USING FRACTIONAL COLORING
TO SOLVE
SPARSE ITERATIVE SYSTEMS IN PARALLEL

by

Melanie A. Marchant

B.S., Dickinson College, 1983

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Master of Science

Department of Mathematics

1991
This thesis for the Master of Science
degree by
Melanie A. Marchant
has been approved for the
Department of Mathematics
by

David C. Fisher

Jennifer Ryan

J. Richard Lundgren

April 30, 1991
Date
Marchant, Melanie A. (M.S., Applied Mathematics)

Using Fractional Coloring to Solve Sparse Iterative Systems in Parallel

Thesis directed by Assistant Professor David C. Fisher

Coloring algorithms have been widely used to find parallel implementations of iterative methods, such as Successive Over-Relaxation. We study how fractional coloring can be used to solve sparse iterative systems in parallel, as well as speed the convergence of the iterative method. We also study the fractional coloring number of a graph $\chi_f(G)$. Using properties of linear programming, we prove that for any graph $G$, $\omega(G) \leq \omega_f(G) = \chi_f(G) \leq \chi(G)$, where $\omega(G)$ is the clique number of $G$, $\omega_f(G)$ is the fractional clique number of $G$, and $\chi(G)$ is the chromatic number of $G$. We also prove that $\chi_f(G)$ is on the order of $\chi(G)$ for random graphs, and that the fractional coloring can be computed to a desired accuracy. Finally, experimental evidence is given that $\chi_f(G)$ can be computed more quickly than $\chi(G)$.

The form and content of this abstract are approved. I recommend its publication.

Signed

David C. Fisher
To my husband Eric

and

my son Aaron

for

their unconditional love
CONTENTS

Figures ................................................. vi
Acknowledgements ................................. vii

1 USING FRACTIONAL COLORING TO SOLVE SPARSE ITERATIVE SYSTEMS IN PARALLEL 1

1.1 Parallelism of Symmetric Systems 3

1.2 Comparing Coloring and Clique Numbers 7
  1.2.1 A Sequence of Inequalities .............. 8
  1.2.2 Examples ................................ 9
  1.2.3 Numerical Bounds ......................... 12

1.3 Calculating the Fractional Coloring Number 16
  1.3.1 Column Generation ......................... 16
  1.3.2 Fractional Coloring Algorithm .......... 17
  1.3.3 Maximal Independent Set Algorithm .... 20
  1.3.4 Fractional Coloring Program ............. 24
    1.3.4.1 Cycling ................................ 24
    1.3.4.2 Refinement for Rounding Error ....... 24
    1.3.4.3 Adjustment for Large Graphs ......... 26
    1.3.4.4 Runtime Results ....................... 27

1.4 Efficient Parallelism 28
  1.4.1 Updating Schemes ......................... 29
  1.4.2 Iterative Methods ......................... 30

1.5 Parallelism of Non-Symmetric Systems 32

1.6 Summary 34

Appendix 36

References 56
FIGURES

1 A 5-cycle .............................................................. 4
2 A 6-cycle .............................................................. 9
3 A 7-cycle .............................................................. 10
4 The Mycielskian of a triangle ................................. 11
5 Grotzsch’s graph .................................................... 12
6 Limits for the fractional coloring number of almost all graphs 16
7 Maximal independent set algorithm graph (i) .............. 22
8 Maximal independent set algorithm graph (ii) ............. 23
9 Maximal independent set algorithm graph (iii) ............. 23
10 Fractional solution vs. running time ......................... 28
11 Dependency graph .................................................. 32
ACKNOWLEDGEMENTS

I would like to thank David C. Fisher and Jennifer Ryan for giving me this opportunity and sharing their expertise. Above all, I would like to thank Dr. Fisher for his support, guidance, and immeasurable patience.
1 USING FRACTIONAL COLORING TO SOLVE SPARSE ITERATIVE SYSTEMS IN PARALLEL

Iterative algorithms are those of the form $x(t + 1) = f(x(t))$, $t = 0, 1, \ldots$, where $f : \mathbb{R}^n \to \mathbb{R}^n$. They are used to solve numerical problems such as systems of equations and optimization. Each algorithm commences with an initial approximation $x(0)$, and the iterative scheme generates a sequence of vectors, $\{x(0), x(1), x(2), \ldots\}$, which may converge to a fixed point of $f$. Naturally, it is desirable to seek an iterative scheme attaining the fastest possible convergence of this sequence.

Following the notation used in Bertsekas and Tsitsiklis [1, section 1.2.4], let $x_i(t)$ denote the $i^{th}$ component of $x(t)$ and let $f_i$ denote the $i^{th}$ component of the function $f$. When we write $x_i(t + 1) = f_i(x_1(t), \ldots, x_n(t))$, we are indicating that updating the $i^{th}$ component of $x$ may require the values of $x_1(t), \ldots, x_n(t)$. An iteration is defined as a sequence of component updates such that each component is updated at least once. If components $x_1, \ldots, x_n$ are updated exactly once in an iteration, then we say there are $n$ steps per iteration. In an iteration, it may also be that several components will not require the values of all the components. Furthermore, those components not requiring similar component values could be updated simultaneously using parallel computers. In this case, it is possible to
update components simultaneously in \( m \) parallel steps, where \( m < n \).

An example of parallelism is a parallel implementation of the iterative method Successive Over-Relaxation, i.e., SOR, to solve the system \( Ax = b \), where \( A \) is sparse and symmetric. Coloring algorithms have been widely used to find parallel implementations of the SOR method (Bertsekas and Tsitsiklis [1]). Initially, the system is interpreted as a graph, \( G \), where the variables of the system are represented by the nodes in the graph. Thereafter, a coloring of \( G \) is found and the nodes of the same color are updated in parallel using SOR.

A coloring of \( G \) can be thought of as a collection of independent sets such that each node is in at least one set. The coloring number of \( G \), \( \chi(G) \), is the fewest number of independent sets that color \( G \). A fractional coloring of \( G \) is a collection of weighted independent sets such that each node of \( G \) is in sets whose weights sum to at least one. The fractional coloring number of \( G \), \( \chi_f(G) \), is the minimal sum of the weights in a fractional coloring of \( G \). We will see that the weighted independent sets suggest an updating scheme for parallel computation on the nodes of \( G \), and that this scheme could speed the rate of convergence of the iterative method by a factor of \( \chi(G)/\chi_f(G) \). Furthermore, a minimal fractional coloring of \( G \) can usually be found much more quickly than a minimal coloring of \( G \).
1.1 Parallelism of Symmetric Systems

It is possible to accelerate the convergence of iterative methods by using parallel computers to update different components simultaneously. Graph colorings can be used to specify which components of x may be updated in parallel and to suggest a scheduling of the $x_i$'s for updating. To do so, first construct a graph $G$ from the given system $Ax = b$, where $A$ is symmetric and positive definite. Let $V = \{1, \ldots, n\}$ be the nodes of $G$, and $E = \{(i, j) \mid x_i \text{ depends on } x_j\}$. In other words, for every column in $A$, there is a node in $G$, and an edge is placed between nodes $i$ and $j$ in $G$ if $a_{ij} \neq 0$ in $A$. For example, if a given system $Ax = b$ is translated as an iteration,

\[
\begin{align*}
    x_1(t + 1) &= f_1(x_1(t), x_2(t), x_5(t)) \\
    x_2(t + 1) &= f_2(x_1(t), x_2(t), x_3(t)) \\
    x_3(t + 1) &= f_3(x_2(t), x_3(t), x_4(t)) \\
    x_4(t + 1) &= f_4(x_3(t), x_4(t), x_5(t)) \\
    x_5(t + 1) &= f_5(x_1(t), x_4(t), x_5(t)),
\end{align*}
\]

then the adjacency matrix for $G$ is

\[
\text{adj}(G) = \begin{bmatrix}
    1 & 1 & 0 & 0 & 1 \\
    1 & 1 & 1 & 0 & 0 \\
    0 & 1 & 1 & 1 & 0 \\
    0 & 0 & 1 & 1 & 1 \\
    1 & 0 & 0 & 1 & 1
\end{bmatrix}
\]

and $G$ is a 5-cycle or $C_5$ (see figure 1).
Figure 1: A 5-cycle

Next, partition the nodes of $G$ into independent sets, or color classes. An independent set is a subset of nodes such that there are no edges between any two nodes. Thus, if nodes $i$ and $j$ are in an independent set, then $x_j$ is not needed to compute $x_i$ and $x_i$ is not needed to compute $x_j$, so $x_i$ and $x_j$ may be updated in parallel. Furthermore, the number of parallel computers, or processors, required per iteration is bounded by the cardinality of the largest independent set, or the independence number, $\alpha(G)$. If we partition the nodes of $G$ into as few independent sets as possible, we could maximize the number of nodes per independent set and minimize the number of parallel steps per iteration. We would also optimize the use of multiple processors, leaving less processors idle per iteration. Thus to maximize parallelism, we need to find a minimum coloring of $G$, or its chromatic number, $\chi(G)$. For example, the coloring number of $C_5$ is 3. In other words, at least three colors are required to color the nodes of $G$. We could color the independent sets $\{1, 3\}$ in red, $\{2, 4\}$ in blue, and $\{5\}$ in white. Two processors are required to perform an iteration as follows: update nodes 1 and 3 in parallel, update nodes 2 and 4 in parallel, update node 5. Thus in $\chi(G) = 3$ parallel steps,
all nodes are updated once; in six parallel steps all nodes are updated twice; etc.

Finding the minimum coloring of $G$ is an NP-hard problem, so there is no known polynomial algorithm to find $\chi(G)$. One can reformulate the coloring problem as an integer programming problem, though, to determine $\chi(G)$ and specify which nodes may be updated simultaneously. Let $K$ be the node-independent set incidence matrix, where the rows of $K$ represent the nodes of $G$, and the columns represent all independent sets of $G$. If node $i$ is in independent set $j$, then $k_{ij} = 1$. Otherwise, $k_{ij} = 0$. The following matrix is the node independent set incidence matrix $K$ of $G = C_5$.

$$K = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{bmatrix}$$

To find $\chi(G)$, we could solve the following integer programming problem (IP),

$$\chi(G) = \min 1^T x$$

subject to $Kx \geq 1$

$x \geq 0$, $x$ is integer.

It may be difficult to find a minimal coloring number this way for large graphs, since the number of columns in the (IP) could be exponential in the size of the graph. To avoid this problem, we solve a linear programming relaxation of the (IP), and use a linear programming technique called column generation. The solution of the linear programming problem (LP) is called the fractional coloring.
number, \( \chi_f(G) \) (Larsen, Propp, and Ullman [6]).

\[
\chi_f(G) = \min 1^T x \\
\text{subject to } Kx \geq 1 \\
x \geq 0.
\]

In the (LP), it is beneficial to think of \( x_i \) as a fractional weight of an independent set. Thus, \( \chi_f(G) = \sum x_i \), the sum of the weights on the independent sets. The first set of constraints, \( Kx \geq 1 \), are satisfied if each node of \( G \) is in independent sets whose weights sum to at least one. Also note that \( x \in \mathbb{R}^+ \) allows independent sets to have fractional weights. Thus, \( \chi_f(G) \) may be non-integer. In fact, since \( \chi_f(G) \) is the solution to an (LP), we know that it will always be a rational number. In Larsen, Propp, and Ullman [6], they also show that every rational number greater than two is \( \chi_f(G) \) for some graph \( G \).

The fractional coloring number of a \( C_5 \) is 2.5, corresponding to the independent sets, \{1,3\}, \{2,4\}, \{3,5\}, \{1,4\}, and \{2,5\}, each with weight 1/2. Two processors are required to perform an iteration as follows: update nodes 1 and 3 in parallel; update nodes 2 and 4 in parallel; update nodes 3 and 5 in parallel; update nodes 1 and 4 in parallel; updates node 2 and 5 in parallel. Note that each node is updated twice in five parallel steps, as opposed to six parallel steps as determined by \( \chi(G) \). This would suggest a theoretical speedup of 3.0/2.5 = 1.2, or \( \chi(G)/\chi_f(G) \).
1.2 Comparing Coloring and Clique Numbers

Before examining the method of column generation and its role in determining which nodes to update in parallel, it is beneficial to understand how a graph’s clique number, $\omega(G)$, and fractional clique number, $\omega_f(G)$, relate to its coloring numbers. The clique number of a graph, $\omega(G)$, is the number of nodes in the largest complete subgraph of $G$. Integer programming can be used to solve the clique problem. To find $\omega(G)$, we solve the following (IP),

$$\omega(G) = \max 1^T y$$
subject to $y^T K \leq 1^T$
$$y \geq 0, \ y \text{ is integer.}$$

We can use linear programming to find $\omega_f(G)$. To find $\omega_f(G)$ we solve the following (LP),

$$\omega_f(G) = \max 1^T y$$
subject to $y^T K \leq 1^T$
$$y \geq 0.$$ 

It is beneficial to think of $y_i$ as the fractional weight of a node. As with the coloring number problems, the rows of $K$ are indexed by the nodes in the graph, and its columns represent the independent sets of the graph. The first set of constraints, $y^T K \leq 1^T$, is satisfied if the sum of the weights on the nodes in any independent set is one or less. Also note that $y \in \mathcal{R}^+$ allows nodes of $G$ to have non-integer weights. Since $\omega_f(G) = \sum y_i$, the fractional clique number may also be non-integer. In fact, it is always rational since it is the solution of an (LP).

The following inequality first appeared in Hell [5], and subsequently in Larsen, Propp, and Ullman [6]: $\omega(G) \leq \omega_f(G) = \chi_f(G) \leq \chi(G)$. We will prove the
inequality in a new context and establish numerical bounds on the clique and coloring numbers.

1.2.1 A Sequence of Inequalities

First, we will show that $\omega(G) \leq \omega_f(G) = \chi_f(G) \leq \chi(G)$. It is well known that $\omega(G) \leq \chi(G)$ for any graph. We will need to show that $\omega(G) \leq \omega_f(G)$, $\omega_f(G) = \chi_f(G)$, and $\chi_f(G) \leq \chi(G)$.

**Lemma 1** $\omega(G) \leq \omega_f(G)$.

*Proof:* The optimal solution to the (IP) solving the clique problem is also a feasible solution to the (LP) solving the fractional clique problem. Since the maximal solution to the (LP) is greater than or equal to any of its feasible solutions, $\omega(G) \leq \omega_f(G)$. □

**Lemma 2** $\omega_f(G) = \chi_f(G)$.

*Proof:* To show that $\omega_f(G) = \chi_f(G)$, we need to observe the (LP)'s which calculate the fractional coloring and clique numbers. The (LP)'s are dual problems and we will refer to the (LP) which computes $\chi_f(G)$ as the *primal* and the (LP) which computes $\omega_f(G)$ as the *dual*. The Strong Duality Theorem of linear programming guarantees that the primal and the dual will have equal optimal objective function values if the problems have optimal solutions (see e.g., Winston [10]). Thus, $\omega_f(G) = \chi_f(G)$. □
**Lemma 3** $\chi_f(G) \leq \chi(G)$.

*Proof:* The optimal solution to the (IP) solving the coloring problem is also a feasible solution to the (LP) solving the fractional coloring problem. Since the minimal solution to the (LP) is less than or equal to any of its feasible solutions, $\chi_f(G) \leq \chi(G)$. □

**Theorem 1** [4] $\omega(G) \leq \omega_f(G) = \chi_f(G) \leq \chi(G)$.

*Proof:* By the results in the previous lemmas and the fact that $\omega(G) \leq \chi(G)$, we have proved the inequality. □

### 1.2.2 Examples

As an illustration of Theorem 1, we look at $n$-cycles. If $n$ is even, such as a $C_6$ in figure 2, then

$$\omega(G) = 2 \leq \omega_f(G) = 2 = \chi_f(G) \leq \chi(G) = 2.$$

![Figure 2: A 6-cycle](image-url)
If $n$ is odd, such as a $C_7$ in figure 3, then

$$\omega(G) = 2 \leq \omega_f(G) = 2n/(n-1) = \chi_f(G) \leq \chi(G) = 3.$$ 

$$\omega(G) = 2 \leq \omega_f(G) = 7/3 = \chi_f(G) \leq \chi(G) = 3$$

**Figure 3: A 7-cycle**

As another illustration, we look at Mycielskian graphs. Mycielskian graphs are a sequence of graphs, $G_n$, for which $G_{n+1}$ can be constructed from $G_n$, and some interesting properties apply. Following the notation in Larsen, Propp, and Ullman [6], the Mycielskian $\mu(G)$ of a graph $G = (V, E)$ is defined as follows. Let $V = \{v_1, \ldots, v_m\}$. Define $\mu(G)$ by $V(\mu(G)) = \{x_1, x_2, \ldots, x_m, y_1, \ldots, y_m, z\}$ with $E(\mu(G)) = $

$$\{(x_i, x_j) \iff (v_i, v_j) \in E(G), \}

$$((x_i, y_j) \iff (v_i, v_j) \in E(G),$$

$$(y_i, z)(\forall i = 1, \ldots, m)\}.$$

Larsen, Propp, and Ullman also proved the following theorem.
Theorem 2 Let $\mu(G)$ be the Mycielskian of a graph $G$ with at least one edge.

Then

1. $\omega(\mu(G)) = \omega(G);$ 

2. $\chi(\mu(G)) = \chi(G) + 1;$ 

3. $\chi_f(\mu(G)) = \chi_f(G) + \frac{1}{\chi_f(G)}.$

An illustration of Theorems 1 and 2 is the Mycielskian of a $K_3$, (i.e., triangle)
in figure 4.

\[
\omega(\mu(G)) = 3 ; \quad \chi_f(\mu(G)) = 3 + \frac{1}{3} = \frac{10}{3} ; \quad \chi(\mu(G)) = 3 + 1 = 4
\]

Figure 4: The Mycielskian of a triangle

As another illustration, let $G_2 = K_2$, and define $G_{n+1} = \mu(G_n)$ for $n \geq 2,$
then $G_3$ is a $C_5$ and $G_4$ is Grotzsch’s graph in figure 5. Grotzsch’s graph is the
smallest graph where the clique and chromatic numbers differ by more than one.
\[
\omega(\mu(G)) = 2 \quad \text{and} \quad \chi_f(\mu(G)) = \frac{5}{2} + \frac{2}{5} = \frac{29}{10} \quad \text{and} \quad \chi(\mu(G)) = 3 + 1 = 4
\]

**Figure 5: Grotzsch’s graph**

For this class of Mycielskian graphs, Larsen, Propp, and Ullman also show that \(\chi(G_n) - \chi_f(G_n) \to \infty\) and \(\chi_f(G_n) - \omega(G_n) \to \infty\). In the next section, we will see that the clique and coloring numbers of random graphs behave quite differently.

1.2.3 Numerical Bounds

We have shown that \(\omega(G) \leq \chi_f(G) \leq \chi(G)\) for any graph, but how close to either \(\omega(G)\) or \(\chi(G)\) is \(\chi_f(G)\) for a random graph? We know that our iterative scheme will converge faster with use of fractional coloring. It is reasonable to assume that if \(\chi_f(G)\) is almost always closer in value to \(\omega(G)\) than \(\chi(G)\) for any graph, then the use of fractional coloring is indispensable in solving large, sparse systems in parallel. Unfortunately, we will see that \(\chi_f(G)\) is on the order of \(\chi(G)\) rather
than $\omega(G)$ for random graphs. Keep in mind, though, that $\chi_f(G) \leq \chi(G)$ and the use of fractional coloring is an improvement on the use of integer coloring in the parallel implementation of an iterative method.

First, we prove a theorem which will become useful in the proof of Theorem 6.

**Theorem 3** [4] Let $G$ be a graph with $n$ nodes, and let $\alpha(G)$ be the maximal cardinality of an independent set of $G$. Then $\chi_f(G) \geq \frac{n}{\alpha(G)}$.

**Proof:** From Lemma 2, we know that $\chi_f(G) = \omega_f(G)$, the optimal dual solution. A feasible solution to the (LP) solving the fractional clique problem is a weight assignment of $\frac{1}{\alpha(G)}$ on each node. If there are $n$ nodes, then $\omega_f(G) \geq \frac{n}{\alpha(G)}$. Thus, $\chi_f(G) \geq \frac{n}{\alpha(G)}$. □

Next, we state two theorems on which we base the proof of Theorem 6. Grimmett and McDiarmid (quoted in Palmer [7]) found bounds on the chromatic number as follows in Theorem 4, and Matula (quoted in Palmer [7]) found the clique number of almost every graph as follows in Theorem 5.

**Theorem 4** Let $0 < \epsilon < 1$ be given, and set $b = \frac{1}{1-p}$. With $p$, the probability of an edge, fixed, the chromatic number of almost all graphs $G$ satisfies

$$\frac{(\frac{1}{2} - \epsilon)n}{\log_b(n)} \leq \chi(G) \leq \frac{(1 + \epsilon)n}{\log_b(n)}.$$
Theorem 5 Let $G$ be a graph with $n$ nodes where the probability of an edge is fixed at $p = \frac{1}{b}$. Let the function $d = d(n)$ be defined by

$$d(n) = 2 \log_b(n) - 2 \log_b \log_b(n) + 1 + 2 \log_b \log_b \left( \frac{e}{2} \right).$$

Then for any $\epsilon > 0$,

$$\text{Prob} \left( d(n) - \epsilon \leq \omega(G) \leq d(n) + \epsilon \right) \rightarrow 1.$$

We are now ready to state and prove a theorem revealing bounds on the fractional coloring number.

Theorem 6 [4] Let $n$ be the number of nodes in a graph $G$, and $b = \frac{1}{1-p}$, where $p$ is the fixed probability of an edge. Then, for all $\epsilon > 0$, we have with probability 1 that

$$\frac{(\frac{1}{2} - \epsilon)n}{\log_b(n)} \leq \chi_f(G) \leq \frac{(1 + \epsilon)n}{\log_b(n)}.$$

Proof: Let $0 < \delta < \frac{1}{2}$. Then we know from Theorem 4,

$$\text{Prob} \left( \frac{(\frac{1}{2} - \delta)n}{\log_b(n)} \leq \chi(G) \leq \frac{(1 + \delta)n}{\log_b(n)} \right) \rightarrow 1.$$

Let $\bar{b} = \frac{1}{1-p}$. Then by Theorem 5,

$$\text{Prob} \left( |\omega(G) - d(n)| \leq \delta \right) \rightarrow 1$$

where $d(n) = 2 \log_b(n) - 2 \log_b \log_b(n) + 1 + 2 \log_b \log_b \left( \frac{e}{2} \right)$. Thus,

$$\text{Prob} \left( \omega(G) \leq d(n) + \delta \right) \rightarrow 1.$$
Since $\omega(G) = \alpha(G)$,

$$\text{Prob} \left( \alpha(G) \leq \bar{d}(n) + \delta \right) \to 1,$$

and

$$\text{Prob} \left( \frac{n}{\alpha(G)} \geq \frac{n}{\bar{d}(n) + \delta} \right) \to 1.$$

Since $\chi_f(G) \geq \frac{n}{\omega(G)}$,

$$\text{Prob} \left( \chi_f(G) \geq \frac{n}{\bar{d}(n) + \delta} \right) \to 1.$$

Since $\bar{d}(n) \leq 2 \log_b(n)$ for large $n$,

$$\text{Prob} \left( \chi_f(G) \geq \frac{n}{2 \log_b(n) + \delta} \right) \to 1.$$

Letting $\delta = \frac{2 \log_b(n)}{\frac{1}{2} - \epsilon}$, we write

$$\text{Prob} \left( \chi_f(G) \geq \frac{n}{2 \log_b(n) + \frac{2 \log_b(n)}{\frac{1}{2} - \epsilon}} \right) \to 1.$$

After algebraic simplification, we achieve

$$\text{Prob} \left( \chi_f(G) \geq \frac{\left( \frac{1}{2} - \epsilon \right)n}{\log_b(n)} \right) \to 1.$$

We proceed with notation used in Theorem 4. Recognizing that $b$ in Theorem 5 is the same as $b$ in Theorem 4, we may write

$$\text{Prob} \left( \chi_f(G) \geq \frac{\left( \frac{1}{2} - \epsilon \right)n}{\log_b(n)} \right) \to 1.$$
Since $\chi_f(G) \leq \chi(G)$,

$$\text{Prob}\left( \frac{(1 - \epsilon)n}{\log_6(n)} \leq \chi_f(G) \leq \frac{(1 + \epsilon)n}{\log_6(n)} \right) \rightarrow 1.$$ 

\[\square\]

Figure 6 gives an illustration of the numerical bounds as described above.

\[\begin{array}{cccc}
\omega & \chi_f(G) \leq \chi(G) & \cdots & n \\
0 & 2\log_6(n) & \frac{n}{2\log_6(n)} & \frac{n}{\log_6(n)}
\end{array}\]

Figure 6: Limits for the fractional coloring number of almost all graphs

1.3 Calculating the Fractional Coloring Number

To directly solve the (LP) for $\chi_f(G)$, we would first have to generate all independent sets of $G$. Since the number of independent sets in $G$ is exponential in the number of nodes in $G$, finding the independent sets and the solution of the resulting (LP) would be prohibitively expensive on a graph of even a modest number of nodes. Instead, we examine the method and benefits of column generation.

1.3.1 Column Generation

The method of column generation allows us to start with a reasonable number of independent sets and to add subsequent sets as needed to solve a (LP). Employing
this technique to solve the (LP),

\[
\begin{align*}
\min & \quad 1^T x \\
\text{subject to} & \quad Kx \geq 1 \\
& \quad x \geq 0,
\end{align*}
\]

we start with enough columns in the \( K \) matrix to provide an initial basic feasible solution, such that each node in the graph is represented in at least one independent set. We then run the Simplex algorithm. The algorithm returns two sets of weights. The primal weights in \( x \) are the weights on the independent sets represented by the columns of \( K \). The dual weights in \( y \) are the weights on the nodes in the graph represented by the rows of \( K \). If one can determine an independent set whose dual weights sum to one or greater, then the dual is not feasible. Hence, a new column, or the incidence vector of the generated independent set, is added to \( K \) before running the Simplex algorithm again. This process repeats until one cannot determine any independent set whose dual weights sum to one or greater. Upon this situation, the dual is feasible. Furthermore, since \( \sum x_i = \sum y_i \), the dual and primal are optimal by the Strong Duality Theorem.

\subsection*{1.3.2 Fractional Coloring Algorithm}

We have an algorithm to calculate the fractional coloring number of a graph using the linear programming method of column generation. The details of the algorithm are described directly below in steps 1-5.
1. Create the columns of the $K$ matrix by adding the incidence vectors of enough independent sets of $G$ such that each node of $G$ is in at least one independent set. Nodes already represented in independent sets may be added to a subsequently generated independent set to "fill" or maximize the set's size.

2. Run the Simplex algorithm, which will return primal weights in the vector, $x$, and dual weights in the vector, $y$.

3. Check to see if the dual is feasible using the dual weights from Simplex.
   
   a.) If the dual is not feasible, an independent set may be formed whose dual weights sum to greater than one. Proceed to step 4.
   
   b.) If the dual is feasible, no independent set can be formed whose dual weights sum to greater than one. Proceed to step 5.

4. Add the incidence vector of the generated independent set as a column in the $K$ matrix. Go to step 2.

5. The dual is feasible. In fact, since $\sum x_i = \sum y_i$, the dual and primal are optimal by the Strong Duality Theorem. Thus, $\chi_f(G) = \sum x_i$.

As an illustration of the fractional coloring algorithm, we will follow the details in calculating $\chi_f(G)$ for $G = C_5$. Let $y$ and $x$ denote the dual and primal weight
vectors, respectively. We begin with enough independent sets to cover the nodes of $G$.

$$K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Simplex is run giving the following independent sets, primal weights, and dual weights.

$$x_{\{1,3\}} = x_{\{2,4\}} = x_{\{2,5\}} = 1$$

$$y = (1, 0, 0, 1, 1)$$

Is the dual feasible? No. We can add $\{1,4\}$ as a column of $K$.

$$K = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Simplex is run giving the following primal and dual weights.

$$x_{\{1,3\}} = x_{\{2,4\}} = x_{\{2,5\}} = 1; x_{\{1,4\}} = 0$$

$$y = (0, 0, 1, 1, 1)$$

Is the dual feasible? No. We can add $\{3,5\}$ as a column of $K$.

$$K = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$
Simplex is run giving the following primal and dual weights.

\[ x_{\{1,3\}} = x_{\{2,4\}} x_{\{2,5\}} = x_{\{1,4\}} = x_{\{3,5\}} = 0.5 \]

\[ y = (0.5, 0.5, 0.5, 0.5, 0.5) \]

Is the dual feasible? Yes, there are no independent sets whose dual weights sum to one or greater. Since the dual is feasible, and \( \sum x_i = \sum y_i \), the dual and primal are optimal by the Strong Duality Theorem. Thus, \( \chi_f(G) = 2.5 \).

1.3.3 Maximal Independent Set Algorithm

When generating columns in \( K \), one could simply select the first independent set whose dual weights sum to greater than one. To find \( \chi_f(G) \) most efficiently, though, a "maximal" independent set should be added to \( K \) when the dual is not feasible. Ideally, this independent set has a maximum dual weight sum and contains a maximum number of nodes. Although finding a maximal independent set may take more time than selecting the first eligible independent set, it is probable that less total time will be required to solve the (LP) (Chvatal [3]).

We have an algorithm which exhaustively searches for a maximal independent set whose dual weights sum to one or greater. This algorithm chooses nodes for the set according to their dual weights and degrees as follows below in steps 1-7. Note that it is appropriate to insert the steps of this algorithm in the fractional coloring algorithm whenever the coloring algorithm generates an independent set.
1. Initially, consider those nodes whose dual weights are of significance. Hence, we are considering a subgraph of $G$ in steps 1-6, where we determine an independent set with nodes of significant weight. We continue by maximizing the size of the independent set with nodes of insignificant weight in step 7.

2. Calculate a "weight ratio" of available nodes. Let $v$ represent a node in the graph, and $y(v)$ represent the dual weight of that node. Let $N(v)$ be the set of nodes adjacent to $v$ and available for selection.

$$\text{ratio}(v) = \frac{y(v)}{\sum_{a \in N(v)} y(a) + \epsilon}$$

Note that $\epsilon$ is added in the denominator to avoid possible division by zero if $v$ is an isolated node.

3. Calculate the degree of each available node.

4. Select node for the maximal independent set with the greatest weight ratio and least degree.

5. Disregard selected node's adjacent nodes in any subsequent steps.

6. Repeat steps 2-5 until no nodes are available for selection from the subgraph of significant nodes.

7. Fill the independent set with whatever nodes of insignificant weight can be added accordingly.
As an illustration of the maximal independent set algorithm, we are given the graph and node weights in figure 7. Let \( r \) and \( d \) denote the ratio and degree vectors, respectively. Thus, we begin to generate a maximal independent set, \( I \), with node 1 such that \( I = \{1\} \).

\[
\mathbf{r}(1, 2, 3, 4, 5, 6) = (0.5, 0.4, 0.5, 0.5, 0.5, 0.5) \\
\mathbf{d}(1, 2, 3, 4, 5, 6) = (1, 3, 2, 2, 2, 2)
\]

**Figure 7: Maximal independent set algorithm graph (i)**

Since node 1 was chosen for the independent set, its adjacent node, 2, is disregarded and weight ratios and degrees are recalculated on the following subgraph in figure 8.
\begin{align*}
\mathbf{6} & \quad 3 \\
5 & \quad 4
\end{align*}

\[ r(3, 4, 5, 6) = (1, 0.5, 0.5, 1) \]
\[ d(3, 4, 5, 6) = (1, 2, 2, 1) \]

**Figure 8: Maximal independent set algorithm graph (ii)**

We add node 3 to the independent set giving \( I = \{1, 3\} \). Node 4 is disregarded and weight ratios and degrees are recalculated on the following subgraph in figure 9.

\[ \begin{array}{c}
\bullet 6 \\
\downarrow \\
\bullet 5
\end{array} \]

\[ r(5, 6) = (1, 1) \]
\[ d(5, 6) = (1, 1) \]

**Figure 9: Maximal independent set algorithm graph (iii)**

We add node 5 to the independent set giving \( I = \{1, 3, 5\} \). Node 6 is disregarded. No subgraph remains and there are no additional nodes with insignificant weights to consider, so the independent set \( I = \{1, 3, 5\} \) with total weight of 5/3 may be added as a column in \( K \). Note that maximal independent sets were selected in the illustration of the fractional coloring algorithm.
1.3.4 Fractional Coloring Program

Jennifer Ryan and I wrote a program to implement the fractional coloring algorithm and maximal independent set algorithm. We also incorporated a version of the Simplex Algorithm written by The Rand Corporation. In doing this we discovered a number of problems, as well as some interesting results.

1.3.4.1 Cycling

It is very rare that the Simplex algorithm cycles. It appears to be common, though, in linear programming when using column generation. Cycling occurred in our program execution due to degenerate pivots. The problem was corrected after coding in Bland’s least index pivot rule.

1.3.4.2 Refinement for Rounding Error

Rounding errors also affected the execution of our code. Passing tolerances and making other checks did not completely remedy the situation. Finally, a change was made to the fractional coloring algorithm in the way it determined the feasibility of the dual problem. According to the dual (LP), the dual is feasible if \( y^T k \leq 1 \) for all independent sets’ incidence vectors, \( k \). Due to rounding errors, it was possible that the dual weights sum for a particular independent set was slightly greater than one. The independent set was added to the \( K \) matrix and Simplex readded the dual weights and found that the sum was either one or slightly less than one plus some tolerance. Simplex did not use the indepen-
dent set and returned the same dual weights from the previous execution. This resulted in looping and repeated objective function values. The problem was remedied after the feasibility of the dual was changed such that $y^T k \leq 1 + \epsilon$ for all $k$, where $\epsilon$ is arbitrary. It was also discovered that the value of $\epsilon$ allowed one to determine an accuracy of the fractional coloring number as suggested by the following theorem.

**Theorem 7** [4] Given any graph on $n$ nodes, find its fractional coloring number by solving an (LP) using column generation. Let $y$ and $x$ be its dual and primal weights, respectively, as returned by the Simplex algorithm. Let $I$ be any independent set in the graph and $N$ be the number of independent set incidence vectors (i.e., columns of $K$). For any $\epsilon > 0$, if $\sum_{i \in I} y_i \leq 1 + \epsilon$ for all $I$, then the corresponding feasible fractional coloring has

$$\chi_f(G) \leq \sum_{j=1}^{N} x_j \leq (1 + \epsilon)\chi_f(G).$$

**Proof:** Recall that a graph’s dual weights represent the weights on its nodes and a graph’s primal weights represent weights on its independent sets.

Let $y'_i = \frac{y_i}{1+\epsilon}$. Then $\sum_{i \in I} y'_i \leq 1$ since $\sum_{i \in I} y_i \leq 1 + \epsilon$. So $y'$ is a feasible dual solution. Thus, $\sum_{i=1}^{n} y'_i \leq \omega_f(G) = \chi_f(G)$. This implies that $\sum_{i=1}^{n} \frac{y_i}{1+\epsilon} \leq \chi_f(G)$, or $\sum_{i=1}^{n} y_i \leq (1 + \epsilon)\chi_f(G)$. Recall that $\chi_f(G) \leq \sum_{j=1}^{N} x_j$, a feasible solution to the primal. Also note that the Simplex algorithm always returns dual and primal
weights such that $\sum_{j=1}^{N} x_j = \sum_{i=1}^{n} y_i$. Thus,

$$\chi_f(G) \leq \sum_{j=1}^{N} x_j \leq (1 + \epsilon)\chi_f(G).$$

\[\square\]

1.3.4.3 Adjustment for Large Graphs

A final situation of interest occurred when the fractional coloring program was executed for a random graph on 100 nodes. After many executions of the Simplex code, the program took an unreasonable amount of time to find an independent set whose dual weights sum was greater than $1 + \epsilon$. It was found that a clique on four nodes contained the only nodes out of 100 with any significant weight. To avoid further situations like this when generating a maximal independent set, we changed our code to initially consider a subgraph of $G$ with those nodes whose dual weights were greater than $\frac{\epsilon}{2n}$. From this subset of nodes with significant weights, an independent set was formed, and nodes with insignificant weights were added subsequently to maximize the size of the set. We continued to check dual feasibility by determining if any independent set could be formed with a dual weight sum greater than $1 + \epsilon$. Thus, our code actually checks for dual feasibility by determining if an independent set can be generated with a dual weight sum greater than $1 + \frac{\epsilon}{2}$. This is done so that if all $n$ nodes of a graph have insignificant weights, then no independent set could be generated. Hence,

$$\sum y_i \leq 1 + \frac{\epsilon}{2} + (n)(\frac{\epsilon}{2n}) = 1 + \epsilon.$$
Not only did this strategy speed the convergence of the fractional coloring number for large graphs, but similar results were achieved for graphs of all sizes.

1.3.4.4 Runtime Results

The fractional coloring algorithm has been compared to an integer coloring algorithm (Syslo, Narsingh, and Kowalik [8]). It was found for a random graph on 40 nodes, with probability of an edge equal to 0.5, that it took 3.04 seconds to compute $\chi_f(G) = 7.64846$, as compared to the 993.01 seconds it took to compute $\chi(G) = 8$ (see figure 10). As expected, the fractional coloring number was less than the chromatic number for the graph. More importantly, though, the fractional algorithm converged in less time to $\chi_f(G)$ than the integer algorithm converged to $\chi(G)$. This is of great importance if an algorithm is given a limited amount of time to find the coloring number of a graph. Note that a copy of the fractional coloring program in FORTRAN is provided in the Appendix of this paper.
1.4 Efficient Parallelism

There are several reasons to use fractional coloring as opposed to integer coloring in solving large, sparse systems. Probably the most important one is the theoretical speed-up of the iterative scheme. In fact, we quantified this theoretical speed-up in the iterative scheme as $\chi(G)/\chi_f(G)$. Recall the $C_5$ example with a possible speed-up of $3.0/2.5 = 1.2$. In that discussion we had neglected to consider how to achieve this speed-up given the columns or independent sets in the final $K$ matrix. We will consider some possible criteria using this example, and also examine different iterative methods to determine the elements of efficient parallelism.
1.4.1 Updating Schemes

In the 5-cycle example in figure 1, the final run of Simplex gave the independent sets \{1,3\}, \{2,4\}, \{3,5\}, \{1,4\}, and \{2,5\} each with weight 1/2. One criterion in achieving a speed-up of the iterative scheme might be to update the nodes in each independent set \(x_i/\chi_f(G)\) of the time. In the \(G_5\) example, each independent set should be updated 0.5/2.5 of the time, or with equivalent consideration. Another criterion might be to find an ordering of the independent sets so that consecutive sets in the iterative scheme do not contain like nodes. An efficient ordering for the \(G_5\) example is simply \{1,3\}, \{2,4\}, \{3,5\}, \{1,4\}, \{2,5\}. As graphs increase in size and complexity, though, there may be additional criteria to consider. Furthermore, it is uncertain if optimal updating schemes could even exist for all graphs.

Consider the difficulty of an updating scheme with the following example.

When calculating \(\chi_f(G)\) for Grotzsch’s graph in figure 5, the final execution of Simplex returns the following independent sets and primal weights.

\[x_{\{6,7,8,9,10\}} = 0.4\]

\[x_{\{2,4,7,9\}} = x_{\{2,5,7,10\}} = x_{\{1,4,6,9\}} = x_{\{3,5,8,10\}} = 0.3\]

\[x_{\{1,3,11\}} = x_{\{2,4,11\}} = x_{\{2,5,11\}} = x_{\{1,4,11\}} = x_{\{3,5,11\}} = 0.2\]

To achieve the theoretical speed-up of \(\chi(G)/\chi_f(G)\), or 4/2.9, we might need to find an ordering of the independent sets to meet criteria such as that insuring no
consecutive sets contain like nodes and \( \{6,7,8,9,10\} \) is updated 4/29 of the time, 
\( \{1,3,11\} \), \( \{2,4,11\} \), \( \{2,5,11\} \), \( \{1,4,11\} \), \( \{3,5,11\} \) are each updated 3/29 of the time, 
and \( \{2,4,7,9\} \), \( \{2,5,7,10\} \), \( \{1,4,6,9\} \), \( \{3,5,8,10\} \), \( \{1,3,6,8\} \) are each updated 2/29 of the time. Clearly, this is no simple task.

1.4.2 Iterative Methods

Throughout this discussion, we referred to an iterative method as \( x(t+1) = f(x_1(t),...,x_n(t)) \). This iterative scheme is known as the Jacobi method. A possible improvement to the Jacobi method is computing \( x_i(t+1) \) using the 
values of \( x_1(t+1),...,x_{i-1}(t+1),x_{i+1}(t),...,x_n(t) \). In other words, we can 
use the most recent calculations available in our iterative scheme such that 
\[ x_i(t+1) = f_i(x_1(t+1),...,x_{i-1}(t+1),x_{i+1}(t),x_{n}(t)), i = 1,...,n. \] 
This iterative technique is known as the Gauss-Seidel method. It is possible to accelerate 
the computation in a Gauss-Seidel iteration using parallel computers. Recall 
that if entries in the system matrix \( A \), \( a_{ij} = a_{ji} = 0 \), then \( x_j(t) \) is not needed 
to compute \( x_i(t) \), and \( x_i(t) \) is not needed to compute \( x_j(t) \). Thus, \( x_i(t) \) and 
\( x_j(t) \) can be computed simultaneously, and component values calculated in any 
prior parallel step of the iteration may be used for component computation in 
the current parallel step. It is also possible that a Gauss-Seidel iteration is non-
parallelizable. For example, if each component, \( x_i \), depends on all other compo-
nents, \( x_1,x_2,...,x_{i-1},x_{i+1},...,x_n \), then only one component may be updated at
a time. In a sparse system, though, where \( a_{ij} = a_{ji} = 0 \) for numerous \( i \) and \( j \), it is always desirable to study an efficient parallel implementation of the iterative method.

If the system matrix, \( A \), is strictly diagonally dominant, then both the Jacobi and Gauss-Seidel methods converge to the solution for any arbitrary \( x(0) \) (Varga [9]). There is a way to accelerate the rate of convergence for systems which converge by the Gauss-Seidel method. These methods are called Successive Over-Relaxation, i.e., SOR. Furthermore, if \( A \) is positive definite and symmetric, then the SOR method is guaranteed to converge for any initial approximation, \( x(0) \) (Burden and Faires [2]).

It would appear that the fastest rate of convergence could occur with a parallel implementation of the SOR iterative method. It was found, though, that an efficient ordering scheme in the parallel implementation was necessary to achieve this desired speed. Prior to observing the effects of parallelism, systems of equations were solved using a random SOR method. Components were randomly updated as follows. After every execution of Simplex, the program returned dual weights in \( y \) and each node, \( i \), was assigned a “weight ratio,” \( w_i = y_i / \sum y_i \), such that \( 0 < w_i < 1 \). Thereafter, the SOR algorithm generated a random number, \( r \), such that \( 0 < r < 1 \). If \( w_{i-1} \leq r \leq w_i \), and \( y_i \neq 0 \), then node \( i \) was updated. Unfortunately, the convergence rate of the random SOR was on the order of a Jacobi iteration, which is much slower than SOR (Fisher [4]). Thus, a desirable
parallelism requires an efficient updating scheme as well as an effectual iterative method.

1.5 Parallelism of Non-Symmetric Systems

Thus far, we have only considered solving $Ax = b$ if $A$ was symmetric. Another interesting problem is posed when $A$ is not symmetric. Consider solving a large, sparse system $Ax = b$ where $A$ is non-symmetric and parallel computing may be used to speed the rate of convergence of the iterative method.

Following the notation used in Bertsekas and Tsitsiklis [1, section 1.2.4], define a corresponding dependency graph, $G$, as $G = (V, E)$ where $V = \{1, \ldots, n\}$ and $E = \{(i, j) | f_j \text{ depends on } x_i\}$. For example, if a given system, $Ax = b$, is translated as the following Jacobi iteration,

\[
\begin{align*}
    x_1(t + 1) &= f_1(x_1(t), x_5(t)) \\
    x_2(t + 1) &= f_2(x_1(t), x_2(t)) \\
    x_3(t + 1) &= f_3(x_2(t), x_3(t)) \\
    x_4(t + 1) &= f_4(x_3(t), x_4(t)) \\
    x_5(t + 1) &= f_5(x_4(t), x_5(t)),
\end{align*}
\]

then the corresponding dependency graph, $G$, is that in figure 11.

\[\text{Figure 11: Dependency graph}\]
The adjacency matrix for $G$ is

\[
\text{adj}(G) = \begin{bmatrix}
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
\end{bmatrix}.
\]

If we think of independent sets as those cycleless sets in the graph, then we can update nodes in a cycleless set in parallel using a different processor for each node in the set. Furthermore, the number of processors required per iteration is bounded by the size of the largest cycleless set. Thus, a symmetric system can be defined in a non-symmetric context by generating a directed graph from the undirected graph by replacing edges with double arcs between nodes. Thus, nodes connected with an edge could not possibly be placed in the same cycleless set.

As an example of non-symmetric parallelism, consider the dependency graph in figure 11, where $\chi(G) = 2$. We could choose the cycleless sets \{1,2,3,4\} and \{5\} since no cycle exists among nodes in each set. In this context, maximizing parallelism is equivalent to optimally coloring a dependency graph as suggested by the following proposition in Bertsekas and Tsitsiklis [1, Proposition 2.5].

**Proposition 1** The following statements are equivalent:

1. There exists an ordering of the variables such that an iteration of the corresponding Gauss-Seidel algorithm can be performed in $K$ parallel steps.
2. There exists a coloring of the dependency graph that uses $K$ colors and with the property that the induced subgraph on each color class has no cycles.

For the dependency graph in figure 11, four processors are required to perform an iteration as follows: update nodes 4, 3, 2, and 1 in the first parallel step, and update node 5. Note that each node is updated once in $\chi(G) = 2$ parallel steps, or $m$ times in $2m$ parallel steps.

If we solve the same problem using fractional coloring, $\chi_f(G) = 5/4$, corresponding to cycleless sets $\{1,2,3,4\}$, $\{2,3,4,5\}$, $\{3,4,5,1\}$, $\{4,5,1,2\}$, $\{5,1,2,3\}$, each with weight $1/4$. Four processors are required to perform an iteration as follows: update nodes 4, 3, 2, and 1 in the first parallel step, update nodes 5, 4, 3, and 2 in the second parallel step, update nodes 1, 5, 4, and 3 in the third parallel step, update nodes 2, 1, 5, and 4 in the fourth parallel step, and update nodes 3, 2, 1, and 5 in the fifth parallel. Note that each node is updated eight times in ten parallel steps as opposed to sixteen parallel steps as determined by $\chi(G)$. This would suggest a theoretical speedup of $2.0/1.25 = 1.6$. Note, once again, that regardless of the symmetric qualities of the system or the iterative method, an efficient updating scheme is required to obtain this theoretical speedup.

1.6 Summary

In this paper, we have studied fractional coloring and its related parameter, $\chi_f(G)$. We proved that $\omega(G) \leq \omega_f(G) = \chi_f(G) \leq \chi(G)$, using properties of linear
programming, and that $\chi_f(G)$ is on the order of $\chi(G)$ for random graphs. We also showed that the fractional coloring program can calculate a graph's fractional coloring number to a desired accuracy. We have also studied how fractional coloring can be used to solve sparse, symmetric iterative systems in parallel, and speed the convergence of the iterative method. Thereafter, we noted some of the complexities of efficient parallelism regardless of the iterative method. Finally, we showed that fractional coloring can also be used to solve non-symmetric systems. Open areas of study include efficient updating schemes, a modification of the fractional coloring number program to accommodate non-symmetric systems, and a program to execute parallel implementations of SOR in solving sparse iterative systems.
APPENDIX

C THE FRACTIONAL COLORING PROGRAM
C
C THIS PROGRAM CALCULATES THE FRACTIONAL COLORING NUMBER
C OF A GRAPH. THE USER CAN EITHER HAVE A GRAPH READ FROM
C FOR008.DAT OR ALLOW THE GENERATION OF A RANDOM GRAPH.
C TO GENERATE A RANDOM GRAPH, ONE MUST ENTER THE NUMBER
C OF NODES IN THE GRAPH, A SEED FOR THE RANDOM GENERATOR,
C THE PROBABILITY OF AN EDGE, AND AN EPSILON WHICH WILL
C DETERMINE THE ACCURACY OF THE CALCULATED FRACTIONAL
C COLORING NUMBER.
C
C VARIABLES FOR SIMPLEX
    REAL AM(200,600),BM(200),CM(600)
    INTEGER KM(7)
    REAL PS(600),DS(200),NEWCOL(200),NDS(200)
    REAL BINV(40000)
    INTEGER KB1(600),basis(200)
    REAL PE1(200),Xi(200)
    EQUIVALENCE (FEAS1,KM(6))
    LOGICAL SIMFLAG,VER1,NEG1,FEAS1
    INTEGER NVM1,NROWS,TEMPVARS,NCOLMNS
    REAL LPVAL,ADDDRED,EP5
    INTEGER ADDIND,SEED
    INTEGER SURP1,SURP2

C
C VARIABLES FOR COLOURING
    INTEGER EDGES(38000),NODEPOINT(200)
    INTEGER COUNTER
    LOGICAL FLAG,MAXSET

C
C READ IN DATA FROM UNIT 8
C
C    READ(8,*) NODES
C    COUNTER=1
C    DO 10 I=1,NODES
C       NODEPOINT(I)=COUNTER
C    READ(8,*) NADJ,(EDGES(J),J=COUNTER,(COUNTER+NADJ-1))
C    COUNTER=COUNTER+NADJ
C 10 CONTINUE
C NODEPOINT(NODES+1)=COUNTER
C READ(8,*) EPS
C
C READ IN RANDOM GRAPH
C
ACCEPT *,NODES,SEED,PROB,EPS
CALL RANDOMGRAPH(SEED,PROB,NODES,NODEPOINT,EDGES)
C
C CALL ROUTINE TO SET UP THINGS FOR THE SIMPLEX ALGORITHM
CALL COLINIT(AM,BM,CM,NODES,KB1,BASIS,EDGES,NODEPOINT,
  * NVARS,SURP1,SURP2)
C LOOP TO REPEETITIVELY ADD COLUMNS STARTS HERE
C CALL THE SIMPLEX ALGORITHM FOR THE FIRST TIME
NCOLMNS=200
NROWS=NODES
CALL SIMPLE(2,NROWS,NVARS,AM,BM,CM,KM,KB1,DS,BASIS,BINV,
  * 0,ADDRD,VER1,PE1,X1)
DO 15 J=1,NVARS
  KBJ=KB1(J)
  PS(J)=0
  IF (KBJ.NE.0) PS(J)=X1(KBJ)
15 CONTINUE
999 FORMAT(100F5.2)
CALL PRINTSOL(PS,DS,CM,NVARS,NROWS,SURP1,SURP2,AM)
C CHECK TO SEE IF THERE IS A NEW COLUMN TO ADD
DO 30 I=1,NODES
  NDS(I)=-DS(I)
30 CONTINUE
FLAG = MAXSET(EDGES,NODEPOINT,NODES,NDS,NEWCOL,EPS)
IF (FLAG) THEN
C ADDING A COLUMN
  PROD=0
  DO 8 I=1,NROWS
    PROD = PROD + DS(I)*NEWCOL(I)
  8 CONTINUE
  ADDRD=PROD + 1
  IF (NVARS.LT.NCOLMNS) THEN
NVARS=NVARS+1
TEMPVARS=NVARS
ELSE
   I=1
99 IF (KB1(I).EQ.0) THEN
   TEMPVARS=I
   GO TO 88
END IF
I=I+1
GO TO 99
END IF
88 DO 13 I=1,NROWS
   AM(I,TEMPVARS)=NEWCOL(I)
13 CONTINUE
CM(TEMPVARS)=1
PS(TEMPVARS)=0
CALL SIMPLE(0,NROWS,NVARS,AM,BM,CM,KM,KB1,DS,BASIS,BINV,
* TEMPVARS,ADRED,VER1,PE1,X1)
DO 17 J=1,NVARS
   KBJ=KB1(J)
   PS(J)=0
   IF (KBJ.NE.0) PS(J)=X1(KBJ)
17 CONTINUE
CALL PRINTSOL(PS,DS,CM,NVARS,NROWS,SURP1,SURP2,AM)
GO TO 11
ELSE
C WE ARE DONE
WRITE(9,*) 'THE LAST SOLN WAS OPTIMAL'
PRINT *, 'THE LAST SOLN WAS OPTIMAL'
ENDIF
END

C
C
SUBROUTINE RANDOMGRAPH(SEED,PROB,W,POINTERS,EDGES)
INTEGER SEED,I,J,K,ADJ(200,200),POINTERS(1),EDGES(1)
REAL PROB,RANDOM
DO 20 I=2,W
   DO 10 J=1,I-1
      IF(RANDOM(SEED).LT.PROB) THEN
         ADJ(J,I)=1
         ADJ(I,J)=1
10 CONTINUE
20 CONTINUE
ELSE
   ADJ(J,I)=0
   ADJ(I,J)=0
END IF
10 CONTINUE
   ADJ(I,I)=0
20 CONTINUE
   K=1
   DO 40 I=1,N
      POINTERS(I)=K
   DO 30 J=1,N
      IF(ADJ(I,J).EQ.1) THEN
         EDGES(K)=J
         K=K+1
      END IF
30 CONTINUE
40 CONTINUE
   POINTERS(N+1)=K
END

C
FUNCTION RANDOM(SEED)
   INTEGER SEED
   SEED=MOD(25173*SEED+13849,65536)
   RANDOM=(1.0*SEED)/65536
END

C

SUBROUTINE PRINTSOL(PS,DS,CM,NVARS,NOBS,SURP1,SURP2,AM)
   REAL PS(1),DS(1),CM(1),AM(200,1)
   INTEGER SURP1,SURP2
C PRINT OUT THE PRIMAL SOLUTION TO THE LP
   WRITE(9,900)
   PRINT 900
   WRITE(9,910)
   PRINT 910
   DO 12 J=1,NVARS
      WRITE(9,920) J,PS(J)
      PRINT 920,J,PS(J)
      IF ((PS(J).GT.0.0).AND.
         C         ((J.LT.SURP1).OR.(J.GT.SURP2))) THEN
         WRITE(9,990)
PRINT 990
DO 13 I=1,NOBS
   IF (AM(I,J).GT.0.0) THEN
      PRINT 991,I
      WRITE(9,991) I
   END IF
13 CONTINUE
END IF
12 CONTINUE
C PRINT OUT THE DUAL SOLUTION TO THE LP
WRITE(9,930)
PRINT 930
WRITE(9,940)
PRINT 940
DO 14 I=1,NOBS
   WRITE(9,920) I,-1*DS(I)
   PRINT 920,I,-1*DS(I)
14 CONTINUE
C CALCULATE AND PRINT OUT THE OBJECTIVE VALUE OF THE LP
VALUE=0
DO 16 J=1,NVARS
   VALUE=VALUE+CM(J)*PS(J)
16 CONTINUE
WRITE(9,950) VALUE
PRINT 950,VALUE
900 FORMAT (2X,'THE PRIMAL SOLUTION IS')
910 FORMAT (9X,'J',20X,'IJ')
920 FORMAT (7X,I3,13X,F15.8)
930 FORMAT (2X,'THE DUAL SOLUTION IS')
940 FORMAT (9X,'I',20X,'I')
950 FORMAT (2X,'THE OBJECTIVE VALUE IS',3X,F15.8)
990 FORMAT (2X,'THE INDEPENDENT SET IS NODES:')
991 FORMAT (2X,I3)
RETURN
END

C

SUBROUTINE COLINIT(A,B,C,NODES,KB,BASIS,EDGES,POINTER, *
   NVARS,SURP1,SURP2)
C
REAL A(200,1),B(1),C(1)
INTEGER EDGES(1), POINTER(1), KB(1), BASIS(1), NEW(200)
REAL WEIGHTS(200), SET(200)
LOGICAL FLAG, MAXSET, FIRST, FOUND, COVERED(200)
INTEGER SURP1, SURP2

C

C SET UP A MATRIX AND BASIS VECTORS
DO 10 I=1,NODES
   WEIGHTS(I)=2
   BASIS(I) = 0
10 CONTINUE
NVARS=0
NFEAS=0
DO 11 I=1,NODES
   NEW(I) = 0
11 CONTINUE
15 IF (NFEAS.EQ.NODES) GO TO 21
   NVARS=NVARS+1
   FLAG = MAXSET(EDGES, POINTER, NODES, WEIGHTS, SET, EPS)
DO 20 I=1,NODES
   A(I,NVARS)=SET(I)
   IF ((SET(I).EQ.1.0).AND.(WEIGHTS(I).EQ.2.0)) THEN
      WEIGHTS(I)=0
      NFEAS=NFEAS+1
      NEW(I) = NVARS
   END IF
20 CONTINUE
GO TO 15

21 DO 25 I=1,NODES
   COVERED(I)=.FALSE.
25 CONTINUE
DO 29 J=NVARS,1,-1
   FOUND=.FALSE.
   DO 33 I=1,NODES
      IF (FOUND) GO TO 35
      IF (.NOT.COVERED(I)).AND.(NEW(I).EQ.J)) THEN
         FOUND=.TRUE.
         BASIS(I)=J
         KB(J)=I
      ENDFI
33 CONTINUE
35 IF (FOUND) THEN
   DO 37 I=1,NODES
      IF (A(I,J) .NE. 0.0) COVERED(I) = .TRUE.
   CONTINUE
37 ENDIF
29 CONTINUE
   DO 22 I=1,NODES
      IF (BASIS(I) .EQ. 0) THEN
         BASIS(I) = NVARS + I
         KB(NVARS + I) = I
      ELSE
         KB(NVARS + I) = 0
      ENDIF
22 CONTINUE

C SET UP COSTS
   DO 30 I=1,NVARS
      C(I) = 1
30 CONTINUE
C SET UP RIGHT HAND SIDE
   DO 40 I=1,NODES
      B(I) = 1
40 CONTINUE
C SET UP SURPLUSES
   DO 50 I=1,NODES
      DO 50 J=1,NODES
         A(I,NVARS + J) = 0
50 CONTINUE
      A(I,NVARS + I) = -1
50 CONTINUE
SURT1 = NVARS + 1
SURT2 = NVARS + NODES
C COSTS FOR SURPLUSES
   DO 70 I=1,NODES
      C(NVARS + I) = 0
70 CONTINUE
NVARS = NVARS + NODES
RETURN
END
C
SUBROUTINE SIMPLE(INFLAG,MX,NN,A,B,C,KOUT,KB,P,JH,E,
    NEWIND,NEWRED,VER,PE,X)
    * 
    DIMENSION A(200,1),B(1),C(1) 
    DIMENSION KOUT(7),P(1),JH(1),XOLD(200) 
    DIMENSION X(1),Y(200),PE(1),E(1),KO(7),KB(1),OLDKB(600) 
    EQUIVALENCE (K,KO(1)),(ITER,KO(2)),(INVC,KO(3)),
    1(NUMVR,KO(4)),(NUPIV,KO(5)),(FEAS,KO(6)),(JT,KO(7)) 
    EQUIVALENCE (XX,LL) 
C THE FOLLOWING DIMENSION SHOULD BE SAME HERE AS IN CALLER. 
    INTEGER NEWIND 
    REAL NEWRED 
    LOGICAL NOMPI,FEAS,VER,NEG,TRI,TRG,KQ,ABSC 
C 
    M=MX 
    N=NN 
    TEXP=.5**14 
    MAXDEC = 10 
    NDEC = 0 
    NCUT=4*M+10 
    NVER=M/2+ 5 
    M2=M**2 
    NOMPI=.TRUE. 
    IF (NEWIND.EQ.0) GO TO 1348 
C ELSE NEW COLUMN TO ADD, NO NEED TO SELECT PIVOT 
C RESET SOME OUTPUTS 
    KO(1)=0 
    KO(2)=0 
    KO(3)=0 
    JT = NEWIND 
    BB = NEWRED 
    GO TO 600 
C MOVE INPUTS ... ZERO OUTPUTS 
    1348 TRIG = .FALSE. 
    DO 1341 I=1,7 
      KO(I)=0 
    1341 CONTINUE 
    IF(INFLAG.NE.0) GO TO 1400
C* 'NEW'  START PHASE ONE WITH SINGLETON BASIS
  D01402 J=1,N
  KB(J)=0
  KQ=.FALSE.
  D01403 I=1,M
  IF(A(I,J).EQ.0.0) GO TO 1403
  IF(KQ.OR.A(I,J).LT.0.0) GO TO 1402
  KQ=.TRUE.
  1403 CONTINUE
  KB(J)=1
  1402 CONTINUE
  1400 IF(INFLAG.GT.1) GO TO 1320
  D01401 I=1,M
  JH(I)=-1
  1401 CONTINUE
C* 'VER'  CREATE INVERSE FROM 'KB' AND 'JH'
  1320 VER=.TRUE.
  1121 INVC=0
  1122 NUMVR=NUMVR+1
  D01101 I=1,M
  E(I)=0.0
  1101 CONTINUE
  MM=1
  D01113 I=1,M
  E(MM)=1.0
  PE(I)=0.0
  X(I)=B(I)
  IF(JH(I).NE.0) JH(I)=-1
  MM=MM+M+1
  1113 CONTINUE
C  FORM INVERSE
  JT=0
  1101 JT=JT+1
  IF(KB(JT).EQ.0.0) GO TO 1102
  GO TO 600
C  600 CALL JMY
C  CHOOSE PIVOT
  1114 TY=0.0
  D01104 I=1,M
  IF(JH(I).NE.-1) GO TO 1104
  IF(ABS(Y(I)).LE.TY) GO TO 1104
IR=1
TY=ABS(Y(I))
1104 CONTINUE
KB(JT)=0
C TEST PIVOT
   IF(TY.LT.TPIV) GO TO 1102
C PIVOT
       JH(IR)=JT
       KB(JT)=IR
       GO TO 900
C 900 CALL PIV
1102 CONTINUE
   IF(JT.LT.N) GO TO 11101
C RESET ARTIFICIALS
   DO1109 I=1,M
   IF(JH(I).EQ.-1) JH(I)=0
1109 CONTINUE
1200 VER=.FALSE.
C PERFORM ONE ITERATION
C* 'XCK' DETERMINE FEASIBILITY
   FEAS=.TRUE.
   NEG=.FALSE.
   DO1201 I=1,M
   IF(X(I).LT.0.0) GO TO 1250
   IF(JH(I).EQ.0) FEAS=.FALSE.
1201 CONTINUE
C* 'GET' GET APPLICABLE PRICES
   IF(.NOT.FEAS) GO TO 501
C PRIMAL PRICES
   DO 503 I=1,M
       P(I)=PE(I)
503 CONTINUE
   ABSC=.FALSE.
   GO TO 599
C COMPOSITE PRICES
1250 FEAS=.FALSE.
   NEG=.TRUE.
   DO504 J=1,M
       P(J)=0.
      504 CONTINUE
   ABSC=.TRUE.
D0505I=1,M
MM=I
IF(X(I).GE.0.0)GO TO 507
ABSC=.FALSE.
D0508J=1,M
P(J)=P(J)+E(MM)
MM=MM+M
508 CONTINUE
GO TO 505
507 IF(JH(I).NE.0) GO TO 505
IF(X(I).NE.0) ABSC=.FALSE.
D0510J=1,M
P(J)=P(J)-E(MM)
MM=MM+M
510 CONTINUE
505 CONTINUE
C* 'MIN' FIND MINIMUM REDUCED COST
599 JT=0
IF (NDEG.GE.MAXDEG) JT = NVARS+1
BB=0.0
D0701J=1,N
C SKIP COLUMNS IN BASIS
IF(KB(J).NE.0) GO TO 701
DT=0.0
D0303I=1,M
IF(A(I,J).NE.0.0) DT=DT+P(I)*A(I,J)
303 CONTINUE
IF(FEAS) DT=DT+C(J)
IF(ABSC) DT=-ABS(DT)
IF((NDEG.GE.MAXDEG.AND.DT.LT.0.0.AND.J.LT.JT).OR.
* (NDEG.LT.MAXDEG.AND.(DT.LT.BB)))THEN
IF (NDEG.GE.MAXDEG) WRITE(6,*) 'ANTI-CYCLING KICKING IN'
BB=DT
JT=J
END IF
701 CONTINUE
C TEST FOR NO PIVOT COLUMN
IF((JT.LE.0.AND.NDEG.LT.MAXDEG).OR.
* (JT.GE.NVARS+1.AND.NDEG.GE.MAXDEG))
GO TO 203
C TEST FOR ITERATION LIMIT EXCEEDED
590 IF(ITER.GE.NCUT) GO TO 160
   ITER=ITER+1
C* 'JMY'  MULTIPLY INVERSE TIMES A(.J,TJ)
600 DO610I=1,M
   Y(I)=0.0
610 CONTINUE
   LL=0
   COST=C(JT)
   DO605I=1,M
   AIJT=A(I,JT)
   IF(AIJT.EQ.0.) GO TO 602
   COST=COST+AIJT*PE(I)
   DO606J=1,M
   LL=LL+1
   Y(J)=Y(J)+AIJT*E(II)
606 CONTINUE
GO TO 605
602 LL=LL+M
605 CONTINUE
C  COMPUTE PIVOT TOLERANCE
   YMAX=0.0
   DO620I=1,M
      YMAX=AMAX1(ABS(Y(I)),YMAX)
620 CONTINUE
   TPIV=YMAX*TEXP
C  RETURN TO INVERSION ROUTINE, IF INVERTING
   IF(VER) GO TO 1114
C  COST TOLERANCE CONTROL
   IF(TRIG.AND.BB.GE.-TPIV) GO TO 203
   TRIG=.FALSE.
   IF(BB.GE.-TPIV) THEN
      TRIG=.TRUE.
   ELSE
      NOPIV = .FALSE.
   ENDIF
C* 'ROW'  SELECT PIVOT ROW
C  AMONG EQUATIONS WITH X=0, FIND MAXIMUM Y
C  AMONG ARTIFICIALS, OR, IF NONE,
C  GET MAX POSITIVE Y(I) AMONG REALS.
1000 IF (NDEG.GT.10) THEN
   IR=0
IND = M+1
DO 1051 I=1,M
   IF (X(I).NE.0.0.OR.Y(I).LE.TPIV) GO TO 1050
   IF (JH(I).GE.IND) GO TO 1051
   IND=JH(I)
   IR=I
1051 CONTINUE
ELSE
   IR=0
   AA=0.0
   KQ=.FALSE.
   DO1050I=1,M
      IF(X(I).NE.0.0.OR.Y(I).LE.TPIV) GO TO 1050
      IF(JH(I).EQ.0) GO TO 1044
      IF(KQ) GO TO 1050
   1045 IF(Y(I).LE.AA) GO TO 1050
   GO TO 1047
   1044 IF(KQ) GO TO 1045
   KQ=.TRUE.
   1047 AA=Y(I)
   IR=I
   1050 CONTINUE
END IF
   IF(IR.NE.0) GO TO 1099
1001 AA=1.0E+20
C FIND MIN. PIVOT AMONG POSITIVE EQUATIONS
   DO1010I=1,M
      IF(Y(I).LE.TPIV.OR.X(I).LE.0.0.OR.Y(I)*AA.LE.X(I))
         GO TO 1010
      AA=X(I)/Y(I)
      IR=I
   1010 CONTINUE
IF(.NOT.NEG) GO TO 1099
C FIND PIVOT AMONG NEGATIVE EQUATIONS,
C IN WHICH X/Y IS LESS THAN THE
C MINIMUM X/Y IN THE POSITIVE EQUATIONS8
C THAT HAS THE LARGEST ABSF(Y).
1016 BB=-TPIV
   DO1030I=1,M
      IF(X(I).GE.0.0.OR.Y(I).GE.BB.OR.Y(I)*AA.GT.X(I))
         GO TO 1030
1030 CONTINUE
BB=Y(I)
IR=I
1030 CONTINUE
C TEST FOR NO PIVOT ROW.
1099 IF(IR.LE.0) GO TO 207
C* 'PIV' PIVOT ON (IR,JT)
900 IF (X(IR).LE.0) THEN
   NDEG=NDEG+1
ELSE
   NDEG=0
END IF
C LEAVE TRANSFORMED COLUMN IN Y(I).
   NUMPV=NUMPV+1
   YI=-Y(IR)
   Y(IR)=-1.0
   LL=0
C TRANSFORM INVERSE
   DO904J=1,M
   L=LL+IR
   IF(E(L).NE.0.0) GO TO 905
   LL=LL+M
   GO TO 904
905 XY=E(L)/YI
   PE(J)=PE(J)+COST*XY
   E(L)=0.0
   DO906I=1,M
   LL=LL+1
   E(LL)=E(LL)+XY*Y(I)
906 CONTINUE
904 CONTINUE
C TRANSFORM X.
   XY=X(IR)/YI
   DO908I=1,M
   XNEW=X(I)+XY*Y(I)
   IF(VER.OR.XNEW.GE.0.0.OR.Y(I).GT.TPIV.OR.X(I).LT.0.)
      GO TO 907
   X(I)=0.0
   GO TO 908
907 X(I)=XNEW
908 CONTINUE
C RESTORE Y(IR)
Y(IR)=-YI
X(IR)=-XY
IF(VER) GO TO 1102
221 IA=JH(IR)
    IF(IA.GT.0) KB(IA)=0
213 KB(JT)=IR
    JH(IR)=JT
    IF(NUMPV.LE.0) GO TO 1200
C TEST FOR INVERSION ON THIS ITERATION.
    INV=INV+1
    IF(INV.EQ.NVER) GO TO 1320
    GO TO 1200
C* END OF ALGORITHM, SET EXIT VALUES.
C INFINITE SOLUTION.
    207 K=2
    GO TO 250
C PROBLEM IS CYCLING.
    160 K=4
    GO TO 250
C FEASIBLE OR INFEASIBLE SOLUTION.
    203 K=0
    250 IF(.NOT.FEAS) K=K+1
C SET 'KOUT'.
1392 D01393I=1,7
    KOUT(I)=KO(I)
1393 CONTINUE
RETURN
END
C
C

LOGICAL FUNCTION MAXSET(EDGES,NODEPOINT,NODENUM,INWT,
    C
    INDSET,EPS)

INTEGER FILLNUM,I,J,K,NODE,NNUM,LEVEL,NODENUM
INTEGER EDGES(1),NODEPOINT(1),LIST(200)
INTEGER EDGE(38000),PT(200),FILLNODE(200)
INTEGER SIGNODE(200)
REAL INWT(1),WT(200),INDSET(1),EPS
LOGICAL FLAG,SETFOUND

50
DO 1 I=1,NODENUM
   SIGNODE(I)=0
   PT(I)=0
   WT(I)=0
   INDSET(I)=0
1 CONTINUE
DO 2 I=1,38000
   EDGE(I)=0
2 CONTINUE
   K=1
   FILLNUM=0
   NNUM=0
DO 5 I=1,NODENUM
   IF (ABS(INWT(I)).GT.(EPS/(2*NODENUM))) THEN
      NNUM=NNUM+1
      PT(NNUM)=K
      SIGNODE(NNUM)=I
      WT(NNUM)=INWT(I)
   C
   IF (NODEPOINT(I).NE.(NODEPOINT(I+1)-1)) THEN
      DO 3 J=NODEPOINT(I),NODEPOINT(I+1)-1
      IF (INWT(EDGES(J)).GT.(EPS/(2*NODENUM))) THEN
         EDGE(K)=EDGES(J)
         K=K+1
      END IF
3 CONTINUE
   C
   END IF
ELSE
   FILLNUM=FILLNUM+1
   FILLERNODE(FILLNUM)=I
END IF
5 CONTINUE
   PT(NNUM+1)=K
   IF (NNUM.NE.1) THEN
      DO 7 I=1,K-1
         J=1
      6 IF (EDGE(I).EQ.SIGNODE(J)) THEN
         EDGE(I)=J
      ELSE
         J=J+1
      GO TO 6
   END IF
7 CONTINUE
CONTINUE
END IF
FLAG=.FALSE.
FLAG=SETFOUND(INDSET,NNUM,WT,PT,EDGE,SIGNODE, 
C EPS,LIST,LEVEL)
IF (FLAG) THEN
CALL FILLGRAPH(FILLNUM,LEVEL,LIST,FILLERNODE, 
C INDSET,NODEPOINT,EDGES,NNUM,SIGNODE)
MAXSET=.TRUE.
ELSE
MAXSET=.FALSE.
END IF
RETURN
END

LOGICAL FUNCTION SETFOUND(INDSET,NNUM,WT,PT,EDGE, 
C SIGNODE,EPS,LIST,LEVEL)

INTEGER I,J, NODE, NNUM, GETVAR, LEVEL 
INTEGER SUBG(200), LIST(200) 
INTEGER EDGE(38000), PT(200), SIGNODE(200) 
REAL WT(200), INDSET(200), AVAILWEIGHT, EPS 

LEVEL=0
SETFOUND=.FALSE.
IF (NNUM.EQ.1) THEN
  SETFOUND=.TRUE.
  LEVEL=1
  LIST(1)=SIGNODE(1)
  INDSET(SIGNODE(1))=1
  RETURN
END IF
AVAILWEIGHT=0.0
DO 10 I=1, NNUM
  SUBG(I)=-1
  AVAILWEIGHT=AVAILWEIGHT + WT(I)
10   CONTINUE
16 IF (AVAILWEIGHT.GT.(1.0+(EPS/2))) THEN
    NODE = GETVAR(NNUM, SUBG, PT, EDGE, WT)
52
IF (NODE.EQ.-1) THEN
    AVAILWEIGHT = 0.0
    DO 22 I=1,LEVEL
        AVAILWEIGHT = AVAILWEIGHT + WT(LIST(I))
    22   CONTINUE
    IF (AVAILWEIGHT.LE.(1.0+(EPS/2))) THEN
        GOTO 25
    ELSE
        SETFOUND=.TRUE.
        RETURN
    END IF
END IF
LEVEL=LEVEL + 1
LIST(LEVEL)=NODE
SUBG(NODE)=LEVEL
C IF (PT(NODE).NE.(PT(NODE+1)-1)) THEN
    DO 20 I=PT(NODE),PT(NODE+1)-1
        J=EDGE(I)
        IF (SUBG(J).EQ.-1) THEN
            SUBG(J)=LEVEL
            AVAILWEIGHT=AVAILWEIGHT-WT(J)
        END IF
    20   CONTINUE
C ELSE
    IF (LEVEL.LE.0) THEN
        RETURN
    END IF
25    NODE=LIST(LEVEL)
    AVAILWEIGHT=AVAILWEIGHT-WT(NODE)
    SUBG(NODE)=LEVEL-1
    DO 24 I=1,NNUM
        IF (SUBG(I).EQ.LEVEL) THEN
            SUBG(I)=-1
            AVAILWEIGHT=AVAILWEIGHT+WT(I)
        END IF
    24   CONTINUE
    LEVEL=LEVEL - 1
END IF
GO TO 16
END
INTEGER FUNCTION GETVAR(NNUM, SUBG, PT, EDGE, WT)

INTEGER SUBG(200), PT(200), EDGE(38000)
INTEGER NNUM, I, J, NODEDEG, LOWDEG
REAL WT(200), DEN(200), RAT(200), HIGH

HIGH=0.0
LOWDEG=1000000
GETVAR=-1

DO 40 I = 1, NNUM
   IF (SUBG(I) .EQ. -1) THEN
      NODEDEG=0
      DEN(I)=0.0
      C
      IF (PT(I) .NE. (PT(I+1)-1)) THEN
         DO 30 J = PT(I), (PT(I+1)-1)
            IF (SUBG(EDGE(J)) .EQ. -1) THEN
               DEN(I)=DEN(I) + WT(EDGE(J))
               NODEDEG=NODEDEG + 1
            END IF
         30 CONTINUE
      C
   END IF
   RAT(I)=WT(I)/(DEN(I)+0.000001)
   IF (((RAT(I) .EQ. HIGH .AND. NODEDEG .LT. LOWDEG)
      .OR. RAT(I) .GT. HIGH) THEN
      HIGH=RAT(I)
      GETVAR=I
      LOWDEG=NODEDEG
   END IF
40 CONTINUE
END

SUBROUTINE FILLGRAPH(FILLNUM, LEVEL, LIST, FILLERNODE, INSET, NODEPOINT, EDGES, NNUM, SIGNODE)
C
INTEGER FILLNUM, LEVEL, LIST(1), FILLERNODE(1)
INTEGER NODEPOINT(1), EDGES(1), NNUM, SIGNODE(1), J
REAL INSET(1)

54
IF (NNUM.EQ.1) THEN
    GO TO 1988
END IF
DO 1987 I=1,LEVEL
    LIST(I)=SIGNODE(LIST(I))
    INDSET(LIST(I))=1
1987 CONTINUE
DO 1988 I=1,FILLNUM
    J=0
1989 J=J+1
    IF (J.GT.LEVEL) THEN
        LEVEL=LEVEL+1
        LIST(LEVEL)=FILLERNODE(I)
        INDSET(LIST(LEVEL))=1
    ELSE
        K=NODEPOINT(LIST(J))
    IF (K.GE.NODEPOINT(LIST(J)+1)) THEN
        GO TO 1989
    END IF
    IF (FILLERNODE(I).EQ.EDGES(K)) THEN
        GO TO 1991
    ELSE
        K=K+1
    END IF
1990 CONTINUE
END IF
GO TO 1990
END
REFERENCES


