01 Dec 1987

Heuristic Coloring Algorithm for the Composite Graph Coloring Problem

Johnnie C. Roberts

Billy E. Gillett
Missouri University of Science and Technology

HEURISTIC COLORING ALGORITHM FOR THE COMPOSITE GRAPH COLORING PROBLEM

Johnnie C. Roberts* and Billy E. Gillett

CSc-87-19

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 December 1987.

ABSTRACT

A composite graph is a finite undirected graph in which a positive integer known as a chromaticity is associated with each vertex of the graph. The composite graph coloring problem (CGCP) is the problem of finding the chromatic number of a composite graph, i.e., the minimum number of colors (positive integers) required to assign a sequence of consecutive colors to each vertex of the graph in a manner such that adjacent vertices are not assigned sequences with colors in common and the sequence assigned to a vertex has the number of colors indicated by the chromaticity of the vertex. The CGCP problem is an NP-complete problem that has applications to scheduling and resource allocation problems in which the tasks to be scheduled are of unequal durations.

The pigeonhole principle gives rise to a problem reduction technique for the CGCP and a vertex ordering used in the vertex-sequential-with-interchange (VSI) algorithm, LFPHI. An upper bound on the chromatic number of a composite graph is obtained from the definition of a color-sequential coloring algorithm for the CGCP.

The performances of twelve heuristic coloring algorithms are compared on a variety of random composite graphs. Three VSI algorithms (LF1I, LFPHI, and LFCDI) performed superior to the other algorithms on graphs having the lower numbers of vertices and low edge densities while two color-sequential algorithms (RLF1 and RLFD1) were superior on graphs having the higher numbers of vertices and high edge densities.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

I. INTRODUCTION

A coloring of a finite undirected graph G which has no loops is an assignment of a color, a positive integer, to each vertex of G in a manner such that two adjacent vertices are not assigned the same color. The chromatic number of G is the minimum number of colors required for a coloring of G. The standard graph coloring problem (GCP) is the problem of finding the chromatic number of a graph G. The GCP has been applied to a variety of problems. Among these problems is a class of scheduling and resource allocation problems in which the tasks to be scheduled are of equal duration and use a set of serially reusable resources.

The composite graph coloring problem (CGCP) was introduced to solve similar scheduling and resource allocation problems in which the tasks to be scheduled can have unequal durations [1]. A composite graph is a finite undirected graph which has no loops and in which each vertex has associated with it a positive integer known as the chromaticity of the vertex. The chromaticity of a vertex indicates the number of consecutive colors that are to be assigned to the vertex in a coloring. A coloring of a composite graph G is an assignment of a sequence of consecutive colors to each vertex of G in a manner such that the sequence has the number of colors determined by the chromaticity of the vertex and two adjacent vertices of G are not assigned sequences of colors that have a color in common. The chromatic number of a composite graph G is the minimum number of colors required for a coloring of G. The CGCP is the problem of finding the chromatic number of a composite graph G.

The standard graph coloring problem [2] and the composite graph coloring problem are NP-complete, so the existence of polynomial-time algorithms that will solve these problems exactly is doubtful. For the GCP, several heuristic algorithms have been developed to color a graph yielding an upper bound on the chromatic number of the graph. This upper bound is often considered an approximation of the chromatic number, but the quality of the approximation can be rather poor [3,4].

Three such algorithms for the GCP serve as motivation for the algorithms for the CGCP to be discussed in this paper. These algorithms for the GCP are: (1) the largest-first (LF) algorithm, (2) the largest-first-with-interchange (LFI) algorithm, and (3) the recursive largest-first (RLF) algorithm.

The LF and the LFI algorithms use the degrees of the vertices as a measure to determine the order in which the vertices are to be colored. In these algorithms, vertices of higher degrees are colored prior to those of lower degrees. The order in which the vertices will be colored is determined prior to any vertex being colored. The vertices are ordered in decreasing order according to their degrees. In the LF algorithm, the vertices in this order are assigned the least color possible. The LFI algorithm colors the vertices in the same fashion as the LF algorithm except when a vertex is to be assigned a color that has not been assigned previously. In this case, an interchange technique is used to attempt to rearrange the colors of some of the previously colored vertices to prevent the use of the new color.

Clementson and Elphick [1] presented four heuristic algorithms (LF1, LF2, LF1I, LF2I) for coloring a composite graph. The largest-first-by-chromaticity (LF1) and the largest-first-by-chromatic-degree (LF2) algorithms are generalizations of the LF algorithm for the GCP. The LF1I and the LF2I algorithms use an interchange technique in a fashion similar to the LFI algorithm to reduce the number of colors assigned by the LF1 and the LF2 algorithms, respectively. The measures used to determine the order in which the vertices are colored are the chromaticities of the vertices and the chromatic degrees of the vertices. These algorithms order the vertices in decreasing order according to a primary measure and suborder the vertices of equal primary measure in decreasing order according to a secondary measure. In the LF1 ordering, the primary measure is the chromaticity of a vertex and the secondary measure is the chromatic degree of a vertex. In the LF2 ordering, the primary measure is the chromatic degree of a vertex and the secondary measure is the chromaticity of a vertex. Clementson and Elphick reported that the ordering according to decreasing chromaticities of the vertices yielded better results than the ordering according to decreasing chromatic degrees of the vertices. These results indicate that vertices of high chromaticities are reasonable candidates for preference when ordering the vertices for a coloring algorithm.

The composite graph coloring problem inherits all the difficulties inherent in the standard graph coloring problem and includes any additional difficulties associated with the chromaticities. The LF and the LFI algorithms for the GCP achieve their results by giving preference to vertices of high degrees. The LF1 and the LF1I algorithms for the CGCP achieve their results by giving preference to vertices of high chromaticities. The new "largest-first" algorithms for the CGCP to be discussed blend these strategies so that the chromaticity of a vertex is not the overriding factor to determine when the vertex is to be colored. In some situations, it may be desirable to color a vertex of intermediate (or even low) chromaticity and high degree prior to a vertex of high chromaticity and low degree. The algorithms being presented order the vertices according to a function of two or more of the following measures: (1) the chromaticities of the vertices, (2) the chromatic degrees of the vertices, and (3) the degrees of the vertices. The largest-first-by-pigeonhole-measure (LFPH) algorithm orders the vertices in decreasing order according to the vertices' pigeonhole measures. The pigeonhole measure of a vertex is a function of the three measures mentioned above motivated by the pigeonhole principle. The largest-first-by-chromaticity-times-degree (LFCD) algorithm orders the vertices in decreasing order according to the product of the chromaticity and the degree of a vertex. This product has the desirable properties being sought and is a major term in the pigeonhole measure of a vertex. The LFPHI and the LFCDI algorithms use the interchange technique of Clementson and Elphick to reduce the number of colors assigned by the LFPH and the LFCD algorithms, respectively.

The RLF algorithm for the GCP does not preorder the vertices as do the LF and the LFI algorithms. The RLF algorithm proceeds through the colors in order coloring as many vertices as possible with a color before proceeding to the next color. The algorithm starts with 1 as the current color. The vertex of the highest degree is selected to be the first vertex to be colored with the current color. The algorithm continues to select vertices to be colored with the current color until all remaining uncolored vertices are adjacent to a colored vertex of the current color. A vertex to be selected is in some manner "closest" to the colored vertices of the current color. (The vertex is an uncolored vertex adjacent to the greatest number of uncolored vertices that are adjacent to colored vertices of the current color.) When no more vertices can be colored with the current color, the algorithm is applied to the induced subgraph on the set of remaining uncolored vertices using the next color as the current color. In the RLF algorithm, the measures to determine the next vertex to be colored are of a dynamic nature. The measures are modified as the coloring progresses. The RLF1 and the RLFD1 algorithms are generalizations of the RLF algorithm using the maximum chromaticity of the vertices as the primary measure for selecting the next vertex to be colored. The remaining measures retain the dynamic nature evident in the RLF algorithm.

The remaining two new algorithms to be presented are the dynamic-pigeonhole-measure (DYNPH) and the dynamic-floating-point-pigeonhole-measure (DYNFPH) algorithms. These algorithms use measures based upon the pigeonhole measure to determine the next vertex to be colored. For each iteration of these algorithms, the vertex with the largest measure is selected to be colored and is assigned the lowest possible sequence of colors. After the vertex is colored, the measures are updated to reflect the effects of coloring the vertex. The measures for these algorithms are of a dynamic nature. These measures are also dependent on the colors that are assigned unlike the measures used in the RLF1 and the RLFD algorithms.

Two problem reduction techniques for the CGCP are described in Chapter VIII. These techniques can be used to reduce the size of the graph to be colored by eliminating vertices from the graph that are guaranteed not to increase the number of colors required for a coloring. The colors for the eliminated vertices can be determined easily after the remaining vertices are colored.

To compare the algorithms that have been described for the CGCP, the algorithms were used to color a variety of random composite graphs. These random composite graphs have various numbers of vertices, edge densities, and distributions of the chromaticities. The results of these experiments and the conclusions drawn from these results are presented in Chapter IX.

II. PRELIMINARY DEFINITIONS

A. NOTATION

The set of integers is denoted by $\mathbb{Z}$ and the set of positive integers by $\mathbb{Z}^+$. A sequence of consecutive integers is represented by $I[a,b]$ where $I[a,b]$ denotes the set of integers in the closed interval $[a,b]$.

$$I[a,b] = \{i \in \mathbb{Z}: a \leq i \leq b\} = [a,b] \cap \mathbb{Z}.$$

The notation (u,v) denotes an unordered pair and $\langle u,v\rangle$ denotes an ordered pair. So, if $u \neq v$, then $(u,v) = (v,u)$ but $\langle u,v\rangle \neq \langle v,u\rangle$.

B. FINITE UNDIRECTED GRAPH

A <u>finite undirected graph</u> is an ordered triple $\langle V,E,\Phi\rangle$ where V is a nonempty finite set of elements known as <u>vertices</u>, E is a finite set of elements known as <u>edges</u>, and Φ is a function from E into the set of unordered pairs on V [5]. The <u>endpoints</u> of an edge e are the vertices u and v such that $\Phi(e) = (u,v)$. An edge e is <u>incident to</u> a vertex v and vertex v is <u>incident to</u> edge e provided v is an endpoint of e. Two vertices u and v are <u>adjacent</u> provided $(u,v) \in \Phi(E)$, that is, there is an edge with u and v as its endpoints. Two edges e_1 and e_2 are <u>adjacent</u> provided e_1 and e_2 have a common endpoint. An edge e is a <u>loop</u> provided $\Phi(e) = (v,v)$ for some $v \in V$. Two distinct edges e_1 and e_2 are <u>multiple edges</u> provided $\Phi(e_1) = \Phi(e_2)$, that is, they have the same endpoints. The <u>degree</u> of a vertex v in a graph G, denoted $d_G(v)$ or $d(v;G)$, is the number of edges in

G incident to v with loops being counted twice. When the graph G is clear from the context, $d_G(v)$ will be written $d(v)$. An <u>isolated vertex</u> is a vertex that is not adjacent to any other vertices in the graph. A <u>complete graph</u> is a graph in which every vertex is adjacent to all other vertices.

Figure 1 illustrates an example of a finite undirected graph $G = \langle V, E, \Phi \rangle$ where $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$, and Φ is defined as follows: $\Phi(e_1) = (v_1, v_2)$, $\Phi(e_2) = (v_2, v_4)$, $\Phi(e_3) = (v_3, v_4)$, $\Phi(e_4) = (v_1, v_3)$, $\Phi(e_5) = (v_1, v_4)$, $\Phi(e_6) = (v_3, v_4)$, and $\Phi(e_7) = (v_2, v_2)$. Edges e_3 and e_6 are a pair of multiple edges connecting vertices v_3 and v_4. Edge e_7 is a loop. Vertex v_5 is an isolated vertex.

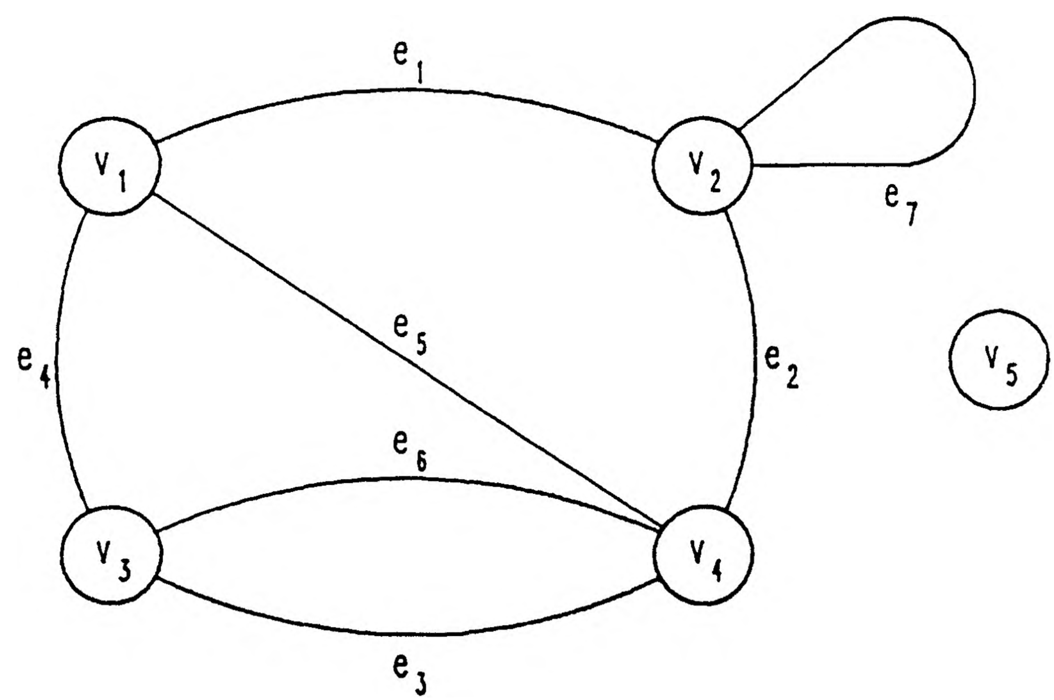


Figure 1. Example of a finite undirected graph

A graph $H = \langle V_H, E_H, \Phi_H \rangle$ is a <u>subgraph</u> of a graph $G = \langle V, E, \Phi \rangle$ provided

(1) $V_H \subseteq V$,

(2) $E_H \subseteq E$, and

(3) $\Phi_H(e) = \Phi(e)$ for each $e \in E_H$.

An <u>induced subgraph</u> of a graph $G = \langle V, E, \Phi \rangle$ on a set of vertices U where $U \subseteq V$ is the graph $H = \langle U, E_H, \Phi_H \rangle$ where $E_H = \{e \in E: \Phi(e) = (u, v)$ for some $u \in U$ and $v \in U\}$ and $\Phi_H(e) = \Phi(e)$ for each $e \in E_H$. If $U \subset V$, then the notation $G - U$ denotes the induced subgraph of $G = \langle V, E, \Phi \rangle$ on the set of vertices $V - U$. If U is a singleton, say $U = \{u\}$, then the shorthand notation $G - u$ is used for $G - \{u\}$. If $U \subseteq V$, then the notation $\langle U \rangle$ denotes the induced subgraph of G on U.

In the graph coloring problems to be considered, multiple edges do not change the result of the problem. A group of multiple edges connecting two vertices can be replaced by a single edge connecting the two vertices. For a graph $G = \langle V, E, \Phi \rangle$ with no multiple edges, Φ maps E one-to-one onto $\Phi(E)$. In such a graph, E is "essentially" the same as $\Phi(E)$. So, G is commonly referred to as an ordered pair $\langle V, E \rangle$ where E is a set of unordered pairs on V.

C. STANDARD GRAPH COLORING PROBLEM

Let $G = \langle V, E, \Phi \rangle$ be a finite undirected graph with no loops and no multiple edges. A <u>coloring</u> of the graph G is a function from V into $\mathbb{Z}^+$ such that for each $u \in V$ and $v \in V$, if u is adjacent to v, then $f(u) \neq f(v)$. The positive integer $f(u)$ is called the color of the vertex u. For a coloring f of G, the highest color assigned to any vertex is denoted by $\chi(f;G)$.

$$\chi(f;G) = \max \{f(u): u \in V\}.$$

For an integer k, a coloring f of G is a k-coloring of G provided $\chi(f;G) \leq k$. The graph G is k-colorable provided G has a k-coloring. The chromatic number of G, denoted $\chi(G)$, is the minimum number of colors required for a coloring of G.

$$\chi(G) = \min \{\chi(f;G): f \text{ is a coloring of } G\}.$$

If G is k-colorable but not (k - 1)-colorable, then $\chi(G) = k$. A coloring f of G is an optimal coloring if $\chi(f;G) = \chi(G)$. The standard graph coloring problem (GCP) is the problem of finding the chromatic number of a finite undirected graph.

D. COMPOSITE GRAPH

A composite graph is an ordered quadruple $\langle V,E,\Phi,C\rangle$ where $\langle V,E,\Phi\rangle$ is a finite undirected graph with no loops and no multiple edges and C is a function from V into $\mathbb{Z}^{+}$. The value $C(v)$ for $v \in V$ is known as the chromaticity of the vertex v. The chromatic degree of a vertex v, denoted $\Delta_G(v)$ or $\Delta(v;G)$, is the sum of the chromaticity of v and the chromaticities of all vertices adjacent to v.

$$\Delta_G(v) = C(v) + \sum_{u \in \Gamma(v)} C(u)$$

where $\Gamma(v) = \{u \in V\colon u \text{ is adjacent to } v\}$. When the graph G is clear from the context, $\Delta_G(v)$ will be written $\Delta(v)$.

E. COMPOSITE GRAPH COLORING PROBLEM

Let $G = \langle V, E, \Phi, C\rangle$ be a composite graph. A <u>coloring</u> F of G is a function from V into $\{I[a,b]\colon a \in \mathbb{Z}^+,\ b \in \mathbb{Z}^+,\ a \leq b\}$ such that

(1) for each $v \in V$, $|F(v)| = C(v)$ and

(2) for each $v \in V$ and $u \in V$, if u is adjacent to v, then $F(u) \cap F(v) = \phi$.

For a coloring F, each positive integer in the sequence of consecutive integers, F(v), assigned to a vertex v is a <u>color</u> of v. For a coloring F of G, the highest color used in the coloring is denoted by $\mathfrak{X}(F;G)$.

$$\mathfrak{X}(F;G) = \max\left[\bigcup_{v \in V} F(v)\right].$$

The <u>chromatic number</u> of a composite graph G, denoted $\mathfrak{X}(G)$, is the minimum number of colors required for a coloring of G.

$$\mathfrak{X}(G) = \min\{\mathfrak{X}(F;G)\colon F \text{ is a coloring of } G\}.$$

A coloring F of G is an <u>optimal coloring</u> if $\mathfrak{X}(F;G) = \mathfrak{X}(G)$. The <u>composite graph coloring problem</u> (CGCP) is the problem of finding the chromatic number of a composite graph.

The composite graph coloring problem for a composite graph with the chromaticities of all the vertices equal to 1 (or any positive integer) is equivalent to the standard graph coloring problem. Thus, the standard graph coloring problem is a special case of the composite graph coloring problem.

III. REVIEW OF LITERATURE

A. HEURISTIC COLORING ALGORITHMS FOR THE STANDARD GRAPH COLORING PROBLEM

For the standard graph coloring problem, several polynomial-time algorithms have been developed to color a graph and yield an approximation of the chromatic number of the graph. Most of these algorithms are from three types of coloring algorithms: vertex-sequential coloring algorithms, vertex-sequential-with-interchange coloring algorithms, and color-sequential coloring algorithms.

1. Vertex-sequential Coloring Algorithms. Vertex-sequential coloring algorithms arrange the vertices of the graph in some order and then assign to each vertex in sequence the least color possible. A vertex-sequential coloring algorithm coloring a graph G whose vertices are arranged in the order $v_1, v_2, v_3, \ldots, v_n$ would generate a coloring f for which $f(v_1) = 1$ and $f(v_k) = \min \{i \in \mathbb{Z}^+ : i \neq f(v_j)$ if v_j is adjacent to v_k for some $j \in I[1,k-1]\}$ for each $k \in I[2,n]$. Two vertex-sequential algorithms that are commonly discussed in the literature are the largest-first (LF) and the smallest-last (SL) algorithms [6]. The LF algorithm orders the vertices in order of decreasing degrees. The SL algorithm arranges the vertices in an order in which each vertex has the smallest degree in the induced subgraph on the set of vertices preceding and including the vertex. If $v_1, v_2, v_3, \ldots, v_n$ is a SL ordering of the

vertices of a graph G and G_i is the induced subgraph of G on $\{v_1, v_2, v_3, \ldots, v_i\}$ for each $i \in I[1,n]$, then $d_{G_i}(v_i) \leq d_{G_i}(v_j)$ for each $j \in I[1,i]$ and $i \in I[1,n]$. The determining of a SL ordering for the n vertices of a graph G can be described recursively as follows:

(1) Select a vertex u with the smallest degree to be the n-th vertex in the ordering.

(2) Order the remaining n - 1 vertices by determining a SL ordering of the n - 1 vertices of the graph G - u.

The LF and SL orderings are both based on the rationale that vertices of higher degrees should be colored before those of lower degrees.

<u>2. Vertex-sequential-with-interchange Coloring Algorithms.</u> A vertex-sequential-with-interchange (VSI) algorithm colors a graph in the same fashion as a vertex-sequential algorithm except when a vertex is to be assigned a color that has not been assigned previously. In this case, a VSI algorithm uses an interchange technique to attempt to prevent the use of the new color. The interchange technique searches previously colored vertices to determine whether interchanging two colors on some of these vertices can prevent the use of the new color. Suppose $G = \langle V, E, \Phi \rangle$ is a graph that is being colored and f is the coloring that is being generated. For two distinct vertex colors i and j, if $U \subseteq V$ such that $f(v) \in \{i, j\}$ for each $v \in U$, then an <u>(i,j)-interchange</u> on U is a redefinition

of f such that for each vertex $v \in U$ which originally had $f(v) = i$, $f(v)$ is redefined to j, and vice versa.

Two interchange techniques have been presented in the literature for the standard graph coloring problem. To discuss these interchange techniques, let us assume that a graph G is being colored by a VSI coloring algorithm and a coloring f is being generated. The vertices of G are arranged in the order $v_1, v_2, v_3, \ldots, v_n$. Suppose the first $k - 1$ vertices have been assigned colors and v_k is the next vertex to be colored. Let M be the number of colors used to color the first $k - 1$ vertices, that is, $M = \max \{f(v_i): i \in I[1,k - 1]\}$. Let p be the least color that can be assigned to v_k if the colors for the previous vertices are left unchanged, that is, $p = \min \{i \in \mathbb{Z}^+: i \neq f(v_j)$ if v_j is adjacent to v_k for some $j \in I[1,k - 1]\}$. If $p \leq M$, then the vertex v_k is assigned the color p, else an interchange technique is applied at this time.

The interchange technique described by Matula, Marble, and Isaacson [6] considers the colors that are currently assigned to exactly one vertex adjacent to v_k as the candidate colors to participate in an interchange. Let K be the set of these colors. $K = \{i \in I[1,M]$: there is exactly one $j \in I[1,k - 1]$ such that v_j is adjacent to v_k and $f(v_j) = i\}$. For $i,j \in I[1,M]$, define G_{ij} to be the induced subgraph of G on $\{v_h: h \in I[1,k - 1]$ and $f(v_h) \in \{i,j\}\}$. If, for some $i,j \in K$, there is a component of G_{ij} that has exactly one vertex adjacent to v_k, then perform an

(i,j)-interchange on that component of G_{ij}. The (i,j)-interchange will result in either i or j not being assigned to a vertex adjacent to v_k. Assign v_k that color. If no interchange was possible, assign v_k the color M + 1.

The interchange technique described by Johnson [3,7] is an extension of the interchange technique of Matula, Marble, and Isaacson. In this interchange technique, if there are $i,j \in I[1,M]$ such that each component of G_{ij} has vertices adjacent to v_k of only one color, then an interchange can be performed. Select such an i and j and perform an (i,j)-interchange on each component of G_{ij} which contains a vertex v which is adjacent to v_k and $f(v) = i$. If no interchange was possible, assign v_k the color M + 1.

The largest-first-with-interchange (LFI) and the smallest-last-with-interchange (SLI) algorithms are VSI algorithms in which the vertices are arranged in a LF ordering and a SL ordering, respectively. In the literature, the names, "LFI" and "SLI", have referred to algorithms using either of the two interchange techniques.

3. Color-sequential Coloring Algorithms. Color-sequential coloring algorithms color a graph in a manner such that before a vertex is assigned a color k all vertices that are to have colors less than k have already been assigned those colors. A color-sequential algorithm starting with the color 1 assigns a color to as many vertices as possible before proceeding to assign the next color to vertices. Upon completion of assigning a color k

to vertices of a graph, each remaining uncolored vertex is adjacent to at least one vertex that has been assigned the color j for each $j \in I[1,k]$. Below is a pseudocode description of a color-sequential coloring algorithm to color a graph $G = \langle V,E,\Phi \rangle$. This description serves as a framework for four color-sequential algorithms that have appeared in the literature. These algorithms differ in the manner in which the next vertex to be colored is selected. In the algorithm description, U is the set of all uncolored vertices, initially V, i is the current color being assigned by the algorithm, and U_1 is the set of all uncolored vertices that are not adjacent to vertices of the current color.

<u>Color-sequential Coloring Algorithm</u>

```
Let i = 0.
Let U = V.
WHILE U ≠ φ
  i = i + 1.
  U1 = U.
  WHILE U1 ≠ φ
    Select a vertex v ∈ U1 to be colored.
    Assign v the color i.
    U1 = U1 - ({v} U {u ∈ U1: u is adjacent to v}).
    U = U - {v}.
  END WHILE
END WHILE
```

Welsh and Powell [8] described a color-sequential coloring algorithm that is equivalent to the LF vertex-sequential coloring algorithm. The two algorithms are equivalent in the sense that there is a LF ordering of the vertices of the graph being colored for which the LF algorithm generates the same coloring as the Welsh and Powell algorithm. The Welsh and Powell algorithm selects an uncolored vertex that is not adjacent to a vertex of the current color to be the next vertex to be colored. The algorithm selects a vertex $v \in U_1$ such that $d(v) = \max \{d(u): u \in U_1\}$. Williams [9] refers to this algorithm as the "Peck-Williams heuristic procedure".

Williams described a variation of the Peck-Williams heuristic procedure in which the next vertex to be colored is not determined by the degree vector but by a vector obtained by multiplying a power of the adjacency matrix of the graph by the degree vector. If the vertices of the graph are labelled $v_1, v_2, v_3, \ldots, v_n$, then the degree vector $\bar{d}$ is the vector whose components are $d_i = d(v_i)$ for all $i \in I[1,n]$. Let A be the adjacency matrix of the graph being colored. For $k \in \mathbb{Z}^+$, the vector $A^k \bar{d}$ approximates an eigenvector corresponding to the dominant eigenvalue of the adjacency matrix of the graph. For a particular $k \in \mathbb{Z}^+$, calculate $\bar{d}^{(k)} = A^{k-1} \bar{d}$. To select the next vertex to be colored, the algorithm selects an uncolored vertex $v_i \in U_1$ such that $d_i^{(k)} = \max \{d_j^{(k)}: j \in I[1,n] \text{ and } v_j \in U_1\}$. According to Williams, for a graph having n vertices.

choosing $k \approx \sqrt[3]{n}$ is generally sufficient to observe a significant improvement in the heuristic. In [10], it was reported that proceeding beyond $k = 1$ does not appear to result in significant improvements.

Johnson [3] described a color-sequential coloring algorithm known as the approximately maximum independent set (AMIS) coloring algorithm. The AMIS algorithm exhibits a better worst case behavior than the previously described heuristic algorithms. On a graph G with n vertices, the AMIS algorithm uses at most $O(\frac{n}{\log n})\ \mathfrak{X}(G)$ colors, while the previously described algorithms can use $O(n)\ \mathfrak{X}(G)$ colors. In the AMIS algorithm, the next vertex to be colored is an uncolored vertex of minimum degree in the induced subgraph of G on the set of uncolored vertices that are not adjacent to a vertex of the current color. The AMIS algorithm selects a vertex $v \in U_1$ such that $d(v;\langle U_1\rangle) = \min\ \{d(u;\langle U_1\rangle):\ u \in U_1\}$. Computational experience has shown that, on the average, the AMIS algorithm produces colorings that are substantially inferior to those produced by the LF and the SL algorithms [7].

Leighton [7] described a color-sequential coloring algorithm known as the recursive largest-first (RLF) algorithm that "combines the strategy of the LF algorithm with the structure of the AMIS algorithm." The RLF algorithm selects the next vertex to be colored by considering an uncolored vertex's relationship to two disjoint subsets, U_1 and U_2, of U, the set of all uncolored vertices. U_1 is the set of all uncolored vertices that are not adjacent to a vertex of the current color and U_2 is the set of all uncolored vertices that are adjacent to a vertex of the current color. The first vertex to be assigned a color k is a vertex of the highest degree in the induced subgraph on U_1. Each remaining vertex to be assigned the color k is selected by choosing a vertex in U_1 that is adjacent to the greatest number of vertices in U_2. If there is more than one such vertex, then a vertex with the lowest degree in $\langle U_1 \rangle$ is selected from among these vertices. The number of vertices in U_2 adjacent to a vertex v is $d(v;\langle U_2 \cup \{v\}\rangle)$. Below is a pseudocode description of the RLF algorithm to color a graph $G = \langle V,E,\Phi \rangle$.

<u>Recursive Largest-first Coloring Algorithm</u>

Let $i = 0$.

Let $U = V$.

WHILE $U \neq \phi$

  $i = i + 1$.

  $U_1 = U$.

  Select a vertex $v \in U_1$ such that

  $d(v;\langle U_1 \rangle) = \max \{d(u;\langle U_1 \rangle): u \in U_1\}$.

  Assign v the color i.

  $U_2 = \{u \in U_1$: u is adjacent to v$\}$.

  $U_1 = U_1 - (\{v\} \cup \{u \in U_1$: u is adjacent to v$\})$.

  $U = U - \{v\}$.

WHILE $U_1 \neq \phi$

Let $s = \max \{d(u;\langle U_2 \cup \{u\}\rangle: u \in U_1\}$.

Let $Q = \{u \in U_1: d(u;\langle U_2 \cup \{u\}\rangle) = s\}$.

Select a vertex $v \in Q$ such that

$d(v;\langle U_1\rangle) = \min \{d(u;\langle U_1\rangle): u \in Q\}$.

Assign v the color i.

$U_2 = \{u \in U_1: u$ is adjacent to $v\}$.

$U_1 = U_1 - (\{v\} \cup \{u \in U_1: u$ is adjacent to $v\})$.

$U = U - \{v\}$.

END WHILE

END WHILE

4. <u>Other Heuristic Coloring Algorithms.</u> The previously described heuristic algorithms are by no means an exhaustive list of the heuristic algorithms that have been described in the literature but a list of algorithms that are related to the heuristic algorithms for the CGCP to be discussed. A few other algorithms are described briefly below.

Brelaz [11] described three heuristic algorithms (Dsatur, DSI, Matula-Dsatur) that yielded good experimental results. The Dsatur algorithm selects an uncolored vertex of highest saturation degree to be the next vertex to be colored. If there is more than one such vertex, then one of these vertices with the highest degree in the induced subgraph on the set of uncolored vertices is selected. The saturation degree of a vertex is the number of distinct colors assigned to adjacent vertices. The least possible color is assigned to the selected vertex. The DSI algorithm is obtained by using the interchange technique of Matula, Marble, and Isaacson with the Dsatur algorithm. The Dsatur and SL algorithms may be used to obtain "approximately" maximal cliques of a graph. The Matula-Dsatur algorithm selects the larger of the two cliques to begin the coloring of the graph and completes the coloring of the graph using the Dsatur algorithm.

In [12], Wood described an algorithm that colors pairs of vertices in decreasing order of similarity. If two vertices are not adjacent, then the similarity of the pair of vertices is the number of vertices that are adjacent to both vertices of the pair. Otherwise, the similarity is 0.

In [13], Dutton and Brigham described an algorithm that determines a coloring of a graph by merging nonadjacent vertices until a complete graph is formed.

Schneider [14] presented a classification scheme for heuristic coloring algorithms for the GCP and the results of an experimental study of twenty-three heuristic algorithms.

B. EXACT COLORING ALGORITHMS FOR THE STANDARD GRAPH COLORING PROBLEM

Two articles [15,16] provide a good survey of the exact coloring algorithms for the GCP. Korman gives a broad survey of exact coloring algorithms while Kubale and Jackowski provide a survey of Brown's algorithm [17] and its variations. The topics discussed by Korman include (1) (0,1) integer programming formulations of the GCP, (2) vertex-sequential coloring algorithms for the GCP, and (3) color-sequential coloring algorithms for the GCP. The GCP can be formulated as a set partitioning problem and as a set covering problem. The vertex-sequential and the color-sequential coloring algorithms use implicit enumeration to find an optimal coloring of a graph. The vertex-sequential coloring algorithms include Brown's algorithm and a dichotomous search algorithm based on a theorem of Zykov. Brown's algorithm first colors the vertices of a graph using a vertex-sequential coloring algorithm (as described previously in Section A of this chapter) and then attempts to improve the resulting coloring by means of backtracking. At each level of its search, the dichotomous search algorithm either coalesces two nonadjacent vertices or joins two nonadjacent vertices by an edge until a complete graph is created. In the color-sequential algorithms at each level of the search, the vertices of a maximal independent set of the current uncolored subgraph are all assigned the same color.

A number of the exact coloring algorithms that have appeared in the literature are variations of Brown's algorithm. A recent article of Kubale and Jackowski [16] presented a generalized implicit enumeration algorithm which serves as a framework for Brown's algorithm and its variations. This article provides an excellent survey of these algorithms and describes corrected versions of two erroneous algorithms [18(p. 70-71), 11] that have appeared in the literature.

C. PROBLEM REDUCTION TECHNIQUES FOR THE STANDARD GRAPH COLORING PROBLEM

A problem reduction technique is a method of changing the problem of finding the chromatic number and an optimal coloring of a graph into the problem of finding the chromatic numbers and optimal colorings of one or more subgraphs of the original graph. The chromatic number and an optimal coloring of the original graph should be easily obtainable from the chromatic numbers and optimal colorings of the subgraphs. Three problem reduction techniques that have appeared in the literature are briefly discussed below.

A rather obvious problem reduction technique is to remove those vertices with relatively low degrees from the graph being colored. The following theorem determines the vertices that can be removed by this technique.

Theorem. Let $G = \langle V, E, \Phi \rangle$ be a finite undirected graph with no loops. Let M be a positive integer. If $d_G(v) < M$ for $v \in V$, then any coloring f of $G - v$ can be extended to a coloring of G in which $f(v) \leq M$.

In a problem in which the chromatic number, $\chi(G)$, is being sought, M could be a known lower bound on $\chi(G)$. In some practical applications (for examples, see [19] and [10]), a coloring that is not optimal may be acceptable. In these cases, M could be the largest acceptable number of colors for the application. A vertex v with $d_G(v) < M$ can be removed from the graph and be assigned a color less than or equal to M after the graph G - v has been colored. Several vertices can be removed from the graph G and upon coloring the remaining subgraph, the removed vertices can be colored in reverse order as they were removed.

A second problem reduction technique [15] results from the following theorem.

Theorem. Let $G = \langle V, E, \Phi \rangle$ be a finite undirected graph with no loops. For each $v \in V$, let $\Gamma(v)$ be the set of vertices that are adjacent to v. If $u \in V$ and $v \in V$ such that $\Gamma(u) \subsetneq \Gamma(v)$, then any coloring f of G - u can be extended to be a coloring of G such that $\mathfrak{X}(f;G - u) = \mathfrak{X}(f;G)$ by defining $f(u) = f(v)$.

By repeated applications of the theorem, several vertices can be removed from the graph and upon coloring the remaining subgraph, the removed vertices can be assigned colors as prescribed by the theorem in reverse order as they were removed.

Korman [15] describes another problem reduction technique in which the problem of finding an optimal coloring of a graph G can be transformed into the problems of finding optimal colorings of two or more subgraphs of G. The colorings of the subgraphs are easily combined to obtain an optimal coloring of G. Use of this problem reduction technique is possible if G contains a clique whose removal divides G into two or more disjoint components.

D. HEURISTIC COLORING ALGORITHMS FOR THE COMPOSITE GRAPH COLORING PROBLEM

1. Vertex-sequential Coloring Algorithms. A vertex-sequential (VS) coloring algorithm for the CGCP arranges the vertices of the composite graph to be colored in some order and then assigns to the vertices in this order the lowest possible sequence of colors. A VS coloring algorithm coloring a composite graph $G = \langle V, E, \Phi, C \rangle$ whose vertices are arranged in the order $v_1, v_2, v_3, \ldots, v_n$ would generate a coloring F for which $F(v_1) = I[1, C(v_1)]$ and, for each $k \in I[2,n]$, $F(v_k) = I[A(v_k), A(v_k) + C(v_k) - 1]$ where $A(v_k) = \min \{i \in \mathbb{Z}^+ : I[i, i + C(v_k) - 1] \cap F(v_j) = \phi$ when v_j is adjacent to v_k for $j \in I[1, k - 1]\}$.

Clementson and Elphick [1] described two VS algorithms, LF1 and LF2, which are generalizations of the LF algorithm for the GCP. The largest-first-by-chromaticity (LF1) algorithm orders the vertices in decreasing chromaticity order and sub-orders vertices with equal chromaticities in decreasing chromatic degree order. The largest-first-by-chromatic-degree (LF2) algorithm orders the vertices in decreasing chromatic degree order and sub-orders vertices with equal chromatic degrees in decreasing chromaticity order.

2. Vertex-sequential-with-interchange Coloring Algorithms. A vertex-sequential-with-interchange (VSI) coloring algorithm for the CGCP uses an interchange technique in a fashion similar to a VSI coloring algorithm for the GCP. After arranging the vertices of a composite graph in some order, a VSI algorithm assigns to each vertex in this order the lowest possible sequence of colors. When the sequence of colors assigned to a vertex contains a color greater than the colors assigned to the previous vertices, the VSI algorithm applies an interchange technique to attempt to reduce the number of colors currently assigned by the algorithm. Unlike the interchange techniques described for the VSI algorithms for the GCP in which several vertices can be involved in the interchange of colors, the interchange technique for the VSI algorithms for the CGCP involves changing the colors of two vertices, the current vertex and a vertex adjacent to the current vertex. The interchange technique searches for a possible recoloring of the current vertex and an adjacent vertex in which the current vertex is assigned some of the colors currently assigned to the adjacent vertex and for which the number of colors used in the coloring is reduced.

Suppose that a composite graph $G = \langle V, E, \Phi, C \rangle$ is being colored by a VSI coloring algorithm and a coloring F is being generated. The vertices of G are arranged in the order $v_1, v_2, v_3, \ldots, v_n$. Suppose the first $k - 1$ vertices have been assigned sequences of colors and v_k is the next vertex to be colored. Let $M = \max \left[\bigcup_{i=1}^{k-1} F(v_i) \right]$, the highest color assigned to the previous vertices. Let $p = \min \{i \in \mathbb{Z}^+ : I[i, i + C(v_k) - 1] \cap F(v_j) = \phi$ when v_j is adjacent to v_k for $j \in I[1, k - 1]\}$. The VSI algorithm assigns the color sequence with initial color p to v_k, that is, $F(v_k) = I[p, p + C(v_k) - 1]$. If $p + C(v_k) - 1 \leq M$, then the algorithm proceeds to color v_{k+1}. Otherwise, the interchange technique is applied to attempt to reduce the number of colors currently being used in the coloring. A description of the interchange technique is given below.

(1) Determine a set P of candidate initial colors for the vertex v_k. A color i is an element of P provided exactly one vertex adjacent to v_k has colors in the sequence $I[i, i + C(v_k) - 1]$. $P = \{i \in I[1, p - 1]$: there is exactly one $j \in I[1, k - 1]$ such that v_j is adjacent to v_k and $I[i, i + C(v_k) - 1] \cap F(v_j) \neq \phi\}$. If $P = \phi$, then no interchange is possible and the algorithm proceeds to color v_{k+1}. Let us assume $P \neq \phi$.

(2) Determine possible recolorings of v_k and an adjacent vertex for each $i \in P$. For each $i \in P$, define $J(i) \in I[1, k - 1]$ where $v_{J(i)}$ is the vertex that is adjacent to v_k and $I[i, i + C(v_k) - 1] \cap F(v_{J(i)}) \neq \phi$. For each $i \in P$, define Q(i) to be the lowest permissible

initial color for $v_{J(i)}$ if v_k were assigned the color sequence $I[i, i + C(v_k) - 1]$. For each $i \in P$, $Q(i) = \{r \in \mathbb{Z}^+$: $I[r, r + C(v_{J(i)}) - 1] \cap I[i, i + C(v_k) - 1] = \phi$ and $I[r, r + C(v_{J(i)}) - 1] \cap F(v_j) = \phi$ when v_j is adjacent to $v_{J(i)}$ for $j \in I[1, k - 1]\}$. Note that, for any $i \in P$, the vertices v_k and $v_{J(i)}$ could be validly recolored by redefining

$F(v_k) = I[i, i + C(v_k) - 1]$ and

$F(v_{J(i)}) = I[Q(i), Q(i) + C(v_{J(i)}) - 1]$.

(3) Choose, if possible, a recoloring that reduces the number of colors being used for the coloring. Define $R = \{i \in P: Q(i) + C(v_{J(i)}) < p + C(v_k)\}$. If $R = \phi$, then any recoloring as described above will not reduce the number of colors currently used in the coloring. So, no interchange is possible and the VSI algorithm proceeds to color v_{k+1}. Let us assume $R \neq \phi$. Choose $i^* \in R$ such that $\max \{i^* + C(v_k), Q(i^*) + C(v_{J(i^*)})\} = \min_{i \in R} \max \{i + C(v_k), Q(i) + C(v_{J(i)})\}$. Recolor v_k and $v_{J(i^*)}$ by redefining

$F(v_k) = I[i^*, i^* + C(v_k) - 1]$ and

$F(v_{J(i^*)}) = I[Q(i^*), Q(i^*) + C(v_{J(i^*)}) - 1]$. The interchange technique is completed and the VSI algorithm proceeds to color v_{k+1}.

The LF1I and the LF2I algorithms are the VSI algorithms corresponding to the LF1 and the LF2 VS algorithms, respectively.

3. Experimental Results. Clementson and Elphick compared the four algorithms on a series of random composite graphs. The random composite graphs had 100 vertices and edge densities $\mu = 0.2, 0.3, 0.4$. For a random composite graph having n vertices, each of the possible $\frac{n(n - 1)}{2}$ edges has probability μ of being placed in the graph. The chromaticities of the vertices were distributed according to the truncated Poisson distribution with parameter $\lambda = 1$. The probability that vertex v has chromaticity k is given by

$$P(C(v) = k) = \frac{\lambda^k}{(e^\lambda - 1)(k!)} \quad \text{for } k = 1, 2, 3, \ldots$$

These random composite graphs had a high percentage of vertices with low chromaticities. Approximately 58% of the vertices had chromaticity 1 and approximately 29% had chromaticity 2. The LF1 and the LF1I algorithms yielded colorings superior to those of the LF2 and the LF2I algorithms, respectively. Each of the vertex-sequential-with-interchange algorithms performed better than its corresponding vertex-sequential algorithm. In the results on fifteen random composite graphs documented in the article, the LF1I algorithm produced colorings at least as good as those produced by the other three algorithms. Results for composite graphs with other distributions of chromaticity were not reported.

E. APPLICATIONS OF GRAPH COLORING

1. Applications of the Standard Graph Coloring Problem. Graph coloring has a variety of applications. Schneider [14] listed several areas in which graph coloring has applications: "code design, circuit troubleshooting, decomposition of Boolean functions and automata, distribution of computer memory, design of multilayer integrated circuits, and quality control of printed paths, object classification, and timetabling." In an application of graph coloring, a graph is used to represent conflicts or incompatibilities between objects or events. This graph (its adjacency matrix) is referred to in the literature as a conflict graph (matrix), an interference graph (matrix), or an incompatibility graph (matrix).

As an example, let us consider one of the more commonly mentioned applications of graph coloring, examination timetabling. The examination timetabling problem is the problem of finding the minimum number of periods required to schedule final examinations for several classes without scheduling two examinations for a student during the same period. For this application, a vertex represents a final examination for a class, an edge represents that the final examinations for two classes cannot be given during the same period, and the color to be assigned to a vertex represents the period in which the final examination for the class is scheduled.

The GCP is directly applicable to a scheduling (resource allocation) problem in which (1) the tasks to be scheduled use serially reusable resources, (2) the tasks are of equal duration, say one time period, (3) each task requires a specific set of resources (one resource cannot be used in place of another), (4) a resource is assigned to a task for the duration of the task, and (5) the objective is to minimize the number of time periods required to complete the tasks. The examination timetabling problem described above is an example of such a problem in which the students are the "resources" and the final examinations for the classes are the "tasks." In practical applications, a variety of additional constraints are necessary or desirable to be satisfied. Two examples of such constraints are preassignments of some tasks to certain time periods and precedence constraints that require that one task precede another. For examples of various additional constraints, see [7], [10], [20], and [21].

To provide an insight into the variety of practical applications that have been reported on in the literature, six articles are briefly described below. Each article provides a description of a particular practical application of graph coloring.

In [19], Chaitin describes a graph coloring approach to perform register allocation and to make spill decisions in an optimizing compiler.

Ambler and Trawick [22] used a graph coloring algorithm to allocate positions for attributes within the nodes of a Diana graph. Diana is an intermediate language for Ada that specifies a graph representation for an Ada program.

Garey, Johnson, and So [23] describe a method for testing printed circuit boards for possible defects in the form of short circuits. The problem of minimizing the number of tests is the problem of finding an optimal coloring of a special graph, called a line-of-sight graph.

Butler and Matthews [20] describe an application of graph coloring to scheduling of work at a railway depot. The article describes how several constraints were handled in the graph or in the heuristic algorithm to color the graph.

Coleman and Moré [24] use the GCP to attack the problem of minimizing the number of function evaluations needed to estimate a sparse Jacobian matrix by differences. The authors provide results for two sets of "real-world" problems. The authors give the number of vertices and the edge density for each of the sixty-three graphs that result from these problems.

Carter [10] presents a survey of practical applications of examination timetabling algorithms. Many of these algorithms are based on graph coloring. Carter gives statistics (for example, number of examinations, number of students, number of periods available, conflict density) for some of the practical problems to which these algorithms have been applied.

2. Applications of the Composite Graph Coloring Problem. The composite graph coloring problem was introduced to overcome the limitation of the standard graph coloring problem to handle school timetabling problems with multiple period lessons [1]. The CGCP is applicable to scheduling (resource allocation) problems as described for the GCP in which the tasks to be scheduled are allowed to have unequal durations.

The store economy problem can be modeled by the CGCP. The store economy problem involves minimizing the memory required by a program by determining which variables can occupy the same locations in store [1].

F. RELATED PROBLEMS

In our discussion of graph coloring, we have restricted the discussion to vertex colorings. In the literature, two other forms of colorings of a graph have been presented: edge coloring [25] in which the edges of a graph are assigned colors and total coloring [26] in which both the vertices and the edges of a graph are assigned colors. The problems of finding an edge coloring and a total coloring of a graph can be easily transformed to an equivalent problem of finding a vertex coloring.

The composite graph which we have defined is a vertex-composite graph as defined by Clementson and Elphick [1]. Clementson and Elphick also define an edge-composite graph and describe a corresponding edge coloring problem.

In a survey of developments in deterministic sequencing and scheduling, Lawler, Lenstra, and Rinnooy Kan [27] describe a disjunctive graph model of the general job shop problem. The GCP and the CGCP can be modeled by the disjunctive graph model. The disjunctive graph can be obtained from the graph to be colored by: (1) adding two new vertices, a source and a sink, with weights 0, (2) inserting directed edges from the source to each vertex of the original graph, (3) inserting directed edges from each vertex of the original graph to the sink, (4) assigning a weight equal to the chromaticity of the vertex to each vertex of the original graph, and (5) replacing each edge of the original graph by a pair of directed edges in opposite directions. An acyclic directed subgraph of the disjunctive graph in which one directed edge of each pair of oppositely directed edges is selected corresponds to a coloring that could be produced by a VS algorithm. The number of colors used by the coloring is the weight of the maximum weight path from the source to the sink in the acyclic directed subgraph.

pp 37

IV. RECURSIVE LARGEST-FIRST ALGORITHMS

Two color-sequential coloring algorithms, RLF1 and RLFD1, for the CGCP are presented. The concept of a color-sequential coloring algorithm for the CGCP is a generalization of the concept of a color-sequential coloring algorithm for the GCP. In a color-sequential coloring algorithm for the CGCP, the current color being assigned by the algorithm is the initial color of the color sequences being assigned to the vertices. Before describing the recursive largest-first algorithms, RLF1 and RLFD1, color-sequential algorithms for the CGCP are described more fully and an upper bound on the chromatic number of a composite graph resulting from the color-sequential algorithms is presented.

A. COLOR-SEQUENTIAL COLORING ALGORITHMS FOR THE COMPOSITE GRAPH COLORING PROBLEM

Color-sequential algorithms to color a composite graph assign color sequences to the vertices in a manner such that all vertices that are to be assigned color sequences with initial colors less than a color k are assigned their color sequences prior to the vertices that are to be assigned color sequences with initial color k. Starting with the color 1 as the current color, a color-sequential algorithm assigns color sequences with the current color as the initial color to as many vertices as possible before proceeding to assign color sequences with the next possible initial color. There Upon completion of assigning color sequences with initial color k to vertices of the composite graph, each remaining uncolored vertex is adjacent to at least one vertex that has been assigned the color j for each $j \in I[1,k]$. Below is a pseudocode description of a color-sequential algorithm to color a composite graph $G = \langle V,E,\Phi,C \rangle$. This description can serve as a framework for several algorithms which differ in the manner in which the next vertex to be colored is selected. The recursive largest-first algorithms, RLF1 and RLFD1, to be described are two such algorithms. In the color-sequential algorithms, a quantity LB(v) is introduced for each vertex $v \in V$ to indicate the lowest possible color that can be assigned to the vertex v. In the algorithm description, U is the set of all uncolored vertices, i is the current color which is the initial color of the color sequences being assigned by the algorithm, and U_1 is the set of all uncolored vertices that are not adjacent to a vertex that has been assigned the current color.

<u>Color-sequential Coloring Algorithm</u>

For each $v \in V$, let LB(v) = 1.

Let i = 1.

Let U = V.

Let U_1 = U.

```
WHILE U ≠ φ
  WHILE U_1 ≠ φ
    Select a vertex v ∈ U_1 to be colored.
    Assign vertex v the color sequence I[i,i + C(v) - 1].
    For each u ∈ U such that u is adjacent to v, let
    LB(u) = max {LB(u),i + C(v)}.
    U_1 = U_1 - ({v} U {u ∈ U_1: u is adjacent to v}).
    U = U - {v}.
  END WHILE  U_1 ≠ φ
  IF U ≠ φ THEN
    i = min {LB(u): u ∈ U}.
    U_1 = {u ∈ U: LB(u) = i}.
  END IF
END WHILE  U ≠ φ
```

The quantity LB for each vertex is not needed in the color-sequential algorithms for the GCP since each vertex is assigned only one color. After a color k has been assigned to as many vertices as possible, all remaining uncolored vertices are available to be colored with color k + 1. In the color-sequential algorithms for the CGCP, the quantity LB for each vertex is used to determine the next initial color to be assigned and to determine those vertices that can be assigned color sequences with that initial color. After color sequences with an initial color k have been assigned to as many vertices as possible, some (possibly all) of the remaining uncolored vertices may not be able to be assigned color sequences with initial color k + 1.

B. AN UPPER BOUND ON THE CHROMATIC NUMBER OF A COMPOSITE GRAPH

An obvious upper bound on the chromatic number of a chromatic graph $G = \langle V, E, \Phi, C \rangle$ is the sum of the chromaticities of the vertices of G, $\sum_{v \in V} C(v)$. The color-sequential algorithms provide a tighter upper bound on the chromatic number. From the description of a color-sequential coloring algorithm, note that, for each uncolored vertex v and for each $j \in I[1, LB(v) - 1]$, there is a vertex that is adjacent to v and has been assigned the color j. Using this fact, it can be shown that the maximum chromatic degree of a vertex, $\max \{\Delta(v) : v \in V\}$, is an upper bound on the chromatic number of a composite graph G.

Consider a coloring F of a composite graph $G = \langle V, E, \Phi, C \rangle$ generated by a color-sequential coloring algorithm. Let v^* be any vertex of G such that $\max F(v^*) = \mathfrak{X}(F;G)$. Let $m = \min F(v^*)$. For each $j \in I[1, m - 1]$, there is a vertex adjacent to v^* that has been assigned the color j. Hence,

$$\bigcup_{v \in \Gamma(v^*)} F(v) = I[1, m - 1].$$

$$m - 1 = \left| \bigcup_{v \in \Gamma(v^*)} F(v) \right| \leq \sum_{v \in \Gamma(v^*)} |F(v)| = \Delta(v^*) - C(v^*)$$

where $\Gamma(v^*) = \{u \in V : u \text{ is adjacent to } v^*\}$.

$$\chi(G) \le \chi(F;G) = m + C(v^*) - 1$$
$$\le (\Delta(v^*) - C(v^*) + 1) + C(v^*) - 1 = \Delta(v^*)$$
$$\le \max\ \{\Delta(v)\colon v \in V\}.$$

So, $\chi(G) \le \max\ \{\Delta(v)\colon v \in V\}$. The upper bound, $\max\ \{\Delta(v)\colon v \in V\}$, on the chromatic number, $\chi(G)$, is mainly of theoretical interest because in most cases the bound is not close enough to the chromatic number to be of practical interest.

C. THE RECURSIVE LARGEST-FIRST ALGORITHMS

The RLF1 and the RLFD1 coloring algorithms are generalizations of the RLF coloring algorithm for the GCP. The results of Clementson and Elphick [1] indicate that it is advantageous to give preference to vertices of high chromaticity in a coloring algorithm. The RLF1 and the RLFD1 algorithms do this by interposing an additional criterion for selecting the next vertex to be colored in front of the criteria used by the RLF algorithm. As in the discussion of the RLF algorithm, U denotes the set of all uncolored vertices, U_1 the set of all uncolored vertices that are not adjacent to a vertex that has been assigned the current color, and U_2 the set of all uncolored vertices that are adjacent to a vertex that has been assigned the current color. In the RLF1 and the RLFD1 algorithms, the next vertex to be colored is a vertex from U_1 with the maximum chromaticity. For a composite graph, the chromatic degree of a vertex is a measure of the neighborhood of a vertex as is its degree. The RLF1 algorithm uses the chromatic degree of a vertex in place of the degree of the vertex as used in the RLF algorithm. The RLFD1 algorithm uses the degree of a vertex as used in the RLF algorithm.

For a particular initial color, the first vertex to be assigned a color sequence with that initial color is selected to be a vertex that is "difficult" to color and the remaining vertices to be assigned color sequences with that initial color are chosen to be vertices that are "difficult" to color and are "close" to the vertices that have already been assigned the current color. The "difficulty" to color a vertex is measured by its chromaticity. In the RLF1 algorithm, the "closeness" of a vertex u to the vertices that have been assigned the current color is measured by the sum of the chromaticities of the vertices that are adjacent to the vertex u and also adjacent to vertices that have been assigned the current color. This sum can be written as

CD $C(u;\langle U_2 \cup \{u\}\rangle) - C(u)$ and will be referred to as the U_2 chromatic degree of u. In the RLFD1 algorithm, the "closeness" of Start here a vertex u to the vertices that have been assigned the current integer ~~color~~ is measured by the number of vertices that are adjacent to the vertex u and also adjacent to vertices that have been assigned the current integer ~~color~~. This number can be written as D $d(u;\langle U_2 \cup \{u\}\rangle)$ and will be referred to as the U_2 degree of u.

In the following discussion, two or three criteria are listed for the selection of a vertex to be colored. A higher numbered criterion serves as a tie breaker in the case where more than one vertex satisfies the preceding criteria. If, after all the criteria are applied and more than one vertex remains, any of the remaining vertices can be selected. In the implementations of these algorithms ~~(See Appendix B)~~, the vertex that appeared first in the vertex list is selected.

F1 Coloring algorithm

In the RLF1 coloring algorithm, the first vertex to be assigned consecutive integers ~~a color sequence~~ with the current integer ~~color~~ as its initial integer ~~color~~ is selected from U_1 according to the following criteria:

(1) maximum chromaticity and

(2) maximum chromatic degree in the uncolored subgraph, $\langle U \rangle$.

The remaining vertices to be assigned consecutive integers ~~color sequences~~ with the current integer ~~color~~ as their initial integer ~~color~~ are selected from U_1 according to the following criteria:

(1) maximum chromaticity,

(2) maximum U_2 chromatic degree, and

(3) minimum U_1 chromatic degree (the chromatic degree in the subgraph $\langle U_1 \rangle$).

Below is a pseudocode description of the RLF1 coloring algorithm.

<u>Recursive Largest-first Coloring Algorithm</u>

<u>Using Chromatic Degree - RLF1 Coloring Algorithm</u>

.RS

.IP

For each $v \in V$, let $LB(v) = 1$.

Let $i = 1$.

.RE

Let $U = V$.

Let $U_1 = U$.

Let $U_2 = \phi$.

WHILE $U \neq \phi$

/* Select the first vertex to be assigned the current integer color

i as its initial ~~color~~ integer. */

Let $s = \max \{C(u) : u \in U_1\}$.

Let $Q = \{u \in U_1 : C(u) = s\}$.

Select a vertex $v \in Q$ such that

CD $\Delta(v;\langle U\rangle) = \max$ CD $\{\Delta(u;\langle U\rangle) : u \in Q\}$.

Assign vertex v the ~~color sequence~~ consecutive integers $I[i, i + C(v) - 1]$.

For each $u \in U$ such that u is adjacent to v, let

$LB(u) = \max \{LB(u), i + C(v)\}$.

$U_2 = U_2 \cup \{u \in U_1 : u \text{ is adjacent to } v\}$.

$U_1 = U_1 - (\{v\} \cup \{u \in U_1 : u \text{ is adjacent to } v\})$.

$U = U - \{v\}$.

/* Select the remaining vertices to be assigned the current

~~color~~ integer i as their initial ~~color~~ integer. */

WHILE $U_1 \neq \phi$

Let $s = \max \{C(u) : u \in U_1\}$.

Let $Q = \{u \in U_1 : C(u) = s\}$.

Let $t = \max$ CD $\{\Delta(u;\langle U_2 \cup \{u\}\rangle) - C(u) : u \in Q\}$.

Let $R = \{u \in Q :$ CD $\Delta(u;\langle U_2 \cup \{u\}\rangle) - C(u) = t\}$.

Select a vertex $v \in R$ such that

CD $\Delta(v;\langle U_1\rangle) = \min$ CD $\{\Delta(u;\langle U_1\rangle) : u \in R\}$.

Assign vertex v the ~~color sequence~~ consecutive integer $I[i, i + C(v) - 1]$.

For each $u \in U$ such that u is adjacent to v, let

$LB(u) = \max\{LB(u), i + C(v)\}$.

$U_2 = U_2 \cup \{u \in U_1: u \text{ is adjacent to } v\}$.

$U_1 = U_1 - (\{v\} \cup \{u \in U_1: u \text{ is adjacent to } v\})$.

$U = U - \{v\}$.

END WHILE /* $U_1 \neq \phi$ */

IF $U \neq \phi$ THEN

$i = \min\{LB(u): u \in U\}$.

$U_1 = \{u \in U: LB(u) = i\}$.

$U_2 = U - U_1$.

END IF

END WHILE $U \neq \phi$ /* $U \neq \phi$ */

RLFD1 Coloring Algorithm

The RLFD1 algorithm is very similar to the RLF1 algorithm except that degree is used instead of chromatic degree. In the RLFD1 coloring algorithm, the first vertex to be assigned ~~a color sequence~~ consecutive integers with the current ~~color~~ integer as its initial ~~color~~ integer is selected from U_1 according to the following criteria:

(1) maximum chromaticity and

(2) maximum degree in the uncolored subgraph, $\langle U \rangle$.

The remaining vertices to be assigned ~~color sequences~~ consecutive integers with the current ~~color~~ integer as their initial ~~color~~ integer are selected from U_1 according to the following criteria:

(1) maximum chromaticity,

(2) maximum U_2 degree, and

(3) minimum U_1 degree (the degree in the subgraph $\langle U_1 \rangle$).

(stop here)

Below is a pseudocode description of the RLFD1 coloring algorithm.

<u>Recursive Largest-first Coloring Algorithm</u>

<u>Using Degree - RLFD1 Coloring Algorithm</u>

For each $v \in V$, let $LB(v) = 1$.

Let $i = 1$.

Let $U = V$.

Let $U_1 = U$.

Let $U_2 = \phi$.

WHILE $U \neq \phi$

/* Select the first vertex to be assigned the current color i as its initial color. */

  Let $s = \max \{C(u)\colon u \in U_1\}$.

  Let $Q = \{u \in U_1\colon C(u) = s\}$.

  Select a vertex $v \in Q$ such that

  $d(v;\langle U\rangle) = \max \{d(u;\langle U\rangle)\colon u \in Q\}$.

  Assign vertex v the color sequence $I[i, i + C(v) - 1]$.

  For each $u \in U$ such that u is adjacent to v, let

  $LB(u) = \max \{LB(u), i + C(v)\}$.

  $U_2 = U_2 \cup \{u \in U_1\colon u \text{ is adjacent to } v\}$.

  $U_1 = U_1 - (\{v\} \cup \{u \in U_1\colon u \text{ is adjacent to } v\})$.

  $U = U - \{v\}$.

/* Select the remaining vertices to be assigned the current color i as their initial color. */

  WHILE $U_1 \neq \phi$

    Let $s = \max \{C(u)\colon u \in U_1\}$.

Let $Q = \{u \in U_1 : C(u) = s\}$.

Let $t = \max \{d(u;\langle U_2 \cup \{u\}\rangle) : u \in Q\}$.

Let $R = \{u \in Q : d(u;\langle U_2 \cup \{u\}\rangle) = t\}$.

Select a vertex $v \in R$ such that

$d(v;\langle U_1\rangle) = \min \{d(u;\langle U_1\rangle) : u \in R\}$.

Assign vertex v the color sequence $I[i, i + C(v) - 1]$.

For each $u \in U$ such that u is adjacent to v, let

$LB(u) = \max \{LB(u), i + C(v)\}$.

$U_2 = U_2 \cup \{u \in U_1 : u \text{ is adjacent to } v\}$.

$U_1 = U_1 - (\{v\} \cup \{u \in U_1 : u \text{ is adjacent to } v\})$.

$U = U - \{v\}$.

END WHILE

IF $U \neq \phi$ THEN

$i = \min \{LB(u) : u \in U\}$.

$U_1 = \{u \in U : LB(u) = i\}$.

$U_2 = U - U_1$.

END IF

END WHILE

V. PIGEONHOLE MEASURES

When coloring a composite graph one vertex at a time, some of the vertices remaining to be colored may not require that any new colors be used to color them regardless of how the other uncolored vertices are eventually colored. These vertices can be considered to be "easy" to color and their coloring may be postponed until after the vertices that are more "difficult" to color are colored. The pigeonhole measures to be presented assign low values to vertices that are easy to color as described above and higher values to vertices that are likely to require that new colors be used in the coloring being generated. The word "pigeonhole" was chosen to identify these measures because they are based upon the pigeonhole principle of combinatorics.

The pigeonhole principle is an easily understood principle whose generalizations involve some of the most difficult results of combinatorial theory. Tucker describes the pigeonhole principle in the following manner:

> "If there are more pigeons than pigeonholes, then some pigeonhole must contain two or more pigeons. More generally, if there are more than k times as many pigeons as pigeonholes, then some pigeonhole must contain at least k + 1 pigeons." [28(p. 15)]

The following observation about the average of a list of real numbers is closely related to the pigeonhole principle:

For a list of real numbers, there is at least one number in the list that is at least as large as the average of the list. If $x_1, x_2, x_3, \ldots, x_n$

are real numbers with average $\bar{x}$, then there is an $i \in I[1,n]$ such that $x_i \geq \bar{x}$. In particular, if $x_1, x_2, x_3, \ldots, x_n$ are integers, then there is an $i \in I[1,n]$ such that $x_i \geq [\![\bar{x}]\!]$.

The relationship between the pigeonhole principle and the observation can be seen easily by letting n correspond to the number of pigeonholes and x_j correspond to the number of pigeons in pigeonhole j for each $j \in I[1,n]$.

To discuss how the pigeonhole measures are obtained from the pigeonhole principle, the assumptions to be used need to be set forth. Let us assume a composite graph $G = \langle V,E,\Phi,C\rangle$ is partially colored and U is the set of uncolored vertices. Let M be the highest color that we wish to use in the coloring of G. In other words, if F is the coloring of G that is being generated, we desire $\mathfrak{X}(F;G) \leq M$. This is a goal which may or may not be attained. To continue coloring G, a vertex is to be selected from U to be colored next. For each uncolored vertex $v \in U$, define an adjacent colors bit array, A(v), as follows: for each $j \in I[1,M]$, if a vertex adjacent to v has been assigned the color j, then A(v)[j] = 1; otherwise, A(v)[j] = 0. The term <u>gap</u> will be used to describe a maximal contiguous sequence of zeroes in an adjacent colors bit array. For the following discussion, v is assumed to be any uncolored vertex.

If a gap in A(v) has length greater than or equal to C(v), then some of the colors corresponding to the gap could be used to color the vertex v. But if some vertices adjacent to v are colored prior to v, then the gaps may become smaller and v may possibly no longer be able to be colored with colors less than or equal to M. We desire to find a condition to indicate when a vertex v is guaranteed to be able to be colored with colors less than or equal to M regardless of what colors are eventually assigned to the remaining uncolored adjacent vertices.

If, after the vertices adjacent to v are colored, the average length of a gap is greater than C(v) - 1, then there is a gap of length at least as large as C(v) and consequently the vertex v can be colored with colors less than or equal to M. We will find a lower bound on the average length of the gaps after the adjacent vertices are colored. If this lower bound is greater than C(v) - 1, then v will be able to be colored with colors less than or equal to M regardless of the colors to eventually be assigned to the uncolored adjacent vertices.

Let F be any partial coloring of G in which the vertices in V - U have the same colors assigned by the partial coloring previously described, the vertices adjacent to v have been colored, and v has not been colored. Let B(v) be the adjacent colors bit array for v for the partial coloring F. Define $\bar{g}$ to be the average length of a gap in B(v), that is,

$$\bar{g} = \frac{\text{number of zeroes in } B(v)}{\text{number of gaps in } B(v)} .$$

(If the number of gaps in B(v) is 0, assume $\bar{g} = 0$.) Now, for the use of the pigeonhole principle, consider the zeroes in B(v) as being stored in the gaps in B(v). If there are any gaps in B(v), then there must be a gap with length at least $[\![\bar{g}]\!]$. If $[\![\bar{g}]\!] \geq C(v)$ or equivalently $\bar{g} \geq C(v) - 1$, then v can be colored with colors less than or equal to M. To obtain a lower bound on $\bar{g}$, we find a lower bound on the above fraction.

$$\bar{g} \geq \frac{\text{minimum possible number of zeroes}}{\text{maximum possible number of gaps}} .$$

For each uncolored adjacent vertex that is subsequently colored, a gap in A(v) could become two smaller gaps yielding a possible net gain of one gap per uncolored adjacent vertex. So, the maximum possible number of gaps is the current number of gaps in A(v) plus the number of uncolored adjacent vertices. The minimum possible number of zeroes is M reduced by the number of distinct colors already assigned to vertices adjacent to v and the number of distinct colors that could possibly be assigned to the uncolored vertices adjacent to v. In the worst case, each uncolored adjacent vertex could be assigned colors not previously assigned to other adjacent vertices. In this case, the number of colors used for the presently uncolored adjacent vertices would be the sum of the chromaticities of these vertices. Define the following identifiers for the quantities mentioned above:

UC(v) : the number of distinct colors currently assigned to vertices adjacent to vertex v ("used colors")

RCD(v) : the sum of the chromaticities of the uncolored vertices adjacent to vertex v ("reduced chromatic degree")

NG(v) : the number of gaps in the adjacent colors bit array of vertex v ("number of gaps")

RD(v) : the number of uncolored vertices adjacent to vertex v ("reduced degree")

If $NG(v) + RD(v) \neq 0$, define

$$L(v) = \frac{M - (UC(v) + RCD(v))}{NG(v) + RD(v)} .$$

If $NG(v) + RD(v) = 0$, the colors to be eventually assigned to vertex v must all be greater than M. In this case, define $L(v) = -\infty$. L(v) is a lower bound on $\bar{g}$ that is independent of the colors that are eventually to be assigned to the uncolored vertices adjacent to v. If $L(v) > C(v) - 1$, then $\bar{g} > C(v) - 1$ and $[\![\bar{g}]\!] \geq C(v)$. So, if $L(v) > C(v) - 1$, then the vertex is "easy" to color.

We will refer to the quantity $C(v) - 1 - L(v)$ as the floating-point pigeonhole measure of the vertex v, FPH(v). If $NG(v) + RD(v) \neq 0$, then

$$FPH(v) = \frac{(C(v) - 1)(NG(v) + RD(v)) + (UC(v) + RCD(v)) - M}{NG(v) + RD(v)}$$

Otherwise, $FPH(v) = +\infty$. If $FPH(v) < 0$, the vertex v is "easy" to color. Notice that the sign of FPH(v) is determined by the numerator of the fraction. We will refer to the quantity

$$(C(v) - 1)(NG(v) + RD(v)) + (UC(v) + RCD(v))$$

as the pigeonhole measure of the vertex v, PH(v). If PH(v) < M, then the vertex v is "easy" to color. PH(v) can be interpreted as the highest color that could be assigned to an uncolored vertex adjacent to v while not leaving a gap of length C(v) (taking into account the number of colors already assigned to vertices adjacent to v), that is, all present gaps and potential gaps from coloring the adjacent vertices being of length C(v) - 1.

The following three chapters discuss applications of the pigeonhole measures PH and FPH. Chapter VI discusses a vertex-sequential coloring algorithm and a vertex-sequential-with-interchange coloring algorithm using the pigeonhole measure PH. Chapter VII discusses two coloring algorithms that use PH and FPH in a dynamic fashion to determine the next vertex to color. Chapter VIII discusses a problem reduction technique derived from the pigeonhole measure PH.

VI. LARGEST-FIRST ALGORITHMS

Two VS coloring algorithms for the CGCP and their corresponding VSI algorithms are described. The vertex orderings for these algorithms are not primarily determined by one simple measure such as the chromaticity, the chromatic degree, or the degree of a vertex. For the LF1 vertex ordering, the ordering is primarily determined by the chromaticities of the vertices. For the LF2 vertex ordering, the ordering is primarily determined by the chromatic degrees of the vertices. The orderings for the algorithms to be discussed are compromises for giving preference to vertices of high chromaticity and to vertices of high degree (or chromatic degree). The vertex orderings are determined by ordering the vertices in decreasing order according to a function of two or more of the three measures mentioned above. For each of these functions, the function value increases as the chromaticity increases and as the degree (or chromatic degree) increases.

The largest-first-by-pigeonhole-measure (LFPH) coloring algorithm is a VS coloring algorithm that orders the vertices in decreasing static pigeonhole measure order. The static pigeonhole measure (SPH) of a vertex is the pigeonhole measure of the vertex for the conditions prior to coloring any vertices of the graph. The number of colors to be used in the coloring will be at least 1, so it is reasonable to consider $M \geq 1$. For a vertex v of the composite graph to be colored, the following quantities have the indicated values prior to coloring any vertices: $NG(v) = 1$, $UC(v) = 0$, $RCD(v) = \Delta(v) - C(v)$, and $RD(v) = d(v)$. By substituting these values into the formula

$$PH(v) = UC(v) + RCD(v) + (C(v) - 1)(NG(v) + RD(v)) ,$$

we obtain

$$SPH(v) = \Delta(v) + (C(v) - 1)\ d(v) - 1 .$$

$SPH(v)$ is the maximum number of colors that can be used in a coloring without guaranteeing that a gap of length $C(v)$ appears in the adjacent colors bit array of the vertex v. This number of colors is attained if all colors assigned to the vertices adjacent to v are distinct and the sequences of colors assigned to the adjacent vertices are separated, preceded, and followed by gaps of length $C(v) - 1$. When coloring the composite graph, v can be colored with colors less than or equal to $SPH(v) + 1$.

The largest-first-by-chromaticity-times-degree (LFCD) coloring algorithm is a VS coloring algorithm in which the vertices are arranged in decreasing order according to the product of a vertex's chromaticity and its degree. If the vertices of a composite graph $G = \langle V,E,\Phi,C\rangle$ are arranged in the order v_1, v_2, v_3, ..., v_n by the LFCD algorithm, then $C(v_{i-1})\ d(v_{i-1}) \leq C(v_i)\ d(v_i)$ for each $i \in I[2,n]$. The product of the chromaticity and the degree of a vertex is of interest because it has the properties being sought and also it can possibly be a large portion of the SPH of a vertex.

The LFPH and the LFCD algorithms are generalizations of the LF algorithm for the GCP. This can be easily seen by noting that $SPH(v) = d(v)$ and $C(v)\, d(v) = d(v)$ when $C(v) = 1$.

The largest-first-by-pigeonhole-measure-with-interchange (LFPHI) and the largest-first-by-chromaticity-times-degree-with-interchange (LFCDI) coloring algorithms are the VSI coloring algorithms corresponding to the LFPH and the LFCD algorithms, respectively.

VII. DYNAMIC PIGEONHOLE MEASURE ALGORITHMS

The dynamic-pigeonhole-measure (DYNPH) and the dynamic-floating-point-pigeonhole-measure (DYNFPH) coloring algorithms use the pigeonhole measure PH and the floating-point pigeonhole measure FPH, respectively, to determine the order in which the vertices of a composite graph are colored. For the LFPH and the LFPHI algorithms, the pigeonhole measure PH is calculated once for each vertex and is not modified during the coloring of the graph. Upon coloring a vertex, the dynamic pigeonhole measure algorithms update the pigeonhole measures of the uncolored vertices to reflect the effect of coloring the vertex. In the RLF1 and the RLFD1 algorithms, some of the measures that determine the next vertex to be colored are updated after the coloring of a vertex. The changes in these measures reflect the fact that a vertex has been colored but are not dependent upon the particular colors assigned to that vertex. The dynamic pigeonhole measures are dependent upon the particular vertices that are colored (through the quantities RCD and RD) and upon the particular colors assigned to those vertices (through the quantities UC and NG).

In the DYNPH (DYNFPH) algorithm, a vertex with the largest PH (FPH) is selected to be the next vertex to be colored. After a vertex is colored, the PH (FPH) of each uncolored adjacent vertex is updated. Below is a pseudocode description of the DYNPH coloring algorithm in which V is the vertex set of the composite graph being colored and U is the set of uncolored vertices. (For selecting the initial value of M, see Chapter V. In our implementation, M is initially chosen to be 0 and subsequently increased after the first vertex is colored.)

<u>Dynamic Pigeonhole Measure Coloring Algorithm</u>

```
Select an initial value for M.
Initialize PH(v) for each v ∈ V.
U = V.
WHILE U ≠ ϕ
  Select a vertex v ∈ U such that
  PH(v) = max {PH(u): u ∈ U}.
  Color the vertex v with the lowest possible sequence of
  colors.
  U = U - {v}.
  Update PH(u) for each u ∈ U such that u is adjacent to v.
  IF the highest color assigned to the vertex v is greater
  than M THEN
    Update M.
    Update the PH(u) for each u ∈ U.  (The color range has
    expanded.  A new gap at the end of the color range may
    be created for some vertices.)
  END IF
END WHILE
```

A description of the DYNFPH algorithm can be obtained by replacing the pigeonhole measure PH by the floating-point pigeonhole measure in the description of the DYNPH algorithm.

VIII. PROBLEM REDUCTION TECHNIQUES

For the CGCP, it is desirable to determine those vertices of the composite graph whose coloring will not affect the number of colors used in the coloring of the graph. The coloring of these vertices may be postponed until the other vertices in the graph are colored. Two problem reduction techniques will be discussed here.

The first problem reduction technique results from the following theorem.

<u>Theorem.</u> Let $G = \langle V, E, \Phi, C \rangle$ be a composite graph. For each $v \in V$, let $\Gamma(v)$ be the set of vertices adjacent to the vertex v, that is, $\Gamma(v) = \{u \in V: (u,v) \in \Phi(E)\}$. If $u \in V$ and $v \in V$ such that $C(u) \leq C(v)$ and $\Gamma(u) \subseteq \Gamma(v)$, then $\mathfrak{X}(G) = \mathfrak{X}(G - u)$. Furthermore, if F is a coloring of $G - u$, then F can be extended to be a coloring of G such that $\mathfrak{X}(F;G) = \mathfrak{X}(F;G - u)$.

<u>Proof.</u> Let F be a coloring of $G - u$. To extend F to be a coloring of G, define $F(u) = I[A(v), A(v) + C(u) - 1]$ where $A(v) = \min F(v)$. To show that F is a valid coloring of G, we need to show that $F(u) \cap F(w) = \phi$ for each $w \in \Gamma(u)$. Since $C(u) \leq C(v)$, $F(u) \subseteq F(v)$. Let $w \in \Gamma(u)$. Since $\Gamma(u) \subseteq \Gamma(v)$, it follows that $w \in \Gamma(v)$ and $F(v) \cap F(w) = \phi$. $F(u) \cap F(w) \subseteq F(v) \cap F(w)$. Hence, $F(u) \cap F(w) = \phi$.

To show that $\mathfrak{X}(G) = \mathfrak{X}(G - u)$, assume F is an optimal coloring of $G - u$ and extend F to be a coloring of G as described above. Since $G - u$ is a subgraph of G, $\mathfrak{X}(G - u) \leq \mathfrak{X}(G)$.

$$\mathfrak{X}(G - u) \leq \mathfrak{X}(G) \leq \mathfrak{X}(F;G) = \mathfrak{X}(F;G - u) = \mathfrak{X}(G - u) .$$

Hence, $\mathfrak{X}(G) = \mathfrak{X}(G - u)$.

By the theorem above, if there are vertices u and v such that $C(u) \leq C(v)$ and $\Gamma(u) \subseteq \Gamma(v)$, then the graph $G - u$ can be colored and the coloring of $G - u$ can easily be extended to be a coloring of G without requiring any new colors. Several vertices may be removed from the graph G and upon coloring the remaining subgraph, the removed vertices can be assigned colors in reverse order as they were removed. Coloring a graph by first applying this problem reduction technique can be described recursively as follows.

```
To reduce-and-color a composite graph G:
IF there are vertices u and v of G such that
C(u) ≤ C(v) and Γ(u) ⊆ Γ(v) THEN
  Reduce-and-color the composite graph G - u.
  Assign u the first C(u) colors assigned to v.
ELSE
  Color the composite graph G.
END IF
```

The second problem reduction technique is obtained by means of the pigeonhole principle PH. The problem reduction technique requires a number M that specifies the number of colors that is acceptable to be used in the coloring to be generated. M might be a lower bound on the chromatic number obtained by prior analysis of the composite graph or a number of colors that would be acceptable for a particular application, for example, the number of time periods available for a schedule in a resource allocation problem. Prior to coloring any vertices, the pigeonhole measure of a vertex v simplifies to

$$PH(v) = \Delta(v) + (C(v) - 1)\, d(v) - 1$$

where $\Delta(v)$ is the chromatic degree, $C(v)$ is the chromaticity, and $d(v)$ is the degree of the vertex v. If $PH(v) < M$, then the vertex v will be able to be colored with colors less than or equal to M regardless of what colors are assigned to the other vertices. In this case, the graph $G - v$ can be colored and the coloring can be extended to be a coloring of G by assigning the lowest possible sequence of colors to the vertex v. Several vertices can be removed from a composite graph. After the removal of a vertex, the pigeonhole measures of the vertices that were adjacent to the removed vertex can be updated to reflect the removal of the vertex. After the remaining vertices are colored, the vertices that were removed can be restored to the composite graph and assigned colors in reverse order as they were removed from the graph.

If the coloring algorithm used to color the composite graph does not recolor a vertex once it has been colored, then the problem reduction technique can continue to be used to remove vertices from the graph. The pigeonhole measure

PH of a vertex v is given by the formula

$$PH(v) = UC(v) + RCD(v) + (C(v) - 1)(NG(v) + RD(v))$$

as described in Chapter V. While coloring the composite graph, it is reasonable to have M at least as large as the number of colors currently being used by the coloring and to update M as the number of colors increases. As before, if $PH(v) < M$ for some vertex v, the vertex v can be removed from the graph. A pseudocode description of an algorithm that uses the problem reduction technique prior to and during the coloring of the composite graph is given below.

Pigeonhole Measure Problem Reduction Technique

```
WHILE there is a vertex v in the graph such that PH(v) < M
  Remove the vertex v from the graph.
  Update PH of each vertex that was adjacent to v.
  Place the vertex v on the removed vertex stack.
END WHILE
WHILE some vertex in the graph is uncolored
  Select a vertex to be colored. (The vertex is determined
  by the coloring algorithm being applied to the graph.)
  Color the vertex.
  Update PH for each uncolored vertex adjacent to the
  current vertex.
  IF the highest color assigned to the vertex is greater
  than M THEN
    Update M.
```

```
    Update the PH for each uncolored vertex. (The color
    range has expanded.  A new gap at the end of the color
    range may be created for some vertices.)
  END IF
  WHILE there is an uncolored vertex v in the
    graph such that PH(v) < M
    Remove the vertex v from the graph.
    Update PH of each vertex that was adjacent to v.
    Place the vertex v on the removed vertex stack.
  END WHILE
END WHILE
WHILE removed vertex stack is not empty
  Pop a vertex off the stack, call it v.
  Restore the vertex v to the graph.
  Color the vertex v with the lowest possible sequence of
  colors.
END WHILE
```

The updating of the pigeonhole measures of the vertices can involve a great deal of overhead. If the overhead is too great for a particular application, the overhead can be reduced by using the technique in a fashion that would potentially remove fewer vertices from the graph but is considerably simpler. Instead of updating the pigeonhole measures of adjacent vertices when a vertex is colored or a vertex is removed from the graph, calculate the pigeonhole measure only when the vertex is selected to be colored. If the PH of the selected vertex is less than M, then remove the vertex from the graph and select another vertex to be colored.

Start here

IX. RESULTS AND CONCLUSIONS

A. EXPERIMENTS: COLORING RANDOM COMPOSITE GRAPHS

To assess the performance of the heuristic coloring algorithms for the CGCP (v-composite graph coloring problem) that have been described, the algorithms were applied to several groups of random composite graphs. Each random composite graph is described by three characteristics of the graph: n; the number of vertices, μ; the edge density, and d; the distribution of the chromaticities of the vertices. A random composite graph $G(n,\mu,d)$ is a composite graph of n vertices in which each of the $\frac{n(n-1)}{2}$ possible edges has probability μ of being in the graph and the chromaticities of the n vertices are a random sample of size n from the probability distribution d. The edges of $G(n,\mu,d)$ are selected by $\frac{n(n-1)}{2}$ independent Bernoulli trials, one for each possible edge, in which the probability of success (the edge being included in the graph) is μ. The probability distributions for the chromaticities of the vertices used in the experiments will be described later.

~~1.~~ Goals of the Experiments. The experimental results of Clementson and Elphick [1] were for the four coloring algorithms, LF1, LF2, LF1I, and LF2I, on random composite graphs of 100 vertices with chromaticities distributed according to a truncated Poisson distribution with parameter $\lambda = 1$ and with edge densities $\mu = 0.2, 0.3, 0.4$. In our experiments, ~~N~~ninety groups of random composite graphs were selected to be used,

~~in our experiments~~. Each group consisted of 25 random composite graphs of a type G(n,μ,d). The ordered triple (n,μ,d) can be used to identify the group of random composite graphs of type G(n,μ,d). The ninety ordered triples that were selected were chosen to produce experiments to achieve the following goals:

(1) to corroborate and to expand upon the results of Clementson and Elphick[3],

(2) to investigate the effect of changing the number of vertices of a graph on the performance of the coloring algorithms,

(3) to investigate the effect of changing the edge density of a graph on the performance of the coloring algorithms, and

(4) to investigate the effect of changing the chromaticity distribution of a graph on the performance of the coloring algorithms.

Before proceeding to describe the ordered triples (n,μ,d) that were selected, the chromaticity distributions that were used need to be described.

2. <u>The Chromaticity Distributions.</u> Five probability distributions were chosen to be the distributions of the chromaticities of the vertices in the random composite graphs. These distributions were assigned three-letter identifiers: TRP, DNR, BIN, UNI, UPR.

The TRP distribution is the truncated Poisson distribution with parameter λ = 1. If X is a random variable distributed according to the truncated Poisson distribution with parameter λ, then

$$P(X = k) = \frac{\lambda^k}{(e^\lambda - 1)(k!)} \quad \text{for } k = 1, 2, 3, \ldots$$

Also, if Y is a Poisson random variable with mean λ, then

$$P(X = k) = P(Y = k \mid Y \neq 0) \quad \text{for } k = 1, 2, 3, \ldots$$

omit Top

replace w/ (A)

The DNR distribution is ~~what we have chosen to~~ referred ~~refer~~ to as a "down ramp" distribution. Consider the following probability density function f where

$$f(x) = -\frac{2}{w^2}x + \frac{2}{w} \quad \text{for } 0 < x < w.$$

This pdf is a decreasing linear function on the interval $(0,w)$. We refer to this distribution as the continuous down ramp distribution on $(0,w)$. We now define the discrete down ramp distribution on $I[a,b]$ where $a,b \in \mathbb{Z}$ and $a < b$. Let Y be a random variable distributed according to the continuous down ramp distribution on $(0,w)$ where $w = b - a + 1$. A random variable X is distributed according to the discrete down ramp distribution on $I[a,b]$ provided

$$P(X = k) = P([\![Y]\!] = k - a) \quad \text{for each } k \in I[a,b],$$

where $[\![Y]\!]$ is the greatest integer less than or equal to Y.

Evaluating the probability on the right hand side of the equation yields

$$P(X = k) = \frac{2w - 2k + 1}{w^2} \quad \text{for each } k \in I[a,b].$$

The DNR distribution is the discrete down ramp distribution on $I[1,10]$.

The BIN distribution is a shifted binomial distribution. Let Y be a random variable distributed according to a binomial distribution with parameters n and p. Recall that

$$P(Y = k) = \binom{n}{k} p^k (1 - p)^{n-k} \quad \text{for each } k \in I[0,n].$$

We define the shifted binomial distribution on I[a,b] with parameter p as the binomial distribution with parameters n and p where n = b - a and the distribution has been shifted a units to the right. If X is a random variable distributed according to the shifted binomial distribution on I[a,b] with parameter p, then

$$P(X = k) = P(Y = k - a) \quad \text{for each } k \in I[a,b]$$

where Y is a random variable distributed according to a binomial distribution with parameters n and p where n = b - a. The BIN distribution is the shifted binomial distribution on I[1,10] with parameter p = 0.5.

Start here → The UNI distribution is the uniform distribution on I[1,10]. So, if X is a random variable distributed according to the UNI distribution, then

$$P(X = k) = 0.1 \quad \text{for each } k \in I[1,10].$$

The UPR distribution is what we have chosen to refer to as an "up ramp" distribution. Consider the following probability density function f where

$$f(x) = \frac{2}{w^2} x \quad \text{for } 0 < x < w.$$

This pdf is an increasing linear function on the interval (0,w). We refer to this distribution as the continuous up ramp distribution on (0,w). We now define the discrete up ramp distribution on I[a,b] where $a, b \in \mathbb{Z}$ and $a < b$. Let Y be a random variable distributed according to the continuous up ramp distribution on (0,w) where $w = b - a + 1$. A random variable X is distributed according to the discrete up ramp distribution on I[a,b] provided

$$P(X = k) = P([\![Y]\!] = k - a) \quad \text{for each } k \in I[a,b].$$

Evaluating the probability on the right hand side of the equation yields

$$P(X = k) = \frac{2k - 2a + 1}{w^2} \quad \text{for each } k \in I[a,b].$$

The UPR distribution is the discrete up ramp distribution on I[1,10].

start here

The TRP distribution was chosen as a chromaticity distribution so comparisons could be made to the experimental results of Clementson and Elphick.[3] The other four distributions were chosen to give a variety of distributions of chromaticities taken from the integers from 1 to 10. For the DNR distribution, the lower integers are more probable. For the BIN distribution, the central integers are more probable. For the UPR distribution, the higher integers are more probable. For the UNI distribution, each of the integers is equally probable. Table I contains the probabilities (to three decimal places) of the integers from 1 to 10 for the five distributions as well as the mean, the variance, and the standard deviation for each distribution.

TABLE I

PROBABILITIES FOR THE CHROMATICITY DISTRIBUTIONS

P(X = k)

k	DISTRIBUTION TRP	DNR	BIN	UNI	UPR
1	0.582	0.190	0.002	0.100	0.010
2	0.291	0.170	0.018	0.100	0.030
3	0.097	0.150	0.070	0.100	0.050
4	0.024	0.130	0.164	0.100	0.070
5	0.005	0.110	0.246	0.100	0.090
6	0.001	0.090	0.246	0.100	0.110
7	0.000	0.070	0.164	0.100	0.130
8	0.000	0.050	0.070	0.100	0.150
9	0.000	0.030	0.018	0.100	0.170
10	0.000	0.010	0.002	0.100	0.190
Mean	1.582	3.850	5.500	5.500	7.150
Variance	0.661	5.528	2.250	8.250	5.528
Std. Dev.	0.813	2.351	1.500	2.872	2.351

72

3. The Groups of Random Composite Graphs for the Experiments. Eighteen pairs (n, μ) of a number of vertices and an edge density were selected. For each pair (n, μ), each of the five chromaticity distributions was used to complete the description of a group of random composite graphs. According to Leighton [7], the edge densities of the graphs tend to be small (generally less than 0.25) for most large-scale applications for the non-composite g.c.p. ~~GCP~~. Leighton cited a particular examination timetabling application which resulted in a graph having 273 vertices and an edge density of approximately 0.18. Since similar applications should exist for the v.c.g.c.p. ~~CGCP~~, the edge densities for the experiments were chosen with these observations in mind. The numbers of vertices were chosen to be the multiples of 100 from 100 to 500. For the random composite graphs of 100 vertices, it was decided to perform experiments for all chosen edge densities μ = 0.10, 0.15, 0.20, 0.30, 0.40, 0.50. For the remaining numbers of vertices (n = 200, 300, 400, 500), the experiments were restricted to the edge densities μ = 0.10, 0.15, 0.20. In summary, the ordered triples (n, μ, d) that were selected are all possible triples that satisfy the following conditions:

(1) $n \in \{100, 200, 300, 400, 500\}$,

(2) ~~if~~ n = 100,

~~then~~ $\mu \in \{0.10, 0.15, 0.20\}$; ~~0.30, 0.40, 0.50},~~

~~otherwise, $\mu \in \{0.10, 0.15, 0.20\}$, and~~

(3) $d \in$ {TRP, DNR, BIN, UNI, UPR}.

omit

4. The Experiments and Their Results. Each experiment consisted of coloring each graph of a group of twenty-five random composite graphs by means of the ~~twelve~~ three heuristic coloring algorithms: LF1, ~~LF2~~, ~~LFPH~~, ~~LFCD~~, ~~LF1I~~, ~~LF2I~~, ~~LFPHI~~, ~~LFCDI~~, RLF1, and RLFD1, ~~DYNPH~~, ~~and~~ ~~DYNFPH~~. ~~For each algorithm, the number of colors used for the coloring of each graph was recorded.~~ ~~For each group of graphs, the average number of co~~

~~each algorithm,~~ ~~the minimum and the maximum numbers of colors used in the colorings of the twenty-five graphs were found for each algorithm, and the number of wins for each algorithm was found.~~ The number of wins for an algorithm is the number of times the algorithm used no more colors to color a graph than any of the other algorithms. ~~(A "tie" is considered a win.)~~ Tables II through XIX summarize the results of the ninety experiments. In each table, the column labelled "MIN" corresponds to a thirteenth algorithm that selects a coloring that uses the least number of colors from among the twelve colorings produced by the coloring algorithms. Each entry in Tables II through XIX for a particular group of graphs and a particular algorithm consists of four numeric values. From top to bottom, these numbers are:

describe tables

(1) the minimum number of colors used by the algorithm for a coloring of any of the twenty-five graphs,

(2) the average number of colors used per graph by the algorithm,

(3) the maximum number of colors used by the algorithm for a coloring of any of the twenty-five graphs, and

(4) the number of wins for the algorithm.

TABLE II

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

$n = 100, \mu = 0.10$

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	10	11	10	10	10	11	9	10	9	9	11	10	9
TRP	11.92	13.08	12.04	12.00	11.08	12.08	11.00	10.96	11.24	11.32	11.96	11.80	10.64
	14	16	14	14	13	14	13	13	13	13	13	14	12
	4	0	4	5	15	3	17	17	12	8	5	4	
	28	29	27	28	24	26	24	24	25	25	25	27	24
DNR	31.64	34.92	33.56	32.40	29.32	30.80	28.32	29.24	30.56	30.16	32.72	31.40	27.56
	38	42	40	40	34	37	32	34	36	36	38	38	31
	2	0	0	0	8	1	17	8	2	5	1	2	
	38	39	38	37	32	35	33	33	35	34	38	37	32
BIN	42.76	44.68	43.40	42.20	38.08	38.60	36.64	36.80	39.56	39.52	41.88	40.36	35.56
	49	51	50	50	41	44	41	40	44	45	48	44	38
	0	0	0	0	3	2	11	11	2	3	0	0	
	38	45	39	38	33	38	34	36	37	36	39	38	33
UNI	44.04	50.48	45.28	44.76	39.80	43.16	40.32	40.24	43.44	42.04	44.72	43.20	38.24
	52	58	54	53	47	48	49	45	53	47	56	48	43
	0	0	1	0	9	1	9	7	1	4	1	0	
	49	54	51	48	44	45	44	44	47	46	50	49	44
UPR	55.64	58.92	56.16	55.12	49.36	51.00	49.40	49.04	52.80	52.40	54.60	54.88	47.36
	63	67	64	67	55	59	56	56	60	58	66	64	53
	0	0	0	0	9	2	6	9	1	2	0	0	

TABLE III

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 100, μ = 0.15

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	13	13	12	13	12	12	12	11	12	12	12	12	11
TRP	14.24	16.28	14.56	14.64	13.28	14.88	13.24	13.32	13.68	13.72	14.28	14.08	12.88
	18	20	17	17	16	18	16	15	17	17	16	17	15
	2	0	2	1	15	4	17	15	11	9	3	3	
	33	36	34	33	29	32	28	27	31	30	32	31	27
DNR	39.52	44.96	39.40	38.60	35.56	39.08	34.88	35.20	37.44	37.16	39.24	38.40	33.76
	45	53	44	43	42	46	40	40	42	42	46	43	38
	0	0	0	0	8	1	10	11	3	3	0	0	
	49	50	48	48	43	44	42	42	44	44	47	46	42
BIN	52.92	56.72	52.84	52.52	47.36	49.12	46.44	46.20	49.80	48.88	52.60	52.56	45.44
	56	65	58	59	50	53	52	50	55	55	57	60	50
	0	0	0	0	10	1	11	15	2	4	0	0	
	47	52	47	47	43	44	44	44	44	45	46	47	43
UNI	54.68	61.76	56.08	55.32	49.36	53.20	49.36	49.56	51.40	51.32	54.76	53.56	47.44
	65	76	67	65	59	60	55	57	59	62	60	60	55
	0	0	0	0	9	1	8	7	5	5	0	1	
	62	63	58	61	55	59	54	55	56	57	62	60	54
UPR	70.40	74.56	70.80	68.88	61.96	65.04	62.64	60.76	64.16	64.72	69.32	66.44	59.64
	77	83	80	76	68	73	69	66	73	73	79	78	65
	0	0	0	0	6	2	7	13	3	4	0	0	

TABLE IV

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 100, μ = 0.20

	d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
min ave max # wins	TRP	15 17.24 19 0	16 19.68 24 0	15 16.80 19 1	15 16.92 20 1	14 15.68 18 12	15 17.56 21 0	13 15.52 17 16	14 15.72 18 13	13 15.88 17 9	14 16.04 19 8	14 16.76 19 0	14 16.64 19 2	13 15.08 17
	DNR	40 46.12 53 0	44 52.20 61 0	38 45.72 54 0	39 46.20 55 0	33 42.08 49 7	40 46.24 56 0	34 41.72 49 10	36 41.60 46 10	36 43.48 49 3	37 43.60 50 3	36 45.92 55 0	39 45.76 51 0	33 40.24 46
	BIN	58 63.12 74 0	58 66.40 74 0	54 63.08 75 0	56 62.68 71 0	51 57.08 62 6	54 59.40 65 2	50 56.08 60 9	51 56.00 61 11	53 58.00 65 3	52 57.28 64 4	54 62.04 68 1	56 62.20 72 0	50 54.32 59
	UNI	58 65.36 77 0	57 73.40 85 0	56 65.52 78 0	57 64.84 72 0	54 59.56 67 5	57 64.64 79 0	51 58.44 67 14	53 59.20 66 6	56 61.68 72 3	51 60.16 69 5	56 65.56 74 0	54 63.16 75 1	51 56.88 66
	UPR	74 81.04 89 0	83 91.44 104 0	73 84.32 93 0	71 80.72 91 1	67 74.24 84 7	67 79.72 88 0	67 74.68 84 4	69 73.32 80 10	65 76.08 84 5	65 75.48 87 7	73 81.48 91 0	71 80.20 89 1	65 71.20 75

TABLE V

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 100, μ = 0.30

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	19	21	20	19	17	19	19	17	17	17	19	18	17
TRP	22.20	25.84	22.16	21.96	20.68	22.56	20.40	20.36	20.28	20.52	21.84	21.28	19.72
	25	30	25	24	24	26	23	22	22	23	24	24	22
	0	0	0	1	8	2	13	11	14	11	0	4	
	51	61	50	49	46	51	47	46	47	48	50	49	46
DNR	59.48	68.72	59.92	58.60	53.48	60.76	54.56	54.16	55.32	55.60	58.72	58.72	52.40
	67	79	70	64	58	68	62	58	63	64	65	64	57
	0	0	0	0	13	0	6	9	3	4	0	0	
	76	83	77	75	67	73	66	69	70	70	75	74	66
BIN	83.20	90.44	84.12	82.20	75.44	79.08	75.20	73.92	75.40	75.40	83.28	80.92	72.12
	90	99	90	90	80	87	81	82	81	81	91	90	79
	0	0	0	0	7	1	4	8	7	7	0	0	
	75	83	75	72	68	75	66	69	71	70	75	71	66
UNI	85.72	95.84	84.48	83.28	77.76	85.64	76.40	76.68	79.12	80.68	85.04	82.48	74.72
	103	122	97	94	86	92	90	90	91	98	97	97	86
	0	0	1	0	7	0	10	12	1	2	0	0	
	94	110	98	98	88	94	90	85	89	87	92	97	85
UPR	106.16	119.96	109.04	108.68	97.60	104.56	98.36	97.12	98.44	99.96	106.92	106.40	94.68
	121	132	122	121	107	116	109	108	109	115	120	115	105
	0	0	0	0	5	1	5	9	5	4	0	0	

TABLE VI

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

$n = 100,\ \mu = 0.40$

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	25	26	24	23	22	24	22	23	22	22	24	22	22
TRP	27.76	31.24	27.04	26.92	25.28	28.20	25.20	25.08	25.04	25.00	26.56	26.04	24.32
	31	35	29	30	28	34	28	27	28	28	30	29	27
	0	0	0	0	11	0	10	12	12	12	1	2	
	61	68	62	63	59	64	58	58	56	56	62	59	56
DNR	72.36	84.12	72.96	73.04	68.12	75.68	66.80	66.88	67.76	67.92	72.60	72.24	65.12
	83	97	83	81	75	86	74	73	78	77	83	80	71
	0	0	0	0	1	0	9	8	5	7	1	0	
	94	100	91	91	86	91	88	85	81	81	95	91	81
BIN	101.68	113.36	104.84	101.60	93.12	99.24	94.00	92.96	92.88	93.92	101.88	99.32	90.04
	112	127	113	108	99	107	98	101	101	100	112	111	97
	0	0	0	0	5	0	5	11	9	5	0	0	
	95	103	90	92	86	94	85	86	84	88	91	92	84
UNI	104.84	120.20	105.28	103.92	96.08	105.32	95.80	96.24	99.44	99.12	103.36	105.24	93.60
	124	157	125	120	112	123	113	111	115	118	124	126	111
	0	0	0	0	10	0	7	9	2	2	0	0	
	119	136	120	120	110	122	112	111	111	113	124	118	110
UPR	130.76	148.00	135.36	132.52	120.28	131.00	121.76	122.24	123.12	124.64	133.08	129.92	117.56
	143	171	148	144	131	143	133	134	137	138	142	146	128
	0	0	0	0	8	0	8	5	5	3	0	0	

TABLE VII

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 100, μ = 0.50

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	29	33	29	30	27	29	27	26	26	26	29	27	26
TRP	33.04	37.20	32.68	32.76	30.44	33.88	30.68	30.44	30.52	30.80	32.60	31.32	29.72
	36	42	36	36	33	38	34	33	34	34	36	34	32
	0	0	0	0	11	0	9	12	11	8	0	4	
	80	86	74	74	73	77	71	70	67	70	72	76	67
DNR	89.04	100.44	87.16	88.28	81.20	91.00	80.44	81.76	80.60	80.76	86.16	86.80	78.56
	101	116	97	98	92	102	93	90	88	89	100	99	87
	0	0	0	0	7	0	7	4	10	6	0	0	
	112	124	113	112	107	111	103	105	104	103	111	108	103
BIN	122.48	137.24	125.52	121.24	114.88	118.84	114.44	112.96	111.88	112.60	123.36	119.32	109.48
	137	154	137	132	125	127	122	120	117	119	133	129	117
	0	0	0	0	3	0	4	6	9	7	0	2	
	110	130	115	115	107	115	104	108	104	106	110	109	104
UNI	126.04	144.16	128.40	125.56	117.56	129.00	116.04	116.64	116.72	118.04	124.32	125.32	113.52
	153	159	154	139	139	150	138	136	129	137	143	143	128
	0	0	0	0	1		11	7	7	7	0	0	
	141	167	150	147	137	151	138	136	135	135	140	147	135
UPR	159.24	178.96	164.00	161.92	148.28	161.88	149.44	148.56	149.20	149.28	161.84	158.04	144.08
	176	196	193	179	161	181	161	161	168	172	173	169	157
	0	0	0	0	3	0	4	5	8	6	0	1	

TABLE VIII

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 200, μ = 0.10

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	15	17	14	15	13	15	13	14	14	14	14	15	13
TRP	16.88	20.04	17.12	16.96	15.32	17.92	15.48	15.52	15.80	15.68	16.80	16.36	15.00
	19	24	21	19	18	22	17	18	18	18	20	19	17
	1	0	1	1	17	0	14	14	10	10	2	4	
	44	50	42	41	39	43	38	39	41	41	43	42	38
DNR	48.28	55.56	48.44	47.88	42.88	49.16	42.60	42.80	46.00	45.36	46.92	46.84	41.68
	54	64	56	53	47	56	47	47	52	52	52	52	46
	0	0	0	0	11	0	13	10	3	2	0	1	
	60	64	60	57	53	56	53	50	54	53	59	59	50
BIN	64.64	69.04	65.08	63.56	57.68	60.28	57.16	56.48	58.72	58.36	63.76	62.60	55.20
	70	75	75	67	63	66	63	61	62	65	70	69	59
	0	0	0	0	6	0	7	14	2	4	0	0	
	60	67	62	61	55	57	52	57	56	54	61	61	52
UNI	66.24	75.04	67.20	66.56	61.20	64.96	58.48	60.16	62.28	62.44	66.68	65.64	57.56
	72	82	74	72	68	75	64	67	69	72	75	72	62
	0	0	0	0	1	1	17	5	3	0	0	0	
	76	86	77	77	69	74	70	67	71	71	78	75	67
UPR	81.88	92.72	84.72	83.52	75.00	79.84	74.28	73.52	78.20	77.84	84.04	80.96	72.24
	88	100	92	89	79	89	78	80	88	84	96	87	77
	0	0	0	0	5	1	9	14	2	1	0	0	

TABLE IX

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 200, μ = 0.15

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	19	23	19	20	18	18	18	18	17	17	19	18	17
TRP	21.28	25.16	21.44	21.16	19.60	22.16	19.40	19.84	19.44	19.68	21.12	20.48	18.88
	24	29	24	25	22	25	22	23	22	23	25	23	21
	0	0	0	0	11	0	13	6	13	9	0	0	
	52	61	52	54	48	53	46	48	49	47	50	53	46
DNR	60.52	69.88	60.60	61.00	54.28	62.92	53.88	54.84	57.68	56.84	59.80	59.48	53.12
	69	78	68	67	60	73	60	59	66	63	69	67	58
	0	0	0	0	11	0	15	9	2	1	0	1	
	76	83	77	76	71	75	71	69	71	70	76	75	69
BIN	82.80	91.00	83.12	82.16	74.80	80.24	74.88	74.04	75.48	75.28	81.68	80.24	72.60
	90	101	89	90	78	86	80	79	82	81	89	87	77
	0	0	0	0	5	0	6	13	4	6	0	0	
	78	86	79	78	71	79	69	70	73	72	77	79	69
UNI	84.44	96.88	85.48	83.40	76.80	85.20	75.44	76.40	79.48	80.44	84.92	84.04	74.36
	90	105	92	91	86	90	82	83	90	89	98	90	82
	0	0	0	0	8	0	12	12	2	2	0	0	
	100	109	100	96	89	97	92	90	92	91	100	93	89
UPR	105.32	118.16	108.12	106.04	95.28	105.80	97.60	94.92	97.92	99.92	108.08	102.88	93.56
	112	133	117	114	104	116	102	101	103	110	118	111	98
	0	0	0	0	11	0	3	12	4	2	0	1	

TABLE X

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 200, μ = 0.20

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
TRP	22 25.68 29 0	27 30.44 34 0	23 25.48 28 0	23 25.52 28 0	21 23.56 27 12	24 27.64 32 0	21 23.84 27 7	21 23.88 26 9	21 23.40 26 16	21 23.56 28 15	22 25.36 29 0	21 24.36 28 5	21 23.00 26
DNR	63 71.64 79 0	74 85.16 96 0	63 72.44 81 0	66 72.44 79 0	59 65.64 72 7	68 76.44 82 0	58 65.40 69 12	59 66.12 73 6	59 67.24 76 5	59 67.68 76 6	65 71.72 77 0	63 70.96 79 0	58 64.04 69
BIN	96 100.32 106 0	102 110.60 119 0	97 103.08 110 0	92 100.08 106 0	84 91.12 98 6	91 97.92 106 0	87 91.64 97 7	86 90.92 98 11	84 91.52 101 10	84 91.88 99 8	94 100.56 106 0	92 97.72 102 0	84 88.76 94
UNI	94 100.72 109 0	111 118.52 129 0	94 103.12 112 0	94 101.28 110 0	86 92.20 99 7	96 105.36 115 0	85 92.04 98 11	84 92.20 104 10	85 93.16 99 5	86 93.24 104 3	93 100.76 110 0	91 98.84 107 0	84 89.92 98
UPR	123 128.28 138 0	137 146.96 157 0	123 130.88 144 0	119 129.24 144 0	111 116.16 124 9	122 130.84 141 0	109 117.24 125 6	108 117.20 126 6	109 119.68 127 2	109 117.88 126 7	118 128.96 144 0	116 125.24 133 0	108 114.12 120

TABLE XI

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

$n = 300, \mu = 0.10$

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	20	23	20	20	18	22	18	18	18	18	19	19	18
TRP	21.56	25.92	21.56	21.76	19.76	23.76	19.96	19.76	19.60	19.96	21.16	20.84	19.28
	24	29	23	23	21	26	21	22	21	22	23	23	21
	0	0	0	0	13	0	11	15	18	12	1	1	
	53	61	55	54	48	57	49	48	50	48	54	53	48
DNR	60.20	70.40	60.16	59.68	53.96	62.32	54.28	54.00	56.24	56.12	59.28	58.92	52.80
	69	80	69	66	62	69	60	61	62	62	65	65	57
	0	0	0	0	12	0	10	11	2	3	0	0	
	78	85	78	76	72	75	73	71	69	69	78	75	69
BIN	83.64	91.20	85.60	82.64	75.88	80.28	75.60	74.44	76.76	75.52	83.12	81.80	73.40
	92	97	90	91	80	88	81	78	82	81	89	89	77
	0	0	0	0	6	0	5	13	3	6	0	0	
	78	91	80	77	70	83	74	73	74	72	79	76	70
UNI	85.04	99.08	86.68	84.40	76.92	88.88	77.84	77.84	79.68	80.28	86.32	83.04	75.72
	91	105	94	91	82	96	82	82	87	87	94	88	79
	0	0	0	0	13	0	6	6	6	2	0	0	
	96	114	103	99	90	101	92	91	93	89	102	98	89
UPR	105.64	121.16	109.40	107.36	96.12	106.60	96.96	96.16	98.68	100.08	109.08	104.48	94.36
	113	136	116	115	102	115	102	100	107	109	121	111	99
	0	0	0	0	12	0	4	9	2	3	0	0	

TABLE XII

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 300, μ = 0.15

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	26	31	25	25	24	27	24	24	24	23	26	25	23
TRP	27.84	33.64	27.84	27.72	25.68	30.32	25.92	25.68	25.16	25.24	27.60	26.36	24.88
	30	38	30	30	27	34	28	27	27	27	30	28	26
	0	0	0	0	8	0	7	11	18	16	0	2	
	70	81	69	68	66	73	65	65	65	65	69	69	65
DNR	76.56	91.96	75.84	76.56	70.20	82.32	70.28	69.80	70.20	71.20	76.08	76.48	68.60
	85	109	84	83	78	92	79	76	77	79	83	87	76
	0	0	0	0	3	0	9	8	10	2	0	0	
	100	114	106	102	95	102	96	95	94	92	102	98	92
BIN	107.16	121.20	112.12	108.48	99.48	107.60	100.32	98.92	99.48	98.16	109.48	105.08	97.00
	114	131	120	119	105	114	106	102	108	103	117	113	100
	0	0	0	0	5	0	1	5	8	13	0	1	
	101	120	102	103	95	108	92	91	94	97	101	100	91
UNI	110.28	131.56	113.68	111.24	102.04	115.92	101.36	100.88	102.92	102.12	111.04	108.44	99.36
	116	144	123	121	107	123	106	107	109	107	120	117	104
	0	0	0	0	0	0	7	14	4	10	0	0	
	132	151	134	132	121	130	123	122	119	122	135	128	119
UPR	137.76	159.68	144.16	140.04	126.96	141.12	128.32	127.92	129.00	129.80	142.48	135.72	124.72
	147	176	152	151	133	154	134	134	140	140	150	147	130
	0	0	0	0	9	0	6	7	5	3	0	0	

TABLE XIII

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 300, μ = 0.20

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
TRP	32 34.04 36 0	38 40.84 46 0	32 33.72 35 0	31 34.00 38 0	29 31.84 34 3	33 37.52 41 0	30 31.64 34 4	29 31.68 34 4	29 30.68 32 20	29 30.68 32 20	31 33.32 35 0	29 31.96 34 3	29 30.48 32
DNR	88 93.56 106 0	98 113.92 131 0	84 93.68 101 0	86 93.76 103 0	79 85.68 94 11	93 101.64 114 0	79 85.48 95 11	81 86.68 94 3	79 86.80 98 4	79 86.32 95 7	84 93.48 104 0	84 92.76 102 0	79 83.88 94
BIN	125 133.16 140 0	138 149.36 158 0	128 136.28 145 0	128 133.52 140 0	116 122.12 130 4	129 134.80 146 0	117 122.84 128 1	116 121.76 131 5	112 119.88 126 9	112 119.80 126 12	127 134.32 147 0	121 129.00 136 0	112 118.00 125
UNI	123 133.72 141 0	143 161.08 176 0	126 137.36 148 0	124 135.08 146 0	115 124.24 131 7	133 143.36 152 0	117 124.36 130 4	116 123.60 129 7	117 124.92 136 4	110 123.92 133 9	125 136.12 146 0	124 132.96 142 0	110 121.00 128
UPR	157 168.32 179 0	181 200.16 210 0	168 175.88 189 0	161 170.72 180 0	145 157.04 165 7	168 177.56 186 0	153 159.88 169 1	148 157.92 164 3	144 157.28 169 10	150 157.44 168 6	161 174.92 188 0	159 165.48 173 0	144 154.08 160

TABLE XIV

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 400, μ = 0.10

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	24	28	24	24	22	26	23	23	22	22	24	23	22
TRP	25.92	31.68	25.68	25.96	23.92	28.52	23.80	24.20	23.44	23.68	25.52	24.96	23.12
	28	36	28	28	26	32	26	25	26	26	28	28	25
	0	0	0	0	8	0	10	6	19	12	0	0	
	69	81	66	68	62	71	62	61	62	62	67	68	61
DNR	73.04	86.64	72.76	72.28	66.24	76.72	65.32	65.76	67.08	67.60	72.28	72.08	64.28
	77	93	79	80	72	84	71	69	73	77	77	77	68
	0	0	0	0	6	0	14	8	3	3	0	0	
	94	106	97	96	88	91	88	85	84	87	94	92	84
BIN	99.80	112.12	102.36	100.96	91.40	99.28	91.96	90.72	90.20	90.72	100.68	98.56	88.36
	107	122	110	108	98	106	96	97	97	97	106	105	92
	0	0	0	0	6	0	0	6	11	12	0	0	
	96	112	96	96	85	98	85	87	90	89	96	93	85
UNI	100.48	121.20	102.32	101.12	90.80	106.16	92.00	91.80	95.44	94.08	101.88	98.68	89.56
	106	138	109	107	96	116	96	97	103	103	112	105	93
	0	0	0	0	11	0	5	6	1	2	0	0	
	120	134	126	123	112	123	110	108	109	112	122	118	108
UPR	126.40	149.32	133.16	128.44	115.72	129.36	118.12	115.56	118.04	118.92	129.08	123.92	113.92
	133	162	143	134	121	138	128	124	126	124	138	130	118
	0	0	0	0	12	0	3	10	4	3	0	0	

TABLE XV

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 400, μ = 0.15

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	31	38	32	31	30	34	30	29	29	29	31	30	29
TRP	33.84	41.60	33.88	33.76	31.64	37.80	31.64	31.56	30.48	30.72	33.44	31.92	30.36
	36	45	37	36	34	42	33	33	32	34	37	34	32
	0	0	0	0	4	0	5	5	22	19	0	1	
	87	103	89	88	81	96	82	82	81	79	86	90	79
DNR	94.24	113.56	94.72	94.24	86.72	101.92	86.36	87.08	86.92	86.32	93.92	93.76	83.92
	99	127	103	101	92	114	91	93	92	92	99	98	88
	0	0	0	0	6	0	7	3	6	8	0	0	
	126	139	127	126	117	127	118	118	113	115	130	121	113
BIN	131.92	149.68	136.00	133.36	121.64	132.72	123.12	121.68	119.04	119.08	133.92	129.36	117.52
	139	161	144	143	130	139	130	128	126	126	140	137	123
	0	0	0	0	2	0	1	2	11	10	0	0	
	126	144	129	125	114	130	114	116	116	113	126	121	113
UNI	131.36	158.88	135.08	132.68	121.68	142.40	121.68	121.80	121.64	122.88	132.64	129.12	118.96
	141	172	144	142	131	157	128	130	128	130	140	136	127
	0	0	0	0	5	0	7	5	8	6	0	0	
	161	186	167	161	146	171	147	147	145	147	165	157	145
UPR	167.28	198.76	175.24	169.00	153.68	176.96	156.48	154.28	154.16	156.68	171.64	163.84	151.84
	173	210	187	178	160	188	164	161	163	165	177	172	160
	0	0	0	0	12	0	2	8	8	4	0	0	

TABLE XVI

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

$n = 400,\ \mu = 0.20$

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	39	47	39	39	37	42	37	37	36	36	39	36	36
TRP	41.56	50.76	41.80	41.32	39.40	46.68	39.16	39.24	37.84	37.80	40.96	38.92	37.56
	44	56	45	44	42	52	42	42	41	40	43	43	40
	0	0	0	0	1	0	2	2	19	21	0	4	
	109	130	109	108	101	116	103	102	100	101	109	107	100
DNR	115.76	141.52	115.80	115.16	107.36	127.72	107.00	106.96	105.08	105.28	115.52	114.80	103.72
	122	152	122	123	113	139	112	112	112	114	125	122	109
	0	0	0	0	4	0	3	2	10	10	0	0	
	157	181	161	159	145	160	146	145	140	142	159	150	140
BIN	163.76	187.60	169.00	166.12	150.80	167.00	151.88	151.64	146.60	147.08	165.44	157.40	145.24
	173	197	177	172	159	177	158	159	154	157	175	168	149
	0	0	0	0	1	0	0	3	14	13	0	0	
	153	186	158	156	142	166	144	139	141	144	154	152	139
UNI	160.52	198.64	165.08	161.96	150.76	177.24	150.96	150.36	148.92	149.72	162.68	160.08	146.76
	168	222	175	176	159	187	159	158	159	158	171	168	157
	0	0	0	0	4	0	2	7	12	7	0	0	
	194	228	202	198	185	209	189	186	181	179	202	195	179
UPR	203.28	243.40	213.20	209.28	191.04	219.16	196.64	192.80	191.00	190.36	211.52	202.48	187.48
	213	266	223	219	199	232	204	203	199	200	223	210	194
	0	0	0	0	7	0	0	2	8	11	0	0	

TABLE XVII

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 500, μ = 0.10

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	29	33	28	28	26	31	26	26	26	26	28	27	26
TRP	30.28	36.84	30.12	30.00	27.92	33.60	27.88	27.92	27.12	27.28	29.44	28.60	26.88
	33	42	33	33	30	37	29	29	29	29	32	31	29
	0	0	0	0	6	0	7	4	19	17	0	3	
	79	94	77	78	73	86	72	72	72	71	77	78	71
DNR	83.64	101.24	84.08	83.72	76.64	92.60	76.88	76.48	77.16	77.52	82.80	82.68	74.68
	87	110	92	89	85	102	82	84	86	81	87	91	80
	0	0	0	0	7	0	7	7	6	6	0	0	
	112	122	113	111	101	107	105	103	97	99	111	110	97
BIN	117.20	130.24	118.88	117.44	106.28	116.24	107.64	107.64	104.32	105.40	117.68	114.04	103.24
	125	141	127	122	110	122	112	113	108	111	125	119	107
	0	0	0	0	6	0	2	2	13	8	0	0	
	110	129	110	108	99	116	102	101	102	98	110	108	98
UNI	116.16	140.44	118.92	116.76	107.56	125.16	107.76	107.28	109.52	108.72	117.16	114.20	105.16
	126	153	128	124	114	135	116	116	123	116	127	121	110
	0	0	0	0	8	0	6	8	4	6	0	0	
	139	164	145	142	129	147	132	130	129	128	144	137	128
UPR	145.20	171.88	153.28	147.96	134.40	154.44	138.36	136.24	137.48	136.20	152.40	144.40	132.92
	151	181	168	156	143	164	144	143	145	146	159	153	139
	0	0	0	0	13	0	0	5	3	7	0	0	

TABLE XVIII

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 500, μ = 0.15

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	38	44	37	37	36	42	35	35	34	34	37	35	34
TRP	39.84	48.44	39.52	39.60	37.28	44.56	37.32	37.20	35.56	35.76	39.24	37.20	35.36
	44	53	43	44	40	49	41	40	38	38	42	41	38
	0	0	0	0	0	0	0	0	21	19	0	1	
	102	126	103	103	97	114	97	97	96	94	103	103	94
DNR	109.44	134.92	110.48	109.84	102.08	122.28	101.92	101.84	101.36	100.88	109.96	108.84	99.60
	116	148	117	118	109	130	108	108	110	108	118	117	108
	0	0	0	0	4	0	4	5	13	14	0	0	
	147	163	155	150	138	152	141	139	133	129	151	142	129
BIN	154.08	176.96	160.00	156.48	142.48	156.84	144.52	143.52	138.84	138.68	156.44	149.20	137.44
	158	187	166	162	148	162	149	151	147	147	164	155	144
	0	0	0	0	3	0	0	0	12	13	0	0	
	144	172	149	148	135	159	137	134	132	134	148	141	132
UNI	152.92	188.04	156.68	155.08	142.28	168.20	143.60	142.56	141.96	142.16	154.60	151.04	139.52
	161	204	167	165	151	183	150	150	151	148	164	160	148
	0	0	0	0	9	0	4	5	8	5	0	0	
	184	221	193	190	175	196	179	178	170	174	194	184	170
UPR	192.80	231.52	201.60	197.32	180.12	207.32	184.64	182.20	181.20	180.48	201.04	190.20	177.56
	209	248	212	205	184	216	194	189	193	189	211	200	184
	0	0	0	0	12	0	1	2	6	9	0	0	

TABLE XIX

RESULTS OF COMPOSITE GRAPH COLORING EXPERIMENTS

n = 500, μ = 0.20

d	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH	MIN
	46	56	47	47	43	51	44	44	42	42	46	43	42
TRP	49.12	60.60	49.20	49.28	46.20	55.40	46.24	46.28	44.32	44.36	48.60	46.16	44.12
	53	64	53	53	50	57	49	49	47	47	52	51	47
	0	0	0	0	0	0	0	0	20	19	0	1	
	129	155	127	128	119	139	120	120	116	117	128	127	116
DNR	135.96	166.16	135.96	135.56	126.84	152.24	126.24	126.36	123.96	123.60	135.76	133.72	122.44
	145	183	146	148	134	163	136	133	133	129	145	141	129
	0	0	0	0	2	0	3	1	11	16	0	0	
	183	210	190	180	172	188	176	173	167	166	185	179	166
BIN	188.80	220.60	197.36	192.68	177.96	199.40	181.84	179.04	172.88	170.84	196.00	185.32	170.08
	195	232	204	202	185	211	189	185	179	176	207	193	175
	0	0	0	0	0	0	0	0	9	17	0	0	
	178	218	180	179	167	195	170	168	165	157	180	178	157
UNI	189.72	234.56	192.80	189.88	176.72	210.68	178.16	177.16	174.48	174.08	192.16	187.08	172.20
	202	252	207	199	191	226	188	187	188	193	204	194	182
	0	0	0	0	3	0	1	3	10	12	0	0	
	233	277	240	232	222	248	221	219	215	210	237	228	210
UPR	241.36	293.36	249.44	243.48	227.32	259.88	230.96	227.08	224.44	223.20	247.36	237.16	221.60
	249	306	258	255	237	266	239	237	232	232	258	243	232
	0	0	0	0	1	0	0	4	10	15	0	0	

B. ANALYSIS OF THE RESULTS OF THE EXPERIMENTS

Examining Tables II through XIX, it becomes apparent that some algorithms rather consistently use fewer colors to color a group of graphs than other algorithms. Of course, it should be expected that a VSI algorithm would use no more colors than its corresponding VS algorithm to color a group of graphs. (In our experiments, a VSI algorithm rarely used more colors than its corresponding VS algorithm on a graph. In fact, this occurred in approximately 0.8% of the 9000 possible instances.) Considering the cases in which one algorithm uses no more colors than another algorithm in at least 95% (86 or more) of the experiments, the algorithms can be ranked into three tiers:

Tier 1: LF1I, LFPHI, LFCDI, RLF1, RLFD1;

Tier 2: LF1, LFPH, LFCD, LF2I, DYNPH, DYNFPH; and

Tier 3: LF2.

Each algorithm in Tier 1 used no more colors than each algorithm in Tier 2 in at least 86 experiments. Each algorithm in Tiers 1 and 2 used no more colors than the LF2 algorithm in the 90 experiments. Table XX shows for each pair of coloring algorithms the number of experiments for which the first algorithm of the pair used no more colors than the second algorithm of the pair. It should be noted that within Tier 2 there are pairs of algorithms for which one algorithm could be ranked above the other using the same criterion used to determine the tiers. For example, the DYNPH algorithm used no more colors than the LFPH algorithm

TABLE XX

COMPARISON OF COLORING ALGORITHMS: NUMBER OF EXPERIMENTS

FOR WHICH ALGORITHM A REQUIRED NO MORE COLORS THAN ALGORITHM B

	Algorithm B											
Algorithm A	LF1	LF2	LFPH	LFCD	LF1I	LF2I	LFPHI	LFCDI	RLF1	RLFD1	DYNPH	DYNFPH
LF1	—	90	72	51	0	66	0	0	0	0	49	2
LF2	0	—	0	0	0	0	0	0	0	0	0	0
LFPH	21	90	—	16	0	54	0	0	0	0	4	2
LFCD	41	90	75	—	0	62	0	0	0	0	43	4
LF1I	90	90	90	90	—	90	50	42	55	57	90	87
LF2I	24	90	36	28	0	—	0	0	4	2	29	18
LFPHI	90	90	90	90	43	90	—	34	50	55	90	87
LFCDI	90	90	90	90	53	90	58	—	54	57	90	88
RLF1	90	90	90	90	37	86	40	36	—	51	90	89
RLFD1	90	90	90	90	34	88	36	34	41	—	90	90
DYNPH	41	90	86	48	0	61	0	0	0	0	—	7
DYNFPH	88	90	88	86	3	73	3	3	1	0	84	—

in 86 experiments.

The "raw" numbers as presented in Tables II through XIX do not provide a satisfactory means to observe various trends in the results. In order to more readily visualize various trends in the data, several graphs were plotted using the data in Tables II through XIX. For these graphs, we restricted our consideration to only those algorithms in Tier 1. The graphs were designed to assist in assessing the performance of these algorithms on the random composite graphs as each of the three variables, n, μ, and d, is varied. To arrange the values of the variable d, the chromaticity distributions were ordered in increasing order according to their means and those distributions with equal means were subordered in increasing order according to their variances.

For the graphs, two dependent variables, the number of excess colors used by an algorithm and the number of wins for an algorithm, were chosen to measure the performance of the algorithm. The number of excess colors for an algorithm is the number of colors used for the group of random composite graphs in excess of the number of colors used by the MIN algorithm. Thirty-eight pairs of graphs were plotted and appear in Figures 2 through 77. The first graph of a pair is the graph of the number of excess colors versus one of the independent variables, n, μ, or d. The second graph of a pair is the graph of the number of wins versus the independent variable. Each graph has a curve plotted for each of the algorithms in Tier 1. Below an excess colors graph, the row of numbers labelled "TOTAL COLORS" is the number of colors used by the MIN algorithm for each value of the independent variable.

Figures 2 through 31 are the fifteen pairs of graphs having n as the independent variable. There is a pair of graphs for each ordered pair (μ, d) such that $\mu \in \{0.10, 0.15, 0.20\}$ and $d \in \{TRP, DNR, BIN, UNI, UPR\}$. Figures 32 through 41 are the five pairs of graphs having μ as the independent variable. There is a pair of graphs for each ordered pair (n, d) such that $n = 100$ and $d \in \{TRP, DNR, BIN, UNI, UPR\}$. Figures 42 through 77 are the eighteen pairs of graphs having d as the independent variable. There is a pair of graphs for each ordered pair (n, μ) such that

$n = 100$ and $\mu \in \{0.10, 0.15, 0.20, 0.30, 0.40, 0.50\}$, or

$n \in \{200, 300, 400, 500\}$ and $\mu \in \{0.10, 0.15, 0.20\}$.

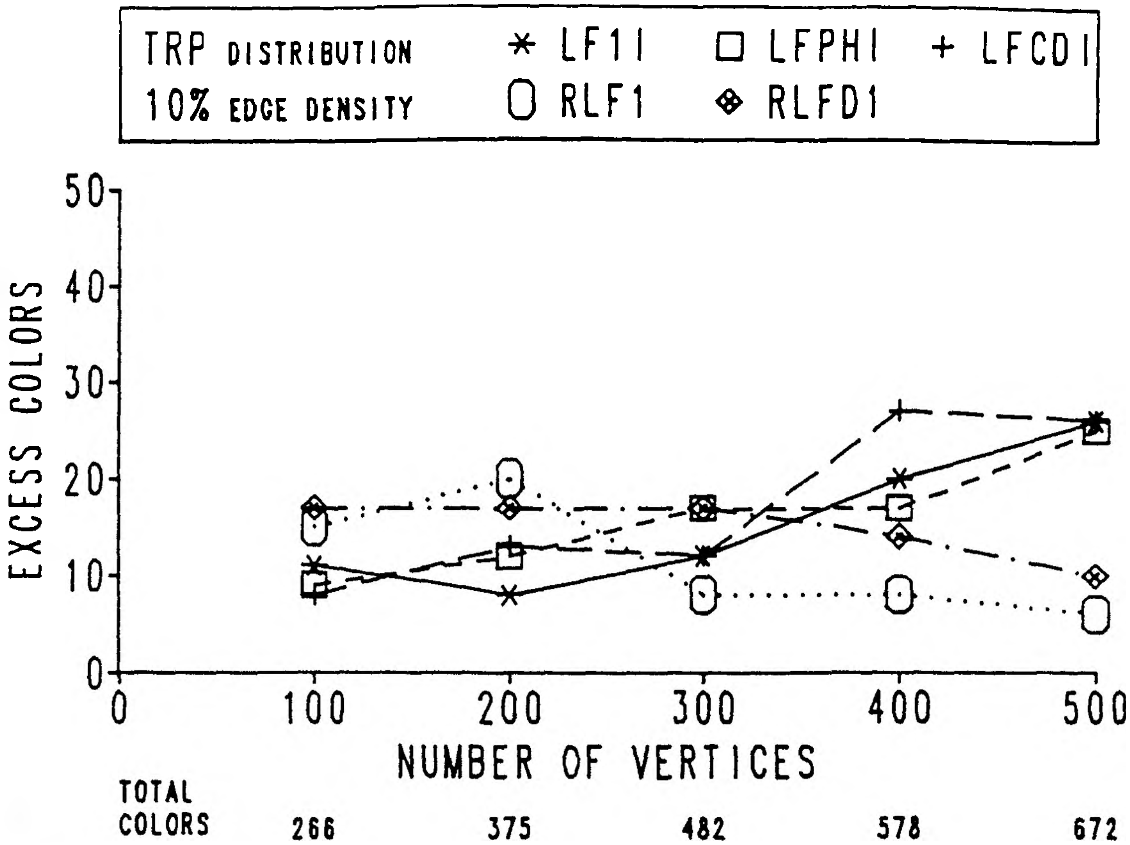


Figure 2. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.10 and d = TRP

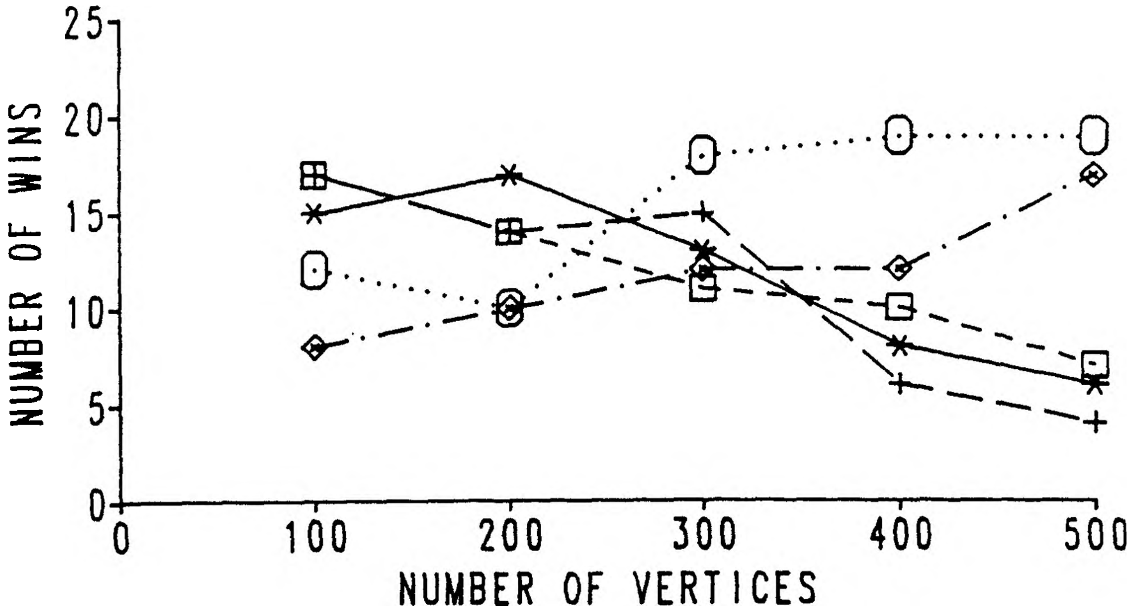


Figure 3. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.10 and d = TRP

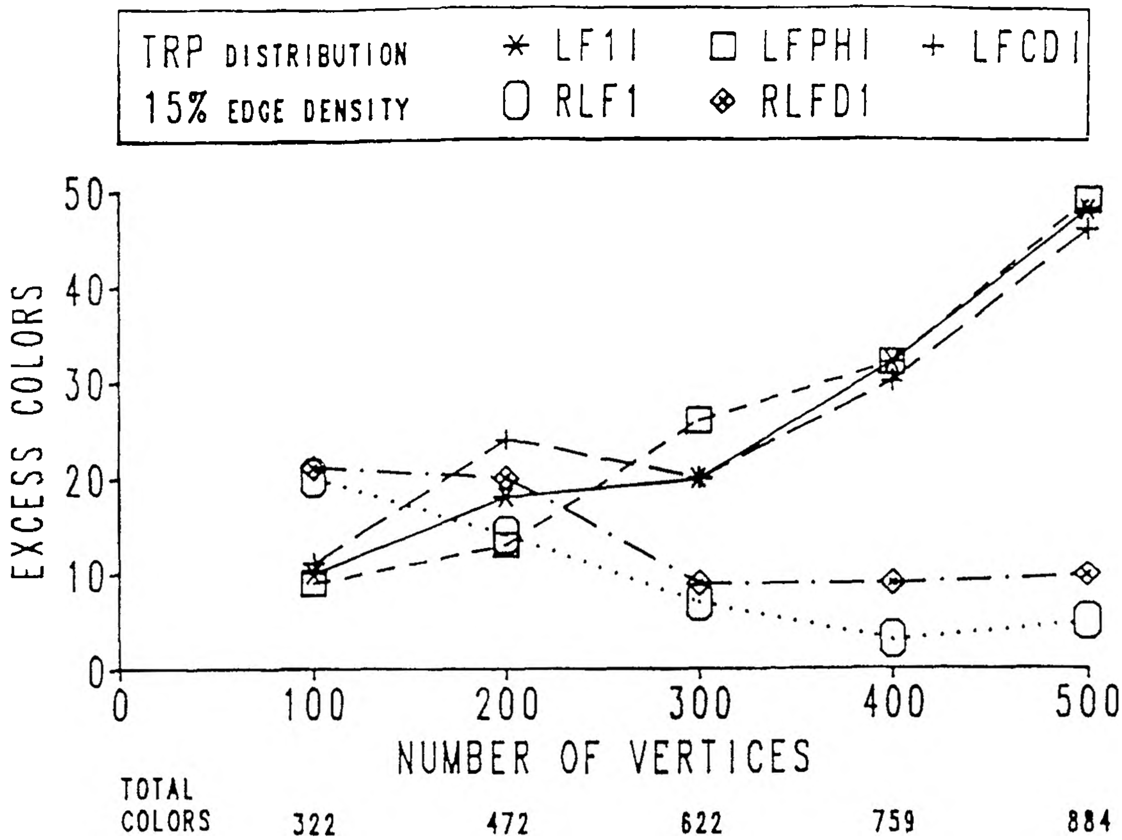


Figure 4. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.15 and d = TRP

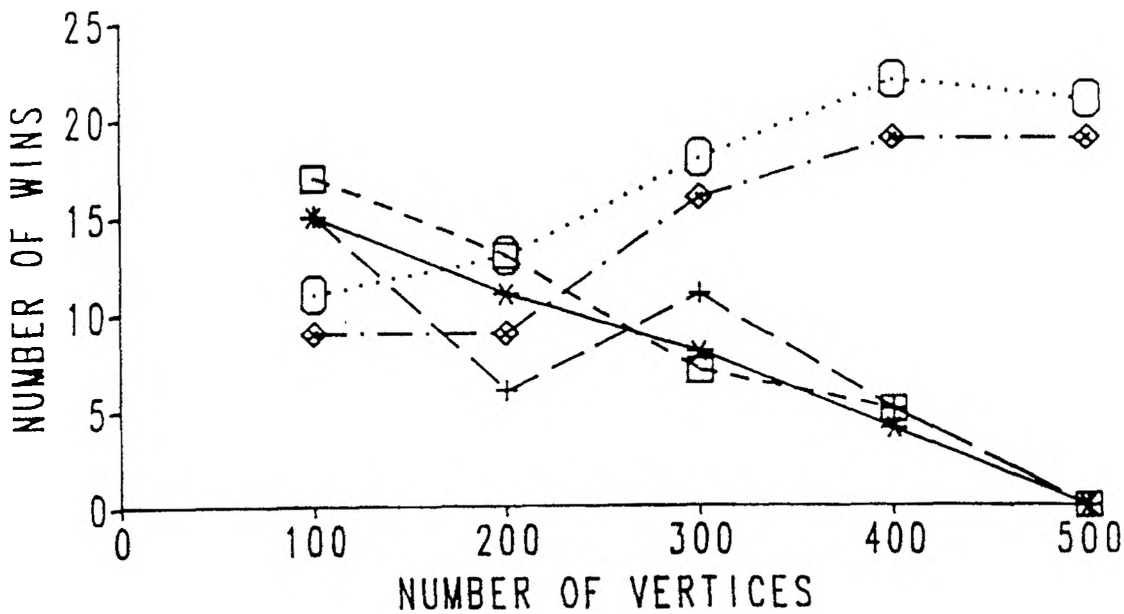


Figure 5. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.15 and d = TRP

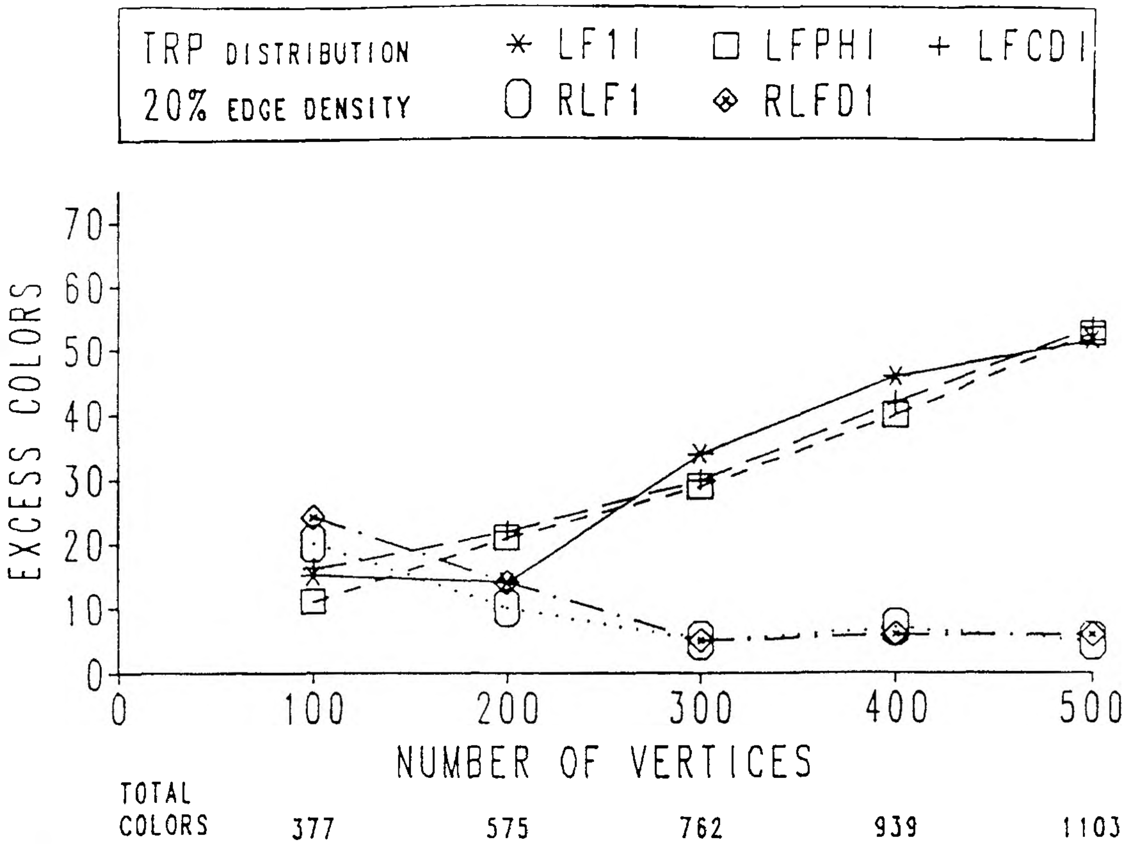


Figure 6. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.20 and d = TRP

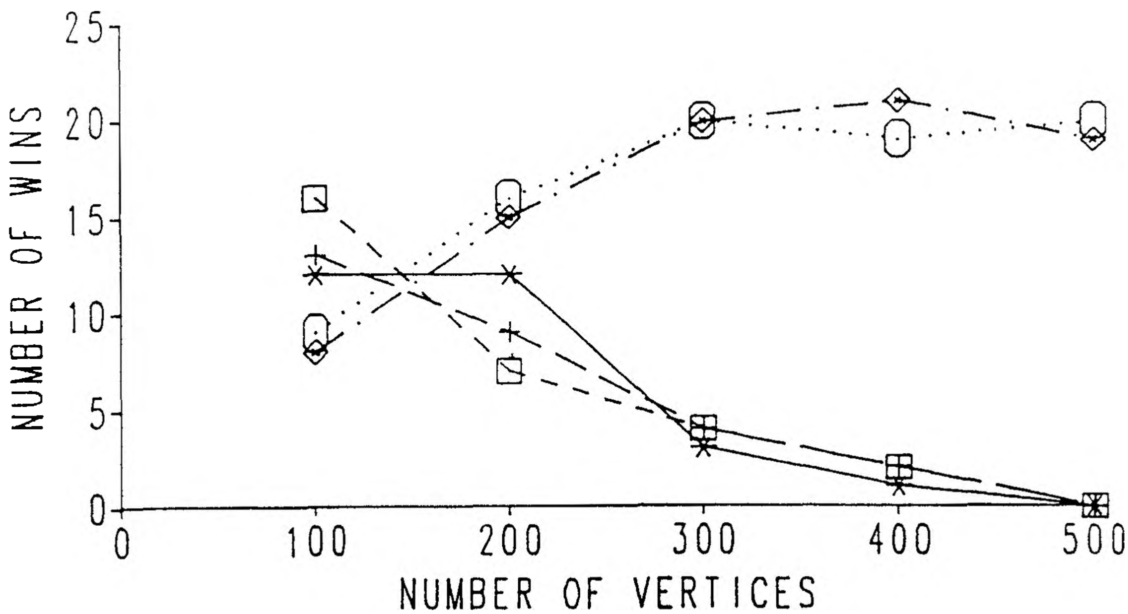


Figure 7. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.20 and d = TRP

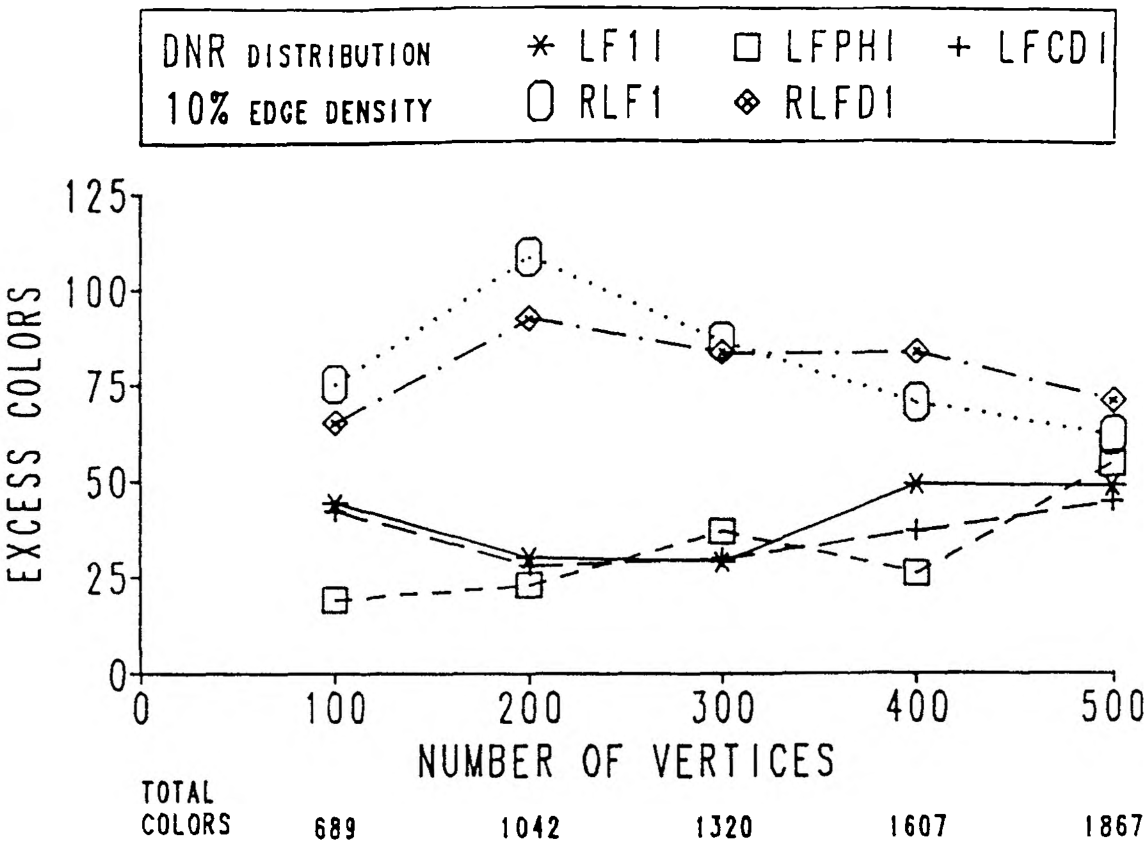


Figure 8. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.10 and d = DNR

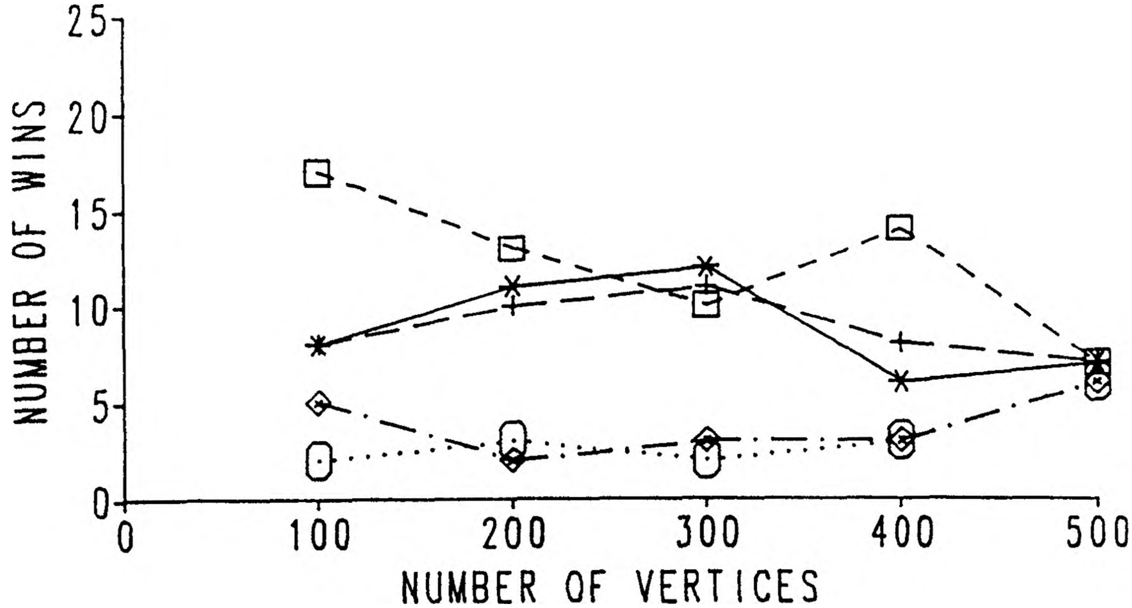


Figure 9. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.10 and d = DNR

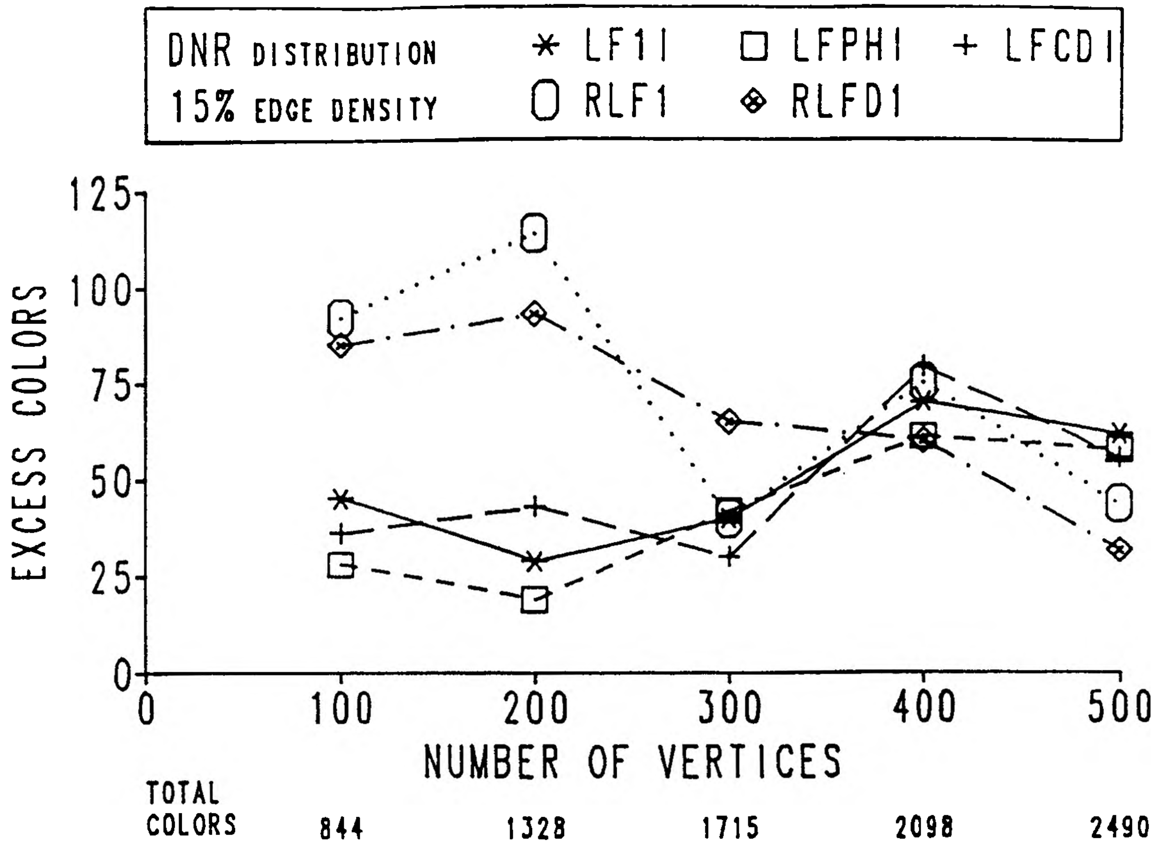


Figure 10. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.15 and d = DNR

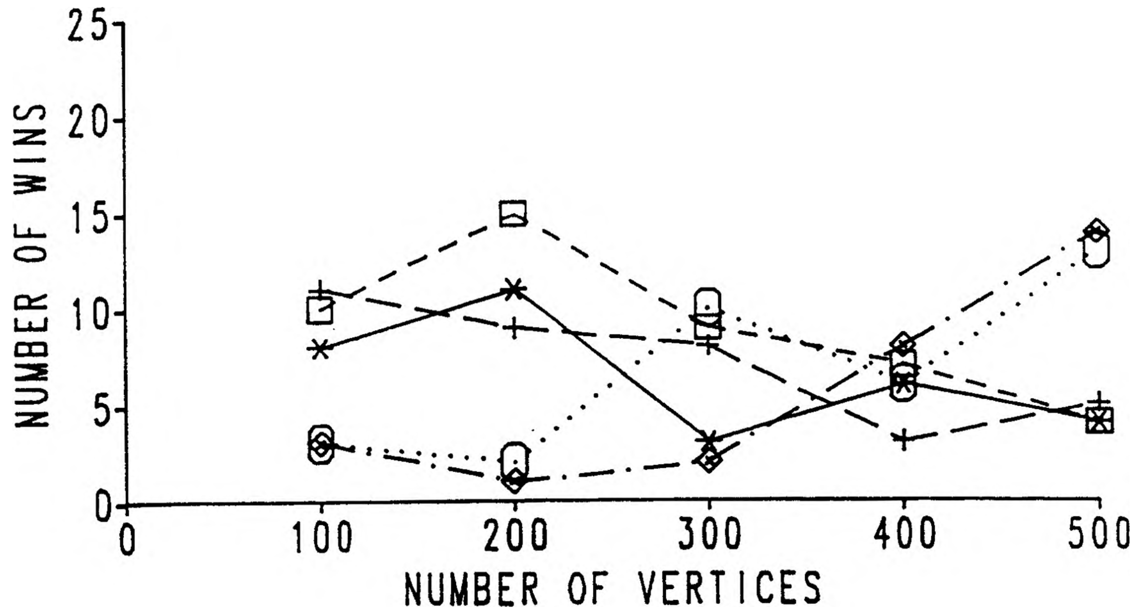


Figure 11. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.15 and d = DNR

BILLG5

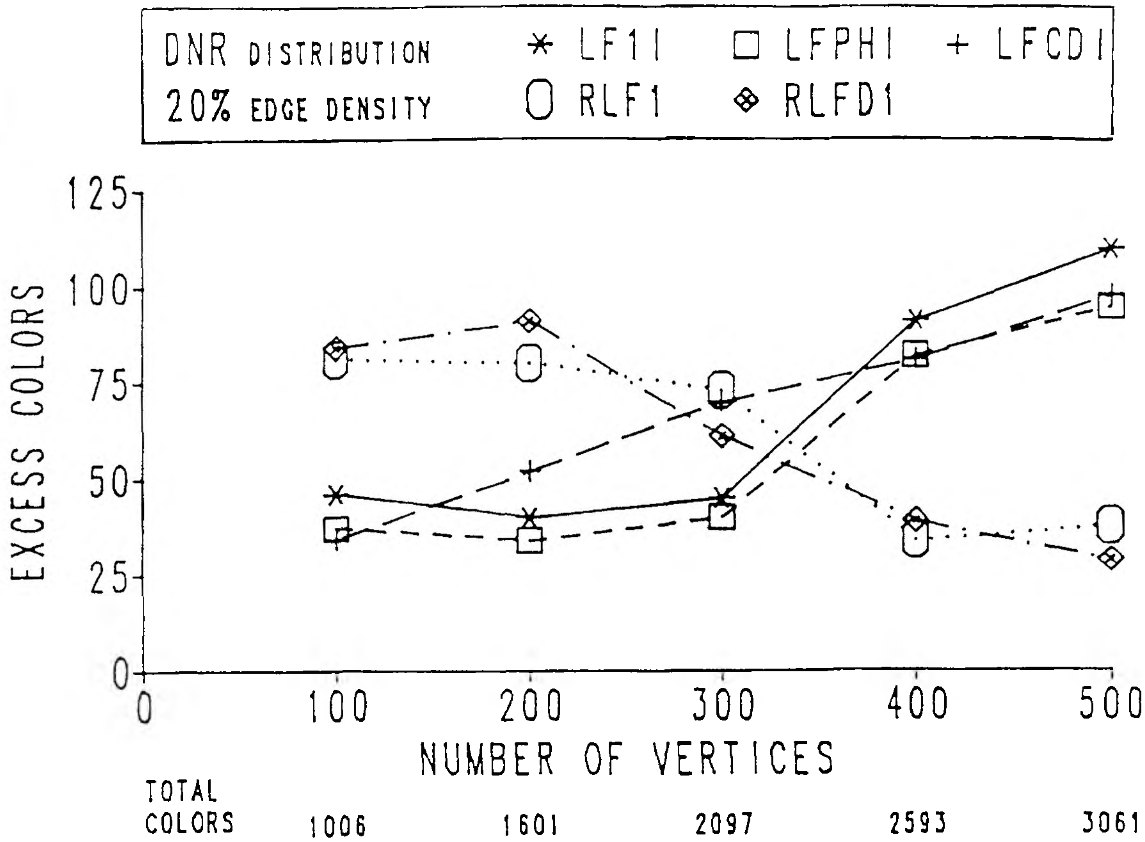


Figure 12. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.20 and d = DNR

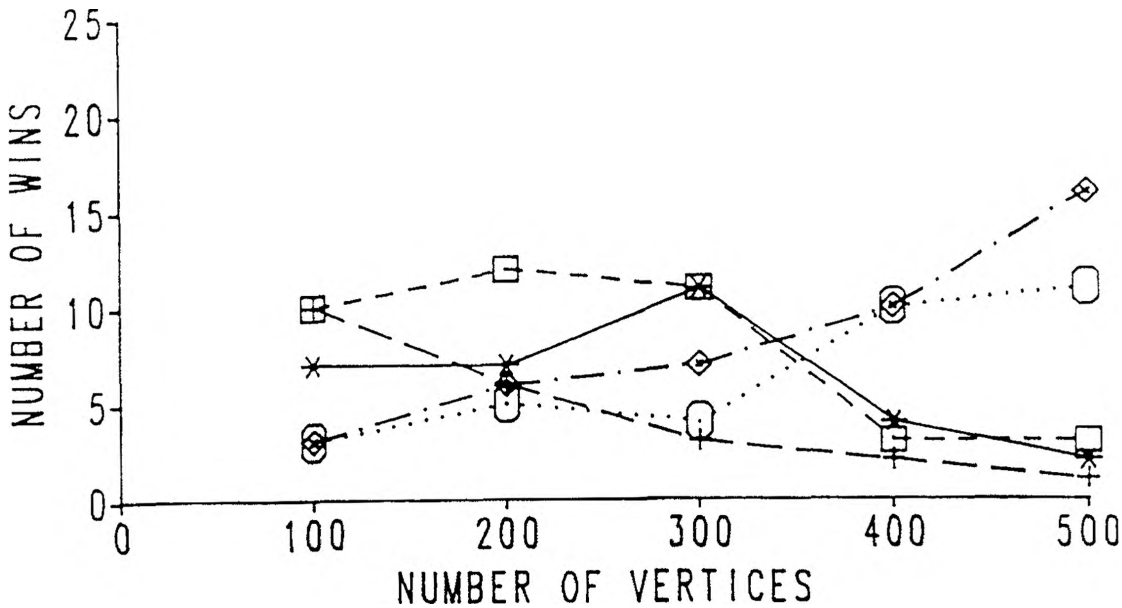


Figure 13. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.20 and d = DNR

BIRG-6

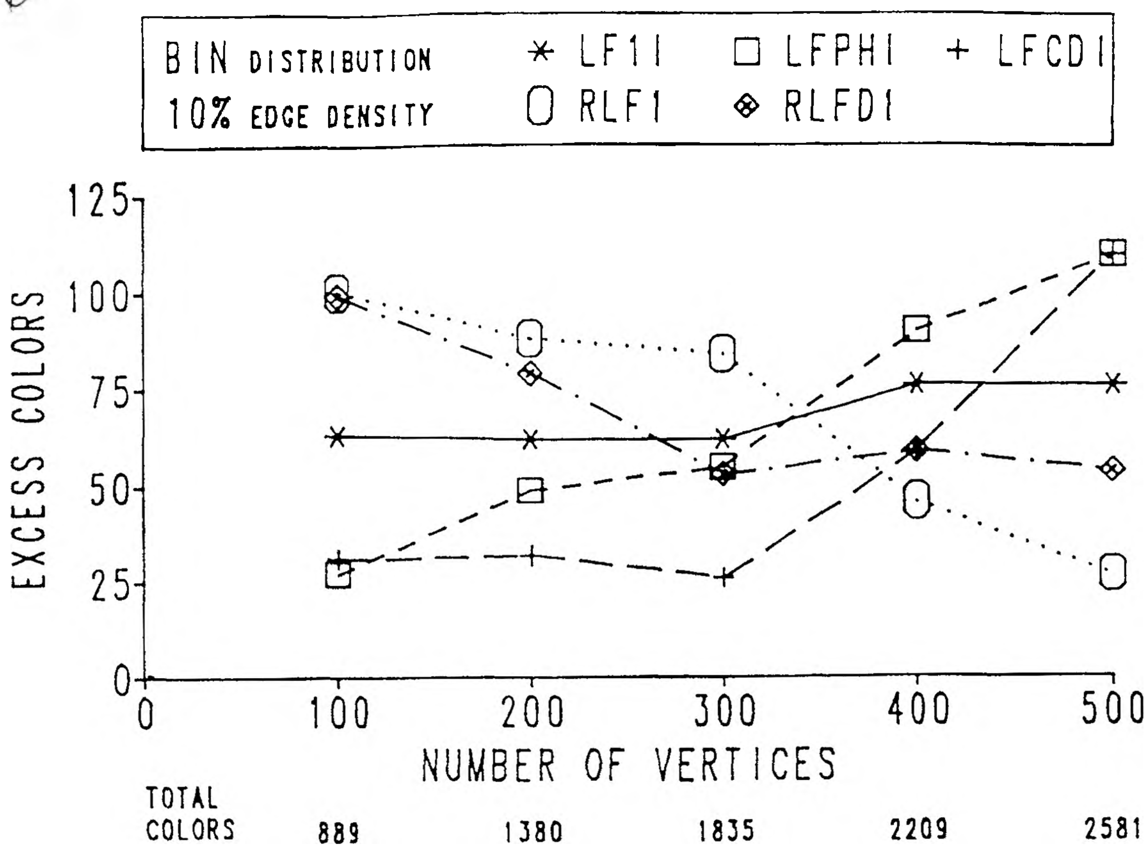


Figure 14. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.10 and d = BIN

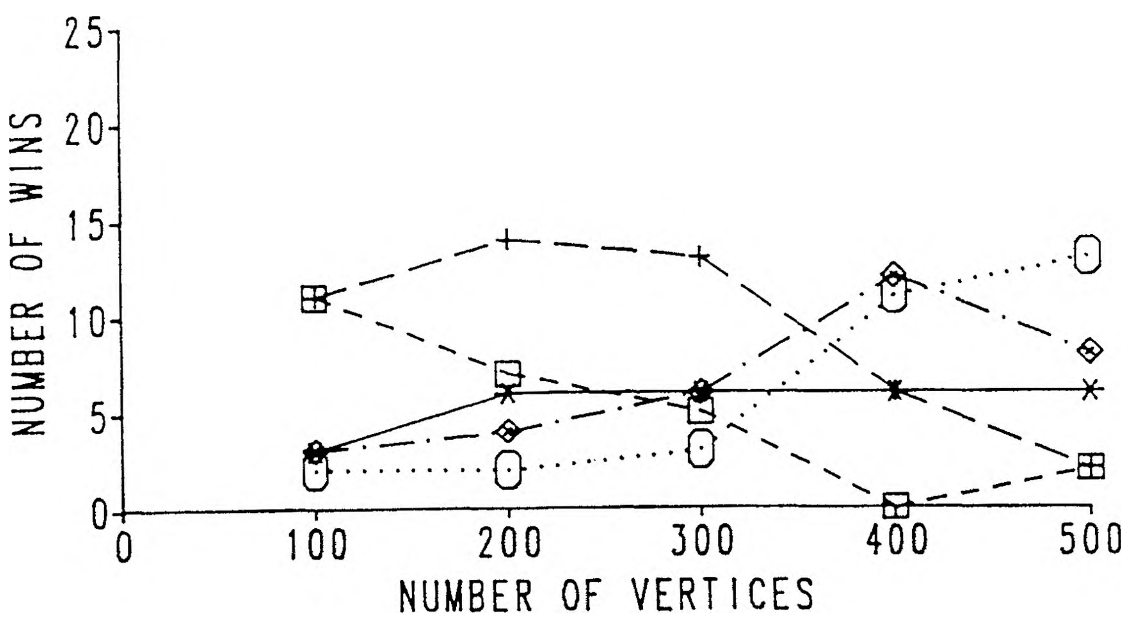


Figure 15. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.10 and d = BIN

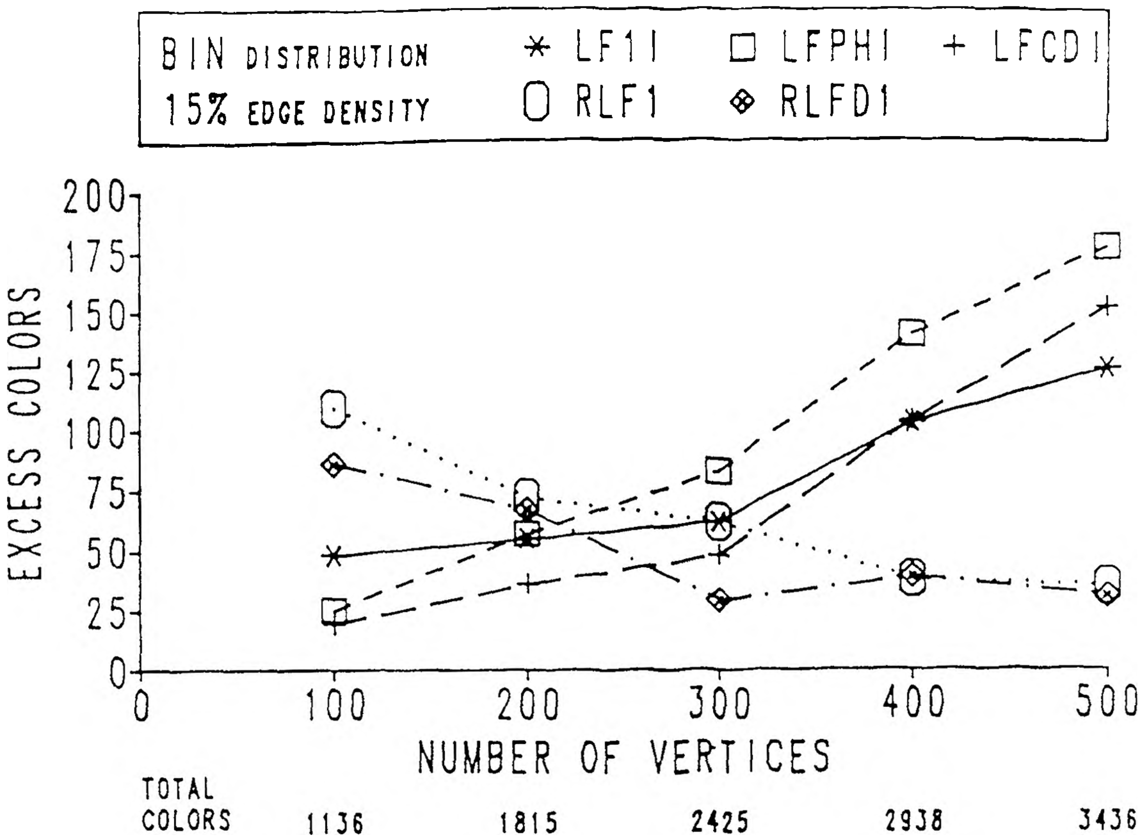


Figure 16. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.15 and d = BIN

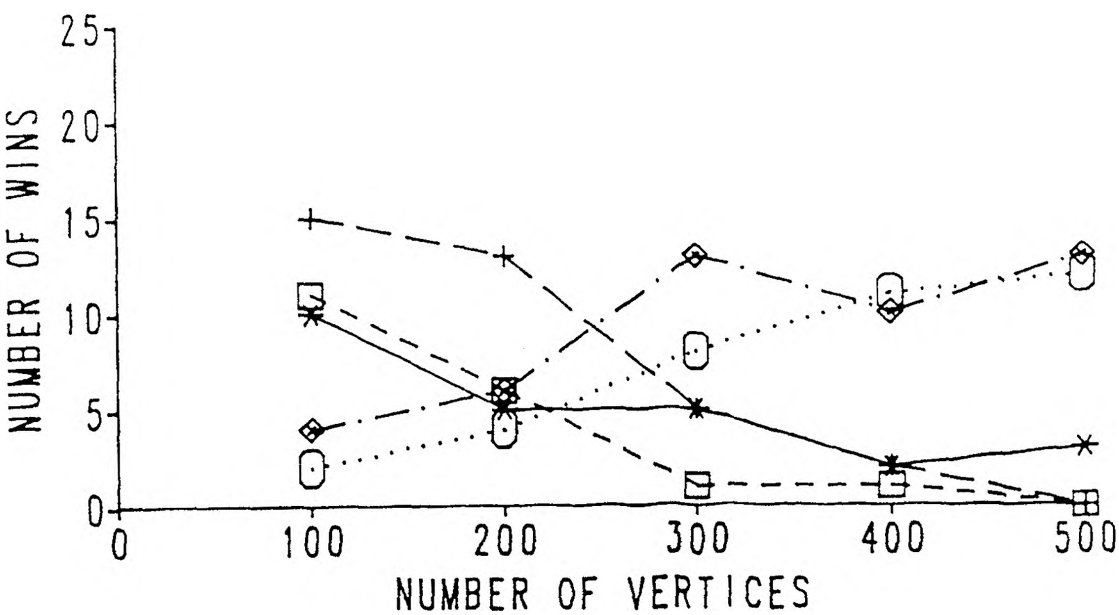


Figure 17. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.15 and d = BIN

Bill 67

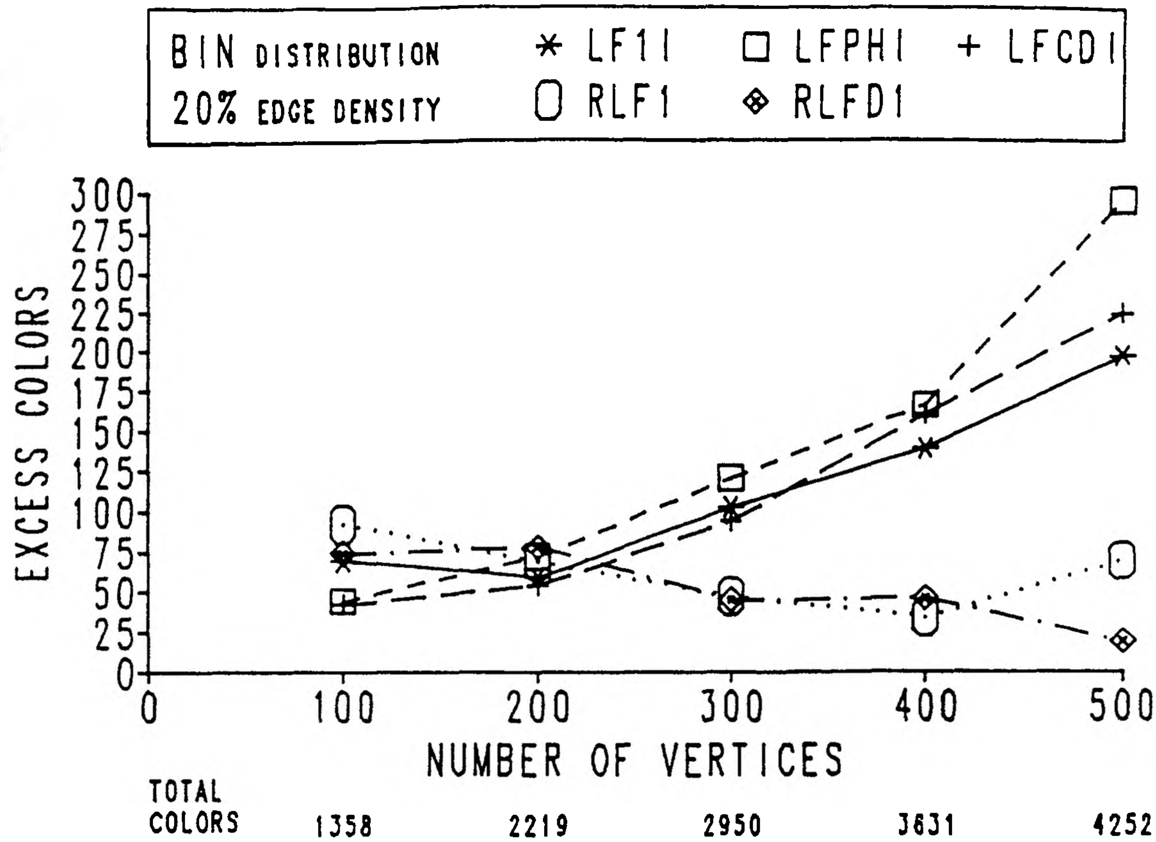


Figure 18. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.20 and d = BIN

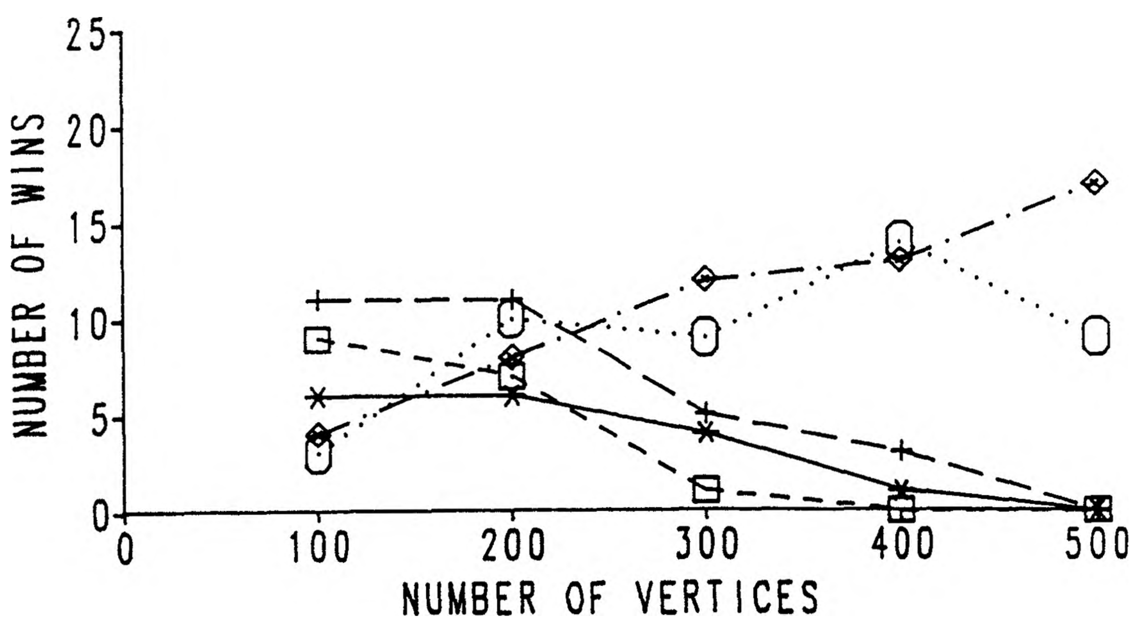
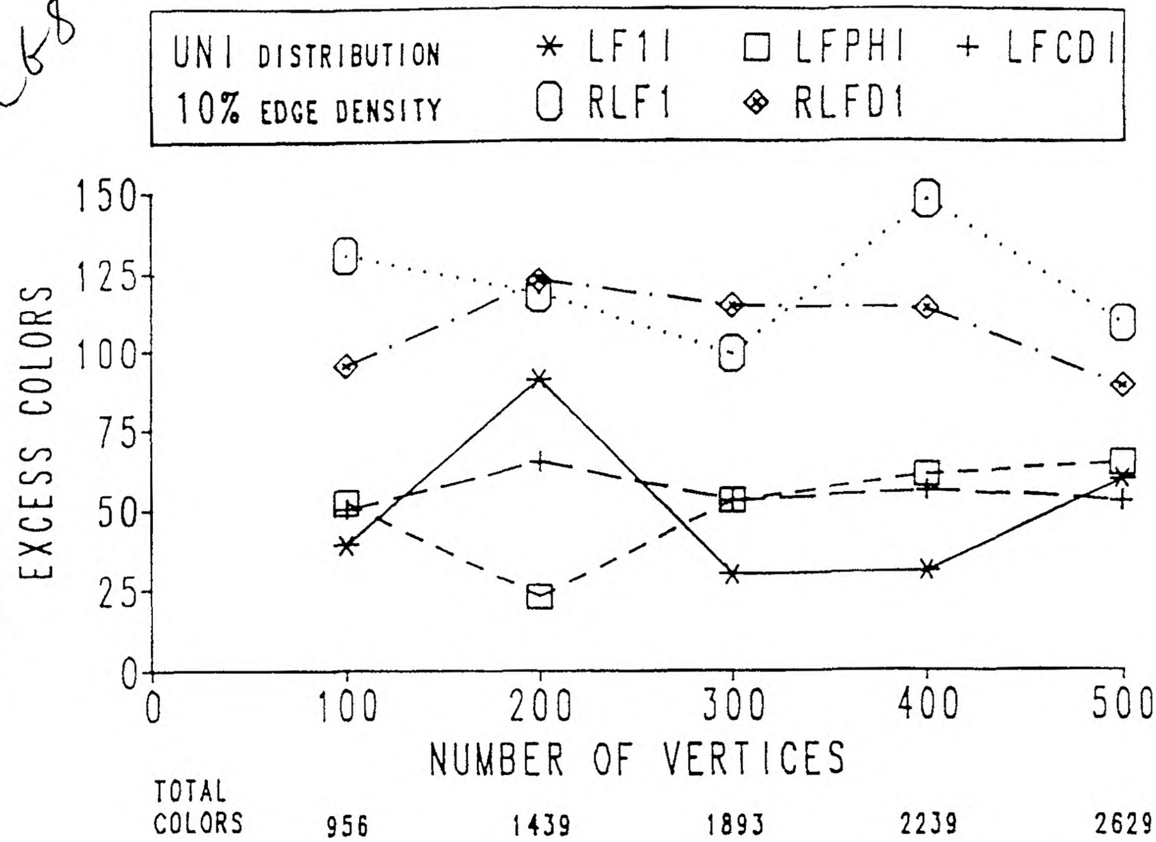


Figure 19. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.20 and d = BIN



That one

2/23

Figure 20. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.10 and d = UNI

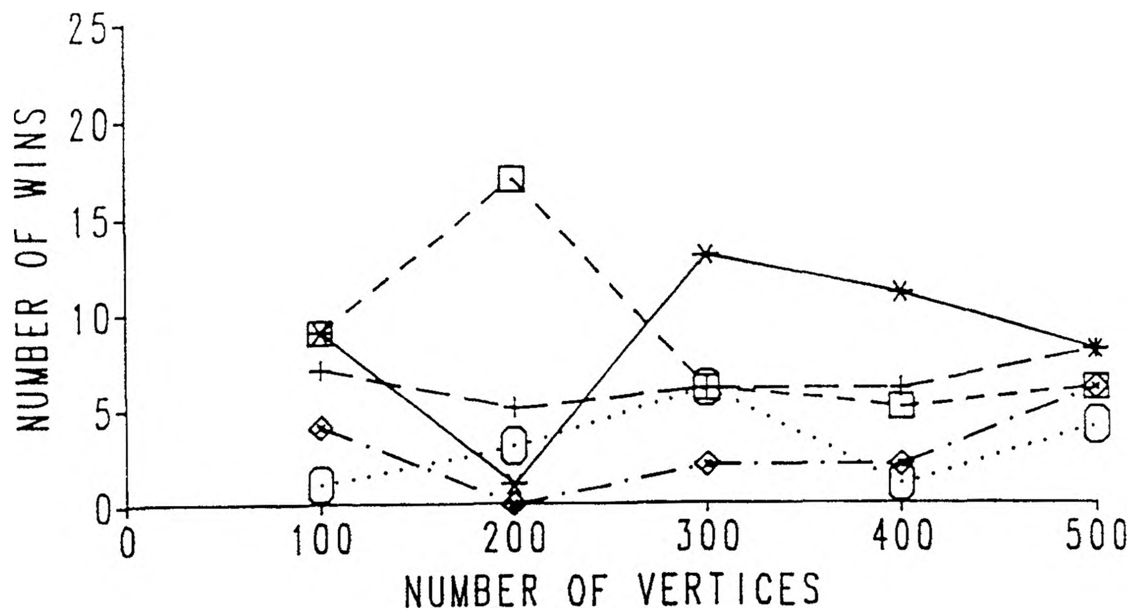


Figure 21. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.10 and d = UNI

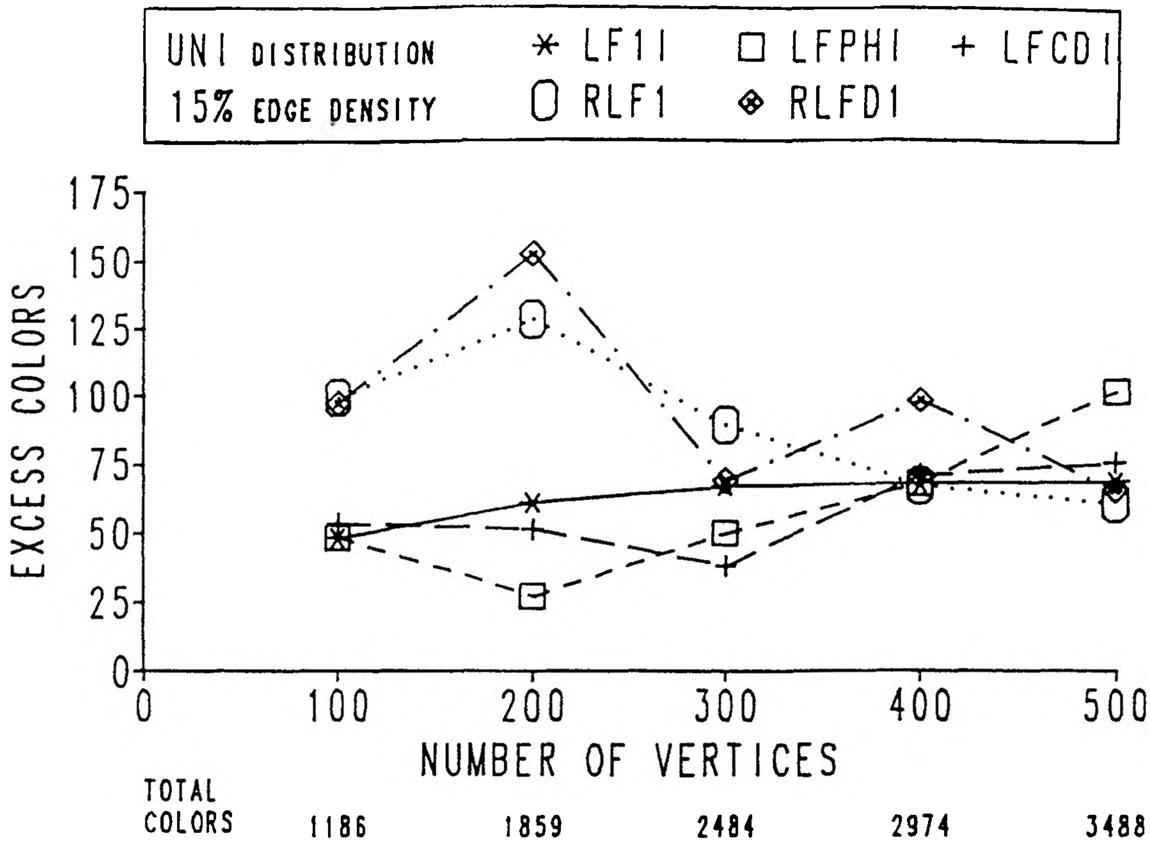


Figure 22. Number of Excess Colors vs. Number of Vertices for Random Composite Graphs with μ = 0.15 and d = UNI

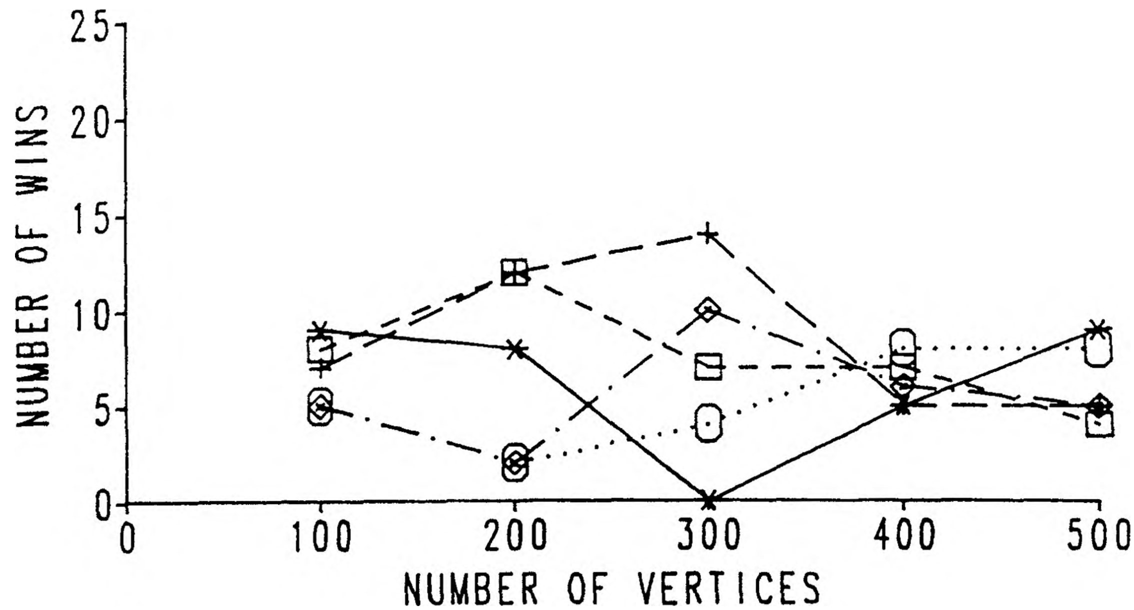


Figure 23. Number of Wins vs. Number of Vertices for Random Composite Graphs with μ = 0.15 and d = UNI

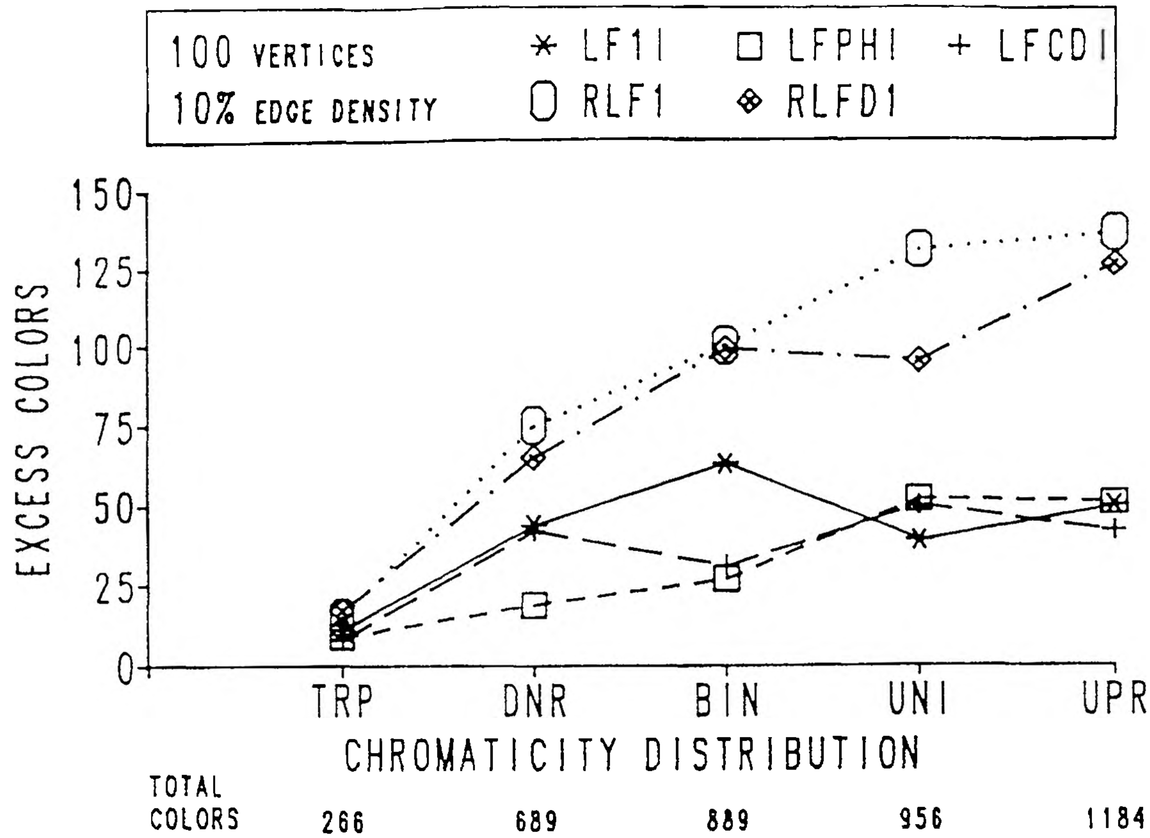


Figure 42. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.10

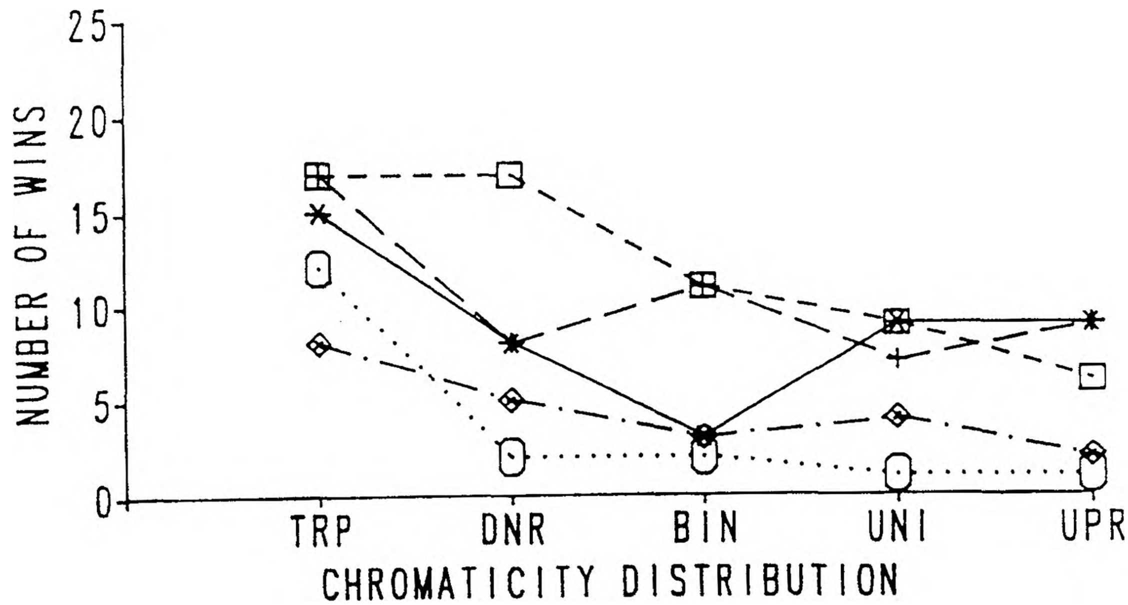


Figure 43. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.10

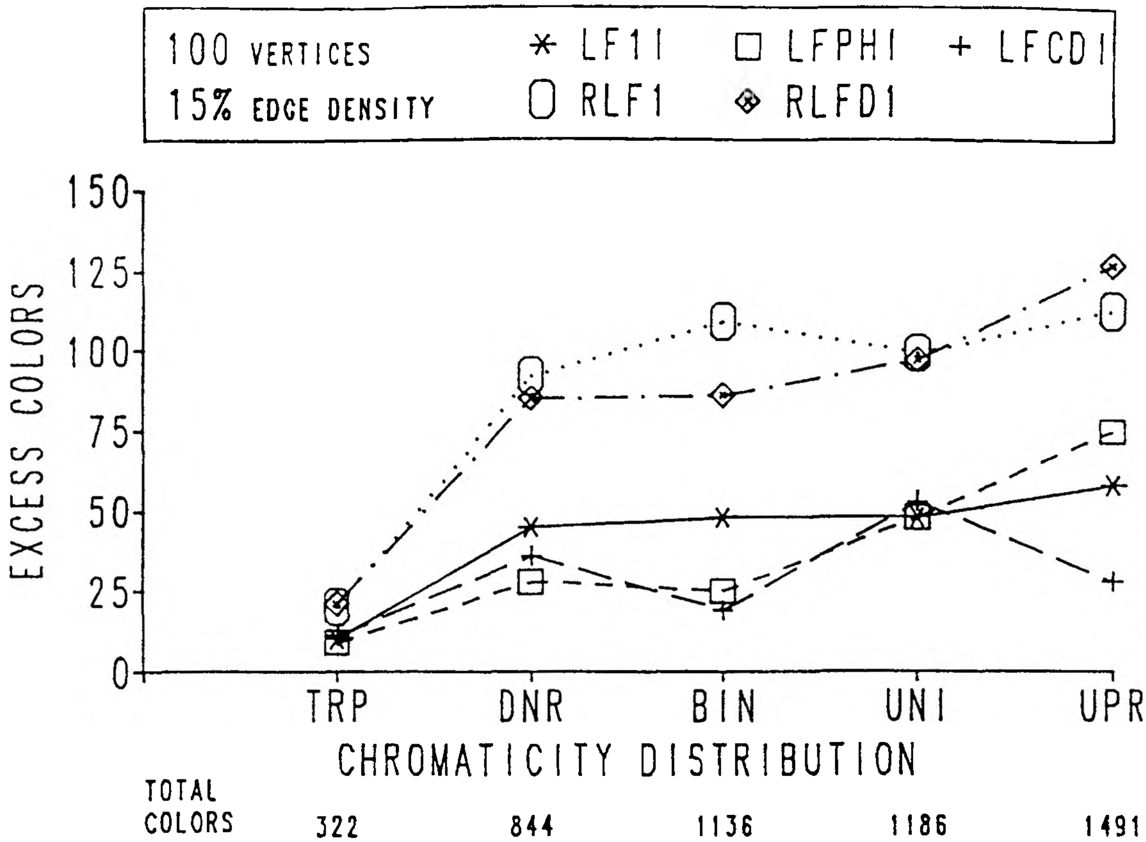


Figure 44. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.15

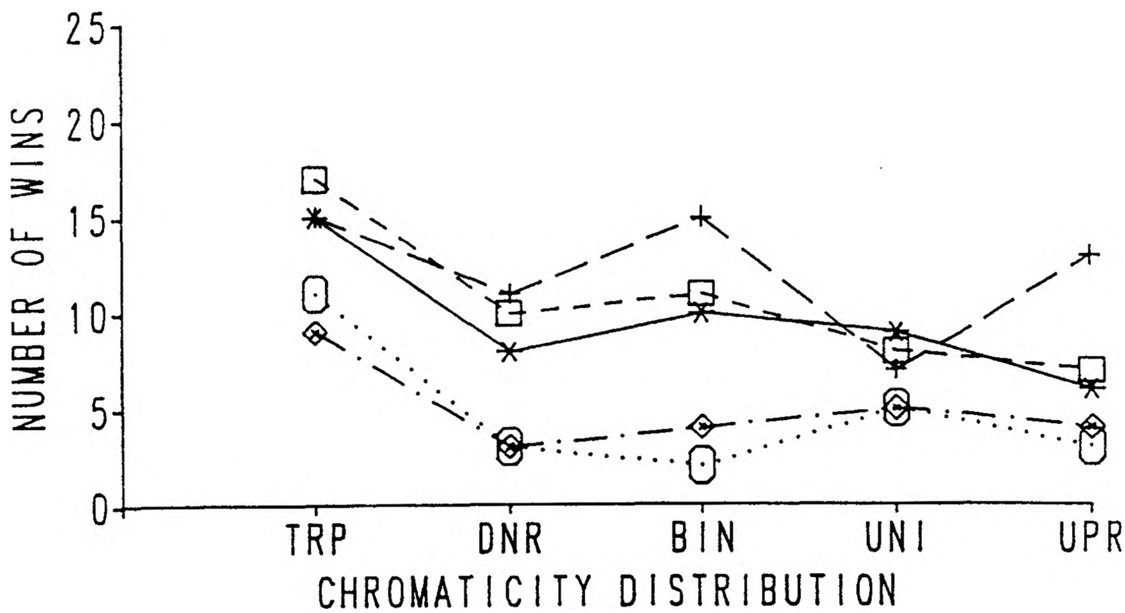


Figure 45. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.15

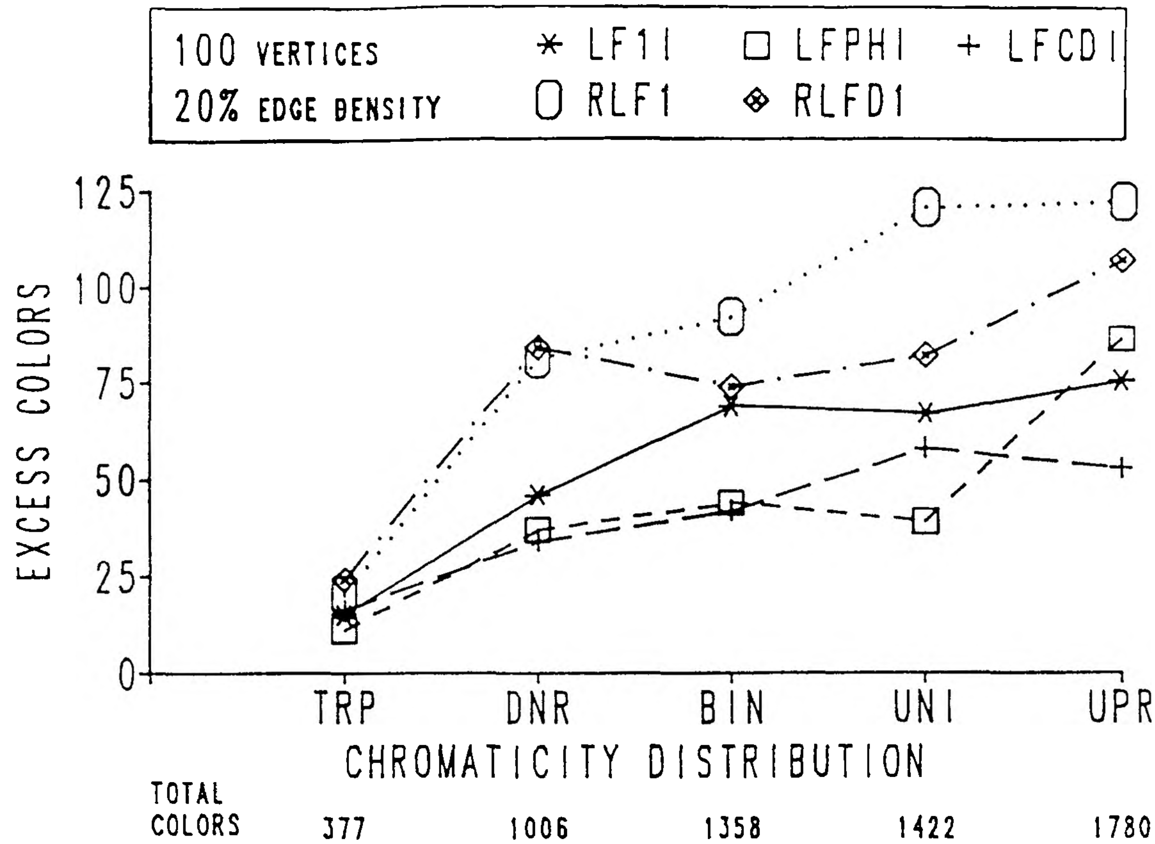


Figure 46. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.20

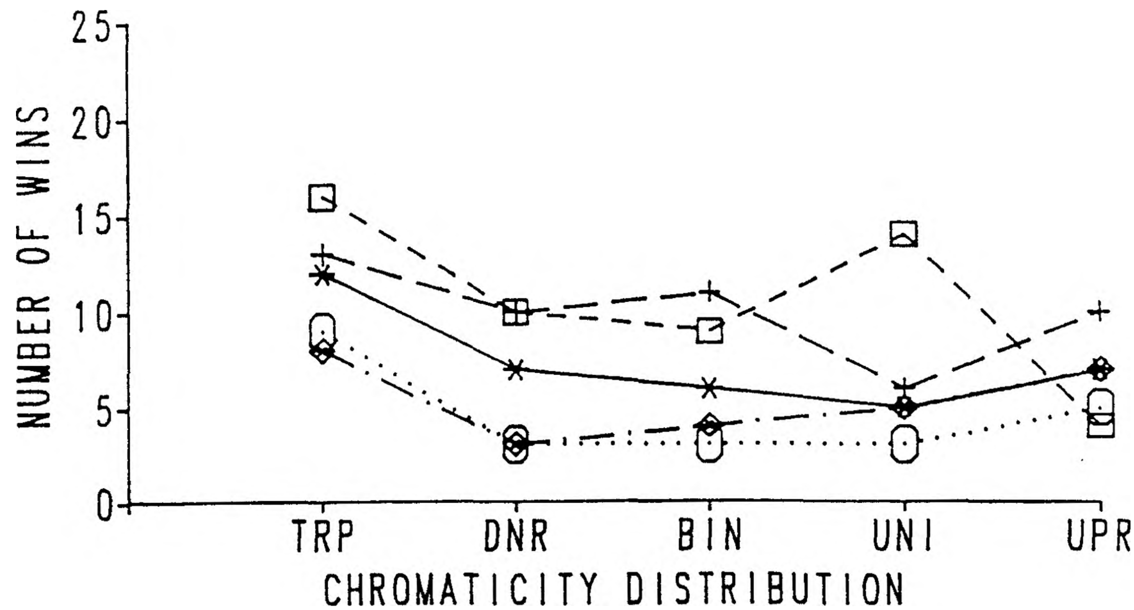


Figure 47. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.20

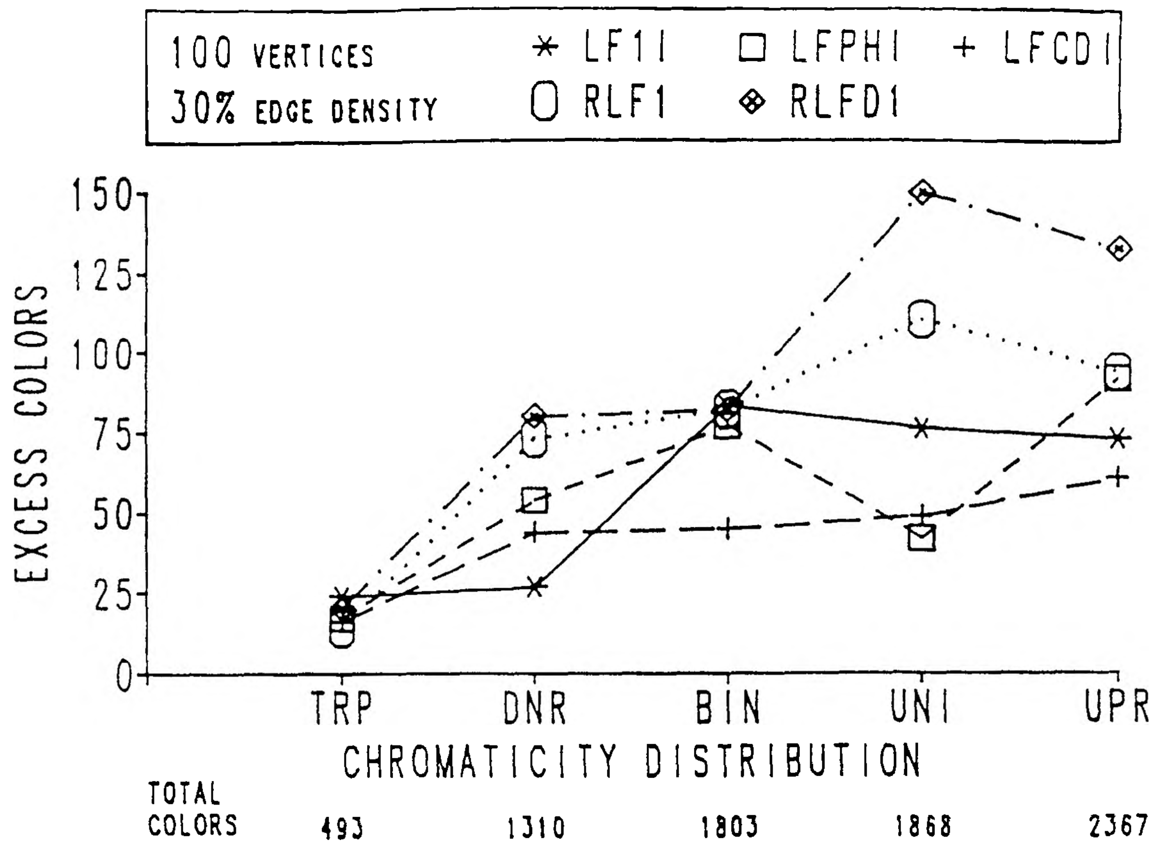


Figure 48. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.30

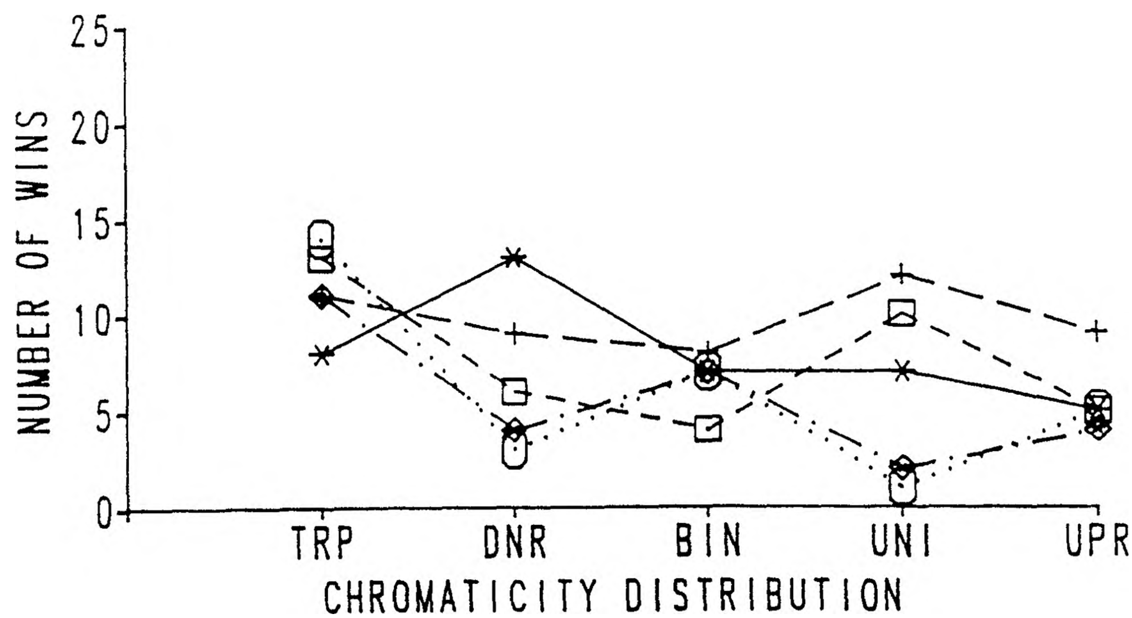


Figure 49. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.30

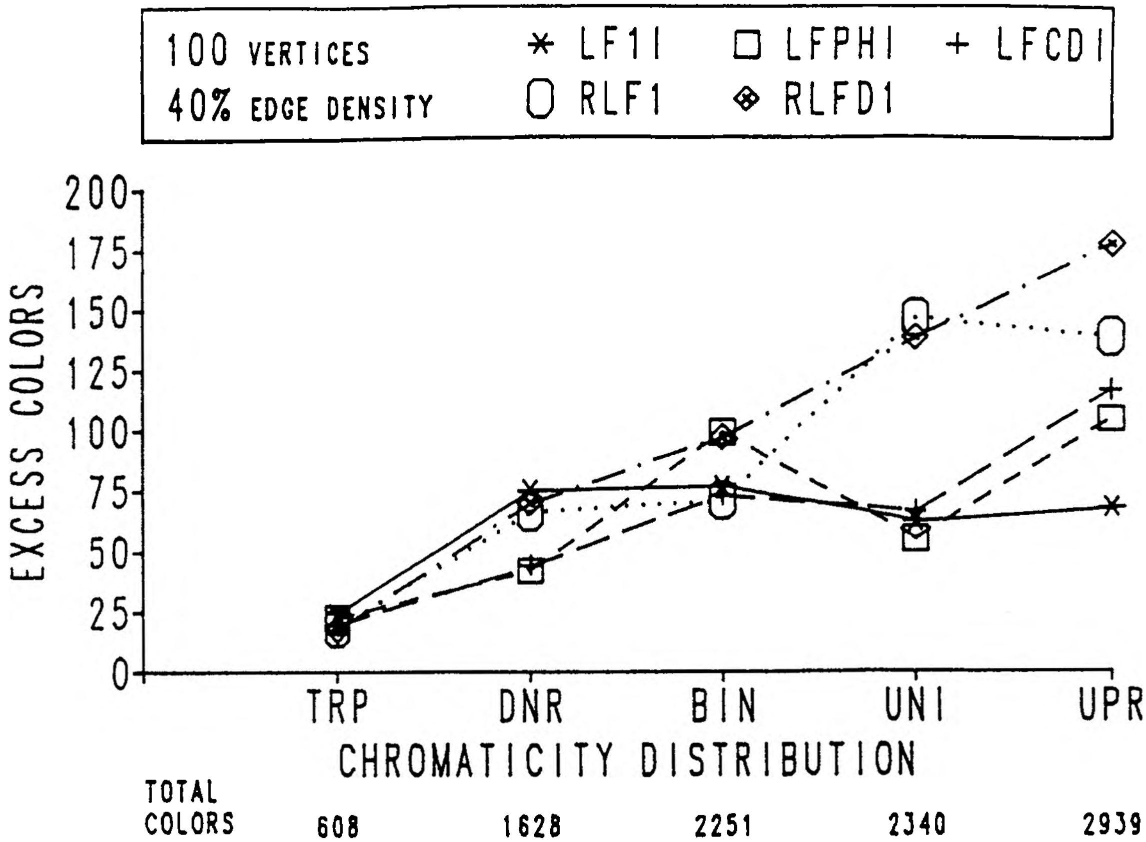


Figure 50. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.40

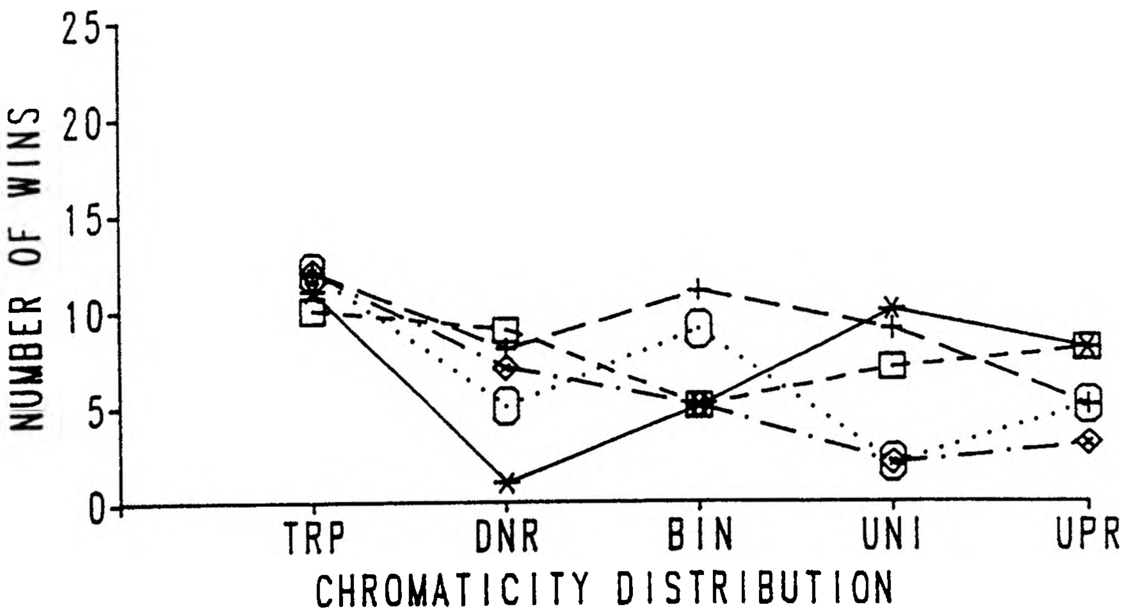


Figure 51. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.40

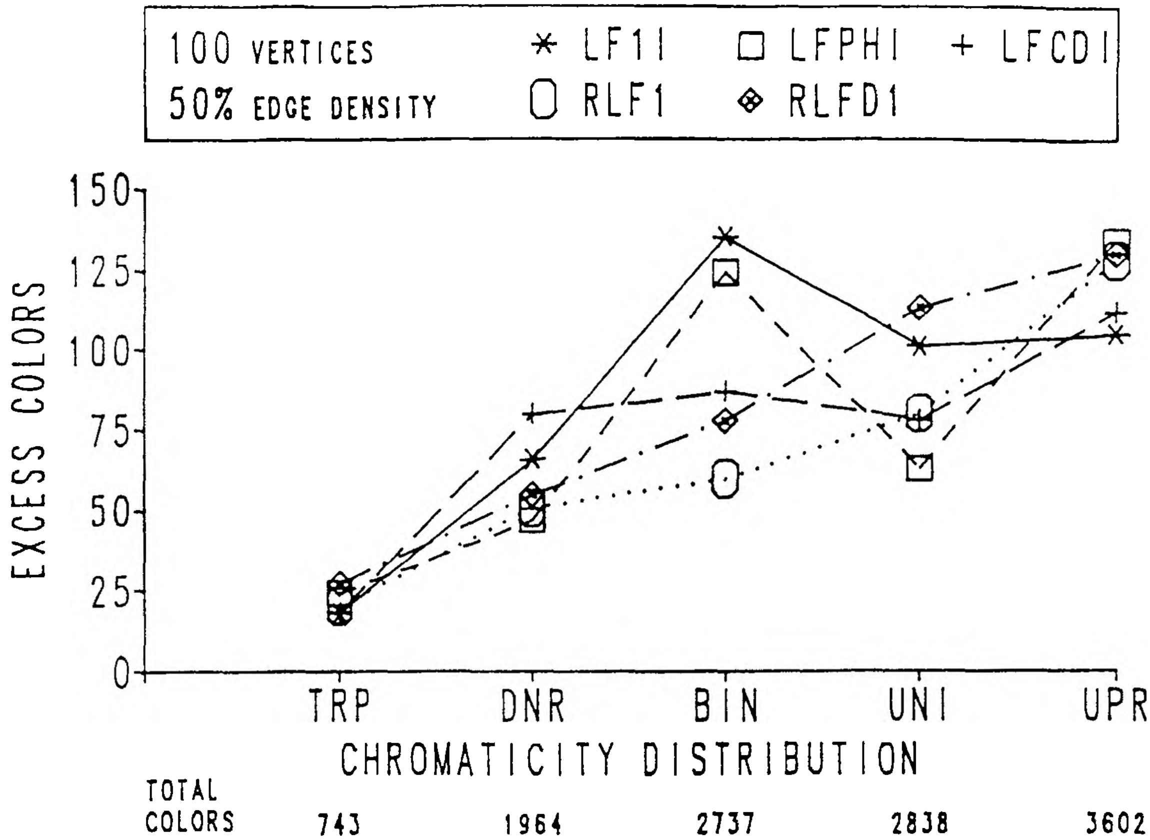


Figure 52. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.50

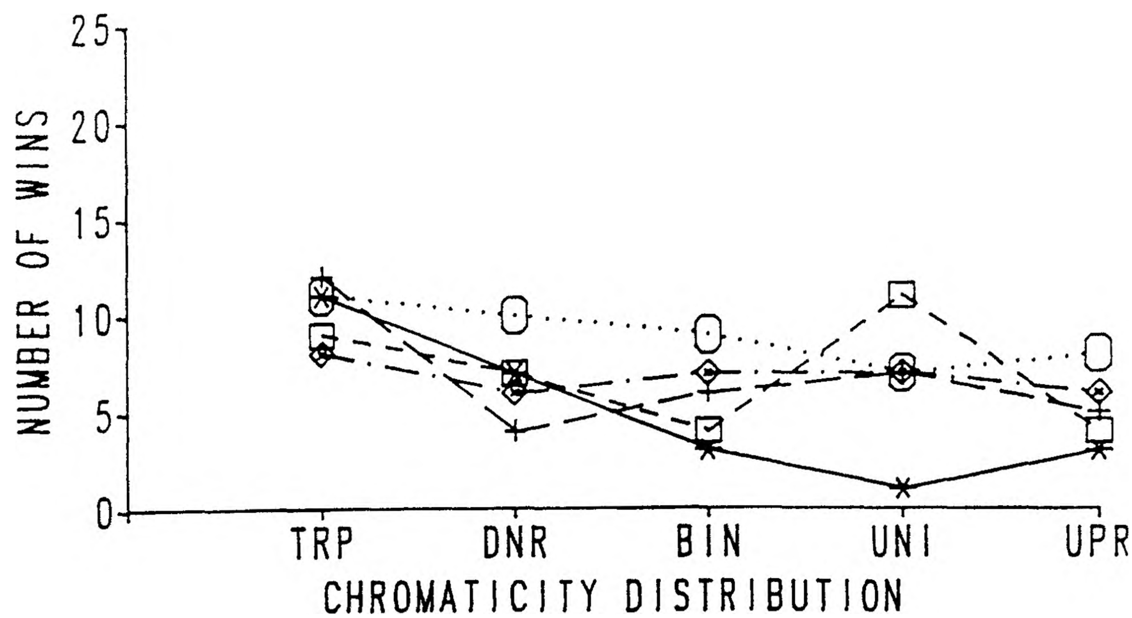


Figure 53. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 100 and μ = 0.50

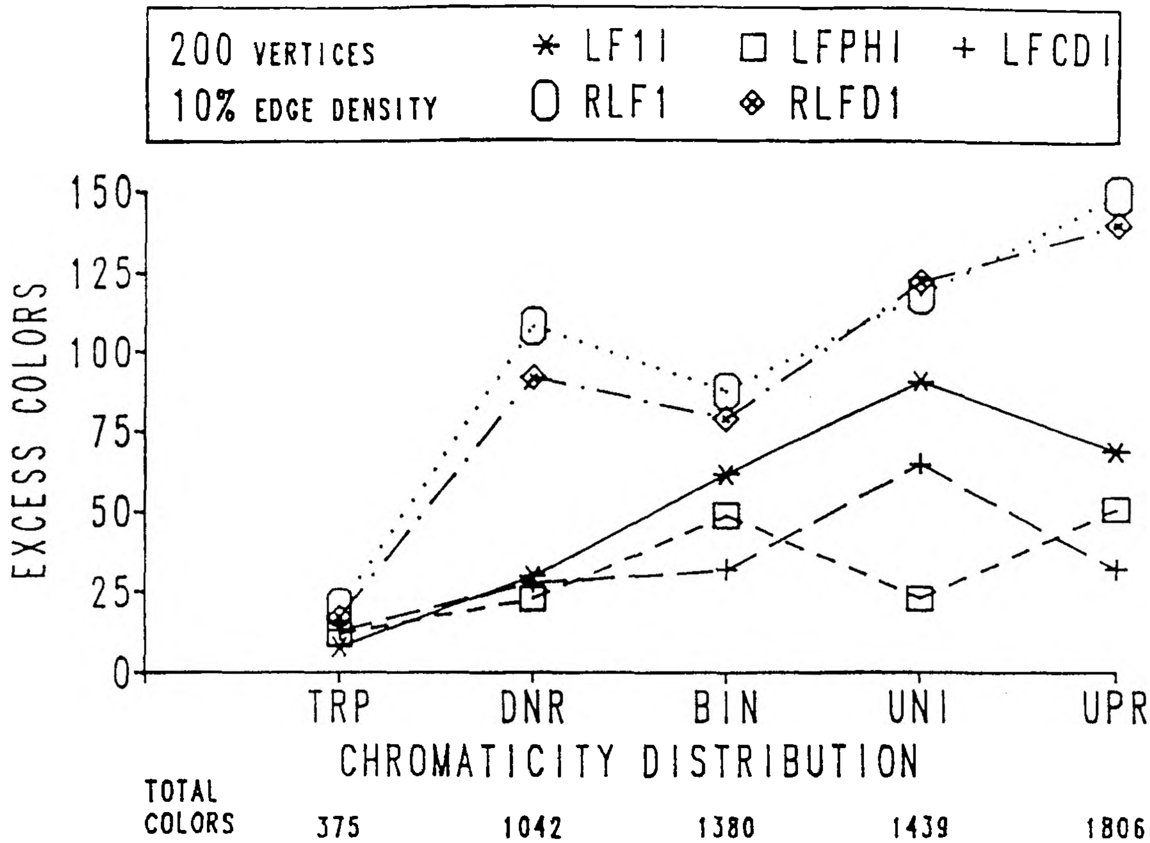


Figure 54. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 200 and $\mu = 0.10$

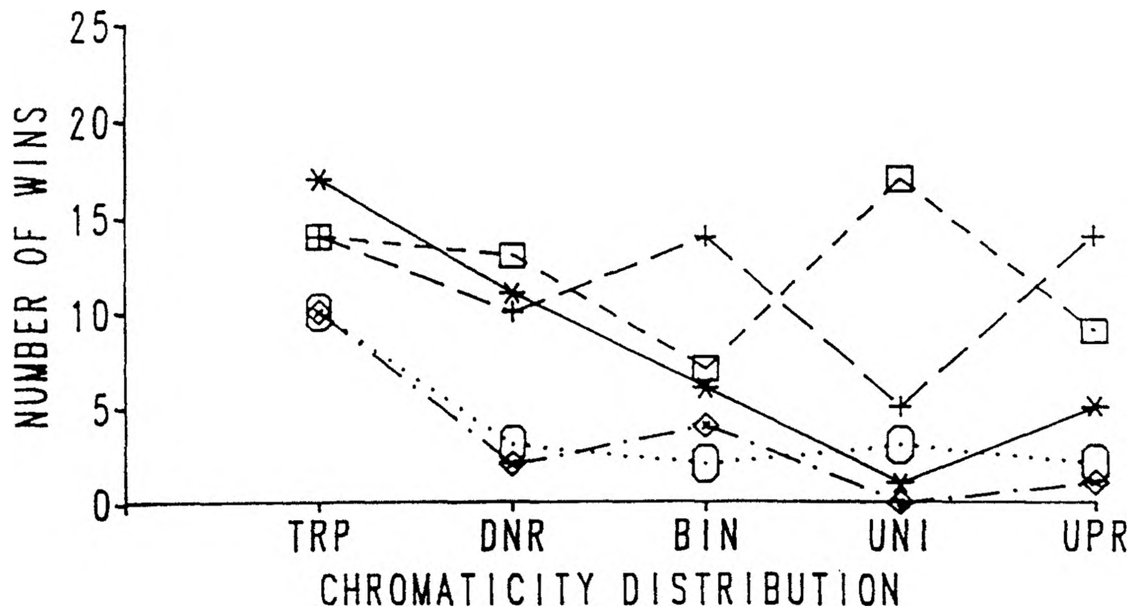


Figure 55. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 200 and $\mu = 0.10$

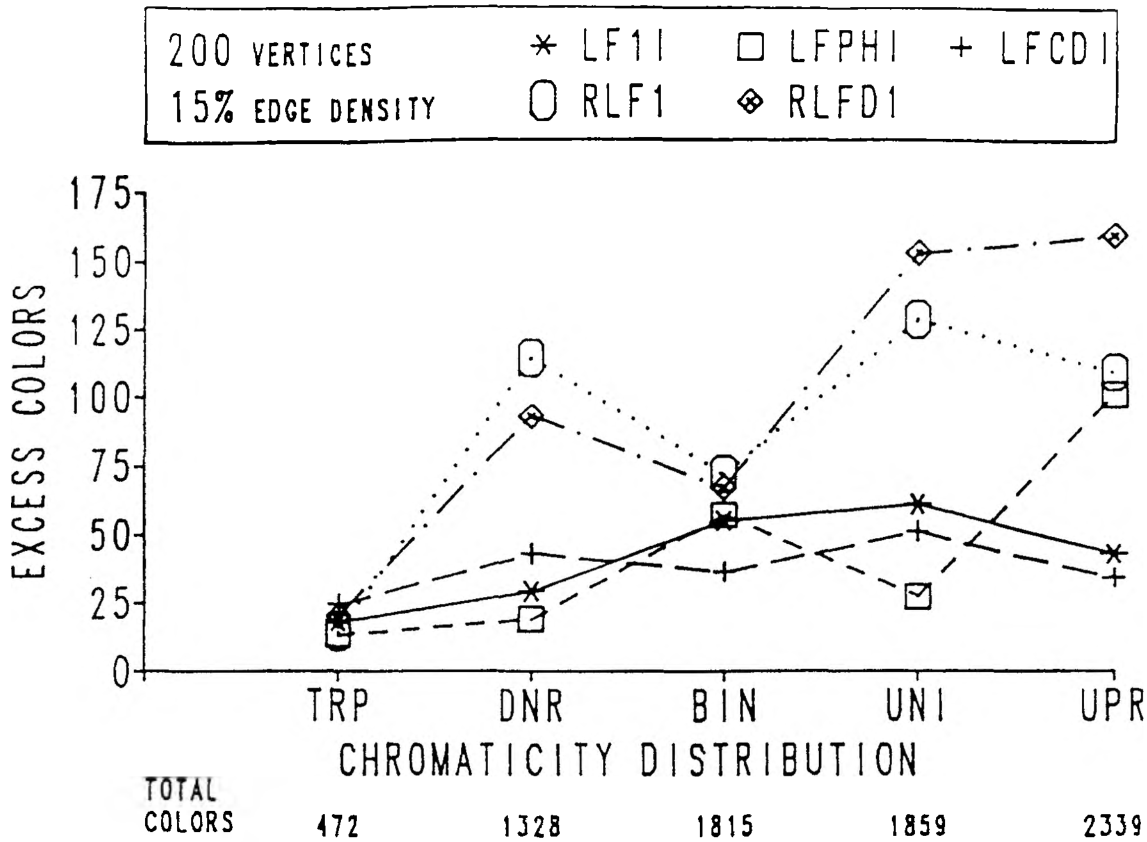


Figure 56. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 200 and μ = 0.15

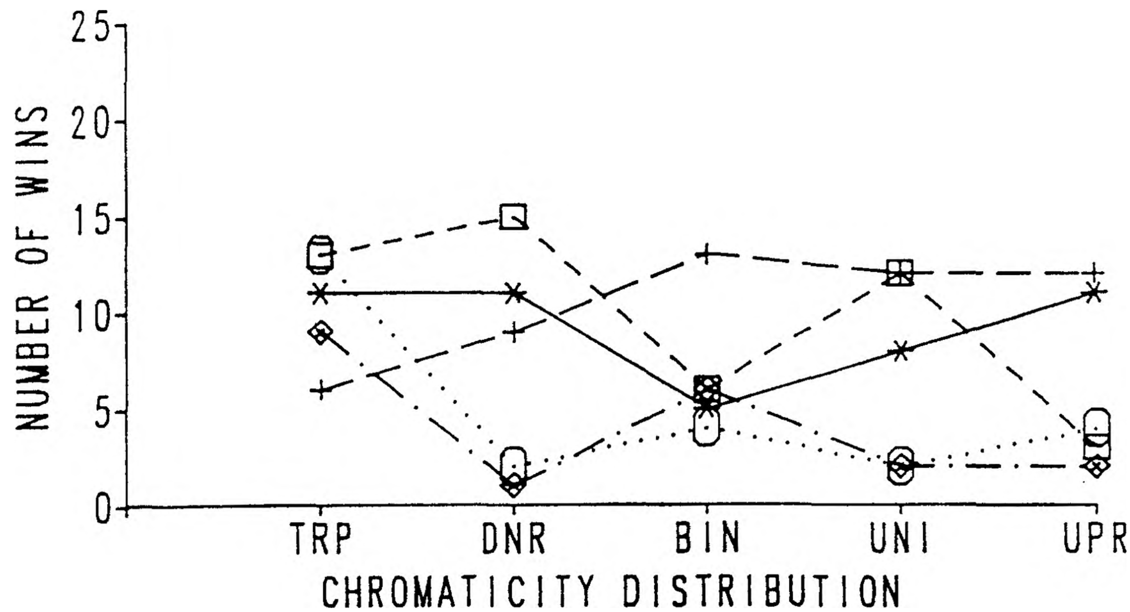


Figure 57. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 200 and μ = 0.15

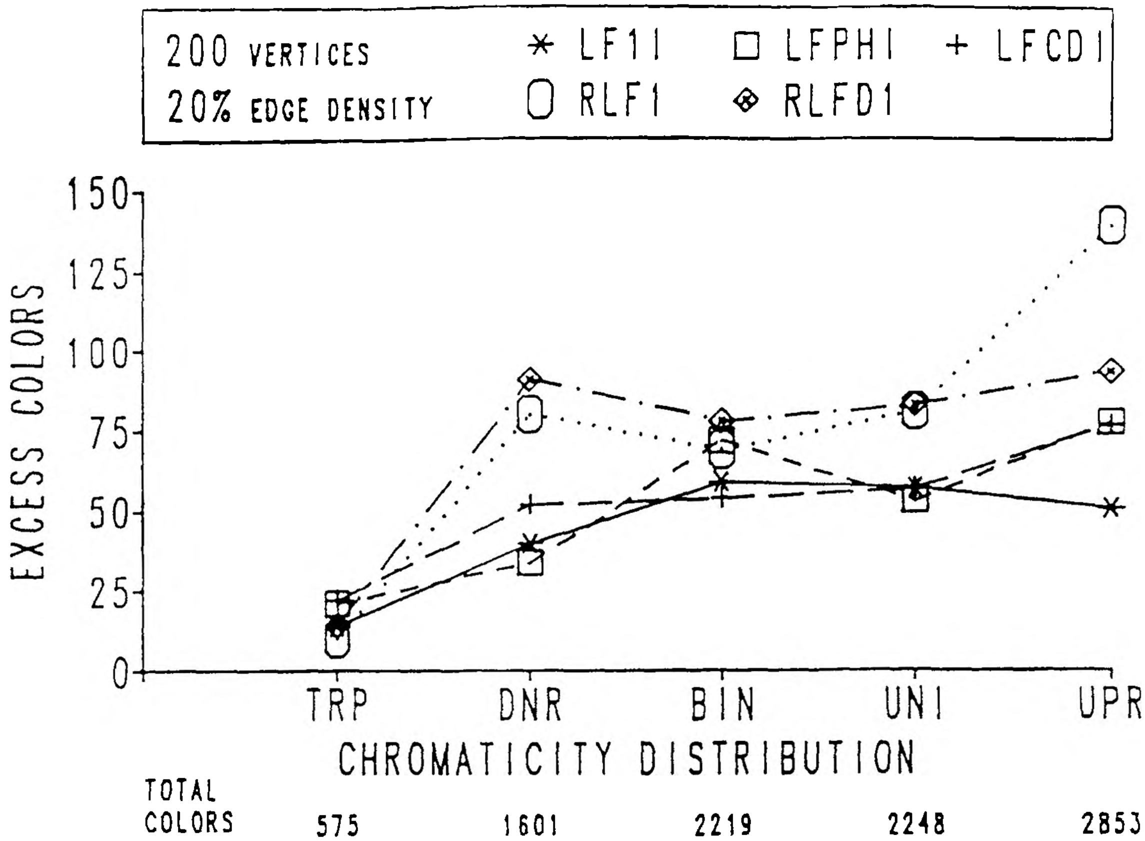


Figure 58. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 200 and μ = 0.20

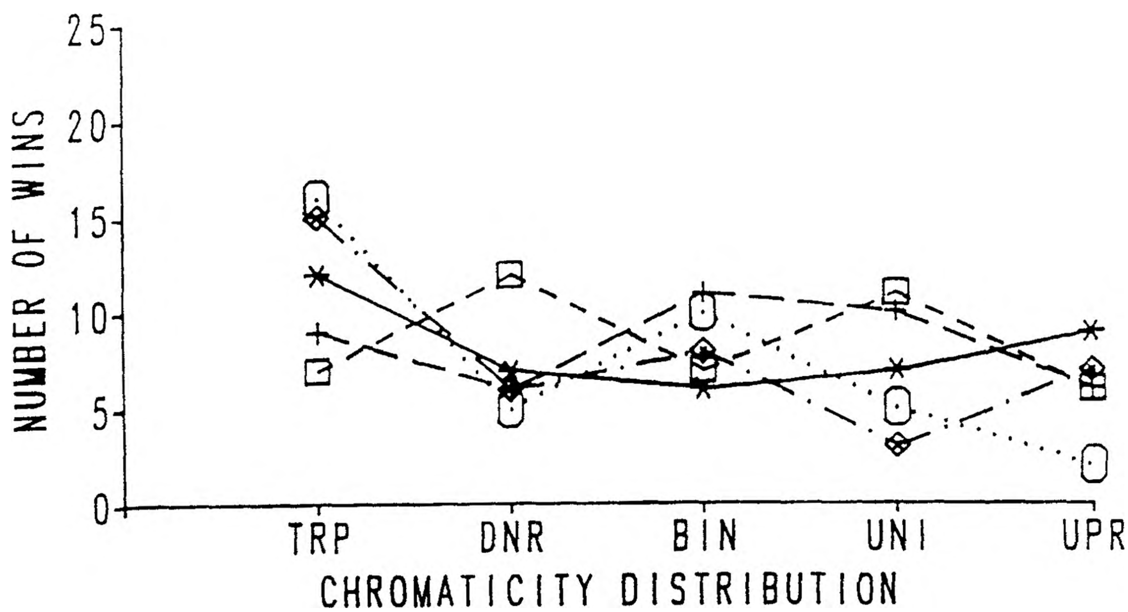


Figure 59. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 200 and μ = 0.20

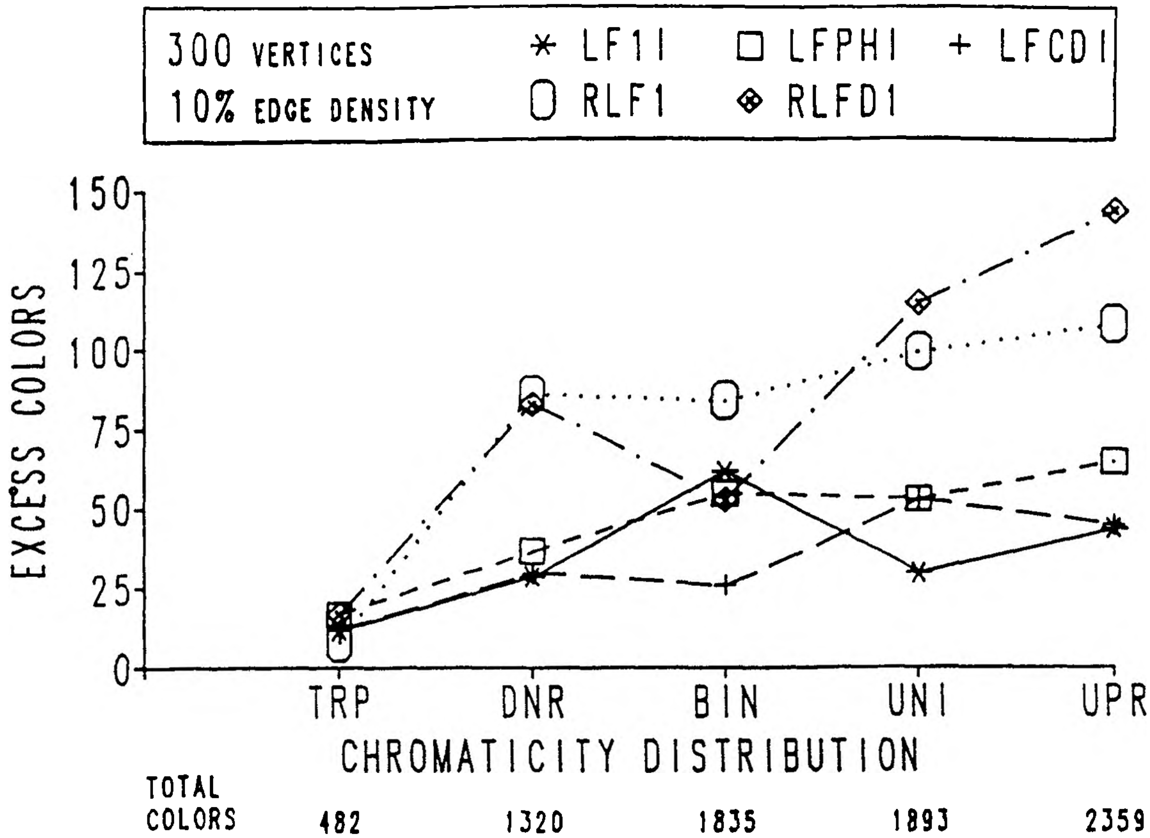


Figure 60. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with $n = 300$ and $\mu = 0.10$

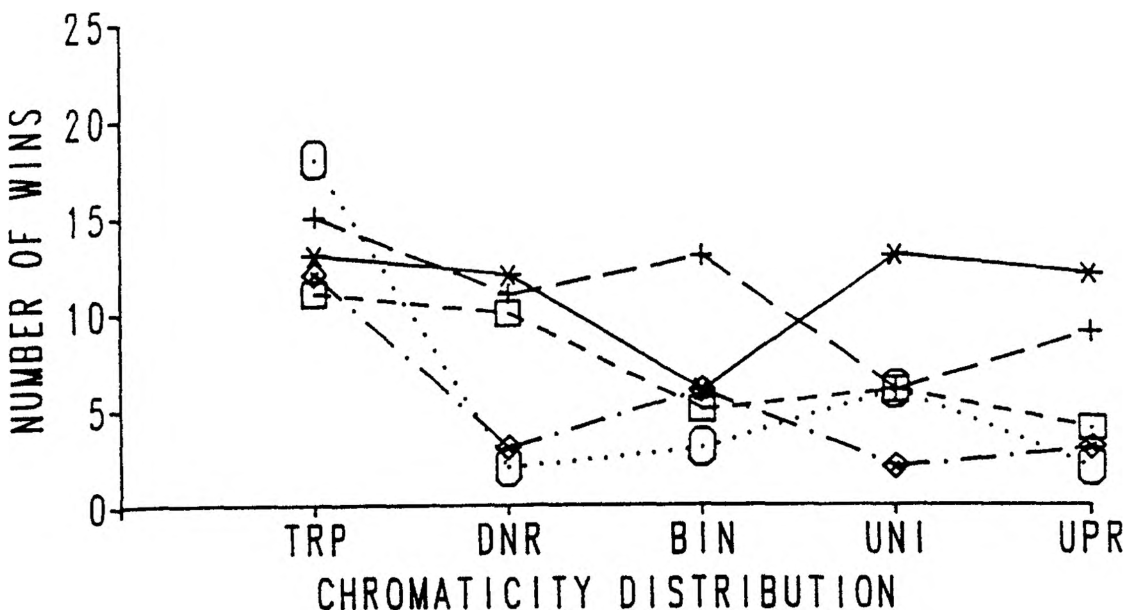


Figure 61. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with $n = 300$ and $\mu = 0.10$

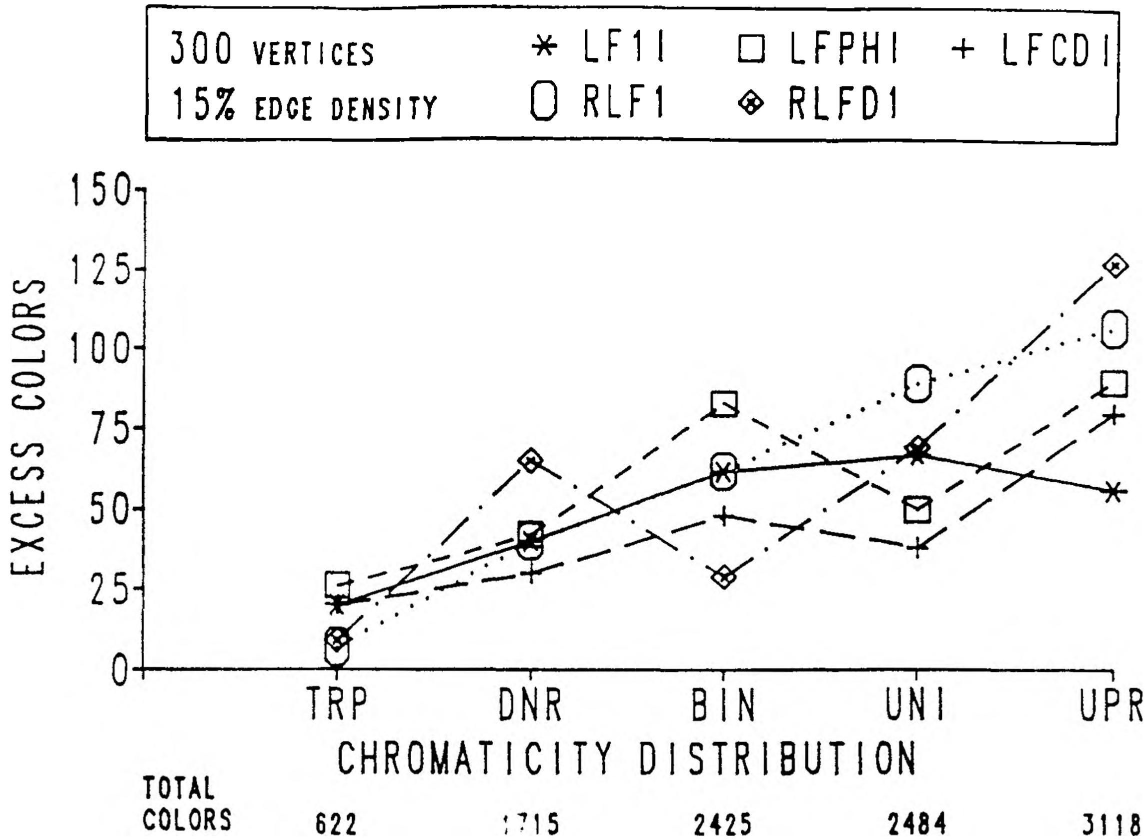


Figure 62. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 300 and μ = 0.15

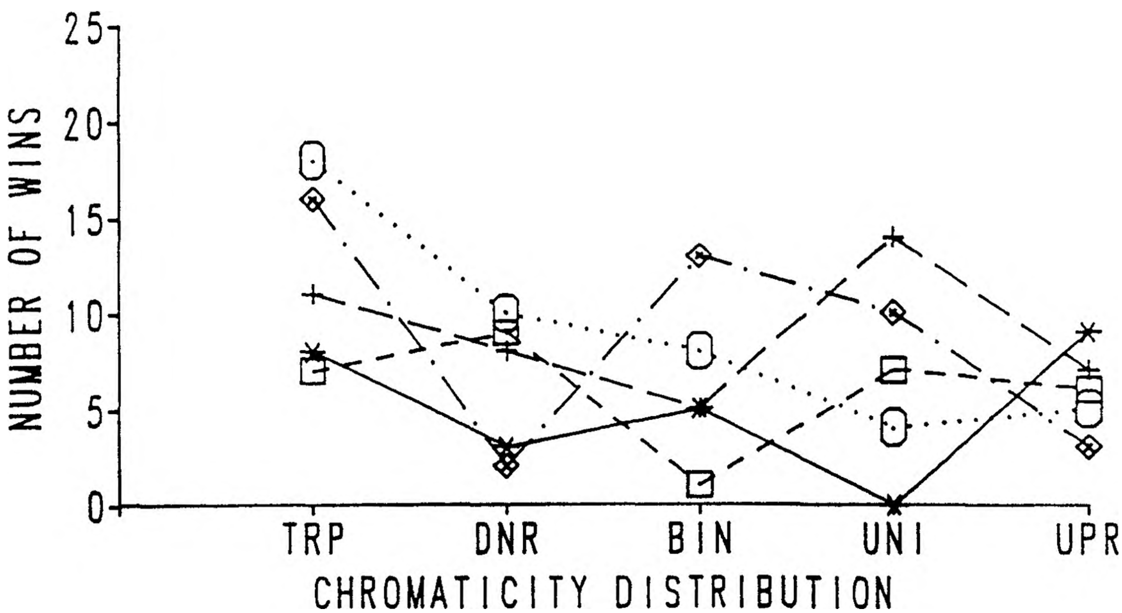


Figure 63. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 300 and μ = 0.15

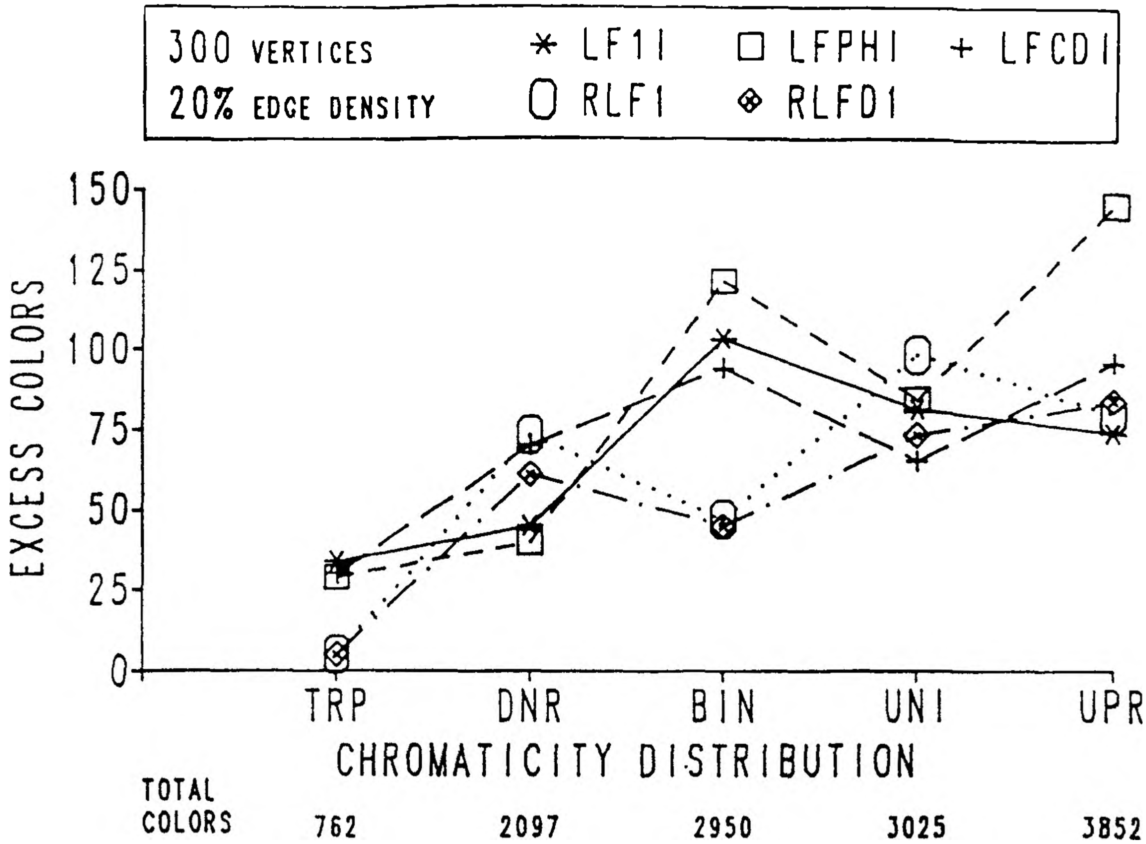


Figure 64. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 300 and μ = 0.20

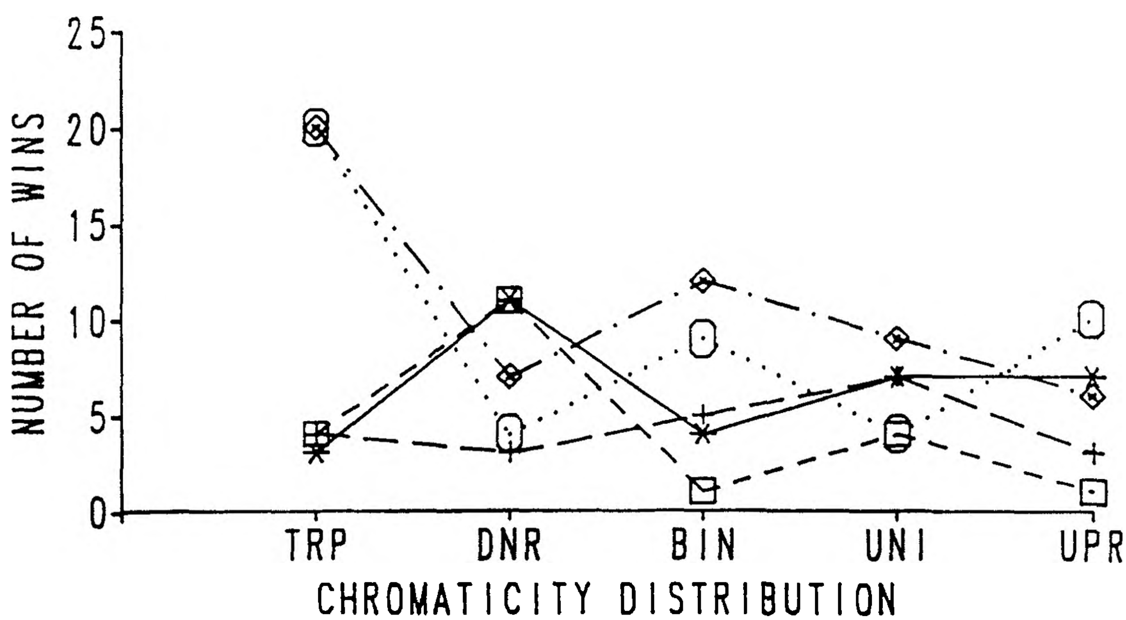


Figure 65. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 300 and μ = 0.20

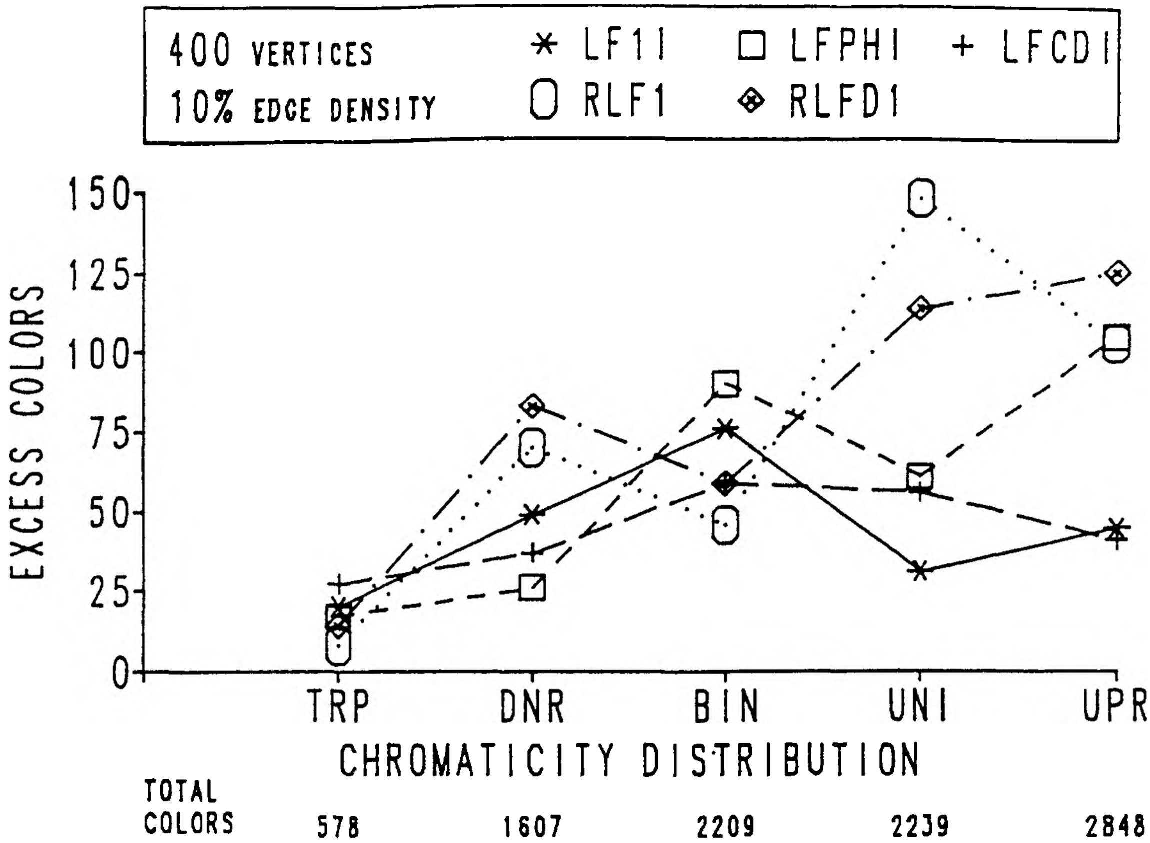


Figure 66. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 400 and μ = 0.10

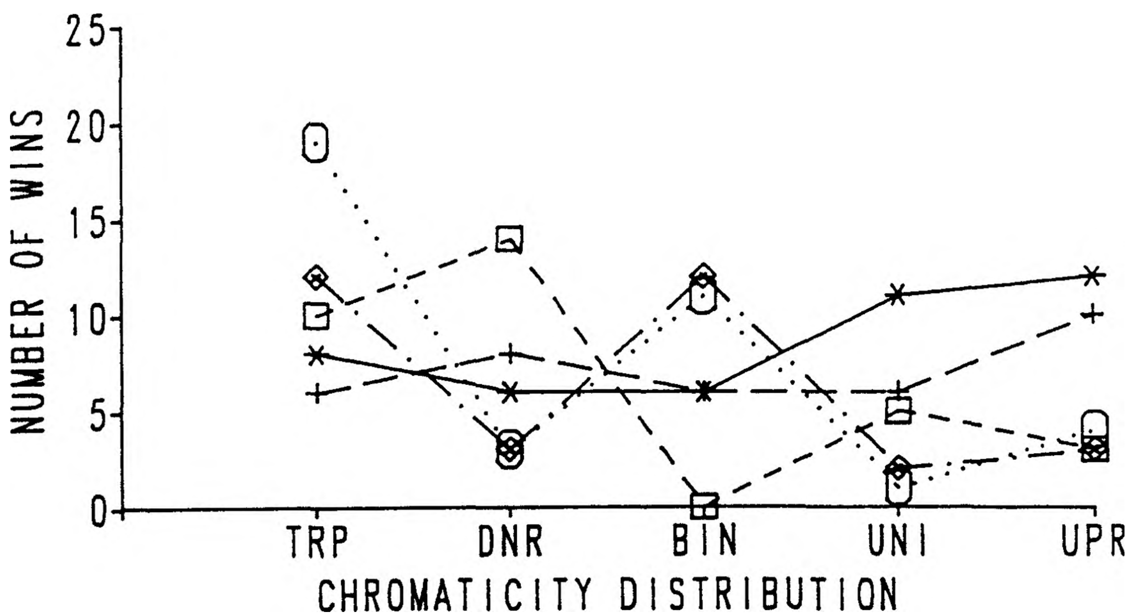


Figure 67. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 400 and μ = 0.10

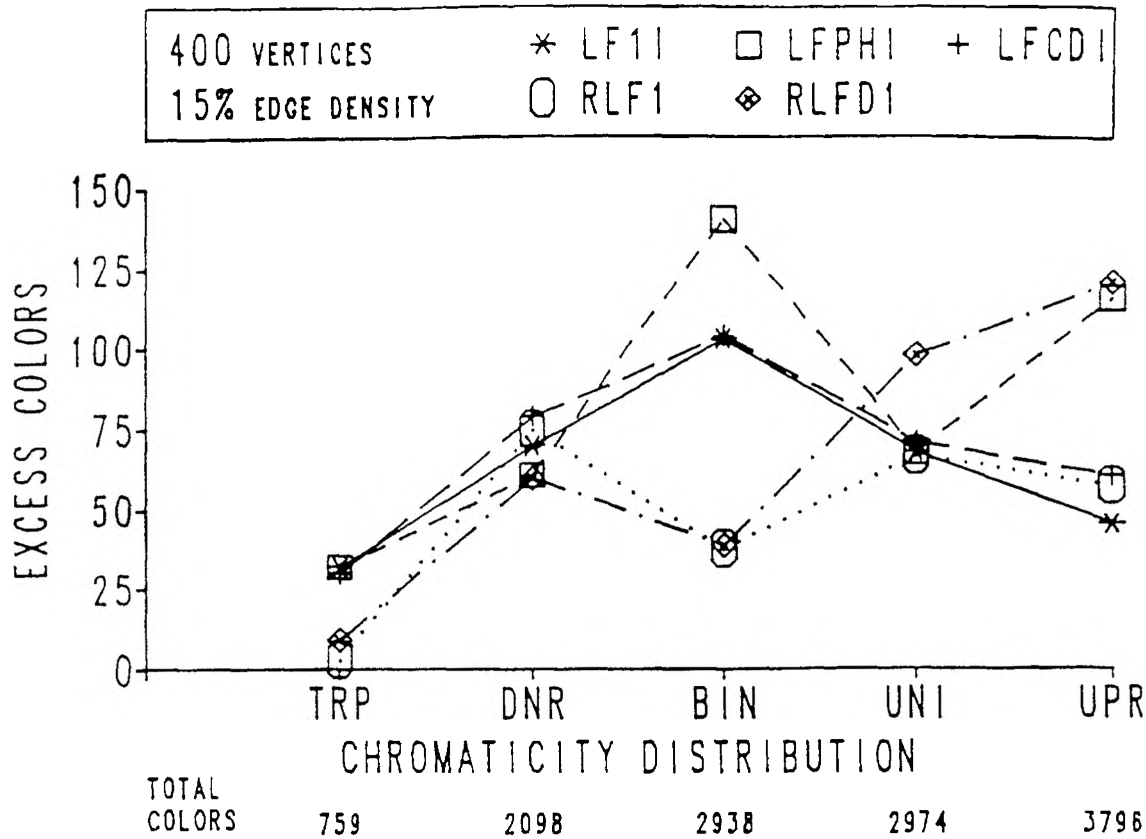


Figure 68. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 400 and μ = 0.15

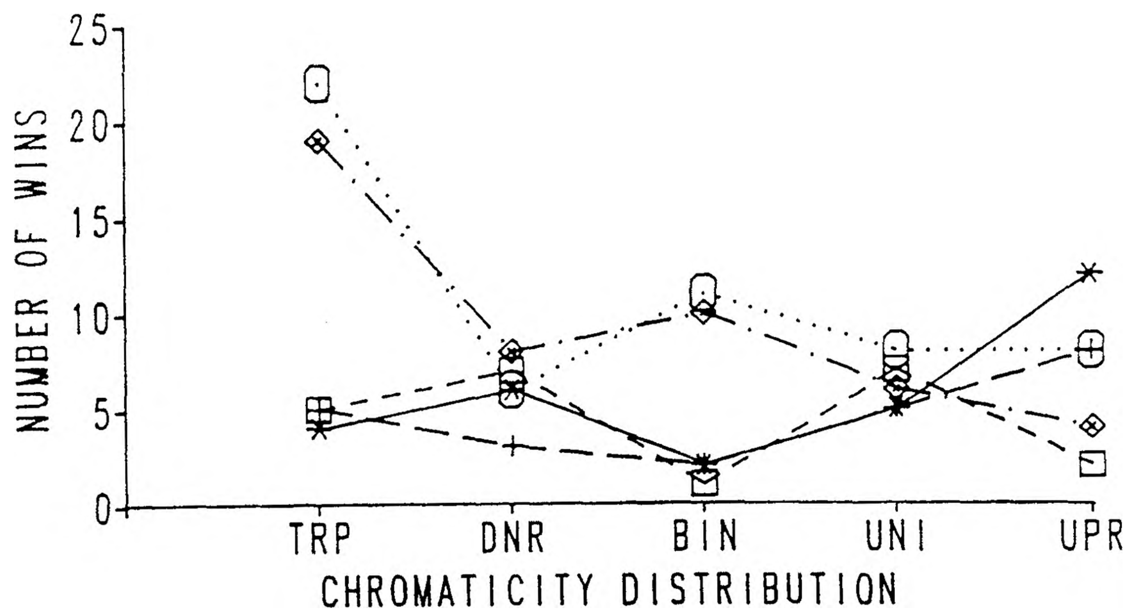


Figure 69. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 400 and μ = 0.15

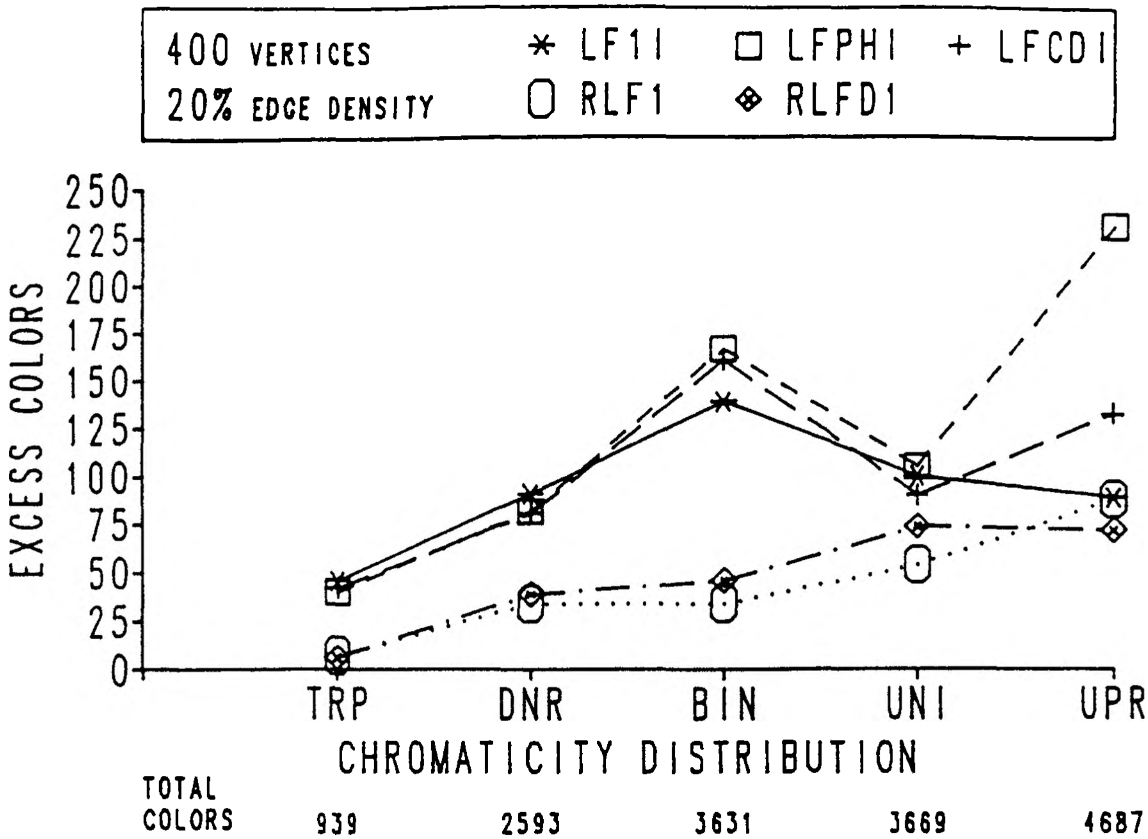


Figure 70. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 400 and μ = 0.20

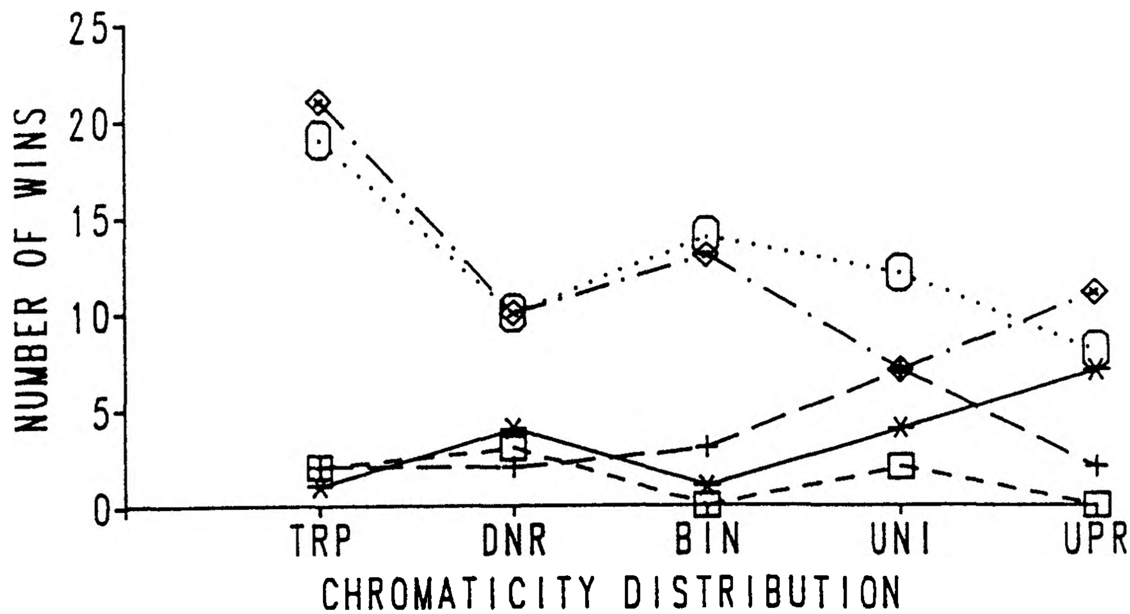


Figure 71. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 400 and μ = 0.20

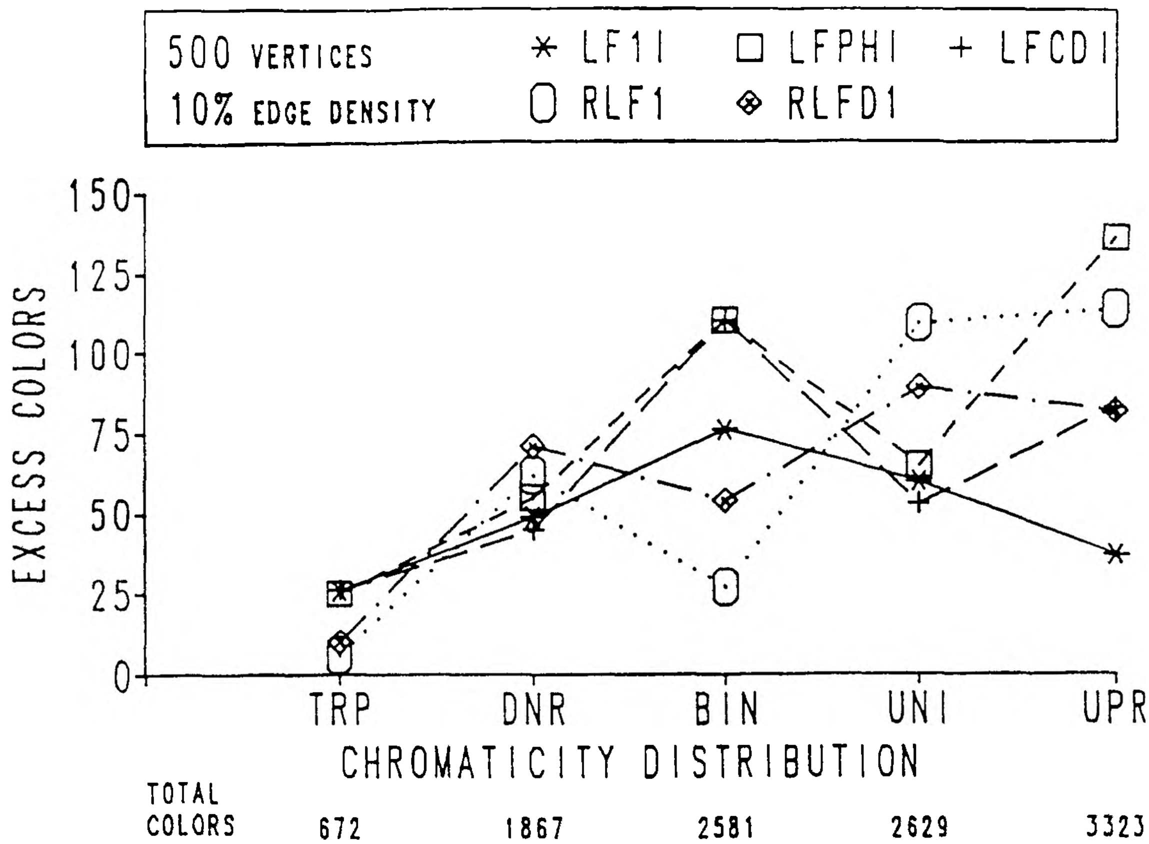


Figure 72. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with $n = 500$ and $\mu = 0.10$

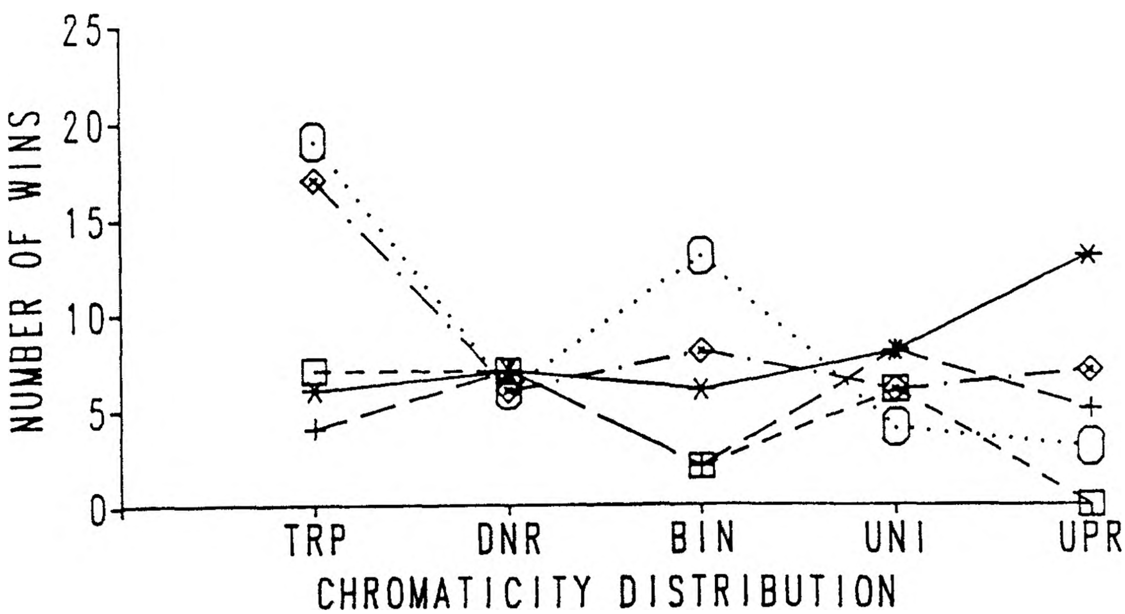


Figure 73. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with $n = 500$ and $\mu = 0.10$

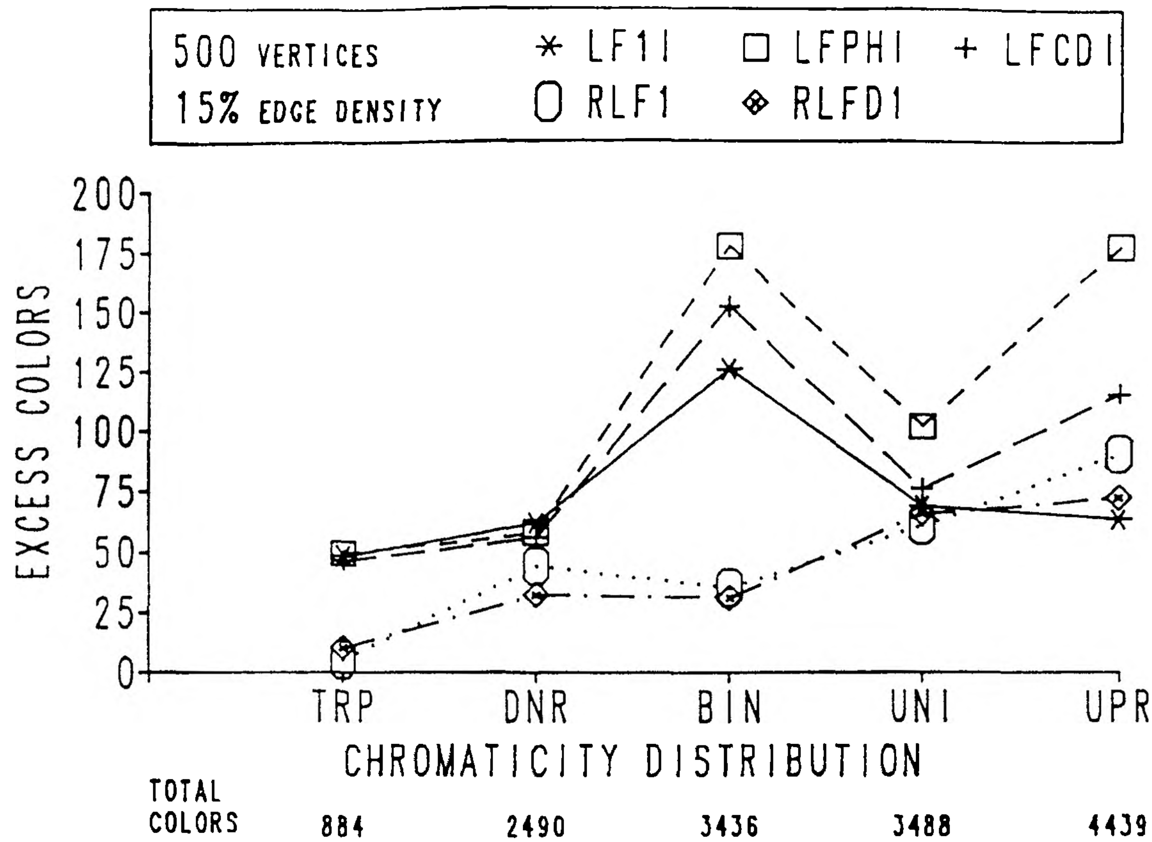


Figure 74. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with $n = 500$ and $\mu = 0.15$

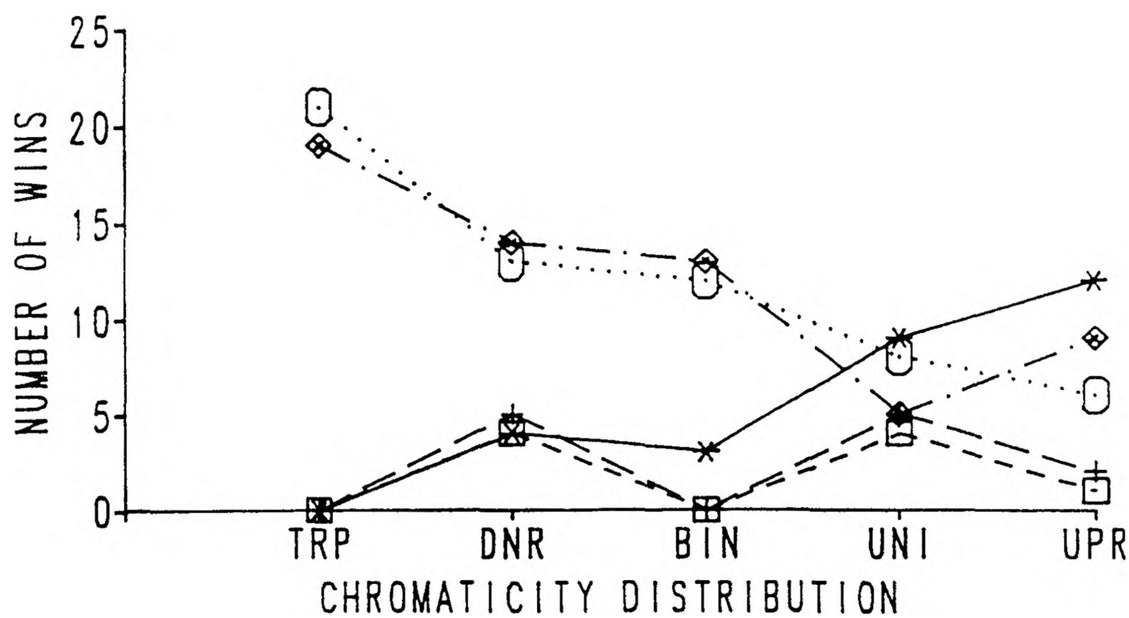


Figure 75. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with $n = 500$ and $\mu = 0.15$

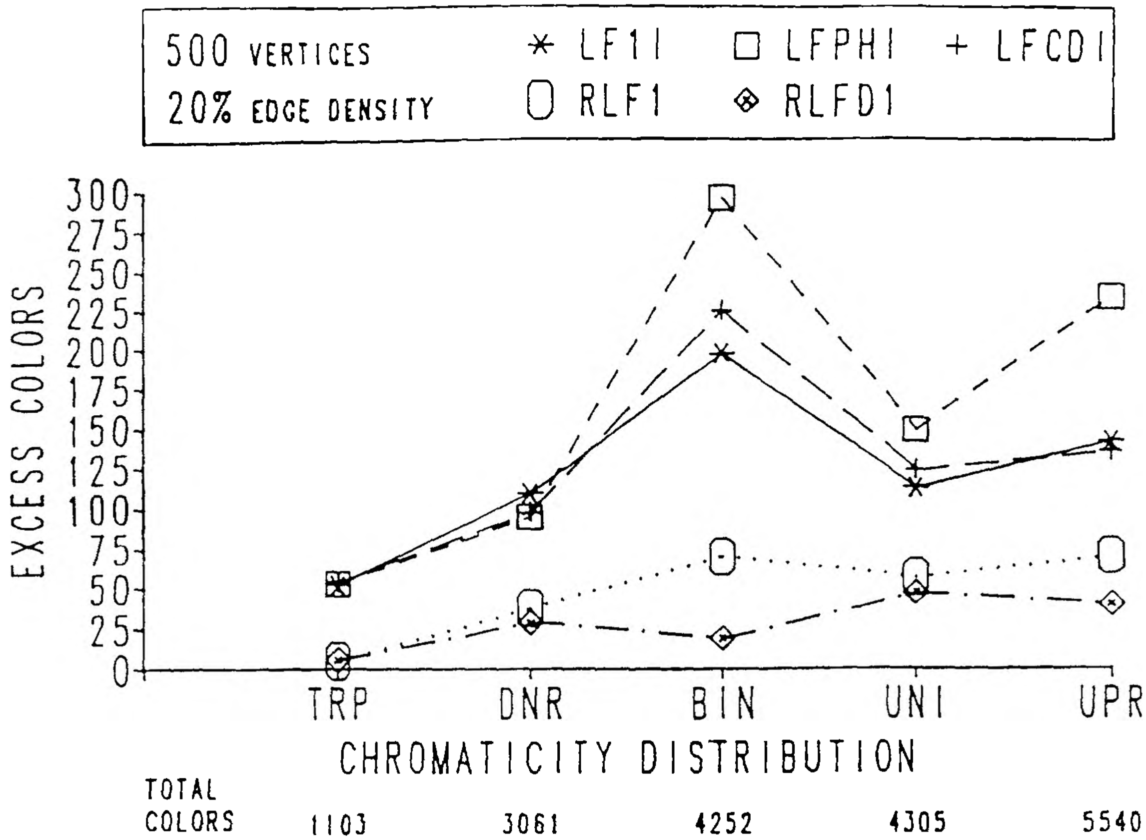


Figure 76. Number of Excess Colors vs. Chromaticity Distribution for Random Composite Graphs with n = 500 and μ = 0.20

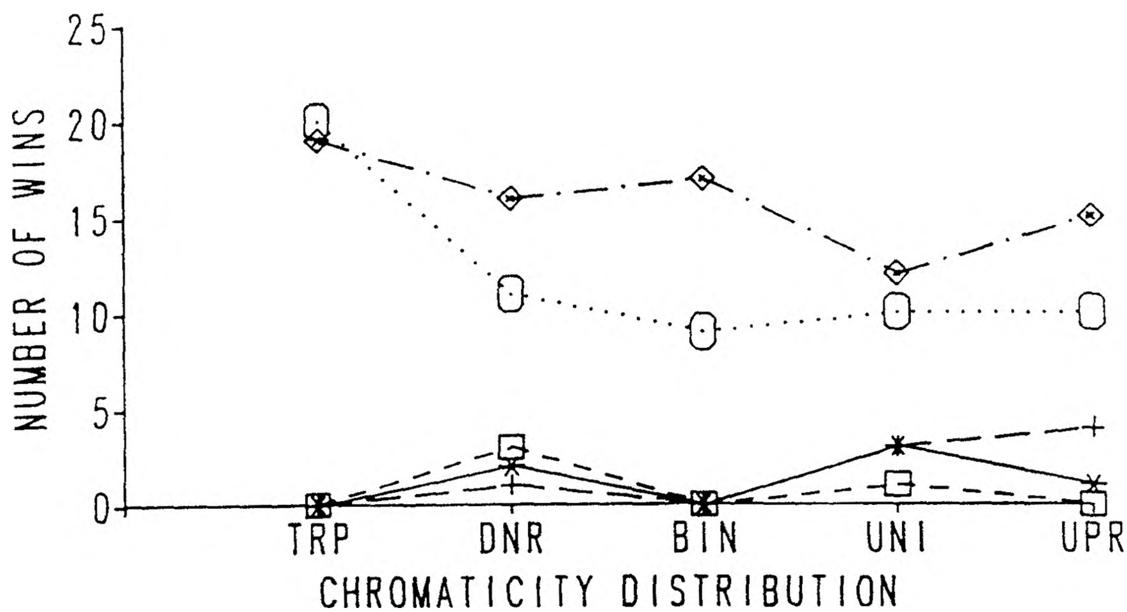


Figure 77. Number of Wins vs. Chromaticity Distribution for Random Composite Graphs with n = 500 and μ = 0.20

C. CONCLUSIONS

One rather obvious conclusion concerning the results of the experiments is that no one algorithm of the coloring algorithms investigated is superior to the other algorithms on all composite graphs. Since the CGCP is an NP-complete problem, this conclusion should not come as a complete surprise. Some of the coloring algorithms did show superiority over the other algorithms for certain groups of the random composite graphs.

By considering the fifteen pairs of graphs in Figures 2 through 31, the effects of changing the number of vertices in the random composite graphs upon the performance of the algorithms in Tier 1 can be seen. The LF1I, the LFPHI, and the LFCDI algorithms perform better for the lower numbers of vertices. The numbers of excess colors for the LF1I, the LFPHI, and the LFCDI algorithms tend to increase as the number of vertices increases. The numbers of wins for the LF1I, the LFPHI, and the LFCDI algorithms tend to decrease as the number of vertices increase. Opposite trends are seen for the RLF1 and the RLFD1 algorithms. As the number of vertices increase, the numbers of excess colors for the RLF1 and the RLFD1 algorithms tend to decrease and the numbers of wins tend to increase. These opposite trends cause the curves for the recursive largest-first algorithms, RLF1 and RLFD1, and the curves for the largest-first algorithms, LF1I, LFPHI, and LFCDI, to cross over each other in most of the graphs in Figures 2 through 31.

There is an indication that the RLF1 and the RLFD1 algorithms improve compared to the LF1I, the LFPHI, and the LFCDI algorithms as the edge density increases. For the experiments on random composite graphs of 100 vertices, this trend is readily apparent for some of the chromaticity distributions, for examples, see Figures 34 through 37. For the experiments on random composite graphs of a higher number of vertices, the trend can be seen by comparing three consecutive pairs of graphs with d as the independent variable (the graphs in Figures 54 through 77) which are for a particular number of vertices and edge densities μ = 0.10, 0.15, and 0.20. For example, compare the three pairs of graphs in Figure 54 through 59 for the experiments on random composite graphs having 200 vertices.

The effects of these trends observed for increasing the number of vertices and the edge density culminate in the near separation in Figures 70 and 71 and the separation in Figures 76 and 77 of the curves for the RLF1 and the RLFD1 algorithms from the curves for the LF1I, the LFPHI, and the LFCDI algorithms. In these instances, the RLF1 and the RLFD1 algorithms are superior to the LF1I, the LFPHI, and the LFCDI algorithms. In the graphs for a low number of vertices combined with a low edge density (for examples, see Figures 42 through 45), a similar separation of the curves for the RLF1 and the RLFD1 algorithms from the curves for the LF1I, the LFPHI, and the LFCDI algorithms can be observed, but in these instances the LF1I, the LFPHI, and the LFCDI algorithms are superior to the RLF1 and the RLFD1 algorithms.

If the edge density were allowed to approach 1, the coloring algorithms should produce colorings that use approximately the same number of colors. Indeed, any of the coloring algorithms should produce a coloring using a number of colors approximately equal to the sum of the chromaticities of the vertices of the composite graph for edge densities near to 1. The numbers of excess colors for the algorithms should tend toward 0 for high edge densities.

Concerning the effects of changing the chromaticity distributions, no clear trends in the performance of one algorithm with respect to another algorithm were observed. One effect of changing the chromaticity distribution that should be expected and that was evident in the results is that as the mean of the chromaticity distribution increases, the number of colors used by a coloring algorithm to color a a composite graph increases.

Due to the large amount of data collected as results of the experiments, not all the data could be included in this document. Summarizing the data as done in Tables II through XIX conceals some differences in the performances of two coloring algorithms on a particular composite graph. Some observations that are possible when inspecting the entire data set are not possible when inspecting the summarized data set.

The averages for the number of colors used by the algorithms in Tables II through XIX conceal differences between the algorithms on particular composite graphs. For example, Algorithm 1 may perform superior to Algorithm 2 on Graph A and Algorithm 2 may perform superior to Algorithm 1 on Graph B. For the 100 vertex random composite graphs, the LF1I, the LFPHI, or the LFCDI algorithm produced the coloring using the least number of colors for most of these graphs. It was not uncommon for there to be a 5 to 10% difference between the approximations of the chromatic number by two of these algorithms. For several graphs, only one of the three algorithms produced the lowest approximation of the chromatic number. In a particular application, a 5 to 10% improvement in the approximation of the chromatic number produced by one of the algorithms could be significant. In such a situation, it could be worthwhile to apply the other two algorithms to the composite graph. The three algorithms can be implemented without requiring three significantly different procedures. A general VSI algorithm can be implemented that arranges the vertices of the composite graph to be colored in decreasing order according to a measure associated with each vertex. An appropriate measure for a vertex v of a composite graph $G = \langle V, E, \Phi, C \rangle$ is given below for each algorithm.

Algorithm	Measure
LF1I	$\Delta_{max}\ C(v) + \Delta(v)$ where $\Delta_{max} = \max\ \{\Delta(u) : u \in V\}$
LFPHI	$\Delta(v) + (C(v) - 1)\ d(v) - 1$
LFCDI	$C(v)\ d(v)$

Though the five coloring algorithms in Tier 1 show superiorities for certain groups of random composite graphs, the number of excess colors used by an algorithm in most cases did not exceed 10% of the number of colors used by the MIN algorithm. The cases in which the number of excess colors for an algorithm exceeded 10% of the number of colors used by the MIN algorithm were for the RLF1 and the RLFD1 algorithms coloring groups of random composite graphs with 100 or 200 vertices. Table XXI shows statistics for the number of excess colors used by each algorithm when considered as a percentage of the colors used by the MIN algorithm. For each experiment, the value

$$\frac{\text{number of excess colors}}{\text{total number of colors}} * 100\%$$

was calculated for each algorithm. The "number of excess colors" is the number of excess colors used by the algorithm and the "total number of colors" is the number of colors used by the MIN algorithm to color the group of random composite graphs. The statistics in Table XXI were calculated using the values for the experiments in which $\mu \in \{0.10, 0.15, 0.20\}$. For the rows indicated by (*), the statistics were calculated including also the values for the experiments in which $n = 100$ and $\mu \in \{0.30, 0.40, 0.50\}$.

TABLE XXI

STATISTICS FOR THE NUMBER OF EXCESS COLORS CONSIDERED AS A PERCENTAGE OF "TOTAL COLORS"

Number of Vertices	LF1I	LFPHI	LFCDI	RLF1	RLFD1
			Mean		
100 (*)	4.06	3.48	3.35	6.29	6.29
100	4.61	3.59	3.66	8.73	8.02
200	3.05	2.71	2.93	5.59	5.59
300	2.59	3.08	2.47	3.11	3.18
400	2.96	3.47	3.08	2.23	2.57
500	3.07	4.00	3.36	1.73	1.59
Overall	3.25	3.37	3.10	4.28	4.19
Overall (*)	3.30	3.37	3.09	4.21	4.25
			Median		
100 (*)	4.07	3.42	3.21	5.76	6.02
100	4.23	3.32	3.49	8.35	8.18
200	2.66	2.75	2.70	5.00	5.68
300	2.49	2.89	2.49	3.24	2.89
400	3.34	3.16	2.84	1.53	2.42
500	2.62	3.99	2.88	1.32	1.29
Overall	3.11	3.16	2.88	3.48	3.52
Overall (*)	3.25	3.24	2.93	3.52	3.69
			Standard Deviation		
100 (*)	1.12	0.89	0.92	3.14	2.43
100	1.02	0.96	1.15	2.46	1.90
200	1.19	0.85	0.99	2.45	2.13
300	0.77	0.70	0.72	1.63	1.80
400	1.08	0.97	1.02	1.65	1.43
500	1.23	1.24	1.17	1.09	0.99
Overall	1.26	1.03	1.07	3.21	2.86
Overall (*)	1.21	1.00	1.00	2.98	2.68

TABLE XXI

(continued)

Number of Vertices	LF1I	LFPHI	LFCDI	RLF1	RLFD1
			Minimum		
100 (*)	2.06	2.20	1.67	2.19	2.80
100	3.11	2.20	1.67	5.31	5.45
200	1.79	1.43	1.45	1.74	2.43
300	1.58	1.91	1.42	0.66	0.66
400	1.21	1.62	1.44	0.40	0.64
500	1.11	2.33	2.02	0.45	0.45
Overall	1.11	1.43	1.42	0.40	0.45
Overall (*)	1.11	1.43	1.42	0.40	0.45
			Maximum		
100 (*)	7.09	5.44	6.10	13.60	11.14
100	7.09	5.44	6.10	13.60	11.14
200	6.32	4.32	5.08	10.36	8.83
300	4.46	4.18	3.94	6.51	6.29
400	4.90	4.89	4.67	6.57	5.16
500	5.43	6.91	5.27	4.15	3.80
Overall	7.09	6.91	6.10	13.60	11.14
Overall (*)	7.09	6.91	6.10	13.60	11.14

Two comments concerning the results of the experiments should be noted. First, the random composite graphs used in the experiments may not be representative of composite graphs that arise from practical problems. So the conclusions that have arisen from these experiments may or may not be applicable to a composite graph arising from a particular practical application. Second, the approximations of the chromatic number produced by the coloring algorithms were compared to each other not the chromatic number. Since, at the present time, no practical method of finding the chromatic number of a composite graph having as many vertices as the graphs used in the experiments exists, no means are now available to compare the approximations of the chromatic number which are produced by the coloring algorithms to the chromatic number. No claims are made about the relative closeness of an approximation of the chromatic number produced by any of the coloring algorithms to the chromatic number.

X. FUTURE RESEARCH

Very few articles have appeared in the literature concerning the area of composite graph coloring. This area appears rich with opportunities for investigation. Some suggestions for future research in this area are briefly described.

The investigation of heuristic coloring algorithms for the CGCP should be continued. As was seen from the results described in the previous chapter, no one coloring algorithm investigated so far is superior for all composite graphs. A collection of "good" coloring algorithms may be necessary to achieve acceptable results for a variety of composite graphs.

The Dsatur coloring algorithm [11] for the GCP yielded good experimental results. A generalization of this algorithm for the CGCP might yield similar favorable results for the CGCP.

The only improvement technique for vertex-sequential coloring algorithms for the CGCP that has been reported in the literature is the interchange technique described by Clementson and Elphick [1]. One idea for an improvement technique that I intend to pursue is an improvement technique to widen an existing gap of colors that have not been assigned to vertices adjacent to the vertex currently being colored in order that the colors in the newly widened gap can be assigned to the current vertex.

Analysis of the worst case behavior and the average behavior of heuristic coloring algorithms for the CGCP on all composite graphs or classes of composite graphs is needed.

More practical lower and upper bounds on the chromatic number in the CGCP and the GCP would be helpful in assessing the performance of heuristic coloring algorithms.

Even though any exact coloring algorithm for the CGCP is probably an exponential-time algorithm, it would be of interest to find an exact algorithm for the CGCP that would complete in reasonable time for composite graphs which have a small number of vertices (say up to 100) and have vertices with low chromaticities. One major problem in developing a backtracking algorithm to solve the CGCP for such composite graphs is that some of the results that help to eliminate the investigation of unprofitable colorings of the graph for the GCP do not generalize to the CGCP.

A major application of composite graph coloring is scheduling. The coloring algorithms discussed tend to assign the lower colors to several vertices while assigning the high colors to few vertices. In a scheduling application, such a coloring corresponds to an unbalanced schedule in which several tasks are assigned to the early time periods and few tasks are assigned to the later time periods. To produce a coloring that corresponds to a more balanced schedule, an algorithm is needed that assigns each color to approximately the same number of vertices, that is, generates a balanced coloring of the graph, or that accepts a coloring of a composite graph and produces a new more balanced coloring of the composite graph.

1 A. PUNTER (1976) Systems for timetabling by computer based on graph colouring. Ph.D. thesis, C.N.A.A., Hatfield Polytechnic.

3 M.A.H. DEMPSTER, D.G. LETHRIDGE and A.M. ULPH (1975) School ——. Educ. Res. 18, 24-31.
italics Bold

BIBLIOGRAPHY

√ [1] Clementson, A. T. and Elphick, C. H. "Approximate Colouring Algorithms for Composite Graphs", Journal of the Operational Research Society. Vol. 34 (1983), No. 6, p. 503-509.

√ [2] Karp, R. M. "Reducibility Among Combinatorial Problems" in Complexity of Computer Computations. Miller, R. E., Thatcher, J. W. (editors), Plenum Press, New York, 1972.

[3] Johnson, D. S. "Worst Case Behavior of Graph Coloring Algorithms" in Proceedings of the Fifth Southeast Conference on Combinatorics, Graph Theory, and Computing. Hoffman, F., Kingsley, R. A., Levow, R. B., Mullin, R. C., Thomas, R. S. D. (editors), Utilitas Mathematica Publishing, Winnipeg, Canada, 1974, p. 513-527.

[4] Mitchem, J. "On various algorithms for estimating the chromatic number of a graph", The Computer Journal. Vol. 19 (1976), No. 2, p. 182-183.

[5] Tremblay, Jean-Paul, and Manohar, R. Discrete Mathematical Structures with Applications to Computer Science. McGraw-Hill, Inc., New York, 1975, p. 468-515.

[6] Matula, D. W., Marble, G., and Isaacson, J. "Graph Coloring Algorithms" in Graph Theory and Computing. Read, R. (editor), Academic Press, New York, 1972, p. 109-122.

√ [7] Leighton, Frank Thomson. "A Graph Coloring Algorithm for Large Scheduling Problems", Journal of Research of the National Bureau of Standards. Vol. 84 (1979), No. 6, p. 489-506

[8] Welsh, D. J. A. and Powell, M. B. "An upper bound for the chromatic number of a graph and its application to timetabling problems", The Computer Journal. Vol. 10 (1967), p. 85-86.

[9] Williams, M. R. "The Coloring of Very Large Graphs" in Combinatorial Structures and Their Applications - Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications (June, 1969), Guy, R., Hanan, H., Sauer, N., and Schonheim, J. (editors), Gordon and Breach, New York, 1970, p. 477-478.

[10] Carter, Michael W. "A Survey of Practical Applications of Examination Timetabling Algorithms", Operations Research. Vol. 34, No. 2 (March-April, 1986), p. 193-202.

[11] Brelaz, D. "New Methods to Color the Vertices of a Graph", Communications of the ACM. Vol. 22 (1979), p. 251-256.

[12] Wood, D. C. "A technique for coloring a graph applicable to large scale timetabling problems problems", The Computer Journal. Vol. 12 (1969), p. 317-319.

[13] Dutton, R. D. and Brigham, R. C. "A new graph colouring algorithm", The Computer Journal. Vol. 24 (1981), No. 1, p. 85-86.

[14] Schneider, A. A. "Classification Analysis of Heuristic Algorithms for Graph Coloring", Cybernetics. Vol. 20, No. 4 (July-August, 1984), p. 484-492.

[15] Korman, S. M. "The Graph-Coloring Problem" in Combinatorial Optimization. Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (editors), Wiley, New York, 1979, p. 211-235.

[16] Kubale, Marek and Jackowski, Bugoslaw. "A Generalized Implicit Enumeration Algorithm for Graph Coloring", Communications of the ACM. Vol. 28, No. 4 (April, 1985), p. 412-418.

[17] Brown, J. Randall. "Chromatic Scheduling and the Chromatic Number Problem", Management Science. Vol. 19, No. 4, Part I (December, 1972), p. 456-463.

[18] Christofides, Nicos. Graph Theory - An Algorithmic Approach. Academic Press, London, 1975, p. 30-78.

[19] Chaitin, G. J. "Register Allocation & Spilling via Graph Coloring", SIGPLAN Notices. Vol. 17, No. 6 (June, 1982), p. 98-105.

[20] Butler, C. and Matthews, L. R. "An application of graph colouring on the railways" in Applications of Combinatorics. Wilson, R. J. (editor), Shiva Publishing Limited, 1982, p. 19-28.

[21] Dempster, M. A. H., Lethbridge, D. G., and Ulph, A. M. "School Timetabling by Computer: a Technical History", Educational Research. Vol. 18 (1975), p. 24-31.

[22] Ambler, Arol and Trawick, Robert. "Chatin's Graph Coloring Algorithm as a Method for Assigning Positions to Diana Attributes", SIGPLAN Notices. Vol. 18, No. 2 (February, 1983), p. 37-38.

[23] Garey, M. R., Johnson, D. S., and So, H. C. "An Application of Graph Coloring to Printed Circuit Testing", IEEE Transactions on Circuits and Systems. Vol. 23 (1976), p. 591-599.

[24] Coleman, Thomas F. and Moré, Jorge J. "Estimation of Sparse Jacobian Matrices and Graph Coloring Problems", SIAM Journal of Numerical Analysis. Vol. 20, No. 1 (February, 1983), p. 187-209.

[25] Bondy, J. A. and Murty, U. S. R. Graph Theory with Applications. North Holland, 1976, p. 91-134.

[26] Behzad, Mehdi. "The Total Chromatic Number of a Graph: A Survey" in Combinatorial Mathematics and Its Applications. Welsh, D. J. A. (editor), Academic Press, New York, 1971, p. 1-8.

[27] Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. "Recent Developments in Deterministic Sequencing and Scheduling: A Survey" in Deterministic and Stochastic Scheduling. Dempster, M. A. H. and others (editors), D. Riedel Publishing Company, Amsterdam, 1982, p. 35-73.

[28] Tucker, Alan. Applied Combinatorics. John Wiley & Sons, Inc., New York, 1980, p. 15-19.

[29] Dantzig, George B. Linear Programming and Extensions. Princeton University Press, Princeton, New Jersey, 1963, p. 535-540.

APPENDIX A

INTEGER PROGRAMMING FORMULATION OF THE COMPOSITE GRAPH COLORING PROBLEM

The following integer programming formulation of the composite graph coloring problem provides a means of finding an exact solution of the CGCP. Using a general-purpose ILP solver, only instances of the CGCP involving very small composite graphs can be expected to complete in a reasonable amount of time. By means of relaxation techniques or some ad hoc technique, problems involving larger composite graphs might be solved. Further research needs to be done in this area.

For the formulation, let us assume that the vertices of the composite graph to be colored are numbered from 1 to n. In the definition of the CGCP, the lowest color that can be assigned to a vertex is the integer 1. In the integer programming formulation of the CGCP, the integers to be assigned to the vertices begin at 0. Listed below are constants and variables that are to be used in our derivation of the formulation.

I = index set for the vertices

$= \{1, 2, 3, \ldots, n\}$

$J_i = \{j$: vertex j is adjacent to vertex i and $j < i\}$

c_i = number of consecutive integers assigned to vertex i

x_i = lowest integer assigned to vertex i

y_i = highest integer assigned to vertex i

z = largest integer to be assigned to any vertex

From the definition of the CGCP, the following mathematical programming problem can be obtained. The choice of either inequality (2a) or (2b) arises from the requirement that the color sequences assigned to adjacent vertices cannot overlap.

minimize z

subject to:

$$y_i \leq z \qquad i \in I$$

$$y_i - x_i = c_i - 1 \qquad i \in I \qquad (1)$$

$$\begin{cases} x_j \geq y_i + 1 & (2a) \\ \quad \text{or} & j \in J_i,\ i \in I \\ x_i \geq y_j + 1 & (2b) \end{cases}$$

$$x_i \geq 0,\ y_i \geq 0 \text{ integers} \qquad i \in I$$

Let K be a known upper bound on the chromatic number of the composite graph. For each pair of inequalities (2a) and (2b), the choice of one of the two inequalities can be implemented by introducing decision variables δ_{ij} [29]. If $\delta_{ij} = 0$, then inequality (2a) is chosen. If $\delta_{ij} = 1$, then inequality (2b) is chosen. For any optimal solution, $0 \leq x_i \leq K - 1$ and $0 \leq y_i \leq K - 1$ for each $i \in I$. So, for any optimal solution, $x_j - y_i - 1 \geq -K$ and $x_i - y_j - 1 \geq -K$ for each $j \in J_i$, $i \in I$. The choice of one of the two inequalities (2a) and (2b) can be replaced by the following two inequalities:

$$x_j - y_i - 1 - \delta_{ij}(-K) \geq 0 \qquad (3a)$$

$$x_i - y_j - 1 - (1 - \delta_{ij})(-K) \geq 0 \qquad (3b)$$

If $\delta_{ij} = 0$, inequality (3a) is equivalent to inequality (2a) and the values of x_i and y_j from any optimal solution will satisfy inequality (3b). If $\delta_{ij} = 1$, inequality (3b) is equivalent to inequality (2b) and the values of x_j and y_i from any optimal solution will satisfy inequality (3a). Simplifying inequalities (3a) and (3b) yields

$$x_j - y_i + K\,\delta_{ij} \geq 1$$

$$x_i - y_j - K\,\delta_{ij} \geq -K + 1$$

For each $i \in I$, either x_i or y_i can be eliminated from the model by using equality (1). Substituting the relationship $x_i = y_i - c_i + 1$ obtained from equality (1) yields

$$(y_j - c_j + 1) - y_i + K\,\delta_{ij} \geq 1$$

$$(y_i - c_i + 1) - y_j - K\,\delta_{ij} \geq -K + 1$$

The composite graph coloring problem can be formulated as the following integer programming problem:

minimize z

subject to:

$$y_i \leq z \qquad i \in I$$

$$y_j - y_i + K\,\delta_{ij} \geq c_j \qquad j \in J_i,\ i \in I$$

$$y_j - y_i + K\,\delta_{ij} \leq K - c_i \qquad j \in J_i,\ i \in I$$

$$y_i \geq 0 \text{ integers} \qquad i \in I$$

$$\delta_{ij} \in \{0,1\} \qquad j \in J_i,\ i \in I$$

Let us assume that an optimal solution of the integer programming problem has been found in which y_i^* is the value for y_i for each $i \in I$ and z^* is the value of the objective function. The corresponding coloring of the composite graph is obtained by assigning vertex i the color sequence $I[y_i^* - c_i + 2, y_i^* + 1]$ for each $i \in I$ and the chromatic number of the composite graph is $z^* + 1$.

APPENDIX B

PROCEDURE LISTINGS

```
TABLES: PROC OPTIONS(MAIN) REORDER;
  DCL
    ALGORITHM_MIN            FIXED BINARY(15),
    AUX_HEAD_PTR             POINTER,
    AVG_WIN_DIFF             FLOAT BINARY(21),
    BOTTOM_MARGIN            FIXED BINARY(15) INIT(9),
    CDSORT                   ENTRY,
    CHROM_DISTRIBUTION       CHARACTER(5),
    CREGRAF                  ENTRY,
    DYNPH                    ENTRY,
    DYNFPH                   ENTRY,
    EDGE_DENSITY             FLOAT BINARY(21),
    FLOAT                    BUILTIN,
    FREGRAF                  ENTRY,
    GRAPH_MAX                FIXED BINARY(15),
    GRAPH_MIN                FIXED BINARY(15),
    GRAPH_NO                 FIXED BINARY(15) INIT(0),
    GROUP_NO                 FIXED BINARY(15) INIT(0),
    HEAD_PTR                 POINTER,
    LAMBDA                   FLOAT BINARY(21),
    LF1SORT                  ENTRY,
    LF2SORT                  ENTRY,
    LOWER_LIMIT              FIXED BINARY(15),
    MAX                      BUILTIN,
    MAX_COLOR                FIXED BINARY(15),
    MAXIMUM(13)              FIXED BINARY(15),
    MEAN(13)                 FLOAT BINARY(21),
    METHOD_MIN               FIXED BINARY(15),
    METHOD_NO                FIXED BINARY(15),
    METHOD_NAME(13)          CHARACTER(6) VARYING
      INIT('LF1','LF2','LFPH','LFCD','LF1I','LF2I','LFPHI',
      'LFCDI','RLF1','RLFD1','DYNPH','DYNFPH','MIN'),
    MIN                      BUILTIN,
    MINIMUM(13)              FIXED BINARY(15),
    NEW_SEED                 FIXED BINARY(31),
    NO_OF_COLORS             FIXED BINARY(15),
    NO_OF_GRAPHS             FIXED BINARY(15) INIT(0),
    NO_OF_NODES              FIXED BINARY(15),
    NO_OF_TRIALS             FIXED BINARY(15),
    NO_OF_WINS(13,13)        FIXED BINARY(15),
    NULL                     BUILTIN,
    OTHER_METHOD             FIXED BINARY(15),
    PHSORT                   ENTRY,
    PROB_OF_SUCCESS          FLOAT BINARY(31),
    RLFD1                    ENTRY,
    RLF1                     ENTRY,
    SEED                     FIXED BINARY(31),
    SEQCOL                   ENTRY,
    SEQINT                   ENTRY,
```

```
   SUM_OF_COLORS             FIXED BINARY(31),
   SYSPRINT                  FILE STREAM OUTPUT PRINT,
   TOTAL_WIN_DIFF(13,13)     FIXED BINARY(31),
   UPPER_LIMIT               FIXED BINARY(15);

/* GENERATE THE FIRST GRAPH OF THE FIRST GROUP.                    */

  CALL CREGRAF(HEAD_PTR,NO_OF_NODES,EDGE_DENSITY,NEW_SEED,
   GROUP_NO,GRAPH_NO,NO_OF_GRAPHS,CHROM_DISTRIBUTION,
   LAMBDA,LOWER_LIMIT,UPPER_LIMIT,NO_OF_TRIALS,
   PROB_OF_SUCCESS);

/* PRINT TABLES UNTIL THERE ARE NO MORE GRAPHS.                    */

  DO WHILE(HEAD_PTR¬=NULL);
   SEED=NEW_SEED;

/* ACCUMULATE STATISTICS FOR ONE GROUP OF GRAPHS.                  */

  BEGIN;
   DCL TABLE(13,NO_OF_GRAPHS) FIXED BINARY(15);
   DO WHILE('1'B);

/* COLOR GRAPH USING LF1 ALGORITHM.                                */

     CALL LF1SORT(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES);
     CALL SEQCOL(AUX_HEAD_PTR,MAX_COLOR);
     TABLE(1,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING LF1I ALGORITHM.                               */

     CALL SEQINT(AUX_HEAD_PTR,MAX_COLOR);
     TABLE(5,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING LF2 ALGORITHM.                                */

     CALL LF2SORT(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES);
     CALL SEQCOL(AUX_HEAD_PTR,MAX_COLOR);
     TABLE(2,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING LF2I ALGORITHM.                               */

     CALL SEQINT(AUX_HEAD_PTR,MAX_COLOR);
     TABLE(6,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING LFPH ALGORITHM.                               */

     CALL PHSORT(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES);
     CALL SEQCOL(AUX_HEAD_PTR,MAX_COLOR);
     TABLE(3,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING LFPHI ALGORITHM.                              */

     CALL SEQINT(AUX_HEAD_PTR,MAX_COLOR);
```

```
        TABLE(7,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING LFCD ALGORITHM.                          */

        CALL CDSORT(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES);
        CALL SEQCOL(AUX_HEAD_PTR,MAX_COLOR);
        TABLE(4,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING LFCDI ALGORITHM.                         */

        CALL SEQINT(AUX_HEAD_PTR,MAX_COLOR);
        TABLE(8,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING RLF1 ALGORITHM.                          */

        CALL RLF1(HEAD_PTR,AUX_HEAD_PTR,MAX_COLOR);
        TABLE(9,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING RLFD1 ALGORITHM.                         */

        CALL RLFD1(HEAD_PTR,AUX_HEAD_PTR,MAX_COLOR);
        TABLE(10,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING DYNPH ALGORITHM.                         */

        CALL DYNPH(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES,
          MAX_COLOR);
        TABLE(11,GRAPH_NO)=MAX_COLOR;

/* COLOR GRAPH USING DYNFPH ALGORITHM.                        */

        CALL DYNFPH(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES,
          MAX_COLOR);
        TABLE(12,GRAPH_NO)=MAX_COLOR;

/* FREE THE STORAGE USED FOR THE GRAPH.                       */
        CALL FREGRAF(HEAD_PTR);

/* TEST FOR THE END OF A GROUP OF GRAPHS.                     */

        IF GRAPH_NO=NO_OF_GRAPHS
          THEN LEAVE;

/* GENERATE ANOTHER GRAPH IN THE GROUP.                       */

        CALL CREGRAF(HEAD_PTR,NO_OF_NODES,EDGE_DENSITY,
          NEW_SEED,GROUP_NO,GRAPH_NO,NO_OF_GRAPHS,
          CHROM_DISTRIBUTION,LAMBDA,LOWER_LIMIT,UPPER_LIMIT,
          NO_OF_TRIALS,PROB_OF_SUCCESS);
    END;

/* PRINT TABLE FOR THE GROUP OF GRAPHS.                       */
```

```
    DO GRAPH_NO=1 TO NO_OF_GRAPHS;
      METHOD_MIN=TABLE(1,GRAPH_NO);
      DO METHOD_NO=2 TO 12;
        METHOD_MIN=
          MIN(METHOD_MIN,TABLE(METHOD_NO,GRAPH_NO));
      END;
      TABLE(13,GRAPH_NO)=METHOD_MIN;
    END;
    DO METHOD_NO=1 TO 13;
      NO_OF_COLORS=TABLE(METHOD_NO,1);
      GRAPH_MIN=NO_OF_COLORS;
      GRAPH_MAX=NO_OF_COLORS;
      SUM_OF_COLORS=NO_OF_COLORS;
      DO GRAPH_NO=2 TO NO_OF_GRAPHS;
        NO_OF_COLORS=TABLE(METHOD_NO,GRAPH_NO);
        GRAPH_MIN=MIN(GRAPH_MIN,NO_OF_COLORS);
        GRAPH_MAX=MAX(GRAPH_MAX,NO_OF_COLORS);
        SUM_OF_COLORS=SUM_OF_COLORS+NO_OF_COLORS;
      END;
      MINIMUM(METHOD_NO)=GRAPH_MIN;
      MAXIMUM(METHOD_NO)=GRAPH_MAX;
      MEAN(METHOD_NO)=FLOAT(SUM_OF_COLORS,21)/
        FLOAT(NO_OF_GRAPHS,21);
    END;

/* PRINT THE HEADING FOR THE TABLE.                              */

    PUT FILE(SYSPRINT) PAGE;
    CALL TABHDG;

    PUT FILE(SYSPRINT) EDIT('GRAPH    '||(35)'*'||
      '  ESTIMATE OF '||'CHROMATIC NUMBER  '||(35)'*')
      (SKIP,A);
    PUT FILE(SYSPRINT) EDIT('NUMBER',
      (CENTER(METHOD_NAME(METHOD_NO),6)
      DO METHOD_NO=1 TO 13))(SKIP,A,(13)(X(2),A));
    PUT FILE(SYSPRINT) EDIT((14)'------  ')(SKIP,COL(1),A);
    DO GRAPH_NO=1 TO NO_OF_GRAPHS;
      ALGORITHM_MIN=TABLE(13,GRAPH_NO);
      PUT FILE(SYSPRINT) EDIT(GRAPH_NO,'   ')
        (COL(2),F(3),A);
      DO METHOD_NO=1 TO 12;
        IF TABLE(METHOD_NO,GRAPH_NO) > ALGORITHM_MIN
          THEN DO;
            PUT FILE(SYSPRINT) EDIT
              (TABLE(METHOD_NO,GRAPH_NO),'   ')
              (X(2),F(4),A);
          END;
          ELSE DO;
            PUT FILE(SYSPRINT) EDIT
              (TABLE(METHOD_NO,GRAPH_NO),' *')
              (X(2),F(4),A);
          END;
      END;
```

```
      PUT FILE(SYSPRINT) EDIT(ALGORITHM_MIN)(X(2),F(4));
    END;
    PUT FILE(SYSPRINT) EDIT('MINIMUM',
      (MINIMUM(METHOD_NO) DO METHOD_NO=1 TO 13))
      (SKIP(2),A,X(2),(13)(F(3),X(5)));
    PUT FILE(SYSPRINT) EDIT('MEAN',
      (MEAN(METHOD_NO) DO METHOD_NO=1 TO 13))
      (SKIP,A,X(5),(13)(F(6,2),X(2)));
    PUT FILE(SYSPRINT) EDIT('MAXIMUM',
      (MAXIMUM(METHOD_NO) DO METHOD_NO=1 TO 13))
      (SKIP,A,X(2),(13)(F(3),X(5)));

/* PRINT ALGORITHM COMPARISON TABLE.                             */

    NO_OF_WINS(*,*)=0;
    TOTAL_WIN_DIFF(*,*)=0;
    DO METHOD_NO=1 TO 13;
      DO OTHER_METHOD=METHOD_NO+1 TO 13;
        IF METHOD_NO=OTHER_METHOD THEN GO TO NEXT_METHOD;
        DO GRAPH_NO=1 TO NO_OF_GRAPHS;
          SELECT;
            WHEN(TABLE(METHOD_NO,GRAPH_NO) <
              TABLE(OTHER_METHOD,GRAPH_NO)) DO;
              NO_OF_WINS(METHOD_NO,OTHER_METHOD)=
                NO_OF_WINS(METHOD_NO,OTHER_METHOD)+1;
              TOTAL_WIN_DIFF(METHOD_NO,OTHER_METHOD)=
                TOTAL_WIN_DIFF(METHOD_NO,OTHER_METHOD)+
                TABLE(OTHER_METHOD,GRAPH_NO)-
                TABLE(METHOD_NO,GRAPH_NO);
            END;
            WHEN(TABLE(METHOD_NO,GRAPH_NO) >
              TABLE(OTHER_METHOD,GRAPH_NO)) DO;
              NO_OF_WINS(OTHER_METHOD,METHOD_NO)=
                NO_OF_WINS(OTHER_METHOD,METHOD_NO)+1;
              TOTAL_WIN_DIFF(OTHER_METHOD,METHOD_NO)=
                TOTAL_WIN_DIFF(OTHER_METHOD,METHOD_NO)+
                TABLE(METHOD_NO,GRAPH_NO)-
                TABLE(OTHER_METHOD,GRAPH_NO);
            END;
            OTHERWISE;
          END; /* SELECT */
        END;
NEXT_METHOD:
      END;
    END;
    CALL ALGCOMP(1,4);
    CALL ALGCOMP(5,13);

/* GENERATE FIRST GRAPH OF A GROUP.                              */

    GRAPH_NO=NO_OF_GRAPHS;
```

```
    CALL CREGRAF(HEAD_PTR,NO_OF_NODES,EDGE_DENSITY,
      NEW_SEED,GROUP_NO,GRAPH_NO,NO_OF_GRAPHS,
      CHROM_DISTRIBUTION,LAMBDA,LOWER_LIMIT,UPPER_LIMIT,
      NO_OF_TRIALS,PROB_OF_SUCCESS);
  END;

/* PROCEDURE TO PRINT ONE PAGE OF THE ALGORITHM              */
/* COMPARISON TABLE                                          */

ALGCOMP: PROC(FIRST_METHOD,LAST_METHOD);
  DCL (FIRST_METHOD,LAST_METHOD) FIXED BINARY(15);
  PUT FILE(SYSPRINT) PAGE;
  CALL TABHDG;

/* PRINT ONE PAGE OF ALGORITHM COMPARISON TABLE.             */

  PUT FILE(SYSPRINT) EDIT((CENTER(METHOD_NAME(METHOD_NO),9)
    DO METHOD_NO=FIRST_METHOD TO LAST_METHOD))
    (SKIP,X(6),(9)(X(3),A));
  PUT FILE(SYSPRINT) EDIT(('   ---------'
    DO METHOD_NO=FIRST_METHOD TO LAST_METHOD))
    (SKIP,X(6),(9)A);
  DO METHOD_NO = 1 TO 13;
    PUT FILE(SYSPRINT) EDIT(METHOD_NAME(METHOD_NO))
      (SKIP(2),A(6));
    DO OTHER_METHOD=FIRST_METHOD TO LAST_METHOD;
      IF METHOD_NO=OTHER_METHOD
        THEN DO;
          PUT FILE(SYSPRINT) EDIT('            ')(A);
          GO TO NEXT_COLUMN;
        END;
      PUT FILE(SYSPRINT) EDIT
        (NO_OF_WINS(METHOD_NO,OTHER_METHOD))(X(3),F(2));
      IF NO_OF_WINS(METHOD_NO,OTHER_METHOD) > 0
        THEN DO;
          AVG_WIN_DIFF=
           FLOAT(TOTAL_WIN_DIFF(METHOD_NO,OTHER_METHOD),21)
           /FLOAT(NO_OF_WINS(METHOD_NO,OTHER_METHOD),21);
          PUT FILE(SYSPRINT) EDIT(AVG_WIN_DIFF)
            (X(2),F(5,2));
        END;
        ELSE DO;
          PUT FILE(SYSPRINT) EDIT('       ')(A);
        END;
NEXT_COLUMN:
    END;
    PUT FILE(SYSPRINT) EDIT('      ')(SKIP,A);
    DO OTHER_METHOD=FIRST_METHOD TO LAST_METHOD;
      IF METHOD_NO=OTHER_METHOD
        THEN DO;
          PUT FILE(SYSPRINT) EDIT('            ')(A);
          GO TO NEXT_COLUMN_2;
        END;
```

```
      PUT FILE(SYSPRINT) EDIT
        (NO_OF_GRAPHS-NO_OF_WINS(OTHER_METHOD,METHOD_NO),
        MEAN(OTHER_METHOD)-MEAN(METHOD_NO))
        (X(3),F(2),X(1),F(6,2));
NEXT_COLUMN_2:
    END;
  END;
END ALGCOMP;
  END; /* BEGIN */

/* PROCEDURE TO PRINT TABLE HEADINGS.                              */

TABHDG: PROC;
  DCL
    STRING_ONE              CHARACTER(5) VARYING,
    STRING_TWO              CHARACTER(5) VARYING,
    OUTPUT_LINE             CHARACTER(132) VARYING;
  PUT FILE(SYSPRINT) EDIT('NUMBER OF NODES:',NO_OF_NODES,
    'EDGE DENSITY: ',EDGE_DENSITY*100.0,'%     SEED: ',
    SEED)(COL(1),A,F(5),X(5),A,F(5,2),A,F(10));
  PUT FILE(SYSPRINT) EDIT('CHROMATICITY DISTRIBUTION:  ')
    (SKIP(2),A);
  SELECT(CHROM_DISTRIBUTION);
    WHEN('TRP  ') DO;
      PUT STRING(STRING_ONE) EDIT(LAMBDA)(F(4,1));
      OUTPUT_LINE='TRUNCATED POISSON (LAMBDA = '||
        DEBLANK(STRING_ONE)||')';
    END;
    WHEN('UNI  ') DO;
      PUT STRING(STRING_ONE) EDIT(LOWER_LIMIT)(F(5));
      PUT STRING(STRING_TWO) EDIT(UPPER_LIMIT)(F(5));
      OUTPUT_LINE='UNIFORM ('||DEBLANK(STRING_ONE)||
        ','||DEBLANK(STRING_TWO)||')';
    END;
    WHEN('DNR  ') DO;
      PUT STRING(STRING_ONE) EDIT(LOWER_LIMIT)(F(5));
      PUT STRING(STRING_TWO) EDIT(UPPER_LIMIT)(F(5));
      OUTPUT_LINE='DOWN RAMP ('||DEBLANK(STRING_ONE)||
        ','||DEBLANK(STRING_TWO)||')';
    END;
    WHEN('UPR  ') DO;
      PUT STRING(STRING_ONE) EDIT(LOWER_LIMIT)(F(5));
      PUT STRING(STRING_TWO) EDIT(UPPER_LIMIT)(F(5));
      OUTPUT_LINE='UP RAMP ('||DEBLANK(STRING_ONE)||
        ','||DEBLANK(STRING_TWO)||')';
    END;
    WHEN('BIN  ') DO;
      PUT STRING(STRING_ONE) EDIT(NO_OF_TRIALS)(F(5));
      PUT STRING(STRING_TWO) EDIT(LOWER_LIMIT)(F(5));
      PUT STRING(OUTPUT_LINE) EDIT(
        'SHIFTED BINOMIAL (N = ',DEBLANK(STRING_ONE),
        ', P = ',PROB_OF_SUCCESS,', SHIFT = ',
        DEBLANK(STRING_TWO),')')(A,A,A,F(6,4),A,A,A);
    END;
```

```
    OTHERWISE SIGNAL ERROR;
  END; /* SELECT */
  PUT FILE(SYSPRINT) EDIT(OUTPUT_LINE)(A);
  PUT FILE(SYSPRINT) SKIP;
END TABHDG;

/* PROCEDURE TO CENTER A CHARACTER STRING.                        */

CENTER: PROC(CHAR_STRING,CENTERED_LENGTH)
  RETURNS(CHARACTER(132) VARYING);
  DCL
    CENTERED_LENGTH            FIXED BINARY(15),
    CENTERED_STRING            CHARACTER(CENTERED_LENGTH),
    CHAR_STRING                CHARACTER(*) VARYING,
    LEFT_SPACES                FIXED BINARY(15),
    LENGTH                     BUILTIN,
    NO_OF_SPACES               FIXED BINARY(15);
  NO_OF_SPACES=CENTERED_LENGTH-LENGTH(CHAR_STRING);
  LEFT_SPACES=NO_OF_SPACES/2;
  PUT STRING(CENTERED_STRING) EDIT(CHAR_STRING)
    (X(LEFT_SPACES),A,X(NO_OF_SPACES-LEFT_SPACES));
  RETURN(CENTERED_STRING);
END CENTER;

/* PROCEDURE TO REMOVE LEADING BLANKS FROM A CHARACTER            */
/* STRING.                                                        */

DEBLANK: PROC(CHAR_STRING) RETURNS(CHARACTER(15) VARYING);
  DCL
    CHAR_STRING                CHARACTER(*) VARYING,
    SUBSTR                     BUILTIN,
    VERIFY                     BUILTIN;
  RETURN(SUBSTR(CHAR_STRING,VERIFY(CHAR_STRING,' ')));
END DEBLANK;
END TABLES;
```

```
CREGRAF: PROC(HEAD_PTR,NO_OF_NODES,EDGE_DENSITY,NEW_SEED,
  GROUP_NO,GRAPH_NO,NO_OF_GRAPHS,CHROM_DISTRIBUTION,LAMBDA,
  LOWER_LIMIT,UPPER_LIMIT,NO_OF_TRIALS,PROB_OF_SUCCESS)
  REORDER;
  DCL
    ACCUM_DEGREE             FIXED BINARY(15),
    CHROM_DEG                FIXED BINARY(15),
    CHROM_DISTRIBUTION       CHARACTER(5),
    DEG                      FIXED BINARY(15),
    EDGE_DENSITY             FLOAT BINARY(21),
    FLOOR                    BUILTIN,
    GRAPH_NO                 FIXED BINARY(15),
    GROUP_NO                 FIXED BINARY(15),
    HEAD_PTR                 POINTER,
    J                        FIXED BINARY(15),
    LAMBDA                   FLOAT BINARY(21),
    LOWER_LIMIT              FIXED BINARY(15),
    MORE_DATA                BIT(1) STATIC INIT('1'B),
    NEW_SEED                 FIXED BINARY(31),
    NO_OF_GRAPHS             FIXED BINARY(15),
    NO_OF_NODES              FIXED BINARY(15),
    NO_OF_TRIALS             FIXED BINARY(15),
    NODE_NO                  FIXED BINARY(15),
    NODE_PTR                 POINTER,
    NULL                     BUILTIN,
    POISSON                  ENTRY
                               RETURNS(FIXED BINARY(31)),
    PROB_OF_SUCCESS          FLOAT BINARY(31),
    RANDOM_NUMBER            FLOAT BINARY(21),
    RANDU                    ENTRY,
    SEED                     FIXED BINARY(31) STATIC
                               INIT(17),
    SUCCESS_COUNT            FIXED BINARY(15),
    SYSIN                    FILE STREAM INPUT,
    TEMP_PTR                 POINTER,
    TRIAL_NO                 FIXED BINARY(15),
    UPPER_LIMIT              FIXED BINARY(15),
    WIDTH                    FLOAT BINARY(31);

  DCL
    1 NODE                   BASED(NODE_PTR),
      2 NUMBER               FIXED BINARY(15),
      2 DEGREE               FIXED BINARY(15),
      2 CHROMATICITY         FIXED BINARY(15),
      2 CHROMATIC_DEGREE     FIXED BINARY(15),
      2 LO_COLOR             FIXED BINARY(15),
      2 HI_COLOR             FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,   /* 40 BYTES */
        3 FILLER             CHAR(40),
      2 ADJ_NODE_PTR(0:DEG REFER(DEGREE)) PTR;

  ON ENDFILE(SYSIN) MORE_DATA='0'B;
  MORE_DATA='1'B;
```

```
/* GET DATA FOR A GROUP OF GRAPHS.                                        */

  IF GRAPH_NO=NO_OF_GRAPHS
    THEN DO;
      GET FILE(SYSIN) EDIT(NO_OF_NODES,EDGE_DENSITY,
        NEW_SEED,NO_OF_GRAPHS)
        (COL(2),F(4),F(5,3),F(10),X(2),F(3));
      IF ¬MORE_DATA
        THEN DO;
          HEAD_PTR=NULL;
          RETURN;
        END;
      GET FILE(SYSIN) EDIT(CHROM_DISTRIBUTION)
        (COL(1),A(5));
      SELECT(CHROM_DISTRIBUTION);
        WHEN('TRP  ')
          GET FILE(SYSIN) EDIT(LAMBDA)(X(1),F(4,1));
        WHEN('UNI  ','DNR  ','UPR  ')
          GET FILE(SYSIN) EDIT(LOWER_LIMIT,UPPER_LIMIT)
            ((2)F(5));
        WHEN('BIN  ')
          GET FILE(SYSIN) EDIT(NO_OF_TRIALS,
            PROB_OF_SUCCESS,LOWER_LIMIT)(F(5),F(5,4),F(5));
        OTHERWISE SIGNAL ERROR;
      END; /* SELECT */
      IF NO_OF_GRAPHS=0 THEN NO_OF_GRAPHS=1;
      IF NEW_SEED¬=0 THEN SEED=NEW_SEED;
      GROUP_NO=GROUP_NO+1;
      GRAPH_NO=0;
    END;

/* GENERATE ONE GRAPH.                                                    */

GENERATE_GRAPH: BEGIN;
  DCL
    FULL_DEGREE(NO_OF_NODES)  FIXED BINARY(15),
    EDGES(NO_OF_NODES,NO_OF_NODES)  BIT(1),
    LOCATOR(NO_OF_NODES)    POINTER;

  GRAPH_NO=GRAPH_NO+1;
  NEW_SEED=SEED;

/* GENERATE EDGES.                                                        */
/* CALCULATE THE DEGREE OF EACH NODE.                                     */

  EDGES(*,*)='0'B;
  FULL_DEGREE(*)=0;

  DO NODE_NO=2 TO NO_OF_NODES;
    DO J=1 TO NODE_NO-1;
      CALL RANDU(SEED,RANDOM_NUMBER);
      IF RANDOM_NUMBER <= EDGE_DENSITY
        THEN DO;
          FULL_DEGREE(NODE_NO)=FULL_DEGREE(NODE_NO)+1;
```

```
        FULL_DEGREE(J)=FULL_DEGREE(J)+1;
        EDGES(NODE_NO,J)='1'B;
        EDGES(J,NODE_NO)='1'B;
      END;
    END;
  END;

/* ALLOCATE SPACE FOR EACH NODE.                                 */
/* INITIALIZE INFORMATION FOR EACH NODE.                         */

  DO NODE_NO=1 TO NO_OF_NODES;
    DEG=FULL_DEGREE(NODE_NO);
    ALLOCATE NODE;
    LOCATOR(NODE_NO)=NODE_PTR;
    NUMBER=NODE_NO;
    LO_COLOR=0;
    HI_COLOR=0;
  END;

/* CREATE LINKED LIST.                                           */
/* NOTE:  ADJ_NODE_PTR(0) IS THE LINK POINTER.                   */

  HEAD_PTR=LOCATOR(1);
  NODE_PTR=HEAD_PTR;
  DO NODE_NO=2 TO NO_OF_NODES;
    TEMP_PTR=LOCATOR(NODE_NO);
    ADJ_NODE_PTR(0)=TEMP_PTR;
    NODE_PTR=TEMP_PTR;
  END;
  ADJ_NODE_PTR(0)=NULL;

/* DETERMINE THE CHROMATICITY OF EACH NODE.                      */

  NODE_PTR=HEAD_PTR;
  SELECT(CHROM_DISTRIBUTION);

/* GENERATE CHROMATICITIES FROM TRUNCATED POISSON                */
/* DISTRIBUTION.                                                 */

    WHEN('TRP  ') DO WHILE(NODE_PTR¬=NULL);
      DO UNTIL(CHROMATICITY>0);
        CHROMATICITY=POISSON(SEED,LAMBDA);
      END;
      NODE_PTR=ADJ_NODE_PTR(0);
    END;

/* GENERATE CHROMATICITIES FROM UNIFORM DISTRIBUTION.            */

    WHEN('UNI  ') DO;
      WIDTH=UPPER_LIMIT-LOWER_LIMIT+1;
      DO WHILE(NODE_PTR¬=NULL);
        CALL RANDU(SEED,RANDOM_NUMBER);
        CHROMATICITY=LOWER_LIMIT+
          FLOOR(WIDTH*RANDOM_NUMBER);
```

```
        NODE_PTR=ADJ_NODE_PTR(0);
      END;
    END;

/* GENERATE CHROMATICITIES FROM DOWN RAMP DISTRIBUTION.    */

    WHEN('DNR  ') DO;
      WIDTH=UPPER_LIMIT-LOWER_LIMIT+1;
      DO WHILE(NODE_PTR¬=NULL);
        CALL RANDU(SEED,RANDOM_NUMBER);
        CHROMATICITY=UPPER_LIMIT-
          FLOOR(WIDTH*(RANDOM_NUMBER**0.5));
        NODE_PTR=ADJ_NODE_PTR(0);
      END;
    END;

/* GENERATE CHROMATICITIES FROM UP RAMP DISTRIBUTION.      */

    WHEN('UPR  ') DO;
      WIDTH=UPPER_LIMIT-LOWER_LIMIT+1;
      DO WHILE(NODE_PTR¬=NULL);
        CALL RANDU(SEED,RANDOM_NUMBER);
        CHROMATICITY=LOWER_LIMIT+
          FLOOR(WIDTH*(RANDOM_NUMBER**0.5));
        NODE_PTR=ADJ_NODE_PTR(0);
      END;
    END;

/* GENERATE THE CHROMATICITIES FROM SHIFTED BINOMIAL      */
/* DISTRIBUTION.                                           */

    WHEN('BIN  ') DO WHILE(NODE_PTR¬=NULL);
      SUCCESS_COUNT=0;
      DO TRIAL_NO=1 TO NO_OF_TRIALS;
        CALL RANDU(SEED,RANDOM_NUMBER);
        IF RANDOM_NUMBER <= PROB_OF_SUCCESS
          THEN SUCCESS_COUNT=SUCCESS_COUNT+1;
      END;
      CHROMATICITY=LOWER_LIMIT+SUCCESS_COUNT;
      NODE_PTR=ADJ_NODE_PTR(0);
    END;
    OTHERWISE SIGNAL ERROR;
  END; /* SELECT */

/* CREATE NETWORK TO REPRESENT THE GRAPH.                  */

  DO NODE_NO=1 TO NO_OF_NODES;
    NODE_PTR=LOCATOR(NODE_NO);
    ACCUM_DEGREE=0;
    DO J=1 TO NO_OF_NODES;
      IF EDGES(NODE_NO,J)
        THEN DO;
          ACCUM_DEGREE=ACCUM_DEGREE+1;
          ADJ_NODE_PTR(ACCUM_DEGREE)=LOCATOR(J);
```

```
      END;
    END;
  END;
END GENERATE_GRAPH;

/* CALCULATE THE CHROMATIC DEGREE OF EACH NODE.                */

  NODE_PTR=HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);
    CHROM_DEG=CHROMATICITY;
    DO J=1 TO DEGREE;
      CHROM_DEG=CHROM_DEG+ADJ_NODE_PTR(J)->CHROMATICITY;
    END;
    CHROMATIC_DEGREE=CHROM_DEG;
    NODE_PTR=ADJ_NODE_PTR(0);
  END;
END CREGRAF;
```

```
/*  PROCEDURE 'POISSON' GENERATES A POISSON RANDOM          */
/*  VARIATE.                                                */
/*                                                          */
/*  SEED - SEED FOR PROCEDURE 'RANDU'                       */
/*  MEAN - MEAN OF THE POISSON DISTRIBUTION                 */
/*                                                          */
/*  PROCEDURES CALLED:  EXPON                               */

POISSON: PROC(SEED,MEAN) RETURNS(FIXED BINARY(31)) REORDER;
  DCL (SEED,COUNT) FIXED BINARY(31),
      (MEAN,SUM INIT(0.0)) FLOAT BINARY(21),
      EXPON ENTRY RETURNS(FLOAT BINARY(21));
  DO COUNT=-1 BY 1 WHILE(SUM<=MEAN);
    SUM=SUM+EXPON(SEED);
    END;
  RETURN(COUNT);
  END POISSON;


/*  PROCEDURE 'EXPON' GENERATES AN EXPONENTIAL RANDOM       */
/*  VARIATE FROM A NEGATIVE EXPONENTIAL DISTRIBUTION        */
/*  WITH MEAN 1.0.                                          */
/*                                                          */
/*  SEED - SEED FOR PROCEDURE 'RANDU'                       */
/*                                                          */
/*  PROCEDURES CALLED:  RANDU                               */

EXPON: PROC(SEED) RETURNS(FLOAT BINARY(21)) REORDER;
  DCL SEED FIXED BINARY(31),
      RANDOM_NUMBER FLOAT BINARY(21),
      RANDU ENTRY,
      LOG BUILTIN;
  CALL RANDU(SEED,RANDOM_NUMBER);
  RETURN(-LOG(RANDOM_NUMBER));
  END EXPON;


/*  THE ROUTINE 'RANDU' GENERATES A UNIFORM (0,1)           */
/*  VARIATE.                                                */
/*                                                          */
/*  SEED - SEED FOR THE RANDOM NUMBER GENERATOR             */
/*  RANDOM_NUMBER - THE RANDOM VARIATE                      */

(NOFOFL):
RANDU: PROC(SEED,RANDOM_NUMBER);
  DCL SEED FIXED BINARY(31),
    RANDOM_NUMBER FLOAT BINARY(21);
  SEED=SEED*65539;
  IF SEED < 0 THEN SEED=(SEED+2147483647)+1;
  RANDOM_NUMBER=SEED*0.4656613E-9;
  RETURN;
  END RANDU;
```

```
SEQCOL: PROC(AUX_HEAD_PTR,MAX_COLOR) REORDER;
  DCL
    AUX_HEAD_PTR                PTR,
    NODE_PTR                    PTR,
    DEG                         FIXED BINARY(15),
    NULL                        BUILTIN,
    MAX_COLOR                   FIXED BINARY(15),
    AVAIL                       ENTRY
                                  RETURNS(FIXED BINARY(15)),
    MAX                         BUILTIN;
  DCL
    1 NODE                      BASED(NODE_PTR),
      2 NUMBER                  FIXED BINARY(15),
      2 DEGREE                  FIXED BINARY(15),
      2 CHROMATICITY            FIXED BINARY(15),
      2 CHROMATIC_DEGREE        FIXED BINARY(15),
      2 LO_COLOR                FIXED BINARY(15),
      2 HI_COLOR                FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,      /* 40 BYTES */
        3 FILLER                CHAR(36),
        3 AUX_F_PTR             PTR,
      2 FORWARD_PTR             PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE COLORS.                                            */

  NODE_PTR=AUX_HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);
    LO_COLOR=0;
    HI_COLOR=0;
    NODE_PTR=AUX_F_PTR;
  END;
  MAX_COLOR=AUX_HEAD_PTR->CHROMATICITY;

/* COLOR THE NODES SEQUENTIALLY.                                 */

  NODE_PTR=AUX_HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);

/* DETERMINE THE LOWEST SEQUENCE OF COLORS THAT CAN BE           */
/* ASSIGNED TO THE CURRENT NODE.                                 */

    LO_COLOR=AVAIL(NODE_PTR,MAX_COLOR);
    HI_COLOR=LO_COLOR+CHROMATICITY-1;
    MAX_COLOR=MAX(MAX_COLOR,HI_COLOR);
    NODE_PTR=AUX_F_PTR;
  END;
  END SEQCOL;
```

```
AVAIL: PROC(NODE_PTR,MAX_COLOR) RETURNS(FIXED BINARY(15))
       REORDER;
  DCL
    NODE_PTR                  PTR,
    MAX_COLOR                 FIXED BINARY(15),
    ADJ_COLORS(MAX_COLOR)     BIT(1),
    UNAVAIL_COLOR             FIXED BINARY(15),
    COLOR_COUNT               FIXED BINARY(15),
    (I,J)                     FIXED BINARY(15),
    ADJ_PTR                   PTR;
  DCL
    1 NODE                    BASED(NODE_PTR),
      2 NUMBER                FIXED BINARY(15),
      2 DEGREE                FIXED BINARY(15),
      2 CHROMATICITY          FIXED BINARY(15),
      2 CHROMATIC_DEGREE      FIXED BINARY(15),
      2 LO_COLOR              FIXED BINARY(15),
      2 HI_COLOR              FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,     /* 40 BYTES */
        3 FILLER              CHAR(36),
        3 AUX_F_PTR           PTR,
      2 FORWARD_PTR           PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* DETERMINE WHICH COLORS ARE ADJACENT TO THE CURRENT     */
/* NODE.                                                  */

  ADJ_COLORS(*)='0'B;
  DO I=1 TO DEGREE;
    ADJ_PTR=ADJ_NODE_PTR(I);
    IF ADJ_PTR->LO_COLOR > 0
      THEN DO J=ADJ_PTR->LO_COLOR TO ADJ_PTR->HI_COLOR;
        ADJ_COLORS(J)='1'B;
        END;
    END;

/* DETERMINE THE LOWEST SEQUENCE OF COLORS THAT CAN BE    */
/* ASSIGNED TO THE CURRENT NODE.                          */

  COLOR_COUNT=0;
  UNAVAIL_COLOR=0;
  DO J=1 TO MAX_COLOR UNTIL(COLOR_COUNT=CHROMATICITY);
    IF ADJ_COLORS(J)
      THEN DO;
        COLOR_COUNT=0;
        UNAVAIL_COLOR=J;
        END;
      ELSE COLOR_COUNT=COLOR_COUNT+1;
    END;
  RETURN(UNAVAIL_COLOR+1);
  END AVAIL;
```

```
SEQINT: PROC(AUX_HEAD_PTR,MAX_COLOR) REORDER;
  DCL
    AUX_HEAD_PTR                PTR,
    NODE_PTR                    PTR,
    DEG                         FIXED BINARY(15),
    NULL                        BUILTIN,
    MAX_COLOR                   FIXED BINARY(15),
    AVAIL                       ENTRY
                                  RETURNS(FIXED BINARY(15)),
    INTCHG                      ENTRY,
    MAX                         BUILTIN;
  DCL
    1 NODE                      BASED(NODE_PTR),
      2 NUMBER                  FIXED BINARY(15),
      2 DEGREE                  FIXED BINARY(15),
      2 CHROMATICITY            FIXED BINARY(15),
      2 CHROMATIC_DEGREE        FIXED BINARY(15),
      2 LO_COLOR                FIXED BINARY(15),
      2 HI_COLOR                FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,      /* 40 BYTES */
        3 FILLER                CHAR(36),
        3 AUX_F_PTR             PTR,
      2 FORWARD_PTR             PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE COLORS.                                         */

  NODE_PTR=AUX_HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);
    LO_COLOR=0;
    HI_COLOR=0;
    NODE_PTR=AUX_F_PTR;
    END;
  MAX_COLOR=AUX_HEAD_PTR->CHROMATICITY;

/* COLOR THE NODES SEQUENTIALLY.                              */

  NODE_PTR=AUX_HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);

/* DETERMINE THE LOWEST SEQUENCE OF COLORS THAT CAN BE        */
/* ASSIGNED TO THE CURRENT NODE.                              */

    LO_COLOR=AVAIL(NODE_PTR,MAX_COLOR);
    HI_COLOR=LO_COLOR+CHROMATICITY-1;

/* IF NUMBER OF COLORS ARE INCREASED, THEN ATTEMPT AN         */
/* INTERCHANGE.                                               */

    IF MAX_COLOR < HI_COLOR
      THEN CALL INTCHG(NODE_PTR,AUX_HEAD_PTR,MAX_COLOR);
    NODE_PTR=AUX_F_PTR;
    END;
  END SEQINT;
```

```
INTCHG: PROC(NODE_PTR,AUX_HEAD_PTR,MAX_COLOR) REORDER;
  DCL
    NODE_PTR                PTR,
    AUX_HEAD_PTR            PTR,
    MAX_COLOR               FIXED BINARY(15),
    SWAP_NODE_PTR           PTR,
    SWAP_LO_COLOR           FIXED BINARY(15),
    SWAP_HI_COLOR           FIXED BINARY(15),
    NEW_LO_COLOR            FIXED BINARY(15),
    NEW_HI_COLOR            FIXED BINARY(15),
    ORIG_LO_COLOR           FIXED BINARY(15),
    TEST_NODE_PTR           PTR,
    TEST_LO_COLOR           FIXED BINARY(15),
    TEST_HI_COLOR           FIXED BINARY(15),
    SWAP_MAX_COLOR          FIXED BINARY(15),
    TEST_MAX_COLOR          FIXED BINARY(15),
    POSS_LO_COLOR           FIXED BINARY(15),
    POSS_HI_COLOR           FIXED BINARY(15),
    NULL                    BUILTIN,
    DEG                     FIXED BINARY(15),
    J                       FIXED BINARY(15),
    ADJ_PTR                 PTR,
    TEMP_PTR                PTR,
    AVAIL                   ENTRY
                              RETURNS(FIXED BINARY(15)),
    MIN                     BUILTIN,
    MAX                     BUILTIN;
  DCL
    1 NODE                  BASED(NODE_PTR),
      2 NUMBER              FIXED BINARY(15),
      2 DEGREE              FIXED BINARY(15),
      2 CHROMATICITY        FIXED BINARY(15),
      2 CHROMATIC_DEGREE    FIXED BINARY(15),
      2 LO_COLOR            FIXED BINARY(15),
      2 HI_COLOR            FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,   /* 40 BYTES */
        3 FILLER            CHAR(36),
        3 AUX_F_PTR         PTR,
      2 FORWARD_PTR         PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;
  SWAP_NODE_PTR=NODE_PTR;
  SWAP_LO_COLOR=LO_COLOR;
  SWAP_HI_COLOR=HI_COLOR;
  NEW_LO_COLOR=LO_COLOR;
  NEW_HI_COLOR=HI_COLOR;
  SWAP_MAX_COLOR=HI_COLOR;

/* DETERMINE POSSIBLE LOW COLORS FOR THE CURRENT NODE.    */

  POSS_LO_COLOR=1;
  ORIG_LO_COLOR=LO_COLOR;
  DO WHILE(POSS_LO_COLOR < ORIG_LO_COLOR);
    POSS_HI_COLOR=POSS_LO_COLOR+CHROMATICITY-1;
    IF POSS_HI_COLOR >= SWAP_MAX_COLOR THEN LEAVE;
```

```
    TEST_NODE_PTR=NULL;
    DO J=1 TO DEGREE;
      ADJ_PTR=ADJ_NODE_PTR(J);
      IF ADJ_PTR->LO_COLOR > 0
        THEN IF POSS_LO_COLOR <= ADJ_PTR->HI_COLOR &
          POSS_HI_COLOR >= ADJ_PTR->LO_COLOR
          THEN IF TEST_NODE_PTR = NULL
            THEN TEST_NODE_PTR = ADJ_PTR;
            ELSE DO;
              POSS_LO_COLOR=MIN(ADJ_PTR->HI_COLOR,
                TEST_NODE_PTR->HI_COLOR);
              GO TO NEXT_POSS_LO_COLOR;
            END;
          ELSE;
        ELSE;
    END;
    LO_COLOR=POSS_LO_COLOR;
    HI_COLOR=POSS_HI_COLOR;
    TEST_MAX_COLOR=MAX(POSS_HI_COLOR,MAX_COLOR);
    TEST_LO_COLOR=AVAIL(TEST_NODE_PTR,TEST_MAX_COLOR);
    TEST_HI_COLOR=TEST_LO_COLOR+
      TEST_NODE_PTR->CHROMATICITY-1;
    TEST_MAX_COLOR=MAX(TEST_HI_COLOR,POSS_HI_COLOR);
    IF TEST_MAX_COLOR < SWAP_MAX_COLOR
      THEN DO;
        SWAP_NODE_PTR=TEST_NODE_PTR;
        SWAP_LO_COLOR=TEST_LO_COLOR;
        SWAP_HI_COLOR=TEST_HI_COLOR;
        NEW_LO_COLOR=POSS_LO_COLOR;
        NEW_HI_COLOR=POSS_HI_COLOR;
        SWAP_MAX_COLOR=TEST_MAX_COLOR;
      END;
      ELSE;
NEXT_POSS_LO_COLOR:
    POSS_LO_COLOR=POSS_LO_COLOR+1;
  END;

/* COLOR THE NODES IN THE INTERCHANGE.                          */

  SWAP_NODE_PTR->LO_COLOR=SWAP_LO_COLOR;
  SWAP_NODE_PTR->HI_COLOR=SWAP_HI_COLOR;
  LO_COLOR=NEW_LO_COLOR;
  HI_COLOR=NEW_HI_COLOR;

/* DETERMINE THE NEW MAXIMUM COLOR USED.                        */

  MAX_COLOR=SWAP_MAX_COLOR;
  TEMP_PTR=AUX_HEAD_PTR;
  DO WHILE(TEMP_PTR¬=NODE_PTR);
    MAX_COLOR=MAX(MAX_COLOR,TEMP_PTR->HI_COLOR);
    TEMP_PTR=TEMP_PTR->AUX_F_PTR;
  END;
END INTCHG;
```

```
/* SORT THE NODE LIST IN THE ORDER:                              */
/*   1. ORDERED ACCORDING TO DECREASING CHROMATICITY AND         */
/*   2. SUB-ORDERED ACCORDING TO DECREASING CHROMATIC            */
/*      DEGREE.                                                  */

LF1SORT: PROC(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES) REORDER;
  DCL
    HEAD_PTR                PTR,
    AUX_HEAD_PTR            PTR,
    NODE_PTR                PTR,
    NO_OF_NODES             FIXED BINARY(15),
    NODE_NO                 FIXED BINARY(15),
    TAG(NO_OF_NODES)        PTR,
    AUX_TAIL_PTR            PTR,
    TAG_NO                  FIXED BINARY(15),
    CURRENT_TAG_NO          FIXED BINARY(15),
    PARENT_TAG_NO           FIXED BINARY(15),
    CHILD_TAG_NO            FIXED BINARY(15),
    RIGHT_TAG_NO            FIXED BINARY(15),
    TEMP_TAG                PTR,
    REMAINING_NODES         FIXED BINARY(15),
    NULL                    BUILTIN;
  DCL
    1 NODE                  BASED(NODE_PTR),
      2 NUMBER              FIXED BINARY(15),
      2 DEGREE              FIXED BINARY(15),
      2 CHROMATICITY        FIXED BINARY(15),
      2 CHROMATIC_DEGREE    FIXED BINARY(15),
      2 LO_COLOR            FIXED BINARY(15),
      2 HI_COLOR            FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,   /* 40 BYTES */
        3 FILLER            CHAR(36),
        3 AUX_F_PTR         PTR,
      2 FORWARD_PTR         PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE TAG ARRAY.                                         */

  TAG(1)=HEAD_PTR;
  NODE_PTR=HEAD_PTR;
  DO NODE_NO=2 TO NO_OF_NODES;
    NODE_PTR=FORWARD_PTR;
    TAG(NODE_NO)=NODE_PTR;
  END;

/* CREATE THE INITIAL HEAP.                                      */

  DO TAG_NO=2 TO NO_OF_NODES;
    CURRENT_TAG_NO=TAG_NO;
    DO UNTIL(CURRENT_TAG_NO=1);
      PARENT_TAG_NO=CURRENT_TAG_NO/2;
      IF GREATER(TAG(CURRENT_TAG_NO),TAG(PARENT_TAG_NO))
        THEN DO;
          TEMP_TAG=TAG(CURRENT_TAG_NO);
```

```
          TAG(CURRENT_TAG_NO)=TAG(PARENT_TAG_NO);
          TAG(PARENT_TAG_NO)=TEMP_TAG;
          CURRENT_TAG_NO=PARENT_TAG_NO;
        END;
        ELSE LEAVE;
    END;
  END;

/* THE GREATEST NODE IS AT THE TOP OF THE HEAP.              */
/* PLACE GREATEST NODE IN SORTED NODE LIST.                  */

  AUX_HEAD_PTR=TAG(1);
  AUX_TAIL_PTR=AUX_HEAD_PTR;

/* SORT REMAINING NODES AND PLACE IN SORTED NODE LIST.       */

  REMAINING_NODES=NO_OF_NODES;
  DO WHILE(REMAINING_NODES>0);
    TEMP_TAG=TAG(REMAINING_NODES);
    REMAINING_NODES=REMAINING_NODES-1;
    PARENT_TAG_NO=1;
    DO CHILD_TAG_NO=2 REPEAT(2*PARENT_TAG_NO)
                      WHILE(CHILD_TAG_NO<=REMAINING_NODES);
      IF CHILD_TAG_NO<REMAINING_NODES
        THEN DO;
          RIGHT_TAG_NO=CHILD_TAG_NO+1;
          IF GREATER(TAG(RIGHT_TAG_NO),TAG(CHILD_TAG_NO))
            THEN CHILD_TAG_NO=RIGHT_TAG_NO;
        END;
      IF GREATER(TAG(CHILD_TAG_NO),TEMP_TAG)
        THEN DO;
          TAG(PARENT_TAG_NO)=TAG(CHILD_TAG_NO);
          PARENT_TAG_NO=CHILD_TAG_NO;
        END;
        ELSE LEAVE;
    END;
    TAG(PARENT_TAG_NO)=TEMP_TAG;

/* APPEND THE GREATEST REMAINING NODE TO SORTED NODE         */
/* LIST.                                                     */

    TEMP_TAG=TAG(1);
    AUX_TAIL_PTR->AUX_F_PTR=TEMP_TAG;
    AUX_TAIL_PTR=TEMP_TAG;
  END;
  AUX_TAIL_PTR->AUX_F_PTR=NULL;

/* THE PROCEDURE 'GREATER' DETERMINES WHETHER NODE-A         */
/* BELONGS BEFORE NODE-B IN THE ORDERING:                    */
/*    1. ORDERED ACCORDING TO DECREASING CHROMATICITY AND    */
/*    2. SUB-ORDERED ACCORDING TO DECREASING CHROMATIC       */
/*       DEGREE.                                             */

GREATER: PROC(TAG_A,TAG_B) RETURNS(BIT(1)) REORDER;
```

```
  DCL
    TAG_A                    PTR,
    TAG_B                    PTR;

  IF TAG_A->CHROMATICITY > TAG_B->CHROMATICITY
    THEN RETURN('1'B);
  IF TAG_A->CHROMATICITY = TAG_B->CHROMATICITY
    THEN
    IF TAG_A->CHROMATIC_DEGREE > TAG_B->CHROMATIC_DEGREE
      THEN RETURN('1'B);
      ELSE
      IF TAG_A->CHROMATIC_DEGREE = TAG_B->CHROMATIC_DEGREE
        THEN IF TAG_A->NUMBER < TAG_B->NUMBER
          THEN RETURN('1'B);
          ELSE;
        ELSE;
  RETURN('0'B);
END GREATER;
END LF1SORT;
```

```
/* SORT THE NODE LIST IN THE ORDER:                                  */
/*    1. ORDERED ACCORDING TO DECREASING CHROMATIC DEGREE */
/*       AND                                                         */
/*    2. SUB-ORDERED ACCORDING TO DECREASING CHROMATICITY.*/

LF2SORT: PROC(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES) REORDER;
  DCL
    HEAD_PTR                PTR,
    AUX_HEAD_PTR            PTR,
    NODE_PTR                PTR,
    NO_OF_NODES             FIXED BINARY(15),
    NODE_NO                 FIXED BINARY(15),
    TAG(NO_OF_NODES)        PTR,
    AUX_TAIL_PTR            PTR,
    TAG_NO                  FIXED BINARY(15),
    CURRENT_TAG_NO          FIXED BINARY(15),
    PARENT_TAG_NO           FIXED BINARY(15),
    CHILD_TAG_NO            FIXED BINARY(15),
    RIGHT_TAG_NO            FIXED BINARY(15),
    TEMP_TAG                PTR,
    REMAINING_NODES         FIXED BINARY(15),
    NULL                    BUILTIN;
  DCL
    1 NODE                  BASED(NODE_PTR),
      2 NUMBER              FIXED BINARY(15),
      2 DEGREE              FIXED BINARY(15),
      2 CHROMATICITY        FIXED BINARY(15),
      2 CHROMATIC_DEGREE    FIXED BINARY(15),
      2 LO_COLOR            FIXED BINARY(15),
      2 HI_COLOR            FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,   /* 40 BYTES */
        3 FILLER            CHAR(36),
        3 AUX_F_PTR         PTR,
      2 FORWARD_PTR         PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE TAG ARRAY.                                              */

  TAG(1)=HEAD_PTR;
  NODE_PTR=HEAD_PTR;
  DO NODE_NO=2 TO NO_OF_NODES;
    NODE_PTR=FORWARD_PTR;
    TAG(NODE_NO)=NODE_PTR;
  END;

/* CREATE THE INITIAL HEAP.                                           */
  DO TAG_NO=2 TO NO_OF_NODES;
    CURRENT_TAG_NO=TAG_NO;
    DO UNTIL(CURRENT_TAG_NO=1);
      PARENT_TAG_NO=CURRENT_TAG_NO/2;
      IF GREATER(TAG(CURRENT_TAG_NO),TAG(PARENT_TAG_NO))
        THEN DO;
          TEMP_TAG=TAG(CURRENT_TAG_NO);
          TAG(CURRENT_TAG_NO)=TAG(PARENT_TAG_NO);
```

```
          TAG(PARENT_TAG_NO)=TEMP_TAG;
          CURRENT_TAG_NO=PARENT_TAG_NO;
        END;
        ELSE LEAVE;
    END;
  END;

/* THE GREATEST NODE IS AT THE TOP OF THE HEAP.                  */
/* PLACE GREATEST NODE IN SORTED NODE LIST.                      */

  AUX_HEAD_PTR=TAG(1);
  AUX_TAIL_PTR=AUX_HEAD_PTR;

/* SORT REMAINING NODES AND PLACE IN SORTED NODE LIST.           */

  REMAINING_NODES=NO_OF_NODES;
  DO WHILE(REMAINING_NODES>0);
    TEMP_TAG=TAG(REMAINING_NODES);
    REMAINING_NODES=REMAINING_NODES-1;
    PARENT_TAG_NO=1;
    DO CHILD_TAG_NO=2 REPEAT(2*PARENT_TAG_NO)
                      WHILE(CHILD_TAG_NO<=REMAINING_NODES);
      IF CHILD_TAG_NO<REMAINING_NODES
        THEN DO;
          RIGHT_TAG_NO=CHILD_TAG_NO+1;
          IF GREATER(TAG(RIGHT_TAG_NO),TAG(CHILD_TAG_NO))
            THEN CHILD_TAG_NO=RIGHT_TAG_NO;
        END;
      IF GREATER(TAG(CHILD_TAG_NO),TEMP_TAG)
        THEN DO;
          TAG(PARENT_TAG_NO)=TAG(CHILD_TAG_NO);
          PARENT_TAG_NO=CHILD_TAG_NO;
        END;
        ELSE LEAVE;
    END;
    TAG(PARENT_TAG_NO)=TEMP_TAG;

/* APPEND THE GREATEST REMAINING NODE TO SORTED NODE             */
/* LIST.                                                         */

    TEMP_TAG=TAG(1);
    AUX_TAIL_PTR->AUX_F_PTR=TEMP_TAG;
    AUX_TAIL_PTR=TEMP_TAG;
  END;
  AUX_TAIL_PTR->AUX_F_PTR=NULL;

/* THE PROCEDURE 'GREATER' DETERMINES WHETHER NODE-A             */
/* BELONGS BEFORE NODE-B IN THE ORDERING:                        */
/*    1. ORDERED ACCORDING TO DECREASING CHROMATIC DEGREE        */
/*       AND                                                     */
/*    2. SUB-ORDERED ACCORDING TO DECREASING CHROMATICITY.       */

GREATER: PROC(TAG_A,TAG_B) RETURNS(BIT(1)) REORDER;
```

```
  DCL
    TAG_A                    PTR,
    TAG_B                    PTR;

  IF TAG_A->CHROMATIC_DEGREE > TAG_B->CHROMATIC_DEGREE
    THEN RETURN('1'B);
  IF TAG_A->CHROMATIC_DEGREE = TAG_B->CHROMATIC_DEGREE
    THEN IF TAG_A->CHROMATICITY > TAG_B->CHROMATICITY
      THEN RETURN('1'B);
      ELSE IF TAG_A->CHROMATICITY = TAG_B->CHROMATICITY
        THEN IF TAG_A->NUMBER < TAG_B->NUMBER
          THEN RETURN('1'B);
          ELSE;
        ELSE;
  RETURN('0'B);
END GREATER;
END LF2SORT;
```

```
/* SORT THE NODE LIST IN THE ORDER OF DECREASING             */
/* PIGEONHOLE MEASURE.                                       */

PHSORT: PROC(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES) REORDER;
  DCL
    HEAD_PTR                    PTR,
    AUX_HEAD_PTR                PTR,
    NODE_PTR                    PTR,
    NO_OF_NODES                 FIXED BINARY(15),
    NODE_NO                     FIXED BINARY(15),
    TAG(NO_OF_NODES)            PTR,
    AUX_TAIL_PTR                PTR,
    TAG_NO                      FIXED BINARY(15),
    CURRENT_TAG_NO              FIXED BINARY(15),
    PARENT_TAG_NO               FIXED BINARY(15),
    CHILD_TAG_NO                FIXED BINARY(15),
    RIGHT_TAG_NO                FIXED BINARY(15),
    TEMP_TAG                    PTR,
    REMAINING_NODES             FIXED BINARY(15),
    NULL                        BUILTIN;
  DCL
    1 NODE                      BASED(NODE_PTR),
      2 NUMBER                  FIXED BINARY(15),
      2 DEGREE                  FIXED BINARY(15),
      2 CHROMATICITY            FIXED BINARY(15),
      2 CHROMATIC_DEGREE        FIXED BINARY(15),
      2 LO_COLOR                FIXED BINARY(15),
      2 HI_COLOR                FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,       /* 40 BYTES */
        3 PIGEON_HOLE           FIXED BINARY(31),
        3 FILLER                CHAR(32),
        3 AUX_F_PTR             PTR,
      2 FORWARD_PTR             PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE TAG ARRAY.                                     */

  TAG(1)=HEAD_PTR;
  NODE_PTR=HEAD_PTR;
  DO NODE_NO=2 TO NO_OF_NODES;
    NODE_PTR=FORWARD_PTR;
    TAG(NODE_NO)=NODE_PTR;
  END;

/* CALCULATE THE PIGEONHOLE MEASURE OF EACH NODE.            */

    NODE_PTR=HEAD_PTR;
    DO WHILE(NODE_PTR¬=NULL);
      PIGEON_HOLE=CHROMATIC_DEGREE+(CHROMATICITY-1)*DEGREE;
      NODE_PTR=FORWARD_PTR;
    END;
```

```
/* CREATE THE INITIAL HEAP.                                  */

  DO TAG_NO=2 TO NO_OF_NODES;
    CURRENT_TAG_NO=TAG_NO;
    DO UNTIL(CURRENT_TAG_NO=1);
      PARENT_TAG_NO=CURRENT_TAG_NO/2;
      IF GREATER(TAG(CURRENT_TAG_NO),TAG(PARENT_TAG_NO))
        THEN DO;
          TEMP_TAG=TAG(CURRENT_TAG_NO);
          TAG(CURRENT_TAG_NO)=TAG(PARENT_TAG_NO);
          TAG(PARENT_TAG_NO)=TEMP_TAG;
          CURRENT_TAG_NO=PARENT_TAG_NO;
        END;
        ELSE LEAVE;
    END;
  END;

/* THE GREATEST NODE IS AT THE TOP OF THE HEAP.              */
/* PLACE GREATEST NODE IN SORTED NODE LIST.                  */

  AUX_HEAD_PTR=TAG(1);
  AUX_TAIL_PTR=AUX_HEAD_PTR;

/* SORT REMAINING NODES AND PLACE IN SORTED NODE LIST.       */

  REMAINING_NODES=NO_OF_NODES;
  DO WHILE(REMAINING_NODES>0);
    TEMP_TAG=TAG(REMAINING_NODES);
    REMAINING_NODES=REMAINING_NODES-1;
    PARENT_TAG_NO=1;
    DO CHILD_TAG_NO=2 REPEAT(2*PARENT_TAG_NO)
                      WHILE(CHILD_TAG_NO<=REMAINING_NODES);
      IF CHILD_TAG_NO<REMAINING_NODES
        THEN DO;
          RIGHT_TAG_NO=CHILD_TAG_NO+1;
          IF GREATER(TAG(RIGHT_TAG_NO),TAG(CHILD_TAG_NO))
            THEN CHILD_TAG_NO=RIGHT_TAG_NO;
        END;
      IF GREATER(TAG(CHILD_TAG_NO),TEMP_TAG)
        THEN DO;
          TAG(PARENT_TAG_NO)=TAG(CHILD_TAG_NO);
          PARENT_TAG_NO=CHILD_TAG_NO;
        END;
        ELSE LEAVE;
    END;
    TAG(PARENT_TAG_NO)=TEMP_TAG;

/* APPEND THE GREATEST REMAINING NODE TO SORTED NODE         */
/* LIST.                                                     */

    TEMP_TAG=TAG(1);
    AUX_TAIL_PTR->AUX_F_PTR=TEMP_TAG;
    AUX_TAIL_PTR=TEMP_TAG;
  END;
```

```
  AUX_TAIL_PTR->AUX_F_PTR=NULL;

/* THE PROCEDURE 'GREATER' DETERMINES WHETHER NODE-A      */
/* BELONGS BEFORE NODE-B IN THE ORDER OF DECREASING       */
/* PIGEONHOLE MEASURE.                                    */

GREATER: PROC(TAG_A,TAG_B) RETURNS(BIT(1)) REORDER;
  DCL
    TAG_A                    PTR,
    TAG_B                    PTR;

  IF TAG_A->PIGEON_HOLE > TAG_B->PIGEON_HOLE
    THEN RETURN('1'B);
  IF TAG_A->PIGEON_HOLE = TAG_B->PIGEON_HOLE
    THEN IF TAG_A->NUMBER < TAG_B->NUMBER
      THEN RETURN('1'B);
      ELSE;
    ELSE;
  RETURN('0'B);
END GREATER;
END PHSORT;
```

```
/* SORT THE NODE LIST IN THE ORDER OF DECREASING PRODUCT */
/* OF CHROMATICITY AND DEGREE.                            */

CDSORT: PROC(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES) REORDER;
  DCL
    HEAD_PTR                PTR,
    AUX_HEAD_PTR            PTR,
    NODE_PTR                PTR,
    NO_OF_NODES             FIXED BINARY(15),
    NODE_NO                 FIXED BINARY(15),
    TAG(NO_OF_NODES)        PTR,
    AUX_TAIL_PTR            PTR,
    TAG_NO                  FIXED BINARY(15),
    CURRENT_TAG_NO          FIXED BINARY(15),
    PARENT_TAG_NO           FIXED BINARY(15),
    CHILD_TAG_NO            FIXED BINARY(15),
    RIGHT_TAG_NO            FIXED BINARY(15),
    TEMP_TAG                PTR,
    REMAINING_NODES         FIXED BINARY(15),
    NULL                    BUILTIN;
  DCL
    1 NODE                  BASED(NODE_PTR),
      2 NUMBER              FIXED BINARY(15),
      2 DEGREE              FIXED BINARY(15),
      2 CHROMATICITY        FIXED BINARY(15),
      2 CHROMATIC_DEGREE    FIXED BINARY(15),
      2 LO_COLOR            FIXED BINARY(15),
      2 HI_COLOR            FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,   /* 40 BYTES */
        3 C_TIMES_D         FIXED BINARY(31),
        3 FILLER            CHAR(32),
        3 AUX_F_PTR         PTR,
      2 FORWARD_PTR         PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE TAG ARRAY.                                  */

  TAG(1)=HEAD_PTR;
  NODE_PTR=HEAD_PTR;
  DO NODE_NO=2 TO NO_OF_NODES;
    NODE_PTR=FORWARD_PTR;
    TAG(NODE_NO)=NODE_PTR;
  END;

/* CALCULATE THE PRODUCT OF CHROMATICITY AND DEGREE FOR   */
/* EACH NODE.                                             */

    NODE_PTR=HEAD_PTR;
    DO WHILE(NODE_PTR¬=NULL);
      C_TIMES_D=CHROMATICITY*DEGREE;
      NODE_PTR=FORWARD_PTR;
    END;
```

```
/* CREATE THE INITIAL HEAP.                                          */

  DO TAG_NO=2 TO NO_OF_NODES;
    CURRENT_TAG_NO=TAG_NO;
    DO UNTIL(CURRENT_TAG_NO=1);
      PARENT_TAG_NO=CURRENT_TAG_NO/2;
      IF GREATER(TAG(CURRENT_TAG_NO),TAG(PARENT_TAG_NO))
        THEN DO;
          TEMP_TAG=TAG(CURRENT_TAG_NO);
          TAG(CURRENT_TAG_NO)=TAG(PARENT_TAG_NO);
          TAG(PARENT_TAG_NO)=TEMP_TAG;
          CURRENT_TAG_NO=PARENT_TAG_NO;
        END;
        ELSE LEAVE;
    END;
  END;

/* THE GREATEST NODE IS AT THE TOP OF THE HEAP.                      */
/* PLACE GREATEST NODE IN SORTED NODE LIST.                          */

  AUX_HEAD_PTR=TAG(1);
  AUX_TAIL_PTR=AUX_HEAD_PTR;

/* SORT REMAINING NODES AND PLACE IN SORTED NODE LIST.               */

  REMAINING_NODES=NO_OF_NODES;
  DO WHILE(REMAINING_NODES>0);
    TEMP_TAG=TAG(REMAINING_NODES);
    REMAINING_NODES=REMAINING_NODES-1;
    PARENT_TAG_NO=1;
    DO CHILD_TAG_NO=2 REPEAT(2*PARENT_TAG_NO)
                  WHILE(CHILD_TAG_NO<=REMAINING_NODES);
      IF CHILD_TAG_NO<REMAINING_NODES
        THEN DO;
          RIGHT_TAG_NO=CHILD_TAG_NO+1;
          IF GREATER(TAG(RIGHT_TAG_NO),TAG(CHILD_TAG_NO))
            THEN CHILD_TAG_NO=RIGHT_TAG_NO;
        END;
      IF GREATER(TAG(CHILD_TAG_NO),TEMP_TAG)
        THEN DO;
          TAG(PARENT_TAG_NO)=TAG(CHILD_TAG_NO);
          PARENT_TAG_NO=CHILD_TAG_NO;
        END;
        ELSE LEAVE;
    END;
    TAG(PARENT_TAG_NO)=TEMP_TAG;

/* APPEND THE GREATEST REMAINING NODE TO SORTED NODE                 */
/* LIST.                                                             */

    TEMP_TAG=TAG(1);
    AUX_TAIL_PTR->AUX_F_PTR=TEMP_TAG;
    AUX_TAIL_PTR=TEMP_TAG;
  END;
```

```
  AUX_TAIL_PTR->AUX_F_PTR=NULL;

/* THE PROCEDURE 'GREATER' DETERMINES WHETHER NODE-A        */
/* BELONGS BEFORE NODE-B IN THE ORDER OF DECREASING         */
/* PRODUCT OF CHROMATICITY AND DEGREE.                      */

GREATER: PROC(TAG_A,TAG_B) RETURNS(BIT(1)) REORDER;
  DCL
    TAG_A                   PTR,
    TAG_B                   PTR;

  IF TAG_A->C_TIMES_D > TAG_B->C_TIMES_D
    THEN RETURN('1'B);
  IF TAG_A->C_TIMES_D = TAG_B->C_TIMES_D
    THEN IF TAG_A->NUMBER < TAG_B->NUMBER
      THEN RETURN('1'B);
      ELSE;
    ELSE;
  RETURN('0'B);
END GREATER;
END CDSORT;
```

```
RLF1: PROC(HEAD_PTR,AUX_HEAD_PTR,MAX_COLOR) REORDER;
  DCL
    HEAD_PTR                  PTR,
    COLOR_HEAD_PTR            PTR,
    COLOR_TAIL_PTR            PTR,
    U1_HEAD_PTR               PTR,
    U1_TAIL_PTR               PTR,
    AUX_HEAD_PTR              PTR,
    AUX_TAIL_PTR              PTR,
    MAX_COLOR                 FIXED BINARY(15),
    MAX                       BUILTIN,
    NULL                      BUILTIN,
    CURRENT_COLOR             FIXED BINARY(15) INIT(1),
    NEXT_COLOR                FIXED BINARY(15) INIT(32767),
    NODE_PTR                  PTR,
    DEG                       FIXED BINARY(15),
    CHROM_DEG                 FIXED BINARY(15),
    CURRENT_CHROMATICITY      FIXED BINARY(15),
    NODE_CHROMATICITY         FIXED BINARY(15),
    ADJ_CHROMATICITY          FIXED BINARY(15),
    CURRENT_NODE_PTR          PTR,
    U2_CHROM_DEGREE           FIXED BINARY(15),
    CURRENT_TOTAL_CHROM_DEGREE FIXED BINARY(15),
    CURRENT_U1_CHROM_DEGREE FIXED BINARY(15),
    CURRENT_U2_CHROM_DEGREE FIXED BINARY(15),
    LB                        FIXED BINARY(15),
    ADJ_PTR                   PTR,
    ADJ2_PTR                  PTR,
    (I,K)                     FIXED BINARY(15);

  DCL
    1 NODE                    BASED(NODE_PTR),
      2 NUMBER                FIXED BINARY(15),
      2 DEGREE                FIXED BINARY(15),
      2 CHROMATICITY          FIXED BINARY(15),
      2 CHROMATIC_DEGREE      FIXED BINARY(15),
      2 LO_COLOR              FIXED BINARY(15),
      2 HI_COLOR              FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,    /* 40 BYTES */
        3 TOTAL_CHROM_DEGREE FIXED BINARY(15),
        3 U1_CHROM_DEGREE     FIXED BINARY(15),
        3 LOWER_BOUND         FIXED BINARY(15),
        3 FILLER              CHAR(18),
        3 U1_B_PTR            PTR,
        3 U1_F_PTR            PTR,
        3 AUX_B_PTR           PTR,
        3 AUX_F_PTR           PTR,
      2 FORWARD_PTR           PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE COLORED NODE LIST.                                  */

  COLOR_HEAD_PTR=NULL;
  COLOR_TAIL_PTR=NULL;
```

```
  MAX_COLOR=0;

/* CREATE UNCOLORED NODE LIST AND U1 NODE LIST.                    */
/* INITIALIZE WORK VARIABLES FOR EACH NODE.                        */

  AUX_HEAD_PTR=HEAD_PTR;
  U1_HEAD_PTR=HEAD_PTR;
  NODE_PTR=HEAD_PTR;
  U1_TAIL_PTR=NULL;
  DO WHILE(NODE_PTR¬=NULL);
    CHROM_DEG=CHROMATIC_DEGREE;
    U1_CHROM_DEGREE=CHROM_DEG;
    TOTAL_CHROM_DEGREE=CHROM_DEG;
    LOWER_BOUND=1;
    LO_COLOR=0;
    AUX_F_PTR=FORWARD_PTR;
    U1_F_PTR=FORWARD_PTR;
    AUX_B_PTR=U1_TAIL_PTR;
    U1_B_PTR=U1_TAIL_PTR;
    U1_TAIL_PTR=NODE_PTR;
    NODE_PTR=FORWARD_PTR;
  END;
  AUX_TAIL_PTR=U1_TAIL_PTR;

/* SELECT NODES TO COLOR UNTIL ALL NODES ARE COLORED.              */

  DO WHILE('1'B);

/* SELECT FIRST NODE TO BE COLORED WITH THE CURRENT                */
/* COLOR ACCORDING TO:                                             */
/*   1. MAXIMUM CHROMATICITY                                       */
/*   2. MAXIMUM CHROMATIC DEGREE                                   */

    CURRENT_CHROMATICITY=0;
    NODE_PTR=U1_HEAD_PTR;
    DO WHILE(NODE_PTR¬=NULL);
      NODE_CHROMATICITY=CHROMATICITY;
      SELECT;
        WHEN(NODE_CHROMATICITY > CURRENT_CHROMATICITY) DO;
          CURRENT_NODE_PTR=NODE_PTR;
          CURRENT_CHROMATICITY=NODE_CHROMATICITY;
          CURRENT_TOTAL_CHROM_DEGREE=TOTAL_CHROM_DEGREE;
        END;
        WHEN(NODE_CHROMATICITY = CURRENT_CHROMATICITY &
        TOTAL_CHROM_DEGREE > CURRENT_TOTAL_CHROM_DEGREE)
        DO;
          CURRENT_NODE_PTR=NODE_PTR;
          CURRENT_TOTAL_CHROM_DEGREE=TOTAL_CHROM_DEGREE;
        END;
        OTHERWISE;
      END;
      NODE_PTR=U1_F_PTR;
    END;
```

```
/* COLOR THE SELECTED NODE.                                       */

    CALL COLOR_NODE;

/* SELECT REMAINING NODES TO BE COLORED WITH THE CURRENT */
/* COLOR ACCORDING TO:                                    */
/*    1. MAXIMUM CHROMATICITY                             */
/*    2. MAXIMUM CHROMATIC DEGREE IN U2                   */
/*    3. MINIMUM CHROMATIC DEGREE IN U1                   */

    DO WHILE(U1_HEAD_PTR¬=NULL);
      CURRENT_CHROMATICITY=0;
      NODE_PTR=U1_HEAD_PTR;
      DO WHILE(NODE_PTR¬=NULL);
        NODE_CHROMATICITY=CHROMATICITY;
        SELECT;
          WHEN(NODE_CHROMATICITY > CURRENT_CHROMATICITY)
          DO;
            CURRENT_NODE_PTR=NODE_PTR;
            CURRENT_CHROMATICITY=NODE_CHROMATICITY;
            CURRENT_U1_CHROM_DEGREE=U1_CHROM_DEGREE;
            CURRENT_U2_CHROM_DEGREE=
              TOTAL_CHROM_DEGREE-CURRENT_U1_CHROM_DEGREE;
          END;
          WHEN(NODE_CHROMATICITY = CURRENT_CHROMATICITY)
          DO;
            U2_CHROM_DEGREE=TOTAL_CHROM_DEGREE-
              U1_CHROM_DEGREE;
            SELECT;
              WHEN
              (U2_CHROM_DEGREE > CURRENT_U2_CHROM_DEGREE)
              DO;
                CURRENT_NODE_PTR=NODE_PTR;
                CURRENT_U1_CHROM_DEGREE=U1_CHROM_DEGREE;
                CURRENT_U2_CHROM_DEGREE=U2_CHROM_DEGREE;
              END;
              WHEN
              (U2_CHROM_DEGREE = CURRENT_U2_CHROM_DEGREE
              & U1_CHROM_DEGREE < CURRENT_U1_CHROM_DEGREE)
              DO;
                CURRENT_NODE_PTR=NODE_PTR;
                CURRENT_U1_CHROM_DEGREE=U1_CHROM_DEGREE;
              END;
              OTHERWISE;
            END;
          END;
          OTHERWISE;
        END;
        NODE_PTR=U1_F_PTR;
      END;

/* COLOR THE SELECTED NODE.                                       */

      CALL COLOR_NODE;
```

```
    END;

/* U1 IS EMPTY.                                                    */
/* ARE ALL NODES COLORED?                                          */

    IF AUX_HEAD_PTR=NULL
      THEN DO;

/* ALL NODES ARE COLORED. COMPLETE COLORED NODE LIST.              */

        CURRENT_NODE_PTR->AUX_F_PTR=NULL;
        AUX_HEAD_PTR=COLOR_HEAD_PTR;
        RETURN;
      END;
      ELSE;

/* CREATE NEW U1 LIST FOR THE NEXT COLOR.                          */

    DO UNTIL(U1_HEAD_PTR¬=NULL);
      NODE_PTR=AUX_HEAD_PTR;
      CURRENT_COLOR=NEXT_COLOR;
      NEXT_COLOR=32767;
      DO WHILE(NODE_PTR¬=NULL);
        IF LOWER_BOUND=CURRENT_COLOR
          THEN DO;

/* INITIALIZE U1_CHROM_DEGREE OF EACH NODE IN NEW U1               */
/* LIST.                                                           */

            U1_CHROM_DEGREE=CHROMATICITY;

/* APPEND NODE TO U1 LIST.                                         */

            IF U1_HEAD_PTR=NULL
              THEN U1_HEAD_PTR=NODE_PTR;
              ELSE U1_TAIL_PTR->U1_F_PTR=NODE_PTR;
            U1_B_PTR=U1_TAIL_PTR;
            U1_TAIL_PTR=NODE_PTR;
          END;

          ELSE IF NEXT_COLOR > LOWER_BOUND
            THEN NEXT_COLOR=LOWER_BOUND;
            ELSE;
        NODE_PTR=AUX_F_PTR;
      END;
    END;

/* END NEW U1 LIST.                                                */

    U1_TAIL_PTR->U1_F_PTR=NULL;

/* COMPUTE U1 CHROMATIC DEGREE OF EACH NODE IN U1 LIST.            */

    NODE_PTR=U1_HEAD_PTR;
```

```
    DO WHILE(NODE_PTR¬=NULL);
      NODE_CHROMATICITY=CHROMATICITY;
      DO I=1 TO DEGREE;
        ADJ_PTR=ADJ_NODE_PTR(I);
        IF ADJ_PTR->U1_CHROM_DEGREE > 0
          THEN ADJ_PTR->U1_CHROM_DEGREE=
            ADJ_PTR->U1_CHROM_DEGREE+NODE_CHROMATICITY;
      END;
      NODE_PTR=U1_F_PTR;
    END;
  END;

/* COLOR THE NODE POINTED TO BY CURRENT_NODE_PTR.             */

COLOR_NODE: PROC;
  NODE_PTR=CURRENT_NODE_PTR;

/* REMOVE THE NODE TO BE COLORED FROM THE UNCOLORED NODE */
/* LIST.                                                 */

  IF AUX_F_PTR=NULL
    THEN AUX_TAIL_PTR=AUX_B_PTR;
    ELSE AUX_F_PTR->AUX_B_PTR=AUX_B_PTR;
  IF AUX_B_PTR=NULL
    THEN AUX_HEAD_PTR=AUX_F_PTR;
    ELSE AUX_B_PTR->AUX_F_PTR=AUX_F_PTR;

/* REMOVE THE NODE TO BE COLORED FROM THE U1 NODE LIST.  */

  U1_CHROM_DEGREE=0;
  IF U1_F_PTR=NULL
    THEN U1_TAIL_PTR=U1_B_PTR;
    ELSE U1_F_PTR->U1_B_PTR=U1_B_PTR;
  IF U1_B_PTR=NULL
    THEN U1_HEAD_PTR=U1_F_PTR;
    ELSE U1_B_PTR->U1_F_PTR=U1_F_PTR;

/* PLACE THE NODE TO BE COLORED AT THE END OF THE        */
/* COLORED NODE LIST.                                    */

  IF COLOR_TAIL_PTR=NULL
    THEN COLOR_HEAD_PTR=NODE_PTR;
    ELSE COLOR_TAIL_PTR->AUX_F_PTR=NODE_PTR;
  COLOR_TAIL_PTR=NODE_PTR;

/* ASSIGN COLORS TO THE NODE.                            */

  LO_COLOR=CURRENT_COLOR;
  LB=CURRENT_COLOR+CURRENT_CHROMATICITY;
  HI_COLOR=LB-1;
  MAX_COLOR=MAX(MAX_COLOR,HI_COLOR);
```

```
/* ADJUST DEGREE INFORMATION OF ADJACENT NODES.                    */

  DO I=1 TO DEGREE;
    ADJ_PTR=ADJ_NODE_PTR(I);
    IF ADJ_PTR->LO_COLOR=0
      THEN DO;

/*        ADJACENT NODE IS UNCOLORED.                              */

        ADJ_PTR->TOTAL_CHROM_DEGREE=
          ADJ_PTR->TOTAL_CHROM_DEGREE-CURRENT_CHROMATICITY;
        IF LB > ADJ_PTR->LOWER_BOUND
          THEN ADJ_PTR->LOWER_BOUND=LB;
          ELSE;
        IF ADJ_PTR->U1_CHROM_DEGREE > 0
          THEN DO;

/*            ADJACENT NODE IS IN U1.  REMOVE IT FROM U1.          */

            ADJ_PTR->U1_CHROM_DEGREE=0;
            IF ADJ_PTR->U1_F_PTR=NULL
              THEN U1_TAIL_PTR=ADJ_PTR->U1_B_PTR;
              ELSE ADJ_PTR->U1_F_PTR->U1_B_PTR=
                ADJ_PTR->U1_B_PTR;
            IF ADJ_PTR->U1_B_PTR=NULL
              THEN U1_HEAD_PTR=ADJ_PTR->U1_F_PTR;
              ELSE ADJ_PTR->U1_B_PTR->U1_F_PTR=
                ADJ_PTR->U1_F_PTR;
            IF NEXT_COLOR > LB
              THEN NEXT_COLOR=LB;
              ELSE;

/*            REDUCE U1 DEGREES.                                   */

            ADJ_CHROMATICITY=ADJ_PTR->CHROMATICITY;
            DO K=1 TO ADJ_PTR->DEGREE;
              ADJ2_PTR=ADJ_PTR->ADJ_NODE_PTR(K);
              IF ADJ2_PTR->U1_CHROM_DEGREE > 0
                THEN ADJ2_PTR->U1_CHROM_DEGREE=
                  ADJ2_PTR->U1_CHROM_DEGREE-
                  ADJ_CHROMATICITY;
                ELSE;
            END;
          END;
          ELSE;
      END;
      ELSE;
  END;
END COLOR_NODE;
END RLF1;
```

```
  AUX_TAIL_PTR->AUX_F_PTR=NULL;

/* THE PROCEDURE 'GREATER' DETERMINES WHETHER NODE-A      */
/* BELONGS BEFORE NODE-B IN THE ORDER OF DECREASING       */
/* PIGEONHOLE MEASURE.                                    */

GREATER: PROC(TAG_A,TAG_B) RETURNS(BIT(1)) REORDER;
  DCL
    TAG_A                   PTR,
    TAG_B                   PTR;

  IF TAG_A->PIGEON_HOLE > TAG_B->PIGEON_HOLE
    THEN RETURN('1'B);
  IF TAG_A->PIGEON_HOLE = TAG_B->PIGEON_HOLE
    THEN IF TAG_A->NUMBER < TAG_B->NUMBER
      THEN RETURN('1'B);
      ELSE;
    ELSE;
  RETURN('0'B);
END GREATER;
END PHSORT;
```

```
/* SORT THE NODE LIST IN THE ORDER OF DECREASING PRODUCT */
/* OF CHROMATICITY AND DEGREE.                           */

CDSORT: PROC(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES) REORDER;
  DCL
    HEAD_PTR                PTR,
    AUX_HEAD_PTR            PTR,
    NODE_PTR                PTR,
    NO_OF_NODES             FIXED BINARY(15),
    NODE_NO                 FIXED BINARY(15),
    TAG(NO_OF_NODES)        PTR,
    AUX_TAIL_PTR            PTR,
    TAG_NO                  FIXED BINARY(15),
    CURRENT_TAG_NO          FIXED BINARY(15),
    PARENT_TAG_NO           FIXED BINARY(15),
    CHILD_TAG_NO            FIXED BINARY(15),
    RIGHT_TAG_NO            FIXED BINARY(15),
    TEMP_TAG                PTR,
    REMAINING_NODES         FIXED BINARY(15),
    NULL                    BUILTIN;
  DCL
    1 NODE                  BASED(NODE_PTR),
      2 NUMBER              FIXED BINARY(15),
      2 DEGREE              FIXED BINARY(15),
      2 CHROMATICITY        FIXED BINARY(15),
      2 CHROMATIC_DEGREE    FIXED BINARY(15),
      2 LO_COLOR            FIXED BINARY(15),
      2 HI_COLOR            FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,   /* 40 BYTES */
        3 C_TIMES_D         FIXED BINARY(31),
        3 FILLER            CHAR(32),
        3 AUX_F_PTR         PTR,
      2 FORWARD_PTR         PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE TAG ARRAY.                                 */

  TAG(1)=HEAD_PTR;
  NODE_PTR=HEAD_PTR;
  DO NODE_NO=2 TO NO_OF_NODES;
    NODE_PTR=FORWARD_PTR;
    TAG(NODE_NO)=NODE_PTR;
  END;

/* CALCULATE THE PRODUCT OF CHROMATICITY AND DEGREE FOR  */
/* EACH NODE.                                            */

    NODE_PTR=HEAD_PTR;
    DO WHILE(NODE_PTR¬=NULL);
      C_TIMES_D=CHROMATICITY*DEGREE;
      NODE_PTR=FORWARD_PTR;
    END;
```

```
/* CREATE THE INITIAL HEAP.                                    */

  DO TAG_NO=2 TO NO_OF_NODES;
    CURRENT_TAG_NO=TAG_NO;
    DO UNTIL(CURRENT_TAG_NO=1);
      PARENT_TAG_NO=CURRENT_TAG_NO/2;
      IF GREATER(TAG(CURRENT_TAG_NO),TAG(PARENT_TAG_NO))
        THEN DO;
          TEMP_TAG=TAG(CURRENT_TAG_NO);
          TAG(CURRENT_TAG_NO)=TAG(PARENT_TAG_NO);
          TAG(PARENT_TAG_NO)=TEMP_TAG;
          CURRENT_TAG_NO=PARENT_TAG_NO;
        END;
        ELSE LEAVE;
    END;
  END;

/* THE GREATEST NODE IS AT THE TOP OF THE HEAP.                */
/* PLACE GREATEST NODE IN SORTED NODE LIST.                    */

  AUX_HEAD_PTR=TAG(1);
  AUX_TAIL_PTR=AUX_HEAD_PTR;

/* SORT REMAINING NODES AND PLACE IN SORTED NODE LIST.         */

  REMAINING_NODES=NO_OF_NODES;
  DO WHILE(REMAINING_NODES>0);
    TEMP_TAG=TAG(REMAINING_NODES);
    REMAINING_NODES=REMAINING_NODES-1;
    PARENT_TAG_NO=1;
    DO CHILD_TAG_NO=2 REPEAT(2*PARENT_TAG_NO)
                      WHILE(CHILD_TAG_NO<=REMAINING_NODES);
      IF CHILD_TAG_NO<REMAINING_NODES
        THEN DO;
          RIGHT_TAG_NO=CHILD_TAG_NO+1;
          IF GREATER(TAG(RIGHT_TAG_NO),TAG(CHILD_TAG_NO))
            THEN CHILD_TAG_NO=RIGHT_TAG_NO;
        END;
      IF GREATER(TAG(CHILD_TAG_NO),TEMP_TAG)
        THEN DO;
          TAG(PARENT_TAG_NO)=TAG(CHILD_TAG_NO);
          PARENT_TAG_NO=CHILD_TAG_NO;
        END;
        ELSE LEAVE;
    END;
    TAG(PARENT_TAG_NO)=TEMP_TAG;

/* APPEND THE GREATEST REMAINING NODE TO SORTED NODE           */
/* LIST.                                                       */

    TEMP_TAG=TAG(1);
    AUX_TAIL_PTR->AUX_F_PTR=TEMP_TAG;
    AUX_TAIL_PTR=TEMP_TAG;
  END;
```

```
  AUX_TAIL_PTR->AUX_F_PTR=NULL;

/* THE PROCEDURE 'GREATER' DETERMINES WHETHER NODE-A      */
/* BELONGS BEFORE NODE-B IN THE ORDER OF DECREASING       */
/* PRODUCT OF CHROMATICITY AND DEGREE.                    */

GREATER: PROC(TAG_A,TAG_B) RETURNS(BIT(1)) REORDER;
  DCL
    TAG_A                       PTR,
    TAG_B                       PTR;

  IF TAG_A->C_TIMES_D > TAG_B->C_TIMES_D
    THEN RETURN('1'B);
  IF TAG_A->C_TIMES_D = TAG_B->C_TIMES_D
    THEN IF TAG_A->NUMBER < TAG_B->NUMBER
      THEN RETURN('1'B);
      ELSE;
    ELSE;
  RETURN('0'B);
END GREATER;
END CDSORT;
```

```
RLF1: PROC(HEAD_PTR,AUX_HEAD_PTR,MAX_COLOR) REORDER;
  DCL
    HEAD_PTR                PTR,
    COLOR_HEAD_PTR          PTR,
    COLOR_TAIL_PTR          PTR,
    U1_HEAD_PTR             PTR,
    U1_TAIL_PTR             PTR,
    AUX_HEAD_PTR            PTR,
    AUX_TAIL_PTR            PTR,
    MAX_COLOR               FIXED BINARY(15),
    MAX                     BUILTIN,
    NULL                    BUILTIN,
    CURRENT_COLOR           FIXED BINARY(15) INIT(1),
    NEXT_COLOR              FIXED BINARY(15) INIT(32767),
    NODE_PTR                PTR,
    DEG                     FIXED BINARY(15),
    CHROM_DEG               FIXED BINARY(15),
    CURRENT_CHROMATICITY    FIXED BINARY(15),
    NODE_CHROMATICITY       FIXED BINARY(15),
    ADJ_CHROMATICITY        FIXED BINARY(15),
    CURRENT_NODE_PTR        PTR,
    U2_CHROM_DEGREE         FIXED BINARY(15),
    CURRENT_TOTAL_CHROM_DEGREE FIXED BINARY(15),
    CURRENT_U1_CHROM_DEGREE FIXED BINARY(15),
    CURRENT_U2_CHROM_DEGREE FIXED BINARY(15),
    LB                      FIXED BINARY(15),
    ADJ_PTR                 PTR,
    ADJ2_PTR                PTR,
    (I,K)                   FIXED BINARY(15);

  DCL
    1 NODE                  BASED(NODE_PTR),
      2 NUMBER              FIXED BINARY(15),
      2 DEGREE              FIXED BINARY(15),
      2 CHROMATICITY        FIXED BINARY(15),
      2 CHROMATIC_DEGREE    FIXED BINARY(15),
      2 LO_COLOR            FIXED BINARY(15),
      2 HI_COLOR            FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,   /* 40 BYTES */
        3 TOTAL_CHROM_DEGREE FIXED BINARY(15),
        3 U1_CHROM_DEGREE   FIXED BINARY(15),
        3 LOWER_BOUND       FIXED BINARY(15),
        3 FILLER            CHAR(18),
        3 U1_B_PTR          PTR,
        3 U1_F_PTR          PTR,
        3 AUX_B_PTR         PTR,
        3 AUX_F_PTR         PTR,
      2 FORWARD_PTR         PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE COLORED NODE LIST.                                   */

  COLOR_HEAD_PTR=NULL;
  COLOR_TAIL_PTR=NULL;
```

```
  MAX_COLOR=0;

/* CREATE UNCOLORED NODE LIST AND U1 NODE LIST.                    */
/* INITIALIZE WORK VARIABLES FOR EACH NODE.                        */

  AUX_HEAD_PTR=HEAD_PTR;
  U1_HEAD_PTR=HEAD_PTR;
  NODE_PTR=HEAD_PTR;
  U1_TAIL_PTR=NULL;
  DO WHILE(NODE_PTR¬=NULL);
    CHROM_DEG=CHROMATIC_DEGREE;
    U1_CHROM_DEGREE=CHROM_DEG;
    TOTAL_CHROM_DEGREE=CHROM_DEG;
    LOWER_BOUND=1;
    LO_COLOR=0;
    AUX_F_PTR=FORWARD_PTR;
    U1_F_PTR=FORWARD_PTR;
    AUX_B_PTR=U1_TAIL_PTR;
    U1_B_PTR=U1_TAIL_PTR;
    U1_TAIL_PTR=NODE_PTR;
    NODE_PTR=FORWARD_PTR;
  END;
  AUX_TAIL_PTR=U1_TAIL_PTR;

/* SELECT NODES TO COLOR UNTIL ALL NODES ARE COLORED.              */

  DO WHILE('1'B);

/* SELECT FIRST NODE TO BE COLORED WITH THE CURRENT                */
/* COLOR ACCORDING TO:                                             */
/*    1. MAXIMUM CHROMATICITY                                      */
/*    2. MAXIMUM CHROMATIC DEGREE                                  */

    CURRENT_CHROMATICITY=0;
    NODE_PTR=U1_HEAD_PTR;
    DO WHILE(NODE_PTR¬=NULL);
      NODE_CHROMATICITY=CHROMATICITY;
      SELECT;
        WHEN(NODE_CHROMATICITY > CURRENT_CHROMATICITY) DO;
          CURRENT_NODE_PTR=NODE_PTR;
          CURRENT_CHROMATICITY=NODE_CHROMATICITY;
          CURRENT_TOTAL_CHROM_DEGREE=TOTAL_CHROM_DEGREE;
        END;
        WHEN(NODE_CHROMATICITY = CURRENT_CHROMATICITY &
        TOTAL_CHROM_DEGREE > CURRENT_TOTAL_CHROM_DEGREE)
        DO;
          CURRENT_NODE_PTR=NODE_PTR;
          CURRENT_TOTAL_CHROM_DEGREE=TOTAL_CHROM_DEGREE;
        END;
        OTHERWISE;
      END;
      NODE_PTR=U1_F_PTR;
    END;
```

```
/* COLOR THE SELECTED NODE.                                    */

   CALL COLOR_NODE;

/* SELECT REMAINING NODES TO BE COLORED WITH THE CURRENT */
/* COLOR ACCORDING TO:                                        */
/*   1. MAXIMUM CHROMATICITY                                  */
/*   2. MAXIMUM CHROMATIC DEGREE IN U2                        */
/*   3. MINIMUM CHROMATIC DEGREE IN U1                        */

   DO WHILE(U1_HEAD_PTR¬=NULL);
     CURRENT_CHROMATICITY=0;
     NODE_PTR=U1_HEAD_PTR;
     DO WHILE(NODE_PTR¬=NULL);
       NODE_CHROMATICITY=CHROMATICITY;
       SELECT;
         WHEN(NODE_CHROMATICITY > CURRENT_CHROMATICITY)
         DO;
           CURRENT_NODE_PTR=NODE_PTR;
           CURRENT_CHROMATICITY=NODE_CHROMATICITY;
           CURRENT_U1_CHROM_DEGREE=U1_CHROM_DEGREE;
           CURRENT_U2_CHROM_DEGREE=
             TOTAL_CHROM_DEGREE-CURRENT_U1_CHROM_DEGREE;
         END;
         WHEN(NODE_CHROMATICITY = CURRENT_CHROMATICITY)
         DO;
           U2_CHROM_DEGREE=TOTAL_CHROM_DEGREE-
             U1_CHROM_DEGREE;
           SELECT;
             WHEN
             (U2_CHROM_DEGREE > CURRENT_U2_CHROM_DEGREE)
             DO;
               CURRENT_NODE_PTR=NODE_PTR;
               CURRENT_U1_CHROM_DEGREE=U1_CHROM_DEGREE;
               CURRENT_U2_CHROM_DEGREE=U2_CHROM_DEGREE;
             END;
             WHEN
             (U2_CHROM_DEGREE = CURRENT_U2_CHROM_DEGREE
             & U1_CHROM_DEGREE < CURRENT_U1_CHROM_DEGREE)
             DO;
               CURRENT_NODE_PTR=NODE_PTR;
               CURRENT_U1_CHROM_DEGREE=U1_CHROM_DEGREE;
             END;
             OTHERWISE;
           END;
         END;
         OTHERWISE;
       END;
       NODE_PTR=U1_F_PTR;
     END;

/* COLOR THE SELECTED NODE.                                    */

     CALL COLOR_NODE;
```

```
    END;

/* U1 IS EMPTY.                                                  */
/* ARE ALL NODES COLORED?                                        */

    IF AUX_HEAD_PTR=NULL
      THEN DO;

/* ALL NODES ARE COLORED. COMPLETE COLORED NODE LIST.            */

        CURRENT_NODE_PTR->AUX_F_PTR=NULL;
        AUX_HEAD_PTR=COLOR_HEAD_PTR;
        RETURN;
      END;
      ELSE;

/* CREATE NEW U1 LIST FOR THE NEXT COLOR.                        */

    DO UNTIL(U1_HEAD_PTR¬=NULL);
      NODE_PTR=AUX_HEAD_PTR;
      CURRENT_COLOR=NEXT_COLOR;
      NEXT_COLOR=32767;
      DO WHILE(NODE_PTR¬=NULL);
        IF LOWER_BOUND=CURRENT_COLOR
          THEN DO;

/* INITIALIZE U1_CHROM_DEGREE OF EACH NODE IN NEW U1             */
/* LIST.                                                         */

            U1_CHROM_DEGREE=CHROMATICITY;

/* APPEND NODE TO U1 LIST.                                       */

            IF U1_HEAD_PTR=NULL
              THEN U1_HEAD_PTR=NODE_PTR;
              ELSE U1_TAIL_PTR->U1_F_PTR=NODE_PTR;
            U1_B_PTR=U1_TAIL_PTR;
            U1_TAIL_PTR=NODE_PTR;
          END;

          ELSE IF NEXT_COLOR > LOWER_BOUND
            THEN NEXT_COLOR=LOWER_BOUND;
            ELSE;
        NODE_PTR=AUX_F_PTR;
      END;
    END;

/* END NEW U1 LIST.                                              */

    U1_TAIL_PTR->U1_F_PTR=NULL;

/* COMPUTE U1 CHROMATIC DEGREE OF EACH NODE IN U1 LIST.          */

    NODE_PTR=U1_HEAD_PTR;
```

```
    DO WHILE(NODE_PTR¬=NULL);
      NODE_CHROMATICITY=CHROMATICITY;
      DO I=1 TO DEGREE;
        ADJ_PTR=ADJ_NODE_PTR(I);
        IF ADJ_PTR->U1_CHROM_DEGREE > 0
          THEN ADJ_PTR->U1_CHROM_DEGREE=
            ADJ_PTR->U1_CHROM_DEGREE+NODE_CHROMATICITY;
      END;
      NODE_PTR=U1_F_PTR;
    END;
  END;

/* COLOR THE NODE POINTED TO BY CURRENT_NODE_PTR.            */

COLOR_NODE: PROC;
  NODE_PTR=CURRENT_NODE_PTR;

/* REMOVE THE NODE TO BE COLORED FROM THE UNCOLORED NODE */
/* LIST.                                                 */

  IF AUX_F_PTR=NULL
    THEN AUX_TAIL_PTR=AUX_B_PTR;
    ELSE AUX_F_PTR->AUX_B_PTR=AUX_B_PTR;
  IF AUX_B_PTR=NULL
    THEN AUX_HEAD_PTR=AUX_F_PTR;
    ELSE AUX_B_PTR->AUX_F_PTR=AUX_F_PTR;

/* REMOVE THE NODE TO BE COLORED FROM THE U1 NODE LIST.  */

  U1_CHROM_DEGREE=0;
  IF U1_F_PTR=NULL
    THEN U1_TAIL_PTR=U1_B_PTR;
    ELSE U1_F_PTR->U1_B_PTR=U1_B_PTR;
  IF U1_B_PTR=NULL
    THEN U1_HEAD_PTR=U1_F_PTR;
    ELSE U1_B_PTR->U1_F_PTR=U1_F_PTR;

/* PLACE THE NODE TO BE COLORED AT THE END OF THE        */
/* COLORED NODE LIST.                                    */

  IF COLOR_TAIL_PTR=NULL
    THEN COLOR_HEAD_PTR=NODE_PTR;
    ELSE COLOR_TAIL_PTR->AUX_F_PTR=NODE_PTR;
  COLOR_TAIL_PTR=NODE_PTR;

/* ASSIGN COLORS TO THE NODE.                            */

  LO_COLOR=CURRENT_COLOR;
  LB=CURRENT_COLOR+CURRENT_CHROMATICITY;
  HI_COLOR=LB-1;
  MAX_COLOR=MAX(MAX_COLOR,HI_COLOR);
```

```
/* ADJUST DEGREE INFORMATION OF ADJACENT NODES.                    */

  DO I=1 TO DEGREE;
    ADJ_PTR=ADJ_NODE_PTR(I);
    IF ADJ_PTR->LO_COLOR=0
      THEN DO;

/*      ADJACENT NODE IS UNCOLORED.                                */

        ADJ_PTR->TOTAL_CHROM_DEGREE=
          ADJ_PTR->TOTAL_CHROM_DEGREE-CURRENT_CHROMATICITY;
        IF LB > ADJ_PTR->LOWER_BOUND
          THEN ADJ_PTR->LOWER_BOUND=LB;
          ELSE;
        IF ADJ_PTR->U1_CHROM_DEGREE > 0
          THEN DO;

/*          ADJACENT NODE IS IN U1.  REMOVE IT FROM U1.            */

            ADJ_PTR->U1_CHROM_DEGREE=0;
            IF ADJ_PTR->U1_F_PTR=NULL
              THEN U1_TAIL_PTR=ADJ_PTR->U1_B_PTR;
              ELSE ADJ_PTR->U1_F_PTR->U1_B_PTR=
                ADJ_PTR->U1_B_PTR;
            IF ADJ_PTR->U1_B_PTR=NULL
              THEN U1_HEAD_PTR=ADJ_PTR->U1_F_PTR;
              ELSE ADJ_PTR->U1_B_PTR->U1_F_PTR=
                ADJ_PTR->U1_F_PTR;
            IF NEXT_COLOR > LB
              THEN NEXT_COLOR=LB;
              ELSE;

/*          REDUCE U1 DEGREES.                                     */

            ADJ_CHROMATICITY=ADJ_PTR->CHROMATICITY;
            DO K=1 TO ADJ_PTR->DEGREE;
              ADJ2_PTR=ADJ_PTR->ADJ_NODE_PTR(K);
              IF ADJ2_PTR->U1_CHROM_DEGREE > 0
                THEN ADJ2_PTR->U1_CHROM_DEGREE=
                  ADJ2_PTR->U1_CHROM_DEGREE-
                  ADJ_CHROMATICITY;
                ELSE;
            END;
          END;
          ELSE;
      END;
      ELSE;
  END;
END COLOR_NODE;
END RLF1;
```

```
RLFD1: PROC(HEAD_PTR,AUX_HEAD_PTR,MAX_COLOR) REORDER;
  DCL
    ADJ_PTR                  PTR,
    ADJ2_PTR                 PTR,
    AUX_HEAD_PTR             PTR,
    AUX_TAIL_PTR             PTR,
    COLOR_HEAD_PTR           PTR,
    COLOR_TAIL_PTR           PTR,
    CURRENT_CHROMATICITY     FIXED BINARY(15),
    CURRENT_COLOR            FIXED BINARY(15) INIT(1),
    CURRENT_NODE_PTR         PTR,
    CURRENT_TOTAL_DEGREE     FIXED BINARY(15),
    CURRENT_U1_DEGREE        FIXED BINARY(15),
    CURRENT_U2_DEGREE        FIXED BINARY(15),
    DEG                      FIXED BINARY(15),
    HEAD_PTR                 PTR,
    (I,K)                    FIXED BINARY(15),
    LB                       FIXED BINARY(15),
    MAX                      BUILTIN,
    MAX_COLOR                FIXED BINARY(15),
    NEXT_COLOR               FIXED BINARY(15) INIT(32767),
    NODE_CHROMATICITY        FIXED BINARY(15),
    NODE_PTR                 PTR,
    NULL                     BUILTIN,
    U1_HEAD_PTR              PTR,
    U1_TAIL_PTR              PTR,
    U2_DEGREE                FIXED BINARY(15);

  DCL
    1 NODE                   BASED(NODE_PTR),
      2 NUMBER               FIXED BINARY(15),
      2 DEGREE               FIXED BINARY(15),
      2 CHROMATICITY         FIXED BINARY(15),
      2 CHROMATIC_DEGREE     FIXED BINARY(15),
      2 LO_COLOR             FIXED BINARY(15),
      2 HI_COLOR             FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,    /* 40 BYTES */
        3 TOTAL_DEGREE       FIXED BINARY(15),
        3 U1_DEGREE          FIXED BINARY(15),
        3 LOWER_BOUND        FIXED BINARY(15),
        3 FILLER             CHAR(18),
        3 U1_B_PTR           PTR,
        3 U1_F_PTR           PTR,
        3 AUX_B_PTR          PTR,
        3 AUX_F_PTR          PTR,
      2 FORWARD_PTR          PTR,
      2 ADJ_NODE_PTR(1:DEG REFER(DEGREE)) PTR;

/* INITIALIZE COLORED NODE LIST.                                    */

  COLOR_HEAD_PTR=NULL;
  COLOR_TAIL_PTR=NULL;
  MAX_COLOR=0;
```

```
/* CREATE UNCOLORED NODE LIST AND U1 NODE LIST.                  */
/* INITIALIZE WORK VARIABLES FOR EACH NODE.                      */

  AUX_HEAD_PTR=HEAD_PTR;
  U1_HEAD_PTR=HEAD_PTR;
  NODE_PTR=HEAD_PTR;
  U1_TAIL_PTR=NULL;
  DO WHILE(NODE_PTR¬=NULL);
    DEG=DEGREE+1;
    U1_DEGREE=DEG;
    TOTAL_DEGREE=DEG;
    LOWER_BOUND=1;
    LO_COLOR=0;
    AUX_F_PTR=FORWARD_PTR;
    U1_F_PTR=FORWARD_PTR;
    AUX_B_PTR=U1_TAIL_PTR;
    U1_B_PTR=U1_TAIL_PTR;
    U1_TAIL_PTR=NODE_PTR;
    NODE_PTR=FORWARD_PTR;
  END;
  AUX_TAIL_PTR=U1_TAIL_PTR;

/* SELECT NODES TO COLOR UNTIL ALL NODES ARE COLORED.            */

  DO WHILE('1'B);

/* SELECT FIRST NODE TO BE COLORED WITH THE CURRENT              */
/* COLOR ACCORDING TO:                                           */
/*   1. MAXIMUM CHROMATICITY                                     */
/*   2. MAXIMUM DEGREE                                           */

    CURRENT_CHROMATICITY=0;
    NODE_PTR=U1_HEAD_PTR;
    DO WHILE(NODE_PTR¬=NULL);
      NODE_CHROMATICITY=CHROMATICITY;
      SELECT;
        WHEN(NODE_CHROMATICITY > CURRENT_CHROMATICITY) DO;
          CURRENT_NODE_PTR=NODE_PTR;
          CURRENT_CHROMATICITY=NODE_CHROMATICITY;
          CURRENT_TOTAL_DEGREE=TOTAL_DEGREE;
        END;
        WHEN(NODE_CHROMATICITY = CURRENT_CHROMATICITY &
        TOTAL_DEGREE > CURRENT_TOTAL_DEGREE) DO;
          CURRENT_NODE_PTR=NODE_PTR;
          CURRENT_TOTAL_DEGREE=TOTAL_DEGREE;
        END;
        OTHERWISE;
      END;
      NODE_PTR=U1_F_PTR;
    END;

/* COLOR THE SELECTED NODE.                                      */

    CALL COLOR_NODE;
```

```
/* SELECT REMAINING NODES TO BE COLORED WITH THE CURRENT */
/* COLOR ACCORDING TO:                                   */
/*   1. MAXIMUM CHROMATICITY                             */
/*   2. MAXIMUM DEGREE IN U2                             */
/*   3. MINIMUM DEGREE IN U1                             */

    DO WHILE(U1_HEAD_PTR¬=NULL);
      CURRENT_CHROMATICITY=0;
      NODE_PTR=U1_HEAD_PTR;
      DO WHILE(NODE_PTR¬=NULL);
        NODE_CHROMATICITY=CHROMATICITY;
        SELECT;
          WHEN(NODE_CHROMATICITY > CURRENT_CHROMATICITY)
          DO;
            CURRENT_NODE_PTR=NODE_PTR;
            CURRENT_CHROMATICITY=NODE_CHROMATICITY;
            CURRENT_U1_DEGREE=U1_DEGREE;
            CURRENT_U2_DEGREE=
              TOTAL_DEGREE-CURRENT_U1_DEGREE;
          END;
          WHEN(NODE_CHROMATICITY = CURRENT_CHROMATICITY)
          DO;
            U2_DEGREE=TOTAL_DEGREE-U1_DEGREE;
            SELECT;
              WHEN
              (U2_DEGREE > CURRENT_U2_DEGREE)
              DO;
                CURRENT_NODE_PTR=NODE_PTR;
                CURRENT_U1_DEGREE=U1_DEGREE;
                CURRENT_U2_DEGREE=U2_DEGREE;
              END;
              WHEN
              (U2_DEGREE = CURRENT_U2_DEGREE
              & U1_DEGREE < CURRENT_U1_DEGREE)
              DO;
                CURRENT_NODE_PTR=NODE_PTR;
                CURRENT_U1_DEGREE=U1_DEGREE;
              END;
              OTHERWISE;
            END;
          END;
          OTHERWISE;
        END;
        NODE_PTR=U1_F_PTR;
      END;

/* COLOR THE SELECTED NODE.                              */

      CALL COLOR_NODE;
    END;
```

```
/* U1 IS EMPTY.                                                    */
/* ARE ALL NODES COLORED?                                          */

     IF AUX_HEAD_PTR=NULL
       THEN DO;

/* ALL NODES ARE COLORED. COMPLETE COLORED NODE LIST.              */

         CURRENT_NODE_PTR->AUX_F_PTR=NULL;
         AUX_HEAD_PTR=COLOR_HEAD_PTR;
         RETURN;
       END;
       ELSE;

/* CREATE NEW U1 LIST FOR THE NEXT COLOR.                          */

     DO UNTIL(U1_HEAD_PTR¬=NULL);
       NODE_PTR=AUX_HEAD_PTR;
       CURRENT_COLOR=NEXT_COLOR;
       NEXT_COLOR=32767;
       DO WHILE(NODE_PTR¬=NULL);
         IF LOWER_BOUND=CURRENT_COLOR
           THEN DO;

/* INITIALIZE U1_DEGREE OF EACH NODE IN NEW U1 LIST.               */

             U1_DEGREE=1;

/* APPEND NODE TO U1 LIST.                                         */

             IF U1_HEAD_PTR=NULL
               THEN U1_HEAD_PTR=NODE_PTR;
               ELSE U1_TAIL_PTR->U1_F_PTR=NODE_PTR;
             U1_B_PTR=U1_TAIL_PTR;
             U1_TAIL_PTR=NODE_PTR;
           END;

           ELSE IF NEXT_COLOR > LOWER_BOUND
             THEN NEXT_COLOR=LOWER_BOUND;
             ELSE;
         NODE_PTR=AUX_F_PTR;
       END;
     END;

/* END NEW U1 LIST.                                                */

     U1_TAIL_PTR->U1_F_PTR=NULL;

/* COMPUTE U1 DEGREE OF EACH NODE IN U1 LIST.                      */

     NODE_PTR=U1_HEAD_PTR;
     DO WHILE(NODE_PTR¬=NULL);
       DO I=1 TO DEGREE;
         ADJ_PTR=ADJ_NODE_PTR(I);
```

```
      IF ADJ_PTR->U1_DEGREE > 0
        THEN ADJ_PTR->U1_DEGREE=ADJ_PTR->U1_DEGREE+1;
    END;
    NODE_PTR=U1_F_PTR;
  END;
END;

/* COLOR THE NODE POINTED TO BY CURRENT_NODE_PTR.          */

COLOR_NODE: PROC;
  NODE_PTR=CURRENT_NODE_PTR;

/* REMOVE THE NODE TO BE COLORED FROM THE UNCOLORED NODE */
/* LIST.                                                  */

  IF AUX_F_PTR=NULL
    THEN AUX_TAIL_PTR=AUX_B_PTR;
    ELSE AUX_F_PTR->AUX_B_PTR=AUX_B_PTR;
  IF AUX_B_PTR=NULL
    THEN AUX_HEAD_PTR=AUX_F_PTR;
    ELSE AUX_B_PTR->AUX_F_PTR=AUX_F_PTR;

/* REMOVE THE NODE TO BE COLORED FROM THE U1 NODE LIST.  */

  U1_DEGREE=0;
  IF U1_F_PTR=NULL
    THEN U1_TAIL_PTR=U1_B_PTR;
    ELSE U1_F_PTR->U1_B_PTR=U1_B_PTR;
  IF U1_B_PTR=NULL
    THEN U1_HEAD_PTR=U1_F_PTR;
    ELSE U1_B_PTR->U1_F_PTR=U1_F_PTR;

/* PLACE THE NODE TO BE COLORED AT THE END OF THE         */
/* COLORED NODE LIST.                                     */

  IF COLOR_TAIL_PTR=NULL
    THEN COLOR_HEAD_PTR=NODE_PTR;
    ELSE COLOR_TAIL_PTR->AUX_F_PTR=NODE_PTR;
  COLOR_TAIL_PTR=NODE_PTR;

/* ASSIGN COLORS TO THE NODE.                             */

  LO_COLOR=CURRENT_COLOR;
  LB=CURRENT_COLOR+CURRENT_CHROMATICITY;
  HI_COLOR=LB-1;
  MAX_COLOR=MAX(MAX_COLOR,HI_COLOR);

/* ADJUST DEGREE INFORMATION OF ADJACENT NODES.           */

  DO I=1 TO DEGREE;
    ADJ_PTR=ADJ_NODE_PTR(I);
    IF ADJ_PTR->LO_COLOR=0
      THEN DO;
```

```
/*       ADJACENT NODE IS UNCOLORED.                                */

         ADJ_PTR->TOTAL_DEGREE=ADJ_PTR->TOTAL_DEGREE-1;
         IF LB > ADJ_PTR->LOWER_BOUND
           THEN ADJ_PTR->LOWER_BOUND=LB;
           ELSE;
         IF ADJ_PTR->U1_DEGREE > 0
           THEN DO;

/*           ADJACENT NODE IS IN U1.  REMOVE IT FROM U1.            */

             ADJ_PTR->U1_DEGREE=0;
             IF ADJ_PTR->U1_F_PTR=NULL
               THEN U1_TAIL_PTR=ADJ_PTR->U1_B_PTR;
               ELSE ADJ_PTR->U1_F_PTR->U1_B_PTR=
                 ADJ_PTR->U1_B_PTR;
             IF ADJ_PTR->U1_B_PTR=NULL
               THEN U1_HEAD_PTR=ADJ_PTR->U1_F_PTR;
               ELSE ADJ_PTR->U1_B_PTR->U1_F_PTR=
                 ADJ_PTR->U1_F_PTR;
             IF NEXT_COLOR > LB
               THEN NEXT_COLOR=LB;
               ELSE;

/*           REDUCE U1 DEGREES.                                     */

             DO K=1 TO ADJ_PTR->DEGREE;
               ADJ2_PTR=ADJ_PTR->ADJ_NODE_PTR(K);
               IF ADJ2_PTR->U1_DEGREE > 0
                 THEN ADJ2_PTR->U1_DEGREE=
                   ADJ2_PTR->U1_DEGREE-1;
                 ELSE;
             END;
           END;
           ELSE;
       END;
       ELSE;
   END;
END COLOR_NODE;
END RLFD1;
```

```
DYNPH: PROC(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES,MAX_COLOR)
       REORDER;
  DCL
    ADJ_COLOR_BIT               BIT(1),
    ADJ_NODE_NO                 FIXED BINARY(15),
    ADJ_PTR                     POINTER,
    AUX_HEAD_PTR                POINTER,
    AUX_TAIL_PTR                POINTER,
    CEIL                        BUILTIN,
    COLOR_COUNT                 FIXED BINARY(15),
    COLOR_DATA_PTR              POINTER,
    COLOR_NO                    FIXED BINARY(15),
    HEAD_PTR                    POINTER,
    LOCATOR(NO_OF_NODES)        POINTER,
    MAX                         BUILTIN,
    MAX_COLOR                   FIXED BINARY(15),
    NG                          FIXED BINARY(15),
    NO_OF_ADJ_COLORS            FIXED BINARY(15),
    NO_OF_NODES                 FIXED BINARY(15),
    NODE_NO                     FIXED BINARY(15),
    NODE_PTR                    POINTER,
    NULL                        BUILTIN,
    OLD_COLOR_DATA_PTR          POINTER,
    PREV_COLOR_BIT              BIT(1),
    REMAINING_NODES             FIXED BINARY(15),
    SUBSCRIPT                   FIXED BINARY(15),
    TEMP_PTR                    POINTER,
    UC                          FIXED BINARY(15),
    UNAVAILABLE_COLOR           FIXED BINARY(15);

  DCL
    1 NODE                      BASED(NODE_PTR),
      2 NUMBER                  FIXED BINARY(15),
      2 DEGREE                  FIXED BINARY(15),
      2 CHROMATICITY            FIXED BINARY(15),
      2 CHROMATIC_DEGREE        FIXED BINARY(15),
      2 LO_COLOR                FIXED BINARY(15),
      2 HI_COLOR                FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,     /* 40 BYTES */
        3 USED_COLORS           FIXED BINARY(15),
        3 NO_OF_GAPS            FIXED BINARY(15),
        3 REDUCED_DEGREE        FIXED BINARY(15),
        3 REDUCED_CHROMATIC_DEGREE FIXED BINARY(15),
        3 MAX_ADJ_COLOR         FIXED BINARY(15),
        3 HEAP_POSITION         FIXED BINARY(15),
        3 PH                    FIXED BINARY(31),
        3 ADJ_COLOR_PTR         POINTER,
        3 FILLER                CHAR(16),
        3 AUX_FORWARD_PTR       POINTER,
      2 FORWARD_PTR             POINTER,
      2 ADJ_NODE_PTR(DEG REFER(DEGREE)) POINTER;
```

```
DCL
  1 COLOR_DATA              BASED(COLOR_DATA_PTR),
    2 ADJ_COLOR_SIZE        FIXED BINARY(15),
    2 ADJ_COLOR(NO_OF_ADJ_COLORS REFER(ADJ_COLOR_SIZE))
                              BIT(1);

/* ALLOCATE ADJACENT COLOR ARRAY FOR EACH NODE.                */
/* INITIALIZE WORK VARIABLES FOR EACH NODE.                    */

  NO_OF_ADJ_COLORS=16;
  NODE_PTR=HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);
    ALLOCATE COLOR_DATA;
    ADJ_COLOR_PTR=COLOR_DATA_PTR;

    ADJ_COLOR(*)='0'B;
    LO_COLOR=0;
    HI_COLOR=0;
    USED_COLORS=0;
    NO_OF_GAPS=0;
    REDUCED_DEGREE=DEGREE;
    REDUCED_CHROMATIC_DEGREE=CHROMATIC_DEGREE-CHROMATICITY;
    PH=REDUCED_CHROMATIC_DEGREE+REDUCED_DEGREE*
      (CHROMATICITY-1);
    MAX_ADJ_COLOR=0;

    NODE_PTR=FORWARD_PTR;
  END;

/* INITIALIZE LOCATOR ARRAY FOR THE HEAP.                      */

  NODE_PTR=HEAD_PTR;
  DO NODE_NO = 1 TO NO_OF_NODES;
    LOCATOR(NODE_NO)=NODE_PTR;
    HEAP_POSITION=NODE_NO;
    NODE_PTR=FORWARD_PTR;
  END;

/* CREATE THE HEAP.                                            */

  DO NODE_NO=2 TO NO_OF_NODES;
    NODE_PTR=LOCATOR(NODE_NO);
    CALL UPHEAP(NODE_PTR);
  END;

/* COLOR THE NODE OF HIGHEST PIGEONHOLE MEASURE.               */

  MAX_COLOR=0;
  NODE_PTR=LOCATOR(1);
  AUX_HEAD_PTR=NODE_PTR;
  AUX_TAIL_PTR=NODE_PTR;
  LO_COLOR=1;
  HI_COLOR=CHROMATICITY;
  REMAINING_NODES=NO_OF_NODES;
```

```
  DO WHILE(REMAINING_NODES>1);

/* FILL THE TOP OF HEAP WITH NEW NODE.                         */
    TEMP_PTR=LOCATOR(REMAINING_NODES);
    TEMP_PTR->HEAP_POSITION=1;
    LOCATOR(1)=TEMP_PTR;
    REMAINING_NODES=REMAINING_NODES-1;
    CALL DNHEAP(TEMP_PTR);

/* ADJUST THE PIGEONHOLE MEASURE OF THOSE NODES WITH           */
/* "NEW" GAPS AT THE END OF THE COLOR RANGE.                   */

    IF HI_COLOR>MAX_COLOR
      THEN DO;
        DO SUBSCRIPT=1 TO REMAINING_NODES;
          TEMP_PTR=LOCATOR(SUBSCRIPT);
          IF TEMP_PTR->MAX_ADJ_COLOR = MAX_COLOR
            THEN DO;
              TEMP_PTR->PH=TEMP_PTR->PH+
               TEMP_PTR->CHROMATICITY-1;
              TEMP_PTR->NO_OF_GAPS=TEMP_PTR->NO_OF_GAPS+1;
              CALL UPHEAP(TEMP_PTR);
            END;
        END;

/* THE MAXIMUM NUMBER OF COLORS HAS INCREASED.  UPDATE         */
/* THE MAXIMUM NUMBER OF COLORS USED.                          */

        MAX_COLOR=HI_COLOR;
      END;

/* UPDATE NODES ADJACENT TO COLORED NODE.                      */

    DO ADJ_NODE_NO=1 TO DEGREE;
      ADJ_PTR=ADJ_NODE_PTR(ADJ_NODE_NO);
      IF ADJ_PTR->LO_COLOR>0 THEN GO TO NEXT_ADJ_NODE;
      COLOR_DATA_PTR=ADJ_PTR->ADJ_COLOR_PTR;

/* ENLARGE ADJACENT COLORS ARRAY, IF NECESSARY.                */

      IF HI_COLOR>ADJ_COLOR_SIZE
        THEN DO;
          NO_OF_ADJ_COLORS=CEIL(HI_COLOR/16)*16;
          OLD_COLOR_DATA_PTR=COLOR_DATA_PTR;
          ALLOCATE COLOR_DATA;
          ADJ_PTR->ADJ_COLOR_PTR=COLOR_DATA_PTR;
          DO COLOR_NO=1 TO ADJ_PTR->MAX_ADJ_COLOR;
            ADJ_COLOR(COLOR_NO)=
              OLD_COLOR_DATA_PTR->ADJ_COLOR(COLOR_NO);
          END;
          DO COLOR_NO=ADJ_PTR->MAX_ADJ_COLOR+1 TO
                      NO_OF_ADJ_COLORS;
            ADJ_COLOR(COLOR_NO)='0'B;
```

```
          END;
          FREE OLD_COLOR_DATA_PTR->COLOR_DATA;
        END;

/* UPDATE MAX ADJACENT COLOR.                                      */

      ADJ_PTR->MAX_ADJ_COLOR=
        MAX(HI_COLOR,ADJ_PTR->MAX_ADJ_COLOR);

/* UPDATE ADJACENT COLORS ARRAY, NUMBER OF GAPS, AND               */
/* NUMBER OF USED COLORS.                                          */

      UC=ADJ_PTR->USED_COLORS;
      NG=ADJ_PTR->NO_OF_GAPS;
      IF LO_COLOR>1
        THEN PREV_COLOR_BIT=ADJ_COLOR(LO_COLOR-1);
        ELSE PREV_COLOR_BIT='1'B;
      IF ¬PREV_COLOR_BIT
        THEN NG=NG+1;
      DO COLOR_NO=LO_COLOR TO HI_COLOR;
        ADJ_COLOR_BIT=ADJ_COLOR(COLOR_NO);
        ADJ_COLOR(COLOR_NO)='1'B;
        IF ¬PREV_COLOR_BIT & ADJ_COLOR_BIT
          THEN NG=NG-1;    /* END OF A GAP. */
        IF ¬ADJ_COLOR_BIT
          THEN UC=UC+1;    /* ANOTHER COLOR USED. */
        PREV_COLOR_BIT=ADJ_COLOR_BIT;
      END;
      IF ¬PREV_COLOR_BIT
        THEN DO;
          IF HI_COLOR<ADJ_PTR->MAX_ADJ_COLOR
            THEN ADJ_COLOR_BIT=ADJ_COLOR(HI_COLOR+1);
            ELSE ADJ_COLOR_BIT=(MAX_COLOR=HI_COLOR);
          IF ADJ_COLOR_BIT
            THEN NG=NG-1;
        END;

/* UPDATE ALL WORK VARIABLES.                                      */

      ADJ_PTR->USED_COLORS=UC;
      ADJ_PTR->NO_OF_GAPS=NG;
      ADJ_PTR->REDUCED_DEGREE=ADJ_PTR->REDUCED_DEGREE-1;
      ADJ_PTR->REDUCED_CHROMATIC_DEGREE=
        ADJ_PTR->REDUCED_CHROMATIC_DEGREE-CHROMATICITY;
      ADJ_PTR->PH=UC+ADJ_PTR->REDUCED_CHROMATIC_DEGREE
        +(NG+ADJ_PTR->REDUCED_DEGREE)
        *(ADJ_PTR->CHROMATICITY-1);

/* ADJUST ADJACENT NODE'S LOCATION IN THE HEAP.                    */

      CALL DNHEAP(ADJ_PTR);

NEXT_ADJ_NODE:
    END;
```

```
/* COLOR THE NODE OF HIGHEST PIGEONHOLE MEASURE.                    */

    NODE_PTR=LOCATOR(1);
    AUX_TAIL_PTR->AUX_FORWARD_PTR=NODE_PTR;
    AUX_TAIL_PTR=NODE_PTR;

/* DETERMINE LOWEST SEQUENCE OF AVAILABLE COLORS.                   */

    COLOR_DATA_PTR=ADJ_COLOR_PTR;
    UNAVAILABLE_COLOR=0;
    COLOR_COUNT=0;
    DO COLOR_NO=1 TO MAX_ADJ_COLOR
                UNTIL(COLOR_COUNT=CHROMATICITY);
      IF ADJ_COLOR(COLOR_NO)
        THEN DO;
          COLOR_COUNT=0;
          UNAVAILABLE_COLOR=COLOR_NO;
        END;
        ELSE COLOR_COUNT=COLOR_COUNT+1;
    END;

/* ASSIGN THE COLORS.                                               */

    LO_COLOR=UNAVAILABLE_COLOR+1;
    HI_COLOR=UNAVAILABLE_COLOR+CHROMATICITY;
  END;

/* COMPLETE COLORED NODE LIST.                                      */

  AUX_FORWARD_PTR=NULL;
  MAX_COLOR=MAX(MAX_COLOR,HI_COLOR);

/* FREE ADJACENT COLORS STRUCTURE FOR EACH NODE.                    */

  NODE_PTR=HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);
    COLOR_DATA_PTR=ADJ_COLOR_PTR;
    FREE COLOR_DATA;
    NODE_PTR=FORWARD_PTR;
  END;
  RETURN;

UPHEAP: PROC(NODE_PTR);
  DCL
    NODE_PH                     FIXED BINARY(31),
    NODE_PTR                    POINTER,
    NODE_SUBSCRIPT              FIXED BINARY(15),
    PARENT_PTR                  POINTER,
    PARENT_SUBSCRIPT            FIXED BINARY(15);
  NODE_SUBSCRIPT=NODE_PTR->HEAP_POSITION;
  NODE_PH=NODE_PTR->PH;
  DO WHILE(NODE_SUBSCRIPT>1);
    PARENT_SUBSCRIPT=NODE_SUBSCRIPT/2;
```

```
    PARENT_PTR=LOCATOR(PARENT_SUBSCRIPT);
    IF NODE_PH<=PARENT_PTR->PH
      THEN LEAVE;
    LOCATOR(NODE_SUBSCRIPT)=PARENT_PTR;
    PARENT_PTR->HEAP_POSITION=NODE_SUBSCRIPT;
    NODE_SUBSCRIPT=PARENT_SUBSCRIPT;
  END;
  LOCATOR(NODE_SUBSCRIPT)=NODE_PTR;
  NODE_PTR->HEAP_POSITION=NODE_SUBSCRIPT;
  RETURN;
END UPHEAP;

DNHEAP: PROC(NODE_PTR);
  DCL
    CHILD_PH                 FIXED BINARY(31),
    CHILD_PTR                POINTER,
    CHILD_SUBSCRIPT          FIXED BINARY(15),
    NODE_PH                  FIXED BINARY(31),
    NODE_PTR                 POINTER,
    NODE_SUBSCRIPT           FIXED BINARY(15),
    RIGHT_PH                 FIXED BINARY(31),
    RIGHT_PTR                POINTER,
    RIGHT_SUBSCRIPT          FIXED BINARY(15);
  NODE_SUBSCRIPT=NODE_PTR->HEAP_POSITION;
  NODE_PH=NODE_PTR->PH;
  CHILD_SUBSCRIPT=2*NODE_SUBSCRIPT;
  DO WHILE(CHILD_SUBSCRIPT<=REMAINING_NODES);
    CHILD_PTR=LOCATOR(CHILD_SUBSCRIPT);
    CHILD_PH=CHILD_PTR->PH;
    RIGHT_SUBSCRIPT=CHILD_SUBSCRIPT+1;
    IF RIGHT_SUBSCRIPT<=REMAINING_NODES
      THEN DO;
        RIGHT_PTR=LOCATOR(RIGHT_SUBSCRIPT);
        RIGHT_PH=RIGHT_PTR->PH;
        IF RIGHT_PH>CHILD_PH
          THEN DO;
            CHILD_SUBSCRIPT=RIGHT_SUBSCRIPT;
            CHILD_PTR=RIGHT_PTR;
            CHILD_PH=RIGHT_PH;
          END;
        END;
      IF NODE_PH>=CHILD_PH
        THEN LEAVE;
      LOCATOR(NODE_SUBSCRIPT)=CHILD_PTR;
      CHILD_PTR->HEAP_POSITION=NODE_SUBSCRIPT;
      NODE_SUBSCRIPT=CHILD_SUBSCRIPT;
      CHILD_SUBSCRIPT=2*NODE_SUBSCRIPT;
    END;
    LOCATOR(NODE_SUBSCRIPT)=NODE_PTR;
    NODE_PTR->HEAP_POSITION=NODE_SUBSCRIPT;
    RETURN;
  END;
END DYNPH;
```

```
DYNFPH: PROC(HEAD_PTR,AUX_HEAD_PTR,NO_OF_NODES,MAX_COLOR)
        REORDER;
  DCL
    ADJ_COLOR_BIT               BIT(1),
    ADJ_NODE_NO                 FIXED BINARY(15),
    ADJ_PTR                     POINTER,
    AUX_HEAD_PTR                POINTER,
    AUX_TAIL_PTR                POINTER,
    CEIL                        BUILTIN,
    COLOR_COUNT                 FIXED BINARY(15),
    COLOR_DATA_PTR              POINTER,
    COLOR_NO                    FIXED BINARY(15),
    HEAD_PTR                    POINTER,
    LOCATOR(NO_OF_NODES)        POINTER,
    MAX                         BUILTIN,
    MAX_COLOR                   FIXED BINARY(15),
    NG                          FIXED BINARY(15),
    NO_OF_ADJ_COLORS            FIXED BINARY(15),
    NO_OF_NODES                 FIXED BINARY(15),
    NODE_NO                     FIXED BINARY(15),
    NODE_PTR                    POINTER,
    NULL                        BUILTIN,
    OLD_COLOR_DATA_PTR          POINTER,
    OLD_FPH                     FLOAT BINARY(21),
    PREV_COLOR_BIT              BIT(1),
    REMAINING_NODES             FIXED BINARY(15),
    SUBSCRIPT                   FIXED BINARY(15),
    TEMP_PTR                    POINTER,
    UC                          FIXED BINARY(15),
    UNAVAILABLE_COLOR           FIXED BINARY(15);

  DCL
    1 NODE                      BASED(NODE_PTR),
      2 NUMBER                  FIXED BINARY(15),
      2 DEGREE                  FIXED BINARY(15),
      2 CHROMATICITY            FIXED BINARY(15),
      2 CHROMATIC_DEGREE        FIXED BINARY(15),
      2 LO_COLOR                FIXED BINARY(15),
      2 HI_COLOR                FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,       /* 40 BYTES */
        3 USED_COLORS           FIXED BINARY(15),
        3 NO_OF_GAPS            FIXED BINARY(15),
        3 REDUCED_DEGREE        FIXED BINARY(15),
        3 REDUCED_CHROMATIC_DEGREE FIXED BINARY(15),
        3 MAX_ADJ_COLOR         FIXED BINARY(15),
        3 HEAP_POSITION         FIXED BINARY(15),
        3 PH                    FIXED BINARY(31),
        3 FLOAT_PH              FLOAT BINARY(21),
        3 ADJ_COLOR_PTR         POINTER,
        3 FILLER                CHAR(12),
        3 AUX_FORWARD_PTR       POINTER,
      2 FORWARD_PTR             POINTER,
      2 ADJ_NODE_PTR(DEG REFER(DEGREE)) POINTER;
```

```
  DCL
    1 COLOR_DATA               BASED(COLOR_DATA_PTR),
      2 ADJ_COLOR_SIZE         FIXED BINARY(15),
      2 ADJ_COLOR(NO_OF_ADJ_COLORS REFER(ADJ_COLOR_SIZE))
                                 BIT(1);

/* ALLOCATE ADJACENT COLOR ARRAY FOR EACH NODE.                  */
/* INITIALIZE WORK VARIABLES FOR EACH NODE.                      */

  MAX_COLOR=0;
  NO_OF_ADJ_COLORS=16;
  NODE_PTR=HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);
    ALLOCATE COLOR_DATA;
    ADJ_COLOR_PTR=COLOR_DATA_PTR;

    ADJ_COLOR(*)='0'B;
    LO_COLOR=0;
    HI_COLOR=0;
    USED_COLORS=0;
    NO_OF_GAPS=0;
    REDUCED_DEGREE=DEGREE;
    REDUCED_CHROMATIC_DEGREE=CHROMATIC_DEGREE-CHROMATICITY;
    PH=REDUCED_CHROMATIC_DEGREE+
      REDUCED_DEGREE*(CHROMATICITY-1);
    MAX_ADJ_COLOR=0;
    CALL CALCFPH(NODE_PTR);

    NODE_PTR=FORWARD_PTR;
  END;

/* INITIALIZE LOCATOR ARRAY FOR THE HEAP.                        */

  NODE_PTR=HEAD_PTR;
  DO NODE_NO = 1 TO NO_OF_NODES;
    LOCATOR(NODE_NO)=NODE_PTR;
    HEAP_POSITION=NODE_NO;
    NODE_PTR=FORWARD_PTR;
  END;

/* CREATE THE HEAP.                                              */

  DO NODE_NO=2 TO NO_OF_NODES;
    NODE_PTR=LOCATOR(NODE_NO);
    CALL UPHEAP(NODE_PTR);
  END;

/* COLOR THE NODE OF HIGHEST PIGEONHOLE MEASURE.                 */

  NODE_PTR=LOCATOR(1);
  AUX_HEAD_PTR=NODE_PTR;
  AUX_TAIL_PTR=NODE_PTR;
  LO_COLOR=1;
  HI_COLOR=CHROMATICITY;
```

```
  REMAINING_NODES=NO_OF_NODES;

  DO WHILE(REMAINING_NODES>1);

/* FILL THE TOP OF HEAP WITH NEW NODE.                              */
    TEMP_PTR=LOCATOR(REMAINING_NODES);
    TEMP_PTR->HEAP_POSITION=1;
    LOCATOR(1)=TEMP_PTR;
    REMAINING_NODES=REMAINING_NODES-1;
    CALL DNHEAP(TEMP_PTR);

/* ADJUST THE PIGEONHOLE MEASURE OF THOSE NODES WITH                */
/* "NEW" GAPS AT THE END OF THE COLOR RANGE.                        */

    IF HI_COLOR>MAX_COLOR
      THEN DO;
        DO SUBSCRIPT=1 TO REMAINING_NODES;
          TEMP_PTR=LOCATOR(SUBSCRIPT);
          IF TEMP_PTR->MAX_ADJ_COLOR = MAX_COLOR
            THEN DO;
              TEMP_PTR->PH=TEMP_PTR->PH+
                TEMP_PTR->CHROMATICITY-1;
              TEMP_PTR->NO_OF_GAPS=TEMP_PTR->NO_OF_GAPS+1;
            END;
        END;

/* THE MAXIMUM NUMBER OF COLORS HAS INCREASED.  UPDATE              */
/* THE MAXIMUM NUMBER OF COLORS USED.                               */

        MAX_COLOR=HI_COLOR;

/* CALCULATE NEW FLOATING POINT PIGEONHOLE MEASURE FOR              */
/* EACH NODE.  MAINTAIN THE HEAP.                                   */

        DO SUBSCRIPT=1 TO REMAINING_NODES;
          TEMP_PTR=LOCATOR(SUBSCRIPT);
          CALL CALCFPH(TEMP_PTR);
          CALL UPHEAP(TEMP_PTR);
        END;
      END;

/* UPDATE NODES ADJACENT TO COLORED NODE.                           */

    DO ADJ_NODE_NO=1 TO DEGREE;
      ADJ_PTR=ADJ_NODE_PTR(ADJ_NODE_NO);
      IF ADJ_PTR->LO_COLOR>0 THEN GO TO NEXT_ADJ_NODE;
      COLOR_DATA_PTR=ADJ_PTR->ADJ_COLOR_PTR;

/* ENLARGE ADJACENT COLORS ARRAY, IF NECESSARY.                     */

      IF HI_COLOR>ADJ_COLOR_SIZE
        THEN DO;
          NO_OF_ADJ_COLORS=CEIL(HI_COLOR/16)*16;
          OLD_COLOR_DATA_PTR=COLOR_DATA_PTR;
```

```
      ALLOCATE COLOR_DATA;
      ADJ_PTR->ADJ_COLOR_PTR=COLOR_DATA_PTR;
      DO COLOR_NO=1 TO ADJ_PTR->MAX_ADJ_COLOR;
        ADJ_COLOR(COLOR_NO)=
          OLD_COLOR_DATA_PTR->ADJ_COLOR(COLOR_NO);
      END;
      DO COLOR_NO=ADJ_PTR->MAX_ADJ_COLOR+1 TO
                     NO_OF_ADJ_COLORS;
        ADJ_COLOR(COLOR_NO)='0'B;
      END;
      FREE OLD_COLOR_DATA_PTR->COLOR_DATA;
    END;

/* UPDATE MAX ADJACENT COLOR.                                */

   ADJ_PTR->MAX_ADJ_COLOR=
     MAX(HI_COLOR,ADJ_PTR->MAX_ADJ_COLOR);

/* UPDATE ADJACENT COLORS ARRAY, NUMBER OF GAPS, AND         */
/* NUMBER OF USED COLORS.                                    */

   UC=ADJ_PTR->USED_COLORS;
   NG=ADJ_PTR->NO_OF_GAPS;
   IF LO_COLOR>1
     THEN PREV_COLOR_BIT=ADJ_COLOR(LO_COLOR-1);
     ELSE PREV_COLOR_BIT='1'B;
   IF ¬PREV_COLOR_BIT
     THEN NG=NG+1;
   DO COLOR_NO=LO_COLOR TO HI_COLOR;
     ADJ_COLOR_BIT=ADJ_COLOR(COLOR_NO);
     ADJ_COLOR(COLOR_NO)='1'B;
     IF ¬PREV_COLOR_BIT & ADJ_COLOR_BIT
       THEN NG=NG-1;   /* END OF A GAP. */
     IF ¬ADJ_COLOR_BIT
       THEN UC=UC+1;   /* ANOTHER COLOR USED. */
     PREV_COLOR_BIT=ADJ_COLOR_BIT;
   END;
   IF ¬PREV_COLOR_BIT
     THEN DO;
       IF HI_COLOR<ADJ_PTR->MAX_ADJ_COLOR
         THEN ADJ_COLOR_BIT=ADJ_COLOR(HI_COLOR+1);
         ELSE ADJ_COLOR_BIT=(MAX_COLOR=HI_COLOR);
       IF ADJ_COLOR_BIT
         THEN NG=NG-1;
     END;

/* UPDATE ALL WORK VARIABLES.                                */

   ADJ_PTR->USED_COLORS=UC;
   ADJ_PTR->NO_OF_GAPS=NG;
   ADJ_PTR->REDUCED_DEGREE=ADJ_PTR->REDUCED_DEGREE-1;
   ADJ_PTR->REDUCED_CHROMATIC_DEGREE=
     ADJ_PTR->REDUCED_CHROMATIC_DEGREE-CHROMATICITY;
```

```
      ADJ_PTR->PH=UC+ADJ_PTR->REDUCED_CHROMATIC_DEGREE
        +(NG+ADJ_PTR->REDUCED_DEGREE)
        *(ADJ_PTR->CHROMATICITY-1);
      OLD_FPH=ADJ_PTR->FLOAT_PH;
      CALL CALCFPH(ADJ_PTR);

/* ADJUST ADJACENT NODE'S LOCATION IN THE HEAP.                  */

      IF ADJ_PTR->FLOAT_PH < OLD_FPH
        THEN CALL DNHEAP(ADJ_PTR);
        ELSE CALL UPHEAP(ADJ_PTR);

NEXT_ADJ_NODE:
    END;

/* COLOR THE NODE OF HIGHEST PIGEONHOLE MEASURE.                 */

    NODE_PTR=LOCATOR(1);
    AUX_TAIL_PTR->AUX_FORWARD_PTR=NODE_PTR;
    AUX_TAIL_PTR=NODE_PTR;

/* DETERMINE LOWEST SEQUENCE OF AVAILABLE COLORS.                */

    COLOR_DATA_PTR=ADJ_COLOR_PTR;
    UNAVAILABLE_COLOR=0;
    COLOR_COUNT=0;
    DO COLOR_NO=1 TO MAX_ADJ_COLOR
                UNTIL(COLOR_COUNT=CHROMATICITY);
      IF ADJ_COLOR(COLOR_NO)
        THEN DO;
          COLOR_COUNT=0;
          UNAVAILABLE_COLOR=COLOR_NO;
        END;
        ELSE COLOR_COUNT=COLOR_COUNT+1;
    END;

/* ASSIGN THE COLORS.                                            */

    LO_COLOR=UNAVAILABLE_COLOR+1;
    HI_COLOR=UNAVAILABLE_COLOR+CHROMATICITY;
  END;

/* COMPLETE COLORED NODE LIST.                                   */

  AUX_FORWARD_PTR=NULL;
  MAX_COLOR=MAX(MAX_COLOR,HI_COLOR);

/* FREE ADJACENT COLORS STRUCTURE FOR EACH NODE.                 */
  NODE_PTR=HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);
    COLOR_DATA_PTR=ADJ_COLOR_PTR;
    FREE COLOR_DATA;
    NODE_PTR=FORWARD_PTR;
  END;
```

```
  RETURN;

CALCFPH: PROC(NODE_PTR);
  DCL
    INFINITY                    FLOAT BINARY(21) STATIC
                                 INIT(1.0E10),
    NODE_PTR                    POINTER,
    FLOAT                       BUILTIN,
    DENOMINATOR                 FIXED BINARY(15);
  DENOMINATOR=NODE_PTR->NO_OF_GAPS+
    NODE_PTR->REDUCED_DEGREE;
  IF DENOMINATOR ¬= 0
    THEN NODE_PTR->FLOAT_PH=
      FLOAT((NODE_PTR->PH-MAX_COLOR),21)/
      FLOAT(DENOMINATOR,21);
    ELSE NODE_PTR->FLOAT_PH=INFINITY;
END CALCFPH;

UPHEAP: PROC(NODE_PTR);
  DCL
    NODE_FPH                    FLOAT BINARY(21),
    NODE_PTR                    POINTER,
    NODE_SUBSCRIPT              FIXED BINARY(15),
    PARENT_PTR                  POINTER,
    PARENT_SUBSCRIPT            FIXED BINARY(15);
  NODE_SUBSCRIPT=NODE_PTR->HEAP_POSITION;
  NODE_FPH=NODE_PTR->FLOAT_PH;
  DO WHILE(NODE_SUBSCRIPT>1);
    PARENT_SUBSCRIPT=NODE_SUBSCRIPT/2;
    PARENT_PTR=LOCATOR(PARENT_SUBSCRIPT);
    IF NODE_FPH<=PARENT_PTR->FLOAT_PH
      THEN LEAVE;
    LOCATOR(NODE_SUBSCRIPT)=PARENT_PTR;
    PARENT_PTR->HEAP_POSITION=NODE_SUBSCRIPT;
    NODE_SUBSCRIPT=PARENT_SUBSCRIPT;
  END;
  LOCATOR(NODE_SUBSCRIPT)=NODE_PTR;
  NODE_PTR->HEAP_POSITION=NODE_SUBSCRIPT;
  RETURN;
END UPHEAP;

DNHEAP: PROC(NODE_PTR);
  DCL
    CHILD_FPH                   FLOAT BINARY(21),
    CHILD_PTR                   POINTER,
    CHILD_SUBSCRIPT             FIXED BINARY(15),
    NODE_FPH                    FLOAT BINARY(21),
    NODE_PTR                    POINTER,
    NODE_SUBSCRIPT              FIXED BINARY(15),
    RIGHT_FPH                   FLOAT BINARY(21),
    RIGHT_PTR                   POINTER,
    RIGHT_SUBSCRIPT             FIXED BINARY(15);
  NODE_SUBSCRIPT=NODE_PTR->HEAP_POSITION;
  NODE_FPH=NODE_PTR->FLOAT_PH;
```

```
  CHILD_SUBSCRIPT=2*NODE_SUBSCRIPT;
  DO WHILE(CHILD_SUBSCRIPT<=REMAINING_NODES);
    CHILD_PTR=LOCATOR(CHILD_SUBSCRIPT);
    CHILD_FPH=CHILD_PTR->FLOAT_PH;
    RIGHT_SUBSCRIPT=CHILD_SUBSCRIPT+1;
    IF RIGHT_SUBSCRIPT<=REMAINING_NODES
      THEN DO;
        RIGHT_PTR=LOCATOR(RIGHT_SUBSCRIPT);
        RIGHT_FPH=RIGHT_PTR->FLOAT_PH;
        IF RIGHT_FPH>CHILD_FPH
          THEN DO;
            CHILD_SUBSCRIPT=RIGHT_SUBSCRIPT;
            CHILD_PTR=RIGHT_PTR;
            CHILD_FPH=RIGHT_FPH;
          END;
        END;
      IF NODE_FPH>=CHILD_FPH
        THEN LEAVE;
      LOCATOR(NODE_SUBSCRIPT)=CHILD_PTR;
      CHILD_PTR->HEAP_POSITION=NODE_SUBSCRIPT;
      NODE_SUBSCRIPT=CHILD_SUBSCRIPT;
      CHILD_SUBSCRIPT=2*NODE_SUBSCRIPT;
    END;
    LOCATOR(NODE_SUBSCRIPT)=NODE_PTR;
    NODE_PTR->HEAP_POSITION=NODE_SUBSCRIPT;
    RETURN;
  END;
END DYNFPH;
```

```
/* RELEASE STORAGE FOR THE GRAPH.                                            */

FREGRAF: PROC(HEAD_PTR) REORDER;
  DCL
    HEAD_PTR                    PTR,
    NODE_PTR                    PTR,
    NULL                        BUILTIN;
  DCL
    1 NODE                      BASED(NODE_PTR),
      2 NUMBER                  FIXED BINARY(15),
      2 DEGREE                  FIXED BINARY(15),
      2 CHROMATICITY            FIXED BINARY(15),
      2 CHROMATIC_DEGREE        FIXED BINARY(15),
      2 LO_COLOR                FIXED BINARY(15),
      2 HI_COLOR                FIXED BINARY(15),
      2 WORK_VARIABLE_SPACE,       /* 40 BYTES */
        3 FILLER                CHAR(40),
      2 ADJ_NODE_PTR(0:DEG REFER(DEGREE)) PTR;
        /* ADJ_NODE_PTR(0) IS THE FORWARD LINK POINTER. */

  NODE_PTR=HEAD_PTR;
  DO WHILE(NODE_PTR¬=NULL);
    HEAD_PTR=ADJ_NODE_PTR(0);
    FREE NODE;
    NODE_PTR=HEAD_PTR;
  END;
END FREGRAF;
```