Graph Coloring for Class Scheduling

Amal Dandashi
MSc Student
Department of Computer Science,
University of Balamand
Koura, Lebanon
amal_dandashi@hotmail.com

Mayez Al-Mouhamed
Department of Computer Engineering
King Fahd University of Petroleum and Minerals
31261 Dhahran,
Kingdom of Saudi Arabia
mayez@kfupm.edu.sa

Abstract— **The class scheduling problem can be modeled by a graph where the vertices and edges represent the courses and the common students, respectively. The problem is to assign the courses a given number of time slots (colors), where each time slot can be used for a given number of class rooms. The Vertex Coloring (VC) algorithm is a polynomial time algorithm which produces a conflict free solution using the least number of colors [9]. However, the VC solution may not be implementable because it uses a number of time slots that exceed the available ones with unbalanced use of class rooms. We propose a heuristic approach VC* to (1) promote uniform distribution of courses over the colors and to (2) balance course load for each time slot over the available class rooms. The performance function represents the percentage of students in all courses that could not be mapped to time slots or to class rooms. A randomized simulation of registration of four departments with up to 1200 students is used to evaluate the performance of proposed heuristic.**

Keywords- class scheduling, graph coloring, algorithms, vertex coloring, uniform distribution.

I. INTRODUCTION

A classical problem in graph theory, the graph coloring problem is to color the nodes of an undirected graph with as few colors as possible, such that no two adjacent nodes share a color [1]. Graph coloring has many applications including map coloring, task scheduling, parallel computation, network design,etc. Here we are only concerned about the problem of course scheduling, where graph coloring can provide an algorithm which will prevent or at least minimize conflicting schedules.

There are many graph coloring algorithms which have all proved to be NP complete, such as the Saturation algorithm, the Recursive Largest First algorithm, Degree of Saturation Algorithm, Simulated Annealing algorithm, Greedy algorithm, and many others [1]. The application of computers to timetabling problems has a long and varied history. In 1967, the problem of course scheduling was applied to graph coloring. The vertices of the graph were ordered sequentially, according to degree and the graph was colored without using an upper limit on the number of colors [2]. In 1972 the problem of graph coloring proved to be NP complete [3]. In 1995, a method using graph coloring was developed for optimizing solutions to the problem of graph scheduled, and was compared to an expert system which arrived at a partial solution and then performed simulated annealing to fill out the solution. The latter system did not schedule all courses, whereas the graph coloring method was found to be quicker and handles all hard constraints of the problem [4]. In 2005, the Robust Graph Coloring (RGC) problem on paths was studied, where given a red path and a blue path, with costs, for the same set of vertices, a 3-coloring is found on the blue path that minimizes the sum of the costs of the red edges whose ends have the same color. An exponential time algorithm, a randomized algorithm, and a greedy algorithm were proposed [5]. In 2007, an alternative graph coloring method was presented for university timetabling that incorporates room assignment during the coloring process [6].

In 2008, the Koala graph coloring library was developed. It includes many practical applications of graph coloring, and is based on C++. Future directions include efficiency improvements, implementation of graph class recognition, and better graph visualization algorithms [7]. In 2009, four learning automata-based approximation algorithms were proposed for solving the minimum vertex coloring problem (being NP-hard). The algorithms find the possible colorings of the graph and depending on the response from the environment, color sets are rewarded or penalized. Then the minimum coloring of the graph with the highest probability is found. The algorithm an improved efficiency [8].

The VC algorithm is a polynomial time algorithm which produces a conflict free solution using the least number of colors [9]. As there still may be conflicts in the scheduling of courses produced by VC due to limitations in the available time slots or class rooms, a proposed heuristic VC* is proposed to achieve more uniform distribution of courses among time slots and class rooms. Uniform distribution will ensure less conflict with regard to time and space limitations. Randomized simulation of registration data is used to assess the performance of proposed heuristics.

This paper is organized as follows. In section II, basic of graph coloring are presented. In section III the VC is described. Section IV contains an adaptation of VC to course scheduling. In Section V we present the heuristic VC*. Sections VI and VII present the objective function and the evaluation, respectively. We conclude in section VII.

II. BACKGROUND

In graph theory, graph coloring is an assignment of labels (colors) to elements of a loopless graph subject to certain constraints. It is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color, called vertex

coloring. For the specific course scheduling application, a color represents one time slot, which is allocated to a number of physical rooms. A vertex represents a course, while the edges between the vetices represents the common students between the courses. A proper m-coloring of a graph (G) is an assignment of a unique color to each vertex of G such that no two adjacent vertices are assigned the same color. The smallest number of colors needed to color G is known as it's chromatic number X(G). A graph that can be assigned a proper m-coloring is called m-colorable, and it is m-chromatic if it's chromatic number is exactly m. The chromatic polynomial counts the number of ways a graph can be colored using no more than a given number of colors. A set of vertices in which all pairs of vertices share an edge make up a clique. Graphs with large cliques have high chromatic numbers but not vice-versa. Graphs with high chromatic number must have high maximum vertex degree. The degree of a vertex is the number of it's distinct edges [9].

For small problems, it does not matter which graph coloring algorithm is used as long as it solves the problem correctly. However, for many bigger problems the only known algorithms take so long to compute the solution that they are practically useless. A polynomial-time algorithm is one whose number of computational steps is always bounded by a polynomial function of the size of the input, thus it is actually useful in practice. The class of polynomial-time algorithms is denoted by P. For some problems, there are no known polynomial-time algorithms but they do have nondeterministic polynomial-time algorithms, denoted by NP. NP problems are such that any polynomial-time algorithm for them can be transformed, in polynomial-time, into a polynomial-time algorithm for every problem in NP. Such problems are called NP-complete [9]. The problem of trying to find a proper m-coloring of the vertices of a graph, for any fixed integer m greater than 2, is known to be NP-complete [3].

III. The Vertex Coloring Algorithm

VC is a polynomial-time algorithm for producing an m-chromatic solution for a graph. Dharwadker proves that every graph (G) with n vertices and maximum vertex degree Δ must have chromatic number X(G) less than or equal to $\Delta+1$ and that the algorithm will always find proper m-coloring of the vertices of G with m less than or equal to $\Delta+1$ [9].

Consider a loopless graph G, with n vertices that consists of a set of vertices V and a set of edges E, where each edge is an unordered pair of distinct vertices. The maximum degree of all vertices of G is denoted by Δ. Given graphs G and H, the Cartesian product GxH is defined as the graph whose set of vertices is V(G)xV(H) with an edge connecting vertex (u1,v1) with vertex (u2,v2), only if either u1=u2 and {v1,v2} is an edge in H or v1=v2 and {u1,u2} is an edge in G. A subset of vertices where every unordered pair of vertices is an edge is called a clique, denoted by Q. If all vertices of a graph form a clique, the graph is said to be complete and is denoted by K_m, m being the number of vertices. A set of vertices where no unordered pair of vertices is an edge is an independent set S. A maximum independent set is an independent set with the largest number of vertices. The chromatic number X(G) of a graph is the minimum value of m for which there exists a proper m-coloring of the vertices of G.

The algorithm starts with the Cartesian Lemma which allows the conversion of the problem of finding proper m-coloring of the n vertices of a graph to the logically equivalent problem of finding an independent set S of size n in the Cartesian product GxK_m. G with n vertices is m-colorable if and only if the Cartesian product GxK_m has an independent set S of size n. K_m being a clique, represents a color per vertex, for which each color will be assigned to each independent set in G [13]. For example, consider a graph G_n = {u1, u2, u3} which is m-colorable, and consider a clique K_m= {v1, v2} as shown on Figure 1.

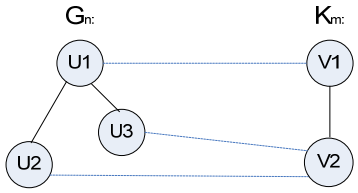

Figure 1: Graph corresponding to G_n x K_m

The algorithm first constructs the Cartesian Product GxK_m, then it searches for an independent set S of size n in the Cartesian product as shown on Figure 2.

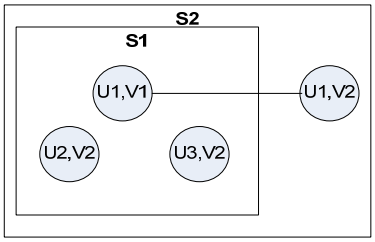

Figure 2: Subsets S1 and S2

Set S1 is an independent set: for the vertices {u1,v1} and {u2,v2}, u1 ≠ u2, v1 ≠ v2. For the vertices {u1,v1} and {u3,v2}, u1 ≠ u3, v1 ≠ v2. For the vertices {u2,v2} and {u3,v2}, u2 ≠ u3 and while v2=v2, u2 and u3 don't share an edge in G. Therefore they are properly m-colored with the assigned vertices of K_m. Set S2, on the other hand is not an independent set: for the vertices {u1, v1} and {u1, v2}, u1=u1 and v1 ≠ v2, the two vertices are dependent. In the above example, u1 is assigned colors v1, and u2 and u3 are assigned the color v2. G is properly 2-colored by the Cartesian product G_3 x K_2 [9]. This algorithm was implemented using C++.

IV. Graph Coloring: Students' Registration Code

In order to implement the VC program to the problem of student conflict-free course scheduling, we have created a program named Graph Coloring (GC), using the Java programming language, which reads in information of students' and their respective registered courses for a given semester from a text file, and outputs this information in the form of a binary graph text file, in the format required by VC.

The binary graph required as input by VC is an adjacency matrix of graph G. It is an nxn matrix with the entry in row u

and column v equal to 1 if u and v share an edge, otherwise the entry would equal 0. An example of a small (11x11) graph:

The Bondy-Murty Graph:[9]

11

0	0	1	1	1	1	0	1	1	1	1
0	0	1	1	1	1	0	1	1	1	1
1	1	0	1	0	0	1	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	1	1	0	0	0	0
1	1	0	0	1	0	1	0	0	0	0
0	0	1	1	1	1	0	1	1	1	1
1	1	0	0	0	0	1	0	1	0	0
1	1	0	0	0	0	1	1	0	0	0
1	1	0	0	0	0	1	0	0	0	1
1	1	0	0	0	0	1	0	0	1	0

This graph's coloring output is as follows:

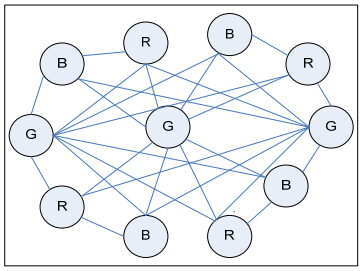

Figure 3: The Bondy-Murty Graph

V. UNIFORM DISTRIBUTION ALGORITHM

After running the VC program on the output graph file produced by the GC program, we have a result of proper coloring allocation to each course and its respective students. Each color represents one time slot with a set of rooms available in that time slot.

The next problem to address is the number of colors (N_C) allocated by the VC program with respect to the number time-slots (N_{TS}) available for use. For example, if we have a graph of 50 courses registered by a number of students, and this graph is run on the VC program, and was properly colored with 12 colors. Suppose that each color represents one time slot and an average of four available rooms in that time. So these four rooms are independent and can be run in parallel, which is why many courses share the same color.

Colors	C1	C2	C3	C4
Number of Courses	4	3	1	7
Number of available rooms	4	4	4	4

TABLE I.

The number of courses ($N_{CO}(C)$) allocated to a particular time-slot (color) may be greater than, equal or below the number of available rooms (N_{RM})for use. For example, in the above example the color C2, having 4 rooms available for use, has only 3 courses allocated, thus one room will be unused. Or the color C4, having 4 rooms available for use, has 7 courses allocated to it, thus 3 of the courses will not have a room.

Ideally, we need to favor a uniform distribution (UD) of courses with respect to colors, in order to avoid conflicts and to maximize the use of available room/time slots. The following Heuristics H-1 and H-2 were developed to favor uniform and balanced distribution of time slots and rooms.

We define a heuristic H-1 that operates on the solution provided by VC with the objective to promote uniform distribution of courses over the colors. Heuristic H-1 applies when the N_C allocated by the VC program is the maximum N_{TS} available for use, and if the $N_{CO}(C)$ allocated to a particular color C exceeds the number of available rooms N_{RM} for use in that color, ex: C4 having one time slot and 4 available rooms, has 7 courses allocated to it. Only 4 of these courses can take place due to limited rooms. The extra 3 courses cannot be assigned to any other color with an extra slot since they would create a conflict.

A formal description of heuristic H1 is as follows. H1 starts by allocating N_{TS} time slots to the top colors or group of courses (S_{Alloc}) that have the highest student contingency. S_{Alloc} is being the set of colors that are allocated time slots. Unallocated colors are included is a set S_{Nalloc}. The courses that have been colored by: (1) an allocated color but having a set of courses exceeding the number of room N_{RM} , or (2) an unallocated color, are assigned to the allocated time slot that has the least conflict edges among all allocated colors that have a number o courses below the number of available rooms N_{RM} . Intuitively, H-1 identifies the number of conflict edges (common students) between each of the 3 courses and the rest of the colors with a free room slot. Start with the largest class. Chose the color with the least conflict edges with the course, and in order to favor more uniform distribution, drop the number of students that conflict. Then assign the course to the color chosen.

We define a heuristic H-2 that operates on the solution provided by VC with the objective to balance course load for given time slots over the available classrooms. Intuitively, H-2 find all extra courses, group them into independent sets, where each set of courses are independent of each other and may be taken in the same time (same color). Sort the sets of courses in descending order according to weight. Color the highest priority set of courses with remaining new colors. If the N_{CO} allocated to a new color > the amount of available room for use in this color, repeat Heuristic 1. If the number of extra classes < the number of remaining colors, then there is no problem as all courses will have received a color. If not all extra courses received a new color (not enough colors), then go to Heuristic 1.

A formal description of heuristic H2 is as follows. Heuristic H-1 applies when the number of colors, each consists of a group of courses, is below the number of time slots ($N_C =< N_{TS}$) but some colors have a number of courses that exceeds the number of available rooms N_{RM}. For each color, the courses in excess of N_{RM} which have the lowest contingency are allocated to one of the remaining time slots ($N_C =< N_{TS}$).

Using the previously defined heuristics H-1 and H2, we proposed algorithm VC* that (1) promote the uniform distribution of courses over the time slots (colors) and (2) balance the course load for a given time slot over the available classrooms. VC* operates as follows: (1) run VC over the problem, if (NC < NTS) then apply H-2, else apply H-1.

VI. Objective Function

Here we define the objective function. VC solution uses N_C time slots (colors) and each time slot is used for $N_{CO}(C)$ courses. In the following we build two sets of courses A and B for each solution that will be used to evaluate the objective function of the solution. N is the number of students.

Set A is defined by all the colors that could not be mapped to some time slot as only N_{TS} color can be mapped to time slots. A can be constructed by sorting the colors in descending degrees D(C), i.e. number of students for all courses in a given time slot. The top N_{TS} colors are mapped to available time slots. The remaining colors fall into A. Set B is defined by all the courses for all colors that could not be assigned class rooms. For each color, the top N_{CR} courses are mapped to available class rooms. The remaining courses fall into B.
The performance function represents the percentage of dropped students (PDS) in all courses which could not be mapped to time slots or to class rooms. In other words:

$$PDS = \left(\sum_{\{C \text{ in } A \cup B\}} N_{ST}(CO: CO \text{ in } C) \right) / N$$

VII. Evalaution

To generate registration data, we used a randomized simulation to evaluate the performance of proposed algorithm VC*. For this we consider the courses from four departments, where each department has N/4 students. Each course has on the average 10 students which means there are, on the average, N/40 courses for each department. The total number of courses should not exceed the number room-time-slots ($N/40 =< \mathbf{N_{TS}} * \mathbf{N_{RM}}$) which allows finding the $\mathbf{N_{RM}}$ when $\mathbf{N_{TS}}$ is reasonably selected ($\mathbf{N_{TS}} =<10$). Each student registers 3 courses (60%) from his department and two courses (40%) from another department. The number of students N is taken as 400, 600, 800, 1000, and 1200. The student course registration follows a uniform distribution. Each of the above random generation define one instance of the registration data. For each problem instance we run VC and VC*. Each performance point P(VC) or P(VC*) in the plot shown below results from averaging 10 problem instances.

Figure 4 shows the percentage of dropped students (PDS) for (1) VC solution, (2) VC with heuristic H1, (3) VC with heuristic H-2, and (4) VC*. Results confirm that the PDS out the native VC solution can easily become unacceptable as the PDS may exceed 10% of total enrollment. Moreover, PDS quasi linearly increases with problem size. While H-2 has shown to produce a marginal improvement over VC solution, H-1 seems to have useful impact in matching the solution with the resource. Overall, VC* benefits from combining H-1 and H-2 and achieved the least PDS which is within 3.22% for the studied cases, i.e. reducing the PDS of VC by about 65%.

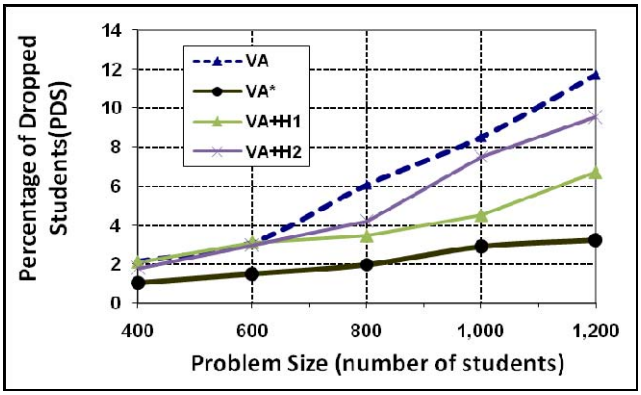

Figure 4: Percentage of dropped students for (1) VC solution, (2) VC with heuristic H1, (3) VC with heuristic H-2, and (4) VC*.

VIII. Conclusion

The Vertex Coloring (VC) algorithm produces solutions that may not be implementable Class Scheduling due to the use of a number of time slots that exceed the available ones with an unbalanced use of class rooms. A randomized simulation of registration of four departments with up to 1200 students is used to evaluate the performance of proposed heuristic. Results confirm that the percentage of dropped students PDS out the native VC solution can easily become unacceptable as the PDS may exceed 10% of total enrollment. VC* achieved the significantly lower PDS which is within 3.22% for the studied cases, i.e. reducing the PDS of VC by about 65%. Although graph coloring may not eliminate all existing conflicts, it greatly reduces them and significantly improves usage of available resources. The VC* is proposed to the Balamand University as a tool to facilitate student registration.

References

[1] W. Klotz, "Graph Coloring Algorithms," Mathematic- Bericht 5, TU Clausthal, 2002.

[2] D.J.A. Welsh, and M.B. Powell, "An Upper Bound for the Chromatic Number of a Graph and it's Application to Timetabling Problems," Comp. Jrnl. 10, 1967.

[3] R.M. Karp, "Reducibility among Combinatorial Problems In Complexity of Computer Computations," In Complexity of Computer Computations, Plenum Press, New York 1972.

[4] S.K. Miner, S. Elmohamed, and H.W. Yau, "Optimizing Timetabling Solutions using Graph Coloring," NPAC, Syracuse University, 1995.

[5] R.L. Bracho, J.R. Rodriguez, and F.J. Martinez, "Algorithms for Robust Graph Coloring on Paths," ICEEE, and CIE, Mexico, Sept 2005.

[6] T.A. Redl, "University Timetabling via Graph Coloring: An Alternative Approach," University of Houston, Houston, 2007.

[7] T. Dobrolowski, D. Dereniowski, and L. Kuszner, "Koala Graph Coloring Library: an Open Graph Coloring Library for Real World Applications," IT, Gdansk, Poland, May 2008.

[8] J.A. Torkestani, and M.R. Meybodi, "Graph Coloring Problem Based on Learning Automata," ICME, 2009.

[9] A. Dharwadker, "The Vertex Coloring Algorithm," unpublished, 2006.