A New Exam Scheduling Algorithm Using Graph Coloring

Mohammad Malkawi[1], Mohammad Al-Haj Hassan[2], and Osama Al-Haj Hassan[3]
[1]SUN Microsystems, Network Circle, USA
[2]Faculty of IT, Middle East University for Graduate Studies, Jordan
[3]Department of Computer Science, University of Georgia, USA

Abstract: *This paper presents a graph-coloring-based algorithm for the exam scheduling application, with the objective of achieving fairness, accuracy, and optimal exam time period. Through the work, we consider few assumptions and constraints, closely related to the general exam scheduling problem, and mainly driven from accumulated experience at various universities. The performance of the algorithm is also a major concern of this paper.*

Keywords: *Exam scheduling, graph algorithms, graph coloring, performance analysis.*

Received May 23, 2006; Accepted September 18, 2006

1. Introduction

An undirected graph G is an ordered pair (V, E) where V is a set of nodes and E is a set of non-directed edges between nodes. Two nodes are said to be adjacent if there is an edge between them. The graph coloring is a well-known problem [1, 2, 3, 4, 5]. Node coloring assigns colors to the nodes of the graph such that no two adjacent nodes have the same color. Edge coloring assigns colors to the edges of the graph such that no two adjacent edges have the same color. Two edges are said to be adjacent if they both share a node in common. General graph coloring algorithms are well known and have been extensively studied by researchers [1, 2, 5, 7, 8, 9, 11, 12, 13, 14, 16].

Exam scheduling is a challenging task that universities and colleges face several times every year. The challenge is to schedule so many exams of courses in a limited, and usually short, period of time. An Exam schedule should avoid conflicts, in the sense that no two or more exams for the same student are scheduled at the same time. Part of the challenge is to achieve fairness for the students. A fair schedule does not schedule more than two exams, for example for a student on one day. In the meantime, a fair schedule does not leave a big gap between exams for the students. The exam scheduling problem is defined as follows: *"We first represent the courses by nodes of a graph, where 2 nodes are adjacent if the 2 corresponding courses are registered by at least one student. Then, it is required to assign each course represented by a node a time slot, such that no two adjacent nodes have the same slot, in condition that a set of constraints imposed on the problem are also met."* We solve this problem by using node graph coloring technique.

This study provides a mechanism for automatic exam-schedule generation that achieves fairness, and minimizes the exam period. As a result, this paper presents a graph-coloring-based algorithm for the exam scheduling application which achieves the objectives of fairness, accuracy, and optimal exam time period. Numerous studies have considered the problem of exam scheduling [9, 10, 15, 17]. The main difference between various studies is the set of assumptions and constraints taken into consideration. Burke, Elliman and Weare [9], for example, followed a similar approach using graph coloring. However, in their algorithm, they addressed only the conflicts without any constraints. Moreover, the algorithm presented in [9] does not eliminate conflicts, and only aims at minimizing conflicts. In this paper, we consider few but important assumptions and constraints, closely related to the general exam scheduling, and mainly driven from the real life requirements collected through the experience at various universities. Such assumptions and constraints are distinct from those present in more general graph coloring problems. We summarize the main assumptions and constraints as follows:

1. The number of exam periods per day (Time Slots (TS)) can be set by the user. TS depend on college/department specific constraints. For example, a university that uses a 2-hours exam period and begins the exam day at 8:00 am and finish at 8:00 pm, may set TS to 5.
2. The number of concurrent exam sessions or concurrency level (N_p) depends on the number of available halls, and the availability of faculty to conduct the exams. N_p is determined by the registrar's office. This paper assumes that N_p is a system parameter and the scheduling algorithm has been examined with several N_p values.

3. A student shall not have more than (y) exams per day (fairness requirement), and is treated as a system tunable parameter.
4. A student shall not have a gap of more than (x) days between two successive exams, and this factor is to be determined by the college or department (another fairness requirement).
5. The schedule shall be done in the minimal possible period of time, i.e., minimize the number of exam slots and/or number of exam days. The exam time period is an outcome of the scheduling algorithm.
6. Next, we give some more definitions that are relevant to the underlined problem. Let C be a list of all courses to be scheduled. The length of this list is n. In other words, n is the number of courses in the list. *A* course at position i in the list C is referred to using an index c_i. Let G be the graph that represents the list C of courses. We impose a weight w_{ij} to each edge of G, where w_{ij} is defined as the number of students present in both courses c_i and c_j. An edge e_{ij} exists between nodes c_i and c_j iff w_{ij} is not 0. We define a weight matrix W to be an nxn matrix, where n is the number of courses to be scheduled for the exams, and w_{ij} equals the weight of the edge e_{ij} that joins the courses c_i and c_j. Such a weight imposed on the edges of G represents the exam conflict complexity present in courses c_i and c_j. A multi-section course is considered as one course. However, the number of sections per course is taken into consideration in the process of hall assignment.

The degree d_i of a node c_i is defined as the number of edges connected to a node. A large degree of a node c_i indicates that there is a large number of students registered in this course and d_i other courses. The degree d_i is also a measure of conflict complexity. An example of a weighted graph G and the corresponding weight matrix W is given in Figure 1 and Table 1, respectively. In Figure 1, c_2 and c_5 both have degree 3. In Table 1, the weight of the edge e_{15} is 4.

2. The Coloring Scheme

The coloring scheme for the exam-scheduling problem uses a double indexed color (R_{IJ}), where the index (I) represents the day of the exam and (J) represents the exam time slot on a given day. The range of (J), i.e., the number of exam time slots is determined by the registrar and/or the faculty.

The range of the index (I) is a parameter generated as an outcome by the algorithm. Minimizing the index (I) is one objective of the algorithm. The parameter I can also be set by the registrar and/or the faculty. It is bound by the absolute minimal number of colors for the given graph. However, finding the absolute minimal is known to be NP complete. The algorithm presented in this paper is claimed to achieve near optimal performance (close to minimal number of colors) in polynomial time.

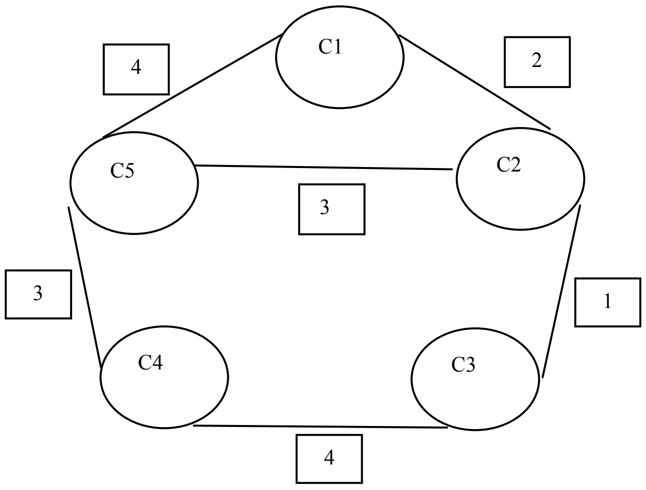

Figure 1. A weighted graph G.

Table 1. A weight matrix W of the graph.

	C₁	C₂	C₃	C₄	C₅
C₁	0	2	0	0	4
C₂	2	0	1	0	3
C₃	0	1	0	4	0
C₄	0	0	4	0	3
C₅	4	3	0	3	0

We define the weight of a color to be $W(R_{IJ}) = (I-1)*k + J$; k is the range of J. A color R_{IJ} is said to be smaller than color R_{GH} if the weight $W(R_{IJ})$ is smaller than $W(R_{GH})$. The coloring scheme allows two or more non-adjacent nodes to have the same color (R_{IJ}). The number of nodes having the same color provides the number of concurrent exam sessions, which is bounded by the number of available halls and the maximum allowable concurrent sessions by the registrar and/or the faculty. In general graph coloring problems, there is no restriction on the assignment of the same color to non-adjacent nodes in the graph. The exam-scheduling problem as explained above imposes a constraint on the maximum number of nodes assigned the same color. The scheduling algorithm (provided in the next section) allows the user to impose a maximum limit on the number of available instances of color R_{IJ}. The number of instances of a color R_{IJ} is referred to as the concurrency limit of the color R_{IJ} denoted $CL(R_{IJ})$. Note that a course with multiple sections is assigned one color. However, the multiple sections will consume multiple instances of the same color, assuming that each section will make the exam in a separate hall.

2.1. Fairness of the Algorithm

In order to achieve fairness, as discussed in the introduction, the algorithm defines the following parameters:

1. Internal distance (D_I): This is the distance between two colors (R_{IJ}) and (R_{IK}) with the same index (I) and indexes J and K, and defined by

$$D_I = |K-J| \qquad (1)$$

D_1 represents the exam scattering on the same day I for the same set of students.

2. External distance (D_2): This is the distance between two colors (R_{IJ}) and (R_{KL}), and defined by

$$D_2 = |K-I| \qquad (2)$$

D_2 represents the exam scattering across different days.

3. The total distance between colors (R_{IJ}) and (R_{KG}) is given by

$$D = \gamma * D_2 + D_I, \qquad (3)$$

or

$$D = \gamma *(|K-I|) + |G-J| \qquad (4)$$

The factor (γ) can be varied to provide a different coloring scheme. The distance D is a major design parameter of the algorithm.

2.2. Specific Considerations

The scheduling problem has its own peculiarities, which have to be taken into consideration at the implementation level. For example, the node with a large degree represents a course in which many students are registered to many other courses (different group of students may be registered to different courses). Also, nodes with large degrees have large number of students as well. In order to have an efficient schedule; the nodes with larger degrees should be colored first. Giving priority to the nodes with the larger degrees is in line with typical university schedules which tend to schedule the university required courses early in the exam period. The nodes representing university and college requirement courses have large degrees.

The weight of an edge indicates the number of common students registered at both courses (nodes) connected to that edge. Giving priority in the coloring algorithm to nodes connected to a large weight-edge will enable a solution optimization geared towards the larger groups of students. Another point to consider before we describe the algorithm is the multi-section courses. Multi-sections of a multi-sections course should be scheduled at the same time, and thus the corresponding nodes should have one color. Also, they typically occupy several halls. The number of halls used by a course has an impact on the concurrency level per time slot. When such multi-sections are scheduled for a time slot, i.e. assigned a color, the concurrency level is to be reduced by the number of sections for that course. For implementation purposes, we augment the nodes of the graph with a value equal to the number of sections in the course; we shall call this value the course concurrency level *CL* (c_i). Thus, we assign a concurrency limit for each color N_p (C_{IJ}). After assigning a color to a node C_i, we reduce the concurrency limit of the color by *CL* (c_i). The concurrency limit is set by the registrar and depends on the number of available halls, and staff to monitor the exams.

3. Algorithm Color Schedule

The algorithm consists of two major steps. The first step builds the weight matrix and graph. The second step assigns colors to the nodes of the graph.

3.1. The Algorithm

A. *Build Weight Matrix and Graph*

1. *Locate the files for students' listings in all the courses, which need to be scheduled for the exam. Each course corresponds to a node in the matrix. Set the concurrency level of each node to the number of sections for the given course.*
2. *For each node (course) find the set of adjacent nodes, and the weight of the edges connecting the node to its adjacent nodes. Fill the weight matrix W with weight values w.*
3. *Create an undirected graph using the weight matrix.*
4. *Find the degree for each node.*

B. *Color the Graph*

```
Sort the nodes in the weight matrix in a descending
order based on the degree of nodes. Nodes with
similar degrees are ordered based on the largest
weight w in its adjacency list. Nodes with similar
degrees d and weights w are ordered based on their
node ID (smallest ID first).
Set C = The sorted list of nodes mentioned
in Step 1.
Set No-Of-Colored-Courses = 0
for i = 1 to C-length do
  Begin
    If No-Of-Colored-Courses = No-Of-
    Courses then exit loop and finish
    If $c_i$ is not colored then
      Begin
        If i = 1 then
        Begin
          $R_{ab}$ = get-First-Node-Color ($c_i$)
          If $R_{ab}$ = null then Exit and finish,
          {No schedule is possible.}
        End
        Else
        Begin
          $R_{ab}$ = get-Smallest-Available-Color ($c_i$)
        End
    If $R_{ab}$ != null then
      Begin
        Set Color ($c_i$) = $R_{ab}$
          No-Of-Colored-Courses = No-Of-
        Colored-Courses + 1
        CL ($R_{ab}$) = CL ($R_{ab}$) - CL ($c_i$)
      End
    End
    Set Array M = get-Ordered-Adjacency-
      Courses-Of-$c_i$ ()
    For j = 1 to No-Of-Courses-In-Array-M do
      Begin
        If $M_j$ is not colored then
          Begin
            $R_{cd}$ = get-Smallest-Available-Color ($M_j$)
            If Color$_{cd}$ != null then
              Begin
                Set Color ($M_j$) = $R_{cd}$
                No-Of-Colored-Courses = No-Of-
                Colored-Courses + 1
                CL ($R_{cd}$) = CL ($R_{cd}$) - CL ($M_j$)
              End
          End
      End
  End
```

C. Color the neighbor

1. Description of Sub-routine "get-First-Node-Color":

```
Input  : The course $c_i$ that needs to be
Colored.
Output: The color assigned to $c_i$ or null.
Algorithm:
For j = 1 to Max-Schedule-Days do:
  For k = 1 to No-Of-Time-Slots do:
    If CL (Color$_{jkj}$) ≥ CL ($c_i$) then return Color$_{jk}$
    return null
```

2. Description of Sub-routine "get-Smallest-Available-Color":

```
Input:  The course $c_i$ that needs to be
colored.
Output: The color assigned to $c_i$ or null.
Algorithm:
get AL($c_i$), the Adjacency-List of $c_i$
For j = 1 to Max-Schedule-Days do
  Begin
    For k = 1 to No-Of-Time-Slots do:
      Begin
        Set valid = true
        For r = 1 to Length (AL ($c_i$)) do
          Begin
            $R_{ef}$ = Color ($AL_r$)
            If $R_{ef!}$ = null then
              Begin
                If e! =j or f! =k then
                  Begin
                    If $D_2$ {($R_{ef}$), ($R_{jk}$)} = 0 then
                      Begin
                        If $D_1$ {($R_{ef}$), ($R_{jk}$)} <= 1 then
                          Begin
                            Valid = false
                            Exit loop
                          End
                      End
                    If CL ($R_{jk}$) <= CL ($c_i$) then
                      Begin
                        Valid = false
                        Exit loop
                      End
                    If Check-3Exams-Constraint ($c_i$, $R_{jk}$ , j) =
                    False then
                      Begin
                        Valid = false
                        Exit loop
                      End
                  End
                Else
                  Begin
                    Valid = false
                    Exit loop
                  End
              End
            Else Exit the current iteration of loop
          End
        If valid = true then Return $R_{jk}$
      End
  End
return null
```

3. Description of Sub-routine "Check-3Exams-Constraint":

```
Input  : The course $c_i$ that needs to be
colored.
The color $R_{jk}$ that needs to be tested.
The day j for Color$_{kd}$
Output: returns true if color is valid,
Otherwise it returns false
Algorithm:
get a list of students Si registered in course
$c_i$
For r = 1 to Length (Si) do:
  Begin
    Set Counter = 0
    For q=1 to No-Of-Time-Slots do:
      Begin
        Get a list of courses CRS assigned to $R_{jq}$
        For u = 1 to Length (CRS) do:
          Begin
            Get a list of students $S_u$ registered in
            course $c_u$
            If $Si_r$ exists in list $S_u$ then
              Begin
                Counter = Counter + 1
                If Counter = 2 then return false
              End
          End
      End
  End
return true
```

3.2. Complexity Analysis

A. Assume the largest degree $d = d_1$; and that node v_1 has degree K_1

A.1. The first step assigns the smallest color, say c_1 to node v_1. The total number of steps required to color all the nodes in the neighbor list of v_1 is

$$1+2+3+ \dots + d_1 = (d_1^2+ d_1)/2 = O(d_1^2) \quad (5)$$

A.2. Repeat the coloring procedure for the next node v_2 with degree d_2. The number of steps required to color all the nodes adjacent to node v_2 is

$$1+2+3+ \dots + d_2 = (d_2^2+ d_2)/2 = O(d_2^2) \quad (6)$$

B. In general, the number of steps required to color all the nodes in the neighbor list of any node v_i with degree d_i is

$$(d_i^2+ d_i)/2 = O(d_i^2) \quad (7)$$

C. Let the average degree of nodes be ρ. Then the average number of steps required to color the neighbors of node v_i with degree ρ is $O(\rho^2)$

C.1. Repeat the coloring procedure in steps 1 and 2 until all nodes are colored.

C.2. Since each coloring step colors on the average ρ nodes, the coloring procedure will be repeated on the average (n/ρ), where n is the number of nodes.

C.3. The total number of coloring steps required to color all nodes, on the average is

$$O((n/\rho).(p\ 2) = O(n.\rho) \quad (8)$$

The complexity equation (above) can be expressed as

$$\sum_{i=1}^{n} \rho \text{, where } \rho = (\sum_{i=1}^{n} d_i)/n. \quad (9)$$

3.3. Algorithm Efficiency Analysis

Our algorithm has a linear complexity, except when *(ρ = n-1)* and hence a polynomial solution of the second degree. We prove the following:

Lemma: The algorithm described above achieves minimal number of colors, when the upper bound of colors is given by the clique (largest completely connected sub-graph).

Proof: A completely connected graph with size *K* requires *K+1* color. The algorithm detects the clique in the graph. The algorithm also detects the clique related to each node in the graph starting from the node with the largest degree. Then, the algorithm colors the largest completely connected sub-graphs first, thus utilizing the minimal available colors to color the sub-graphs. For each node, the algorithm will not use more colors than those required by the largest completely connected sub-graph. Thus the largest number of colors used by the algorithm is only that required by the largest sub-graph, which is the absolute minimal possible number of colors.

4. Performance Analysis

The algorithm Color Schedule was applied to a course list of a university. The number of courses in the test bed is 546 with an average of 2 sections per course, for a total of 1092 exam sessions to schedule. The graph produced for the courses has an avergae degree of 54 and a maximum degree of 434. The coloring algorithm completed in 90 seconds (almost the same for all runs). We ran the algorithm with different paramters. The variables are the number of exam slots per day (3, 4, 5, 6, 7). The concurrency limit is varied between 10 and 100. The constraint is that a student will not have more than 2 exams per day. The results for the varius runs of the algorithm are plotted in Figure 2 below. The registrar office can use the plots to decide on the number of days and number of exam sessions per day for the schedule. For eample, with 7 exam slots per day, the exam period can be copleted in 12 days with 50 sessions per day. Note that the registrar office can produce several schedules in a short period of time (90 seconds per schedule) and select the appropriate schedule. Figure 3 shows the time analysis performance of the algorithm. Note that the execution time is a linear function of the number of courses. The average degree of the graph is also shown in the figure. The avergae degree does not increase at the same rate as the number of courses. This is typical of university courses. Furthermore, we have tested our algorithm against 5 samples of the 13 Toronto data sets collected from 13 institutions. In this test, we took two factors into account, namely the number of slots and the penalty. With respect to the former factor, their results slightly outperform ours. With respect to the later factor, close results are obtained, where our algorithm have beated in some sets, and has been beated in some other sets. The results are shown in Table 2, and plotted in Figures 4 and 5, respectively. Still some comments are in order. In Toronto case, there was nothing mentioned about the maximum possible concurrent exams per time slot, which means that they did not impose a constraint on that issue. In our case we did. Also, in Toronto case, there is nothing mentioned about the number of days, they only use number of time slots. So, when we run to compare with respect to this factor, we have counted the number of time slots used in all days of our algorithms to get the total number of time slots used by the schedule.

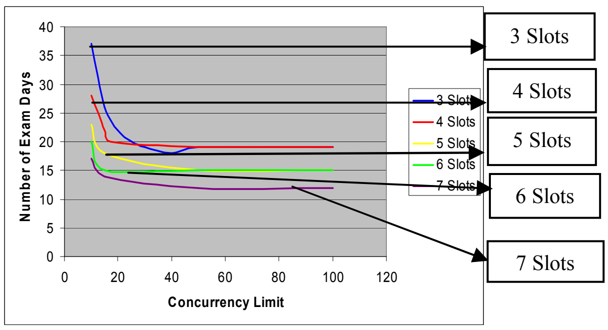

Figure 2. Algorithm color_schedule performance.

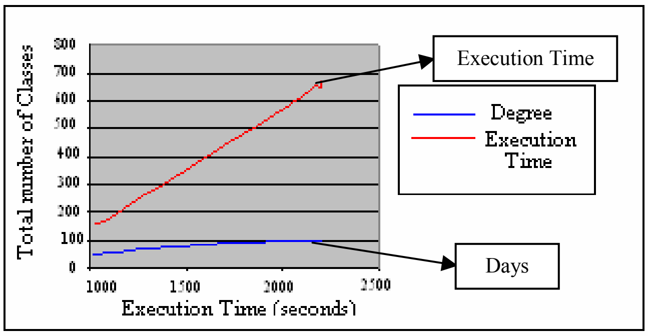

Figure 3. Execution time performance of the color schedule algorithm.

Table 2. Results of our algorithm against toronto results.

Benchmark Data	Toronto Results		Our Algorithm	
	Slots	Penalty	Slots	Penalty
CAR91	35	4.42	61	5.00059
CAR92	32	3.74	56	3.90447
KFU93	20	12.96	32	14.208
TRE92	23	7.75	42	6.41089
YOR83	21	34.84	38	23.203

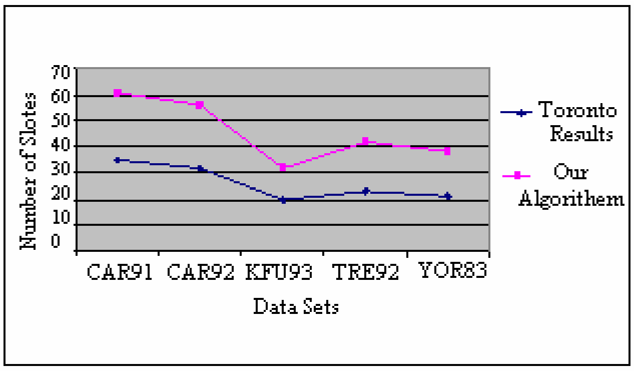

Figure 4. Results with respect to the number of slots.

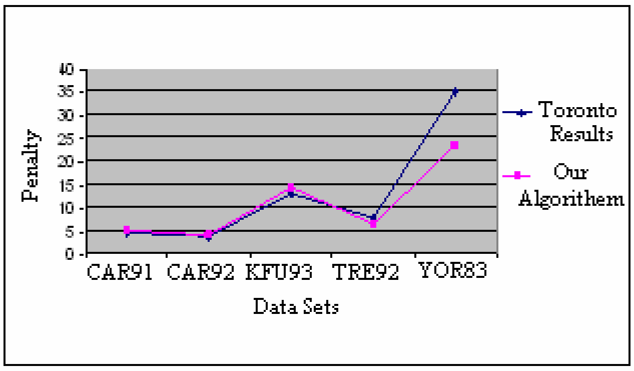

Figure 5. Results with respect to penalty.

5. Conclusion and Future Work

As discussed above, the number of concurrent exam sessions or concurrency level (N_p) depends on the number of available halls, and the availability of faculty to conduct the exams. The value of N_p is usually determined by the registrar's office, and the paper assumes that N_p is a system parameter, and we will run the scheduling algorithm with several N_p values. In a later work, the actual distribution of exam sessions to halls will be included. Also, the algorithm presented in this paper is claimed to achieve near optimal performance (close to minimal number of colors) in polynomial time. We are currently investigating a modification of the algorithm, which will achieve the absolute minimal for a certain set of graphs.

References

[1] Alon N., "A Note on Graph Colorings and Graph Polynomials," *Journal of Combinatorial Theory Series B*, vol. 70, no. 1, pp. 197-201, 1997.

[2] Baldi P., "On a Generalized Family of Colorings," *Graphs and Combinatorics*, vol. 6, no. 2, 1990.

[3] Bang-Jensen J. and Gutin G., *Digraphs: Theory, Algorithms and Applications*, Springer-Verlag, 2000.

[4] Bar-Noy A., Motwani R., and Naor J., "The Greedy Algorithm is Optimal for On-line Edge Coloring," *Information Processing Letters*, vol. 44, no. 5, pp. 251-253, 1992.

[5] Batenburg K. and Palenstijn W., "A New Exam Timetabling Algorithm," Leiden Institute of Advanced Computer Science (LIACS), http://visielab.ua.ac.be/staff/batenburg/papers/bapa_bnaic_2003.pdf.

[6] Bauernoppel F. and Jung H., "Fast Parallel Vertex Coloring," *in L. Budach (eds.), Fundamentals of Computation Theory, FCT '85*, Cottbus, GDR, Sept, 1985, vol. 199 of Lecture Notes in Computer Science, pp. 28-35, Springer-Verlag, Berlin, 1985.

[7] Bean D., "Effective Coloration," *The Journal of Symbolic Logic*, vol. 41, no.2, pp. 469-480, June 1976.

[8] Burke E. and Petrovic S., "Recent Research Directions in Automated Timetabling," *European Journal of Operational Research (EJOR)*, vol. 140, no. 2, pp 266-280, 2002.

[9] Burke E., Elliman D., and Weare R., "A University Timetabling System Based on Graph Coloring and Constraint Manipulation," *Journal of Research on Computing in Education*, vol. 27, no. 1, pp. 1-18, 1994.

[10] Burke E., Elliman D., and Weare R., "Automated Scheduling of University Exams," Department of Computer Science, University of Nottingham, 1993.

[11] Christofides N., *Graph Theory: An Algorithmic Approach*, Academic Press, 1975.

[12] Gross J. and Yellen J., *Handbook of Graph Theory*, Discrete Mathematics and its Applications, CRC Press, vol. 25, 2003.

[13] Gross J. and Yellen J., *Graph Theory and its Applications*, 2nd ed., CRC Press, 2005.

[14] Husseini S., Malkawi M., and Vairavan K., "Analysis of a Graph Coloring Based Distributed Load Balancing Algorithm," *Journal of Parallel & Distributed Systems*, vol. 10, no. 2, pp. 160-166, 1990.
[15] Husseini S., Malkawi M., and Vairavan K., "Distributed Algorithms for Edge Coloring of Graphs," *in the 5th ISMM International Conference on Parallel and Distributed Computing Systems*, 1992.
[16] Husseini S., Malkawi M., and Vairavan K., "Graph Coloring Based Distributed Load Balancing Algorithm and Its Performance Evaluation," *in the 4th Annual Symposium on Parallel Processing*, 1990.
[17] Jensen T. and Toft B., *Graph Coloring Problems*, Wiley-Interscience, 1995.

Mohammad Malkawi received his PhD degree from the University of Illinois at Urbana-Champaign in computer engineering in 1986. He received his MSc in electrical and computer engineering from Yarmouk University in 1983 and his BSc degree in computer engineering from Tashkent Polytechnic Institute in 1980. Currently, Malkawi is a senior staff engineer at SUN Microsystems, USA. He is working on the DARPA sponsored project on "High Productivity Computing Systems". His research interests include reliable and high availability computing systems, distributed and parallel high performance systems, memory architecture, wireless communication, and graph theory.

Mohammad Al-Haj Hassan obtained his BSc and MSc from The University of Jordan in 1973 and 1977, respectively, and his PhD degree in computer science from Clarkson University at NY, USA. His main field of specialization is computer algorithms with specific concentration on graph algorithms & their applications. Other fields of research are: machine learning, parallel computations and algorithms, distance learning and web-based courses design. He taught many computer courses in both undergraduate and postgraduate levels. In addition, he was the dean of several faculties in several universities for 11 years. He has enrolled in lot of other activities such as: software evaluation, chairing and/or member of a variety of university committees, implementing many university and social services, computer lab construction. He also supervised 3 master thesis, an examiner of 4 master thesis, attended more than 15 conferences, most of them as speaker, published 16 papers, an author of 4 books, and involved in accreditation issues for more than 14 years and has been involved in quality assurance issues through some activities of the Quality Assurance Agency (QAA) of United Kingdom. He is a member in the editorial board and/or referee of several journals. He has been the secretary general of the International Arab Conference on Information Technology (ACIT) for 4 years. Since September 2005, he is a professor in computer science and the vice president at the University of Graduate Studies (UGS), Jordan.

Osama Al-Haj Hassan obtained his BSc degree in computer science from Princess Summaya University of Technology in 2002, his MSc from New York Institute of Technology, and he is currently a PhD student at the University of Georgia, USA. His research interests include peer-to-peer networks, and applications of graph algorithms. He taught introduction to computer science and java courses. He developed websites and applications for the universities in which he was working.