClr) and (not FoundOne) and (not TryedOne) do
              begin
                if (Clr1 in FeasibleSetArray[Vert1]) and (Clr1 <> Vert1Clr) then
```

```
                    begin
                      TryedOne := true; Changed := true;
                      AssgnArray[Vert1] := 0;
                      DoUpDate1 (Vert1,Vert1Clr);
                      UpdateFeasibleSets4 (Vert1,Vert1Clr);
                      if (FeasibleSetArray[Vertex]<>[]) then
                        begin
                          AssgnArray[Vert1] := Clr1; DoUpdate2 (Vert1,Clr1);
                          UpdateFeasibleSets3(Vert1,Clr1);
                          FoundOne := true;
                          GenSmallestFeasibleColor (MaxClr,Vertex,Color);
                        end
                      else
                        AssgnArray[Vert1] := SaveAssgnArray[Vert1];
                    end;
                  Clr1 := Clr1 + 1;
                end;
            end;
          I := I + 1;
        end;
      if (not FoundOne) and (Changed) then
        begin
          FeasibleSetArray := SaveFeasibleSetArray;
          AdjColorArray := SaveAdjColorArray;
        end;
    end;

{*** SLFI1 implementation ***}
procedure DoLFI1Coloring;
  var
    FoundOne : boolean; I,J,NumOfAdjVert : word;
    Vert,Vertex : VertexType; Clr,Color,MaxColor : ColorType;
  begin
    GenLFVertOrderArray; Vertex := VertOrderArray[1].Vert;
    AssgnArray[Vertex] := 1;
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        AdjColorArray[Vert] := AdjColorArray[Vert] + [1];
      end;
    LFI1Colors := 1; UpdateFeasibleSets1 (LFI1Colors);
    for I := 2 to NumOfVert do
      begin
```

```
          Vertex := VertOrderArray[I].Vert; FoundOne := false;
          while (not FoundOne) do
            if (FeasibleSetArray[Vertex]<>[]) then
              begin
                FoundOne := true;
                GenSmallestFeasibleColor (I,LFI1Colors,Vertex,Color);
              end
            else
              begin
                AttemptAnInterchange1 (LFI1Colors,I-1,Vertex,Color,FoundOne);
                if (not FoundOne) then
                  begin
                    LFI1Colors := LFI1Colors + 1;
                    UpdateFeasibleSets2 (LFI1Colors);
                  end;
              end;
          AssgnArray[Vertex] := Color;
          NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
          for J := 1 to NumOfAdjVert do
            begin
              Vert := AdjVertArray[Vertex]^.VertArray[J];
              AdjColorArray[Vert] := AdjColorArray[Vert] + [Color];
            end;
          UpdateFeasibleSets3 (Vertex,Color);
        end;
    end;

{*** SSLI1 implementation ***}
procedure DoSLI1Coloring;
    var
      FoundOne : boolean; I,J,NumOfAdjVert : word;
      Vert,Vertex : VertexType; Clr,Color,MaxColor : ColorType;
    begin
      GenSLVertOrderArray; Vertex := VertOrderArray[1].Vert;
      AssgnArray[Vertex] := 1;
      NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
      for I := 1 to NumOfAdjVert do
        begin
          Vert := AdjVertArray[Vertex]^.VertArray[I];
          AdjColorArray[Vert] := AdjColorArray[Vert] + [1];
        end;
      SLI1Colors := 1; UpdateFeasibleSets1 (SLI1Colors);
      for I := 2 to NumOfVert do
        begin
```

```
        Vertex := VertOrderArray[I].Vert; FoundOne := false;
        while (not FoundOne) do
          if (FeasibleSetArray[Vertex]<>[]) then
            begin
              FoundOne := true;
              GenSmallestFeasibleColor (SLI1Colors,Vertex,Color);
            end
          else
            begin
              AttemptAnInterchange1 (SLI1Colors,I-1,Vertex,Color, FoundOne);
              if (not FoundOne) then
                begin
                  SLI1Colors := SLI1Colors + 1;
                  UpdateFeasibleSets2 (SLI1Colors);
                end;
            end;
        AssgnArray[Vertex] := Color;
        NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
        for J := 1 to NumOfAdjVert do
          begin
            Vert := AdjVertArray[Vertex]^.VertArray[J];
            AdjColorArray[Vert] := AdjColorArray[Vert] + [Color];
          end;
        UpdateFeasibleSets3 (Vertex,Color);
      end;
  end;

{*** Check AssgnArray for feasibility ***}
procedure CheckAssgnArray (ColorNum,Method : word);
  var
    I,NumOfAdjVert : word; Clr : byte; Vert1,Vert2 : VertexType;
  begin
    FeasibleArray[Method] := true; Vert1 := 1;
    while (Vert1<=NumOfVert) and (FeasibleArray[Method]) do
      begin
        for Clr := 1 to ColorNum do
          if (Clr=AssgnArray[Vert1]) then
            begin
              NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert; I := 1;
              while (I<=NumOfAdjVert) and (FeasibleArray[Method]) do
                begin
                  Vert2 := AdjVertArray[Vert1]^.VertArray[I];
                  if (Clr=AssgnArray[Vert2]) then
                    begin
```

```
                        FeasibleArray[Method] := false;
                        writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                      end
                    else
                      I := I + 1;
                  end;
              end;
          Vert1 := Vert1 + 1;
        end;
    end;

procedure GenReport (Count : word);
  var
    LineArray : array [1..85] of string[80]; Cnt,I : byte;
  begin
    reset (Out); Cnt := 0;
    while (not eof(Out)) do
      begin
        Cnt := Cnt + 1; readln (Out,LineArray[Cnt]);
      end;
    close (Out); rewrite (Out);
    for I := 1 to Cnt do
      writeln (Out,LineArray[I]);
    writeln (Out,UpperCase(FileName):11,LFI1Colors:8,SLI1Colors:11);
    writeln (Out,LFI1Time:22:2,SLI1Time:11:2);
    write (Out,' ':10);
    for I := 1 to 2 do
      write (Out,FeasibleArray[I]:11);
    writeln (Out);
    if (Count=EndGraphNum) then
      begin
        AvgLFI1Colors := AvgLFI1Colors/NumOfGraphs;
        AvgSLI1Colors := AvgSLI1Colors/NumOfGraphs;
        AvgLFI1Time := AvgLFI1Time/NumOfGraphs;
        AvgSLI1Time := AvgSLI1Time/NumOfGraphs;
        writeln (Out,'---------- ------- -------');
        writeln (Out,' Averages',AvgLFI1Colors:11:2,AvgSLI1Colors:11:2);
        writeln (Out,AvgLFI1Time:22:2,AvgSLI1Time:11:2);
      end;
    close (Out);
  end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
```

```
  var
    BegTime,EndTime : real;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Prn,'con'); rewrite (Prn); assign (Out, 'REPORT.PRN'); rewrite (Out);
  ObtainUserInfo; PrintReportHeadingsAndInitTotals; close (Out);
  for Count := BegGraphNum to EndGraphNum do
    begin
      if (Count<10) then
        begin
          FileName := FileNamePrefix + '.00'; str(Count:1,Extension);
        end
      else
        begin
          FileName := FileNamePrefix + '.0'; str(Count:2,Extension);
        end;
      FileName := FileName + Extension;
      assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName);
      reset (DataIn); writeln; writeln ('Processing ',FileName);
      ReadData; close (DataIn); Initialize;
      writeln('Working on LFI1'); GetTime (Hour1,Min1,Sec1,Frac1);
      DoLFI1Coloring;
      GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
      LFI1Time := ElapsedTime;
      AvgLFI1Time := AvgLFI1Time + LFI1Time;
      AvgLFI1Colors := AvgLFI1Colors + LFI1Colors;
      CheckAssgnArray(LFI1Colors,1);
      Initialize; writeln('Working on SLI1'); GetTime (Hour1,Min1,Sec1,Frac1);
      DoSLI1Coloring;
      GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
      SLI1Time := ElapsedTime;
      AvgSLI1Time := AvgSLI1Time + SLI1Time;
      AvgSLI1Colors := AvgSLI1Colors + SLI1Colors;
      CheckAssgnArray(SLI1Colors,2);
      writeln (UpperCase(FileName):11,LFI1Colors:8,SLI1Colors:11);
      writeln (LFI1Time:22:2,SLI1Time:11:2); write (' ':10);
      for I := 1 to 2 do
```

```
          write (FeasibleArray[I]:11);
        writeln; GenReport (Count);
      end;
end.
```

6. SVSI2.Pas -- Implementation of SLF, SLFI2, SSL, and SSLI2

```
{$M 65520,0,655360}
{*** Implements SLF, SLFI2, SSL, and SSLI2 ***}
program SVSI2;
uses DOS,CRT;
const
  MaxVert = 500;  {Max no. of vertices}
  MaxColor = 80; {Upper bound for chromatic number}
type
  String3 = string[3]; VertexType = 1..MaxVert; VertexSetType = set of 1..255;
  ColorType = 0..MaxColor+1; ColorSetType = set of ColorType;
  String11 = string[11];
  VertexOrderArrayType = record
                           Vert : VertexType; Deg : word;
                         end;
  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  NumOfAdjVert : word; VertArray : array [1..350] of word;
                end;
  ConnectedCompPtrType = ^ConnectedCompType;
  ConnectedCompType = record
                        CC : array[1..2] of VertexSetType; NumOfCCVert : word;
                        CCArray : array[1..300] of word;
                        NextPtr : ConnectedCompPtrType;
                      end;
var
  FileName,FileNamePrefix,Extension : String11;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  NumOfVert,NumOfEdges,BegGraphNum,EndGraphNum : word;
  I,Count,NumOfGraphs,LFColors,LFIColors,SLColors,SLIColors : word;
  LFTime,LFITime,SLTime,SLITime,ElapsedTime : real;
  AvgLFColors,AvgLFIColors,AvgLFTime : real;
  AvgLFITime,AvgSLTime,AvgSLITime,AvgSLColors,AvgSLIColors : real;
  VertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
  AssgnArray : array[VertexType] of ColorType;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  AltAdjVertArray : array[VertexType,1..2] of VertexSetType;
  AdjColorArray : array[VertexType] of ColorSetType;
```

```
  DegArray : array[VertexType] of word;
  FeasibleArray : array [1..4] of boolean;
  CCHeadPtr,CCRearPtr : ConnectedCompPtrType;
  StackPtr : ^word;
  DataIn,Out : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word; ConvertedName : String11;
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  var
    Code : integer;
  begin
    ClrScr; writeln ('Application:  Standard Graph Coloring'); writeln;
    writeln ('Program:  SVSI2.Pas -- Heuristic'); writeln;
    write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
    write ('Enter Number of Vertices:  '); readln (NumOfVert); writeln;
    write ('Enter Begin Graph Number: '); readln (BegGraphNum); writeln;
    write ('Enter End Graph Number: '); readln (EndGraphNum);
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Count : byte; Vert : VertexType; AdjListPtr : AdjListPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin
        new(AdjListPtr); AdjVertArray[Vert] := AdjListPtr;
      end;
    new (CCHeadPtr);
    CCHeadPtr^.NextPtr := nil; CCRearPtr := CCHeadPtr; mark (StackPtr);
    AvgLFColors := 0.0; AvgLFIColors := 0.0;
    AvgSLColors := 0.0; AvgSLIColors := 0.0;
```

```
    AvgLFTime := 0.0; AvgLFITime := 0.0;
    AvgSLTime := 0.0; AvgSLITime := 0.0;
    writeln (Out,'Application: Standard Graph Coloring');
    writeln (Out,'Program:     S_VSI2.PAS -- Heuristic');
    writeln (Out,'Ordering:    Largest First and Smallest Last'); writeln (Out);
    writeln (Out,' File Name       LF       LFI2       SL       SLI2  ');
    writeln (Out,'-----------  --------  --------  --------  --------');
  end;

procedure Initialize;
  var
    Vert : VertexType;
  begin
    for Vert := 1 to NumOfVert do
      begin
        AssgnArray[Vert] := 0; AdjColorArray[Vert] := [];
      end;
  end;


{*** Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I,Vert,Vertex,Vertex1,Vertex2,Chrom : word;
  procedure AddNewAdjVert1 (Vert1,Vert2 : VertexType);
    var
      I : word;
    begin
      AdjVertArray[Vert1]^.NumOfAdjVert :=
        AdjVertArray[Vert1]^.NumOfAdjVert + 1;
      I := AdjVertArray[Vert1]^.NumOfAdjVert;
      AdjVertArray[Vert1]^.VertArray[I] := Vert2;
    end;
  procedure AddNewAdjVert2 (Vert1,Vert2 : VertexType);
    var
       I : word; Vert : VertexType;
    begin
      if (Vert2<=250) then
        begin
          I := 1; Vert := Vert2;
        end
      else
        begin
          I := 2; Vert := Vert2 - 250;
```

```
        end;
      AltAdjVertArray[Vert1,I] := AltAdjVertArray[Vert1,I] + [Vert];
    end;
  begin
    LFColors := 999; LFIColors := 999; SLColors := 999;  SLIColors := 999;
    for Vert := 1 to NumOfVert do
      begin
        AdjVertArray[Vert]^.NumOfAdjVert := 0; AltAdjVertArray[Vert,1] := [];
        AltAdjVertArray[Vert,2] := []; DegArray[Vert] := 0;
      end;
    for I := 1 to 6 do
      readln (DataIn);
    readln (DataIn,NumOfVert); readln (DataIn);
    readln (DataIn,NumOfEdges); readln (DataIn);
    for I := 1 to NumOfVert do
      begin
        read (DataIn,Vertex,Chrom);
      end;
    readln(DataIn);  readln(DataIn);
    for I := 1 to NumOfEdges do
      begin
        read (DataIn,Vertex1,Vertex2);
        AddNewAdjVert1 (Vertex1,Vertex2); AddNewAdjVert1 (Vertex2,Vertex1);
        AddNewAdjVert2 (Vertex1,Vertex2); AddNewAdjVert2 (Vertex2,Vertex1);
        DegArray[Vertex1] := DegArray[Vertex1]+1;
        DegArray[Vertex2] := DegArray[Vertex2]+1;
      end;
  end;

function IsIn1 (Vert : VertexType; AdjListPtr : AdjListPtrType) : boolean;
  var
    I,NumOfAdjVert : word; FoundIt : boolean;
  begin
    NumOfAdjVert := AdjListPtr^.NumOfAdjVert; FoundIt := false; I := 1;
    while (not FoundIt)and (I<=NumOfAdjVert) do
      begin
        if (Vert=AdjListPtr^.VertArray[I]) then  FoundIt := true;
        I := I + 1;
      end;
    IsIn1 := FoundIt;
  end;

function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
  var
```

```
    I : word; Vert : VertexType;
  begin
    if (Vert1<=250) then
      begin I := 1; Vert := Vert1; end
    else
      begin I := 2; Vert := Vert1 - 250; end;
    IsIn2 := Vert in AltAdjVertArray[Vert2,I]
  end;

function IsIn3 (Vert : VertexType; CCPtr : ConnectedCompPtrType) : boolean;
  var
    I : word;
  begin
    if (Vert<=250) then I := 1
    else
      begin I := 2; Vert := Vert - 250; end;
    IsIn3 := Vert in CCPtr^.CC[I]
  end;

{*** Gen an array containg the vertex indices in largest first order ***}
procedure GenLFVertOrderArray;
  var
    I,J,K : word; Temp : VertexOrderArrayType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].Deg := DegArray[I];
      end;
    for I := 1 to (NumOfVert-1) do
      begin
        K := I;
        for J := (I+1) to NumOfVert do
          if (VertOrderArray[J].Deg>VertOrderArray[K].Deg) then
            K := J;
        if (K<>I) then
          begin
            Temp := VertOrderArray[I];
            VertOrderArray[I] := VertOrderArray[K];
            VertOrderArray[K] := Temp;
          end;
      end;
  end;

{*** Generate an array containg the vertex indices in smallest last order ***}
```

```
procedure GenSLVertOrderArray;
  var
    I,J,K : word; Temp : VertexOrderArrayType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].Deg := DegArray[I];
      end;
    for I := 1 to (NumOfVert-1) do
      begin
        K := I;
        for J := (I+1) to NumOfVert do
          if (VertOrderArray[J].Deg<VertOrderArray[K].Deg) then K := J;
        if (K<>I) then
          begin
            Temp := VertOrderArray[I];
            VertOrderArray[I] := VertOrderArray[K];
            VertOrderArray[K] := Temp;
          end;
        for J := (I+1) to NumOfVert do
          if (IsIn2(VertOrderArray[I].Vert,VertOrderArray[J].Vert)) then
            VertOrderArray[J].Deg := VertOrderArray[J].Deg - 1;
      end;
    for I := 1 to (NumOfVert div 2) do
      begin
        Temp := VertOrderArray[I];
        VertOrderArray[I] := VertOrderArray[NumOfVert-I+1];
        VertOrderArray[NumOfVert-I+1] := Temp;
      end;
  end;

{*** Identify smallest feasible color which can be assigned to Vertex ***}
procedure GenSmallestFeasibleColor (MaxClr:ColorType ; Vertex:VertexType; var
Color:ColorType);
  var
    FoundOne : boolean; Clr : ColorType;
  begin
    FoundOne := false; Clr := 1;
    while (Clr<=MaxClr) and (not FoundOne) do
      if (not (Clr in AdjColorArray[Vertex])) then
        begin Color := Clr; FoundOne := true; end
      else
        Clr := Clr + 1;
    if (not FoundOne) then Color := MaxClr + 1;
```

```
  end;

{*** Is Vert already in a connected component? ***}
procedure CheckForPrevProc (Vert:VertexType; var AlreadyProcessed : boolean);
  var
    CCPtr : ConnectedCompPtrType;
  begin
    AlreadyProcessed := false; CCPtr := CCHeadPtr^.NextPtr;
    while (CCPtr<>nil) and (not AlreadyProcessed) do
      if IsIn3(Vert,CCPtr) then AlreadyProcessed := true
      else CCPtr := CCPtr^.NextPtr;
  end;

{*** Build new connected component ***}
procedure AddNewConnectedComponent (Vert : VertexType);
  var
    CCPtr : ConnectedCompPtrType;
  begin
    new (CCPtr); CCPtr^.CC[1] := [];
    CCPtr^.CC[2] := []; CCPtr^.NumOfCCVert := 0;
    CCPtr^.NextPtr := nil; CCRearPtr^.NextPtr := CCPtr; CCRearPtr := CCPtr;
  end;

{*** Generate all connected components ***}
procedure GenConnectedComponents (Vertex,Vert1 : VertexType;
                                  Clr1,Clr2 : ColorType;
                                  var FoundClr1,FoundClr2,FoundAnInterchange : boolean);
  var
    I,NumOfAdjVert,NumOfCCVert : word;
    Vert2 : VertexType; AdjPtr : AdjListPtrType;
  begin
    if (IsIn2(Vert1,Vertex)) then
      if (AssgnArray[Vert1]=Clr1) then FoundClr1 := true
      else FoundClr2 := true;
    if (FoundClr1) and (FoundClr2) then
      FoundAnInterchange := false;
    if (Vert1<=250) then
      CCRearPtr^.CC[1] := CCRearPtr^.CC[1] + [Vert1]
    else CCRearPtr^.CC[2] := CCRearPtr^.CC[2] + [Vert1-250];
    CCRearPtr^.NumOfCCVert := CCRearPtr^.NumOfCCVert + 1;
    NumOfCCVert := CCRearPtr^.NumOfCCVert;
    CCRearPtr^.CCArray[NumOfCCVert] := Vert1;
    AdjPtr := AdjVertArray[Vert1];
    NumOfAdjVert := AdjPtr^.NumOfAdjVert;
```

```
    I := 1;
    while (I<=NumOfAdjVert) and (FoundAnInterchange) do
      begin
        Vert2 := AdjPtr^.VertArray[I];
        if ((AssgnArray[Vert2]=Clr1) or (AssgnArray[Vert2]=Clr2)) and
            (not IsIn3(Vert2,CCRearPtr)) then
          GenConnectedComponents(Vertex,Vert2,Clr1,Clr2,FoundClr1,
                                 FoundClr2,FoundAnInterchange);
        I := I + 1;
      end;
  end;

{*** Update data structures ***}
procedure DoUpdate1 (Vert1 : VertexType; Clr : ColorType);
  var
    StillIn : boolean; I1,I2 : word; NumOfAdjVert1,NumOfAdjVert2 : word;
    Vert2,Vert3 : VertexType; AdjPtr1,AdjPtr2 : AdjListPtrType;
  begin
    AdjPtr1:=AdjVertArray[Vert1];NumOfAdjVert1:=AdjPtr1^.NumOfAdjVert;
    I1 := 1;
    while (I1<=NumOfAdjVert1) do
      begin
        StillIn := false; Vert2 := AdjPtr1^.VertArray[I1];
        AdjPtr2 := AdjVertArray[Vert2];
        NumOfAdjVert2 := AdjPtr2^.NumOfAdjVert; I2 := 1;
        while (I2<=NumOfAdjVert2) and (not StillIn) do
          begin
            Vert3 := AdjPtr2^.VertArray[I2];
            if (AssgnArray[Vert3]=Clr) then StillIn := true
            else I2 := I2 + 1;
          end;
        if (not StillIn) then AdjColorArray[Vert2] := AdjColorArray[Vert2] - [Clr];
        I1 := I1 + 1;
      end;
  end;

procedure DoUpdate2 (Vert1 : VertexType; Clr : ColorType);
  var
    I,NumOfAdjVert : word; AdjPtr : AdjListPtrType;
  begin
    AdjPtr := AdjVertArray[Vert1]; NumOfAdjVert := AdjPtr^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      AdjColorArray[AdjPtr^.VertArray[I]] := AdjColorArray[AdjPtr^.VertArray[I]]
                                             + [Clr];
```

```
  end;

{*** Implement interchange ***}
procedure DoInterchange (Vertex : VertexType; Clr1,Clr2 : ColorType);
  var
    SwapColors:boolean; I,NumOfCCVert:word; Vert:VertexType;
    CCPtr:ConnectedCompPtrType;
  begin
    CCPtr := CCHeadPtr^.NextPtr;
    while (CCPtr<>nil) do
      begin
        SwapColors := false; NumOfCCVert := CCPtr^.NumOfCCVert; I := 1;
        while (I<=NumOfCCVert) and (not SwapColors) do
          begin
            Vert := CCPtr^.CCArray[I];
            if (AssgnArray[Vert]=Clr1) and IsIn2(Vert,Vertex) then
              SwapColors := true
            else I := I + 1;
          end;
        if (SwapColors) then
          begin
            I := 1; Vert := CCPtr^.CCArray[I];
            while (I<=NumOfCCVert) do
              begin
                Vert := CCPtr^.CCArray[I];
                if (AssgnArray[Vert]=Clr1) then
                  begin
                    AssgnArray[Vert] := 0; DoUpdate1 (Vert,Clr1);
                    AssgnArray[Vert] := Clr2; DoUpdate2 (Vert,Clr2);
                  end
                else
                  begin
                    AssgnArray[Vert] := 0; DoUpdate1 (Vert,Clr2);
                    AssgnArray[Vert] := Clr1; DoUpdate2 (Vert,Clr1);
                  end;
                I := I + 1;
              end;
          end;
        CCPtr := CCPtr^.NextPtr;
      end;
  end;

procedure ReclaimDynamicMemory;
  begin
```

```
    CCHeadPtr^.NextPtr := nil; CCRearPtr := CCHeadPtr; release (StackPtr);
  end;

{*** Attempt an interchange ***}
procedure AttemptAnInterchange (Vertex : VertexType; var Color : ColorType);
  var
    AlreadyProcessed,FoundAnInterchange,FoundClr1,FoundClr2 : boolean;
    I,NumOfAdjVert : word; Vert2,Vert : VertexType;
    MaxClr,Clr1,Clr2 : ColorType; AdjPtr : AdjListPtrType;
  begin
    MaxClr := Color - 1; Clr1 := 1; FoundAnInterchange := false;
    while (Clr1 <= MaxClr-1) and (not FoundAnInterchange) do
      begin
        Clr2 := Clr1 + 1;
        while (Clr2 <= MaxClr) and (not FoundAnInterchange) do
          begin
            FoundAnInterchange := true; AdjPtr := AdjVertArray[Vertex];
            NumOfAdjVert := AdjPtr^.NumOfAdjVert; I := 1;
            while (I <= NumOfAdjVert) and (FoundAnInterchange) do
              begin
                FoundClr1 := false; FoundClr2 := false;
                Vert := AdjPtr^.VertArray[I];
                if ((AssgnArray[Vert]=Clr1) or (AssgnArray[Vert]=Clr2)) then
                  begin
                    CheckForPrevProc (Vert,AlreadyProcessed);
                    if (not AlreadyProcessed) then
                      begin
                        AddNewConnectedComponent (Vert);
                        GenConnectedComponents(Vertex,Vert,Clr1,Clr2,FoundClr1,
                                               FoundClr2, FoundAnInterchange);
                      end;
                  end;
                I := I + 1;
              end;
            if (FoundAnInterchange) then
              begin DoInterchange (Vertex,Clr1,Clr2); Color := Clr1; end;
            ReclaimDynamicMemory;
            Clr2 := Clr2 + 1;
          end;
        Clr1 := Clr1 + 1;
      end;
  end;

{*** Implements SLF algorithm ***}
```

```
procedure DoLFColoring;
  var
    I : word; Vert : VertexType; Color : ColorType;
  begin
    GenLFVertOrderArray; Vert := VertOrderArray[1].Vert;
    AssgnArray[Vert] := 1; DoUpdate2 (Vert,1); LFColors := 1;
    for I := 2 to NumOfVert do
      begin
        Vert := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (LFColors,Vert, Color);
        AssgnArray[Vert] := Color; DoUpdate2 (Vert,Color);
        if (Color>LFColors) then LFColors := Color;
      end;
  end;
{*** Implements SLFI2 algorithm ***}
procedure DoLFI2Coloring;
  var
    I : word; Vert : VertexType; Color : ColorType;
  begin
    GenLFVertOrderArray; Vert := VertOrderArray[1].Vert;
    AssgnArray[Vert] := 1; DoUpdate2 (Vert,1); LFIColors := 1;
    for I := 2 to NumOfVert do
      begin
        Vert := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (LFIColors,Vert, Color);
        if (Color<=LFIColors) then AssgnArray[Vert] := Color
        else
          begin
            AttemptAnInterchange (Vert,Color); AssgnArray[Vert] := Color;
            if (Color>LFIColors) then LFIColors := Color;
          end;
        DoUpdate2 (Vert,Color);
      end;
  end;

{*** Implements SSL algorithm ***}
procedure DoSLColoring;
  var
    I : word; Vert : VertexType; Color : ColorType;
  begin
    GenSLVertOrderArray; Vert := VertOrderArray[1].Vert;
    AssgnArray[Vert] := 1; DoUpdate2 (Vert,1); SLColors := 1;
    for I := 2 to NumOfVert do
      begin
```

```
        Vert := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (SLColors,Vert, Color);
        AssgnArray[Vert] := Color; DoUpdate2 (Vert,Color);
        if (Color>SLColors) then SLColors := Color;
      end;
  end;

{*** Implements SSLI2 algorithm ***}
procedure DoSLI2Coloring;
  var
    I : word; Vert : VertexType; Color : ColorType;
  begin
    GenSLVertOrderArray; Vert := VertOrderArray[1].Vert;
    AssgnArray[Vert] := 1; DoUpdate2 (Vert,1); SLIColors := 1;
    for I := 2 to NumOfVert do
      begin
        Vert := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (SLIColors,Vert, Color);
        if (Color<=SLIColors) then AssgnArray[Vert] := Color
        else
          begin
            AttemptAnInterchange (Vert,Color);
            AssgnArray[VertOrderArray[I].Vert] := Color;
            if (Color>SLIColors) then
              SLIColors := Color;
          end;
        DoUpdate2 (Vert,Color);
      end;
  end;

{*** Check AssgnArray for feasiblity ***}
procedure CheckAssgnArray (ColorNum,Method : word);
  var
    I,NumOfAdjVert,Clr : word; Vert1,Vert2 : VertexType;
    AdjPtr : AdjListPtrType;
  begin
    FeasibleArray[Method] := true; Vert1 := 1;
    while (Vert1<=NumOfVert) and (FeasibleArray[Method]) do
      begin
        for Clr := 1 to ColorNum do
          if (Clr=AssgnArray[Vert1]) then
            begin
              AdjPtr := AdjVertArray[Vert1];
              NumOfAdjVert := AdjPtr^.NumOfAdjVert; I := 1;
```

```
                while (I<NumOfAdjVert) and (FeasibleArray[Method]) do
                  begin
                    Vert2 := AdjPtr^.VertArray[I];
                    if (Clr=AssgnArray[Vert2]) then
                      begin
                        FeasibleArray[Method] := false;
                        writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                      end
                    else I := I + 1;
                  end;
              end;
            Vert1 := Vert1 + 1;
        end;
    end;

{*** Generate report ***}
procedure GenReport (Count : word);
    var
      LineArray : array [1..85] of string[80]; Cnt,I : byte;
    begin
      reset (Out); Cnt := 0;
      while (not eof(Out)) do
        begin
          Cnt := Cnt + 1; readln (Out,LineArray[Cnt]);
        end;
      close (Out); rewrite (Out);
      for I := 1 to Cnt do
        writeln (Out,LineArray[I]);
      writeln (Out,UpperCase(FileName):11,LFColors:8,LFIColors:11,
            SLColors:11,SLIColors:11);
      writeln (Out,LFTime:22:2,LFITime:11:2,SLTime:11:2,SLITime:11:2);
      write (Out,' ':10);
      for I := 1 to 4 do
        write (Out,FeasibleArray[I]:11);
      writeln (Out);
      if (Count=EndGraphNum) then
        begin
          AvgLFColors := AvgLFColors/NumOfGraphs;
          AvgLFIColors := AvgLFIColors/NumOfGraphs;
          AvgSLColors := AvgSLColors/NumOfGraphs;
          AvgSLIColors := AvgSLIColors/NumOfGraphs;
          AvgLFTime := AvgLFTime/NumOfGraphs;
          AvgLFITime := AvgLFITime/NumOfGraphs;
          AvgSLTime := AvgSLTime/NumOfGraphs;
```

```
        AvgSLITime := AvgSLITime/NumOfGraphs;
        writeln (Out,'---------- -------- -------- -------- --------');
        writeln (Out,'  Averages',AvgLFColors:11:2,AvgLFIColors:11:2,
              AvgSLColors:11:2,AvgSLIColors:11:2);
        writeln (Out,AvgLFTime:22:2,AvgLFITime:11:2,AvgSLTime:11:2,
              AvgSLITime:11:2);
      end;
    close (Out);
  end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : real;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Out, 'REPORT.PRN'); rewrite (Out); ObtainUserInfo;
  PrintReportHeadingsAndInitTotals; close (Out);
  for Count := BegGraphNum to EndGraphNum do
    begin
      if (Count<10) then
        begin FileName := FileNamePrefix + '.00'; str(Count:1,Extension); end
      else
        begin FileName := FileNamePrefix + '.0'; str(Count:2,Extension); end;
      FileName := FileName + Extension;
      assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName);
      reset (DataIn); writeln; writeln ('Processing ',FileName);
      ReadData; close (DataIn);
      Initialize; writeln('Working on SLF'); GetTime (Hour1,Min1,Sec1,Frac1);
      DoLFColoring;
      GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
      LFTime := ElapsedTime; AvgLFTime := AvgLFTime + LFTime;
      AvgLFColors := AvgLFColors + LFColors; CheckAssgnArray(LFColors,1);
      Initialize; writeln('Working on SLFI2'); GetTime (Hour1,Min1,Sec1,Frac1);
      DoLFI2Coloring;
      GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
      LFITime := ElapsedTime; AvgLFITime := AvgLFITime + LFITime;
      AvgLFIColors := AvgLFIColors + LFIColors;
```

```
        CheckAssgnArray(LFIColors,2);
        Initialize; writeln('Working on SSL'); GetTime (Hour1,Min1,Sec1,Frac1);
        DoSLColoring;
        GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
        SLTime := ElapsedTime; AvgSLTime := AvgSLTime + SLTime;
        AvgSLColors := AvgSLColors + SLColors; CheckAssgnArray(SLColors,3);
        Initialize; writeln('Working on SLI2'); GetTime (Hour1,Min1,Sec1,Frac1);
        DoSLI2Coloring;
        GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
        SLITime := ElapsedTime;
        AvgSLITime := AvgSLITime + SLITime;
        AvgSLIColors := AvgSLIColors + SLIColors;
        CheckAssgnArray(SLIColors,4);
        writeln (UpperCase(FileName):11,LFColors:8,LFIColors:11,
                SLColors:11,SLIColors:11);
        writeln (LFTime:22:2,LFITime:11:2,SLTime:11:2,SLITime:11:2);
        write (' ':10);
        for I := 1 to 4 do
          write (FeasibleArray[I]:11);
        writeln; GenReport (Count);
    end;
end.
```

7. SVSI3.Pas -- Implementation of SDsaturI1

```
{$M 65520,0,655360}
{*** Implements SDsaturI1 ***}
program SVSI3;
uses DOS,CRT;
const
  MaxVert = 500;  {Max no. of vertices}
  MaxColor = 80;  {Upper bound for chromatic number}
type
  VertexType = 1..MaxVert; VertexSetType = set of 1..250;
  ColorType = 0..MaxColor;
  ColorSetType = set of ColorType; String11 = string[11];
  VertexOrderArrayType = record
                           Vert : VertexType; CDeg : word; WDeg : word;
                         end;
  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  NumOfAdjVert : word; VertArray : array[1..300] of VertexType;
                end;
  AltAdjListPtrType = ^AltAdjListType;
```

```
  AltAdjListType = record
                     VertSetArray : array[1..2] of VertexSetType;
                   end;
var
  FileName,FileNamePrefix,Extension : String11;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2,NumOfVert,NumOfEdges:word;
  I,Count,NumOfGraphs,BegGraphNum,EndGraphNum,DynVSI1Colors : word;
  DynVSI1Time,ElapsedTime,AvgElapsedTime : real;
  AvgDynVSI1Colors,AvgDynVSI1Time : real;
  SaveVertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
  VertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
  AssgnArray,SaveAssgnArray : array[VertexType] of ColorType;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  AltAdjVertArray : array[VertexType] of AltAdjListPtrType;
  AdjColorArray,SaveAdjColorArray : array[VertexType] of ColorSetType;
  WDegArray : array[VertexType] of word;
  FeasibleSetArray,SaveFeasibleSetArray : array[VertexType] of ColorSetType;
  FeasibleArray : array[1..5] of boolean;
  DataIn,Out,Prn : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word; ConvertedName : String11;
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  var
    Code : integer;
  begin
    ClrScr;
    writeln ('Application:  Standard Graph Coloring'); writeln;
    writeln ('Program:  SVSI3.PAS -- Heuristic'); writeln;
    write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
    write ('Enter Num of Vertices -- '); readln (NumOfVert); writeln;
    write ('Enter Begin Graph Number:  '); readln (BegGraphNum); writeln;
    write ('Enter End Graph Number:  '); readln (EndGraphNum);
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
```

```
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Vert : VertexType; AdjListPtr : AdjListPtrType;
    AltAdjListPtr : AltAdjListPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin new(AdjListPtr); AdjVertArray[Vert] := AdjListPtr; end;
    for Vert := 1 to NumOfVert do
      begin new(AltAdjListPtr); AltAdjVertArray[Vert] := AltAdjListPtr; end;
    AvgDynVSI1Colors := 0.0; AvgDynVSI1Time := 0.0;
    writeln (Out,'Application: Standard Graph Coloring');
    writeln (Out,'Program:     SVSI3.PAS -- Heuristic');
    writeln (Out,'Method:      SDsatur'); writeln (Out);
    writeln (Out,' File Name    SDsaturI1'); writeln (Out,'----------   ---------');
  end;

procedure Initialize;
  var
    Vert : VertexType;
  begin
    for Vert := 1 to MaxVert do
      begin
        AssgnArray[Vert] := 0; AdjColorArray[Vert] := [];
        FeasibleSetArray[Vert] := [];
      end;
  end;

{*** Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I,Vert,Vertex,Vertex1,Vertex2,Chrom : word;
  procedure AddNewAdjVertex1 (Vert1,Vert2 : VertexType);
    var
      NumOfAdjVert : word;
    begin
      AdjVertArray[Vert1]^.NumOfAdjVert :=
        AdjVertArray[Vert1]^.NumOfAdjVert + 1;
      NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
      AdjVertArray[Vert1]^.VertArray[NumOfAdjVert] := Vert2;
    end;
  procedure AddNewAdjVertex2 (Vert1,Vert2 : VertexType);
```

```
    var
       I : word; Vert : VertexType;
    begin
      if (Vert2<=250) then
        begin I := 1; Vert := Vert2; end
      else
        begin I := 2; Vert := Vert2 - 250; end;
      AltAdjVertArray[Vert1]^.VertSetArray[I] :=
        AltAdjVertArray[Vert1]^.VertSetArray[I]+[Vert];
    end;
  begin
    DynVSI1Colors := 999;
    for Vert := 1 to NumOfVert do
      begin
        AdjVertArray[Vert]^.NumOfAdjVert := 0;
        AltAdjVertArray[Vert]^.VertSetArray[1] := [];
        AltAdjVertArray[Vert]^.VertSetArray[2] := []; WDegArray[Vert] := 0;
      end;
    for I := 1 to 6 do
      readln (DataIn);
    readln (DataIn,NumOfVert); readln (DataIn); readln (DataIn,NumOfEdges);
    readln (DataIn);
    for I := 1 to NumOfVert do
      read (DataIn,Vertex,Chrom);
    readln(DataIn);  readln(DataIn);
    for I := 1 to NumOfEdges do
      begin
        read (DataIn,Vertex1,Vertex2);
        AddNewAdjVertex1(Vertex1,Vertex2);
        AddNewAdjVertex1(Vertex2,Vertex1);
        AddNewAdjVertex2(Vertex1,Vertex2);
        AddNewAdjVertex2(Vertex2,Vertex1);
        WDegArray[Vertex1] := WDegArray[Vertex1] + 1;
        WDegArray[Vertex2] := WDegArray[Vertex2] + 1;
      end;
  end;

function IsIn1 (Vert : VertexType; AdjListPtr : AdjListPtrType) : boolean;
  var
    I,NumOfAdjVert : word; FoundIt : boolean;
  begin
    NumOfAdjVert := AdjListPtr^.NumOfAdjVert; FoundIt := false; I := 1;
    while (not FoundIt)and (I<=NumOfAdjVert) do
      begin
```

```
        if (Vert=AdjListPtr^.VertArray[I]) then FoundIt := true;
        I := I + 1;
      end;
    IsIn1 := FoundIt;
  end;

function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
  var
    I : word; Vert : VertexType;
  begin
    if (Vert1<=250) then
      begin I := 1; Vert := Vert1; end
    else
      begin I := 2; Vert := Vert1 - 250; end;
    IsIn2 := Vert in AltAdjVertArray[Vert2]^.VertSetArray[I];
  end;

procedure UpdateFeasibleSets1;
  var
    Vert : VertexType; Color,MaxClr : ColorType;
  begin
    MaxClr := DynVSI1Colors;
    for Vert := 1 to NumOfVert do
      FeasibleSetArray[Vert] := [];
    for Vert := 1 to NumOfVert do
      for Color := 1 to MaxClr do
        if (not (Color in AdjColorArray[Vert])) then
          FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Color];
  end;

procedure UpdateFeasibleSets2;
  var
    ColorIsFeasible : boolean; Vert : VertexType; Color,MaxClr : ColorType;
  begin
    MaxClr := DynVSI1Colors;
    for Vert := 1 to NumOfVert do
        if (not (MaxClr in AdjColorArray[Vert])) then
          FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [MaxClr];
  end;

procedure UpdateFeasibleSets3 (Vertex : VertexType; Color : ColorType);
  var
    Vert : VertexType; I,NumOfAdjVert : word;
  begin
```

```
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        if (Color in FeasibleSetArray[Vert]) then
          FeasibleSetArray[Vert] := FeasibleSetArray[Vert] - [Color];
      end;
  end;

procedure UpdateFeasibleSets4 (Vertex : VertexType; Color : ColorType);
  var
    Vert : VertexType; I,NumOfAdjVert : word;
  begin
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        if (not (Color in AdjColorArray[Vert])) then
          FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Color];
      end;
  end;

procedure GenSmallestFeasibleColor (Vertex : VertexType; var Color : ColorType);
  var
    FoundIt : boolean; Clr,MaxClr : ColorType;
  begin
    MaxClr := DynVSI1Colors; FoundIt := false; Clr := 1;
    while (Clr< =MaxClr) and (not FoundIt) do
      if (Clr in FeasibleSetArray[Vertex]) then
        begin Color := Clr; FoundIt := true; end
      else
        Clr := Clr + 1;
  end;

{*** Update data structures ***}
procedure DoUpdate1 (Vert1 : VertexType; Clr : ColorType);
  var
    StillIn : boolean; I1,I2,J : word; NumOfAdjVert1,NumOfAdjVert2 : word;
    Vert2,Vert3 : VertexType; AdjPtr1,AdjPtr2 : AdjListPtrType;
  begin
    AdjPtr1:= AdjVertArray[Vert1];NumOfAdjVert1:= AdjPtr1^.NumOfAdjVert;
    for I1 := 1 to NumOfAdjVert1 do
      begin
        Vert2 := AdjPtr1^.VertArray[I1]; J := 1;
```

```
        while (VertOrderArray[J].Vert<>Vert2) do
          J := J + 1;
        VertOrderArray[J].WDeg := VertOrderArray[J].WDeg + 1;
        AdjPtr2 := AdjVertArray[Vert2];
        NumOfAdjVert2 := AdjPtr2^.NumOfAdjVert;
        StillIn := false; I2 := 1;
        while (I2<=NumOfAdjVert2) and (not StillIn) do
          begin
            Vert3 := AdjPtr2^.VertArray[I2];
            if (AssgnArray[Vert3]=Clr) then StillIn := true
            else I2 := I2 + 1;
          end;
        if (not StillIn) then
          begin
            AdjColorArray[Vert2] := AdjColorArray[Vert2] - [Clr];
            VertOrderArray[J].CDeg := VertOrderArray[J].CDeg - 1;
          end;
      end;
  end;

procedure DoUpdate2 (Vert1 : VertexType; Clr : ColorType);
  var
    I,J,NumOfAdjVert : word; Vert2 : VertexType; AdjPtr : AdjListPtrType;
  begin
    AdjPtr := AdjVertArray[Vert1]; NumOfAdjVert := AdjPtr^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert2 := AdjPtr^.VertArray[I]; J := 1;
        while (VertOrderArray[J].Vert<>Vert2) do
          J := J + 1;
        VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
        if (not (Clr in AdjColorArray[Vert2])) then
          begin
            AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Clr];
            VertOrderArray[J].CDeg := VertOrderArray[J].CDeg + 1;
          end;
      end;
  end;

procedure AttemptAnInterchange1 (UpLimit : word; Vertex : VertexType;
                          var Color : ColorType; var FoundOne : boolean);
  var
    Changed,TryedOne : boolean; Vert1 : VertexType;
    MaxClr,Clr1,Vert1Clr : ColorType;
```

```
begin
  MaxClr := DynVSI1Colors; SaveFeasibleSetArray := FeasibleSetArray;
  SaveAdjColorArray := AdjColorArray; SaveAssgnArray := AssgnArray;
  SaveVertOrderArray := VertOrderArray; FoundOne := false;
  Changed := false; I := 1;
  while (not FoundOne) and (I <= UpLimit) do
    begin
      if (Changed) then
        begin
          FeasibleSetArray := SaveFeasibleSetArray;
          AdjColorArray := SaveAdjColorArray;
          VertOrderArray := SaveVertOrderArray;
        end;
      Changed := false; Vert1 := VertOrderArray[I].Vert;
      if (IsIn2(Vert1,Vertex)) and (FeasibleSetArray[Vert1] <> []) then
        begin
          Vert1Clr := AssgnArray[Vert1]; TryedOne := false; Clr1 := 1;
          while (Clr1 <= MaxClr) and (not FoundOne) and (not TryedOne) do
            begin
              if (Clr1 in FeasibleSetArray[Vert1]) and (Clr1 <> Vert1Clr) then
                begin
                  TryedOne := true; Changed := true;
                  AssgnArray[Vert1] := 0;
                  DoUpDate1 (Vert1,Vert1Clr);
                  UpdateFeasibleSets4 (Vert1,Vert1Clr);
                  if (FeasibleSetArray[Vertex] <> []) then
                    begin
                      AssgnArray[Vert1] := Clr1; DoUpdate2 (Vert1,Clr1);
                      UpdateFeasibleSets3(Vert1,Clr1);
                      FoundOne := true;
                      GenSmallestFeasibleColor (Vertex,Color);
                    end
                  else
                    AssgnArray[Vert1] := SaveAssgnArray[Vert1];
                end;
              Clr1 := Clr1 + 1;
            end;
        end;
      I := I + 1;
    end;
  if (not FoundOne) and (Changed) then
    begin
      FeasibleSetArray := SaveFeasibleSetArray;
      AdjColorArray := SaveAdjColorArray;
```

```
        VertOrderArray := SaveVertOrderArray;
      end;
  end;

{*** Generate an array containg the vertex indices in DynVS2 order ***}
procedure GenSDsaturVertOrderArray (I : word);
  var
    J,K : word; Temp : VertexOrderArrayType;
  begin
    K := I;
    for J := (I+1) to NumOfVert do
      if (VertOrderArray[J].CDeg>VertOrderArray[K].CDeg) then K := J
      else if (VertOrderArray[J].CDeg=VertOrderArray[K].CDeg) and
            (VertOrderArray[J].WDeg>VertOrderArray[K].WDeg) then K := J;
    if (K<>I) then
      begin
        Temp := VertOrderArray[I]; VertOrderArray[I] := VertOrderArray[K];
        VertOrderArray[K] := Temp;
      end;
  end;

procedure UpdateArrays (I : word; Color : ColorType);
  var
    J,NumOfAdjVert : word; Vert1,Vert2 : VertexType; Clr : ColorType;
  begin
    Vert1 := VertOrderArray[I].Vert; AssgnArray[Vert1] := Color;
    for J := 1 to NumOfVert do
      begin
        Vert2 := VertOrderArray[J].Vert;
        if IsIn2(Vert2,Vert1) then
          begin
            VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
            if (not (Color in AdjColorArray[Vert2])) then
              begin
                AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Color];
                VertOrderArray[J].CDeg := VertOrderArray[J].CDeg + 1;
              end;
          end;
      end;
  end;

{*** Implements the SDsaturI1 algorithm ***}
procedure DoSDsaturI1Coloring;
  var
```

```
    FoundOne : boolean; I : word; Vert : VertexType; Color : ColorType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].CDeg := 0;
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    GenSDsaturVertOrderArray (1); UpdateArrays(1,1);
    DynVSI1Colors := 1; UpdateFeasibleSets1;
    for I := 2 to NumOfVert do
      begin
        GenSDsaturVertOrderArray (I); Vert := VertOrderArray[I].Vert;
        FoundOne := false;
        while (not FoundOne) do
          begin
            if (FeasibleSetArray[Vert]<>[]) then
              begin
                FoundOne := true; GenSmallestFeasibleColor (Vert,Color);
              end
            else
              begin
                AttemptAnInterchange1 (I-1,Vert,Color,FoundOne);
                if (not FoundOne) then
                  begin
                   DynVSI1Colors := DynVSI1Colors + 1; UpdateFeasibleSets2;
                  end;
              end;
          end;
        UpdateArrays (I,Color); UpdateFeasibleSets3 (Vert,Color);
      end;
  end;

{*** Check AssgnArray for feasibility ***}
procedure CheckAssgnArray (ColorNum,Method : word);
  var
    I,NumOfAdjVert : word; Clr : byte; Vert1,Vert2 : VertexType;
  begin
    FeasibleArray[Method] := true; Vert1 := 1;
    while (Vert1<=NumOfVert) and (FeasibleArray[Method]) do
      begin
        for Clr := 1 to ColorNum do
          if (Clr=AssgnArray[Vert1]) then
            begin
              NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert; I := 1;
```

```
                while (I<=NumOfAdjVert) and (FeasibleArray[Method]) do
                  begin
                    Vert2 := AdjVertArray[Vert1]^.VertArray[I];
                    if (Clr=AssgnArray[Vert2]) then
                      begin
                        FeasibleArray[Method] := false;
                        writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                      end
                    else
                      I := I + 1;
                  end;
              end;
            Vert1 := Vert1 + 1;
        end;
    end;

{*** Generate report ***}
procedure GenReport (Count : word);
    var
      LineArray : array [1..85] of string[80]; Cnt,I : byte;
    begin
      reset (Out); Cnt := 0;
      while (not eof(Out)) do
        begin  Cnt := Cnt + 1; readln (Out,LineArray[Cnt]); end;
      close (Out); rewrite (Out);
      for I := 1 to Cnt do
        writeln (Out,LineArray[I]);
      writeln (Out,UpperCase(FileName):11,DynVSI1Colors:8);
      writeln (Out,DynVSI1Time:22:2); write (Out,' ':10);
      for I := 1 to 1 do
        write (Out,FeasibleArray[I]:11);
      writeln (Out);
      if (Count=EndGraphNum) then
        begin
          AvgDynVSI1Colors := AvgDynVSI1Colors/NumOfGraphs;
          AvgDynVSI1Time := AvgDynVSI1Time/NumOfGraphs;
          writeln (Out,'----------  --------');
          writeln (Out,'  Averages',AvgDynVSI1Colors:11:2);
          writeln (Out,AvgDynVSI1Time:22:2);
        end;
      close (Out);
    end;

{*** Calculate the ellapsed time between two check points. ***}
```

```
procedure CalcElapsedTime;
  var
    BegTime,EndTime : real;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Prn,'con'); rewrite (Prn); assign (Out, 'REPORT.PRN'); rewrite (Out);
  ObtainUserInfo; PrintReportHeadingsAndInitTotals; close (Out);
  for Count := BegGraphNum to EndGraphNum do
    begin
      if (Count<10) then
        begin FileName := FileNamePrefix + '.00'; str(Count:1,Extension); end
      else
        begin FileName := FileNamePrefix + '.0'; str(Count:2,Extension); end;
      FileName := FileName + Extension;
      assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName);
      reset (DataIn); writeln;
      writeln ('Processing ',FileName); ReadData; close (DataIn);
      Initialize;writeln('Working on DynVSI1');GetTime(Hour1,Min1,Sec1,Frac1);
      DoSDsaturI1Coloring;GetTime(Hour2,Min2,Sec2,Frac2);CalcElapsedTime;
      DynVSI1Time := ElapsedTime;
      AvgDynVSI1Time := AvgDynVSI1Time + DynVSI1Time;
      AvgDynVSI1Colors := AvgDynVSI1Colors + DynVSI1Colors;
      CheckAssgnArray(DynVSI1Colors,1);
      writeln (UpperCase(FileName):11,DynVSI1Colors:8);
      writeln (DynVSI1Time:22:2);
      write (' ':10);
      for I := 1 to 1 do
        write (FeasibleArray[I]:11);
      writeln;
      GenReport (Count);
    end;
end.
```

8. SVSI4.Pas -- Implementation of SDsatur and SDsaturI2

```
{$M 65520,0,655360}
{*** Implements SDsatur and SDsaturI2 ***}
program VSI2;
```

```
uses DOS,CRT;
const
  MaxVert = 500; {Max no. of vertices}
  MaxColor = 80; {Upper bound for chromatic number}
type
  VertexType = 1..MaxVert; VertexSetType = set of 1..255;
  ColorType = 0..MaxColor+1;
  ColorSetType = set of ColorType; String11 = string[11];
  VertexOrderArrayType = record
                           Vert : VertexType; CDeg : word; WDeg : word;
                         end;
  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  NumOfAdjVert : word; VertArray : array [1..350] of word;
                end;

  ConnectedCompPtrType = ^ConnectedCompType;
  ConnectedCompType = record
                        CC : array[1..2] of VertexSetType; NumOfCCVert : word;
                        CCArray : array[1..300] of word;
                        NextPtr : ConnectedCompPtrType;
                      end;
var
  FileName,FileNamePrefix,Extension : String11;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2,NumOfVert,NumOfEdges:word;
  I,Count,NumOfGraphs,DynVSColors,DynVSI2Colors : word;
  BegGraphNum,EndGraphNum : word;
  DynVSTime,DynVSI2Time,ElapsedTime,AvgDynVSColors : real;
  AvgDynVSTime,AvgDynVSI2Time,AvgDynVSI2Colors : real;
  VertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
  AssgnArray : array[VertexType] of ColorType;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  AltAdjVertArray : array[VertexType,1..2] of VertexSetType;
  AdjColorArray : array[VertexType] of ColorSetType;
  WDegArray : array[VertexType] of word;
  FeasibleArray : array [1..4] of boolean;
  CCHeadPtr,CCRearPtr : ConnectedCompPtrType;
  StackPtr : ^word;
  DataIn,Out : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word; ConvertedName : String11;
```

```
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  var
    Code : integer;
  begin
    ClrScr; writeln ('Application:  Standard Graph Coloring'); writeln;
    writeln ('Program:  SVSI4.PAS -- Heuristic'); writeln;
    write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
    write ('Enter Number of Vertices:  '); readln (NumOfVert); writeln;
    write ('Enter Begin Graph Number: '); readln (BegGraphNum); writeln;
    write ('Enter End Graph Number: '); readln (EndGraphNum);
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Count : byte; Vert : VertexType; AdjListPtr : AdjListPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin
        new(AdjListPtr); AdjVertArray[Vert] := AdjListPtr;
      end;
    new (CCHeadPtr); CCHeadPtr^.NextPtr := nil;
    CCRearPtr := CCHeadPtr; mark (StackPtr);
    AvgDynVSColors := 0.0; AvgDynVSI2Colors := 0.0;
    AvgDynVSTime := 0.0; AvgDynVSI2Time := 0.0;
    writeln (Out,'Application: Standard Graph Coloring');
    writeln (Out,'Program:     SVSI4.Pas -- Heuristic');
    writeln (Out,'Ordering:    SDsatur'); writeln (Out);
    writeln (Out,' File Name     SDsatur   SDsaturI2');
    writeln (Out,'-----------  --------  ---------');
  end;

procedure Initialize;
  var
    Vert : VertexType;
```

```
  begin
    for Vert := 1 to NumOfVert do
      begin AssgnArray[Vert] := 0; AdjColorArray[Vert] := []; end;
  end;

{*** Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I,Vert,Vertex,Vertex1,Vertex2,Chrom : word;
  procedure AddNewAdjVert1 (Vert1,Vert2 : VertexType);
    var
      I : word;
    begin
      AdjVertArray[Vert1]^.NumOfAdjVert :=
        AdjVertArray[Vert1]^.NumOfAdjVert + 1;
      I := AdjVertArray[Vert1]^.NumOfAdjVert;
      AdjVertArray[Vert1]^.VertArray[I] := Vert2;
    end;
  procedure AddNewAdjVert2 (Vert1,Vert2 : VertexType);
    var
       I : word; Vert : VertexType;
    begin
      if (Vert2<=250) then
        begin I := 1; Vert := Vert2; end
      else
        begin I := 2; Vert := Vert2 - 250; end;
      AltAdjVertArray[Vert1,I] := AltAdjVertArray[Vert1,I] + [Vert];
    end;
  begin
    DynVSColors := 999; DynVSI2Colors := 999;
    for Vert := 1 to NumOfVert do
      begin
        AdjVertArray[Vert]^.NumOfAdjVert := 0; AltAdjVertArray[Vert,1] := [];
        AltAdjVertArray[Vert,2] := []; WDegArray[Vert] := 0;
      end;
    for I := 1 to 6 do
      readln (DataIn);
    readln (DataIn,NumOfVert); readln (DataIn);
    readln (DataIn,NumOfEdges); readln (DataIn);
    for I := 1 to NumOfVert do
      read (DataIn,Vertex,Chrom);
    readln(DataIn);  readln(DataIn);
    for I := 1 to NumOfEdges do
      begin
```

```
          read (DataIn,Vertex1,Vertex2);
          AddNewAdjVert1 (Vertex1,Vertex2); AddNewAdjVert1 (Vertex2,Vertex1);
          AddNewAdjVert2 (Vertex1,Vertex2); AddNewAdjVert2 (Vertex2,Vertex1);
          WDegArray[Vertex1] := WDegArray[Vertex1] + 1;
          WDegArray[Vertex2] := WDegArray[Vertex2] + 1;
        end;
  end;

function IsIn1 (Vert : VertexType; AdjListPtr : AdjListPtrType) : boolean;
  var
    I,NumOfAdjVert : word; FoundIt : boolean;
  begin
    NumOfAdjVert := AdjListPtr^.NumOfAdjVert; FoundIt := false; I := 1;
    while (not FoundIt)and (I<=NumOfAdjVert) do
      begin
        if (Vert=AdjListPtr^.VertArray[I]) then FoundIt := true;
        I := I + 1;
      end;
    IsIn1 := FoundIt;
  end;

function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
  var
    I : word; Vert : VertexType;
  begin
    if (Vert1<=250) then
      begin I := 1; Vert := Vert1; end
    else
      begin I := 2; Vert := Vert1 - 250; end;
    IsIn2 := Vert in AltAdjVertArray[Vert2,I]
  end;

function IsIn3 (Vert : VertexType; CCPtr : ConnectedCompPtrType) : boolean;
  var
    I : word;
  begin
    if (Vert<=250) then I := 1
    else
      begin I := 2; Vert := Vert - 250; end;
    IsIn3 := Vert in CCPtr^.CC[I]
  end;

procedure GenSmallestFeasibleColor (MaxClr:ColorType; Vertex:VertexType; var
Color:ColorType);
```

```
  var
    FoundOne : boolean; Clr : ColorType;
  begin
    FoundOne := false; Clr := 1;
    while (Clr<=MaxClr) and (not FoundOne) do
      if (not (Clr in AdjColorArray[Vertex])) then
        begin Color := Clr; FoundOne := true; end
      else Clr := Clr + 1;
    if (not FoundOne) then Color := MaxClr + 1;
  end;

procedure CheckForPrevProc (Vert:VertexType; var AlreadyProcessed : boolean);
  var
    CCPtr : ConnectedCompPtrType;
  begin
    AlreadyProcessed := false; CCPtr := CCHeadPtr^.NextPtr;
    while (CCPtr<>nil) and (not AlreadyProcessed) do
      if IsIn3(Vert,CCPtr) then AlreadyProcessed := true
      else CCPtr := CCPtr^.NextPtr;
  end;

procedure AddNewConnectedComponent (Vert : VertexType);
  var
    CCPtr : ConnectedCompPtrType;
  begin
    new (CCPtr); CCPtr^.CC[1] := [];
    CCPtr^.CC[2] := []; CCPtr^.NumOfCCVert := 0;
    CCPtr^.NextPtr := nil; CCRearPtr^.NextPtr := CCPtr; CCRearPtr := CCPtr;
  end;

procedure GenConnectedComponents (Vertex,Vert1 : VertexType;
                                  Clr1,Clr2 : ColorType;
                                  var FoundClr1,FoundClr2,FoundAnInterchange : boolean);
  var
    I,NumOfAdjVert,NumOfCCVert : word; Vert2 : VertexType;
    AdjPtr : AdjListPtrType;
  begin
    if (IsIn2(Vert1,Vertex)) then
      if (AssgnArray[Vert1]=Clr1) then FoundClr1 := true
      else FoundClr2 := true;
    if (FoundClr1) and (FoundClr2) then FoundAnInterchange := false;
    if (Vert1<=250) then CCRearPtr^.CC[1] := CCRearPtr^.CC[1] + [Vert1]
    else CCRearPtr^.CC[2] := CCRearPtr^.CC[2] + [Vert1-250];
    CCRearPtr^.NumOfCCVert := CCRearPtr^.NumOfCCVert + 1;
```

```
    NumOfCCVert := CCRearPtr^.NumOfCCVert;
    CCRearPtr^.CCArray[NumOfCCVert] := Vert1;
    AdjPtr := AdjVertArray[Vert1];
    NumOfAdjVert := AdjPtr^.NumOfAdjVert; I := 1;
    while (I<=NumOfAdjVert) and (FoundAnInterchange) do
      begin
        Vert2 := AdjPtr^.VertArray[I];
        if ((AssgnArray[Vert2]=Clr1) or (AssgnArray[Vert2]=Clr2)) and
            (not IsIn3(Vert2,CCRearPtr)) then
          GenConnectedComponents(Vertex,Vert2,Clr1,Clr2,FoundClr1,
                                 FoundClr2,FoundAnInterchange);
        I := I + 1;
      end;
  end;

{*** Update data structures ***}
procedure DoUpdate1 (Vert1 : VertexType; Clr : ColorType);
  var
    StillIn : boolean; I1,I2,J : word; NumOfAdjVert1,NumOfAdjVert2 : word;
    Vert2,Vert3 : VertexType; AdjPtr1,AdjPtr2 : AdjListPtrType;
  begin
    AdjPtr1:=AdjVertArray[Vert1];NumOfAdjVert1:=AdjPtr1^.NumOfAdjVert;
    for I1 := 1 to NumOfAdjVert1 do
      begin
        Vert2 := AdjPtr1^.VertArray[I1]; J := 1;
        while (VertOrderArray[J].Vert<>Vert2) do
          J := J + 1;
        VertOrderArray[J].WDeg := VertOrderArray[J].WDeg + 1;
        AdjPtr2 := AdjVertArray[Vert2];
        NumOfAdjVert2 := AdjPtr2^.NumOfAdjVert; StillIn := false; I2 := 1;
        while (I2<=NumOfAdjVert2) and (not StillIn) do
          begin
            Vert3 := AdjPtr2^.VertArray[I2];
            if (AssgnArray[Vert3]=Clr) then StillIn := true
            else I2 := I2 + 1;
          end;
        if (not StillIn) then
          begin
            AdjColorArray[Vert2] := AdjColorArray[Vert2] - [Clr];
            VertOrderArray[J].CDeg := VertOrderArray[J].CDeg - 1;
          end;
      end;
  end;
```

```
procedure DoUpdate2 (Vert1 : VertexType; Clr : ColorType);
  var
    I,J,NumOfAdjVert : word; Vert2 : VertexType; AdjPtr : AdjListPtrType;
  begin
    AdjPtr := AdjVertArray[Vert1]; NumOfAdjVert := AdjPtr^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert2 := AdjPtr^.VertArray[I]; J := 1;
        while (VertOrderArray[J].Vert<>Vert2) do
          J := J + 1;
        VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
        if (not (Clr in AdjColorArray[Vert2])) then
          begin
            AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Clr];
            VertOrderArray[J].CDeg := VertOrderArray[J].CDeg + 1;
          end;
      end;
end;

procedure DoInterchange (Vertex : VertexType; Clr1,Clr2 : ColorType);
  var
    SwapColors : boolean; I,NumOfCCVert : word;
    Vert : VertexType; CCPtr : ConnectedCompPtrType;
  begin
    CCPtr := CCHeadPtr^.NextPtr;
    while (CCPtr<>nil) do
      begin
        SwapColors := false; NumOfCCVert := CCPtr^.NumOfCCVert; I := 1;
        while (I<=NumOfCCVert) and (not SwapColors) do
          begin
            Vert := CCPtr^.CCArray[I];
            if (AssgnArray[Vert]=Clr1) and IsIn2(Vert,Vertex) then
              SwapColors := true
            else I := I + 1;
          end;
        if (SwapColors) then
          begin
            I := 1; Vert := CCPtr^.CCArray[I];
            while (I<=NumOfCCVert) do
              begin
                Vert := CCPtr^.CCArray[I];
                if (AssgnArray[Vert]=Clr1) then
                  begin
                    AssgnArray[Vert] := 0; DoUpdate1 (Vert,Clr1);
```

```
                    AssgnArray[Vert] := Clr2; DoUpdate2 (Vert,Clr2);
                  end
                else
                  begin
                    AssgnArray[Vert] := 0; DoUpdate1 (Vert,Clr2);
                    AssgnArray[Vert] := Clr1; DoUpdate2 (Vert,Clr1);
                  end;
                I := I + 1;
              end;
          end;
        CCPtr := CCPtr^.NextPtr;
      end;
  end;

procedure ReclaimDynamicMemory;
  begin
    CCHeadPtr^.NextPtr := nil; CCRearPtr := CCHeadPtr; release (StackPtr);
  end;

procedure AttemptAnInterchange (Vertex : VertexType; var Color : ColorType);
  var
    AlreadyProcessed,FoundAnInterchange,FoundClr1,FoundClr2 : boolean;
    I,NumOfAdjVert : word; Vert2,Vert : VertexType;
    MaxClr,Clr1,Clr2 : ColorType; AdjPtr : AdjListPtrType;
  begin
    MaxClr := Color - 1; Clr1 := 1; FoundAnInterchange := false;
    while (Clr1<=MaxClr-1) and (not FoundAnInterchange) do
      begin
        Clr2 := Clr1 + 1;
        while (Clr2<=MaxClr) and (not FoundAnInterchange) do
          begin
            FoundAnInterchange := true; AdjPtr := AdjVertArray[Vertex];
            NumOfAdjVert := AdjPtr^.NumOfAdjVert; I := 1;
            while (I<=NumOfAdjVert) and (FoundAnInterchange) do
              begin
                FoundClr1 := false; FoundClr2 := false;
                Vert := AdjPtr^.VertArray[I];
                if ((AssgnArray[Vert]=Clr1) or (AssgnArray[Vert]=Clr2)) then
                  begin
                    CheckForPrevProc (Vert,AlreadyProcessed);
                    if (not AlreadyProcessed) then
                      begin
                        AddNewConnectedComponent (Vert);
                        GenConnectedComponents(Vertex,Vert,Clr1,Clr2,FoundClr1,
```

```
                                        FoundClr2,FoundAnInterchange);
                    end;
                  end;
                I := I + 1;
              end;
            if (FoundAnInterchange) then
              begin DoInterchange (Vertex,Clr1,Clr2); Color := Clr1; end;
            ReclaimDynamicMemory; Clr2 := Clr2 + 1;
          end;
        Clr1 := Clr1 + 1;
      end;
  end;

{*** Generate an array containg the vertex indices in DynVS2 order ***}
procedure GenSDsaturVertOrderArray (I : word);
  var
    J,K : word; Temp : VertexOrderArrayType;
  begin
    K := I;
    for J := (I+1) to NumOfVert do
      if (VertOrderArray[J].CDeg>VertOrderArray[K].CDeg) then K := J
      else if (VertOrderArray[J].CDeg=VertOrderArray[K].CDeg) and
           (VertOrderArray[J].WDeg>VertOrderArray[K].WDeg) then K := J;
    if (K<>I) then
      begin
        Temp := VertOrderArray[I]; VertOrderArray[I] := VertOrderArray[K];
        VertOrderArray[K] := Temp;
      end;
  end;

procedure UpdateArrays (I : word; Color : ColorType);
  var
    J,NumOfAdjVert : word; Vert1,Vert2 : VertexType; Clr : ColorType;
  begin
    Vert1 := VertOrderArray[I].Vert; AssgnArray[Vert1] := Color;
    for J := 1 to NumOfVert do
      begin
        Vert2 := VertOrderArray[J].Vert;
        if IsIn2(Vert2,Vert1) then
          begin
            VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
            if (not (Color in AdjColorArray[Vert2])) then
              begin
                AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Color];
```

```
                VertOrderArray[J].CDeg := VertOrderArray[J].CDeg + 1;
              end;
          end;
      end;
  end;

{*** Implement SDsatur algorithm ***}
procedure DoSDsaturColoring;
  var
    I : word; Vert : VertexType; Color : ColorType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].CDeg := 0;
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    GenSDsaturVertOrderArray (1); UpdateArrays(1,1); DynVSColors := 1;
    for I := 2 to NumOfVert do
      begin
        GenSDsaturVertOrderArray (I); Vert := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (DynVSColors,Vert, Color);
        UpdateArrays(I,Color);
        if (Color>DynVSColors) then DynVSColors := Color;
      end;
  end;

{*** Implement SDsaturI2 algorithm ***}
procedure DoSDsaturI2Coloring;
  var
    I : word; Vert : VertexType; Color : ColorType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].CDeg := 0;
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    GenSDsaturVertOrderArray (1); UpdateArrays(1,1); DynVSI2Colors := 1;
    for I := 2 to NumOfVert do
      begin
        GenSDsaturVertOrderArray (I); Vert := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (DynVSI2Colors,Vert,Color);
        if (Color>DynVSI2Colors) then AttemptAnInterchange (Vert,Color);
        UpdateArrays(I,Color);
        if (Color>DynVSI2Colors) then DynVSI2Colors := Color;
```

```
      end;
  end;

{*** Check AssgnArray for feasibility ***}
procedure CheckAssgnArray (ColorNum,Method : word);
  var
    I,NumOfAdjVert,Clr : word; Vert1,Vert2 : VertexType; AdjPtr :
AdjListPtrType;
  begin
    FeasibleArray[Method] := true; Vert1 := 1;
    while (Vert1<=NumOfVert) and (FeasibleArray[Method]) do
      begin
        if (AssgnArray[Vert1]<=0) or (AssgnArray[Vert1]>ColorNum) then
          begin
            writeln ('Assgn Array for Vert ',Vert1,' is invalid.');
            FeasibleArray[Method] := false;
          end;
        for Clr := 1 to ColorNum do
          if (Clr=AssgnArray[Vert1]) then
            begin
              AdjPtr := AdjVertArray[Vert1];
              NumOfAdjVert := AdjPtr^.NumOfAdjVert;
              I := 1;
              while (I<NumOfAdjVert) and (FeasibleArray[Method]) do
                begin
                  Vert2 := AdjPtr^.VertArray[I];
                  if (Clr=AssgnArray[Vert2]) then
                    begin
                      FeasibleArray[Method] := false;
                      writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                    end
                  else
                    I := I + 1;
                end;
            end;
        Vert1 := Vert1 + 1;
      end;
  end;

{*** Generate report ***}
procedure GenReport (Count : word);
  var
    LineArray : array [1..85] of string[80]; Cnt,I : byte;
  begin
```

```
    reset (Out); Cnt := 0;
    while (not eof(Out)) do
      begin Cnt := Cnt + 1; readln (Out,LineArray[Cnt]); end;
    close (Out); rewrite (Out);
    for I := 1 to Cnt do
      writeln (Out,LineArray[I]);
    writeln (Out,UpperCase(FileName):11,DynVSColors:8,DynVSI2Colors:11);
    writeln (Out,DynVSTime:22:2,DynVSI2Time:11:2); write (Out,' ':10);
    for I := 1 to 2 do
      write (Out,FeasibleArray[I]:11);
    writeln (Out);
    if (Count=EndGraphNum) then
      begin
        AvgDynVSColors := AvgDynVSColors/NumOfGraphs;
        AvgDynVSI2Colors := AvgDynVSI2Colors/NumOfGraphs;
        AvgDynVSTime := AvgDynVSTime/NumOfGraphs;
        AvgDynVSI2Time := AvgDynVSI2Time/NumOfGraphs;
        writeln (Out,'---------- -------- --------');
        writeln (Out,' Averages',AvgDynVSColors:11:2,AvgDynVSI2Colors:11:2);
        writeln (Out,AvgDynVSTime:22:2,AvgDynVSI2Time:11:2);
      end;
    close (Out);
  end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : real;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Out, 'REPORT.PRN'); rewrite (Out); ObtainUserInfo;
  PrintReportHeadingsAndInitTotals; close (Out);
  for Count := BegGraphNum to EndGraphNum do
    begin
      if (Count<10) then
        begin FileName := FileNamePrefix + '.00'; str(Count:1,Extension); end
      else
        begin FileName := FileNamePrefix + '.0'; str(Count:2,Extension); end;
```

```
        FileName := FileName + Extension;
        assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName);
        reset (DataIn); writeln; writeln ('Processing ',FileName);
        ReadData; close (DataIn);
        Initialize; writeln('Working on SDsatur'); GetTime (Hour1,Min1,Sec1,Frac1);
        DoSDsaturColoring; GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
        DynVSTime := ElapsedTime;
        AvgDynVSTime := AvgDynVSTime + DynVSTime;
        AvgDynVSColors := AvgDynVSColors + DynVSColors;
        CheckAssgnArray(DynVSColors,1);
        Initialize; writeln('Working on DynVSI2'); GetTime (Hour1,Min1,Sec1,Frac1);
        DoSDsaturI2Coloring; GetTime(Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
        DynVSI2Time := ElapsedTime;
        AvgDynVSI2Time := AvgDynVSI2Time + DynVSI2Time;
        AvgDynVSI2Colors := AvgDynVSI2Colors + DynVSI2Colors;
        CheckAssgnArray(DynVSI2Colors,2);
        writeln (UpperCase(FileName):11,DynVSColors:8,DynVSI2Colors:11);
        writeln (DynVSTime:22:2,DynVSI2Time:11:2);
        write (' ':10);
        for I := 1 to 2 do
          write (FeasibleArray[I]:11);
        writeln;
        GenReport (Count);
      end;
end.
```

9. CRLF.Pas -- Implementation of CRLF

```
{$M 65520,0,655360}
{*** Implements CRLF ***}
program CRLF;
uses DOS,CRT;
const
  MaxVert = 1000; {Max no. of vertices}
  MaxColor = 200; {Upper bound for chromatic number}
type
  String2 = string[2]; VertexType = 1..MaxVert; VertexSetType = set of 1..255;
  ColorType = 0..MaxColor; ColorSetType = set of ColorType;
  String11 = string[11];
  VertexStatsArrayType = record
                           Vert : VertexType; LB : byte; Chrom : byte;
                           UChromDeg : word; UDeg : word; U1ChromDeg : word;
                           U1Deg : word; U2ChromDeg : word; U2Deg : word;
                         end;
```

```
  AltAdjListPtrType = ^AltAdjListType;
  AltAdjListType = record
                     VertSetArray : array[1..4] of VertexSetType;
                   end;
  VertListPtrType = ^VertListType;
  VertListType = record
                   Vert : VertexType; UNextPtr : VertListPtrType;
                   U1NextPtr : VertListPtrType;
                 end;
  CandidateArrayPtrType = ^CandidateArrayType;
  CandidateArrayType = record
                         NumOfCandidates : word;
                         CandidateArray : array[1..700] of word;
                       end;
var
  Feasible : boolean;
  FileName,FileNamePrefix,Extension : String11;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  NumOfVert,NumOfUVert,NumOfU1Vert,MaxClr : word;
  I,Count,NumOfGraphs,BegGraphNum,EndGraphNum,RLFColors,MinLB:word;
  NumOfEdges : longint;
  ElapsedTime,AvgElapsedTime,RLFTime,AvgRLFColors,AvgRLFTime : real;
  VertStatsArray : array[1..MaxVert] of VertexStatsArrayType;
  AssgnArray : array[VertexType] of ColorSetType;
  AltAdjVertArray : array[VertexType] of AltAdjListPtrType;
  VertListHeadPtr : VertListPtrType;
  CandidateArray1Ptr,CandidateArray2Ptr : CandidateArrayPtrType;
  DataIn,Out,Prn : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word; ConvertedName : String11;
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  var
    Code : integer;
```

```
  begin
    ClrScr; writeln ('Application:  Composite Graph Coloring'); writeln;
    writeln ('Program:  CRLF.PAS -- Heuristic'); writeln;
    write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
    write ('Enter Num of Vertices -- '); readln (NumOfVert); writeln;
    write ('Enter Begin Graph Number:  '); readln (BegGraphNum); writeln;
    write ('Enter End Graph Number:  '); readln (EndGraphNum);
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Count : byte; Vert : VertexType; AltAdjListPtr : AltAdjListPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin
        new(AltAdjListPtr); AltAdjVertArray[Vert] := AltAdjListPtr;
      end;
    new (VertListHeadPtr);
    VertListHeadPtr^.UNextPtr := nil; VertListHeadPtr^.U1NextPtr := nil;
    new (CandidateArray1Ptr); new (CandidateArray2Ptr);
    AvgRLFColors := 0.0; AvgRLFTime := 0.0;
    writeln (Out,'Application: Composite Graph Coloring');
    writeln (Out,'Program:     CRLF.PAS -- Heuristic');
    writeln (Out,'Method:      RLF'); writeln (Out);
    writeln (Out,' File Name       RLF  ');
    writeln (Out,'----------   --------');
  end;

procedure Initialize;
  var
    I : VertexType; VertListPtr,VertListRearPtr : VertListPtrType;
  begin
    VertListRearPtr := VertListHeadPtr;
    for I := 1 to NumOfVert do
      begin
        new(VertListPtr); VertListPtr^.Vert := I; VertListPtr^.UNextPtr := nil;
        VertListPtr^.U1NextPtr := nil; VertListRearPtr^.UNextPtr := VertListPtr;
        VertListRearPtr := VertListPtr;
      end;
    for I := 1 to NumOfVert do
      AssgnArray[I] := [];
  end;
```

```
{*** Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I : longint; Vert,Vertex,Vertex1,Vertex2,Chrom : word;
  procedure AddNewAdjVert2 (Vert1,Vert2 : VertexType);
    var
      I : word; Vert : VertexType;
    begin
      case Vert2 of
        1..250    : begin I := 1; Vert := Vert2; end;
        251..500  : begin I := 2; Vert := Vert2 - 250; end;
        501..750  : begin I := 3; Vert := Vert2 - 500; end;
        751..1000 : begin I := 4; Vert := Vert2 - 750; end;
      end;
      AltAdjVertArray[Vert1]^.VertSetArray[I] :=
        AltAdjVertArray[Vert1]^.VertSetArray[I]+[Vert];
    end;
  begin
    RLFColors := 999;
    for Vert := 1 to NumOfVert do
      begin
        AltAdjVertArray[Vert]^.VertSetArray[1] := [];
        AltAdjVertArray[Vert]^.VertSetArray[2] := [];
        AltAdjVertArray[Vert]^.VertSetArray[3] := [];
        AltAdjVertArray[Vert]^.VertSetArray[4] := [];
        VertStatsArray[Vert].Vert := Vert; VertStatsArray[Vert].LB      := 1;
        VertStatsArray[Vert].UChromDeg := 0; VertStatsArray[Vert].UDeg := 0;
      end;
    for I := 1 to 6 do
      readln (DataIn);
    readln (DataIn,NumOfVert); readln (DataIn); readln (DataIn,NumOfEdges);
    readln (DataIn);
    for I := 1 to NumOfVert do
      begin
        read (DataIn,Vertex,Chrom); VertStatsArray[Vertex].Chrom := Chrom;
      end;
    readln(DataIn); readln(DataIn);
    for I := 1 to NumOfEdges do
      begin
        read (DataIn,Vertex1,Vertex2); AddNewAdjVert2 (Vertex1,Vertex2);
        AddNewAdjVert2 (Vertex2,Vertex1);
        VertStatsArray[Vertex1].UChromDeg :=
          VertStatsArray[Vertex1].UChromDeg + VertStatsArray[Vertex2].Chrom;
        VertStatsArray[Vertex2].UChromDeg :=
```

```
          VertStatsArray[Vertex2].UChromDeg + VertStatsArray[Vertex1].Chrom;
        VertStatsArray[Vertex1].UDeg := VertStatsArray[Vertex1].UDeg + 1;
        VertStatsArray[Vertex2].UDeg := VertStatsArray[Vertex2].UDeg + 1;
      end;
  end;

procedure SelectVertToColor1 (var Vertex : VertexType);
  var
    I,J,Vert : VertexType; MaxChrom,MaxU1ChromDeg,MaxU1Deg : integer;
    VertListPtr : VertListPtrType;
  begin
    MaxChrom := -999; VertListPtr := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (VertStatsArray[Vert].Chrom>MaxChrom) then
          MaxChrom := VertStatsArray[Vert].Chrom;
        VertListPtr := VertListPtr^.U1NextPtr;
      end;
    CandidateArray1Ptr^.NumOfCandidates := 0;
    VertListPtr := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (VertStatsArray[Vert].Chrom=MaxChrom) then
          begin
            CandidateArray1Ptr^.NumOfCandidates :=
                    CandidateArray1Ptr^.NumOfCandidates + 1;
            J := CandidateArray1Ptr^.NumOfCandidates;
            CandidateArray1Ptr^.CandidateArray[J] := Vert;
          end;
        VertListPtr := VertListPtr^.U1NextPtr;
      end;
    MaxU1ChromDeg := -999;
    for I := 1 to CandidateArray1Ptr^.NumOfCandidates do
      begin
        Vert := CandidateArray1Ptr^.CandidateArray[I];
        if (VertStatsArray[Vert].U1ChromDeg>MaxU1ChromDeg) then
          MaxU1ChromDeg := VertStatsArray[Vert].U1ChromDeg;
      end;
    CandidateArray2Ptr^.NumOfCandidates := 0;
    for I := 1 to CandidateArray1Ptr^.NumOfCandidates do
      begin
        Vert := CandidateArray1Ptr^.CandidateArray[I];
```

```
        if (VertStatsArray[Vert].U1ChromDeg=MaxU1ChromDeg) then
          begin
            CandidateArray2Ptr^.NumOfCandidates :=
                    CandidateArray2Ptr^.NumOfCandidates + 1;
            J := CandidateArray2Ptr^.NumOfCandidates;
            CandidateArray2Ptr^.CandidateArray[J] := Vert;
          end;
      end;
    MaxU1Deg := -999;
    for I := 1 to CandidateArray2Ptr^.NumOfCandidates do
      begin
        Vert := CandidateArray2Ptr^.CandidateArray[I];
        if (VertStatsArray[Vert].U1Deg>MaxU1Deg) then
          begin
            Vertex := Vert; MaxU1Deg := VertStatsArray[Vert].U1Deg;
          end;
      end;
  end;

procedure SelectVertToColor2 (var Vertex : VertexType);
  var
    I,J,Vert : VertexType;
    MaxChrom,MaxU2ChromDeg,MaxU2Deg,MinU1ChromDeg,MinU1Deg :
integer;
    VertListPtr : VertListPtrType;
  begin
    MaxChrom := -999; VertListPtr := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (VertStatsArray[Vert].Chrom>MaxChrom) then
          MaxChrom := VertStatsArray[Vert].Chrom;
        VertListPtr := VertListPtr^.U1NextPtr;
      end;
    CandidateArray1Ptr^.NumOfCandidates := 0;
    VertListPtr := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (VertStatsArray[Vert].Chrom=MaxChrom) then
          begin
            CandidateArray1Ptr^.NumOfCandidates :=
                    CandidateArray1Ptr^.NumOfCandidates + 1;
            J := CandidateArray1Ptr^.NumOfCandidates;
```

```
            CandidateArray1Ptr^.CandidateArray[J] := Vert;
          end;
        VertListPtr := VertListPtr^.U1NextPtr;
      end;
    MaxU2ChromDeg := -999;
    for I := 1 to CandidateArray1Ptr^.NumOfCandidates do
      begin
        Vert := CandidateArray1Ptr^.CandidateArray[I];
        if (VertStatsArray[Vert].U2ChromDeg > MaxU2ChromDeg) then
          MaxU2ChromDeg := VertStatsArray[Vert].U2ChromDeg;
      end;
    CandidateArray2Ptr^.NumOfCandidates := 0;
    for I := 1 to CandidateArray1Ptr^.NumOfCandidates do
      begin
        Vert := CandidateArray1Ptr^.CandidateArray[I];
        if (VertStatsArray[Vert].U2ChromDeg = MaxU2ChromDeg) then
          begin
            CandidateArray2Ptr^.NumOfCandidates :=
                    CandidateArray2Ptr^.NumOfCandidates + 1;
            J := CandidateArray2Ptr^.NumOfCandidates;
            CandidateArray2Ptr^.CandidateArray[J] := Vert;
          end;
      end;
    MaxU2Deg := -999;
    for I := 1 to CandidateArray2Ptr^.NumOfCandidates do
      begin
        Vert := CandidateArray2Ptr^.CandidateArray[I];
        if (VertStatsArray[Vert].U2Deg > MaxU2Deg) then
          MaxU2Deg := VertStatsArray[Vert].U2Deg;
      end;
    CandidateArray1Ptr^.NumOfCandidates := 0;
    for I := 1 to CandidateArray2Ptr^.NumOfCandidates do
      begin
        Vert := CandidateArray2Ptr^.CandidateArray[I];
        if (VertStatsArray[Vert].U2Deg = MaxU2Deg) then
          begin
            CandidateArray1Ptr^.NumOfCandidates :=
                    CandidateArray1Ptr^.NumOfCandidates + 1;
            J := CandidateArray1Ptr^.NumOfCandidates;
            CandidateArray1Ptr^.CandidateArray[J] := Vert;
          end;
      end;
    MinU1ChromDeg := 999;
    for I := 1 to CandidateArray1Ptr^.NumOfCandidates do
```

```
        begin
          Vert := CandidateArray1Ptr^.CandidateArray[I];
          if (VertStatsArray[Vert].U1ChromDeg<MinU1ChromDeg) then
            MinU1ChromDeg := VertStatsArray[Vert].U1ChromDeg;
        end;
      CandidateArray2Ptr^.NumOfCandidates := 0;
      for I := 1 to CandidateArray1Ptr^.NumOfCandidates do
        begin
          Vert := CandidateArray1Ptr^.CandidateArray[I];
          if (VertStatsArray[Vert].U1ChromDeg=MinU1ChromDeg) then
            begin
              CandidateArray2Ptr^.NumOfCandidates :=
                      CandidateArray2Ptr^.NumOfCandidates + 1;
              J := CandidateArray2Ptr^.NumOfCandidates;
              CandidateArray2Ptr^.CandidateArray[J] := Vert;
            end;
        end;
      MinU1Deg := 999;
      for I := 1 to CandidateArray2Ptr^.NumOfCandidates do
        begin
          Vert := CandidateArray2Ptr^.CandidateArray[I];
          if (VertStatsArray[Vert].U1Deg<MinU1Deg) then
            begin
              Vertex := Vert; MinU1Deg := VertStatsArray[Vert].U1Deg;
            end;
        end;
    end;

function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
    var
      I : word;
      Vert : VertexType;
    begin
      case Vert1 of
        1..250    : begin I := 1; Vert := Vert1; end;
        251..500  : begin I := 2; Vert := Vert1 - 250; end;
        501..750  : begin I := 3; Vert := Vert1 - 500; end;
        751..1000 : begin I := 4; Vert := Vert1 - 750; end;
      end;
      IsIn2 := Vert in AltAdjVertArray[Vert2]^.VertSetArray[I];
    end;

procedure DeleteUNode (Vertex : VertexType);
    var
```

```
    Vert : VertexType; FoundIt : boolean; VertListPtr,PrevPtr : VertListPtrType;
  begin
    NumOfUVert := NumOfUVert - 1; FoundIt := false;
    PrevPtr := VertListHeadPtr;
    VertListPtr := VertListHeadPtr^.UNextPtr;
    while (not FoundIt) do
      if (Vertex=VertListPtr^.Vert) then FoundIt := true
      else
        begin
          PrevPtr := VertListPtr; VertListPtr := VertListPtr^.UNextPtr
        end;
    PrevPtr^.UNextPtr := VertListPtr^.UNextPtr; dispose (VertListPtr);
    VertListPtr := VertListHeadPtr^.UNextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (IsIn2(Vert,Vertex)) then
          begin
            if (VertStatsArray[Vert].LB<MaxClr+1) then
              VertStatsArray[Vert].LB := MaxClr + 1;
            VertStatsArray[Vert].UChromDeg :=
              VertStatsArray[Vert].UChromDeg-VertStatsArray[Vertex].Chrom;
            VertStatsArray[Vert].UDeg := VertStatsArray[Vert].UDeg - 1;
          end;
        VertListPtr := VertListPtr^.UNextPtr;
      end;
  end;

procedure DeleteU1Node (Vertex : VertexType);
  var
    Vert : VertexType; FoundIt : boolean; VertListPtr,PrevPtr : VertListPtrType;
  begin
    NumOfU1Vert := NumOfU1Vert - 1; FoundIt := false; PrevPtr :=
VertListHeadPtr;
    VertListPtr := VertListHeadPtr^.U1NextPtr;
    while (not FoundIt) do
      if (Vertex=VertListPtr^.Vert) then FoundIt := true
      else
        begin
          PrevPtr := VertListPtr; VertListPtr := VertListPtr^.U1NextPtr;
        end;
    PrevPtr^.U1NextPtr := VertListPtr^.U1NextPtr;
  end;
```

```
procedure UpdateU1Neighbors (Vert1 : VertexType);
  var
    Vert2 : VertexType; VertListPtr2 : VertListPtrType;
  begin
    VertListPtr2 := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr2 <> nil) do
      begin
        Vert2 := VertListPtr2^.Vert;
        if (IsIn2(Vert2,Vert1)) then
          begin
            VertStatsArray[Vert2].U1ChromDeg :=
              VertStatsArray[Vert2].U1ChromDeg - VertStatsArray[Vert1].Chrom;
            VertStatsArray[Vert2].U1Deg := VertStatsArray[Vert2].U1Deg - 1;
            VertStatsArray[Vert2].U2ChromDeg :=
             VertStatsArray[Vert2].U2ChromDeg + VertStatsArray[Vert1].Chrom;
            VertStatsArray[Vert2].U2Deg := VertStatsArray[Vert2].U2Deg + 1;
          end;
        VertListPtr2 := VertListPtr2^.U1NextPtr;
      end;
  end;

procedure DoUpdates (Vertex : VertexType);
  var
    Vert1,Vert2 : VertexType; VertListPtr1,VertListPtr2,PrevPtr : VertListPtrType;
  begin
    DeleteUNode (Vertex); DeleteU1Node (Vertex); PrevPtr := VertListHeadPtr;
    VertListPtr1 := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr1 <> nil) do
      begin
        Vert1 := VertListPtr1^.Vert;
        if (IsIn2(Vert1,Vertex)) then
          begin
            NumOfU1Vert := NumOfU1Vert - 1;
            VertStatsArray[Vert1].U1ChromDeg :=
             VertStatsArray[Vert1].U1ChromDeg - VertStatsArray[Vertex].Chrom;
            VertStatsArray[Vert1].U1Deg := VertStatsArray[Vert1].U1Deg - 1;
            PrevPtr^.U1NextPtr := VertListPtr1^.U1NextPtr;
            UpdateU1Neighbors (Vert1);
          end
        else
          PrevPtr := VertListPtr1;
        VertListPtr1 := VertListPtr1^.U1NextPtr;
      end;
  end;
```

```
procedure GenMinLB;
  var
    Vert : VertexType; VertListPtr : VertListPtrType;
  begin
    MinLB := 999;
    VertListPtr := VertListHeadPtr^.UNextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (VertStatsArray[Vert].LB<MinLB) then
          MinLB := VertStatsArray[Vert].LB;
        VertListPtr := VertListPtr^.UNextPtr;
      end;
  end;

procedure InitializeU1List;
  var
    Vert : VertexType; VertListPtr : VertListPtrType;
  begin
    NumOfU1Vert := NumOfUVert;
    VertListHeadPtr^.U1NextPtr := VertListHeadPtr^.UNextPtr;
    VertListPtr := VertListHeadPtr^.UNextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        VertStatsArray[Vert].U1ChromDeg:=VertStatsArray[Vert].UChromDeg;
        VertStatsArray[Vert].U1Deg := VertStatsArray[Vert].UDeg;
        VertStatsArray[Vert].U2ChromDeg := 0;
        VertStatsArray[Vert].U2Deg := 0;
        VertListPtr^.U1NextPtr := VertListPtr^.UNextPtr;
        VertListPtr := VertListPtr^.UNextPtr;
      end;
  end;

procedure ReduceU1List;
  var
    Vert : VertexType; VertListPtr : VertListPtrType;
  begin
    VertListPtr := VertListHeadPtr^.U1NextPtr;
    while (VertListPtr<>nil) do
      begin
        Vert := VertListPtr^.Vert;
        if (VertStatsArray[Vert].LB>MinLB) then
          begin
```

```
            DeleteU1Node(Vert);
            UpdateU1Neighbors (Vert);
          end;
        VertListPtr := VertListPtr^.U1NextPtr;
      end;
  end;

procedure DoRLFColoring;
  var
    Clr : ColorType; Vertex : VertexType;
  begin
    RLFColors := 0; NumOfUVert := NumOfVert;
    while (NumOfUVert>0) do
      begin
        InitializeU1List; GenMinLB; ReduceU1List;
        SelectVertToColor1 (Vertex);
        MaxClr := MinLB + VertStatsArray[Vertex].Chrom - 1;
        for Clr := MinLB to MaxClr do
          AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
        if (MaxClr>RLFColors) then RLFColors := MaxClr;
        DoUpDates (Vertex);
        while (NumOfU1Vert>0) do
          begin
            SelectVertToColor2 (Vertex);
            MaxClr := MinLB + VertStatsArray[Vertex].Chrom - 1;
            for Clr := MinLB to MaxClr do
              AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
            if (MaxClr>RLFColors) then RLFColors := MaxClr;
            DoUpdates (Vertex);
          end;
      end;
  end;

procedure CheckAssgnArray (ColorNum : word);
  var
    Clr : word; Vert1,Vert2 : Verte  Type;
  begin
    Feasible := true; Vert1 := 1;
    while (Vert1 <=NumOfVert) and (Feasible) do
      begin
        for Clr := 1 to ColorNum do
          if (Clr in AssgnArray[Vert1]) then
            begin
              Vert2 := 1;
```

```
                while (Vert2<=NumOfVert) and (Feasible) do
                  begin
                    if (IsIn2(Vert2,Vert1)) and (Clr in AssgnArray[Vert2]) then
                      begin
                        Feasible := false;
                        writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                      end
                    else
                      Vert2 := Vert2 + 1;
                  end;
              end;
          Vert1 := Vert1 + 1;
        end;
    end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : real;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Prn,'con'); rewrite (Prn); assign (Out, 'REP1.PRN'); rewrite (Out);
  ObtainUserInfo; PrintReportHeadingsAndInitTotals;
  for Count := BegGraphNum to EndGraphNum do
    begin
      if (Count<10) then
        begin FileName := FileNamePrefix + '.00'; str(Count:1,Extension); end
      else
        begin FileName := FileNamePrefix + '.0'; str(Count:2,Extension); end;
      FileName := FileName + Extension;
      assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName);
      reset (DataIn); writeln; writeln ('Processing ',FileName);
      ReadData; close (DataIn);
      Initialize; writeln('Working on CRLF'); GetTime (Hour1,Min1,Sec1,Frac1);
      DoRLFColoring; GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
      RLFTime := ElapsedTime; AvgRLFTime := AvgRLFTime + RLFTime;
      AvgRLFColors := AvgRLFColors + RLFColors;
      CheckAssgnArray(RLFColors);
```

```
      writeln (UpperCase(FileName):11,RLFColors:8); writeln (RLFTime:22:2);
      writeln (' ':10,Feasible:11); writeln (Out,RLFColors:8);
      {writeln (Out,UpperCase(FileName):11,RLFColors:8);
      writeln (Out,RLFTime:22:2); writeln (Out,' ':10,Feasible:11);}
    end;
  AvgRLFColors := AvgRLFColors/NumOfGraphs;
  AvgRLFTime := AvgRLFTime/NumOfGraphs;
  writeln (Out,'----------  --------');
  writeln (Out,'  Averages',AvgRLFColors:11:2);
  writeln (Out,AvgRLFTime:22:2);
  close (Out);
end.
```

10. CVSI1.Pas -- Implements CLFI1 and CSLI1

```
{$N+,E-,M 65520,0,655360}
{*** Implements CLFI1 and CSLI1 ***}
program CVSI1;
uses DOS,CRT;
const
  MaxVert = 500; {Max no. of vertices}
  MaxColor = 115; {Upper bound for chromatic number}
type
  VertexType = 1..MaxVert; VertexSetType = set of 1..250;
  ColorType = 0..MaxColor;
  ColorSetType = set of ColorType; String11 = string[11];
  VertexOrderArrayType = record
                           Vert : VertexType; Chrom : word; CDeg : word;
                           WChromDeg : word; WDeg : word;
                         end;
  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  NumOfAdjVert : word; VertArray : array[1..300] of VertexType;
                end;
  AltAdjListPtrType = ^AltAdjListType;
  AltAdjListType = record
                     VertSetArray : array[1..2] of VertexSetType;
                   end;
  ColorSetsPtrType = ^ColorSetsType;
  ColorSetsType = record
                    ColorSets : array[1..5] of ColorSetType;
                  end;
var
  Line : array [1..100] of string[50];
```

```
  FileName,FileNamePrefix,Extension : String11;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  MaxVertChrom,NumOfVert,NumOfEdges : word;
  I,Count,NumOfGraphs,BegGraphNum,EndGraphNum,NumOfLines : word;
  LF1I1Colors,SL1I1Colors : word;
  LF1I1Time,SL1I1Time,ElapsedTime,AvgElapsedTime : double;
  AvgLF1I1Colors,AvgSL1I1Colors,AvgLF1I1Time,AvgSL1I1Time : double;
  VertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
  AssgnArray,SaveAssgnArray : array[VertexType] of ColorSetType;
  VertFirstColorArray : array[1..MaxVert] of byte;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  AltAdjVertArray : array[VertexType] of AltAdjListPtrType;
  AdjColorArray,SaveAdjColorArray : array[VertexType] of ColorSetType;
  FeasibleSetArray,SaveFeasibleSetArray : array[VertexType] of ColorSetType;
  ChromArray,WChromDegArray,WDegArray : array[VertexType] of word;
  ColorSetsArray : array [1..MaxColor] of ColorSetsPtrType;
  FeasibleArray : array[1..5] of boolean;
  DataIn,Out,Prn : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word; ConvertedName : String11;
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  var
    Code : integer;
  begin
    ClrScr;
    writeln ('Application:  Composite Graph Coloring'); writeln;
    writeln ('Program:  CVSI1.PAS -- Heuristic'); writeln;
    write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
    write ('Enter Num of Vertices -- '); readln (NumOfVert); writeln;
    write ('Enter Begin Graph Number:  '); readln (BegGraphNum); writeln;
    write ('Enter End Graph Number:  '); readln (EndGraphNum);
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
  end;
```

```
{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Color : ColorType; Vert : VertexType; AdjListPtr : AdjListPtrType;
    AltAdjListPtr : AltAdjListPtrType; ColorSetsPtr : ColorSetsPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin
        new(AdjListPtr); AdjVertArray[Vert] := AdjListPtr;
      end;
    for Vert := 1 to NumOfVert do
      begin
        new(AltAdjListPtr); AltAdjVertArray[Vert] := AltAdjListPtr;
      end;
    for Color := 1 to MaxColor do
      begin
        new(ColorSetsPtr); ColorSetsArray[Color] := ColorSetsPtr;
        ColorSetsPtr^.ColorSets[1] := [Color];
        ColorSetsPtr^.ColorSets[2] := [Color,Color+1];
        ColorSetsPtr^.ColorSets[3] := [Color,Color+1,Color+2];
        ColorSetsPtr^.ColorSets[4] := [Color,Color+1,Color+2,Color+3];
        ColorSetsPtr^.ColorSets[5]:=[Color,Color+1,Color+2,Color+3,Color+4];
      end;
    AvgLF1I1Colors := 0.0; AvgSL1I1Colors := 0.0;
    AvgLF1I1Time := 0.0; AvgSL1I1Time := 0.0;
    writeln (Out,'Application: Composite Graph Coloring');
    writeln (Out,'Program:     C_VSI1.PAS -- Heuristic');
    writeln (Out,'Ordering:    LF1 and SL1'); writeln (Out);
    writeln (Out,' File Name      LF1I1      SL1I1 ');
    writeln (Out,'-----------  --------  --------');
  end;

procedure Initialize;
  var
    Vert : VertexType;
  begin
    for Vert := 1 to MaxVert do
      begin
        AssgnArray[Vert] := []; VertFirstColorArray[Vert] := 0;
        AdjColorArray[Vert] := []; FeasibleSetArray[Vert] := [];
      end;
  end;

{*** Input problem definition data and update graph definition structures. ***}
```

```
procedure ReadData;
  var
    I,Vert,Vertex,Vertex1,Vertex2,Chrom : word;
  procedure AddNewAdjVertex1 (Vert1,Vert2 : VertexType);
    var
      NumOfAdjVert : word;
    begin
      AdjVertArray[Vert1]^.NumOfAdjVert :=
        AdjVertArray[Vert1]^.NumOfAdjVert + 1;
      NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
      AdjVertArray[Vert1]^.VertArray[NumOfAdjVert] := Vert2;
    end;
  procedure AddNewAdjVertex2 (Vert1,Vert2 : VertexType);
    var
       I : word; Vert : VertexType;
    begin
      if (Vert2<=250) then
        begin I := 1; Vert := Vert2; end
      else
        begin I := 2; Vert := Vert2 - 250; end;
      AltAdjVertArray[Vert1]^.VertSetArray[I] :=
        AltAdjVertArray[Vert1]^.VertSetArray[I] + [Vert];
    end;
  begin
    LF1I1Colors := 999; SL1I1Colors := 999; MaxVertChrom := 0;
    for Vert := 1 to NumOfVert do
      begin
        AdjVertArray[Vert]^.NumOfAdjVert := 0;
        AltAdjVertArray[Vert]^.VertSetArray[1] := [];
        AltAdjVertArray[Vert]^.VertSetArray[2] := [];
        WChromDegArray[Vert] := 0; WDegArray[Vert] := 0;
      end;
    for I := 1 to 6 do
      readln (DataIn);
    readln (DataIn,NumOfVert); readln (DataIn);
    readln (DataIn,NumOfEdges); readln (DataIn);
    for I := 1 to NumOfVert do
      begin
        read (DataIn,Vertex,Chrom); ChromArray[Vertex] := Chrom;
        if (Chrom>MaxVertChrom) then MaxVertChrom := Chrom;
      end;
    readln(DataIn);  readln(DataIn);
    for I := 1 to NumOfEdges do
      begin
```

```
          read (DataIn,Vertex1,Vertex2);
                AddNewAdjVertex1(Vertex1,Vertex2);
AddNewAdjVertex1(Vertex2,Vertex1);
          AddNewAdjVertex2(Vertex1,Vertex2);
          AddNewAdjVertex2(Vertex2,Vertex1);
          WDegArray[Vertex1] := WDegArray[Vertex1] + 1;
          WDegArray[Vertex2] := WDegArray[Vertex2] + 1;
          WChromDegArray[Vertex1] := WChromDegArray[Vertex1] +
                                     ChromArray[Vertex2];
          WChromDegArray[Vertex2] := WChromDegArray[Vertex2] +
                                     ChromArray[Vertex1];
        end;
    end;

function IsIn1 (Vert : VertexType; AdjListPtr : AdjListPtrType) : boolean;
    var
      I,NumOfAdjVert : word; FoundIt : boolean;
    begin
      NumOfAdjVert := AdjListPtr^.NumOfAdjVert; FoundIt := false; I := 1;
      while (not FoundIt)and (I<=NumOfAdjVert) do
        begin
          if (Vert=AdjListPtr^.VertArray[I]) then FoundIt := true;
          I := I + 1;
        end;
      IsIn1 := FoundIt;
    end;

function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
    var
      I : word; Vert : VertexType;
    begin
      if (Vert1<=250) then
        begin I := 1; Vert := Vert1; end
      else
        begin I := 2; Vert := Vert1 - 250; end;
      IsIn2 := Vert in AltAdjVertArray[Vert2]^.VertSetArray[I];
    end;

{*** Gen an array containg the vertex indices in largest first order ***}
procedure GenLF1VertOrderArray;
    var
      I,J,K : word; Temp : VertexOrderArrayType;
    begin
      for I := 1 to NumOfVert do
```

```
      begin
        VertOrderArray[I].Vert := I;
        VertOrderArray[I].Chrom := ChromArray[I];
        VertOrderArray[I].WChromDeg := WChromDegArray[I];
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    for I := 1 to (NumOfVert-1) do
      begin
        K := I;
        for J := (I+1) to NumOfVert do
          if (VertOrderArray[J].Chrom > VertOrderArray[K].Chrom) then
            K := J
          else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
           (VertOrderArray[J].WChromDeg > VertOrderArray[K].WChromDeg)
           then K := J
          else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
           (VertOrderArray[J].WChromDeg = VertOrderArray[K].WChromDeg)
           and (VertOrderArray[J].WDeg > VertOrderArray[K].WDeg) then
           K := J;
        if (K < > I) then
          begin
            Temp := VertOrderArray[I];
            VertOrderArray[I] := VertOrderArray[K];
            VertOrderArray[K] := Temp;
          end;
      end;
  end;

{*** Gen an array containg the vertex indices in smallest last order ***}
procedure GenSL1VertOrderArray;
  var
    I,J,K : word; Temp : VertexOrderArrayType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].Chrom := ChromArray[I];
        VertOrderArray[I].WChromDeg := WChromDegArray[I];
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    for I := 1 to (NumOfVert-1) do
      begin
        K := I;
        for J := (I+1) to NumOfVert do
          if (VertOrderArray[J].Chrom < VertOrderArray[K].Chrom) then K := J
```

```
            else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
              (VertOrderArray[J].WChromDeg < VertOrderArray[K].WChromDeg)
              then K := J
            else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
              (VertOrderArray[J].WChromDeg = VertOrderArray[K].WChromDeg)
             and (VertOrderArray[J].WDeg < VertOrderArray[K].WChromDeg) then
              K := J;
          if (K < > I) then
            begin
              Temp := VertOrderArray[I];
              VertOrderArray[I] := VertOrderArray[K];
              VertOrderArray[K] := Temp;
            end;
          for J := (I+1) to NumOfVert do
            if (IsIn2(VertOrderArray[I].Vert,
              VertOrderArray[J].Vert)) then
              begin
                VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
                VertOrderArray[J].WChromDeg :=
                 VertOrderArray[J].WChromDeg - VertOrderArray[I].WChromDeg;
              end;
        end;
      for I := 1 to (NumOfVert div 2) do
        begin
          Temp := VertOrderArray[I];
          VertOrderArray[I] := VertOrderArray[NumOfVert-I+1];
          VertOrderArray[NumOfVert-I+1] := Temp;
        end;
    end;

procedure UpdateFeasibleSets1 (MaxClr : ColorType);
  var
    VertChrom : byte; Vert : VertexType; Color : ColorType;
  begin
    for Vert := 1 to NumOfVert do
      FeasibleSetArray[Vert] := [];
    for Vert := 1 to NumOfVert do
      begin
        VertChrom := ChromArray[Vert];
        for Color := 1 to (MaxClr-ChromArray[Vert]+1) do
          if (ColorSetsArray[Color]^.ColorSets[VertChrom] *
                AdjColorArray[Vert]=[]) then
            FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Color];
      end;
```

```
  end;

procedure UpdateFeasibleSets2 (MaxClr : ColorType);
  var
    Vert : VertexType; Color : ColorType;
  begin
    for Vert := 1 to NumOfVert do
        begin
          Color := MaxClr - ChromArray[Vert] + 1;
          if (ColorSetsArray[Color]^.ColorSets[ChromArray[Vert]] *
              AdjColorArray[Vert]=[]) then
            FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Color];
        end;
  end;

procedure UpdateFeasibleSets3 (MaxClr : ColorType; Vertex : VertexType;
                               VertexFirstColor : ColorType);
  var
    ColorIsFeasible : boolean; Vert : VertexType;
    Clr,StartClr,EndClr : ColorType; I,NumOfAdjVert : word;
  begin
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        StartClr := VertexFirstColor - ChromArray[Vert] + 1;
        EndClr := VertexFirstColor + ChromArray[Vertex] - 1;
        if (StartClr<1) then StartClr := 1;
        if (EndClr > MaxClr) then EndClr := MaxClr;
        for Clr := StartClr to EndClr do
          if (Clr in FeasibleSetArray[Vert]) then
            FeasibleSetArray[Vert] := FeasibleSetArray[Vert] - [Clr];
      end;
  end;

procedure UpdateFeasibleSets4 (MaxClr : ColorType; Vertex : VertexType;
VertexFirstColor : ColorType);
  var
    Vert : VertexType; VertChrom : byte;
    Clr,StartClr,EndClr : ColorType; I,NumOfAdjVert : word;
  begin
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
```

```
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        VertChrom := ChromArray[Vert];
        StartClr := VertexFirstColor - VertChrom + 1;
        EndClr := VertexFirstColor + ChromArray[Vertex] - 1;
        if (StartClr<1) then StartClr := 1;
        if (EndClr+ChromArray[Vert]-1>MaxClr) then
          EndClr := MaxClr-VertChrom+1;
        for Clr := StartClr to EndClr do
          if (ColorSetsArray[Clr]^.ColorSets[VertChrom] *
              AdjColorArray[Vert]=[]) then
            FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Clr];
      end;
  end;

procedure GenSmallestFeasibleColor (MaxClr : ColorType; Vertex : VertexType;
                                    var Color : ColorType);
  var
    FoundIt : boolean; Clr : ColorType;
  begin
    FoundIt := false; Clr := 1;
    while (Clr<=MaxClr) and (not FoundIt) do
      if (Clr in FeasibleSetArray[Vertex]) then
        begin Color := Clr; FoundIt := true; end
      else Clr := Clr + 1;
  end;

procedure DoUpdate1 (Vert1 : VertexType; Color : ColorType);
  var
    StillIn,Again : boolean; Clr2,MaxClr2 : ColorType; Vert2,Vert3 : VertexType;
    I1,NumOfAdjVert1,I2,NumOfAdjVert2 : word;
    AdjPtr1,AdjPtr2 : AdjListPtrType;
  begin
    NumOfAdjVert1 := AdjVertArray[Vert1]^.NumOfAdjVert;
    for I1 := 1 to NumOfAdjVert1 do
      begin
        Vert2 := AdjVertArray[Vert1]^.VertArray[I1];
        MaxClr2 := Color + ChromArray[Vert1] - 1;
        for Clr2 := Color to MaxClr2 do
          begin
            StillIn := false; Again := true;
            NumOfAdjVert2 := AdjVertArray[Vert2]^.NumOfAdjVert; I2 := 1;
            while (I2<=NumOfAdjVert2) and (Again) do
              begin
                Vert3 := AdjVertArray[Vert2]^.VertArray[I2];
```

```
                if (Clr2 in AssgnArray[Vert3]) then
                  begin StillIn := true; Again := false; end
                else I2 := I2 + 1;
              end;
            if (not StillIn) then
              AdjColorArray[Vert2] := AdjColorArray[Vert2] - [Clr2];
          end;
      end;
  end;

procedure DoUpdate2 (Vert1 : VertexType; Color : ColorType);
  var
    Clr2,MaxClr2 : ColorType; Vert2 : VertexType; I,NumOfAdjVert : word;
  begin
    NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert2 := AdjVertArray[Vert1]^.VertArray[I];
        MaxClr2 := Color + ChromArray[Vert1] - 1;
        for Clr2 := Color to MaxClr2 do
          if (not (Clr2 in AdjColorArray[Vert2])) then
            AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Clr2];
      end;
  end;

procedure AttemptAnInterchange1 (MaxClr : ColorType; UpLimit : word;
                                 Vertex : VertexType; var Color : ColorType;
                                 var FoundOne,FirstTry : boolean);
  var
    Changed : boolean; I : word; Vert1Chrom : byte;
    Vert1 : VertexType; MaxClr1,Clr1,Vert1FirstColor,Clr2,MaxClr2 : ColorType;
  begin
    SaveFeasibleSetArray := FeasibleSetArray;
    SaveAdjColorArray := AdjColorArray;
    SaveAssgnArray := AssgnArray; FoundOne := false; Changed := false; I := 1;
    while (not FoundOne) and (I<=UpLimit) do
      begin
        if (Changed) then
          begin
            FeasibleSetArray := SaveFeasibleSetArray;
            AdjColorArray := SaveAdjColorArray;
          end;
        Changed := false; Vert1 := VertOrderArray[I].Vert;
        if (IsIn2(Vert1,Vertex)) and (FeasibleSetArray[Vert1]<>[]) then
```

```
begin
  Vert1FirstColor := VertFirstColorArray[Vert1];
  Vert1Chrom := ChromArray[Vert1];
  if (FirstTry) then Clr1 := 1
  else Clr1 := MaxClr - Vert1Chrom + 1;
  MaxClr1 := MaxClr - Vert1Chrom + 1;
  while (Clr1 <= MaxClr1) and (not FoundOne) do
     begin
       if (Changed) then
         begin
           FeasibleSetArray := SaveFeasibleSetArray;
           AdjColorArray := SaveAdjColorArray;
         end;
       if (Clr1 in FeasibleSetArray[Vert1])
         and (Clr1 <> Vert1FirstColor) then
         begin
           Changed := true; AssgnArray[Vert1] := [];
           DoUpDate1 (Vert1,Vert1FirstColor);
           UpdateFeasibleSets4 (MaxClr,Vert1,Vert1FirstColor);
           if (FeasibleSetArray[Vertex] <> []) then
             begin
               MaxClr2 := Clr1 + Vert1Chrom - 1;
               for Clr2 := Clr1 to MaxClr2 do
                 AssgnArray[Vert1] := AssgnArray[Vert1] + [Clr2];
               VertFirstColorArray[Vert1] := Clr1;
               DoUpdate2 (Vert1,Clr1);
               UpdateFeasibleSets3(MaxClr,Vert1,Clr1);
               if (FeasibleSetArray[Vertex] <> []) then
                 begin
                   FoundOne := true;
                   GenSmallestFeasibleColor(MaxClr,Vertex,Color);
                 end
               else
                 begin
                   AssgnArray[Vert1] := SaveAssgnArray[Vert1];
                   VertFirstColorArray[Vert1] := Vert1FirstColor;
                 end;
             end
           else
             begin
               AssgnArray[Vert1] := SaveAssgnArray[Vert1];
               VertFirstColorArray[Vert1] := Vert1FirstColor;
             end;
         end;
```

```
                Clr1 := Clr1 + 1;
              end;
          end;
        I := I + 1;
      end;
    if (not FoundOne) and (Changed) then
      begin
        FeasibleSetArray := SaveFeasibleSetArray;
        AdjColorArray := SaveAdjColorArray;
      end;
    FirstTry := false;
  end;

procedure DoCLF1I1Coloring;
  var
    FoundOne,FirstTry : boolean; I,J,NumOfAdjVert : word;
    Vert,Vertex : VertexType; MaxClr,Clr,Color,MaxColor : ColorType;
  begin
    GenLF1VertOrderArray; Vertex := VertOrderArray[1].Vert;
    for Clr := 1 to ChromArray[Vertex] do
      AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
    VertFirstColorArray[Vertex] := 1;
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        AdjColorArray[Vert] := AdjColorArray[Vert] + AssgnArray[Vertex];
      end;
    LF1I1Colors := ChromArray[Vertex]; UpdateFeasibleSets1 (LF1I1Colors);
    for I := 2 to NumOfVert do
      begin
        Vertex := VertOrderArray[I].Vert; FoundOne := false; FirstTry := true;
        while (not FoundOne) do
          if (FeasibleSetArray[Vertex]<>[]) then
            begin
              FoundOne := true;
              GenSmallestFeasibleColor (LF1I1Colors,Vertex,Color);
            end
          else
            begin
              AttemptAnInterchange1 (LF1I1Colors,I-1,Vertex,Color,
                                     FoundOne,FirstTry);
              if (not FoundOne) then
                begin
```

```
                    LF1I1Colors := LF1I1Colors + 1;
                    UpdateFeasibleSets2 (LF1I1Colors);
                  end;
              end;
          MaxClr := Color + ChromArray[Vertex] - 1;
          for Clr := Color to MaxClr do
            AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
          VertFirstColorArray[Vertex] := Color;
          NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
          for J := 1 to NumOfAdjVert do
            begin
              Vert := AdjVertArray[Vertex]^.VertArray[J];
              AdjColorArray[Vert] := AdjColorArray[Vert] + AssgnArray[Vertex];
            end;
          UpdateFeasibleSets3 (LF1I1Colors,Vertex,Color);
        end;
    end;

procedure DoCSL1I1Coloring;
  var
    FoundOne,FirstTry : boolean; I,J,NumOfAdjVert : word;
    Vert,Vertex : VertexType; MaxClr,Clr,Color,MaxColor : ColorType;
  begin
    GenSL1VertOrderArray; Vertex := VertOrderArray[1].Vert;
    for Clr := 1 to ChromArray[Vertex] do
      AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
    VertFirstColorArray[Vertex] := 1;
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        AdjColorArray[Vert] := AdjColorArray[Vert] + AssgnArray[Vertex];
      end;
    SL1I1Colors := ChromArray[Vertex];
    UpdateFeasibleSets1 (SL1I1Colors);
    for I := 2 to NumOfVert do
      begin
        Vertex := VertOrderArray[I].Vert; FoundOne := false; FirstTry := true;
        while (not FoundOne) do
          if (FeasibleSetArray[Vertex]<>[]) then
            begin
              FoundOne := true;
              GenSmallestFeasibleColor (SL1I1Colors,Vertex,Color);
            end
```

```
            else
              begin
                AttemptAnInterchange1 (SL1I1Colors,I-1,Vertex,Color,
                                       FoundOne,FirstTry);
                if (not FoundOne) then
                  begin
                    SL1I1Colors := SL1I1Colors + 1;
                    UpdateFeasibleSets2 (SL1I1Colors);
                  end;
              end;
            MaxClr := Color + ChromArray[Vertex] - 1;
            for Clr := Color to MaxClr do
              AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
            VertFirstColorArray[Vertex] := Color;
            NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
            for J := 1 to NumOfAdjVert do
              begin
                Vert := AdjVertArray[Vertex]^.VertArray[J];
                AdjColorArray[Vert] := AdjColorArray[Vert] + AssgnArray[Vertex];
              end;
            UpdateFeasibleSets3 (SL1I1Colors,Vertex,Color);
          end;
      end;

procedure CheckAssgnArray (ColorNum,Method : word);
  var
    I,NumOfAdjVert : word; Clr : byte; Vert1,Vert2 : VertexType;
  begin
    FeasibleArray[Method] := true; Vert1 := 1;
    while (Vert1 <= NumOfVert) and (FeasibleArray[Method]) do
      begin
        for Clr := 1 to ColorNum do
          if (Clr in AssgnArray[Vert1]) then
            begin
              NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
              I := 1;
              while (I <= NumOfAdjVert) and (FeasibleArray[Method]) do
                begin
                  Vert2 := AdjVertArray[Vert1]^.VertArray[I];
                  if (Clr in AssgnArray[Vert2]) then
                    begin
                      FeasibleArray[Method] := false;
                      writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                    end
```

```
                    else
                      I := I + 1;
                  end;
              end;
          Vert1 := Vert1 + 1;
        end;
    end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : double;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Prn,'prn'); rewrite (Prn); assign (Out, 'REPORT.PRN'); rewrite (Out);
  PrintReportHeadingsAndInitTotals; close (Out); ObtainUserInfo;
  for Count := BegGraphNum to EndGraphNum do
    begin
      if (Count<10) then
        begin FileName := FileNamePrefix + '.00'; str(Count:1,Extension); end
      else
        begin FileName := FileNamePrefix + '.0'; str(Count:2,Extension); end;
      FileName := FileName + Extension;
      assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName); reset (DataIn);
      writeln; writeln ('Processing ',FileName); ReadData; close (DataIn);
      Initialize; writeln('Working on CLF1I1'); GetTime (Hour1,Min1,Sec1,Frac1);
      DoCLF1I1Coloring; GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
      LF1I1Time := ElapsedTime;
      AvgLF1I1Time := AvgLF1I1Time + LF1I1Time;
      AvgLF1I1Colors := AvgLF1I1Colors + LF1I1Colors;
      CheckAssgnArray (LF1I1Colors,1);
      Initialize; writeln('Working on SL1I1'); GetTime (Hour1,Min1,Sec1,Frac1);
      DoCSL1I1Coloring; GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
      SL1I1Time:= ElapsedTime; AvgSL1I1Time:= AvgSL1I1Time + SL1I1Time;
        AvgSL1I1Colors := AvgSL1I1Colors + SL1I1Colors;
CheckAssgnArray(SL1I1Colors,2);
      writeln (UpperCase(FileName):11,LF1I1Colors:8,SL1I1Colors:11);
      writeln (LF1I1Time:22:2,SL1I1Time:11:2);
```

```
        write (' ':10);
        for I := 1 to 2 do
          write (FeasibleArray[I]:11);
        writeln; reset (Out); NumOfLines := 0;
        while (not eof(Out)) do
          begin
            NumOfLines := NumOfLines + 1; readln (Out,Line[NumOfLines]);
          end;
        close (Out); rewrite (Out);
        for I := 1 to NumOfLines do
          writeln (Out,Line[I]);
        writeln (Out,UpperCase(FileName):11,LF1I1Colors:8,SL1I1Colors:11);
        writeln (Out,LF1I1Time:22:2,SL1I1Time:11:2); write (Out,' ':10);
        for I := 1 to 2 do
          write (Out,FeasibleArray[I]:11);
        writeln (Out); close (Out);
      end;
    reset (Out); NumOfLines := 0;
    while (not eof(Out)) do
      begin
        NumOfLines := NumOfLines + 1; readln (Out,Line[NumOfLines]);
      end;
    close (Out); rewrite (Out);
    for I := 1 to NumOfLines do
      writeln (Out,Line[I]);
    AvgLF1I1Colors := AvgLF1I1Colors/NumOfGraphs;
    AvgSL1I1Colors := AvgSL1I1Colors/NumOfGraphs;
    AvgLF1I1Time := AvgLF1I1Time/NumOfGraphs;
    AvgSL1I1Time := AvgSL1I1Time/NumOfGraphs;
    writeln (Out,'---------- -------- --------');
    writeln (Out,'  Averages',AvgLF1I1Colors:11:2,AvgSL1I1Colors:11:2);
    writeln (Out,AvgLF1I1Time:22:2,AvgSL1I1Time:11:2);
    close (Out);
end.
```

11. CVSI2.Pas -- Implements CLF, CLFI2, CSL, and CSLI2

```
{$N+,E-,M 65520,0,655360}
{*** Implements CLF, CLFI2, CSL, and CSLI2 ***}
program CVSI2;
uses DOS,CRT;
const
  MaxVert = 500; {Max no. of vertices} MaxColor = 130; {Upper bound for
chromatic number}
```

```
type
  VertexType = 1..MaxVert; VertexSetType = set of 1..255;
  ColorType = 0..MaxColor+1;
  ColorSetType = set of ColorType; String11 = string[11];
  VertexOrderArrayType = record
                           Vert : VertexType; Chrom : word; CDeg : word;
                           WChromDeg : word; WDeg : word;
                         end;
  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  NumOfAdjVert : word; VertArray : array [1..350] of word;
                end;
  ConnectedCompPtrType = ^ConnectedCompType;
  ConnectedCompType = record
                        CC : array[1..2] of VertexSetType; NumOfCCVert : word;
                        CCArray : array[1..300] of word;
                        NextPtr : ConnectedCompPtrType;
                      end;
  ColorSetsPtrType = ^ColorSetsType;
  ColorSetsType = record
                    ColorSets : array[1..5] of ColorSetType;
                  end;
var
  FileName,FileNamePrefix,Extension : String11;
  GraphType : char;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2,I,Count,NumOfGraphs : word;
  LF1Colors,LF1I2Colors,SL1Colors,SL1I2Colors : word;
  MaxVertChrom,NumOfVert,NumOfEdges,BegGraphNum,EndGraphNum:word;
  ElapsedTime,LF1Time,LF1I2Time,SL1Time,SL1I2Time : double;
  AvgLF1Time,AvgLF1I2Time,AvgSL1Time,AvgSL1I2Time : double;
  AvgLF1Colors,AvgLF1I2Colors,AvgSL1Colors,AvgSL1I2Colors : double;
  VertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
  AssgnArray : array[VertexType] of ColorSetType;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  AltAdjVertArray : array[VertexType,1..2] of VertexSetType;
  AdjColorArray : array[VertexType] of ColorSetType;
  ColorSetsArray : array[1..MaxColor] of ColorSetsPtrType;
  ChromArray,WChromDegArray,WDegArray : array[VertexType] of word;
  FeasibleArray : array [1..4] of boolean;
  CCHeadPtr,CCRearPtr : ConnectedCompPtrType;
  StackPtr : ^word;
  DataIn,Out,Prn : text;

{*** Convert the file name to all uppercase for printing. ***}
```

```
function UpperCase (FileName : String11) : String11;
  var
    Pos : word; ConvertedName : String11;
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  var
    Code : integer;
  begin
    ClrScr; writeln ('Application:  Composite Graph Coloring'); writeln;
    writeln ('Program:  CVSI2.PAS -- Heuristic'); writeln;
    write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
    write ('Enter Number of Vertices:  '); readln (NumOfVert); writeln;
    write ('Enter Begin Graph Number: '); readln (BegGraphNum); writeln;
    write ('Enter End Graph Number: '); readln (EndGraphNum);
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Count : byte; Color : ColorType; Vert : VertexType;
    AdjListPtr : AdjListPtrType; ColorSetsPtr : ColorSetsPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin
        new(AdjListPtr); AdjVertArray[Vert] := AdjListPtr;
      end;
    for Color := 1 to MaxColor do
      begin
        new(ColorSetsPtr); ColorSetsArray[Color] := ColorSetsPtr;
        ColorSetsPtr^.ColorSets[1] := [Color];
        ColorSetsPtr^.ColorSets[2] := [Color,Color+1];
        ColorSetsPtr^.ColorSets[3] := [Color,Color+1,Color+2];
        ColorSetsPtr^.ColorSets[4] := [Color,Color+1,Color+2,Color+3];
        ColorSetsPtr^.ColorSets[5]:=[Color,Color+1,Color+2,Color+3,Color+4];
      end;
    new (CCHeadPtr); CCHeadPtr^.NextPtr := nil;
```

```
    CCRearPtr := CCHeadPtr; mark (StackPtr);
    AvgLF1Colors := 0.0; AvgLF1I2Colors := 0.0;
    AvgSL1Colors := 0.0; AvgSL1I2Colors := 0.0;
    AvgLF1Time := 0.0; AvgLF1I2Time := 0.0;
    AvgSL1Time := 0.0; AvgSL1I2Time := 0.0;
    writeln (Out,'Application: Composite Graph Coloring');
    writeln (Out,'Program:     C_VSI2.PAS -- Heuristic');
    writeln (Out,'Ordering:    LF1 and SL1'); writeln (Out);
    writeln (Out,' File Name       LF1       LF1I2       SL1       SL1I2 ');
    writeln (Out,'-----------  --------  --------  --------  --------');
  end;

procedure Initialize;
  var
    Vert : VertexType;
  begin
    for Vert := 1 to NumOfVert do
      begin AssgnArray[Vert] := []; AdjColorArray[Vert] := []; end;
  end;

{*** Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I,Vert,Vertex,Vertex1,Vertex2,Chrom : word;
  procedure AddNewAdjVert1 (Vert1,Vert2 : VertexType);
    var
      I : word;
    begin
      AdjVertArray[Vert1]^.NumOfAdjVert :=
        AdjVertArray[Vert1]^.NumOfAdjVert + 1;
      I := AdjVertArray[Vert1]^.NumOfAdjVert;
      AdjVertArray[Vert1]^.VertArray[I] := Vert2;
    end;
  procedure AddNewAdjVert2 (Vert1,Vert2 : VertexType);
    var
      I : word; Vert : VertexType;
    begin
      if (Vert2<=250) then
        begin I := 1; Vert := Vert2; end
      else
        begin I := 2; Vert := Vert2 - 250; end;
      AltAdjVertArray[Vert1,I] := AltAdjVertArray[Vert1,I] + [Vert];
    end;
  begin
```

```
    LF1Colors := 999; LF1I2Colors := 999; SL1Colors := 999; SL1I2Colors := 999;
    for Vert := 1 to NumOfVert do
      begin
        AdjVertArray[Vert]^.NumOfAdjVert := 0; AltAdjVertArray[Vert,1] := [];
        AltAdjVertArray[Vert,2] := [];
        WChromDegArray[Vert] := 0; WDegArray[Vert] := 0;
      end;
    for I := 1 to 6 do
      readln (DataIn);
    readln (DataIn,NumOfVert); readln (DataIn); readln (DataIn,NumOfEdges);
    readln (DataIn); MaxVertChrom := 0;
    for I := 1 to NumOfVert do
      begin
        read (DataIn,Vertex,Chrom);
        if (Chrom>MaxVertChrom) then MaxVertChrom := Chrom;
        ChromArray[Vertex] := Chrom;
      end;
    readln(DataIn);  readln(DataIn);
    for I := 1 to NumOfEdges do
      begin
        read (DataIn,Vertex1,Vertex2);
       AddNewAdjVert1 (Vertex1,Vertex2); AddNewAdjVert1 (Vertex2,Vertex1);
       AddNewAdjVert2 (Vertex1,Vertex2); AddNewAdjVert2 (Vertex2,Vertex1);
        WChromDegArray[Vertex1] := WChromDegArray[Vertex1] +
                                   ChromArray[Vertex2];
        WChromDegArray[Vertex2] := WChromDegArray[Vertex2] +
                                   ChromArray[Vertex1];
        WDegArray[Vertex1] := WDegArray[Vertex1] + 1;
        WDegArray[Vertex2] := WDegArray[Vertex2] + 1;
      end;
  end;

function IsIn1 (Vert : VertexType; AdjListPtr : AdjListPtrType) : boolean;
  var
    I,NumOfAdjVert : word;
    FoundIt : boolean;
  begin
    NumOfAdjVert := AdjListPtr^.NumOfAdjVert; FoundIt := false; I := 1;
    while (not FoundIt)and (I<=NumOfAdjVert) do
      begin
        if (Vert=AdjListPtr^.VertArray[I]) then FoundIt := true; I := I + 1;
      end;
    IsIn1 := FoundIt;
  end;
```

```
function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
  var
    I : word; Vert : VertexType;
  begin
    if (Vert1 < =250) then
      begin I := 1; Vert := Vert1; end
    else
      begin I := 2; Vert := Vert1 - 250; end;
    IsIn2 := Vert in AltAdjVertArray[Vert2,I]
  end;

function IsIn3 (Vert : VertexType; CCPtr : ConnectedCompPtrType) : boolean;
  var
    I : word;
  begin
    if (Vert< =250) then I := 1
    else
      begin I := 2; Vert := Vert - 250; end;
    IsIn3 := Vert in CCPtr^.CC[I]
  end;

{*** Gen an array containg the vertex indices in largest first order ***}
procedure GenLF1VertOrderArray;
  var
    I,J,K : word; Temp : VertexOrderArrayType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert:= I; VertOrderArray[I].Chrom:= ChromArray[I];
        VertOrderArray[I].WChromDeg := WChromDegArray[I];
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    for I := 1 to (NumOfVert-1) do
      begin
        K := I;
        for J := (I+1) to NumOfVert do
          if (VertOrderArray[J].Chrom>VertOrderArray[K].Chrom) then K := J
          else if (VertOrderArray[J].Chrom=VertOrderArray[K].Chrom) and
            (VertOrderArray[J].WChromDeg>VertOrderArray[K].WChromDeg)
             then K := J
          else if (VertOrderArray[J].Chrom=VertOrderArray[K].Chrom) and
            (VertOrderArray[J].WChromDeg=VertOrderArray[K].WChromDeg)
            and (VertOrderArray[J].WDeg>VertOrderArray[K].WDeg) then
            K := J;
```

```
        if (K<>I) then
          begin
            Temp := VertOrderArray[I];
            VertOrderArray[I] := VertOrderArray[K];
            VertOrderArray[K] := Temp;
          end;
      end;
  end;

{*** Gen an array containg the vertex indices in smallest last order ***}
procedure GenSL1VertOrderArray;
  var
    I,J,K : word; Temp : VertexOrderArrayType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].Chrom := ChromArray[I];
        VertOrderArray[I].WChromDeg := WChromDegArray[I];
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    for I := 1 to (NumOfVert-1) do
      begin
        K := I;
        for J := (I+1) to NumOfVert do
          if (VertOrderArray[J].Chrom<VertOrderArray[K].Chrom) then K := J
          else if (VertOrderArray[J].Chrom=VertOrderArray[K].Chrom) and
            (VertOrderArray[J].WChromDeg<VertOrderArray[K].WChromDeg)
            then K := J
          else if (VertOrderArray[J].Chrom=VertOrderArray[K].Chrom) and
            (VertOrderArray[J].WChromDeg=VertOrderArray[K].WChromDeg)
            and (VertOrderArray[J].WDeg<VertOrderArray[K].WChromDeg) then
            K := J;
        if (K<>I) then
          begin
            Temp := VertOrderArray[I];
            VertOrderArray[I] := VertOrderArray[K];
            VertOrderArray[K] := Temp;
          end;
        for J := (I+1) to NumOfVert do
          if (IsIn2(VertOrderArray[I].Vert,
            VertOrderArray[J].Vert)) then
            begin
              VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
              VertOrderArray[J].WChromDeg :=
```

```
                    VertOrderArray[J].WChromDeg - VertOrderArray[I].WChromDeg;
                end;
            end;
        for I := 1 to (NumOfVert div 2) do
            begin
                Temp := VertOrderArray[I];
                VertOrderArray[I] := VertOrderArray[NumOfVert-I+1];
                VertOrderArray[NumOfVert-I+1] := Temp;
            end;
    end;

procedure GenSmallestFeasibleColor (MaxClr:ColorType; Vertex:VertexType; var
Color:ColorType);
    var
        VertexChrom : byte; FoundOne : boolean; Clr : ColorType;
    begin
        VertexChrom := ChromArray[Vertex]; FoundOne := false; Clr := 1;
        while (Clr<=MaxClr) and (not FoundOne) do
            if (ColorSetsArray[Clr]^.ColorSets[VertexChrom] *
                    AdjColorArray[Vertex]=[]) then
                begin Color := Clr; FoundOne := true; end
            else Clr := Clr + 1;
        if (not FoundOne) then Color := MaxClr + 1;
    end;

procedure CheckForPrevProc (Vert:VertexType; var AlreadyProcessed : boolean);
    var
        CCPtr : ConnectedCompPtrType;
    begin
        AlreadyProcessed := false; CCPtr := CCHeadPtr^.NextPtr;
        while (CCPtr<>nil) and (not AlreadyProcessed) do
            if IsIn3(Vert,CCPtr) then AlreadyProcessed := true
            else CCPtr := CCPtr^.NextPtr;
    end;

procedure AddNewConnectedComponent (Vert : VertexType);
    var
        CCPtr : ConnectedCompPtrType;
    begin
        new (CCPtr); CCPtr^.CC[1] := [];
        CCPtr^.CC[2] := []; CCPtr^.NumOfCCVert := 0;
        CCPtr^.NextPtr := nil; CCRearPtr^.NextPtr := CCPtr; CCRearPtr := CCPtr;
    end;
```

```
procedure GenConnectedComponents (Vertex,Vert1 : VertexType;
                                  ClrSet1,ClrSet2 : ColorSetType;
                                  var FoundClrSet1,FoundClrSet2,FoundAnInterchange
                                  : boolean);
  var
    I,NumOfAdjVert,NumOfCCVert : word;
    Vert2 : VertexType; AdjPtr : AdjListPtrType;
  begin
    if (IsIn2(Vert1,Vertex)) then
      if (AssgnArray[Vert1]<=ClrSet1) then FoundClrSet1 := true
      else FoundClrSet2 := true;
    if (FoundClrSet1) and (FoundClrSet2) then FoundAnInterchange := false;
    if (Vert1<=250) then CCRearPtr^.CC[1] := CCRearPtr^.CC[1] + [Vert1]
    else CCRearPtr^.CC[2] := CCRearPtr^.CC[2] + [Vert1-250];
    CCRearPtr^.NumOfCCVert := CCRearPtr^.NumOfCCVert + 1;
    NumOfCCVert := CCRearPtr^.NumOfCCVert;
    CCRearPtr^.CCArray[NumOfCCVert] := Vert1;
    AdjPtr := AdjVertArray[Vert1];
    NumOfAdjVert := AdjPtr^.NumOfAdjVert; I := 1;
    while (I<=NumOfAdjVert) and (FoundAnInterchange) do
      begin
        Vert2 := AdjPtr^.VertArray[I];
       if ((AssgnArray[Vert2]*ClrSet1<>[]) or (AssgnArray[Vert2]*ClrSet2<>[]))
            and (not IsIn3(Vert2,CCRearPtr)) then
          if (not (AssgnArray[Vert2]<=ClrSet1)) and
            (not (AssgnArray[Vert2]<=ClrSet2)) then
            FoundAnInterchange := false
          else
            GenConnectedComponents (Vertex,Vert2,ClrSet1,ClrSet2,
                                    FoundClrSet1,FoundClrSet2,FoundAnInterchange);
        I := I + 1;
      end;
  end;

procedure DoUpdate1 (Vert1 : VertexType; Color : ColorType);
  var
    StillIn,Again : boolean; Clr2,MaxClr2 : ColorType; Vert2,Vert3 : VertexType;
    I1,NumOfAdjVert1,I2,NumOfAdjVert2 : word;
    AdjPtr1,AdjPtr2 : AdjListPtrType;
  begin
    NumOfAdjVert1 := AdjVertArray[Vert1]^.NumOfAdjVert;
    for I1 := 1 to NumOfAdjVert1 do
      begin
        Vert2 := AdjVertArray[Vert1]^.VertArray[I1];
```

```
          MaxClr2 := Color + ChromArray[Vert1] - 1;
          for Clr2 := Color to MaxClr2 do
            begin
              StillIn := false; Again := true;
              NumOfAdjVert2 := AdjVertArray[Vert2]^.NumOfAdjVert; I2 := 1;
              while (I2<=NumOfAdjVert2) and (Again) do
                begin
                  Vert3 := AdjVertArray[Vert2]^.VertArray[I2];
                  if (Clr2 in AssgnArray[Vert3]) then
                    begin StillIn := true; Again := false;  end
                  else I2 := I2 + 1;
                end;
              if (not StillIn) then
                AdjColorArray[Vert2] := AdjColorArray[Vert2] - [Clr2];
            end;
        end;
    end;

procedure DoUpdate2 (Vert1 : VertexType; Color : ColorType);
    var
      Clr2,MaxClr2 : ColorType; Vert2 : VertexType;
      I,NumOfAdjVert : word;
    begin
      NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
      for I := 1 to NumOfAdjVert do
        begin
          Vert2 := AdjVertArray[Vert1]^.VertArray[I];
          MaxClr2 := Color + ChromArray[Vert1] - 1;
          for Clr2 := Color to MaxClr2 do
            if (not (Clr2 in AdjColorArray[Vert2])) then
              AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Clr2];
        end;
    end;

procedure DoInterchange (Vertex : VertexType;ClrSetSize : byte;
                         Clr1,Clr2 : ColorType;
                         ClrSet1,ClrSet2 : ColorSetType);
    var
      SwapColors,FoundMinClrIndex : boolean; MinClr1,MinClr2,Clr : ColorType;
      NewAssgn : ColorSetType; I,J,NumOfCCVert : word; Vert : VertexType;
      ClrArray1,ClrArray2 : array[1..MaxColor] of ColorType;
      CCPtr : ConnectedCompPtrType;
    begin
      Clr := Clr1;
```

```
for I := 1 to ClrSetSize do
  begin ClrArray1[I] := Clr; Clr := Clr + 1; end;
Clr := Clr2;
for I := 1 to ClrSetSize do
  begin ClrArray2[I] := Clr; Clr := Clr + 1; end;
CCPtr:= CCHeadPtr^.NextPtr;
while (CCPtr<>nil) do
  begin
    SwapColors := false; NumOfCCVert := CCPtr^.NumOfCCVert; I := 1;
    while (I<=NumOfCCVert) and (not SwapColors) do
      begin
        Vert := CCPtr^.CCArray[I];
        if (AssgnArray[Vert]<=ClrSet1) and IsIn2(Vert,Vertex) then
          SwapColors := true
        else I := I + 1;
      end;
    if (SwapColors) then
      begin
        I := 1;
        while (I<=NumOfCCVert) do
          begin
            Vert := CCPtr^.CCArray[I];
            if (AssgnArray[Vert]<=ClrSet1) then
              begin
                FoundMinClrIndex := false; J := 1;
                while (not FoundMinClrIndex) do
                  if (ClrArray1[J] in AssgnArray[Vert]) then
                    FoundMinClrIndex := true
                 else J := J + 1;
                MinClr1 := ClrArray1[J]; MinClr2 := ClrArray2[J];
                NewAssgn := [];
                for J := 1 to ClrSetSize do
                  if (ClrArray1[J] in AssgnArray[Vert]) then
                    NewAssgn := NewAssgn + [ClrArray2[J]];
                AssgnArray[Vert] := [];
                DoUpdate1 (Vert,MinClr1);
                AssgnArray[Vert] := NewAssgn;
                DoUpdate2 (Vert,MinClr2);
              end
            else
              begin
                FoundMinClrIndex := false; J := 1;
                while (not FoundMinClrIndex) do
                  if (ClrArray2[J] in AssgnArray[Vert]) then
```

```
                        FoundMinClrIndex := true
                      else J := J + 1;
                    MinClr1 := ClrArray1[J]; MinClr2 := ClrArray2[J];
                    NewAssgn := [];
                    for J := 1 to ClrSetSize do
                      if (ClrArray2[J] in AssgnArray[Vert]) then
                        NewAssgn := NewAssgn + [ClrArray1[J]];
                    AssgnArray[Vert] := [];
                    DoUpdate1 (Vert,MinClr2);
                    AssgnArray[Vert] := NewAssgn;
                    DoUpdate2 (Vert,MinClr1);
                  end;
                I := I + 1;
              end;
          end;
        CCPtr := CCPtr^.NextPtr;
      end;
  end;

procedure ReclaimDynamicMemory;
  begin
    CCHeadPtr^.NextPtr := nil; CCRearPtr := CCHeadPtr; release (StackPtr);
  end;

procedure AttemptAnInterchange (Vertex : VertexType; var Color : ColorType);
  var
    AlreadyProcessed,FoundAnInterchange,FoundClrSet1,FoundClrSet2 : boolean;
    ClrSetSize,SizePlus1,MaxClrSetSize : byte;
    Clr1Uplimit,Clr2Uplimit,Clr2MinusClr1 : integer;
    I,NumOfAdjVert : word; Vert2,Vert : VertexType;
    MaxClr,Clr1,Clr2 : ColorType;
    ClrSet1,ClrSet2 : ColorSetType; AdjPtr : AdjListPtrType;
  begin
    MaxClr := Color - 1; FoundAnInterchange := false;
    ClrSetSize := ChromArray[Vertex]; SizePlus1 := ClrSetSize + 1;
    if (SizePlus1 < MaxVertChrom) then MaxClrSetSize := SizePlus1
    else MaxClrSetSize := MaxVertChrom;
    while (ClrSetSize < = MaxClrSetSize) and (not FoundAnInterchange) do
      begin
        Clr1Uplimit := MaxClr - 2*ClrSetSize + 1;
        if (Clr1Uplimit < = 0) then Clr1Uplimit := 1;
        Clr1 := 1;
        while (Clr1 < = Clr1Uplimit) and (not FoundAnInterchange) do
          begin
```

```
    ClrSet1 := ColorSetsArray[Clr1]^.ColorSets[ClrSetSize];
    Clr2 := Clr1 + ClrSetSize;
    Clr2Uplimit := MaxClr - ClrSetSize + 1;
    Clr2 := Clr1 + ClrSetSize;
    Clr2MinusClr1 := Clr2 - Clr1;
    while (Clr2 <= Clr2Uplimit) and (Clr2MinusClr1 >= ClrSetSize)
        and (not FoundAnInterchange) do
      begin
        ClrSet2 := ColorSetsArray[Clr2]^.ColorSets[ClrSetSize];
        FoundAnInterchange := true;
        AdjPtr := AdjVertArray[Vertex];
        NumOfAdjVert := AdjPtr^.NumOfAdjVert; I := 1;
        while (I <= NumOfAdjVert) and (FoundAnInterchange) do
          begin
            Vert := AdjPtr^.VertArray[I];
            FoundClrSet1 := false; FoundClrSet2 := false;
            if ((AssgnArray[Vert]*ClrSet1 <> []) or
                (AssgnArray[Vert]*ClrSet2 <> [])) then
              if (not (AssgnArray[Vert] <= ClrSet1)) and
                 (not (AssgnArray[Vert] <= ClrSet2)) then
                FoundAnInterchange := false
              else
                begin
                  CheckForPrevProc (Vert,AlreadyProcessed);
                  if (not AlreadyProcessed) then
                    begin
                      AddNewConnectedComponent (Vert);
                      GenConnectedComponents (Vertex,
                                Vert,ClrSet1,ClrSet2,
                                FoundClrSet1,FoundClrSet2,
                                FoundAnInterchange);
                    end;
                end;
            I := I + 1;
          end;
        if (FoundAnInterchange) then
          begin
            DoInterchange(Vertex,ClrSetSize,Clr1,Clr2,ClrSet1,ClrSet2);
            Color := Clr1;
          end;
        ReclaimDynamicMemory; Clr2 := Clr2 + 1;
      end;
    Clr1 := Clr1 + 1;
end;
```

```
        ClrSetSize := ClrSetSize + 1;
      end;
  end;

procedure DoCLFColoring;
  var
    I : word; Vertex : VertexType; MaxClr,Clr,Color : ColorType;
  begin
    GenLF1VertOrderArray; Vertex := VertOrderArray[1].Vert;
    for Clr := 1 to ChromArray[Vertex] do
      AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
    LF1Colors := ChromArray[Vertex]; DoUpdate2 (Vertex,1);
    for I := 2 to NumOfVert do
      begin
        Vertex := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (LF1Colors,Vertex, Color);
        MaxClr := Color + ChromArray[Vertex] - 1;
        for Clr := Color to MaxClr do
          AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
        DoUpDate2 (Vertex,Color);
        if (MaxClr>LF1Colors) then LF1Colors := MaxClr;
      end;
  end;

procedure DoCLFI2Coloring;
  var
    I : word; Vertex : VertexType; MaxClr,Clr,Color : ColorType;
  begin
    GenLF1VertOrderArray; Vertex := VertOrderArray[1].Vert;
    for Clr := 1 to ChromArray[Vertex] do
      AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
    LF1I2Colors := ChromArray[Vertex]; DoUpdate2 (Vertex,1);
    for I := 2 to NumOfVert do
      begin
        Vertex := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (LF1I2Colors,Vertex,Color);
        MaxClr := Color + ChromArray[Vertex] - 1;
        if (MaxClr>LF1I2Colors) then
          AttemptAnInterchange (Vertex,Color);
        MaxClr := Color + ChromArray[Vertex] - 1;
        for Clr := Color to MaxClr do
          AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
        DoUpDate2 (Vertex,Color);
        if (MaxClr>LF1I2Colors) then LF1I2Colors := MaxClr;
```

```
      end;
  end;

procedure DoCSLColoring;
  var
    I : word; Vertex : VertexType; MaxClr,Clr,Color : ColorType;
  begin
    GenSL1VertOrderArray; Vertex := VertOrderArray[1].Vert;
    for Clr := 1 to ChromArray[Vertex] do
      AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
    SL1Colors := ChromArray[Vertex]; DoUpdate2 (Vertex,1);
    for I := 2 to NumOfVert do
      begin
        Vertex := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (SL1Colors,Vertex,Color);
        MaxClr := Color + ChromArray[Vertex] - 1;
        for Clr := Color to MaxClr do
          AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
        DoUpDate2 (Vertex,Color);
        if (MaxClr>SL1Colors) then SL1Colors := MaxClr;
      end;
  end;

procedure DoCSLI2Coloring;
  var
    I : word; Vertex : VertexType; MaxClr,Clr,Color : ColorType;
  begin
    GenSL1VertOrderArray; Vertex := VertOrderArray[1].Vert;
    for Clr := 1 to ChromArray[Vertex] do
      AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
    SL1I2Colors := ChromArray[Vertex]; DoUpdate2 (Vertex,1);
    for I := 2 to NumOfVert do
      begin
        Vertex := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (SL1I2Colors,Vertex,Color);
        MaxClr := Color + ChromArray[Vertex] - 1;
        if (MaxClr>SL1I2Colors) then AttemptAnInterchange (Vertex,Color);
        MaxClr := Color + ChromArray[Vertex] - 1;
        for Clr := Color to MaxClr do
          AssgnArray[Vertex] := AssgnArray[Vertex] + [Clr];
        DoUpDate2 (Vertex,Color);
        if (MaxClr>SL1I2Colors) then SL1I2Colors := MaxClr;
      end;
  end;
```

```
procedure CheckAssgnArray (ColorNum,Method : word);
  var
    I,NumOfAdjVert : word; Clr : byte; Vert1,Vert2 : VertexType;
  begin
    FeasibleArray[Method] := true; Vert1 := 1;
    while (Vert1 <=NumOfVert) and (FeasibleArray[Method]) do
      begin
        for Clr := 1 to ColorNum do
          if (Clr in AssgnArray[Vert1]) then
            begin
              NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert; I := 1;
              while (I <=NumOfAdjVert) and (FeasibleArray[Method]) do
                begin
                  Vert2 := AdjVertArray[Vert1]^.VertArray[I];
                  if (Clr in AssgnArray[Vert2]) then
                    begin
                      FeasibleArray[Method] := false;
                      writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                    end
                  else
                    I := I + 1;
                end;
            end;
        Vert1 := Vert1 + 1;
      end;
  end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : double;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Prn,'con'); rewrite (Prn); assign (Out, 'REP3.PRN'); rewrite (Out);
  ObtainUserInfo; PrintReportHeadingsAndInitTotals;
  for Count := BegGraphNum to EndGraphNum do
    begin
      if (Count<10) then
```

```
      begin FileName := FileNamePrefix + '.00'; str(Count:1,Extension); end
    else
      begin FileName := FileNamePrefix + '.0'; str(Count:2,Extension); end;
    FileName := FileName + Extension;
    assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName);
    reset (DataIn); writeln; writeln ('Processing ',FileName);
    ReadData; close (DataIn);
    Initialize; writeln('Working on CLF'); GetTime (Hour1,Min1,Sec1,Frac1);
    DoCLFColoring; GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
    LF1Time := ElapsedTime; AvgLF1Time := AvgLF1Time + LF1Time;
    AvgLF1Colors := AvgLF1Colors + LF1Colors;
    CheckAssgnArray(LF1Colors,1);
    Initialize; writeln('Working on CLFI2'); GetTime (Hour1,Min1,Sec1,Frac1);
    DoCLFI2Coloring; GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
    LF1I2Time := ElapsedTime;
    AvgLF1I2Time := AvgLF1I2Time + LF1I2Time;
    AvgLF1I2Colors := AvgLF1I2Colors + LF1I2Colors;
    CheckAssgnArray(LF1I2Colors,2);
    Initialize; writeln('Working on CSL'); GetTime (Hour1,Min1,Sec1,Frac1);
    DoCSLColoring; GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
    SL1Time := ElapsedTime; AvgSL1Time := AvgSL1Time + SL1Time;
    AvgSL1Colors := AvgSL1Colors + SL1Colors;
    CheckAssgnArray(SL1Colors,3);
    Initialize; writeln('Working on SL1I2'); GetTime (Hour1,Min1,Sec1,Frac1);
    DoCSLI2Coloring; GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
    SL1I2Time:= ElapsedTime; AvgSL1I2Time:= AvgSL1I2Time + SL1I2Time;
    AvgSL1I2Colors := AvgSL1I2Colors + SL1I2Colors;
    CheckAssgnArray(SL1I2Colors,4);
    writeln (UpperCase(FileName):11,LF1Colors:8,LF1I2Colors:11,
          SL1Colors:11,SL1I2Colors:11);
    writeln (LF1Time:22:2,LF1I2Time:11:2,SL1Time:11:2,SL1I2Time:11:2);
    write (' ':10);
    for I := 1 to 4 do
      write (FeasibleArray[I]:11);
    writeln;
    writeln(Out,UpperCase(FileName):11,LF1Colors:8,LF1I2Colors:11,
        SL1Colors:11,SL1I2Colors:11);
    writeln (Out,LF1Time:22:2,LF1I2Time:11:2,SL1Time:11:2,SL1I2Time:11:2);
    write (Out,' ':10);
    for I := 1 to 4 do
      write (Out,FeasibleArray[I]:11);
    writeln (Out);
  end;
AvgLF1Colors := AvgLF1Colors/NumOfGraphs;
```

```
  AvgLF1I2Colors := AvgLF1I2Colors/NumOfGraphs;
  AvgSL1Colors := AvgSL1Colors/NumOfGraphs;
  AvgSL1I2Colors := AvgSL1I2Colors/NumOfGraphs;
  AvgLF1Time := AvgLF1Time/NumOfGraphs;
  AvgLF1I2Time := AvgLF1I2Time/NumOfGraphs;
  AvgSL1Time := AvgSL1Time/NumOfGraphs;
  AvgSL1I2Time := AvgSL1I2Time/NumOfGraphs;
  writeln (Out,'----------  --------  --------',
                        '  --------  --------');
  writeln (Out,'  Averages',AvgLF1Colors:11:2,AvgLF1I2Colors:11:2,
          AvgSL1Colors:11:2,AvgSL1I2Colors:11:2);
  writeln (Out,AvgLF1Time:22:2,AvgLF1I2Time:11:2,
          AvgSL1Time:11:2,AvgSL1I2Time:11:2);
  close (Out);
end.
```

12. CVSI3.Pas -- Implements CDsaturI1

```
{$N+,E-,M 65520,0,655360}
{*** Implements CDsaturI1 ***}
program C_VSI3;
uses DOS,CRT;
const
  MaxVert = 500; {Max no. of vertices}
  MaxColor = 115; {Upper bound for chromatic number}
type
  VertexType = 1..MaxVert; VertexSetType = set of 1..250;
  ColorType = 0..MaxColor;
  ColorSetType = set of ColorType; String11 = string[11];
  VertexOrderArrayType = record
                           Vert : VertexType; Chrom : word; CDeg : word;
                           WChromDeg : word; WDeg : word;
                         end;
  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  NumOfAdjVert : word; VertArray : array[1..300] of VertexType;
                end;
  AltAdjListPtrType = ^AltAdjListType;
  AltAdjListType = record
                     VertSetArray : array[1..2] of VertexSetType;
                   end;
  ColorSetsPtrType = ^ColorSetsType;
  ColorSetsType = record
                    ColorSets : array[1..5] of ColorSetType;
```

```
                end;
var
   FileName,FileNamePrefix,Extension : String11;
   Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
   MaxVertChrom,NumOfVert,NumOfEdges : word;
   I,Count,NumOfGraphs,BegGraphNum,EndGraphNum,DynVS2I1Colors : word;
   DynVS2I1Time,ElapsedTime,AvgElapsedTime : double;
   AvgDynVS2I1Colors,AvgDynVS2I1Time : double;
   SaveVertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
   VertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
   AssgnArray,SaveAssgnArray : array[VertexType] of ColorSetType;
   VertFirstColorArray : array[1..MaxVert] of byte;
   AdjVertArray : array[VertexType] of AdjListPtrType;
   AltAdjVertArray : array[VertexType] of AltAdjListPtrType;
   AdjColorArray,SaveAdjColorArray : array[VertexType] of ColorSetType;
   FeasibleSetArray,SaveFeasibleSetArray : array[VertexType] of ColorSetType;
   ChromArray,WChromDegArray,WDegArray : array[VertexType] of word;
   ColorSetsArray : array [1..MaxColor] of ColorSetsPtrType;
   FeasibleArray : array[1..5] of boolean;
   DataIn,Out,Prn : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
   var
      Pos : word; ConvertedName : String11;
   begin
      ConvertedName := '';
      for Pos := 1 to length(FileName) do
         ConvertedName := ConvertedName + upcase(FileName[Pos]);
      UpperCase := ConvertedName;
   end;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
   var
      Code : integer;
   begin
      ClrScr;
      writeln ('Application:  Composite Graph Coloring'); writeln;
      writeln ('Program:  CVSI3.PAS -- Heuristic'); writeln;
      write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
      write ('Enter Num of Vertices -- '); readln (NumOfVert); writeln;
      write ('Enter Begin Graph Number:  '); readln (BegGraphNum); writeln;
      write ('Enter End Graph Number:  '); readln (EndGraphNum);
```

```
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Color : ColorType; Vert : VertexType; AdjListPtr : AdjListPtrType;
    AltAdjListPtr : AltAdjListPtrType; ColorSetsPtr : ColorSetsPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin new(AdjListPtr); AdjVertArray[Vert] := AdjListPtr; end;
    for Vert := 1 to NumOfVert do
      begin new(AltAdjListPtr); AltAdjVertArray[Vert] := AltAdjListPtr; end;
    for Color := 1 to MaxColor do
      begin
        new(ColorSetsPtr); ColorSetsArray[Color] := ColorSetsPtr;
        ColorSetsPtr^.ColorSets[1] := [Color];
        ColorSetsPtr^.ColorSets[2] := [Color,Color+1];
        ColorSetsPtr^.ColorSets[3] := [Color,Color+1,Color+2];
        ColorSetsPtr^.ColorSets[4] := [Color,Color+1,Color+2,Color+3];
        ColorSetsPtr^.ColorSets[5]:=[Color,Color+1,Color+2,Color+3,Color+4];
      end;
    AvgDynVS2I1Colors := 0.0; AvgDynVS2I1Time := 0.0;
    writeln (Out,'Application: Composite Graph Coloring');
    writeln (Out,'Program:     CVSI3.PAS -- Heuristic');
    writeln (Out,'Ordering:    CDsatur'); writeln (Out);
    writeln (Out,' File Name    CDsaturI1'); writeln (Out,'----------   ---------');
  end;

procedure Initialize;
  var
    Vert : VertexType;
  begin
    for Vert := 1 to MaxVert do
      begin
        AssgnArray[Vert] := []; VertFirstColorArray[Vert] := 0;
        AdjColorArray[Vert] := []; FeasibleSetArray[Vert] := [];
      end;
  end;

{*** Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I,Vert,Vertex,Vertex1,Vertex2,Chrom : word;
```

```
procedure AddNewAdjVertex1 (Vert1,Vert2 : VertexType);
  var
    NumOfAdjVert : word;
  begin
    AdjVertArray[Vert1]^.NumOfAdjVert :=
      AdjVertArray[Vert1]^.NumOfAdjVert + 1;
    NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
    if (NumOfAdjVert>300) then
      writeln ('Vertex = ',Vert1,' NumOfAdjVert = ',NumOfAdjVert);
    AdjVertArray[Vert1]^.VertArray[NumOfAdjVert] := Vert2;
  end;
procedure AddNewAdjVertex2 (Vert1,Vert2 : VertexType);
  var
     I : word; Vert : VertexType;
  begin
    if (Vert2<=250) then begin I := 1; Vert := Vert2; end
    else begin I := 2; Vert := Vert2 - 250; end;
    AltAdjVertArray[Vert1]^.VertSetArray[I] :=
      AltAdjVertArray[Vert1]^.VertSetArray[I] + [Vert];
  end;
begin
  DynVS2I1Colors := 999; MaxVertChrom := 0;
  for Vert := 1 to NumOfVert do
    begin
      AdjVertArray[Vert]^.NumOfAdjVert := 0;
      AltAdjVertArray[Vert]^.VertSetArray[1] := [];
      AltAdjVertArray[Vert]^.VertSetArray[2] := [];
      WChromDegArray[Vert] := 0; WDegArray[Vert] := 0;
    end;
  for I := 1 to 6 do
    readln (DataIn);
  readln (DataIn,NumOfVert); readln (DataIn);
  readln (DataIn,NumOfEdges); readln (DataIn);
  for I := 1 to NumOfVert do
    begin
      read (DataIn,Vertex,Chrom); ChromArray[Vertex] := Chrom;
      if (Chrom>MaxVertChrom) then MaxVertChrom := Chrom;
    end;
  readln(DataIn); readln(DataIn);
  for I := 1 to NumOfEdges do
    begin
      read (DataIn,Vertex1,Vertex2);
      AddNewAdjVertex1(Vertex1,Vertex2);
      AddNewAdjVertex1(Vertex2,Vertex1);
```

```
            AddNewAdjVertex2(Vertex1,Vertex2);
            AddNewAdjVertex2(Vertex2,Vertex1);
            WDegArray[Vertex1] := WDegArray[Vertex1] + 1;
            WDegArray[Vertex2] := WDegArray[Vertex2] + 1;
            WChromDegArray[Vertex1] := WChromDegArray[Vertex1] +
                                       ChromArray[Vertex2];
            WChromDegArray[Vertex2] := WChromDegArray[Vertex2] +
                                       ChromArray[Vertex1];
          end;
      end;

function IsIn1 (Vert : VertexType; AdjListPtr : AdjListPtrType) : boolean;
  var
    I,NumOfAdjVert : word; FoundIt : boolean;
  begin
    NumOfAdjVert := AdjListPtr^.NumOfAdjVert; FoundIt := false; I := 1;
    while (not FoundIt)and (I<=NumOfAdjVert) do
      begin
        if (Vert=AdjListPtr^.VertArray[I]) then FoundIt := true;
        I := I + 1;
      end;
    IsIn1 := FoundIt;
  end;

function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
  var
    I : word; Vert : VertexType;
  begin
    if (Vert1<=250) then begin I := 1; Vert := Vert1; end
    else begin I := 2; Vert := Vert1 - 250; end;
    IsIn2 := Vert in AltAdjVertArray[Vert2]^.VertSetArray[I];
  end;

procedure UpdateFeasibleSets1 (MaxClr : ColorType);
  var
    VertChrom : byte; Vert : VertexType; Color : ColorType;
  begin
    for Vert := 1 to NumOfVert do
      FeasibleSetArray[Vert] := [];
    for Vert := 1 to NumOfVert do
      begin
        VertChrom := ChromArray[Vert];
        for Color := 1 to (MaxClr-ChromArray[Vert]+1) do
          if (ColorSetsArray[Color]^.ColorSets[VertChrom] *
```

```
                AdjColorArray[Vert]=[]) then
              FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Color];
      end;
  end;

procedure UpdateFeasibleSets2 (MaxClr : ColorType);
  var
    Vert : VertexType; Color : ColorType;
  begin
    for Vert := 1 to NumOfVert do
        begin
          Color := MaxClr - ChromArray[Vert] + 1;
          if (ColorSetsArray[Color]^.ColorSets[ChromArray[Vert]] *
              AdjColorArray[Vert]=[]) then
            FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Color];
        end;
  end;

procedure UpdateFeasibleSets3 (MaxClr:ColorType; Vertex:VertexType;
                               VertexFirstColor:ColorType);
  var
    ColorIsFeasible : boolean; Vert : VertexType;
    Clr,StartClr,EndClr : ColorType; I,NumOfAdjVert : word;
  begin
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        StartClr := VertexFirstColor - ChromArray[Vert] + 1;
        EndClr := VertexFirstColor + ChromArray[Vertex] - 1;
        if (StartClr<1) then StartClr := 1;
        if (EndClr > MaxClr) then EndClr := MaxClr;
        for Clr := StartClr to EndClr do
          if (Clr in FeasibleSetArray[Vert]) then
            FeasibleSetArray[Vert] := FeasibleSetArray[Vert] - [Clr];
      end;
  end;

procedure UpdateFeasibleSets4 (MaxClr : ColorType; Vertex : VertexType;
                               VertexFirstColor : ColorType);
  var
    Vert : VertexType; VertChrom : byte;
    Clr,StartClr,EndClr : ColorType; I,NumOfAdjVert : word;
  begin
```

```
    NumOfAdjVert := AdjVertArray[Vertex]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert := AdjVertArray[Vertex]^.VertArray[I];
        VertChrom := ChromArray[Vert];
        StartClr := VertexFirstColor - VertChrom + 1;
        EndClr := VertexFirstColor + ChromArray[Vertex] - 1;
        if (StartClr<1) then StartClr := 1;
        if (EndClr+ChromArray[Vert]-1>MaxClr) then
          EndClr := MaxClr-VertChrom+1;
        for Clr := StartClr to EndClr do
          if (ColorSetsArray[Clr]^.ColorSets[VertChrom] *
              AdjColorArray[Vert]=[]) then
            FeasibleSetArray[Vert] := FeasibleSetArray[Vert] + [Clr];
      end;
  end;

procedure GenSmallestFeasibleColor (MaxClr : ColorType; Vertex : VertexType;
                                    var Color : ColorType);
  var
    FoundIt : boolean; Clr : ColorType;
  begin
    FoundIt := false; Clr := 1;
    while (Clr<=MaxClr) and (not FoundIt) do
      if (Clr in FeasibleSetArray[Vertex]) then
        begin Color := Clr; FoundIt := true; end
      else Clr := Clr + 1;
  end;

procedure DoUpdate1 (Vert1 : VertexType; Color : ColorType);
  var
    StillIn,Again : boolean; Clr2,MaxClr2 : ColorType; Vert2,Vert3 : VertexType;
    I1,J,NumOfAdjVert1,I2,NumOfAdjVert2 : word;
    AdjPtr1,AdjPtr2 : AdjListPtrType;
  begin
    NumOfAdjVert1 := AdjVertArray[Vert1]^.NumOfAdjVert;
    for I1 := 1 to NumOfAdjVert1 do
      begin
        Vert2 := AdjVertArray[Vert1]^.VertArray[I1]; J := 1;
        while (VertOrderArray[J].Vert<>Vert2) do
          J := J + 1;
        VertOrderArray[J].WDeg := VertOrderArray[J].WDeg + 1;
        VertOrderArray[J].WChromDeg := VertOrderArray[J].WChromDeg +
                                       ChromArray[Vert1];
```

```
        MaxClr2 := Color + ChromArray[Vert1] - 1;
        for Clr2 := Color to MaxClr2 do
          begin
            StillIn := false; Again := true;
            NumOfAdjVert2 := AdjVertArray[Vert2]^.NumOfAdjVert; I2 := 1;
            while (I2<=NumOfAdjVert2) and (Again) do
              begin
                Vert3 := AdjVertArray[Vert2]^.VertArray[I2];
                if (Clr2 in AssgnArray[Vert3]) then
                  begin StillIn := true; Again := false; end
                else I2 := I2 + 1;
              end;
            if (not StillIn) then
              begin
                AdjColorArray[Vert2] := AdjColorArray[Vert2] - [Clr2];
                VertOrderArray[J].CDeg := VertOrderArray[J].CDeg - 1;
              end;
          end;
      end;
  end;

procedure DoUpdate2 (Vert1 : VertexType; Color : ColorType);
  var
    Clr2,MaxClr2 : ColorType; Vert2 : VertexType; I,J,NumOfAdjVert : word;
  begin
    NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
    for I := 1 to NumOfAdjVert do
      begin
        Vert2 := AdjVertArray[Vert1]^.VertArray[I]; J := 1;
        while (VertOrderArray[J].Vert<>Vert2) do
          J := J + 1;
        VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
        VertOrderArray[J].WChromDeg := VertOrderArray[J].WChromDeg -
                                       ChromArray[Vert1];
        MaxClr2 := Color + ChromArray[Vert1] - 1;
        for Clr2 := Color to MaxClr2 do
          if (not (Clr2 in AdjColorArray[Vert2])) then
            begin
              AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Clr2];
              VertOrderArray[J].CDeg := VertOrderArray[J].CDeg + 1;
            end;
      end;
  end;
```

```
procedure AttemptAnInterchange1 (MaxClr : ColorType; UpLimit : word; Vertex
: VertexType;
                          var Color : ColorType; var FoundOne,FirstTry : boolean);
  var
    Changed : boolean; I : word; Vert1Chrom : byte;
    Vert1 : VertexType; MaxClr1,Clr1,Vert1FirstColor,Clr2,MaxClr2 : ColorType;
  begin
    SaveFeasibleSetArray := FeasibleSetArray;
    SaveAdjColorArray := AdjColorArray;
    SaveAssgnArray := AssgnArray; SaveVertOrderArray := VertOrderArray;
    FoundOne := false; Changed := false; I := 1;
    while (not FoundOne) and (I<=UpLimit) do
      begin
        if (Changed) then
          begin
            FeasibleSetArray := SaveFeasibleSetArray;
            AdjColorArray := SaveAdjColorArray;
            VertOrderArray := SaveVertOrderArray;
          end;
        Changed := false; Vert1 := VertOrderArray[I].Vert;
        if (IsIn2(Vert1,Vertex)) and (FeasibleSetArray[Vert1]<>[]) then
          begin
            Vert1FirstColor := VertFirstColorArray[Vert1];
            Vert1Chrom := ChromArray[Vert1];
            if (FirstTry) then Clr1 := 1
            else Clr1 := MaxClr - Vert1Chrom + 1;
            MaxClr1 := MaxClr - Vert1Chrom + 1;
            while (Clr1<=MaxClr1) and (not FoundOne) do
               begin
                 if (Changed) then
                   begin
                     FeasibleSetArray := SaveFeasibleSetArray;
                     AdjColorArray := SaveAdjColorArray;
                     VertOrderArray := SaveVertOrderArray;
                   end;
                 if (Clr1 in FeasibleSetArray[Vert1]) and
                    (Clr1<>Vert1FirstColor) then
                   begin
                     Changed := true; AssgnArray[Vert1] := [];
                     DoUpDate1 (Vert1,Vert1FirstColor);
                     UpdateFeasibleSets4 (MaxClr,Vert1,Vert1FirstColor);
                     if (FeasibleSetArray[Vertex]<>[]) then
                       begin
                         MaxClr2 := Clr1 + Vert1Chrom - 1;
```

```
                        for Clr2 := Clr1 to MaxClr2 do
                          AssgnArray[Vert1] := AssgnArray[Vert1] + [Clr2];
                        VertFirstColorArray[Vert1] := Clr1;
                        DoUpdate2 (Vert1,Clr1);
                        UpdateFeasibleSets3(MaxClr,Vert1,Clr1);
                        if (FeasibleSetArray[Vertex]<>[]) then
                          begin
                            FoundOne := true;
                            GenSmallestFeasibleColor (MaxClr,
                                        Vertex,Color);
                          end
                        else
                          begin
                            AssgnArray[Vert1] := SaveAssgnArray[Vert1];
                            VertFirstColorArray[Vert1] := Vert1FirstColor;
                          end;
                      end
                    else
                      begin
                        AssgnArray[Vert1] := SaveAssgnArray[Vert1];
                        VertFirstColorArray[Vert1] := Vert1FirstColor;
                      end;
                  end;
                Clr1 := Clr1 + 1;
              end;
          end;
        I := I + 1;
      end;
    if (not FoundOne) and (Changed) then
      begin
        FeasibleSetArray := SaveFeasibleSetArray;
        AdjColorArray := SaveAdjColorArray;
        VertOrderArray := SaveVertOrderArray;
      end;
    FirstTry := false;
  end;

{*** Gen an array containg the vertex indices in DynVS2 order ***}
procedure GenCDsaturVertOrderArray (I : word);
  var
    J,K : word; Temp : VertexOrderArrayType;
  begin
    K := I;
    for J := (I+1) to NumOfVert do
```

```
      if (VertOrderArray[J].Chrom > VertOrderArray[K].Chrom) then K := J
      else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
        (VertOrderArray[J].CDeg > VertOrderArray[K].CDeg) then K := J
      else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
        (VertOrderArray[J].CDeg = VertOrderArray[K].CDeg) and
        (VertOrderArray[J].WChromDeg > VertOrderArray[K].WChromDeg) then
        K := J
      else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
        (VertOrderArray[J].CDeg = VertOrderArray[K].CDeg) and
        (VertOrderArray[J].WChromDeg = VertOrderArray[K].WChromDeg) and
        (VertOrderArray[J].WDeg > VertOrderArray[K].WDeg) then K := J;
    if (K <> I) then
      begin
        Temp := VertOrderArray[I]; VertOrderArray[I] := VertOrderArray[K];
        VertOrderArray[K] := Temp;
      end;
  end;

procedure UpdateArrays (I : word; Color : ColorType);
  var
    I1,J : word; Vert1,Vert2 : VertexType; Clr,MaxClr : ColorType;
  begin
    Vert1 := VertOrderArray[I].Vert;
    MaxClr := Color + ChromArray[Vert1] - 1;
    for Clr := Color to MaxClr do
      AssgnArray[Vert1] := AssgnArray[Vert1] + [Clr];
    VertFirstColorArray[Vert1] := Color;
    for J := 1 to NumOfVert do
      begin
        Vert2 := VertOrderArray[J].Vert;
        if IsIn2(Vert2,Vert1) then
          begin
            VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
            VertOrderArray[J].WChromDeg :=
                  VertOrderArray[J].WChromDeg - ChromArray[Vert1];
            for Clr := Color to (Color+ChromArray[Vert1]-1) do
              if (not (Clr in AdjColorArray[Vert2])) then
                begin
                  AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Clr];
                  VertOrderArray[J].CDeg := VertOrderArray[J].CDeg + 1;
                end;
          end;
      end;
  end;
```

```
procedure DoCDsaturI1Coloring;
  var
    FoundOne,FirstTry : boolean; I : word; Vert,Vertex : VertexType;
    Clr,Color : ColorType; AdjPtr : AdjListPtrType;
  begin
    DynVS2I1Colors := 0;
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I;
        VertOrderArray[I].Chrom := ChromArray[I];
        VertOrderArray[I].CDeg := 0;
        VertOrderArray[I].WChromDeg := WChromDegArray[I];
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    GenCDsaturVertOrderArray (1); UpdateArrays (1,1);
    DynVS2I1Colors := ChromArray[VertOrderArray[1].Vert];
    UpdateFeasibleSets1 (DynVS2I1Colors);
    for I := 2 to NumOfVert do
      begin
        GenCDsaturVertOrderArray (I); Vertex := VertOrderArray[I].Vert;
        FoundOne := false; FirstTry := true;
        while (not FoundOne) do
          if (FeasibleSetArray[Vertex]<>[]) then
            begin
              FoundOne := true;
              GenSmallestFeasibleColor (DynVS2I1Colors,Vertex, Color);
            end
          else
            begin
              AttemptAnInterchange1 (DynVS2I1Colors,I-1,Vertex,Color,
                                     FoundOne,FirstTry);
              if (not FoundOne) then
                begin
                  DynVS2I1Colors := DynVS2I1Colors + 1;
                  UpdateFeasibleSets2 (DynVS2I1Colors);
                end;
            end;
        UpdateArrays(I,Color);
        UpdateFeasibleSets3 (DynVS2I1Colors,Vertex,Color);
      end;
  end;

procedure CheckAssgnArray (ColorNum,Method : word);
  var
```

```
    I,NumOfAdjVert : word; Clr : byte; Vert1,Vert2 : VertexType;
  begin
    FeasibleArray[Method] := true; Vert1 := 1;
    while (Vert1 <= NumOfVert) and (FeasibleArray[Method]) do
      begin
        for Clr := 1 to ColorNum do
          if (Clr in AssgnArray[Vert1]) then
            begin
              NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert; I := 1;
              while (I <= NumOfAdjVert) and (FeasibleArray[Method]) do
                begin
                  Vert2 := AdjVertArray[Vert1]^.VertArray[I];
                  if (Clr in AssgnArray[Vert2]) then
                    begin
                      FeasibleArray[Method] := false;
                      writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                    end
                  else I := I + 1;
                end;
            end;
        Vert1 := Vert1 + 1;
      end;
  end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : double;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Prn,'con'); rewrite (Prn); assign (Out, 'REP4.PRN'); rewrite (Out);
  ObtainUserInfo; PrintReportHeadingsAndInitTotals;
  for Count := BegGraphNum to EndGraphNum do
    begin
      if (Count<10) then
        begin FileName := FileNamePrefix + '.00'; str(Count:1,Extension); end
      else
        begin FileName := FileNamePrefix + '.0'; str(Count:2,Extension); end;
```

```
        FileName := FileName + Extension;
        assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName); reset (DataIn);
        writeln; writeln ('Processing ',FileName); ReadData; close (DataIn);
        Initialize; writeln('Working on CDsaturI1');
        GetTime (Hour1,Min1,Sec1,Frac1);
        DoCDsaturI1Coloring;GetTime(Hour2,Min2,Sec2,Frac2);CalcElapsedTime;
        DynVS2I1Time := ElapsedTime;
        AvgDynVS2I1Time := AvgDynVS2I1Time + DynVS2I1Time;
        AvgDynVS2I1Colors := AvgDynVS2I1Colors + DynVS2I1Colors;
        CheckAssgnArray(DynVS2I1Colors,1);
        writeln (UpperCase(FileName):11,DynVS2I1Colors:11);
        writeln (DynVS2I1Time:22:2); write (' ':10);
        for I := 1 to 1 do
          write (FeasibleArray[I]:11);
        writeln;
        writeln (Out,UpperCase(FileName):11,DynVS2I1Colors:11);
        writeln (Out,DynVS2I1Time:22:2); write (Out,' ':10);
        for I := 1 to 1 do
          write (Out,FeasibleArray[I]:11);
        writeln (Out);
      end;
    AvgDynVS2I1Colors := AvgDynVS2I1Colors/NumOfGraphs;
    AvgDynVS2I1Time := AvgDynVS2I1Time/NumOfGraphs;
    writeln (Out,'----------  --------');
    writeln (Out,'  Averages',AvgDynVS2I1Colors:11:2);
    writeln (Out,AvgDynVS2I1Time:22:2); close (Out);
end.
```

13. CVSI4.Pas -- Implements CDsatur and CDsaturI2

```
{$N+,E-,M 65520,0,655360}
{*** Implements CDsatur and CDsaturI2 ***}
program C_VSI2;
uses DOS,CRT;
const
  MaxVert = 500; {Max no. of vertices}
  MaxColor = 130; {Upper bound for chromatic number}
type
  VertexType = 1..MaxVert; VertexSetType = set of 1..255;
  ColorType = 0..MaxColor+1;
  ColorSetType = set of ColorType; String11 = string[11];
  VertexOrderArrayType = record
                           Vert : VertexType; Chrom : word; CDeg : word;
                           WChromDeg : word; WDeg : word;
```

```
                    end;
  AdjListPtrType = ^AdjListType;
  AdjListType = record
                  NumOfAdjVert : word; VertArray : array [1..350] of word;
                end;
  ConnectedCompPtrType = ^ConnectedCompType;
  ConnectedCompType = record
                    CC : array[1..2] of VertexSetType; NumOfCCVert : word;
                    CCArray : array[1..300] of word;
                    NextPtr : ConnectedCompPtrType;
                  end;
  ColorSetsPtrType = ^ColorSetsType;
  ColorSetsType = record
                    ColorSets : array[1..5] of ColorSetType;
                  end;
var
  FileName,FileNamePrefix,Extension : String11;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  I,Count,NumOfGraphs,DynVSColors,DynVSI2Colors : word;
  MaxVertChrom,NumOfVert,NumOfEdges,BegGraphNum,EndGraphNum:word;
  DynVSTime,DynVSI2Time,AvgDynVSTime,AvgDynVSI2Time : double;
  AvgDynVSColors,AvgDynVSI2Colors,ElapsedTime : double;
  VertOrderArray : array[1..MaxVert] of VertexOrderArrayType;
  AssgnArray : array[VertexType] of ColorSetType;
  AdjVertArray : array[VertexType] of AdjListPtrType;
  AltAdjVertArray : array[VertexType,1..2] of VertexSetType;
  AdjColorArray : array[VertexType] of ColorSetType;
  ColorSetsArray : array[1..MaxColor] of ColorSetsPtrType;
  ChromArray,WChromDegArray,WDegArray : array[VertexType] of word;
  FeasibleArray : array [1..4] of boolean;
  CCHeadPtr,CCRearPtr : ConnectedCompPtrType;
  StackPtr : ^word;
  DataIn,Out : text;

{*** Convert the file name to all uppercase for printing. ***}
function UpperCase (FileName : String11) : String11;
  var
    Pos : word; ConvertedName : String11;
  begin
    ConvertedName := '';
    for Pos := 1 to length(FileName) do
      ConvertedName := ConvertedName + upcase(FileName[Pos]);
    UpperCase := ConvertedName;
  end;
```

```
{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  var
    Code : integer;
  begin
    ClrScr; writeln ('Application:  Composite Graph Coloring'); writeln;
    writeln ('Program:  CVSI4.PAS -- Heuristic'); writeln;
    write ('Enter File Name Prefix:  '); readln (FileNamePrefix); writeln;
    write ('Enter Number of Vertices:  '); readln (NumOfVert); writeln;
    write ('Enter Begin Graph Number: '); readln (BegGraphNum); writeln;
    write ('Enter End Graph Number: '); readln (EndGraphNum);
    NumOfGraphs := EndGraphNum - BegGraphNum + 1;
  end;

{*** Print report headings. ***}
procedure PrintReportHeadingsAndInitTotals;
  var
    Count : byte; Color : ColorType; Vert : VertexType;
    AdjListPtr : AdjListPtrType; ColorSetsPtr : ColorSetsPtrType;
  begin
    for Vert := 1 to NumOfVert do
      begin new(AdjListPtr); AdjVertArray[Vert] := AdjListPtr; end;
    for Color := 1 to MaxColor do
      begin
        new(ColorSetsPtr); ColorSetsArray[Color] := ColorSetsPtr;
        ColorSetsPtr^.ColorSets[1] := [Color];
        ColorSetsPtr^.ColorSets[2] := [Color,Color+1];
        ColorSetsPtr^.ColorSets[3] := [Color,Color+1,Color+2];
        ColorSetsPtr^.ColorSets[4] := [Color,Color+1,Color+2,Color+3];
        ColorSetsPtr^.ColorSets[5]:=[Color,Color+1,Color+2,Color+3,Color+4];
      end;
    new (CCHeadPtr); CCHeadPtr^.NextPtr := nil; CCRearPtr := CCHeadPtr;
    mark (StackPtr);
    AvgDynVSColors := 0.0; AvgDynVSI2Colors := 0.0;
    AvgDynVSTime := 0.0; AvgDynVSI2Time := 0.0;
    writeln (Out,'Application: Composite Graph Coloring');
    writeln (Out,'Program:     C_VSI4.PAS -- Heuristic');
    writeln (Out,'Ordering:    CDsatur');
    writeln (Out);
    writeln (Out,' File Name     CDsatur   CDsaturI2');
    writeln (Out,'-----------  --------  ---------');
  end;

procedure Initialize;
```

```
  var
    Vert : VertexType;
  begin
    for Vert := 1 to NumOfVert do
      begin AssgnArray[Vert] := []; AdjColorArray[Vert] := []; end;
  end;

{*** Input problem definition data and update graph definition structures. ***}
procedure ReadData;
  var
    I,Vert,Vertex,Vertex1,Vertex2,Chrom : word;
  procedure AddNewAdjVert1 (Vert1,Vert2 : VertexType);
    var
      I : word;
    begin
      AdjVertArray[Vert1]^.NumOfAdjVert :=
        AdjVertArray[Vert1]^.NumOfAdjVert + 1;
      I := AdjVertArray[Vert1]^.NumOfAdjVert;
      AdjVertArray[Vert1]^.VertArray[I] := Vert2;
    end;
  procedure AddNewAdjVert2 (Vert1,Vert2 : VertexType);
    var
      I : word; Vert : VertexType;
    begin
      if (Vert2<=250) then begin I := 1; Vert := Vert2; end
      else begin I := 2; Vert := Vert2 - 250; end;
      AltAdjVertArray[Vert1,I] := AltAdjVertArray[Vert1,I] + [Vert];
    end;
  begin
    DynVSColors := 999; DynVSI2Colors := 999;
    for Vert := 1 to NumOfVert do
      begin
        AdjVertArray[Vert]^.NumOfAdjVert := 0;
        AltAdjVertArray[Vert,1] := []; AltAdjVertArray[Vert,2] := [];
        WChromDegArray[Vert] := 0; WDegArray[Vert] := 0;
      end;
    for I := 1 to 6 do
      readln (DataIn);
    readln (DataIn,NumOfVert); readln (DataIn); readln (DataIn,NumOfEdges);
    readln (DataIn); MaxVertChrom := 0;
    for I := 1 to NumOfVert do
      begin
        read (DataIn,Vertex,Chrom);
        if (Chrom>MaxVertChrom) then MaxVertChrom := Chrom;
```

```
          ChromArray[Vertex] := Chrom;
        end;
      readln(DataIn); readln(DataIn);
      for I := 1 to NumOfEdges do
        begin
          read (DataIn,Vertex1,Vertex2);
          AddNewAdjVert1 (Vertex1,Vertex2); AddNewAdjVert1 (Vertex2,Vertex1);
          AddNewAdjVert2 (Vertex1,Vertex2); AddNewAdjVert2 (Vertex2,Vertex1);
          WChromDegArray[Vertex1] := WChromDegArray[Vertex1] +
                                     ChromArray[Vertex2];
          WChromDegArray[Vertex2] := WChromDegArray[Vertex2] +
                                     ChromArray[Vertex1];
          WDegArray[Vertex1] := WDegArray[Vertex1] + 1;
          WDegArray[Vertex2] := WDegArray[Vertex2] + 1;
        end;
    end;

function IsIn1 (Vert : VertexType; AdjListPtr : AdjListPtrType) : boolean;
  var
    I,NumOfAdjVert : word; FoundIt : boolean;
  begin
    NumOfAdjVert := AdjListPtr^.NumOfAdjVert; FoundIt := false; I := 1;
    while (not FoundIt)and (I<=NumOfAdjVert) do
      begin
        if (Vert=AdjListPtr^.VertArray[I]) then FoundIt := true;
        I := I + 1;
      end;
    IsIn1 := FoundIt;
  end;

function IsIn2 (Vert1,Vert2 : VertexType) : boolean;
  var
    I : word; Vert : VertexType;
  begin
    if (Vert1<=250) then begin I := 1; Vert := Vert1; end
    else begin I := 2; Vert := Vert1 - 250; end;
    IsIn2 := Vert in AltAdjVertArray[Vert2,I]
  end;

function IsIn3 (Vert : VertexType; CCPtr : ConnectedCompPtrType) : boolean;
  var
    I : word;
  begin
    if (Vert<=250) then I := 1
```

```
    else begin I := 2; Vert := Vert - 250; end;
    IsIn3 := Vert in CCPtr^.CC[I]
  end;

procedure GenSmallestFeasibleColor (MaxClr:ColorType; Vertex : VertexType;
                                    var Color : ColorType);
  var
    VertexChrom : byte; FoundOne : boolean; Clr : ColorType;
  begin
    VertexChrom := ChromArray[Vertex]; FoundOne := false; Clr := 1;
    while (Clr<=MaxClr) and (not FoundOne) do
      if (ColorSetsArray[Clr]^.ColorSets[VertexChrom] *
          AdjColorArray[Vertex]=[]) then
        begin Color := Clr; FoundOne := true; end
      else Clr := Clr + 1;
    if (not FoundOne) then Color := MaxClr + 1;
  end;

procedure CheckForPrevProc (Vert:VertexType; var AlreadyProcessed : boolean);
  var
    CCPtr : ConnectedCompPtrType;
  begin
    AlreadyProcessed := false; CCPtr := CCHeadPtr^.NextPtr;
    while (CCPtr<>nil) and (not AlreadyProcessed) do
      if IsIn3(Vert,CCPtr) then AlreadyProcessed := true
      else CCPtr := CCPtr^.NextPtr;
  end;

procedure AddNewConnectedComponent (Vert : VertexType);
  var
    CCPtr : ConnectedCompPtrType;
  begin
    new (CCPtr); CCPtr^.CC[1] := [];
    CCPtr^.CC[2] := []; CCPtr^.NumOfCCVert := 0;
    CCPtr^.NextPtr := nil; CCRearPtr^.NextPtr := CCPtr; CCRearPtr := CCPtr;
  end;

procedure GenConnectedComponents (Vertex,Vert1 : VertexType;
                                  ClrSet1,ClrSet2 : ColorSetType;
                                  var FoundClrSet1,FoundClrSet2,
                                  FoundAnInterchange : boolean);
  var
    I,NumOfAdjVert,NumOfCCVert : word;
    Vert2 : VertexType; AdjPtr : AdjListPtrType;
```

```
  begin
    if (IsIn2(Vert1,Vertex)) then
      if (AssgnArray[Vert1]<=ClrSet1) then FoundClrSet1 := true
      else FoundClrSet2 := true;
    if (FoundClrSet1) and (FoundClrSet2) then FoundAnInterchange := false;
    if (Vert1<=250) then CCRearPtr^.CC[1] := CCRearPtr^.CC[1] + [Vert1]
    else CCRearPtr^.CC[2] := CCRearPtr^.CC[2] + [Vert1-250];
    CCRearPtr^.NumOfCCVert := CCRearPtr^.NumOfCCVert + 1;
    NumOfCCVert := CCRearPtr^.NumOfCCVert;
    CCRearPtr^.CCArray[NumOfCCVert] := Vert1;
    AdjPtr := AdjVertArray[Vert1];
    NumOfAdjVert := AdjPtr^.NumOfAdjVert; I := 1;
    while (I<=NumOfAdjVert) and (FoundAnInterchange) do
      begin
        Vert2 := AdjPtr^.VertArray[I];
       if ((AssgnArray[Vert2]*ClrSet1<>[]) or (AssgnArray[Vert2]*ClrSet2<>[]))
            and (not IsIn3(Vert2,CCRearPtr)) then
          if (not (AssgnArray[Vert2]<=ClrSet1)) and
            (not (AssgnArray[Vert2]<=ClrSet2)) then
            FoundAnInterchange := false
          else
            GenConnectedComponents (Vertex,Vert2,ClrSet1,ClrSet2,
                                    FoundClrSet1,FoundClrSet2,FoundAnInterchange);
        I := I + 1;
      end;
  end;

procedure DoUpdate1 (Vert1 : VertexType; Color : ColorType);
  var
    StillIn,Again : boolean; Clr2,MaxClr2 : ColorType; Vert2,Vert3 : VertexType;
    I1,J,NumOfAdjVert1,I2,NumOfAdjVert2 : word;
    AdjPtr1,AdjPtr2 : AdjListPtrType;
  begin
    NumOfAdjVert1 := AdjVertArray[Vert1]^.NumOfAdjVert;
    for I1 := 1 to NumOfAdjVert1 do
      begin
        Vert2 := AdjVertArray[Vert1]^.VertArray[I1]; J := 1;
        while (VertOrderArray[J].Vert<>Vert2) do
          J := J + 1;
        VertOrderArray[J].WDeg := VertOrderArray[J].WDeg + 1;
        VertOrderArray[J].WChromDeg := VertOrderArray[J].WChromDeg +
                                       ChromArray[Vert1];
        MaxClr2 := Color + ChromArray[Vert1] - 1;
        for Clr2 := Color to MaxClr2 do
```

```
            begin
              StillIn := false; Again := true;
              NumOfAdjVert2 := AdjVertArray[Vert2]^.NumOfAdjVert; I2 := 1;
              while (I2<=NumOfAdjVert2) and (Again) do
                begin
                  Vert3 := AdjVertArray[Vert2]^.VertArray[I2];
                  if (Clr2 in AssgnArray[Vert3]) then
                    begin StillIn := true; Again := false; end
                  else
                    I2 := I2 + 1;
                end;
              if (not StillIn) then
                begin
                  AdjColorArray[Vert2] := AdjColorArray[Vert2] - [Clr2];
                  VertOrderArray[J].CDeg := VertOrderArray[J].CDeg - 1;
                end;
            end;
        end;
    end;

procedure DoUpdate2 (Vert1 : VertexType; Color : ColorType);
    var
      Clr2,MaxClr2 : ColorType; Vert2 : VertexType; I,J,NumOfAdjVert : word;
    begin
      NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert;
      for I := 1 to NumOfAdjVert do
        begin
          Vert2 := AdjVertArray[Vert1]^.VertArray[I]; J := 1;
          while (VertOrderArray[J].Vert<>Vert2) do
            J := J + 1;
          VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
          VertOrderArray[J].WChromDeg := VertOrderArray[J].WChromDeg -
                                         ChromArray[Vert1];
          MaxClr2 := Color + ChromArray[Vert1] - 1;
          for Clr2 := Color to MaxClr2 do
            if (not (Clr2 in AdjColorArray[Vert2])) then
              begin
                AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Clr2];
                VertOrderArray[J].CDeg := VertOrderArray[J].CDeg + 1;
              end;
        end;
    end;

procedure DoInterchange (Vertex : VertexType; ClrSetSize : byte;
```

```
                Clr1,Clr2 : ColorType;
                ClrSet1,ClrSet2 : ColorSetType);
var
  SwapColors,FoundMinClrIndex : boolean; MinClr1,MinClr2,Clr : ColorType;
  NewAssgn : ColorSetType; I,J,NumOfCCVert : word; Vert : VertexType;
  ClrArray1,ClrArray2 : array[1..MaxColor] of ColorType;
  CCPtr : ConnectedCompPtrType;
begin
  Clr := Clr1;
  for I := 1 to ClrSetSize do
    begin ClrArray1[I] := Clr; Clr := Clr + 1; end;
  Clr := Clr2;
  for I := 1 to ClrSetSize do
    begin ClrArray2[I] := Clr; Clr := Clr + 1;  end;
  CCPtr:= CCHeadPtr^.NextPtr;
  while (CCPtr<>nil) do
    begin
      SwapColors := false; NumOfCCVert := CCPtr^.NumOfCCVert; I := 1;
      while (I<=NumOfCCVert) and (not SwapColors) do
        begin
          Vert := CCPtr^.CCArray[I];
          if (AssgnArray[Vert]<=ClrSet1) and IsIn2(Vert,Vertex) then
            SwapColors := true
          else I := I + 1;
        end;
      if (SwapColors) then
        begin
          I := 1;
          while (I<=NumOfCCVert) do
            begin
              Vert := CCPtr^.CCArray[I];
              if (AssgnArray[Vert]<=ClrSet1) then
                begin
                  FoundMinClrIndex := false; J := 1;
                  while (not FoundMinClrIndex) do
                    if (ClrArray1[J] in AssgnArray[Vert]) then
                      FoundMinClrIndex := true
                   else J := J + 1;
                  MinClr1 := ClrArray1[J]; MinClr2 := ClrArray2[J];
                  NewAssgn := [];
                  for J := 1 to ClrSetSize do
                    if (ClrArray1[J] in AssgnArray[Vert]) then
                      NewAssgn := NewAssgn + [ClrArray2[J]];
                  AssgnArray[Vert] := []; DoUpdate1 (Vert,MinClr1);
```

```
                    AssgnArray[Vert]:=NewAssgn;DoUpdate2(Vert,MinClr2);
                  end
                else
                  begin
                    FoundMinClrIndex := false; J := 1;
                    while (not FoundMinClrIndex) do
                      if (ClrArray2[J] in AssgnArray[Vert]) then
                        FoundMinClrIndex := true
                      else J := J + 1;
                    MinClr1 := ClrArray1[J]; MinClr2 := ClrArray2[J];
                    NewAssgn := [];
                    for J := 1 to ClrSetSize do
                      if (ClrArray2[J] in AssgnArray[Vert]) then
                        NewAssgn := NewAssgn + [ClrArray1[J]];
                    AssgnArray[Vert] := []; DoUpdate1 (Vert,MinClr2);
                    AssgnArray[Vert]:=NewAssgn;DoUpdate2(Vert,MinClr1);
                  end;
                I := I + 1;
              end;
          end;
        CCPtr := CCPtr^.NextPtr;
      end;
  end;

procedure ReclaimDynamicMemory;
  begin
    CCHeadPtr^.NextPtr := nil; CCRearPtr := CCHeadPtr; release (StackPtr);
  end;

procedure AttemptAnInterchange (Vertex : VertexType; var Color : ColorType);
  var
    AlreadyProcessed,FoundAnInterchange,FoundClrSet1,FoundClrSet2 : boolean;
    ClrSetSize,SizePlus1,MaxClrSetSize : byte;
    Clr1Uplimit,Clr2Uplimit,Clr2MinusClr1 : integer;
    I,NumOfAdjVert : word; Vert2,Vert : VertexType;
    MaxClr,Clr1,Clr2 : ColorType;
    ClrSet1,ClrSet2 : ColorSetType; AdjPtr : AdjListPtrType;
  begin
    MaxClr := Color - 1; FoundAnInterchange := false;
    ClrSetSize := ChromArray[Vertex]; SizePlus1 := ClrSetSize + 1;
    if (SizePlus1<MaxVertChrom) then MaxClrSetSize := SizePlus1
    else MaxClrSetSize := MaxVertChrom;
    while (ClrSetSize<=MaxClrSetSize) and (not FoundAnInterchange) do
     begin
```

```
Clr1Uplimit := MaxClr - 2*ClrSetSize + 1;
if (Clr1Uplimit<=0) then Clr1Uplimit := 1;
Clr1 := 1;
while (Clr1<=Clr1Uplimit) and (not FoundAnInterchange) do
  begin
    ClrSet1 := ColorSetsArray[Clr1]^.ColorSets[ClrSetSize];
    Clr2 := Clr1 + ClrSetSize;
    Clr2Uplimit := MaxClr - ClrSetSize + 1;
    Clr2 := Clr1 + ClrSetSize;
    Clr2MinusClr1 := Clr2 - Clr1;
    while (Clr2<=Clr2Uplimit) and (Clr2MinusClr1>=ClrSetSize)
        and (not FoundAnInterchange) do
      begin
        ClrSet2 := ColorSetsArray[Clr2]^.ColorSets[ClrSetSize];
        FoundAnInterchange := true;
        AdjPtr := AdjVertArray[Vertex];
        NumOfAdjVert := AdjPtr^.NumOfAdjVert; I := 1;
        while (I<=NumOfAdjVert) and (FoundAnInterchange) do
          begin
            Vert := AdjPtr^.VertArray[I]; FoundClrSet1 := false;
            FoundClrSet2 := false;
            if ((AssgnArray[Vert]*ClrSet1<>[]) or
                (AssgnArray[Vert]*ClrSet2<>[])) then
              if (not (AssgnArray[Vert]<=ClrSet1)) and
                 (not (AssgnArray[Vert]<=ClrSet2)) then
                FoundAnInterchange := false
              else
                begin
                  CheckForPrevProc (Vert,AlreadyProcessed);
                  if (not AlreadyProcessed) then
                    begin
                      AddNewConnectedComponent (Vert);
                      GenConnectedComponents (Vertex,
                              Vert,ClrSet1,ClrSet2,
                              FoundClrSet1,FoundClrSet2,
                              FoundAnInterchange);
                    end;
                end;
            I := I + 1;
          end;
        if (FoundAnInterchange) then
          begin
            DoInterchange(Vertex,ClrSetSize,Clr1,Clr2,ClrSet1,ClrSet2);
            Color := Clr1;
```

```
                    end;
                  ReclaimDynamicMemory; Clr2 := Clr2 + 1;
                end;
              Clr1 := Clr1 + 1;
            end;
          ClrSetSize := ClrSetSize + 1;
        end;
    end;

{*** Gen an array containg the vertex indices in CDsatur order ***}
procedure GenCDsaturVertOrderArray (I : word);
  var
    J,K : word; Temp : VertexOrderArrayType;
  begin
    K := I;
    for J := (I+1) to NumOfVert do
      if (VertOrderArray[J].Chrom > VertOrderArray[K].Chrom) then K := J
      else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
           (VertOrderArray[J].CDeg > VertOrderArray[K].CDeg) then K := J
      else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
           (VertOrderArray[J].CDeg = VertOrderArray[K].CDeg) and
           (VertOrderArray[J].WChromDeg > VertOrderArray[K].WChromDeg)
           then K := J
      else if (VertOrderArray[J].Chrom = VertOrderArray[K].Chrom) and
           (VertOrderArray[J].CDeg = VertOrderArray[K].CDeg) and
           (VertOrderArray[J].WChromDeg = VertOrderArray[K].WChromDeg)
           and (VertOrderArray[J].WDeg > VertOrderArray[K].WDeg) then
           K := J;
    if (K>I) then
      begin
        Temp := VertOrderArray[I]; VertOrderArray[I] := VertOrderArray[K];
        VertOrderArray[K] := Temp;
      end;
  end;

procedure UpdateArrays (I : word; Color : ColorType);
  var
    I1,J : word; Vert1,Vert2 : VertexType; Clr,MaxClr : ColorType;
  begin
    Vert1 := VertOrderArray[I].Vert; MaxClr := Color + ChromArray[Vert1] - 1;
    for Clr := Color to MaxClr do
      AssgnArray[Vert1] := AssgnArray[Vert1] + [Clr];
    for J := 1 to NumOfVert do
      begin
```

```
        Vert2 := VertOrderArray[J].Vert;
        if IsIn2(Vert2,Vert1) then
          begin
            VertOrderArray[J].WDeg := VertOrderArray[J].WDeg - 1;
            VertOrderArray[J].WChromDeg :=
                  VertOrderArray[J].WChromDeg - ChromArray[Vert1];
            for Clr := Color to (Color+ChromArray[Vert1]-1) do
              if (not (Clr in AdjColorArray[Vert2])) then
                begin
                  AdjColorArray[Vert2] := AdjColorArray[Vert2] + [Clr];
                  VertOrderArray[J].CDeg := VertOrderArray[J].CDeg + 1;
                end;
          end;
      end;
  end;

procedure DoCDsaturColoring;
  var
    I : word; Vert : VertexType; Color,ColorPlus : ColorType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].Chrom := ChromArray[I];
        VertOrderArray[I].CDeg := 0;
        VertOrderArray[I].WChromDeg := WChromDegArray[I];
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    GenCDsaturVertOrderArray (1); UpdateArrays(1,1);
    DynVSColors := ChromArray[VertOrderArray[1].Vert];
    for I := 2 to NumOfVert do
      begin
        GenCDsaturVertOrderArray (I); Vert := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (DynVSColors,Vert, Color);
        UpdateArrays(I,Color); ColorPlus := Color + ChromArray[Vert] - 1;
        if (ColorPlus>DynVSColors) then DynVSColors := ColorPlus;
      end;
  end;

procedure CheckAssgnArray (ColorNum,Method : word);
  var
    I,NumOfAdjVert : word; Clr : byte; Vert1,Vert2 : VertexType;
  begin
    FeasibleArray[Method] := true; Vert1 := 1;
    while (Vert1<=NumOfVert) and (FeasibleArray[Method]) do
```

```
      begin
        for Clr := 1 to ColorNum do
          if (Clr in AssgnArray[Vert1]) then
            begin
              NumOfAdjVert := AdjVertArray[Vert1]^.NumOfAdjVert; I := 1;
              while (I<=NumOfAdjVert) and (FeasibleArray[Method]) do
                begin
                  Vert2 := AdjVertArray[Vert1]^.VertArray[I];
                  if (Clr in AssgnArray[Vert2]) then
                    begin
                      FeasibleArray[Method] := false;
                      writeln ('Vert1 = ',Vert1,' Vert2 = ',Vert2);
                    end
                  else I := I + 1;
                end;
            end;
        Vert1 := Vert1 + 1;
      end;
  end;

procedure DoCDsaturI2Coloring;
  var
    I : word; Vert : VertexType; Color,ColorPlus : ColorType;
  begin
    for I := 1 to NumOfVert do
      begin
        VertOrderArray[I].Vert := I; VertOrderArray[I].Chrom := ChromArray[I];
        VertOrderArray[I].CDeg := 0;
        VertOrderArray[I].WChromDeg := WChromDegArray[I];
        VertOrderArray[I].WDeg := WDegArray[I];
      end;
    GenCDsaturVertOrderArray (1); UpdateArrays(1,1);
    DynVSI2Colors := ChromArray[VertOrderArray[1].Vert];
    for I := 2 to NumOfVert do
      begin
        GenCDsaturVertOrderArray (I); Vert := VertOrderArray[I].Vert;
        GenSmallestFeasibleColor (DynVSI2Colors,Vert,Color);
        ColorPlus := Color + ChromArray[Vert] - 1;
        if (ColorPlus>DynVSI2Colors) then
          AttemptAnInterchange (Vert,Color);
        UpdateArrays(I,Color);
        ColorPlus := Color + ChromArray[Vert] - 1;
        if (ColorPlus>DynVSI2Colors) then DynVSI2Colors := ColorPlus;
      end;
```

```
  end;

{*** Calculate the ellapsed time between two check points. ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : double;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Out, 'REP5.PRN'); rewrite (Out);
  ObtainUserInfo; PrintReportHeadingsAndInitTotals;
  for Count := BegGraphNum to EndGraphNum do

      if (Count<10) then
        begin FileName := FileNamePrefix + '.00'; str(Count:1,Extension); end
      else
        begin FileName := FileNamePrefix + '.0'; str(Count:2,Extension); end;
      FileName := FileName + Extension;
      assign (DataIn, 'C:\THESIS\GRAPHS\' + FileName); reset (DataIn);
      writeln; writeln ('Processing ',FileName); ReadData; close (DataIn);
      Initialize; writeln('Working on CDsatur'); GetTime (Hour1,Min1,Sec1,Frac1);
      DoCDsaturColoring; GetTime (Hour2,Min2,Sec2,Frac2); CalcElapsedTime;
      DynVSTime := ElapsedTime;
      AvgDynVSTime := AvgDynVSTime + DynVSTime;
      AvgDynVSColors := AvgDynVSColors + DynVSColors;
      CheckAssgnArray(DynVSColors,1);
      Initialize; writeln('Working on CDsaturI2');
      GetTime (Hour1,Min1,Sec1,Frac1);
      DoCDsaturI2Coloring;GetTime(Hour2,Min2,Sec2,Frac2);CalcElapsedTime;
      DynVSI2Time := ElapsedTime;
      AvgDynVSI2Time := AvgDynVSI2Time + DynVSI2Time;
      AvgDynVSI2Colors := AvgDynVSI2Colors + DynVSI2Colors;
      CheckAssgnArray(DynVSI2Colors,2);
      writeln (UpperCase(FileName):11,DynVSColors:8,DynVSI2Colors:11);
      writeln (DynVSTime:22:2,DynVSI2Time:11:2); write (' ':10);
      for I := 1 to 2 do
        write (FeasibleArray[I]:11);
      writeln;
      writeln (Out,UpperCase(FileName):11,DynVSColors:8,DynVSI2Colors:11);
```

```
        writeln (Out,DynVSTime:22:2,DynVSI2Time:11:2); write (Out,' ':10);
        for I := 1 to 2 do
          write (Out,FeasibleArray[I]:11);
        writeln (Out);
      end;
    AvgDynVSColors := AvgDynVSColors/NumOfGraphs;
    AvgDynVSI2Colors := AvgDynVSI2Colors/NumOfGraphs;
    AvgDynVSTime := AvgDynVSTime/NumOfGraphs;
    AvgDynVSI2Time := AvgDynVSI2Time/NumOfGraphs;
    writeln (Out,'----------  --------  --------');
    writeln (Out,'  Averages',AvgDynVSColors:11:2,AvgDynVSI2Colors:11:2);
    writeln (Out,AvgDynVSTime:22:2,AvgDynVSI2Time:11:2); close (Out);
end.
```

14. S_LB1.Pas

```
{$N+,E-}
program S_LB1;
uses DOS,CRT;
var
  FoundIt : boolean;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  N,StartN,EndN,StepSize,K,I,J,MaxClassSize : word;
  Num,Denom,Value1,Value2 : word;
  SumA,NumOfPaths,PrevNumOfPaths : longint;
  P,Q,LnQ,ElapsedTime : double;
  ExpNumOfKColorings,PrevExpNumOfKColorings : double;
  LnExpNumOfKColorings,NewExpTerm : double;
  AArray,NJArray : array [0..100] of word;
  Comb2Array : array [0..2000] of longint;
  LnFactArray : array [0..2000] of double;
  FileIn,Out : text;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  begin
    ClrScr; writeln ('Application: Standard Graph Coloring');
    writeln ('Program: S_LB1.Pas'); write ('Enter Edge Density (P): ');
    readln (P);  write ('Enter Start Graph Size: ');
    readln (StartN); write ('Enter End Graph Size: ');
    readln (EndN); write ('Enter Step Size: '); readln (StepSize);
    write ('Enter Initial Value of K: '); readln (K);
  end;
```

```
{*** Print report headings. ***}
procedure PrintReportHeadings;
  var
    Count : byte;
  begin
    writeln (Out,'Application:   Standard Graph Coloring');
    writeln (Out,'Program:       S_LB1.Pas -- Explicit Enumeration');
    writeln (Out,'Purpose:       Calculate Lower Bound for Chromatic Number');
    writeln (Out,'Edge Density : ',P:4:2); writeln (Out);
    writeln (Out,'                      Exp Num Of        Num Of ',
                 '                   Exp Num Of        Num Of ');
    writeln (Out,'  N       K             K Colorings         Paths ',
                 '     K             K Colorings         Paths ');
    writeln (Out,'-----   -----   ----------------------   -------',
                 '   -----   ----------------------   -------');
  end;

{*** Input an external file which contains values for the natural log of ***}
{    the number of combinations of n taken 2 and n! for n=1,2,...,2000      }
procedure ReadCombFactFile;
  var
    J : byte; N,I : longint;
  begin
    assign (FileIn,'COMB_FAC.DAT'); reset (FileIn);
    for I := 1 to 9 do
      readln (FileIn);
    J := 0;
    Comb2Array[0] := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,Comb2Array[N]);
        J := J + 1;
        if (J=8) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    readln (FileIn); readln (FileIn);
    LnFactArray[0] := 0;
    J := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,LnFactArray[N]);
```

```
          J := J + 1;
          if (J=4) then
            begin
              readln (FileIn);
              J := 0;
            end;
        end;
      close (FileIn);
    end;

{*** Initialize variables ***}
procedure Initialize1;
  var
    I : word;
  begin
    AArray[0] := 255; Q := 1.0 - P; LnQ := ln(Q); writeln ('Q = ',Q:10:8);
    for I := 1 to 1000 do
      NJArray[I] := 0;
  end;

{*** Initialize variables ***}
procedure Initialize2;
  var
    I : byte;
  begin
    for I := 1 to K do
      AArray[I] := 1;
    SumA := 0; ExpNumOfKColorings := 1.0E-300;
    LnExpNumOfKColorings := -87.49823353;
    NumOfPaths := 0; NewExpTerm := LnFactArray[N];
  end;

{*** Recursive procedure which calculates the value of E(K) by generating ***}
{    all sequences of length k which sum to n.                              }
procedure TraverseTree (Level : byte);
  var
    I,Index : byte; MaxA,MinA : word;
    PrevSumA : longint; PrevNewExpTerm : double;
  begin
    if (Level=K) then
      begin
        PrevNewExpTerm := NewExpTerm;
        AArray[Level] := N - SumA;
        Index := AArray[Level];
```

```
      NJArray[Index] := NJArray[Index] + 1;
      NewExpTerm := NewExpTerm - LnFactArray[Index] +
                    Comb2Array[Index] * LnQ;
      for I := 1 to AArray[1] do
        NewExpTerm := NewExpTerm - LnFactArray[NJArray[I]];
      if (NewExpTerm>-150.0) then
       NewExpTerm := exp(NewExpTerm)
      else
        NewExpTerm := 0.0;
      ExpNumOfKColorings := ExpNumOfKColorings + NewExpTerm;
      NumOfPaths := NumOfPaths + 1;
      MaxClassSize := AArray[1];
      NJArray[Index] := NJArray[Index] - 1;
      NewExpTerm := PrevNewExpTerm;
    end
  else
    begin
      PrevSumA := SumA;
      PrevNewExpTerm := NewExpTerm;
      Num := N - SumA;
      Denom := K - Level + 1;
      Value1 := Num - Denom + 1;
      Value2 := AArray[Level-1];
      if (Value1<=Value2) then
        MaxA := Value1
      else
        MaxA := Value2;
      MinA := (Num div Denom);
      if ((Num mod Denom)>0) then
        MinA := MinA + 1;
      SumA := SumA + MinA;
      Index := MinA;
      while (Index<=MaxA) do
        begin
          AArray[Level] := Index;
          NewExpTerm := NewExpTerm - LnFactArray[Index] +
                        Comb2Array[Index] * LnQ;
          NJArray[Index] := NJArray[Index] + 1;
          TraverseTree (Level+1);
          NJArray[Index] := NJArray[Index] - 1;
          NewExpTerm := PrevNewExpTerm;
          SumA := SumA + 1;
          Index := Index + 1;
        end;
```

```
        SumA := PrevSumA;
      end;
  end;

{*** Calculate  elapsed time ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : double;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Out, 'REPORT.PRN');
  rewrite (Out);
  ObtainUserInfo;
  PrintReportHeadings;
  ReadCombFactFile;
  N := StartN;
  while (N<=EndN) do
    begin
      Initialize1;
      writeln ('Processing Graph Size ',N);
      writeln ('   K = ',K);
      FoundIt := false;
      while (not FoundIt) do
        begin
          Initialize2;
          GetTime (Hour1,Min1,Sec1,Frac1);
          TraverseTree (1);
          GetTime (Hour2,Min2,Sec2,Frac2);
          CalcElapsedTime;
          writeln ('      ExpNumOfKColorings = ',ExpNumOfKColorings);
          writeln ('      NumOfPaths = ',NumOfPaths);
          writeln ('      ElapsedTime = ',ElapsedTime:10:5);
          writeln ('      MaxColorClass = ',MaxClassSize);
          if (ExpNumOfKColorings<1.0) then
            begin
              PrevExpNumOfKColorings := ExpNumOfKColorings;
              PrevNumOfPaths := NumOfPaths;
```

```
            K := K + 1;
            writeln ('  K = ',K);
          end
        else
          begin
            FoundIt := true;
            writeln (Out,N:5,' ':3,K-1:5,' ':3,
                  PrevExpNumOfKColorings,' ':3,PrevNumOfPaths:7,
                  ' ':3,K:5,' ':3,ExpNumOfKColorings,' ':3,
                  NumOfPaths:7);
            K := K - 1; {For increments of 2 in N}
          end;
      end;
    N := N + StepSize;
  end;
 close (Out);
end.
```

15. S_LB2.Pas

```
{$N+,E-}
program S_LB2;
uses DOS,CRT;
var
  FoundIt,Backtrack : boolean;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  SumA,N,StartN,EndN,StepSize,K,I,J : word;
  Num,Denom,Value1,Value2,MaxClassSize : word;
  NumOfPaths,PrevNumOfPaths,Adjust : longint;
  P,Q,LnQ,ElapsedTime,Test,SF,PrevSF : double;
  NewExpTerm,BacktrackValue,ExpNumOfKColorings : double;
  PrevExpNumOfKColorings,LnExpNumOfKColorings : double;
  AArray,MaxClassSizeArray,NJArray : array [0..100] of byte;
  Comb2Array : array [0..2000] of longint;
  LnFactArray : array [0..2000] of double;
  FileIn,Out : text;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  begin
    ClrScr; writeln ('Application: Standard Graph Coloring');
    writeln; writeln ('Program: S_LB2.Pas');
    write ('Enter Edge Density (P): '); readln (P);
    write ('Enter Start Graph Size: '); readln (StartN);
```

```
    write ('Enter End Graph Size: '); readln (EndN);
    write ('Enter Step Size: '); readln (StepSize);
    write ('Enter Initial Value of K: '); readln (K);
  end;

{*** Print report headings. ***}
procedure PrintReportHeadings;
  var
    Count : byte;
  begin
    writeln (Out,'Application:   Standard Graph Coloring');
    writeln (Out,'Program:       S_LB2.PAS -- Implicit Enumeration');
    writeln (Out,'Purpose:       Calculate Lower Bound for Chromatic Number');
    writeln (Out,'Edge Density : ',P:4:2);
    writeln (Out);
    writeln (Out,'                    Exp Num Of        Num Of         ',
                 '                Exp Num Of        Num Of         ');
    writeln (Out,' N       K          K Colorings         Paths     SF  ',
                 '    K          K Colorings         Paths     SF  ');
    writeln (Out,'-----   -----   ----------------------   -------   -----',
                 '   -----   ----------------------   -------   -----');
  end;

{*** Input an external file which contains values for the natural log of ***}
{    the number of combinations of n taken 2 and n! for n=1,2,...,2000      }
procedure ReadCombFactFile;
  var
    J : byte; N,I : longint;
  begin
    assign (FileIn,'COMB_FAC.DAT'); reset (FileIn);
    for I := 1 to 9 do
      readln (FileIn);
    J := 0;
    Comb2Array[0] := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,Comb2Array[N]);
        J := J + 1;
        if (J=8) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
```

```
    readln (FileIn); readln (FileIn);
    LnFactArray[0] := 0;
    J := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,LnFactArray[N]);
        J := J + 1;
        if (J=4) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    close (FileIn);
  end;

{*** Initialize variables ***}
procedure Initialize1;
  var
    I : word;
  begin
    AArray[0] := 255; Q := 1.0 - P;
    LnQ := ln(Q); writeln ('Q = ',Q:10:8);
  end;

{*** Initialize variables ***}
procedure Initialize2;
  var
    I : word;
  begin
    SF := SF - 2.0; Backtrack := false; Adjust := 0;
    for I := 1 to K do
      AArray[I] := 1;
    for I := 1 to 1000 do
      NJArray[I] := 0;
    SumA := 0; ExpNumOfKColorings := 1.0E-300;
    LnExpNumOfKColorings := -87.0; BacktrackValue := -300.0;
    NumOfPaths := 0; NewExpTerm := LnFactArray[N];
  end;

{*** Recursive procedure which calculates the value of E(K) by implicitly ***}
{    generating all sequences of length k which sum to n.                    }
procedure TraverseTree (Level : byte);
  var
```

```
  Index,MinA,MaxA : word; PrevSumA : word; PrevNewExpTerm : double;
begin
  if (Level=K) then
    begin
      PrevNewExpTerm := NewExpTerm;
      AArray[Level] := N - SumA;
      Index := AArray[Level];
      Adjust := Comb2Array[Index];
      NJArray[Index] := NJArray[Index] + 1;
      NewExpTerm := NewExpTerm - LnFactArray[Index] +
                    Comb2Array[Index]*LnQ;
      for I := 1 to AArray[1] do
        if (NJArray[I]>1) then
          NewExpTerm := NewExpTerm - LnFactArray[NJArray[I]];
      if (NewExpTerm>BacktrackValue) then
        begin
          NewExpTerm := exp(NewExpTerm);
          ExpNumOfKColorings := ExpNumOfKColorings + NewExpTerm;
          LnExpNumOfKColorings := ln(ExpNumOfKColorings);
          BacktrackValue := LnExpNumOfKColorings + SF;
          NumOfPaths := NumOfPaths + 1;
          MaxClassSize := AArray[1];
          for I := 1 to (K-1) do
            MaxClassSizeArray[I] := AArray[I];
          Backtrack := false;
        end
      else
        Backtrack := true;
      NJArray[Index] := NJArray[Index] - 1;
      NewExpTerm := PrevNewExpTerm;
    end
  else
    begin
      Adjust := Adjust - Comb2Array[AArray[Level]];
      PrevSumA := SumA;
      PrevNewExpTerm := NewExpTerm;
      Num := N - SumA;
      Denom := K - Level + 1;
      Value1 := Num - Denom + 1;
      Value2 := AArray[Level-1];
      if (Value1<=Value2) then
        MaxA := Value1
      else
        MaxA := Value2;
```

```
          MinA := (Num div Denom);
          if ((Num mod Denom)>0) then
            MinA := MinA + 1;
          SumA := SumA + MinA;
          Index := MinA;
          while (Index<=MaxA) and (not Backtrack) do
            begin
              AArray[Level] := Index;
              NewExpTerm := NewExpTerm - LnFactArray[Index] +
                            Comb2Array[Index]*LnQ;
              NJArray[Index] := NJArray[Index] + 1;
              Test := NewExpTerm + Adjust*LnQ;
              for I := (Level+1) to K do
                Test := Test - LnFactArray[AArray[I]];
              for I := 1 to AArray[1] do
                if (NJArray[I]>1) then
                  Test := Test - LnFactArray[NJArray[I]];
              if (Test>BacktrackValue) then
                TraverseTree (Level+1)
              else
                Backtrack := true;
              NJArray[Index] := NJArray[Index] - 1;
              NewExpTerm := PrevNewExpTerm;
              SumA := SumA + 1;
              if (Index=MaxClassSizeArray[Level]) and Backtrack then
                begin
                  Backtrack := false;
                  MaxClassSizeArray[Level] := 0;
                end;
              Index := Index + 1;
            end;
          AArray[Level] := MinA;
          Adjust := Adjust + Comb2Array[MinA];
          SumA := PrevSumA;
        end;
    end;

{*** Calculate elapsed time ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : double;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
```

```
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Out, 'REPORT.PRN');
  rewrite (Out);
  ObtainUserInfo;
  PrintReportHeadings;
  ReadCombFactFile;
  N := StartN;
  while (N<=EndN) do
    begin
      Initialize1;
      writeln ('Processing Graph Size ',N);
      writeln ('   K = ',K);
      FoundIt := false;
      while (not FoundIt) do
        begin
          SF := -24.0;
          NumOfPaths := 0;
          GetTime (Hour1,Min1,Sec1,Frac1);
          while (NumOfPaths<100000) do
            begin
              Initialize2;
              TraverseTree (1);
              writeln ('LnEMinuN = ',SF:5:1,' NumOfPaths = ',NumOfPaths:7,
                       ' ENOKC = ',ExpNumOfKColorings);
            end;
          GetTime (Hour2,Min2,Sec2,Frac2);
          CalcElapsedTime;
          writeln ('      ExpNumOfKColorings = ',ExpNumOfKColorings);
          writeln ('      SF = ',SF:5:1);
          writeln ('      NumOfPaths = ',NumOfPaths);
          writeln ('      ElapsedTime = ',ElapsedTime:10:5);
          writeln ('      MaxColorClass = ',MaxClassSize);
          if (ExpNumOfKColorings<1.0) then
            begin
              PrevExpNumOfKColorings := ExpNumOfKColorings;
              PrevNumOfPaths := NumOfPaths;
              PrevSF := SF;
              K := K + 1;
              writeln ('   K = ',K);
```

```
            end
          else
            begin
              FoundIt := true;
              writeln (Out,N:5,' ':3,K-1:5,' ':3,
                       PrevExpNumOfKColorings,' ':3,PrevNumOfPaths:7,
                       ' ':3,PrevSF:5:1,' ':3,K:5,' ':3,
                       ExpNumOfKColorings,' ':3,NumOfPaths:7,' ':3,
                       SF:5:1);
              K := K - 1; {For increments of 2 in N or p=0.2}
              {K := K + 1;}  {For increments of 25 in N & p=0.5}
            end;
        end;
      N := N + StepSize;
    end;
  close (Out);
end.
```

16. CTP_LB1.Pas

```
{$N+,E-}
program CTP_LB1;
uses DOS,CRT;
var
  FoundIt : boolean;
  Num,Denom,Value1,Value2,SumA : word;
  BaseN,N,StartN,EndN,StepSize,K,I,MaxClassSize : word;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  Diff,Factor,TAdjust,PrevA : integer;
  NumOfPaths,PrevNumOfPaths,Adjust,T : longint;
  ExpNumOfKColorings,PrevExpNumOfKColorings : double;
  LnExpNumOfKColorings,FinalAdj,P : double;
  NewExpTerm,InitAdj,OldExpNumOfKColorings,ElapsedTime : double;
  AArray : array [0..200] of word;
  NJArray : array [1..2000] of word;
  AdjArray,SArray : array [1..6] of longint;
  Comb2Array : array [0..2000] of longint;
  QArray,LnQArray : array [1..6] of double;
  LnFactArray : array [0..2000] of double;
  FileIn,Out : text;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  begin
```

```
    ClrScr;
    writeln ('Application: Composite Graph Coloring');
    writeln ('Program: CTP_LB1.Pas -- Explicit Enumeration');
    writeln ('Purpose: Calculate Minimum Lower Bound for Chromatic Number');
    write ('Enter Edge Density (P): '); readln (P);
    write ('Enter Graph Start Size: '); readln (StartN);
    write ('Enter Graph End Size: '); readln (EndN);
    write ('Enter Step Size: '); readln (StepSize);
    write ('Enter Initial Value of K: '); readln (K);
  end;

{*** Print report headings. ***}
procedure PrintReportHeadings;
  var
    Count : byte;
  begin
    writeln (Out,'Application:   Composite Graph Coloring');
    writeln (Out,'Program:       CTP_LB1.Pas -- Explicit Enumeration');
    writeln (Out,'Purpose:       Calculate Minimum Lower Bound for Chromatic ',
          'Number');
    writeln (Out,'Chrom Distrib: Truncated Poisson');
    writeln (Out,'Edge Density : ',P:4:2);
    writeln (Out);
    writeln (Out,'                    Exp Num Of        Num Of ',
                 '                 Exp Num Of        Num Of ');
    writeln (Out,'  N       K          K Colorings         Paths ',
                 '     K          K Colorings         Paths ');
    writeln (Out,'-----   -----   ----------------------   -------',
                 '   -----   ----------------------   -------');
  end;

{*** Input an external file which contains values for the natural log of ***}
{    the number of combinations of n taken 2 and n! for n=1,2,...,2000      }
procedure ReadCombFactFile;
  var
    J : byte; N,I : longint;
  begin
    assign (FileIn,'COMB_FAC.DAT'); reset (FileIn);
    for I := 1 to 9 do
      readln (FileIn);
    Comb2Array[0] := 0;
    J := 0;
    for N := 1 to 2000 do
      begin
```

```
        read (FileIn,Comb2Array[N]);
        J := J + 1;
        if (J=8) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    readln (FileIn); readln (FileIn);
    LnFactArray[0] := 0;
    J := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,LnFactArray[N]);
        J := J + 1;
        if (J=4) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    close (FileIn);
  end;

{*** Initialize variables ***}
procedure Initialize1;
  var
    I : word;
  begin
    N := round(1.582*BaseN);
    AArray[0] := 65000;
    QArray[1] := 1.0 - P*(1.582*BaseN-2.0+1.0/P)/(1.582*BaseN-1.0);
    QArray[2] := 1.0 - 0.368/(1.582*BaseN-1.0);
    QArray[3] := QArray[2];
    QArray[4] := 1.0 - 0.182/(1.582*BaseN-1.0);
    QArray[5] := 1.0 - 0.063/(1.582*BaseN-1.0);
    QArray[6] := 1.0 - 0.019/(1.582*BaseN-1.0);
    for I := 1 to 6 do
      LnQArray[I] := ln(QArray[I]);
    InitAdj := 2.0 + round(0.291*BaseN)*LnFactArray[2] +
               round(0.097*BaseN)*LnFactArray[3] +
               round(0.024*BaseN)*LnFactArray[4] +
               round(0.005*BaseN)*LnFactArray[5] +
               round(0.001*BaseN)*LnFactArray[6];
```

```
    for I := 1 to 1000 do
      NJArray[I] := 0;
  end;

{*** Initialize variables ***}
procedure Initialize2;
  var
    I : word;
  begin
    MaxClassSize := 0; SumA := 0; Adjust := 0;
    for I := 1 to K do
      AArray[I] := 1;
    T := (K-1)*K div 2;
    for I := 1 to 5 do
      SArray[I] := K-I;
    ExpNumOfKColorings := 1.0E-300; NumOfPaths := 0;
    NewExpTerm := LnFactArray[round(1.582*BaseN)] + LnFactArray[K] -
                  InitAdj;
  end;

{*** Update values of T and S ***}
procedure UpdateTandS (Level : byte);
  begin
    Diff := AArray[Level]-PrevA;
    TAdjust := 0;
    for I := 1 to (Level-1) do
      TAdjust := TAdjust + AArray[I];
    for I := Level+1 to K do
      TAdjust := TAdjust + AArray[I];
    T := T + TAdjust*Diff;
    if (Level>=K-4) or (Level=1) then
      begin
        SArray[1] := N - AArray[K];
        SArray[2] := SArray[1] - AArray[K-1];
        SArray[3] := SArray[2] - AArray[K-2];
        SArray[4] := SArray[3] - AArray[K-3];
        SArray[5] := SArray[4] - AArray[K-4];
        for I := 1 to 5 do
          SArray[I] := AArray[1]*SArray[I];
      end;
  end;

{*** Calculate value of the final adjustment to ExpTerm ***}
procedure CalcFinalAdj;
```

```
  var
    I,J : byte;
  begin
    AdjArray[2] := T - SArray[1];
    for I := 3 to 6 do
      AdjArray[I] := AdjArray[I-1] - SArray[I-1];
    FinalAdj := 0.0;
    for I := 2 to 6 do
      FinalAdj := FinalAdj + AdjArray[I]*LnQArray[I];
  end;

{*** Recursive procedure which calculates the value of E(K) by explicitly ***}
{    generating all sequences of length k which sum to n.                    }
procedure TraverseTree (Level : byte);
  var
    Index,MaxA,MinA : byte;
    PrevSumA : word;
    PrevNewExpTerm : double;
  begin
    if (Level=K) then
      begin
        PrevNewExpTerm := NewExpTerm;
        PrevA := AArray[K];
        AArray[K] := N - SumA;
        Index := AArray[K];
        Adjust := Comb2Array[Index];
        NJArray[Index] := NJArray[Index] + 1;
        UpdateTandS (K);
        CalcFinalAdj;
        NewExpTerm := NewExpTerm - LnFactArray[Index] +
                      Comb2Array[Index]*LnQArray[1] + FinalAdj;
        for I := 1 to AArray[1] do
          if (NJArray[I]>1) then
            NewExpTerm := NewExpTerm - 2*LnFactArray[NJArray[I]];
        NewExpTerm := exp(NewExpTerm);
        ExpNumOfKColorings := ExpNumOfKColorings + NewExpTerm;
        NumOfPaths := NumOfPaths + 1;
        MaxClassSize := AArray[1];
        NJArray[Index] := NJArray[Index] - 1;
        NewExpTerm := PrevNewExpTerm;
      end
    else
      begin
        Adjust := Adjust - Comb2Array[AArray[Level]];
```

```
        PrevSumA := SumA;
        PrevNewExpTerm := NewExpTerm;
        Num := N - SumA;
        Denom := K - Level + 1;
        Value1 := Num - Denom + 1;
        Value2 := AArray[Level-1];
        if (Value1<=Value2) then
          MaxA := Value1
        else
          MaxA := Value2;
        MinA := (Num div Denom);
        if ((Num mod Denom)>0) then
          MinA := MinA + 1;
        SumA := SumA + MinA;
        Index := MinA;
        PrevA := AArray[Level];
        while (Index<=MaxA) do
          begin
            AArray[Level] := Index;
            NewExpTerm := NewExpTerm - LnFactArray[Index] +
                          Comb2Array[Index]*LnQArray[1];
            NJArray[Index] := NJArray[Index] + 1;
            UpdateTandS (Level);
            TraverseTree (Level+1);
            NJArray[Index] := NJArray[Index] - 1;
            NewExpTerm := PrevNewExpTerm;
            SumA := SumA + 1;
            PrevA := Index;
            Index := Index + 1;
          end;
        AArray[Level] := MinA;
        UpdateTandS (Level);
        Adjust := Adjust + Comb2Array[MinA];
        SumA := PrevSumA;
      end;
  end;

{*** Calculate the elapsed processing time ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : double;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
```

```
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Out, 'REPORT.PRN'); rewrite (Out);
  ObtainUserInfo;
  PrintReportHeadings;
  ReadCombFactFile;
  BaseN := StartN;
  while (BaseN<=EndN) do
    begin
      Initialize1;
      writeln ('Processing Graph Size ',BaseN);
      writeln ('   K = ',K);
      FoundIt := false;
      while (not FoundIt) do
        begin
          Initialize2;
          GetTime (Hour1,Min1,Sec1,Frac1);
          TraverseTree (1);
          GetTime (Hour2,Min2,Sec2,Frac2);
          CalcElapsedTime;
          writeln ('      ExpNumOfKColorings = ',ExpNumOfKColorings);
          writeln ('      NumOfPaths = ',NumOfPaths);
          writeln ('      ElapsedTime = ',ElapsedTime:10:5);
          writeln ('      MaxColorClass = ',MaxClassSize);
          if (ExpNumOfKColorings<1.0) then
            begin
              PrevExpNumOfKColorings := ExpNumOfKColorings;
              PrevNumOfPaths := NumOfPaths;
              K := K + 1;
              writeln ('   K = ',K);
            end
          else
            begin
              FoundIt := true;
              writeln (Out,BaseN:5,' ':3,K-1:5,' ':3,PrevExpNumOfKColorings,
                       ' ':3,PrevNumOfPaths:7,' ':3,K:5,' ':3,ExpNumOfKColorings,
                       ' ':3,NumOfPaths:7);
              K := K - 1;   {Use with Step = 2}
            end;
        end;
```

```
      BaseN := BaseN + StepSize;
    end;
  close (Out);
end.
```

17. CTP_LB2.Pas

```
{$N+,E-}
program CTP_LB2;
uses DOS,CRT;
var
  FoundIt : boolean;
  Num,Denom,Value1,Value2,SumA : word;
  BaseN,N,StartN,EndN,StepSize,K,I,MaxClassSize : word;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  Diff,Factor,TAdjust,PrevA : integer;
  NumOfPaths,PrevNumOfPaths,Adjust,T : longint;
  ExpNumOfKColorings,PrevExpNumOfKColorings : double;
  LnExpNumOfKColorings,FinalAdj,P : double;
  NewExpTerm,InitAdj,OldExpNumOfKColorings,ElapsedTime : double;
  AArray : array [0..200] of word;
  NJArray : array [1..2000] of word;
  AdjArray,SArray : array [1..6] of longint;
  Comb2Array : array [0..2000] of longint;
  QArray,LnQArray : array [1..6] of double;
  LnFactArray : array [0..2000] of double;
  FileIn,Out : text;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  begin
    ClrScr;
    writeln ('Application: Composite Graph Coloring');
    writeln ('Program: CTP_LB2.Pas -- Explicit Enumeration');
    writeln ('Purpose: Calculate Maximum Lower Bound for Chromatic Number');
    write ('Enter Edge Density (P): '); readln (P);
    write ('Enter Graph Start Size: '); readln (StartN);
    write ('Enter Graph End Size: '); readln (EndN);
    write ('Enter Step Size: '); readln (StepSize);
    write ('Enter Initial Value of K: '); readln (K);
  end;

{*** Print report headings. ***}
```

```
procedure PrintReportHeadings;
  var
    Count : byte;
  begin
    writeln (Out,'Application:   Composite Graph Coloring');
    writeln (Out,'Program:       CTP_LB2.Pas -- Explicit Enumeration');
    writeln (Out,'Purpose:       Calculate Maximum Lower Bound for Chromatic ',
          'Number');
    writeln (Out,'Chrom Distrib: Truncated Poisson');
    writeln (Out,'Edge Density : ',P:4:2);
    writeln (Out);
    writeln (Out,'                        Exp Num Of        Num Of ',
                 '                    Exp Num Of        Num Of ');
    writeln (Out,' N       K          K Colorings         Paths ',
                 '    K          K Colorings         Paths ');
    writeln (Out,'-----   -----   ----------------------   -------',
                 '   -----   ----------------------   -------');
  end;

{*** Input an external file which contains values for the natural log of ***}
{    the number of combinations of n taken 2 and n! for n=1,2,...,2000      }
procedure ReadCombFactFile;
  var
    J : byte;
    N,I : longint;
  begin
    assign (FileIn,'COMB_FAC.DAT'); reset (FileIn);
    for I := 1 to 9 do
      readln (FileIn);
    Comb2Array[0] := 0;
    J := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,Comb2Array[N]);
        J := J + 1;
        if (J=8) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    readln (FileIn); readln (FileIn);
    LnFactArray[0] := 0;
    J := 0;
```

```
    for N := 1 to 2000 do
      begin
        read (FileIn,LnFactArray[N]);
        J := J + 1;
        if (J=4) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    close (FileIn);
  end;

{*** Initialize variables ***}
procedure Initialize1;
  var
    I : word;
  begin
    N := round(1.582*BaseN);
    AArray[0] := 65000;
    QArray[1] := 1.0 - P*(1.582*BaseN-2.0+1.0/P)/(1.582*BaseN-1.0);
    QArray[2] := 1.0 - 0.368/(1.582*BaseN-1.0);
    QArray[3] := QArray[2];
    QArray[4] := 1.0 - 0.182/(1.582*BaseN-1.0);
    QArray[5] := 1.0 - 0.063/(1.582*BaseN-1.0);
    QArray[6] := 1.0 - 0.019/(1.582*BaseN-1.0);
    for I := 1 to 6 do
      LnQArray[I] := ln(QArray[I]);
    InitAdj := 2.0 + round(0.291*BaseN)*LnFactArray[2] +
               round(0.097*BaseN)*LnFactArray[3] +
               round(0.024*BaseN)*LnFactArray[4] +
               round(0.005*BaseN)*LnFactArray[5] +
               round(0.001*BaseN)*LnFactArray[6];
    for I := 1 to 1000 do
      NJArray[I] := 0;
  end;

{*** Initialize variables ***}
procedure Initialize2;
  var
    I : word;
  begin
    MaxClassSize := 0; SumA := 0; Adjust := 0;
    for I := 1 to K do
```

```
      AArray[I] := 1;
    T := (K-1)*K div 2;
    for I := 1 to 5 do
      SArray[I] := K-I;
    ExpNumOfKColorings := 1.0E-300; NumOfPaths := 0;
    NewExpTerm := LnFactArray[round(1.582*BaseN)] + LnFactArray[K] -
                  InitAdj;
  end;

{*** Update values of T and S ***}
procedure UpdateTandS (Level : byte);
  begin
    Diff := AArray[Level]-PrevA;
    TAdjust := 0;
    for I := 1 to (Level-1) do
      TAdjust := TAdjust + AArray[I];
    for I := Level+1 to K do
      TAdjust := TAdjust + AArray[I];
    T := T + TAdjust*Diff;
    if (Level<=5) or (Level=K) then
      begin
        SArray[1] := N - AArray[1];
        SArray[2] := SArray[1] - AArray[2];
        SArray[3] := SArray[2] - AArray[3];
        SArray[4] := SArray[3] - AArray[4];
        SArray[5] := SArray[4] - AArray[5];
        for I := 1 to 5 do
          SArray[I] := AArray[K]*SArray[I];
      end;
  end;

{*** Calculate value of the final adjustment to ExpTerm ***}
procedure CalcFinalAdj;
  var
    I,J : byte;
  begin
    AdjArray[2] := T - SArray[1];
    for I := 3 to 6 do
      AdjArray[I] := AdjArray[I-1] - SArray[I-1];
    FinalAdj := 0.0;
    for I := 2 to 6 do
      FinalAdj := FinalAdj + AdjArray[I]*LnQArray[I];
  end;
```

```
{*** Recursive procedure which calculates the value of E(K) by explicitly ***}
{    generating all sequences of length k which sum to n.                    }
procedure TraverseTree (Level : byte);
  var
    Index,MaxA,MinA : byte;
    PrevSumA : word;
    PrevNewExpTerm : double;
  begin
    if (Level=K) then
      begin
        PrevNewExpTerm := NewExpTerm;
        PrevA := AArray[K];
        AArray[K] := N - SumA;
        Index := AArray[K];
        Adjust := Comb2Array[Index];
        NJArray[Index] := NJArray[Index] + 1;
        UpdateTandS (K);
        CalcFinalAdj;
        NewExpTerm := NewExpTerm - LnFactArray[Index] +
                      Comb2Array[Index]*LnQArray[1] + FinalAdj;
        for I := 1 to AArray[1] do
          if (NJArray[I]>1) then
            NewExpTerm := NewExpTerm - 2*LnFactArray[NJArray[I]];
        NewExpTerm := exp(NewExpTerm);
        ExpNumOfKColorings := ExpNumOfKColorings + NewExpTerm;
        NumOfPaths := NumOfPaths + 1;
        MaxClassSize := AArray[1];
        NJArray[Index] := NJArray[Index] - 1;
        NewExpTerm := PrevNewExpTerm;
      end
    else
      begin
        Adjust := Adjust - Comb2Array[AArray[Level]];
        PrevSumA := SumA;
        PrevNewExpTerm := NewExpTerm;
        Num := N - SumA;
        Denom := K - Level + 1;
        Value1 := Num - Denom + 1;
        Value2 := AArray[Level-1];
        if (Value1<=Value2) then
          MaxA := Value1
        else
          MaxA := Value2;
        MinA := (Num div Denom);
```

```
          if ((Num mod Denom)>0) then
            MinA := MinA + 1;
          SumA := SumA + MinA;
          Index := MinA;
          PrevA := AArray[Level];
          while (Index<=MaxA) do
            begin
              AArray[Level] := Index;
              NewExpTerm := NewExpTerm - LnFactArray[Index] +
                        Comb2Array[Index]*LnQArray[1];
              NJArray[Index] := NJArray[Index] + 1;
              UpdateTandS (Level);
              TraverseTree (Level+1);
              NJArray[Index] := NJArray[Index] - 1;
              NewExpTerm := PrevNewExpTerm;
              SumA := SumA + 1;
              PrevA := Index;
              Index := Index + 1;
            end;
          AArray[Level] := MinA;
          UpdateTandS (Level);
          Adjust := Adjust + Comb2Array[MinA];
          SumA := PrevSumA;
        end;
    end;

{*** Calculate the elapsed processing time ***}
procedure CalcElapsedTime;
    var
      BegTime,EndTime : double;
    begin
      BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
      EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
      ElapsedTime := EndTime - BegTime;
      if (ElapsedTime<0.0) then
        ElapsedTime := ElapsedTime + 86400.0;
    end;

begin
    assign (Out, 'REPORT.PRN'); rewrite (Out);
    ObtainUserInfo;
    PrintReportHeadings;
    ReadCombFactFile;
    BaseN := StartN;
```

```
  while (BaseN<=EndN) do
    begin
      Initialize1;
      writeln ('Processing Graph Size ',BaseN);
      writeln ('   K = ',K);
      FoundIt := false;
      while (not FoundIt) do
        begin
          Initialize2;
          GetTime (Hour1,Min1,Sec1,Frac1);
          TraverseTree (1);
          GetTime (Hour2,Min2,Sec2,Frac2);
          CalcElapsedTime;
          writeln ('      ExpNumOfKColorings = ',ExpNumOfKColorings);
          writeln ('      NumOfPaths = ',NumOfPaths);
          writeln ('      ElapsedTime = ',ElapsedTime:10:5);
          writeln ('      MaxColorClass = ',MaxClassSize);
          if (ExpNumOfKColorings<1.0) then
            begin
              PrevExpNumOfKColorings := ExpNumOfKColorings;
              PrevNumOfPaths := NumOfPaths;
              K := K + 1;
              writeln ('   K = ',K);
            end
          else
            begin
              FoundIt := true;
              writeln(Out,BaseN:5,'':3,K-1:5,'':3,PrevExpNumOfKColorings,'':3,
                      PrevNumOfPaths:7,' ':3,K:5,' ':3,ExpNumOfKColorings,' ':3,
                      NumOfPaths:7);
              K := K - 1;   {Use with Step = 2}
            end;
        end;
      BaseN := BaseN + StepSize;
    end;
  close (Out);
end.
```

18. CTP_LB3.Pas

```
{$N+,E-}
program CTP_LB3;
uses DOS,CRT;
```

```
var
  FoundIt,Backtrack,Quit : boolean;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  N,BaseN,StartN,EndN,SumA,Num,Denom,Value1,Value2 : word;
  StepSize,K,I,MaxClassSize : word;
  Diff,Factor,TAdjust,PrevA : integer;
  NumOfPaths,PrevNumOfPaths,Adjust,T : longint;
  P,ElapsedTime,Test,LnEMinusN,PrevLnEMinusN,LnEMinus,FinalAdj : double;
  ExpNumOfKColorings,PrevExpNumOfKColorings : double;
  LnExpNumOfKColorings,OldExpNumOfKColorings : double;
  NewExpTerm,InitAdj,BacktrackValue : double;
  QArray,LnQArray : array [1..6] of double;
  AdjArray,SArray : array [1..6] of longint;
  AArray,MaxClassSizeArray,NJArray : array [0..200] of word;
  Comb2Array : array [0..2000] of longint;
  LnFactArray : array [0..2000] of double;
  FileIn,Out : text;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  begin
    ClrScr;
    writeln ('Application: Composite Graph Coloring');
    writeln ('Program: CTP_LB3.PAS -- Implicit Enumeration');
    writeln ('Purpose: Calculate Minimum Lower Bound for Chromatic Number');
    write ('Enter Edge Density (P): '); readln (P);
    write ('Enter Graph Start Size: '); readln (StartN);
    write ('Enter Graph End Size: '); readln (EndN);
    write ('Enter Step Size: '); readln (StepSize);
    write ('Enter Initial Value of K: '); readln (K);
  end;

{*** Print report headings. ***}
procedure PrintReportHeadings;
  var
    Count : byte;
  begin
    writeln (Out,'Application:   Composite Graph Coloring');
    writeln (Out,'Program:       CTP_LB3.PAS -- Implicit Enumeration');
    writeln (Out,'Purpose:      Calculate Minimum Lower Bound for Chromatic ',
          'Number');
    writeln (Out,'Chrom Distrib: Truncated Poisson');
    writeln (Out,'Edge Density : ',P:4:2);
    writeln (Out);
```

```
    writeln (Out,'                    Exp Num Of        Num Of        ',
                 '                 Exp Num Of        Num Of        ');
    writeln (Out,' N       K          K Colorings          Paths     SF  ',
                 '    K          K Colorings          Paths     SF  ');
    writeln (Out,'-----  -----  ----------------------  -------  -----',
                 '  -----  ----------------------  -------  -----');
  end;

{*** Input an external file which contains values for the natural log of ***}
{    the number of combinations of n taken 2 and n! for n=1,2,...,2000      }
procedure ReadCombFactFile;
  var
    J : byte;
    N,I : longint;
  begin
    assign (FileIn,'COMB_FAC.DAT'); reset (FileIn);
    for I := 1 to 9 do
      readln (FileIn);
    Comb2Array[0] := 0;
    J := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,Comb2Array[N]);
        J := J + 1;
        if (J=8) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    readln (FileIn); readln (FileIn);
    LnFactArray[0] := 0;
    J := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,LnFactArray[N]);
        J := J + 1;
        if (J=4) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    close (FileIn);
```

```
  end;

{*** Initialize variables ***}
procedure Initialize1;
  var
    I : word;
  begin
    N := round(1.582*BaseN);
    AArray[0] := 65000;
    QArray[1] := 1.0 - P*(1.582*BaseN-2.0+1.0/P)/(1.582*BaseN-1.0);
    QArray[2] := 1.0 - 0.368/(1.582*BaseN-1.0);
    QArray[3] := QArray[2];
    QArray[4] := 1.0 - 0.182/(1.582*BaseN-1.0);
    QArray[5] := 1.0 - 0.063/(1.582*BaseN-1.0);
    QArray[6] := 1.0 - 0.019/(1.582*BaseN-1.0);
    for I := 1 to 6 do
      LnQArray[I] := ln(QArray[I]);
    InitAdj := ln(2.0) + round(0.291*BaseN)*LnFactArray[2] +
               round(0.097*BaseN)*LnFactArray[3] +
               round(0.024*BaseN)*LnFactArray[4] +
               round(0.005*BaseN)*LnFactArray[5] +
               round(0.001*BaseN)*LnFactArray[6];
    for I := 1 to 1000 do
      NJArray[I] := 0;
  end;

{*** Initialize variables ***}
procedure Initialize2;
  var
    I : byte;
  begin
    if (NumOfPaths<400000) then
      LnEMinusN := LnEMinusN - 4.0
    else
      LnEMinusN := LnEMinusN - 2.0;
    MaxClassSize := 0; Backtrack := false;
    SumA := 0; Adjust := 0;
    for I := 1 to K do
      AArray[I] := 1;
    T := (K-1)*K div 2;
    for I := 1 to 5 do
      SArray[I] := K-I;
    ExpNumOfKColorings := 1.0E-300; LnExpNumOfKColorings := -87.49823353;
    BacktrackValue := -350.0; NumOfPaths := 0;
```

```
      NewExpTerm := LnFactArray[round(1.582*BaseN)] + LnFactArray[K] -
                    InitAdj;
   end;

{*** Update the values of T and S ***}
procedure UpdateTandS (Level : byte);
   begin
      Diff := AArray[Level]-PrevA;
      TAdjust := 0;
      for I := 1 to (Level-1) do
         TAdjust := TAdjust + AArray[I];
      for I := Level+1 to K do
         TAdjust := TAdjust + AArray[I];
      T := T + TAdjust*Diff;
      if (Level>=K-4) or (Level=1) then
         begin
            SArray[1] := N - AArray[K];
            SArray[2] := SArray[1] - AArray[K-1];
            SArray[3] := SArray[2] - AArray[K-2];
            SArray[4] := SArray[3] - AArray[K-3];
            SArray[5] := SArray[4] - AArray[K-4];
            for I := 1 to 5 do
               SArray[I] := AArray[1]*SArray[I];
         end;
   end;

{*** Calculate the final adjustment for ExpTerm ***}
procedure CalcFinalAdj;
   var
      I,J : byte;
   begin
      AdjArray[2] := T - SArray[1];
      for I := 3 to 6 do
         AdjArray[I] := AdjArray[I-1] - SArray[I-1];
      FinalAdj := 0.0;
      for I := 2 to 6 do
         FinalAdj := FinalAdj + AdjArray[I]*LnQArray[I];
   end;

{*** Recursive procedure which calculates the value of E(K) by implicitly ***}
{    generating all sequences of length k which sum to n.                    }
procedure TraverseTree (Level : byte);
   var
      Index,MaxA,MinA : word;
```

```
  PrevSumA : word;
  PrevNewExpTerm : double;
begin
  if (Level=K) then
    begin
      PrevNewExpTerm := NewExpTerm;
      PrevA := AArray[K];
      AArray[K] := N - SumA;
      Index := AArray[K];
      Adjust := Comb2Array[Index];
      NJArray[Index] := NJArray[Index] + 1;
      UpdateTandS (K);
      CalcFinalAdj;
      NewExpTerm := NewExpTerm - LnFactArray[Index] +
                    Comb2Array[Index]*LnQArray[1] + FinalAdj;
      for I := 1 to AArray[1] do
        if (NJArray[I]>1) then
          NewExpTerm := NewExpTerm - 2*LnFactArray[NJArray[I]];
      if (NewExpTerm>BacktrackValue) then
        begin
          NewExpTerm := exp(NewExpTerm);
          ExpNumOfKColorings := ExpNumOfKColorings + NewExpTerm;
          LnExpNumOfKColorings := ln(ExpNumOfKColorings);
          BacktrackValue := LnExpNumOfKColorings + LnEMinusN;
          NumOfPaths := NumOfPaths + 1;
          if (NumOfPaths>=750000) then
            Quit := true;
          MaxClassSize := AArray[1];
          Backtrack := false;
          for I := 1 to (K-1) do
            MaxClassSizeArray[I] := AArray[I];
        end
      else
        Backtrack := true;
      NJArray[Index] := NJArray[Index] - 1;
      NewExpTerm := PrevNewExpTerm;
    end
  else
    begin
      Adjust := Adjust - Comb2Array[AArray[Level]];
      PrevSumA := SumA;
      PrevNewExpTerm := NewExpTerm;
      Num := N - SumA;
      Denom := K - Level + 1;
```

```
Value1 := Num - Denom + 1;
Value2 := AArray[Level-1];
if (Value1 <= Value2) then
  MaxA := Value1
else
  MaxA := Value2;
MinA := (Num div Denom);
if ((Num mod Denom)>0) then
  MinA := MinA + 1;
SumA := SumA + MinA;
Index := MinA;
PrevA := AArray[Level];
while (Index <= MaxA) and (not Backtrack) and (not Quit) do
  begin
    AArray[Level] := Index;
    NewExpTerm := NewExpTerm - LnFactArray[Index] +
                  Comb2Array[Index]*LnQArray[1];
    NJArray[Index] := NJArray[Index] + 1;
    UpdateTandS (Level);
    CalcFinalAdj;
    Test := NewExpTerm + FinalAdj + Adjust*LnQArray[1];
    for I := (Level+1) to K do
      Test := Test - LnFactArray[AArray[I]];
    for I := 1 to AArray[1] do
      if (NJArray[I]>1) then
        Test := Test - 2*LnFactArray[NJArray[I]];
    if (Test>BacktrackValue) then
      TraverseTree (Level+1)
    else
      Backtrack := true;
    NJArray[Index] := NJArray[Index] - 1;
    NewExpTerm := PrevNewExpTerm;
    SumA := SumA + 1;
    PrevA := Index;
    if (Index=MaxClassSizeArray[Level]) and Backtrack then
      begin
        Backtrack := false;
        MaxClassSizeArray[Level] := 0;
      end;
    Index := Index + 1;
  end;
PrevA := AArray[Level];
AArray[Level] := MinA;
UpdateTandS (Level);
```

```
          Adjust := Adjust + Comb2Array[MinA];
          SumA := PrevSumA;
        end;
    end;

{*** Calculate the elapsed processing time ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : double;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Out, 'REPORT.PRN'); rewrite (Out);
  ObtainUserInfo;
  PrintReportHeadings;
  ReadCombFactFile;
  BaseN := StartN;
  while (BaseN<=EndN) do
    begin
      Initialize1;
      writeln ('Processing Graph Size ',BaseN);
      writeln ('   K = ',K);
      FoundIt := false;
      while (not FoundIt) do
        begin
          LnEMinusN := -24;
          NumOfPaths := 0;
          Quit := false;
          GetTime (Hour1,Min1,Sec1,Frac1);
          while (NumOfPaths<500000) and (not Quit) do
            begin
              Initialize2;
              TraverseTree (1);
              writeln ('LnEMinusN = ',LnEMinusN:5:1,' NumOfPaths = ',
                       NumOfPaths:7,' ENOKC = ',ExpNumOfKColorings);
            end;
          GetTime (Hour2,Min2,Sec2,Frac2);
          CalcElapsedTime;
```

```
            writeln ('     ExpNumOfKColorings = ',ExpNumOfKColorings);
            writeln ('     NumOfPaths = ',NumOfPaths);
            writeln ('     ElapsedTime = ',ElapsedTime:10:5);
            writeln ('     MaxColorClass = ',MaxClassSize);
            if (ExpNumOfKColorings<1.0) then
              begin
                PrevExpNumOfKColorings := ExpNumOfKColorings;
                PrevNumOfPaths := NumOfPaths;
                PrevLnEMinusN := LnEMinusN;
                K := K + 1;
                writeln ('  K = ',K);
              end
            else
              begin
                FoundIt := true;
                writeln (Out,BaseN:5,' ':3,K-1:5,' ':3,
                        PrevExpNumOfKColorings,' ':3,PrevNumOfPaths:7,
                        ' ':3,PrevLnEMinusN:5:1,' ':3,K:5,' ':3,
                        ExpNumOfKColorings,' ':3,NumOfPaths:7,' ':3,
                        LnEMinusN:5:1);
                K := K - 1;   {Use with Step = 2}
                {K := K + 2;}   {Use with Step = 25}
              end;
          end;
        BaseN := BaseN + StepSize;
      end;
  close (Out);
end.
```

19. CTP_LB4.Pas

```
{$N+,E-}
program CTP_LB4;
uses DOS,CRT;
var
  FoundIt,Backtrack,Quit : boolean;
  Hour1,Min1,Sec1,Frac1,Hour2,Min2,Sec2,Frac2 : word;
  N,BaseN,StartN,EndN,SumA,Num,Denom,Value1,Value2 : word;
  StepSize,K,I,MaxClassSize : word;
  Diff,Factor,TAdjust,PrevA : integer;
  NumOfPaths,PrevNumOfPaths,Adjust,T : longint;
  P,ElapsedTime,Test,LnEMinusN,PrevLnEMinusN,LnEMinus,FinalAdj : double;
  ExpNumOfKColorings,PrevExpNumOfKColorings : double;
  LnExpNumOfKColorings,OldExpNumOfKColorings : double;
```

```
  NewExpTerm,InitAdj,BacktrackValue : double;
  QArray,LnQArray : array [1..6] of double;
  AdjArray,SArray : array [1..6] of longint;
  AArray,MaxClassSizeArray,NJArray : array [0..200] of word;
  Comb2Array : array [0..2000] of longint;
  LnFactArray : array [0..2000] of double;
  FileIn,Out : text;

{*** Obtain user information wrt desired procesing. ***}
procedure ObtainUserInfo;
  begin
    ClrScr;
    writeln ('Application: Composite Graph Coloring');
    writeln ('Program: CTP_LB4.PAS -- Implicit Enumeration');
    writeln ('Purpose: Calculate Maximum Lower Bound for Chromatic Number');
    write ('Enter Edge Density (P): '); readln (P);
    write ('Enter Graph Start Size: '); readln (StartN);
    write ('Enter Graph End Size: '); readln (EndN);
    write ('Enter Step Size: '); readln (StepSize);
    write ('Enter Initial Value of K: '); readln (K);
  end;

{*** Print report headings. ***}
procedure PrintReportHeadings;
  var
    Count : byte;
  begin
    writeln (Out,'Application:   Composite Graph Coloring');
    writeln (Out,'Program:       CTP_LB4.PAS -- Implicit Enumeration');
    writeln (Out,'Purpose:       Calculate Maximum Lower Bound for Chromatic ',
          'Number');
    writeln (Out,'Chrom Distrib: Truncated Poisson');
    writeln (Out,'Edge Density : ',P:4:2);
    writeln (Out);
    writeln (Out,'                    Exp Num Of        Num Of        ',
                 '                 Exp Num Of        Num Of        ');
    writeln (Out,'  N       K          K Colorings         Paths     SF  ',
                 '    K          K Colorings         Paths     SF  ');
    writeln (Out,'----   -----   ---------------------   -------   -----',
                 '   -----   ---------------------   -------   -----');
  end;

{*** Input an external file which contains values for the natural log of ***}
{    the number of combinations of n taken 2 and n! for n=1,2,...,2000      }
```

```
procedure ReadCombFactFile;
  var
    J : byte;
    N,I : longint;
  begin
    assign (FileIn,'COMB_FAC.DAT'); reset (FileIn);
    for I := 1 to 9 do
      readln (FileIn);
    Comb2Array[0] := 0;
    J := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,Comb2Array[N]);
        J := J + 1;
        if (J=8) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    readln (FileIn); readln (FileIn);
    LnFactArray[0] := 0;
    J := 0;
    for N := 1 to 2000 do
      begin
        read (FileIn,LnFactArray[N]);
        J := J + 1;
        if (J=4) then
          begin
            readln (FileIn);
            J := 0;
          end;
      end;
    close (FileIn);
  end;

{*** Initialize variables ***}
procedure Initialize1;
  var
    I : word;
  begin
    N := round(1.582*BaseN);
    AArray[0] := 65000;
    QArray[1] := 1.0 - P*(1.582*BaseN-2.0+1.0/P)/(1.582*BaseN-1.0);
```

```
    QArray[2] := 1.0 - 0.368/(1.582*BaseN-1.0);
    QArray[3] := QArray[2];
    QArray[4] := 1.0 - 0.182/(1.582*BaseN-1.0);
    QArray[5] := 1.0 - 0.063/(1.582*BaseN-1.0);
    QArray[6] := 1.0 - 0.019/(1.582*BaseN-1.0);
    for I := 1 to 6 do
      LnQArray[I] := ln(QArray[I]);
    InitAdj := ln(2.0) + round(0.291*BaseN)*LnFactArray[2] +
               round(0.097*BaseN)*LnFactArray[3] +
               round(0.024*BaseN)*LnFactArray[4] +
               round(0.005*BaseN)*LnFactArray[5] +
               round(0.001*BaseN)*LnFactArray[6];
    for I := 1 to 1000 do
      NJArray[I] := 0;
  end;

{*** Initialize variables ***}
procedure Initialize2;
  var
    I : byte;
  begin
    if (NumOfPaths<400000) then
      LnEMinusN := LnEMinusN - 4.0
    else
      LnEMinusN := LnEMinusN - 2.0;
    MaxClassSize := 0; Backtrack := false;
    SumA := 0; Adjust := 0;
    for I := 1 to K do
      AArray[I] := 1;
    T := (K-1)*K div 2;
    for I := 1 to 5 do
      SArray[I] := K-I;
    ExpNumOfKColorings := 1.0E-300; LnExpNumOfKColorings := -87.49823353;
    BacktrackValue := -350.0; NumOfPaths := 0;
    NewExpTerm := LnFactArray[round(1.582*BaseN)] + LnFactArray[K] -
                  InitAdj;
  end;

{*** Update the values of T and S ***}
procedure UpdateTandS (Level : byte);
  begin
    Diff := AArray[Level]-PrevA;
    TAdjust := 0;
    for I := 1 to (Level-1) do
```

```
        TAdjust := TAdjust + AArray[I];
      for I := Level+1 to K do
        TAdjust := TAdjust + AArray[I];
      T := T + TAdjust*Diff;
      if (Level<=5) or (Level=K) then
        begin
          SArray[1] := N - AArray[1];
          SArray[2] := SArray[1] - AArray[2];
          SArray[3] := SArray[2] - AArray[3];
          SArray[4] := SArray[3] - AArray[4];
          SArray[5] := SArray[4] - AArray[5];
          for I := 1 to 5 do
            SArray[I] := AArray[K]*SArray[I];
        end;
    end;

{*** Calculate the final adjustment ***}
procedure CalcFinalAdj;
    var
      I,J : byte;
    begin
      AdjArray[2] := T - SArray[1];
      for I := 3 to 6 do
        AdjArray[I] := AdjArray[I-1] - SArray[I-1];
      FinalAdj := 0.0;
      for I := 2 to 6 do
        FinalAdj := FinalAdj + AdjArray[I]*LnQArray[I];
    end;

{*** Recursive procedure which calculates the value of E(K) by implicitly ***}
{    generating all sequences of length k which sum to n.                    }
procedure TraverseTree (Level : byte);
    var
      Index,MaxA,MinA : word;
      PrevSumA : word;
      PrevNewExpTerm : double;
    begin
      if (Level=K) then
        begin
          PrevNewExpTerm := NewExpTerm;
          PrevA := AArray[K];
          AArray[K] := N - SumA;
          Index := AArray[K];
          Adjust := Comb2Array[Index];
```

```
      NJArray[Index] := NJArray[Index] + 1;
      UpdateTandS (K);
      CalcFinalAdj;
      NewExpTerm := NewExpTerm - LnFactArray[Index] +
                    Comb2Array[Index]*LnQArray[1] + FinalAdj;
      for I := 1 to AArray[1] do
        if (NJArray[I]>1) then
          NewExpTerm := NewExpTerm - 2*LnFactArray[NJArray[I]];
      if (NewExpTerm>BacktrackValue) then
        begin
          NewExpTerm := exp(NewExpTerm);
          ExpNumOfKColorings := ExpNumOfKColorings + NewExpTerm;
          LnExpNumOfKColorings := ln(ExpNumOfKColorings);
          BacktrackValue := LnExpNumOfKColorings + LnEMinusN;
          NumOfPaths := NumOfPaths + 1;
          if (NumOfPaths>=750000) then
            Quit := true;
          MaxClassSize := AArray[1];
          Backtrack := false;
          for I := 1 to (K-1) do
            MaxClassSizeArray[I] := AArray[I];
        end
      else
        Backtrack := true;
      NJArray[Index] := NJArray[Index] - 1;
      NewExpTerm := PrevNewExpTerm;
    end
  else
    begin
      Adjust := Adjust - Comb2Array[AArray[Level]];
      PrevSumA := SumA;
      PrevNewExpTerm := NewExpTerm;
      Num := N - SumA;
      Denom := K - Level + 1;
      Value1 := Num - Denom + 1;
      Value2 := AArray[Level-1];
      if (Value1<=Value2) then
        MaxA := Value1
      else
        MaxA := Value2;
      MinA := (Num div Denom);
      if ((Num mod Denom)>0) then
        MinA := MinA + 1;
      SumA := SumA + MinA;
```

```
          Index := MinA;
          PrevA := AArray[Level];
          while (Index<=MaxA) and (not Backtrack) and (not Quit) do
            begin
              AArray[Level] := Index;
              NewExpTerm := NewExpTerm - LnFactArray[Index] +
                            Comb2Array[Index]*LnQArray[1];
              NJArray[Index] := NJArray[Index] + 1;
              UpdateTandS (Level);
              CalcFinalAdj;
              Test := NewExpTerm + FinalAdj + Adjust*LnQArray[1];
              for I := (Level+1) to K do
                Test := Test - LnFactArray[AArray[I]];
              for I := 1 to AArray[1] do
                if (NJArray[I]>1) then
                  Test := Test - 2*LnFactArray[NJArray[I]];
              if (Test>BacktrackValue) then
                TraverseTree (Level+1)
              else
                Backtrack := true;
              NJArray[Index] := NJArray[Index] - 1;
              NewExpTerm := PrevNewExpTerm;
              SumA := SumA + 1;
              PrevA := Index;
              if (Index=MaxClassSizeArray[Level]) and Backtrack then
                begin
                  Backtrack := false;
                  MaxClassSizeArray[Level] := 0;
                end;
              Index := Index + 1;
            end;
          PrevA := AArray[Level];
          AArray[Level] := MinA;
          UpdateTandS (Level);
          Adjust := Adjust + Comb2Array[MinA];
          SumA := PrevSumA;
        end;
    end;

{*** Calculate the elapsed processing time ***}
procedure CalcElapsedTime;
  var
    BegTime,EndTime : double;
  begin
```

```
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  assign (Out, 'REPORT.PRN'); rewrite (Out);
  ObtainUserInfo;
  PrintReportHeadings;
  ReadCombFactFile;
  BaseN := StartN;
  while (BaseN<=EndN) do
    begin
      Initialize1;
      writeln ('Processing Graph Size ',BaseN);
      writeln ('   K = ',K);
      FoundIt := false;
      while (not FoundIt) do
        begin
          LnEMinusN := -24;
          NumOfPaths := 0;
          Quit := false;
          GetTime (Hour1,Min1,Sec1,Frac1);
          while (NumOfPaths<500000) and (not Quit) do
            begin
              Initialize2;
              TraverseTree (1);
              writeln ('LnEMinusN = ',LnEMinusN:5:1,' NumOfPaths = ',
                    NumOfPaths:7,' ENOKC = ',ExpNumOfKColorings);
            end;
          GetTime (Hour2,Min2,Sec2,Frac2);
          CalcElapsedTime;
          writeln ('      ExpNumOfKColorings = ',ExpNumOfKColorings);
          writeln ('      NumOfPaths = ',NumOfPaths);
          writeln ('      ElapsedTime = ',ElapsedTime:10:5);
          writeln ('      MaxColorClass = ',MaxClassSize);
          if (ExpNumOfKColorings<1.0) then
            begin
              PrevExpNumOfKColorings := ExpNumOfKColorings;
              PrevNumOfPaths := NumOfPaths;
              PrevLnEMinusN := LnEMinusN;
              K := K + 1;
```

```
            writeln ('   K = ',K);
          end
        else
          begin
            FoundIt := true;
            writeln (Out,BaseN:5,' ':3,K-1:5,' ':3,
                  PrevExpNumOfKColorings,' ':3,PrevNumOfPaths:7,
                  ' ':3,PrevLnEMinusN:5:1,' ':3,K:5,' ':3,
                  ExpNumOfKColorings,' ':3,NumOfPaths:7,' ':3,
                  LnEMinusN:5:1);
            K := K - 1;   {Use with Step = 2}
            {K := K + 2;}   {Use with Step = 25}
          end;
      end;
    BaseN := BaseN + StepSize;
  end;
close (Out);
end.
```

20. MIS_Est.Pas

```
program MISEst;
uses CRT;
type
  String11 = string[11];
var
  I,StartVertCnt,EndVertCnt,NumOfVert,SetSize : integer;
  Residue,IndSetSize : integer;
  P,Q,ExpNumOfIndSets : real;
  FileName,Extension : String11;
  MaxIndSetSize : array [1..1000,0..35] of byte;
  ProbStr : string[2];
  FileIn,Out : text;

{*** Generate Natural Logarithm of a Power with Integer Exponent ***}
function LnPower (Base : real; Exponent : integer) : real;
  var
    I : integer;
    Result : real;
  begin
    Result := 0.0;
    for I := 1 to Exponent do
      Result := Result + ln(Base);
    LnPower := Result;
```

```
  end;

{*** Calculate Natural Logarithm of Combinations ***}
function LnComb (N,J : integer) : real;
  var
    I : integer;
    Value : real;
  begin
    Value := 0.0;
    for I := 0 to (J-1) do
      Value := Value + ln((N-I)/(J-I));
    LnComb := Value;
  end;

{*** Generate MIS Disk File ***}
procedure GenerateMISFile;
  var
    VertCnt,Residue : integer;
  begin
    assign (Out,'MIS_P'+ProbStr+'.Dat');
    rewrite (Out);
    writeln (Out,'************************************************');
    writeln (Out,'*                                              *');
    writeln (Out,'*        Composite Graph Coloring              *');
    writeln (Out,'*      Maximal Independent Set Size            *');
    writeln (Out,'*              Estimate                        *');
    writeln (Out,'*                                              *');
    writeln (Out,'*          Edge Density = ',P:4:3,'            *');
    writeln (Out,'*                                              *');
    writeln (Out,'************************************************');
    writeln (Out);
    writeln (Out,'                      Residue          ');
    writeln (Out,'Num Of Vert  0  1  2  3  4  5  6  7  8  9',
                 '  10  11  12  13  14  15  16  17  18  19',
                 '  20  21  22  23  24  25  26  27  28  29',
                 '  30  31  32  33  34  35');
    writeln (Out,'----------  ------------------------------',
                 '------------------------------',
                 '------------------------------',
                 '----------');
    for VertCnt := 1 to EndVertCnt do
      begin
        write (Out,VertCnt:8,' ':3);
        for Residue := 0 to 35 do
```

```
            write (Out,MaxIndSetSize[VertCnt,Residue]:4);
          writeln (Out);
        end;
      close (Out);
    end;

begin
  ClrScr;
  writeln ('Program: MIS_Est.Pas');
  writeln;
  write ('Enter the graph density (p) -- ');
  readln (P);
  writeln;
  write ('Enter the DECIMIAL DIGITS only of the graph density (p) -- ');
  readln (ProbStr);
  writeln;
  write ('Enter start graph size -- ');
  readln (StartVertCnt);
  writeln;
  write ('Enter end graph size -- ');
  readln (EndVertCnt);
  Q := 1.0 - P;
  for NumOfVert := 1 to EndVertCnt do
    for Residue := 0 to 35 do
      MaxIndSetSize[NumOfVert,Residue] := 0;
  writeln;
  writeln ('Generating Maximum Independent Set Sizes');
  for NumOfVert := StartVertCnt to EndVertCnt do
    for Residue := 0 to 35 do
      begin
        writeln;
        writeln ('NumOfVert = ',NumOfVert,'  Residue = ',Residue);
        ExpNumOfIndSets := 0.0;
        IndSetSize := 1;
        if (IndSetSize<=NumOfVert) and (Residue<=NumOfVert) then
          ExpNumOfIndSets :=
            exp(LnComb(NumOfVert-Residue,IndSetSize-Residue) +
              LnPower(Q,(IndSetSize*(IndSetSize-1) div 2) -
              (Residue*(Residue-1) div 2)));
        while (ExpNumOfIndSets>=1.0) and (IndSetSize<=NumOfVert) do
          begin
            IndSetSize := IndSetSize + 1;
            if (IndSetSize<=NumOfVert) and (Residue<=NumOfVert) then
              ExpNumOfIndSets :=
```

```
                    exp(LnComb(NumOfVert-Residue,IndSetSize-Residue) +
                      LnPower(Q,(IndSetSize*(IndSetSize-1) div 2) -
                      (Residue*(Residue-1) div 2)));
            end;
          MaxIndSetSize[NumOfVert,Residue] := IndSetSize - 1;
        end;
  GenerateMISFile;
end.
```

21. S_UB.Pas

```
{$N+,E-}
program SUBound;
uses CRT;
type
  String11 = string[11];
var
  ProbStr : string[2];
  I,StartVertCnt,EndVertCnt,NumOfVert,IndSetSize,RemainNumOfVert : integer;
  P,Q,ExpNumOfIndSets : double;
  FileName,Extension : String11;
  MaxIndSetSize,ChromEst : array [1..1000] of integer;
  FileIn,Out : text;

{*** Generate Natural Log of Power with Integer Exponent ***}
function LnPower (Base : double; Exponent : integer) : double;
  var
    I : integer;
    Result : double;
  begin
    LnPower := Exponent * ln(Base);
  end;

{*** Calculate Natural Log of Combinations ***}
function LnComb (N,J : integer) : double;
  var
    I : integer;
    Value : double;
  begin
    Value := 0.0;
    for I := 0 to (J-1) do
      Value := Value + ln((N-I)/(J-I));
    LnComb := Value;
  end;
```

```
{*** Generate Chromaticity Report ***}
procedure PrintChromEstReport;
  var
    Estimate,I,VertCnt1,VertCnt2 : integer;
  begin
    assign (Out,'Rep2.Prn');
    rewrite (Out);
    writeln (Out,'  Standard Graph Coloring ');
    writeln (Out,'     Chromatic Number     ');
    writeln (Out,'         Estimate         ');
    writeln (Out);
    writeln (Out,'Edge Density = ',P:4:3);
    writeln (Out);
    writeln (Out,'Vertex Range       Estimate');
    writeln (Out,'------------       --------');
    VertCnt1 := StartVertCnt;
    VertCnt2 := StartVertCnt;
    Estimate := ChromEst[StartVertCnt];
    I := StartVertCnt + 1;
    while (I<=EndVertCnt) do
      begin
        if (ChromEst[I]>Estimate) or (I>=EndVertCnt) then
          begin
            if (I=EndVertCnt) then
              VertCnt2 := I;
            writeln (Out,VertCnt1:3,' - ',VertCnt2:3,' ':13,Estimate:2);
            VertCnt1 := I;
            VertCnt2 := I;
            Estimate := ChromEst[I];
          end
        else
          VertCnt2 := I;
        I := I + 1;
      end;
    close (Out);
  end;

begin
  ClrScr;
  writeln ('Program: SProbEst.Pas -- Estimate Chromatic Number of Standard',
        ' Graph');
  writeln;
  write ('Enter the graph density (p) -- ');
  readln (P);
```

```
    writeln;
    write ('Enter the DECIMAL DIGITS of the graph density only -- ');
    readln (ProbStr);
    writeln;
    write ('Enter start graph size -- ');
    readln (StartVertCnt);
    writeln;
    write ('Enter end graph size -- ');
    readln (EndVertCnt);
    Q := 1.0 - P;
    assign (FileIn,'MIS_P'+ProbStr+'.DAT');
    reset (FileIn);
    for I := 1 to 13 do
      readln (FileIn);
    for I := 1 to EndVertCnt do
      begin
        read (FileIn,NumOfVert);
        readln (FileIn,MaxIndSetSize[NumOfVert]);
      end;
    close (FileIn);
    writeln;
    writeln ('Generating Chromaticity Estimates');
    for NumOfVert := StartVertCnt to EndVertCnt do
      begin
        ChromEst[NumOfVert] := 1;
        RemainNumOfVert := NumOfVert - MaxIndSetSize[NumOfVert];
        while (RemainNumOfVert>0) do
          begin
            ChromEst[NumOfVert] := ChromEst[NumOfVert] + 1;
            RemainNumOfVert := RemainNumOfVert -
                               MaxIndSetSize[RemainNumOfVert];
          end;
      end;
    PrintChromEstReport;
end.
```

22. CTP_UB.Pas

```
{$N+,E-}
program CTPUBound;
uses DOS,CRT;
type
  String11 = string[11];
var
```

```
  Hour1,Min1,Sec1,Frac1 : word;
  Hour2,Min2,Sec2,Frac2 : word;
  I,J,K,StartVertCnt,EndVertCnt,StepSize,NumOfVert,SetSize : integer;
  TotResidue,RemainNumOfVert,ChromEstimate : integer;
  TotChromEstimate,TotVertChrom,LongNumOfVert : longint;
  Seed : longint;
  P,Q,Param,CumPDF,ElapsedTime : double;
  FileName,Extension : String11;
  Residue,PrevResidue : array[1..10] of byte;
  MaxIndSetSize : array [1..1000,0..35] of byte;
  ChromEst,AvgVertChrom : array [500..1000] of double;
  MaxChromEst,MinChromEst : array [500..1000] of byte;
  ProbStr : string[2];
  FileIn,Out : text;

{*** Generate Power with Integer Exponent ***}
function Power (Base : double; Exponent : integer) : double;
  var
    I : integer;
    Result : double;
  begin
    Power := exp(Exponent*ln(Base));
    {Result := 1.0;
    for I := 1 to Exponent do
      Result := Result * Base;
    Power := Result;}
  end;

{*** Generate N Factorial ***}
function Fact (N : integer) : integer;
  var
    I,Result : integer;
  begin
    Result := 1;
    for I := 1 to N do
      Result := Result * I;
    Fact := Result;
  end;

{*** Random Number Generator ***}
function Random : double;
  const
    A : longint = 16807;
    M : longint = 2147483647;
```

```
    Q : longint = 127773;
    R : longint = 2836;
  var
    Low,High,Test : longint;
  begin
    High := Seed div Q;
    Low := Seed mod Q;
    Test := A*Low - R*High;
    if (Test>0) then
      Seed := Test
    else
      Seed := Test + M;
    Random := Seed/M;
  end;

{*** Generate Graph Chromatic Number Estimate Report ***}
procedure PrintChromEstReport;
  var
    NumOfVert : integer;
  begin
    assign (Out,'Report.Prn');
    rewrite (Out);
    writeln (Out,'              Composite Graph Coloring       ');
    writeln (Out,'                 Chromatic Number          ');
    writeln (Out,'                    Estimate               ');
    writeln (Out);
    writeln (Out,'Vert Chrom Distrib: Truncated Poisson');
    writeln (Out,'Edge Density = ',P:4:3);
    writeln (Out);
    writeln (Out,'Num Of Vert   Avg Est   Min Est   Max Est   Avg Vert Chrom');
    writeln (Out,'-----------  -------  -------  -------  --------------');
    NumOfVert := StartVertCnt;
    while (NumOfVert<=EndVertCnt) do
      begin
        writeln (Out,NumOfVert:7,ChromEst[NumOfVert]:13:2,
                 MinChromEst[NumOfVert]:9,MaxChromEst[NumOfVert]:10,
                 AvgVertChrom[NumOfVert]:15:4);
        NumOfVert := NumOfVert + StepSize;
      end;
    close (Out);
  end;

procedure CalcElapsedTime;
  var
```

```
    BegTime.EndTime : double;
  begin
    BegTime := Hour1*3600.0 + Min1*60.0 + Sec1*1.0 + Frac1/100.0;
    EndTime := Hour2*3600.0 + Min2*60.0 + Sec2*1.0 + Frac2/100.0;
    ElapsedTime := EndTime - BegTime;
    if (ElapsedTime<0.0) then
      ElapsedTime := ElapsedTime + 86400.0;
  end;

begin
  Seed := 493544361;
  Param := 1.0;
  ClrScr;
  writeln ('Program: CTP_Est.Pas');
  writeln;
  write ('Enter the graph density (p) -- ');
  readln (P);
  writeln;
  write ('Enter only the DECIMAL DIGITS of the graph density (p) -- ');
  readln (ProbStr);
  writeln;
  write ('Enter start graph size -- ');
  readln (StartVertCnt);
  writeln;
  write ('Enter end graph size -- ');
  readln (EndVertCnt);
  writeln;
  write ('Enter step size -- ');
  readln (StepSize);
  Q := 1.0 - P;
  assign (FileIn,'MIS_P'+ProbStr+'.DAT');
  reset (FileIn);
  for I := 1 to 13 do
    readln (FileIn);
  for I := 1 to EndVertCnt do
    begin
      read (FileIn,NumOfVert);
      for TotResidue := 0 to 35 do
        read (FileIn,MaxIndSetSize[NumOfVert,TotResidue]);
      readln (FileIn);
    end;
  close (FileIn);
  writeln;
  writeln ('Generating Chromaticity Estimates');
```

```
NumOfVert := StartVertCnt;
while (NumOfVert<=EndVertCnt) do
  begin
    writeln;
    writeln ('NumOfVert = ',NumOfVert);
    TotVertChrom := 0;
    TotChromEstimate := 0;
    MinChromEst[NumOfVert] := 255;
    MaxChromEst[NumOfVert] := 0;
    GetTime (Hour1,Min1,Sec1,Frac1);
    for I := 1 to 1000 do
      begin
        ChromEstimate := 1;
        SetSize := MaxIndSetSize[NumOfVert,0];
        for K := 1 to 10 do
          Residue[K] := 0;
        for J := 1 to SetSize do
          begin
            CumPDF := Param/(Exp(Param)-1);
            K := 1;
            while (CumPDF<Random+0.0635) do
              begin
                K := K + 1;
                CumPDF := CumPDF +
                        Power(Param,K)/((Exp(Param)-1)*Fact(K));
              end;
            if (K>=2) then
              Residue[K-1] := Residue[K-1] + 1;
            TotVertChrom := TotVertChrom + K;
          end;
        TotResidue := 0;
        for K := 1 to 10 do
          TotResidue := TotResidue + Residue[K];
        if (TotResidue>35) then
          writeln('TotResidue = ',TotResidue);
        RemainNumOfVert := NumOfVert - SetSize + TotResidue;
        while (RemainNumOfVert>0) do
          begin
            ChromEstimate := ChromEstimate + 1;
            SetSize := MaxIndSetSize[RemainNumOfVert,TotResidue];
            for K := 1 to 10 do
              PrevResidue[K] := Residue[K];
            for K := 1 to 9 do
              Residue[K] := PrevResidue[K+1];
```

```
                Residue[10] := 0;
                for J := 1 to (SetSize-TotResidue) do
                  begin
                    CumPDF := Param/(Exp(Param)-1);
                    K := 1;
                    while (CumPDF<Random+0.0635) do
                      begin
                        K := K + 1;
                        CumPDF := CumPDF +
                            Power(Param,K)/((Exp(Param)-1)*Fact(K));
                      end;
                    if (K>=2) then
                      Residue[K-1] := Residue[K-1] + 1;
                    TotVertChrom := TotVertChrom + K;
                  end;
                TotResidue := 0;
                for K := 1 to 10 do
                  TotResidue := TotResidue + Residue[K];
                if (TotResidue>35) then
                  writeln('TotResidue = ',TotResidue);
                RemainNumOfVert:= RemainNumOfVert-SetSize + TotResidue;
              end;
            if (ChromEstimate>MaxChromEst[NumOfVert]) then
              MaxChromEst[NumOfVert] := ChromEstimate;
            if (ChromEstimate<MinChromEst[NumOfVert]) then
              MinChromEst[NumOfVert] := ChromEstimate;
            TotChromEstimate := TotChromEstimate + ChromEstimate;
          end;
        GetTime (Hour2,Min2,Sec2,Frac2);
        CalcElapsedTime;
        ChromEst[NumOfVert] := TotChromEstimate/1000.0;
        LongNumOfVert := NumOfVert;
        AvgVertChrom[NumOfVert] := TotVertChrom/(1000.0*LongNumOfVert);
        writeln ('  Chi Estimate = ',ChromEst[NumOfVert]:7:2);
        writeln ('  Avg Vert Chrom = ',AvgVertChrom[NumOfVert]:6:4);
        writeln ('  Elapsed Time = ',ElapsedTime:7:2,' sec');
        NumOfVert := NumOfVert + StepSize;
      end;
    PrintChromEstReport;
end.
```

REFERENCES

[*Bo85*] Bollobas, B. *Random Graphs*, Academic Press, London, 1985, pp. 399-409.

[*Br72*] Brown, J. R. "Chromatic Scheduling and the Chromatic Number Problem," *Management Science*, 19, 4 (1972), pp. 456-463.

[*Br73*] Bron, C. and Kerbosch, J. "Finding All Cliques of an Undirected Graph," *Communications of the ACM*, 16, 9 (1973), pp. 575-577.

[*Br79*] Brelaz, D. "New Methods to Color the Vertices of a Graph," *Communications of the ACM*, 22 (1979), pp. 251-256.

[*BT85*] Bollobas, B. and Thomason, A. "Random Graphs of Small Order," *Annals of Discrete Mathematics*, 28 (1985), pp. 47-97.

[*CE83*] Clementson, A. T. and Elphick, C. H. "Approximate Colouring Algorithms for Composite Graphs," *Journal of the Operational Research Society*, 34, 6 (1983), pp. 503-509.

[*Ch71*] Christofides, N. "An Algorithm for the Chromatic Number of a Graph," *Computer Journal*, 14, 1 (1971), pp. 38-39.

[*Ch75*] Christofides, N. *Graph Theory - An Algorithmic Approach*, Academic Press, London, 1975, pp. 30-78.

[*Ha68*] Hale, W. "Frequency Assignment: Theory and Applications," *Proceedings of the IEEE*, 68, pp. 1497-1514.

[*JA89*] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. "Optimization by Simulated Annealing: An Experimental Evaluation, Part II (Graph Coloring and Number Partitioning)," *Manuscript, AT&T Bell Labs*, Murray Hill, 1989.

[*JM82*] Johri, A. and Matula, D. W. "Probabilistic Bounds and Heuristic Algorithms for Coloring Large Random Graphs," *Technical Report, Southern Methodist University*, Dallas, 1982.

[*Jo74*] Johnson, D. S. "Worst Case Behavior of Graph Coloring Algorithms," *Proceedings of the Fifth Southeast Conference on Combinatorics, Graph Theory and Computing*, Hoffman, F., Kingsley, R. A., Levow, R. B., Mullin, R. C., and Thomas, R. S. (editors) Utilitas Mathematica Publishing, Winnipeg, 1974, pp. 513-527.

[*Ka72*] Karp, R. M. "Reducibility Among Combinatorial Problems," *Complexity of Computer Computations*, Miller, R. E. and Thatcher, J. W. (editors), Plenum Press, New York, 1972, pp. 85-103.

[*KG83*] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. "Optimization by Simulated Annealing," *Science*, **220** (1983), pp. 671-680.

[*KJ85*] Kubale, M. and Jackowski, B. "A Generalized Implicit Enumeration Algorithm for Graph Coloring," *Communications of the ACM*, **28, 4** (1985), pp. 412-418.

[*Ko79*] Korman, S. M. "The Graph Coloring Problem," *Combinatorial Optimization*, Christofides, N., Mingozzi, A., Toth, P., and Sandi, C. (editors), Wiley, New York, 1979, pp. 211-235.

[*Le79*] Leighton, F. T. "A Graph Coloring Algorithm for Large Scheduling Problems," *Journal of Research of the National Bureau of Standards*, 84, **6** (1979), pp. 489-506.

[*Li89*] Lin, S. "Graph Coloring Algorithms on Random Graphs," *Ph.D. Dissertation, University of Missouri-Rolla*, 1989.

[*LS86*] Lotfi, V., Sanjiv, S. "A Graph Coloring Algorithm for Large Scale Scheduling Problems," *Computer and Operations Research*, **13, 1** (1986), pp. 27-32.

[*Ma70*] Matula, D. W. "On the Complete Subgraphs of a Random Graph," *Combinatoric Math and its Applications*, Chapel Hill, N.C., 1970, pp. 356-369.

[*MB83*] Matula, D. W. and Beck, L. L. "Smallest Last Ordering and Clustering and Graph Coloring Algorithms," *Journal of the Association of Computing Machinery*, **30, 3** (1983), pp. 417-427.

[*Me65*] Meyer, P. L. *Introductory Probability and Statistical Applications*, Addison-Wesley Publishing, London, 1965, pp. 22-38.

[*Me81*] Mehta, N. "The Application of a Graph Coloring Method to an Examination Scheduling Problem," *Interfaces*, **11**, 5 (1981), pp. 57-64.

[*Mi76*] Mitchem, J. "On Various Algorithms for Estimating the Chromatic Number of a Graph," *The Computer Journal*, **19**, 2 (1976), pp. 182-183.

[*MM72*] Matula, D. W., Marble, G., and Isaacson, J. "Graph Coloring Algorithms," *Graph Theory and Computing*, Read, R. (editor), Academic Press, New York, 1972, pp. 109-122.

[*MS86*] Morgenstern, C. A. and Shapiro, H. D. "Chromatic Number Approximation Using Simulated Annealing," *Manuscript, University of New Mexico*, Albuquerque, 1986.

[*Pe86*] Peemoller, J. "Numerical Experiences with Graph Coloring Algorithms," *European Journal of Opeations Research*, **24** (1986), pp. 146-151.

[*Ro87*] Roberts, J. "Heuristic Coloring Algorithms For the Composite Graph Coloring Problem," *Ph.D. Dissertation, University of Missouri-Rolla*, 1987.

[*SL90*] Sager, T. J. and Lin, S. J. "A Pruning Procedure for Exact Graph Coloring," *Manuscript, University of Missouri-Rolla*, 1990.

[*VA87*] Van Laarhoven, P. J. and Aarts E. H. *Simulated Annealing: Theory and Applications*, Reidel Publishing Company, Boston, 1987.

[*Wo69*] Wood, D. C. "A Technique for Coloring a Graph Applicable to Large Scale Timetabling Problems," *The Computer Journal*, **12** (1969), pp. 317-319.