12-1992

Vertex Coloring for Weighted Graphs with Application to Timetabling

Lynn Kiaer
Rose-Hulman Institute of Technology

Jay Yellen
Florida Institute of Technology - Melbourne

VERTEX COLORING FOR WEIGHTED GRAPHS WITH APPLICATION TO TIMETABLING

Lynn Kiaer and Jay Yellen

MS TR 92-12

December 1992

Department of Mathematics
Rose-Hulman Institute of Technology
Terre Haute, IN 47803

FAX(812) 877-3198

Phone: (812) 877-8391

Vertex Coloring for Weighted Graphs With Application to Timetabling

Lynn Kiaer*
Department of Mathematics
Rose-Hulman Institute of Technology
5500 Wabash Ave.
Terre Haute, IN 47803

Jay Yellen
Department of Applied Mathematics
Florida Institute of Technology
150 W. University Blvd.
Melbourne, FL 32901

December 1, 1992

*Supported in part by an American Fellowship from the American Association of University Women, 1991-92

1 Introduction.

The problem addressed in this paper can be stated as follows: Given finite sets V and T, a function c: $V \times T \rightarrow [0, \infty)$, a function w: $V \times V \rightarrow [0, \infty)$, and a function r: $T \rightarrow \{1, 2, ...\}$, find a function f: $V \rightarrow T$ that satisfies $\mid f^{-1}(t) \mid \leq r(t)$ for each t(where f^{-1} is set-valued), and minimizes $\sum_{v \in V} c(v, f(v)) + \sum_{u,v:f(u)=f(v)} w(u, v)$.

Instances of this problem often arise as timetabling problems. Such a timetabling problem involves assigning events (elements of V) to nonoverlapping timeslots (elements of T) while avoiding certain assignments and avoiding scheduling certain pairs of events to the same or overlapping timeslots. Restrictions on the number of events that can be assigned to particular timeslots are indicated by the integer-valued function r. For clarity of exposition, elements of V and T will be called events and timeslots, respectively, and subscript notation will be used for the functions c, w and r. Throughout this paper, it is assumed that there are n events ($\mid V \mid = n$) to be scheduled in k timeslots ($\mid T \mid = k$).

Approaches to this timetabling problem have generally used a quadratic assignment model [2] [3] [5] in which heuristic improvement algorithms are used to find a local optimum, but due to the nonconvexity of the objective, no gauge on overall solution quality is obtained. Recently, simulated annealing has been used on a quadratic timetabling problem with some success [1]. For problems in which a conflict-free timetable is the objective (i.e., the chromatic number of the underlying graph is less than or equal to $\mid T \mid$), standard vertex coloring [4] [10] has been used as well as a variety of network-flow approaches [12], but these approaches do not allow the user to distinguish between conflicts of varying severity, and they also make it difficult to deal with side constraints such as a limited number of classrooms.

The principal thrust of this paper is to describe a new, graph-based exact algorithm for the problem. The algorithm uses branch-and-bound but works directly with a weighted graph model developed by the authors [9], and its run-time compares very favorably with that of standard, linear-programming based branch-and-bound. The main reason for the success of the graph-based procedure is the ease with which it obtains upper and lower bounds on the value of an optimal solution. Lower bounds are obtained from an embedded partition matroid using a very fast greedy algorithm. Upper bound are found using an approximate algorithm that is based on a family of heuristics

designed by the authors for coloring the vertices of a weighted graph. The approximate algorithm is of low-order polynomial complexity, and computational results indicate that it performs very well on its own in obtaining good solutions [9]. We show, however, that the existence of a performance bound for any polynomial-time approximate algorithm would imply that $P = NP$.

2 The Weighted Graph Model.

In an ordinary graph model for a timetabling problem, the vertices represent the events to be scheduled, and two vertices are adjacent if the associated courses should not be offered at the same time. In this case, a conflict-free schedule is the objective, and the problem is to find the chromatic number of the graph, where each color represents a timeslot.

When k, the number of available timeslots, is small, as is typical in many timetabling applications, it is not reasonable to expect that a conflict-free schedule exists. In this case, a timetable that minimizes some measure of conflict becomes the objective. The traditional graph model for timetabling does not allow for this kind of problem, and the computational difficulties of the quadratic programming approach led the authors to develop the following weighted graph model [9].

Each event to be scheduled is represented by a vertex of the graph, and the cost w_{uv} of assigning events u and v to the same timeslot is represented by an edge uv having weight w_{uv}. Associated with each vertex v is a cost vector whose k components are the costs c_{vt} of scheduling event v in timeslot t. Scheduling the n events in the k timeslots corresponds to coloring the n vertices of the weighted graph with k colors, where a coloring is simply a function $f\colon V \rightarrow T$. Note that there is no requirement that the resulting k-coloring be a proper one in the traditional sense; we expect that some pairs of adjacent vertices will be assigned the same color.

Definition 1 *A k-coloring of a weighted graph is feasible if each color t is used no more than r_t times, i.e., if $\mid f^{-1}(t) \mid \leq r_t$ for each $t \in T$.*

The timetabling problem can then be stated as the following vertex coloring problem: Find a feasible k-coloring of the vertices of the weighted graph that minimizes the sum of the cost vector components associated with the color used for each vertex plus the sum of the weights on those edges whose

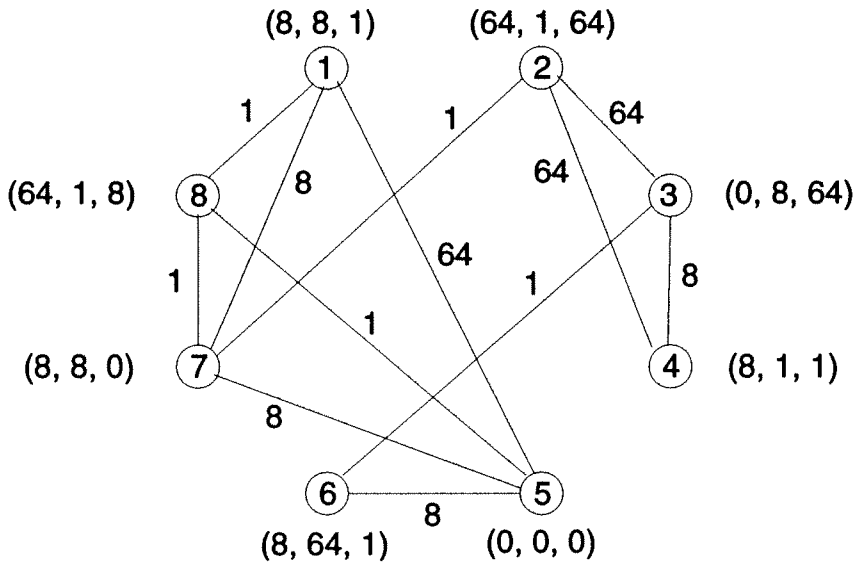

Figure 1: An Example Problem

endpoints receive the same color. This objective function is precisely the objective function in Section 1: $\sum_{v \in V} c(v, f(v)) + \sum_{u,v:f(u)=f(v)} w(u, v)$.

Although this model allows for an unlimited variation of the edge weights which reflect the severity of conflicts, it is often the case that a small number of levels of severity may be sufficient to categorize the conflicts that might arise. A prohibitive conflict is one that must be avoided if the solution is to be useable, and is represented with a large edge weight. In addition we use two lesser levels of conflict, medium and small.

The new model can be illustrated with a simple example: Suppose there are eight events to be scheduled in three time slots, with conflicts weighted as indicated in Figure 1. The three possible edge weights are 1, 8 and 64, corresponding to small, medium, and prohibitive conflicts. For this example, let the classroom vector be $(3, 3, 2)$, indicating that there are three classrooms available in timeslots 1 and 2, and two in timeslot 3. This example will be used to illustrate the exact algorithm as well as the heuristics described in Section 3.

3 The Approximate Algorithm.

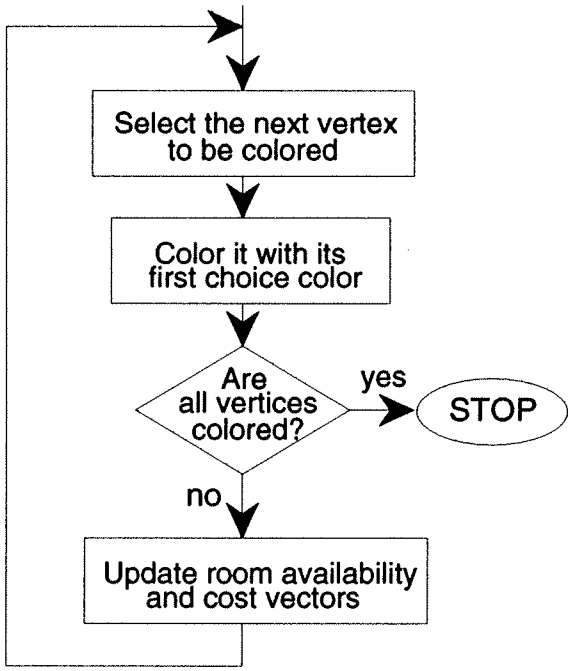

Figure 2: Flowchart for the Approximate Algorithm

The algorithm presented in this section is outlined in Figure 2. The vertices are colored one at a time, and each time a vertex is colored, the cost vectors of the adjacent uncolored vertices are updated.

The algorithm is a modified greedy algorithm that proceeds to sequentially color the vertices of the graph without backtracking, and always colors a vertex with a least-cost color. However, the criterion for selecting the next vertex to color is not simply the one with the cheapest available coloring.

This algorithm has two key features. First, the update ensures that at each iteration, the cost vector of an uncolored vertex reflects the current cost of coloring that vertex with each of the colors. The second feature, and the heart of the algorithm is its first step: vertex selection.

Many vertex coloring heuristics for ordinary (unweighted) graphs are based on the observation that a vertex of large degree (i.e., having a large

number of adjacent vertices) will be more difficult to color than one of smaller degree; see Carter [4], Krarup and deWerra [10], or Matula, Marble, and Isaacson [14] for a discussion of ordinary vertex coloring. The general philosophy that we followed in designing vertex selection criteria for the weighted graph was that the 'most difficult' vertices be colored as early as possible. Each of the criteria described below implements this philosophy in a different way. In each case, the selection criteria used are based on some combination of the current values of a given vertex's cost vector together with the weights of some or all of the edges incident to that vertex.

Definition 2 *At any stage of the coloring algorithm, a color t is said to be available if $r_t > 0$.*

Let A denote the set of available colors and let U denote the set of uncolored vertices.

Definition 3 *For an uncolored vertex v, at any stage of the coloring algorithm,*

i. the cost degree of v, $cdeg(v) = \sum_{t \in A} c_{vt}$.

ii. the uncolored (weighted) degree of v, $udeg(v) = \sum_{u \in U} w_{uv}$.

iii. the total weighted degree of v, $tdeg(v) = cdeg(v) + udeg(v)$.

iv. a color t is forbidden for vertex v if either $r_t = 0$ or the tth component of v's cost vector equals or exceeds some pre-established threshold.
Let F_v denote the set of forbidden colors for vertex v.

v. the forbidden degree of v, $fdeg(v) = \mid F_v \mid$.

vi. Let $K_v = \{t \in A : c_{vt} = \min_{s \in A}\{c_{vs}\}\}$. The difference cost of v,

$$del(v) = \begin{cases} 0 & \text{if } \mid K_v \mid > 1 \\ \infty & \text{if } \mid T - A \mid = 1 \\ \min_{s \in A - K_v}\{c_{vs}\} - \min_{s \in A}\{c_{vs}\} & \text{otherwise.} \end{cases}$$

Clearly, the larger the $fdeg$, the more difficult it is to color that vertex. Forbidden degree used alone results in a large number of ties, since there are only k possible values. For that reason, one of the other parameters is always used as a tiebreaker.

Table 1 gives the values of these parameters for each of the eight vertices in the example problem (see Figure 1). The threshold used for forbidden degree is $n^2 = 64$.

Vertex	$cdeg(v)$	$udeg(v)$	$tdeg(v)$	$fdeg(v)$	$del(v)$
1	17	73	90	0	7
2	129	129	258	2	63
3	72	73	145	1	8
4	10	72	82	0	0
5	0	81	81	0	0
6	73	9	82	1	7
7	16	18	34	0	8
8	73	3	76	1	7

Table 1: Heuristic Parameters for the Example Problem

A family of algorithms results when one or more of the parameters are used to define the selection criterion employed in an approximate coloring procedure. See Kiaer and Yellen [9] for a description and evaluation of the resulting algorithms.

4 The Weighted Graph k-coloring Problem and NP.

The decision problem analog of the weighted graph k-coloring problem is: Given a constant c, does there exist a feasible k-coloring of the weighted graph whose cost is less than or equal to c? It is easy to show that this problem is NP-complete.

It is desirable to be able to provide a performance guarantee for a polynomial-time approximate algorithm. To establish a performance guarantee it is necessary to find a constant c such that the value of the solution found by the approximate algorithm is no more than c times the value of the unknown optimal solution. This has been done in a number of cases, perhaps most notably for bin-packing problems [11] and for travelling salesman problems which satisfy the triangle inequality [21]. However, the following theorem explicitly ties the existence of such a performance guarantee for a polynomial-time

approximate algorithm for this problem to the resolution of the $P \neq NP$ conjecture.

Theorem 1 *If there is a polynomial-time approximate algorithm A for the weighted graph k-coloring problem and a constant c, $1 \leq c < \infty$, such that for all instances I of the problem, the value of the solution of instance I found by the approximate algorithm is less than or equal to c times the value of the optimal solution of instance I, then $P = NP$.*

Proof. The proof proceeds by showing that if such a bound existed, the algorithm would be able to solve the NP-complete problem of determining whether a given graph is k-colorable. Given $G = (V, E)$, with $\mid V \mid = n$, transform the problem into an instance I of the weighted graph k-coloring problem by letting the weight of edge uv be cn, the cost vector for each vertex be $(1, 1, \ldots, 1)$, and $r_t = n$ for all t. If a k-coloring of the vertices of G exists, then I has a k-coloring of cost n, in which the only costs incurred are the 1's from the cost vectors. Thus, if the algorithm A is applied to I, it must yield a solution whose cost is no more than cn. However, if G is not k-colorable, then any k-coloring of I must violate at least one of the edges weighted cn, and so must incur a cost of at least $n(c + 1)$. Thus for any graph G, we can determine whether G is k-colorable by constructing the associated weighted graph k-coloring instance I and applying the algorithm A. Since I can be constructed in polynomial time, and A is by assumption a polynomial-time algorithm, the procedure described provides a polynomial-time algorithm for determining whether an arbitrary graph is k-colorable. □

Thus, finding performance bounds for the weighted graph k-coloring problem is possible only in the unlikely event that $P = NP$.

5 Graph-Based Branch-and-Bound.

The success of any branch-and-bound algorithm hinges on its ability to quickly obtain both feasible solutions and good lower bounds for the subproblems. Without these characteristics, the branch-and-bound procedure will not be able to eliminate enough subtrees of the enumeration tree of solutions, and the number of subproblems that require explicit examination will grow quickly.

In standard integer programming branch-and-bound the lower bound for each subproblem is obtained using its linear programming relaxation. An integer linear programming formulation of the weighted graph k-coloring problem is shown below. Let

$$x_{vt} = \begin{cases} 1 & \text{if event } v \text{ is scheduled in timeslot } t \\ 0 & \text{otherwise} \end{cases}$$

and for each edge uv, let

$$z_{uv} = \begin{cases} 1 & \text{if events } u \text{ and } v \text{ are assigned the same timeslot} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Minimize } \sum_{v=1}^{n} \sum_{t=1} kc_{vt} x_{vt} + \sum_{v=1}^{n-1} \sum_{u=v+1}^{n} w_{uv} z_{uv}$$

$$\begin{array}{lll} \text{subject to} & \sum_{t=1}^{k} x_{vt} = 1 & \text{for all } v \\ & \sum_{v=1}^{n} x_{vt} \leq r_t & \text{for all } t \\ & x_{vt} + x_{ut} - z_{uv} \leq 1 & \text{for all edges } uv \text{ and all } t \\ & x_{vt} \in \{0,1\} & \text{for all } v, t \\ & z_{uv} \in \{0,1\} & \text{for all edges } uv \end{array}$$

In the linear relaxation, setting $x_{vt} = x_{ut} = \frac{1}{2}$ allows constraint (4) to be satisfied with $z_{uv} = 0$, and thus avoids incurring the w_{uv} penalties. The value of this solution is therefore likely to be a very weak lower bound on the integer optimum for the subproblem. Furthermore, since so many of the variables will start out with fractional values, integer solutions will be produced only very deep in the enumeration tree. Thus, neither of the characteristics required for a branch-and-bound procedure to be successful on this problem is possessed by the standard branch-and-bound algorithm.

For this problem the disadvantages of working with an integer programming model can be avoided. Embedding the approximate algorithm and several associated heuristics within a branch-and-bound framework results in the following, graph-based branch-and-bound algorithm. To avoid confusion in referring to the two graph structures, the weighted graph model and the enumeration tree of partial solutions, being discussed here, the weighted

graph will be referred to as having vertices and the enumeration tree as having nodes.

The algorithm will terminate with a solution whose value is within a distance ϵ of the optimum, provided that the problem has a feasible solution. Of course, if ϵ is set to 0 then the algorithm will find an exact solution. In the following algorithm description, UB and LB denote the current upper and lower bounds, respectively, on the value of an optimal solution, and C denotes the set of candidate partial colorings, while $L(PC)$ is the lower bound for a particular partial coloring PC.

Graph-Based Branch-and-Bound Algorithm

Step 0: Initialization

Let $\epsilon \geq 0$ be a user-specified acceptable distance from the value of an optimal solution.

$UB \leftarrow \infty$ and $LB \leftarrow L(originalproblem)$.

$C \leftarrow \{originalproblem\}$.

If $\sum_t r_t < n$, then STOP; original problem is infeasible.

Step 1: Candidate Selection.

Select $PC \in C$ such that $L(PC) = LB$. $C \leftarrow C - \{PC\}$.

Step 2: Checking for a New Incumbent.

Apply the approximate algorithm to PC to obtain $ApproxSol$.

If $Cost(ApproxSol) < UB$,
then $Incumbent \leftarrow ApproxSol$, $UB \leftarrow Cost(ApproxSol)$, and
$C \leftarrow C - \{P : L(P) \leq UB - \epsilon\}$.

Step 3: Branching.

If $L(PC) < UB - \epsilon$, then use a vertex selection heuristic to select a vertex v that is uncolored in PC.

For each available color t,
if $PC(v,t)$ is a complete coloring and $Cost(PC(v,t)) < UB$,
then $Incumbent \leftarrow PC(v,t)$, and $UB \leftarrow Cost(PC(v,t))$.
If $L(PC(v,t)) < UB - \epsilon$, then $C \leftarrow C + \{PC(v,t)\}$.

$LB \leftarrow \min\{L(P) : P \in C\}$.

Step 4: Termination.

If $UB - LB \leq \epsilon$ or $C = \Phi$, then STOP, else go to Step 1.

This algorithm works directly with the weighted graph model, and offers several advantages over standard branch-and-bound. First, feasible solutions are found quickly by applying the approximate algorithm to the candidate partial colorings. Second, basing the branch-and-bound scheme on the underlying graph immediately streamlines the branching process, since the generalized upper bound constraint (2) is automatically recognized. In addition, the depth of a particular node in the standard procedure can be misleading. Indeed, since the standard solution tree will not, in general, fix all the variables associated with a particular vertex in successive branching steps, nodes equivalent to partial colorings with very few fixed colors can appear very deep in the standard solution tree.

The principal advantage of the graph-based branch-and-bound algorithm is the speed with which it is able to obtain the lower bound for each partial coloring. These lower bounds are obtained from a certain matroid structure embedded in the weighted graph k-coloring problem. A matroid consists of a finite set E and a family I of subsets of E such that

i. I and all proper subsets of a set $J \in I$ are in I, and

ii. for any $A \in E$, if J and J' are maximal subsets of A in I, then $\mid J \mid = \mid J' \mid$.

The family I of subsets is said to be a family of independent subsets. A partition matroid $M = (E, I)$ is a matroid in which, given a partition of E into sets P_i, $i = 1, 2, \ldots, m$, and non-negative constants d_i, $i = 1, 2, \ldots, m$, a subset $J \in I$ if and only if $\mid J \cap P_i \mid \leq d_i$ for all i. (See Chapter 7 of Lawler [11] for a complete discussion of matroids.)

Proposition 1 *There exists a partition matroid which is a relaxation of the weighted graph k-coloring problem.*

Proof. For the weighted graph k-coloring problem, the associated partition matroid is defined by letting E be the components of a matrix whose rows are the cost vectors associated with each uncolored vertex of the weighted graph, partitioning the matrix by rows, letting $d_i = 1$, $i = 1, 2, \ldots, m$, where m is the number of uncolored vertices. We now seek a maximum-cardinality

independent set of minimum weight. This optimal solution will clearly be the minimum weight element of each row of the matrix. (In fact, partition matroid problems are generally maximization problems with non-negative weights, and this problem can easily be converted to the more traditional format by subtracting each of the matrix entries from a largest entry, but the transformation is not necessary in this instance.) Clearly, any feasible coloring for the original problem is also feasible for the partition matroid - as well as are those colorings that violate the restrictions on the number of times each color may be used. Also, the optimal solution of the partition matroid provides a lower bound on the value of the original problem, since the edge weight penalties have been ignored.

In general, the matroid lower bound is not as good as the linear-programming relaxation lower bound.

Definition 4 *A partial coloring (in graph-based branch-and-bound) and a partial solution (in linear-programming-based branch-and-bound) are equivalent if*

i. a vertex is colored in the partial coloring if and only if the associated variable x_{vt} is fixed at 1, and

ii. for each variable x_{vt} that is fixed at 0, color t is a forbidden color for vertex v.

Note that a partial coloring and a partial solution are still equivalent if color t is forbidden for vertex v but the associated x_{vt} has not been fixed to 0.

Proposition 2 *Let LB_m denote the matroid lower bound and LB_{lp} denote the linear programming relaxation lower bound. Then for an equivalent partial coloring and partial solution, $LB_m \leq LB_{lp}$.*

Proof. In finding the matroid lower bound, each uncolored vertex is assigned its least-cost color, with no updates being performed. This is equivalent to assigning the value 1 to the associated x_{vt} variables. Such an assignment might violate either the room constraint (15) or the conflict constraint (16), in which case the linear-programming relaxation might be forced to incur additional edge penalties.

The following proposition provides a sufficient condition for the matroid lower bound and the linear-programming relaxation lower bound to be equal.

Proposition 3 *The matroid lower bound is equal to the linear-programming-relaxation lower bound if*

i. for all v, $\mid K_v > 1$, *and*

ii. for each timeslot t, $\sum_{v:t \in K_v} \frac{1}{|K_v|} \leq r_t$.

Proof. The linear-programming relaxation will incur an edge penalty only when variables x_{vt} and x_{ut}, which appear in a constraint of the form $x_{vt} + x_{ut} - z_{uv} \leq 1$, have a sum strictly greater than 1. This can occur only if one of the above conditions is violated.

Clearly it is possible for the matroid and linear-programming-relaxation lower bounds to be equal even when condition (i) is violated, so the above conditions are not necessary.

At the branching step, the partial colorings that result when the chosen vertex v is colored with each color are examined. Denoting the partial coloring that colors v with color t by $PC(v,t)$, that partial coloring may be immediately discarded if any of the following conditions hold.

i. Color t has already been used r_t times (r_t has been decremented to 0).

ii. Color t is too expensive for vertex v. This is the case when $c_{vt} - \min_{t \in A}\{c_{vt}\} + LB(PC) \geq UB - \epsilon$.

iii. The lower bound obtained for $PC(v,t)$, after updating the cost vectors of the uncolored vertices, meets or exceeds the value of the incumbent solution.

It is possible that, after fixing a certain subset of vertices with certain colors, other vertices must receive certain colors if the value of the solution is to be less than that of the incumbent. This situation is detected in the branching step, with the result that a child node may have more than one additional colored vertex.

The vertex selection criteria that form the heart of the approximate algorithm are also important in the branching step of the new branch-and-bound algorithm. This is a critical decision point in any branch-and-bound procedure, and the heuristics chosen here have great impact on the success of the algorithm because they affect both the number of subproblems generated and the strength of the lower bounds.

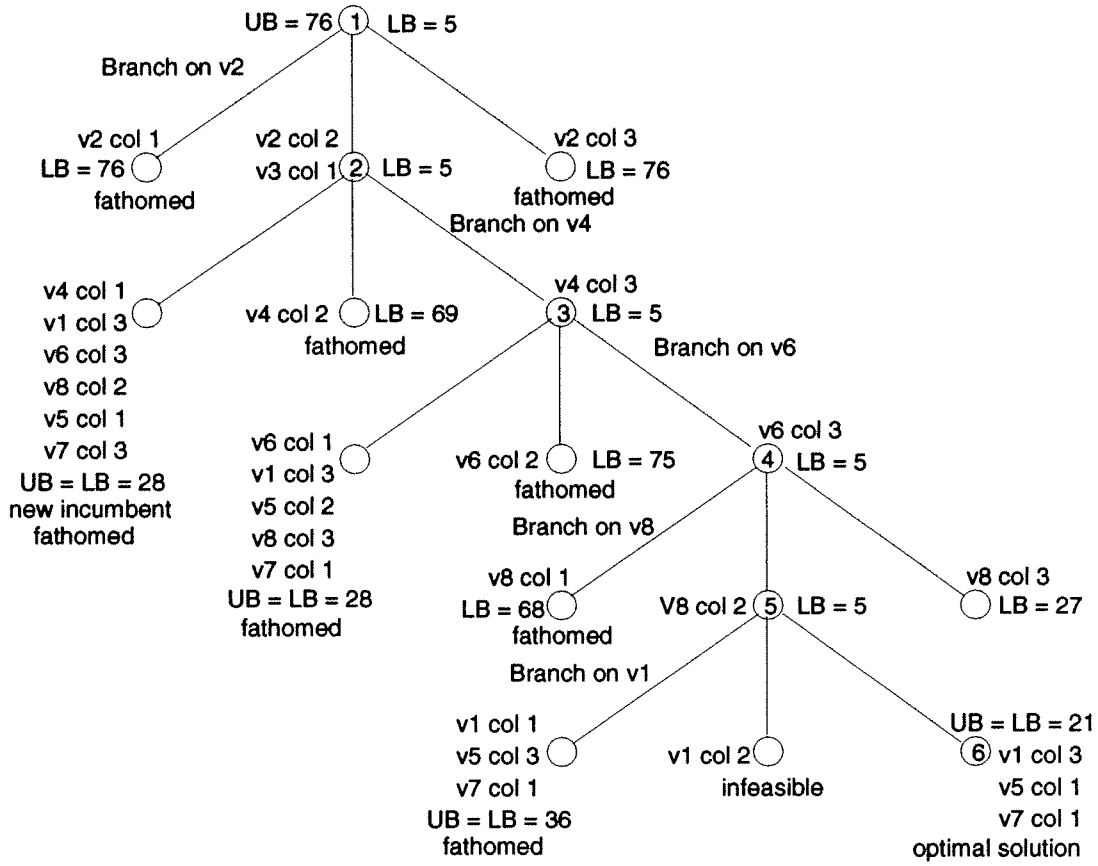

Figure 3: Graph-Based Branch-and-Bound Enumeration Tree

The graph-based branch-and-bound algorithm is illustrated in Figure 3, using an eight-vertex, three-color example. The initial solution found by the approximate algorithm using the *del* vertex selection criterion has vertices 1, 3 and 5 colored with color 1, vertices 2 and 8 colored with color 2, and vertices 4, 6 and 7 colored with color 3 at a total cost of 76. Branching according to largest *cdeg* leads to branching on vertex 2. The complete branch-and-bound tree is shown in Figure 3, with the nodes that must be explicitly examined numbered in the order in which they are examined.

6 Results.

The graph-based algorithm was implemented in Fortran 77 on the Harris HCX-9 and Harris Night Hawk computers. The *del* vertex selection criterion was used to find approximate solutions, based on its performance on a collection 20-, 50- and 100-vertex problems [9] and on initial tests of the graph-based branch-and-bound algorithm on thirty 25-vertex problems which showed that nearly all improvements over the incumbent were found either with the *del* criterion or by fixing all the remaining uncolored vertices in the branching step. Incorporating a second approximate algorithm does not appear to be worthwhile in view of the fact that *del*, working alone, generally found optimal or near-optimal solutions early in the branch-and-bound procedure (see Table 2).

Several sets of ten weighted graph problems were generated involving 25 vertices and from 3 to 7 colors. For all of these problems, the probability that a given edge exists is 0.4, and the cost vector components are equally likely to take on each of the possible values $0, 1, n = 25$, and $n^2 = 625$. The edge weight distributions used are even, with the probability that an edge weight equals $1, n$ or n^2, $P(1) = P(n) = P(n^2) = \frac{1}{3}$; half small, with $P(1) = \frac{1}{2}$ and $P(n) = P(n^2) = \frac{1}{4}$; half medium, with $P(1) = P(n^2) = \frac{1}{4}$, $P(n) = \frac{1}{2}$; and half prohib, with $P(1) = P(n) = \frac{1}{4}$ and $P(n^2) = \frac{1}{2}$.

Each of the problems was then solved using the graph-based branch-and-bound algorithm with each of the five branching criteria *cdeg*, *udeg*, *tdeg*, *del*, and *fdeg* with ties broken by *udeg*. The Friedman Test was performed to test the null hypothesis that the five vertex selection criteria were equally effective in minimizing the number of partial colorings that would have to be explicitly examined. A rank transformation was used to avoid distortions due to non-normality and extreme observations. The null hypothesis was rejected at the $\alpha = 0.01$ level of significance in 11 of the 20 data sets, and at the $\alpha = 0.05$ level of significance in another. The data suggest that the branching criteria had a significant effect on the number of problems that had to be explicitly examined, with *tdeg*, *udeg*, and *fdeg* with *udeg* performing significantly better than the others. Since the partial colorings that must be explicitly examined are only those whose lower bound is less than the value of an optimal solution, a branching criterion which produces fewer such partial colorings will tend to yield a provably optimal solution faster.

Another consideration in evaluating the performance of the branching cri-

teria is the space required for candidate storage. This is especially important as the size of the problem increases. As with the number of partial colorings examined, a rank transformation was performed and the Friedman test used on the resulting data to test the null hypothesis that the choice of branching criteria has no overall impact on the size of the candidate queue. The results were similar to those reported above, with the same criteria performing well. The null hypothesis was rejected at the $\alpha = 0.01$ level of significance in 10 of the 20 data sets, and at the $\alpha = 0.05$ level of significance in three more.

The initial approximate solution was usually improved upon in successive iterations, but optimal or near-optimal solutions were generally found early in the branch-and-bound procedure. Table 2 shows the average ratio of the initial approximate solution to the optimal value and also of the incumbent solution after 10 iterations to the optimal value, as well as the average minimum number of partial colorings explicitly examined and the average minimum queue size.

The combination of obtaining 'good' feasible solutions early in the branch-and-bound procedure, and finding fast lower bounds, however weak, allow the graph-based algorithm to outperform standard branch-and-bound by a significant margin (see Table 3). A comparison of the computational complexity of the algorithms explains this phenomenon.

The simplex algorithm used in each iteration of the standard branch-and-bound, although theoretically exponential, is in the average case no worse than $O(m^3)$ where m is the number of rows (constraints) in the model. In this case m behaves as n^2, since there are n constraints (25), one for each of the n vertices, k constraints (26), one for each of the k timeslots, and approximately $\frac{n^2kd}{2}$ where d is the density of the graph, and the simplex algorithm behaves roughly as an $O(n^6)$ algorithm.

The graph-based algorithm, on the other hand, is no worse than $O(n^2)$ at each node, where n is the number of vertices, since the most computationally demanding step is the update portion of the algorithm. It is not surprising that the graph-based branch-and-bound algorithm is well able to offset the disadvantage of its weaker lower bound. Table 3 shows a comparison of CPU seconds on the HCX-9 for three data sets involving 25 vertices and 4, 5, and 6 colors respectively. The cost vector component distribution was $P(n^2) = 0.10$, $P(n) = 0.25$, $P(1) = 0.25$, and $P(0) = 0.40$, while the edge density was 0.4 and the edge weights were uniformly distributed among the three weights. The times are not proportional due to the fact that the number

Data Set	Initial Ratio	Ratio After 10 Iterations	Mean Min # Iterations	Mean Min Queue Size
1	1.30	1.01	161.2	71.2
2	1.64	1.03	134.9	49.2
3	6.98	1.12	131.0	55.2
4	6.76	1.09	108.5	36.6
5	10.01	1.10	266.5	87.3
6	1.68	1.01	32.5	13.6
7	2.10	1.02	101.1	37.0
8	1.64	1.09	97.7	38.0
9	1.84	1.07	159.8	65.6
10	4.42	1.34	1041.1	351.9
11	1.34	1.00	18.5	8.3
12	1.73	1.05	332.5	122.6
13	2.70	1.06	668.5	289.6
14	1.88	1.22	187.4	87.7
15	3.85	1.21	385.2	147.5
16	1.38	1.07	390.0	66.3
17	3.84	1.06	53.5	205.1
18	4.28	1.25	289.2	23.4
19	6.15	1.25	144.7	108.9
20	8.48	1.31	304.7	94.1

Table 2: Relationship Between Optimal and Incumbent Solutions

Data Set	CPU Time Standard Branch-and-Bound	CPU Time Graph-Based Branch-and-Bound
2	5517.16	19.08
3	709.47	4.79
4	8318.13	14.78

Table 3: Comparison With Standard Branch-and-Bound
(CPU Time in Seconds on HCX-9)

of nodes that must be explicitly examined varies considerably.

The graph-based branch-and-bound algorithm was also applied to sets of ten randomly generated graphs on 40 and 50 vertices with 5, 6 and 7 colors, problems too large to be solved in reasonable CPU time and storage space by standard branch-and-bound. The graph-based branch-and-bound algorithm was used on these problems, branching according to $fdeg$ with $udeg$ for the tiebreaker. This parameter was chosen because of its generally good and consistent performance on the smaller, 25-vertex problems. Table 4 shows these results. The 40-vertex problems were all solved to optimality. However, finding optimal solutions for 50-vertex problems was more difficult, especially when the density of the graph was high. The results shown in the table were obtained with a maximum queue size of 100,000 and a maximum of 200,000 fixed vertices overall. (The results reported for data set 27 include only the problems for which an optimal or ϵ-optimal solution was found.) CPU time was not routinely available for these problems, but for these 40- and 50-vertex problems, approximately 3000 partial colorings were examined per CPU minute.

In addition, the graph-based branch-and-bound algorithm was tried on a 64-event, 6-timeslot problem that had run for 37 hours of CPU time on the Harris HCX-9 without improving on the value of the approximate solution. The best approximate solution had a value of 492 and had been obtained using $fdeg$ with a threshold of 64 and $tdeg$ as the tiebreaker. The linear programming relaxation lower bound for the problem is 12, and at the end of 37 hours, the lower bound obtained by standard branch-and-bound was only 57.

The graph-based branch-and-bound algorithm used the initial matroid

Data Set	# Optimal Solutions	# ϵ-Optimal Solutions	Mean # Iterations	Mean Queue Size
21	10	-	12,134.7	4,020.2
22	10	-	62,467.6	25,307.3
23	10	-	9,820.6	5,639.5
24	10	-	4,599.5	2,024.4
25	10	-	8,784.2	6,762.1
26	10	-	4,843.9	2,351.4
27	1	3	161,021.3	57,294.3
28	6	4	84,170.7	36,782.1
29	10	-	21,511.2	14,433.0

Table 4: Results on Larger Graphs

lower bound of 0, and found an initial incumbent, using the *del* criterion in the embedded approximate algorithm, with value 867. With $\epsilon = 0$, using total weighted degree as the branching criterion, the graph-based branch-and-bound algorithm ran for approximately 2 hours and yielded a lower bound of 73 and feasible solution with value 358. With $fdeg$ together with *udeg* as the tiebreaker for the branching criterion, and approximately the same run-time, a lower bound of 77 was obtained, together with a feasible solution with value 362.

7 Conclusion.

The results obtained using both the approximate coloring algorithm and the graph-based branch-and-bound indicate that the use of the weighted graph model can result in an increase in both the size and generality of the problems that can be solved to optimality within a reasonable computational time. The improbability of obtaining a performance bound for any polynomial-time approximate algorithm argues for the importance of discovering probabilistic bounds and of finding restricted problem classes for which a performance bound can be obtained. This has been done for the travelling salesman problem, for which no performance bound is likely in the general case, but for which a performance bound of $\frac{3}{2}$ has been obtained when the edge weights satisfy the triangle inequality [13].

Various extensions to the weighted graph model enable it to model the general quadratic assignment problem, the job sequencing problem, and to accommodate other complications that frequently occur in timetabling, such as overlapping timeslots, shared resources (e.g., different room types or sizes), and multiple sections. In order to evaluate the usefulness of these extensions, it is necessary to carefully examine the effect that each has on the design and performance of both the approximate and exact algorithms.

The development of the weighted graph model for the timetabling problem, and of both approximate and exact algorithms for solving the model, opens up a fertile area for additional research. The fast lower-bounding capability of the embedded matroid speeds the branch-and-bound procedure considerably, and, together with the approximate algorithm, results in a significant improvement over ordinary branch-and-bound. Indeed, the approximate algorithm solves the problem of finding feasible solutions to the integer linear program. Additional research could lead to even more impressive gains in the size and variety of problems that can be solved to optimality.

References

[1] Abramson, D., 1991. "Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms," *Management Science* 37, 98-113.

[2] Aubin, J. and J.A. Ferland, 1987. "A Large Scale Timetabling Problem," Publication 568, Universit del Montral, Dpartement d'information et de recherche oprationelle.

[3] Carlson, R.C. and G.L. Nemhauser, 1966. "Scheduling to Minimize Interaction Cost," *Operations Research* 14, 52-58.

[4] Carter, M.W., 1986. "A Survey of Practical Applications of Examination Timetabling Algorithms," *Operations Research* 34, 193-202.

[5] Ferland, J.A. and S. Roy, 1985. "Timetabling Problem for University as Assignment of Activities to Resources," *Computers and Operations Research* 12, 207-218.

[6] Johnson, D.S., C.R. Aragon, L.A. McGeoch and C. Schevon, 1989. "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning," *Operations Research* 37, 865-892.

[7] Johnson, D.S., C.R. Aragon, L.A. McGeoch and C. Schevon, 1991. "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning," *Operations Research* 39, 378-406.

[8] Karp, R.M., 1972. "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, (R.E. Miller and J.W. Thatcher, eds.), Plenum Press, New York, 85-103.

[9] Kiaer, L. and J. Yellen, 1991. "Weighted Graphs and University Course Timetabling," *Computers and Operations Research* 19, 59-67.

[10] Krarup, J. and D. deWerra, 1982, "Chromatic Optimisation: Limitations, Objectives, Uses, References", *European Journal of Operational Research* 11, 1-19.

[11] Lawler, E.L., *Combinatorial Optimization, Networks and Matroids.* Holt, Rinehart & Winston, New York, 1976.

[12] Mulvey, John M., "A classroom/time assignment model," *European Journal of Operational Research*, Vol. 9 (1982), pp. 64-70.

[13] Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D. Schmoys, *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley, Chichester, 1985.

[14] Matula, D.W., G. Marble and I.D. Isaacson, "Graph Coloring Algorithms," in *Graph Theory and Computing*, R.C. Read, ed., Academic Press, New York, 1972.

[15] Nemhauser, G.L. and L.A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, Inc., New York, 1988.