GRAPH COLORING FOR EFFICIENT COURSE AND EXAM SCHEDULING USING MATLAB

Evi Papaioannou[1]*, Stavros Athanassopoulos[1], Christos Kaklamanis[1]

[1]University of Patras and CTI "Diophantus", GREECE, {papaioan,athanaso,kakl}@ceid.upatras.gr
*Corresponding Author

Abstract

Graphs are discrete structures composed of vertices and edges connecting these vertices. Graphs are used in almost all disciplines as abstract models for the representation and study of a wide range of relations and processes in physical, biological, social and information systems. Many practical problems in a variety of areas like computer and communication networks, social networks, transportation networks, cellular networks, linguistics, chemistry, physics, biology can be represented and studied by graphs.

Real-world entities - like molecules, persons, groups, roles, species, computing and communication devices, terms - correspond to vertices. Relations among such entities - like preference, domination, independence, interference, proximity, constraints - imply the existence of edges between corresponding vertices. Thus, focusing on the abstract graph model instead of studying each particular instance as a different real-world problem reveals common underlying properties, deficiencies and principles. In this way, efficient approaches to real-world problems emerge from the theoretical study of their abstractions.

In this work, we use graph coloring to propose efficient solutions to scheduling problems arising in higher education. The objective of the graph coloring problem is to assign colors to graph vertices so that adjacent vertices, i.e., vertices connected by an edge, receive different colors. We consider as the objective of scheduling problems in higher education, like lecture and exam scheduling, to assign time/day slots to teaching or examination activities so that the maximum number of students can attend them with the fewest possible conflicts.

Our main motivation has been the crucial issue of efficient course and exam schedules often arising in departments of the University of Patras, Greece. Students usually have to attend lectures or exams scheduled in overlapping or simultaneous time slots. However, course and exam schedules are created based on heuristic approaches which may work well on average but certainly leave several room for improvement.

What if a graph-theoretic approach were used? Courses correspond to vertices of a graph and there is an edge between two vertices if and only if an appropriately selected minimum population of students attends corresponding courses (lectures/exams). Then, a coloring of such an underlying graph suggests an appropriate schedule for teaching/examination activities.

Using a simple coloring algorithm and the MATLAB programming environment, we have designed and developed a scheduling application which receives as input courses and constraints and outputs an efficient lecture/examination schedule. Experimental evaluation suggests that our application works well in practice.

Ongoing work focuses on the use of a more involved coloring algorithm for addressing more complex course scheduling instances while minimizing required time resources.

Keywords: graph coloring, scheduling, higher education

1 INTRODUCTION

Our everyday life is built upon situations involving entities joined together by some sort of relation. For example, pairs of people are joined by some type of relationship, colleagues work together on projects, cities are joined by highways, metro/train stations are joined by railway lines, data centers are joined via communication links, courses are scheduled in weekly timetables and so on. Furthermore, relations among entities are also observed in more specialized contexts, like for example bonds among atoms and elements in chemistry, bonds among RNA chains and proteins in genetics, competition among species or migration paths in biology, interconnection of electronic devices in computer networks, association of individuals and groups within social structures and many more (Bondy, Murty, 1982, Xu, 2003).

Focusing on the fact that entities are related no matter what the particular type of relation is leads to a natural abstraction for representing these relations. We draw a diagram composed of points and lines where points represent entities and a line between two points indicates some sort of relation between the corresponding entities. In a formal mathematical context, these diagram structures are called graphs. Typically, a graph, G, is defined as an ordered pair (V,E), where V is a nonempty set of vertices, i.e., points, and E is a set of edges, i.e., lines, joining together vertices of V according to some incidence function f which associates each edge of E with an unsorted pair of not necessarily distinct vertices of V. Graphs are widely used as models in almost all scientific fields, both theoretical and applied. Graph theory, i.e., the study of structural properties of graphs, has emerged to a branch of mathematics providing deep understanding and helpful insight not only for scientific research questions but also for a wide range of real-world problems.

There exists a long list of graph problems whose study has either revealed important limitations and constraints or has suggested efficient solutions to many real-world equivalents (Roberts, 1978, Bondy, Murty, 1982, Xu, 2003). Real-world problems like for example phasing traffic lights, one-way street assignments, frequency spectrum management in wireless cellular networks, efficient communication, transportation, construction network design are immediately related to graph problems like shortest path, Eulerian chains and paths, independent set, minimum spanning tree etc. In this way, algorithms and methods devised to address graph-theoretical problems can also be exploited to provide solutions to their real counterparts.

A controversial task we frequently come across in an academic environment is the scheduling of lectures and exams under constraints related to student participation. Students usually have to attend lectures or exams scheduled in overlapping or simultaneous time slots. Despite the fact that any type of scheduling is an inherently hard problem, it has been observed that proposed schedules are rather based on heuristic approaches and usually fail to facilitate and encourage student participation. Motivated by this situation, in this work, we exploit the interplay between two important and well-studied graph-theoretical problems, namely graph coloring and scheduling, and present a MATLAB application which computes efficient lecture and examination timetables for higher education departments.

In particular, in our setting, vertices of the graph correspond to courses and there exists an edge between two vertices as long as corresponding courses cannot be scheduled for the same or overlapping time intervals due to student participation constraints. If we imagine different time intervals as different colors, then our course scheduling problem becomes equivalent to a graph (vertex) coloring problem in the underlying graph. In short, the objective of the vertex coloring problems is to color the vertices of a graph so that adjacent vertices, i.e., vertices joined by an edge, are assigned different colors. Therefore, a solution to this instance of vertex coloring forms a solution to our course scheduling problem. Following this approach we start from a list of courses and a list of participation constraints and we first create a graph G. Then, using a simple coloring algorithm we obtain a proper coloring of G, which we subsequently transform into a course schedule. Our experimental evaluation so far shows that even a simple coloring algorithm generates course schedules which are efficient in practice.

The rest of the paper is structured as follows: in Section 2 we present and analyze the coloring algorithm used for addressing our scheduling instance. In Section 3, we provide design and implementation details

concerning our application for lecture and exam scheduling. We conclude and present future plans in Section 4.

2 GRAPH COLORING ALGORITHM FOR SCHEDULING

2.1 Graph coloring

A coloring – or vertex coloring - of a simple graph is the assignment of a color to each graph vertex so that adjacent vertices receive different colors. Such a coloring is also called a proper coloring of the graph. A graph can be colored by assigning each of its vertices a different color. The corresponding optimization problem asks to find a vertex coloring with minimum number of colors.

For most graphs a coloring can be found that uses fewer colors than the number of vertices in the graph. The chromatic number of G, denoted by $\chi(G)$, is the minimum number of colors required to color all the vertices of G. In the left part of Fig. 1 a properly colored hexagon graph is depicted. Indeed, all neighboring vertices are assigned different colors. In the right part of Fig. 1, a coloring of this graph with the minimun number of colors is shown. Hexagon graphs are 3-colorable and therefore their chromatic number is 3.

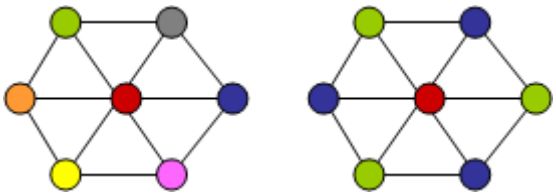

Fig. 1: Colorings of a hexagon graph, H.

Assigning distinct colors to distinct vertices always yields a proper coloring, so for every graph G=(V,E), with |V|=n, it holds that $1 \leq \chi(G) \leq n$. The degree of a vertex $v \in V$ is the number of edges of G incident to v. The chromatic number of a graph is closely related to its maximum degree, denoted $\Delta(G)$, which is the maximum degree of all vertices in a given graph. In particular, the chromatic number of a graph is at most the maximum vertex degree, i.e., $\chi(G) \leq \Delta(G)$, unless the graph is complete or an odd cycle, in which case $\Delta(G)+1$ colors are required (Brooks, 1941, Lovász, 1975).

Furthermore, it holds that $K(G) \leq \chi(G)$, where K(G) is the size of the maximum clique in G, i.e., the size of the maximum set of mutually adjacent vertices in G. For example, in the hexagon graph H of Fig. 1, the maximum clique is a triangle, i.e., it is composed of 3 mutually adjacent vertices e.g., the middle red vertex, the upper-left green vertex and the upper-right blue vertex as depicted in Fig. 2. So, K(H) = 3 and, therefore, $\chi(H) \geq 3$, which implies that H can be colored with at least 3 colors. As another example, in the complete bipartite graph $K_{3,2}$ the size of the maximum clique, is $K(K_{3,2}) = 2$ which implies a lower bound of 2 for its chromatic number (see right part of Fig. 2).

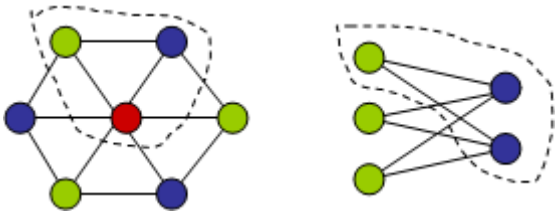

Fig. 2: Maximum cliques in a hexagon graph H (left) and the complete bipartite graph $K_{3,2}$; their size imply a lower bound on the chromatic number of these graphs.

Another crucial number bounding the chromatic number of a graph G is the number of maximum independent sets, denoted as I(G), in which vertices of G can be partitioned. An independent set in a graph G=(V,E) is a collection of mutually non-adjacent vertices of V. It can be easily seen that the notion of an independent set is complementary to the notion of a clique; vertices in an independent set are mutually non-adjacent while vertices in a clique are mutually adjacent. Thus, vertices of an independent set in a graph can be assigned the same color in a proper coloring if this graph. Therefore, it holds that $I(G) \leq \chi(G)$, where the quantity I(G) denotes the number of the maximum independent sets in G.

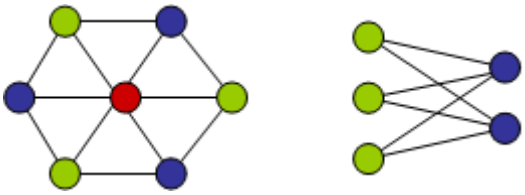

Fig. 3: The number of maximum independent sets in a hexagon graph H (left) and the complete bipartite graph K3,2 (right) implies a lower bound on chromatic number of these graphs.

For example, as depicted in Fig. 3, vertices of the hexagon graph H can be partitioned into 3 independent sets which define 3 color classes containing mutually non-adjacent vertices assigned the same color, i.e., green, blue and red. So, $I(H) = 3$ and, therefore, $\chi(H) \geq 3$, which implies that H can be colored with at least 3 colors. As another example, vertices of the complete bipartite graph $K_{3,2}$ can be partitioned into 2 independent sets. Therefore, $I(K_{3,2}) = 2$ which implies a lower bound of 2 for its chromatic number (see right part of Fig. 3).

However, determining the chromatic number of a graph is a hard problem (an NP-Complete problem) in the sense that the best available approach is to find all possible colorings and then select one which uses the smallest number of colors (Garey and Johnson, 1979, Karp, 1972). The best known algorithms for finding the chromatic number of a graph have exponential worst-case time complexity in the number of vertices of the graph. Even the problem of finding an approximation to the chromatic number of a graph is difficult. It has been shown that if there were an algorithm with polynomial worst-case time complexity that could approximate the chromatic number of a graph up to a factor of 2, then there would also exist an algorithm with polynomial worst-case time complexity for finding the chromatic number of the graph (Pardalos, Mavridou, Xue, 1998, Feige, Kilian, 998, Håstad, 1999, Khot, 2001, Zuckerman, 2007, Lewis, 2015).

Graph coloring has been used to model and address several real world problems. In the following, we indicatively mention a short list of major such problems. Map coloring refers to the coloring of geographical maps so that adjacent regions receive different colors. It has been proved that four colors are sufficient to color any map (Kempe, 1879, Appel and Haken, 1977, 1989). In the area of compiler optimization, register allocation is the process of assigning a large number of target program variables onto a small number of CPU registers. This problem has also been addressed as a graph coloring problem (Chaitin, 1982). The frequency allocation problem in mobile and wireless networks turns out to be a graph coloring problem. The objective of frequency allocation is the assignment of frequencies to all users of a cellular network, so that users in the same or adjacent regions are assigned distinct frequencies avoiding radio interference and the number of frequencies used in minimized. Corresponding frequencies to colors and users who wish to communicate to vertices of an appropriately defined graph, every instance of the frequency allocation problem is equivalent to an instance of graph coloring (Narayanan, Shende, 2001, Khanna, Kumaran, 1998). Graph coloring and its generalizations have proved to be useful tools in modelling a wide range of scheduling and assignment problems (Marx, 2004).

2.2 Scheduling and graph coloring

Assume that a set of interfering jobs must be scheduled. This implies that it has to be determined when each job is executed. Let G be the conflict graph of the jobs. Vertices of the conflict graph correspond to jobs and there is an edge between two vertices if the corresponding jobs cannot be executed simultaneously. Each job requires a unit time slot. Time slots correspond to colors. Then there is a one-to-one correspondence between the feasible job schedules and the colorings of the graph. That is, vertex v receives color i if and only if the corresponding job is executed in time slot i. Clearly, the graph admits a k-coloring if and only if jobs can be executed in k time slots in such a way that interfering jobs are not executed simultaneously. Therefore the chromatic number of the graph equals the minimum makespan of the scheduling problem, i.e., the minimum time required to finish the jobs.

Determining the chromatic number of a graph is a hard problem, hence we cannot expect to solve the scheduling problem efficiently for large graphs. However, graphs arising in several practical scheduling instances have a special structure that makes coloring easier. Furthermore, even if no optimal coloring algorithm can be devised, it may well be the case that an approximation algorithm - which is certainly not optimal, but offers a performance guarantee - produces good, acceptable schedules.

We consider the following scheduling problem. We are given a list C of courses $c_i \in C$, together with pairwise predefined thresholds, $T(c_i, c_j)$, reflecting the least possible number of students who should be able to attend

both courses c_i and c_j, either in terms of lectures or in terms of exams. Consequently, any number of students exceeding $T(c_i, c_j)$ implies that courses c_i and c_j cannot be scheduled for the same or overlapping time slots. For example, assuming that $T(c_A, c_B)=3$, if there exist 10 students who wish to attend both courses A and B, then these courses must be scheduled for non-overlapping time slots; however, the case where only 2 ($<T(c_A, c_B)=3$) students wish to attend both courses A and B does not impose a constraint for courses A and B to be scheduled for the same or overlapping time slots. We, then, address the question: how courses in list C can be scheduled so that all conditions imposed by $T(c_i, c_j)$ are met?

This scheduling problem just stated can be then solved using a graph model where vertices represent courses and there exists an edge between two vertices if there are a minimum common number of students in the courses they represent. Each time slot is represented by a different color. A course scheduling corresponds to a coloring of the associated graph.

Involved graph coloring algorithms and several applications of graph coloring to problems involving scheduling have appeared in the relevant literature. But, since no efficient algorithm is known for graph coloring, this approach has not yielded efficient scheduling algorithms.

However, in this work, our main objective has been to showcase that even a simple graph-theoretic approach can give significant insight and suggest efficient practical solutions to complex problems arising in academic routine. This is why we decided to use a rather simple coloring algorithm for addressing our scheduling instance.

Given an instance of a course scheduling problem, we produce a graph G=(V, E) where courses correspond to graph vertices and there is an edge between two vertices if corresponding courses must be scheduled for non-overlapping time slots. Available time slots correspond to colors represented by non-zero, positive integers, i.e., 1, 2, 3,… In this setting, a coloring of graph G suggests a course schedule.

Our algorithm works as follows. Initially, graph vertices v_1, v_2, v_3,… , v_n are sorted in decreasing order according to their degree so that $\deg(v_1) \geq \deg(v_2) \geq ... \geq \deg(v_n)$ producing a sorted list L. Then, color 1 is assigned to vertex v_1 and also to all subsequent vertices in L (if exist) which are not adjacent to v_1 and not adjacent to vertices which have already received color 1. Then color 2 is assigned to the first vertex in L not already colored. Successively color 2 is assigned to uncolored vertices not adjacent to vertices already assigned color 2. The algorithm proceeds in the same fashion assigning color 3 to still uncolored vertices that meet adjacency constraints and so on. The process terminates when there are not uncolored vertices.

The algorithm can use up to n colors, in case of complete graphs of n vertices. Practically, in our case, this upper bound cannot be reached since actual scheduling requirements never result in complete graphs due to curriculum constraints. In practice, the number of colors used equals $\Delta+1$, where Δ is the maximum degree of the underlying graph, which usually does not exceed a small constant. As far as the running time of the algorithm is concerned, the theoretical upper bound Is $O(n^3)$ since for each of the n available colors, at most n vertices must be checked for adjacency constraints and each such check may require a further check of at most n edges. However, in practice, within our particular context, the algorithm performs significantly better in terms of worst-case running time complexity.

3 COURSE AND EXAM SCHEDULING APPLICATION

We implemented our scheduling application using the MATLAB programming environment. In particular we used MATLAB version R2010a running on a Windows 10 pro machine with an INTEL(R) i7-4770 CPU (3.4 GHz) and a RAM of 8GB.

MATLAB is a numerical computing environment and fourth-generation programming language which allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran. Although it was intended primarily for numerical computing, it also allows symbolic computing, graphical multi-domain simulation and model-based design for dynamic and embedded systems. It has been widely used in academia and industry by users coming from various backgrounds of engineering, science and economics. MATLAB was first adopted by researchers and practitioners in control engineering, and quickly spread to many other domains. It is now also used in education, in particular the teaching of linear algebra and numerical analysis, and is very popular amongst scientists involved in image processing [Moler, 2004].

Our scheduling application receives as input an Excel file whose columns correspond to courses to be scheduled. Within each column, raw values indicate identification numbers of students registered for the particular course. Then a positive integer threshold, T, is set whose value indicates the minimum number of

students that must be registered for a pair of courses, so that these two courses cannot be scheduled for the same time interval. For example, setting T=2 means that if at least 2 students are found in the registration list of two courses, say c1 and c2, then c1 and c2 must be scheduled for different time slots.

Then, using the .xls input file and the threshold value, our application first determines all pairs of courses that cannot be scheduled for the same time interval. This information is then mapped on a graph G=(V,E) as follows. Courses correspond to graph vertices and there is an edge between two vertices if their corresponding courses cannot be scheduled for the same time interval. G is, thus, the conflict graph of our input instance. Furthermore, a coloring of G would directly imply a feasible course schedule: different colors indicate different time slots and vertices of the same color correspond to courses that can be scheduled for the same time slot while vertices of different color correspond to courses that cannot be scheduled for the same time slot.

We represent graph G via its adjacency matrix, i.e., a square |V| × |V| matrix A such that its element Aij is 1 when there is an edge connecting vertex i to vertex j, and 0 otherwise (Cormen, Leiserson, Rivest, Stein, 2001). Note that elements in the diagonal of A are all 0, since in simple graphs loops, i.e., edges from a vertex to itself, are not allowed. Then, we use the simple coloring algorithm of Section 2.2 for coloring G.

Our application returns the number of colors used for the coloring G – which indicates the total 3-hour time slots required - together with suggested schedules in tabular form for the courses provided as part of the input.

3.1 The case of a 2015 fall semester course schedule

In the following, we present a show case based on an actual sample of courses and corresponding registration lists from the 2015 fall semester at the department of Cultural Heritage Management and New Technologies, University of Patras, Greece.

The input .xls file contains 27 columns, one for each course of the fall semester curriculum. Course registration lists contain on average 93 students. Information included in the .xls file provided as input can be visualized as depicted in Fig. 1.

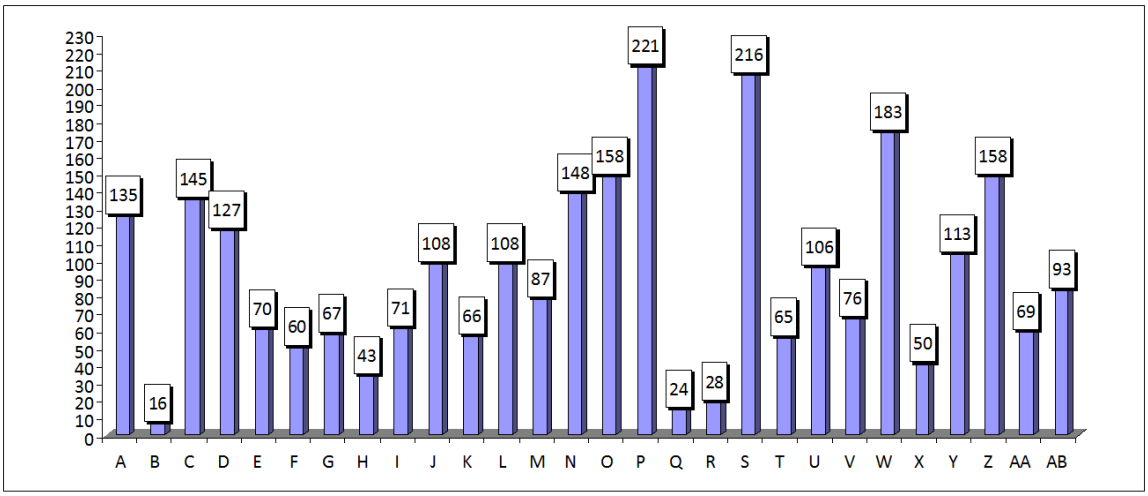

Fig. 1: visualization of the input .xls file containing course registration lists

The required schedule must cover at most 5 working days (from Monday to Friday) for a maximum of 12 hours per day (from 9.00 to 21.00). Each lecture requires a 3-hour time interval. When examining course registration pairwise, increased number of overlaps implies courses of the same year of studies. This means that in the conflict graph corresponding to our input instance, we expect to see compact subgraphs which are almost cliques containing nodes that represent courses of the same year of studies. Given that attendance is not obligatory, we tried several threshold values.

Results returned by our application are summarized below. Fig. 2 shows the number of necessary working days computed by our coloring algorithm as a function of the threshold value. As expected, increasing the threshold value allows more courses to be scheduled simultaneously, thus reducing makespan. Notice that for threshold values greater or equal to 6, feasible schedules, i.e., schedules requiring at most 5 working days, are produced. A threshold value between 6 and 15 is considered to be acceptable for real attendance

requirements.

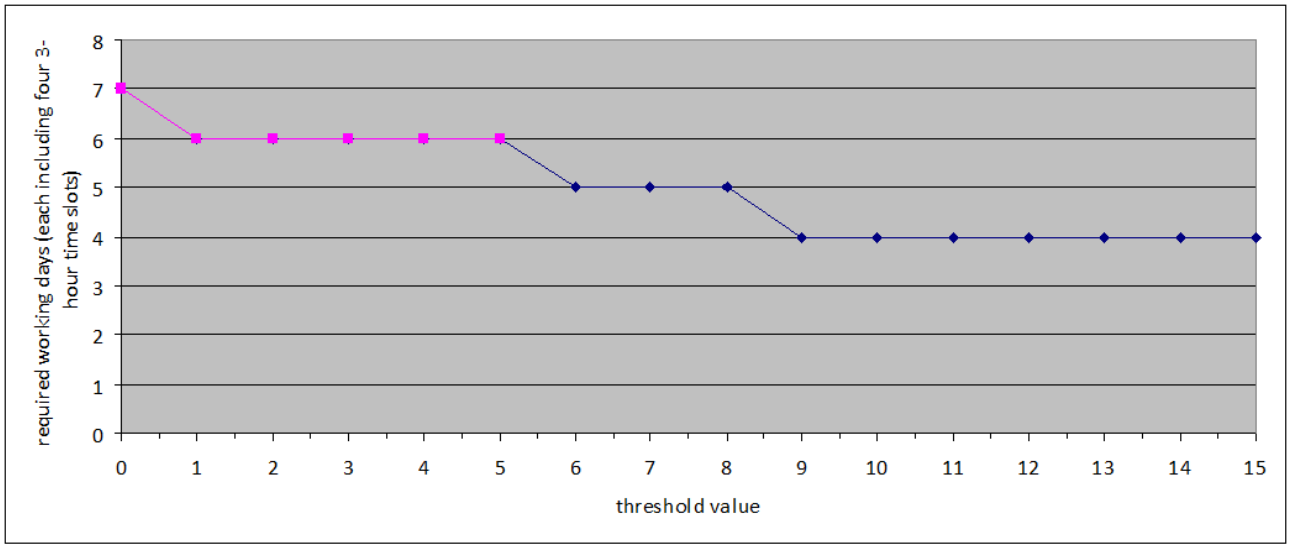

Fig. 2: required working days computed by our coloring algorithm assuming different threshold values

Assignment of courses to time slots for integer threshold values in the range [6,15] are depicted in Fig. 3. Time slots correspond to different colors while courses scheduled for the same time slot correspond to non-neighboring graph vertices which received the same color by the coloring algorithm.

	Threshold = 6	Threshold = 7	Threshold = 8	Threshold = 9	Threshold = 10	Threshold = 11	Threshold = 12	Threshold = 13	Threshold = 14	Threshold = 15
Time slot	Course ID	Course ID	Course ID	Course ID	Course ID	Course ID	Course ID	Course ID	Course ID	Course ID
1	3	3,17	3,17	2,13,17	1,2,17	1,2,17	2,3,18	2,3,18	2,3,18	2,3,18
2	16,17	1,2	2,16	1	10,18	4,18	1,17	1,17	1,17	1,17,24
3	2,10	19	10	18,25	4	3	6,8,25	6,8,25	6,8,25	6,8,16
4	1	16	1	21	3	8,16	23	16	16	4
5	25	10	18,25	16	6,25	10	16	12,24	4	10
6	18,22	18	21	12	8,23	6,25	12,24	10	10,24	11,25
7	19	25	12	10	16	23	10	4	12,27	12,27
8	4	21	4	4	12	12,24	4	11,23	21,26	21,26
9	23	12	8,23	8,23	15,27	19,27	21,26	19,27	5,23	5,19,22
10	21	4	5,15	15,27	21,26	21,26	5,15,20	21,26	15,22	9,23
11	12	8,23	13,27	5,26	5,14	5,14,20	19,27	5,15,20	11,19	7,15,20
12	8,14,20	5,15	22,26	19,24	19,24	15,22	14,22	14,22	9,14,20	14
13	11,26	13,27	19,24	14,22	13,22	11,13	11,13	9,13	7,13	13
14	9,15	24,26	11,14	11,13	9,20	9	9	7		
15	5	14,22	7,2	9,20	11	7	7			
16	13,27	11	9	6,7	7					
17	24	9	6							
18	6	7,20								
19	7	6								

Fig. 3: assignment of courses to time slots for integer threshold values in the range [6,15]

Based on the computed assignment of courses to time slots, the application also returns a suggested weekly schedule in tabular format for the set of courses provided as part of the input. Suggested weekly schedules are automatically extracted to xls files which can be saved and further modified (manually). In Fig. 4, weekly schedules for the 27-course curriculum are depicted for threshold values equal to 6, 7, and 8.

Threshold=6

	Monday	Tuesday	Wednesday	Thursday	Friday
Time slot	Course ID	Course ID	Course ID	Course ID	Course ID
09.00-12.00	3	25	23	5	24
12.00-15.00	16,17	18,22	8,14,20	9,15	13,27
15.00-18.00	2,10	19	12	11,26	7
18.00-21.00	1	4	21	6	

Threshold=7

	Monday	Tuesday	Wednesday	Thursday	Friday
Time slot	Course ID	Course ID	Course ID	Course ID	Course ID
09.00-12.00	19	10	25	12	11
12.00-15.00	3,17	5,15	24,26	7,20	9
15.00-18.00	8,23	13,27	14,22	1,2	6
18.00-21.00	16	18	21	4	

Threshold=8

	Monday	Tuesday	Wednesday	Thursday	Friday
Time slot	Course ID	Course ID	Course ID	Course ID	Course ID
09.00-12.00	10	21	4	6	
12.00-15.00	3,17	18,25	5,15	22,26	11,14
15.00-18.00	2,16	8,23	13,27	19,24	7,2
18.00-21.00	1	12	9		

Fig. 4: weekly schedules for the 27-course curriculum are depicted for threshold values equal to 6, 7, and 8

We have created a template for presenting computed schedules. Column titles (e.g., Time slot, Course ID) as well as time intervals indicating different time slots can be easily modified. For demonstrating purposes, we have used numbers instead of titles for identifying courses; this information is directly received from the xls file provided as input and, therefore, can be easily modified to meet particular schedule presentation requirements.

Regarding the response time of our application, it seems that in practice, given a usually rather short curriculum, it performs pretty well. In Fig. 5, response times are depicted for the input list of 27 courses and threshold values between 0 and 15. The blue line shows the time required by our application for computing a schedule and presenting it in a tabular form. This process includes (a) using the input xls file for constructing a corresponding conflict graph (via its adjacency matrix), (b) coloring the induced graph, (c) corresponding colors to time slots and (d) creating a new xls output file containing the computed schedule in the form of a timetable. The pink line shows the time required just for the main computation of the schedule, i.e., points (a) and (b) of the previously outlined process.

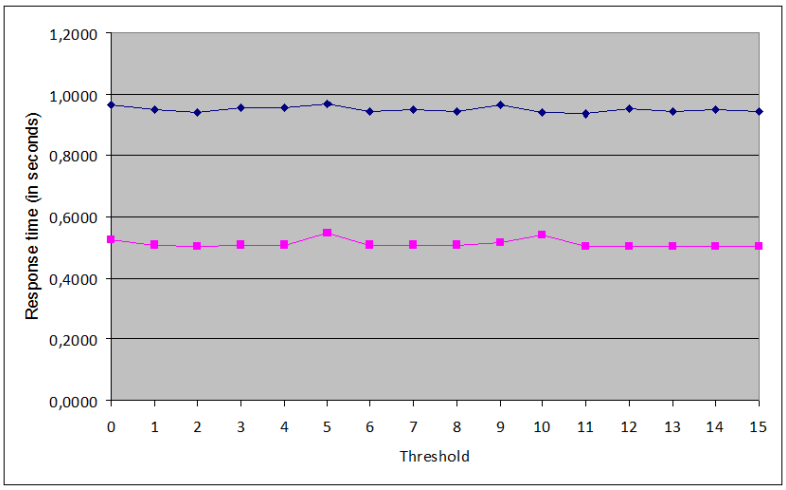

Fig. 5: Response time in seconds for the input list of 27 courses and threshold values between 0 and 15

Despite the fact that theoretical analysis highlights potential limitations of our coloring algorithm, the particular practical contextual framework makes it a quite efficient approach. Even in more complex scenaria (longer course lists, strict attendance requirements, etc) our suggested approach certainly contributes in saving time and effort for lecture and exam scheduling in university departments.

4 CONCLUSION AND FUTURE PLANS

Motivated by the inherently hard task of producing good lecture and exam schedules under constraints regarding student participation frequently arising in academic environments, we exploited the interplay between two important and well-studied graph-theoretical problems, namely graph coloring and scheduling, and developed a MATLAB application which computes efficient lecture and examination timetables for higher education departments. Receiving as input a list of courses and a corresponding list of constraints, our application utilizes a simple coloring algorithm for computing efficient lecture and exam schedules which are then presented in a comprehensive way.

Our work showcases how ideas and results from graph theory can be used to provide efficient solutions to real-world problems highlighting lecture and examination scheduling of courses in higher education. Our findings suggest that even the simple coloring algorithm used, which is theoretically expected to achieve a poor performance in worst-case scenaria, performs well in practice. Experimental evaluation so far shows that even a suboptimal coloring algorithm provides more efficient solutions than those based on heuristics.

Ongoing work focuses on the use of more involved coloring algorithms for addressing more complex course scheduling instances while minimizing required time resources. Furthermore, future plans include the implementation of a more user-friendly graphical interface with additional options for parameterization and scalability based on feedback obtained from practical experience.

REFERENCE LIST

Appel, K. and Haken, W. (1977). The Solution of the Four-Color Map Problem. Scientific American, vol. 237, no. 4, pp. 108-121.

Appel, K. and Haken, W. (1989). Every Planar Map is Four-Colorable. American Mathematical Society, vol. 98.

Bondy J. A. and Murty U. S. R (1982). Graph theory with applications. Elsevier Science Publishing Co., Inc. ISBN: 0-444-19451-7

Brooks, R. L. (1941). On Coloring the Nodes of a Network. Mathematical Proceedings of the Cambridge Philosophical Society, Vol. 37, Issue 2, pp. 194-197.

Chaitin, G. J. (1982). Register allocation & spilling via graph colouring. In Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction, pp. 98–105.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2001). Introduction to Algorithms (2nd ed.), MIT Press and McGraw-Hill, ISBN 0-262-03293-7.

Feige U., Kilian J. (1998). Zero knowledge and the chromatic number. Journal of Computer and System Sciences, vol. 57, pp. 187–199.

Garey, M. R., Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York, 1979.

Håstad J. (1999). Clique is hard to approximate within n^{1-e} . Acta Mathematica, vol. 182, pp. 105–142.

Karp R. M. (1972). Reducibility Among Combinatorial Problems. Complexity of Computer Computations, pp. 85–103.

Kempe, A. B. (1879). On the Geographical Problem of Four Colors. American Journal of Mathematics, vol. 2, no. 3, pp. 193-200.

Khanna S. and Kumaran K. (1998). On Wireless Spectrum Estimation and Generalized Graph Coloring. In Proceedings of the 17th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1998), IEEE, pp. 1449 - 1461.

Khot S. (2001). Improved inapproximability results for MaxClique, Chromatic Number and Approximate Graph Coloring. In Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001), IEEE, pp. 600–609.

Lewis, R. (2015). A Guide to Graph Colouring: Algorithms and Applications. Springer International Publishers.

Lovász, L. (1975). Three Short Proofs in Graph Theory. Journal of Combinatorial Theory, Series B, vol. 19, no. 3, pp. 111-113.

Marx, D. (2004). Graph colouring problems and their applications in scheduling. Periodica Polytechnica, Electrical Engineering, 48 (1–2), pp. 11-16.

Moler, C. (2004). The Origins of MATLAB. Mathworks.

Narayanan L. and Shende S. (2001). Static Frequency Assignment in Cellular Networks. Algorithmica, vol. 29, issue 3, pp 396–409.

Pardalos P. M., Mavridou T., Xue J. (1998). The Graph Coloring Problem: A Bibliographic Survey. Handbook of Combinatorial Optimization, Kluwer Academic Publishers, vol. 2, pp. 331-395.

Roberts, F. S. (1978). Graph theory and its applications to the problems of society, CBMS-NSF Monograph 29, SIAM Publications.

Xu J. (2003). Theory and application of graphs. Springer Science and Business Media, LLC. ISBN 9778-1-4613-4670-8

Zuckerman, D. (2007). Linear degree extractors and the inapproximability of max clique and chromatic number. Theory of Computing, vol. 3, pp. 103–128.