A Hybrid Exam Scheduling Technique based on Graph Coloring and Genetic Algorithms Targeted Towards Student Comfort

Osama Al-Haj Hassan[*,1], Osama Qtaish[†,2], Maher Abuhamdeh[‡,3], Mohammad Al-Haj Hassan[§,4]
[*]Computer Networks Department, Isra University, Amman, Jordan
[†]Software Engineering Department, Isra University, Amman, Jordan
[‡]Computer Information Systems Department, Isra University, Amman, Jordan
[§]Computer Science Researcher, Amman, Jordan

***Abstract*—Scheduling is one of the vital activities needed in various aspects of life. It is also a key factor in generating exam schedules for academic institutions. In this paper we propose an exam scheduling technique that combines graph coloring and genetic algorithms. On one hand, graph coloring is used to order sections such that sections that are difficult to schedule comes first and accordingly scheduled first which helps in increasing the probability of generating valid schedules. On the other hand, we use genetic algorithms to search more effectively for more optimized schedules within the large search space. We propose a two-stage fitness function that is targeted toward increasing student comfort. We also investigate the effect and potency of the crossover operator and the mutation operator. Our experiments are conducted on a realistic dataset and the results show that a mutation only hybrid approach has a low cost and converges faster toward more optimized schedules.**

***Keywords*—Exam scheduling; optimization; graph coloring; genetic algorithms; time tabling; fitness value**

I. Introduction

Several life activities exploit scheduling to ensure efficient and proper operation including exam scheduling that is used in academic institutions for producing high quality exam schedules [1], [2], [3].

Exam scheduling is an optimization problem that aims to produce valid and optimized exam schedules. Several criterions govern the concept of 'Optimized Exam Schedule'. For example, a schedule might be optimized based on finishing the whole examination period in a minimum number of days. It can also be optimized based on having the minimum number of exams for a given student per day. Another way of optimizing an exam schedule is maximizing the time period between exams for a given student which gives the student more time to prepare for his exams. In other words, the criterion upon which an exam schedule is optimized largely depends on the administration and their notion of a good schedule.

Exam scheduling problem is a case of graph coloring problem which is known to be NP-Complete [4]. Therefore, in real-world scenarios where exists a large number of faculties, courses, sections and students; finding the optimum exam schedule is a time consuming and cumbersome process. Accordingly, the aim of exam scheduling algorithms is generating exam schedules that are closer to the optimum schedule. Since it is very difficult to know the optimum schedule in the first place, then any improvement on the fitness of schedules would be considered as one step closer to the optimum schedule.

We consider exam scheduling in an academic institution setting where academic faculties consist of departments. Those departments offer courses that have sections. Therefore, the aim of the exam scheduling process is to schedule sections in a way that generates optimized schedules.

Graph coloring is one of the techniques used for exam scheduling. It depends on a greedy approach that schedules sections in the current best possible slot such that no hard scheduling constraint is violated. It generates high quality exam schedules in an efficient way. However, due to its greedy nature and huge search space, it fails to find more optimized exam schedules.

Exam scheduling problem can be tackled using genetic algorithms which are approaches that depend on randomization to explore the large search space of possible solutions. The search space is explored in a time and resource intensive manner. Due to their randomized nature, they are able to find more optimized exam schedules. However, they are time intensive techniques. We explain our methodology as follows. The techniques that rely on randomization are the most effective for exam scheduling problem. Some of those approaches are genetic algorithms, ant colony, bee colony, particle swarm, and memetic algorithms. The reason behind that success is because it is nearly impossible to find the most optimized solution in problems with large search space using exhaustive search. Therefore, the randomized nature of the previously mentioned algorithms helps in scanning many areas of the large search space to find better solution. However, those randomized approaches tend to be blind in the way they cover the search space and accordingly might miss several good solutions. Consequently, it is strongly advised to support those techniques with other more focused approaches that help in guiding the randomized algorithms to cover the areas of the search space that are more likely to have higher quality solutions. This is why we choose a hybrid approach that utilizes the randomized nature of genetic algorithms and the guided search of graph coloring.

A. Contribution

In this paper, we develop a technique in which we combine graph coloring and genetic algorithms to design a hybrid approach that generates exam schedules that come closer to finding optimum schedules. Our paper contribution is laid out

in the following points:

- Exploiting graph coloring and genetic algorithms in a realistic scenario comprising realistic exam scheduling constraints.
- Proposing a new course registration dataset as a contribution to the research community.
- Designing a unique two-stage fitness function that focuses on increasing student's comfort during the examination period.
- Investigating the necessity of existence of crossover and mutation operators via inspecting their cost and effectiveness.

This paper is organized as follows. In Section II we present literature conducted in the areas of graph coloring and genetic algorithms and both of them combined. Then, in Section III we define the exam scheduling problem in the context of graph coloring and genetic algorithms. After that, we propose our hybrid exam scheduling approach in Section IV. Following that, we evaluate our approach in Section V. Finally, we conclude our work in Section VI.

II. Related Work

In this section, we investigate different research work related to graph coloring, genetic algorithms and combining both of them.

A. Graph Coloring

Graph coloring has been used in several works for scheduling purposes. In [1] authors employ graph coloring to generate course timetabling and exam timetabling and they take into consideration several hard and soft constraints in the process. Their work focuses on measuring the degree of satisfaction of constraints and they try to achieve even distribution of courses. Authors in [2] exploit graph coloring to generate exam schedules by following a two phase scheme such that in phase one they schedule exams regardless of number of available seats in halls. Then, in phase two, they check if the number of needed seats in the generated schedule exceeds the number of available seats. If that happens, then they remove the scheduling of some exams and reschedule them again. In [5], authors provide a performance study for various graph coloring algorithms including First Fit, Welsh and Powell, Largest Degree Ordering, Degree of Saturation, Incidence Degree Ordering and Recursive Largest First. Graph coloring is used in [6] to enhance communication with users in massive multiple input multiple output wireless networks. The idea is to use graph coloring for pilot assignment instead of assigning pilot randomly on users which leads to reduced inter-cell interference between users. Most research focus on using graph coloring in situations where the graph does not change (static). However, the research in [7] investigates using graph coloring in problems where the graph is dynamic. They take into consideration random and heuristic changes on the graph. Graph coloring for dynamic graphs is also investigated in [8] where authors focus on a technique to ensure high effectiveness and high efficiency. They achieve that by performing incremental color propagation as the graph changes. An attempt to reduce the time needed to color a graph is conducted in [9] where authors identify edge cover graphs and independent graphs within the graph to be colored. The graph coloring attempts for solving the scheduling problem only generate good enough solutions but they fail in finding near optimal solutions.

B. Genetic Algorithms

Genetic algorithms are key optimization techniques which have been used in various problems. In [10], a genetic algorithm is developed to help robots choose the most optimal path to their goal. It is also used in real time task scheduling [11] where the goal is to minimize the number of processors needed to accomplish all the given tasks. Authors in [12] focus on solving the exam timetabling problem using a genetic algorithm that maximizes time between student classes. This ensures that students have enough time and comfort to succeed in their exams. Scheduling tasks that include batch delivery on machines is investigated in [13]. It is achieved by using a genetic algorithm that minimizes job completion time while at the same time minimizes machine deterioration. A genetic algorithm for project scheduling is proposed in [14] where it is assumed that project tasks cannot be preempted and they use finite resources. Their goal is to minimize project completion time. Authors in [15] present a genetic algorithm for task scheduling in cloud computing. They analyze user characteristics and type of requests in order to classify jobs and their requirements. A genetic algorithm is utilized in [16] to find optimal buses routes that pick up and drop students at different locations. The employed genetic algorithm is able to find the shortest path for university buses. In [17] the graph coloring problem is solved using a Michigan genetic algorithm where each chromosome of the population represents one vertex and therefore the population represents one solution. Each chromosome evolves to find a better coloring decision and this evolution is also supported with local search to find better coloring decisions. The genetic algorithms that we mentioned in this section have a main shortcoming which is starting with bad solutions and that results in increasing the convergence time for the genetic algorithm.

C. Hybrid Graph Coloring and Genetic Algorithms

Several attempts have been performed to combine graph coloring and genetic algorithms. In [18], authors employ graph coloring to build their initial population and then they utilize genetic algorithms to evolve to better solutions using a fitness function that calculates the number of conflicts for each solution. Authors in [19] propose a technique that depends on varying weights for measuring conflicts and their fitness function relies on minimizing the number of colors used in finding solutions. The research conducted in [20] experiments with an algorithm that uses Tabu search and authors claim that eliminating the Tabu search component from the algorithm does not degrade the algorithm effectiveness. The work proposed in [21] adds parallelism to genetic algorithms and graph coloring combination, authors use sub populations (islands) that evolve to find solutions. Authors in [22] propose using adaptive parent selection and mutation techniques that work with genetic algorithms and graph coloring. Authors in [23] propose a hybrid approach that is targeted to exam scheduling problem and their fitness value depends on number of conflicts

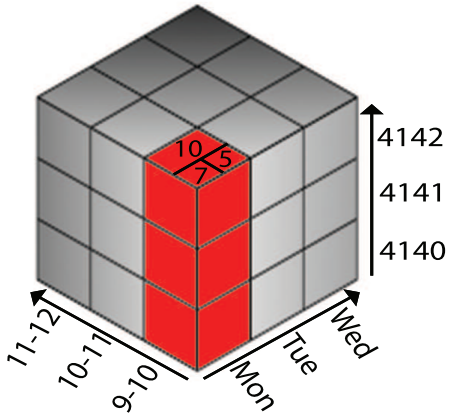

Fig. 1. Representation of 'Slot' and 'Color'.

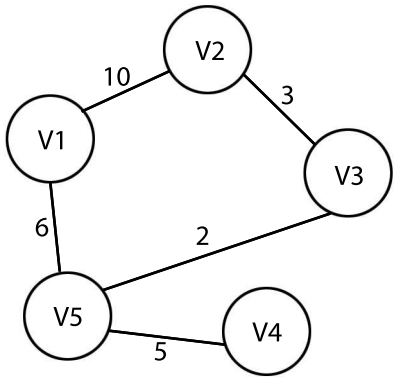

Fig. 2. Graph representation of exam scheduling problem.

in exam schedules. Our work is different in two points; first, none of the previous hybrid approaches investigates the use of graph coloring combined with genetic algorithms specifically designed for exam scheduling settings except the work in [23]. However, the work in [23] does not discuss the possibility of omitting the crossover or the mutation operator. Second, none of the previous hybrid techniques investigates the necessity of existence of crossover or mutation operators except the work in [19]. However, the work in [19] does not target the exam scheduling problem, and therefore their choice of fitness function is too general. Also, the work in [19] does not clarify in details how crossover operator is performed while maintaining valid solutions. Moreover, our two-stage fitness function is different than what is used in the previous works.

III. Problem Definition

To make a concrete definition of exam scheduling problem, it is important to clarify the definition of a 'slot' in which an exam can be scheduled and the definition of a 'color' that a section can be assigned. We will also explain the scheduling problem in the context of graph coloring and genetic algorithms.

A. Slot and Color Definition

Suppose that $DayList = \{D_1, D_2, ..., D_d\}$ is a set of days during which an exam can be scheduled. Also, suppose that $TimeList = \{T_1, T_2, ..., T_t\}$ is a set of existing time periods in each day. Suppose also that $HallList = \{H_1, H_2, ..., H_h\}$ is a set of halls available in each time period. Each intersection of a day D_i and time period T_j is considered a color such that $ColorList$ is a list of length $d * t$ that contains all colors. For example, the q^{th} color in colors list corresponds to day D_i and time period T_j and is denoted as $C_q^{i,j}$. For presentation purposes, we only use the subscript of colors in the manuscript unless we needed to mention the day and time period to which a color belongs. A slot represents preserving number of seats in a given hall in a specific day and time for a single exam. Based on that, $SlotList$ is a set of slots to which exams are assigned such that a single 'slot' is denoted as $S_{k,w}^{i,j}$ which represents holding an exam in day D_i and time period T_j where the exam takes place in hall H_k wherein w seats are reserved for the exam out of the total capacity of hall H_k. This representation of a color and a slot is illustrated in Fig. 1. For example, in Fig. 1 the slot marked with number '5' is a slot in which an exam is scheduled such that it is held on 'Monday' at '9-10' in hall '4142' where '5' seats are reserved for the exam out of 22 seats that represents the total hall capacity. Based on our previous definition, a single slot can be occupied by only one exam. Most related works represent a slot as a (Day,Time) pair. In our opinion, our definition is more precise and closely reflects reality and it eases the stage of implementing the algorithm. Since a color is represented by a day and time pair, then all slots that belong to the same day and time are said to have the same color. For example, in Fig. 1, all slots of day 'Monday' and time '9-10' belong to the same color which we mark in red.

B. Graph Coloring for Exam Scheduling

In graph coloring, the exam scheduling problem can be thought of as follows. Suppose $VertexList = \{V_1, V_2, ..., V_v\}$ is a set of vertices in an undirected graph such that each vertex represents a section of a course. $EdgeList = \{E_1, E_2, ..., E_e\}$ is a set of edges such that an edge between two vertices (sections) represents the number of common students who are currently registered in the two sections. An example is illustrated in Fig. 2. This graph is represented as an adjacency matrix where the value at the intersection of two sections represents the number of common students between them (Table I). Recall that any two slots that share the same day and time are given the same color. Therefore, the exam scheduling problem can be solved by applying graph coloring where the exam of each section is scheduled in a slot such that no two adjacent vertices (sections with common students) are assigned slots of the same color. Graph coloring depends on a greedy approach that schedules sections in the current best possible slot.

C. Genetic Algorithms for Exam Scheduling

Genetic algorithms use the concept of evolution to find high quality schedules. In the context of exam scheduling, a chromosome represents a solution for the exam scheduling problem where each section is scheduled in a slot of a

TABLE I. ADJACENCY MATRIX OF THE GRAPH IN FIGURE 2.

	V_1	V_2	V_3	V_4	V_5
V_1	0	10	0	0	6
V_2	10	0	3	0	0
V_3	0	3	0	0	2
V_4	0	0	0	0	5
V_5	6	0	2	5	0

TABLE II. CHROMOSOME REPRESENTATION EXAMPLE.

0	1	2	3	4	5	6	7
C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
V_1,V_9	V_2	V_4	V_5,V_7	V_3,V_9	V_8	V_{11}	V_{10}

Algorithm 1 Initialization Code
Input: VertexList,dayIndex,TimePeriodIndex
Output: None

```
1: createDayIndex();
2: createTimePeriodIndex();
3: sectionListDegreeOrdering();
4: sectionListWeightOrdering();
```

given color. In other words, a chromosome represents a ready exam schedule. For example, Table II shows an example of a chromosome representing 11 sections and the colors that are used for each section's exam. A population of chromosomes is created where each chromosome has a specific fitness value that determines how good the exam schedule is. Several chromosomes are chosen from this population for mating and mutation. The matting process generates offspring consisting of new chromosomes. The offspring chromosomes replace chromosomes in the population that have fitness value worse than the fitness value of offspring chromosomes. This round of evolution is repeated for several generations hoping that the evolution process will eventually generate more optimized chromosomes (schedules).

IV. THE HYBRID APPROACH

In this section we describe the design and algorithms of the hybrid approach.

A. Day Index and Time Period Index

The forthcoming algorithm normally needs to traverse day list and time period list. For convenience purposes, we create an index for day list and an index for time period list. The indexes are simply arrays that control the order in which we traverse the elements in the day list and time period list. Initially, the day index and the time period index are set to the normal array indexes that start from zero and end with the array size minus 1. However, the hybrid approach relies on randomization, and therefore, these two indexes will be manipulated in a random fashion and utilized in the hybrid approach which helps in generating diverse exam schedules. Consequently, lines 1 and 2 of the code in Algorithm 1 are executed as part of the initial steps of the hybrid approach. Due to presentation purposes, we do not provide pseudocode for most of the functions in the forthcoming algorithms.

B. Section List Ordering

Recall that a section is a vertex in the exam scheduling undirected graph. Therefore, the degree of a section is the number of adjacent nodes. In other words, the degree of a section is the number of sections with which it has common students. The weight of a section is the largest weight among weights of the edges connecting the section with its adjacent sections. For example, in Fig. 2, the degree of section 5 is 2 and its weight is 6. In our algorithm we order sections list in descending order based on their degree. After that, sections of the same degree are ordered in descending order based on their weight. Sections in the start of the list will be scheduled first. The rationale behind this is that sections with high degree and weight have higher number of students in common with other sections, and therefore, scheduling this type of sections is difficult. Consequently, it is better to start scheduling these sections early while we still have many available slots. This is why lines 3 and 4 of the code in Algorithm 1 are executed as part of the initial steps of the hybrid approach.

C. The Graph Coloring Part

The graph coloring algorithm is shown in Algorithm 2. For each section, we first test if the section has already been colored (assigned a slot). If this is true, then we move on to the next section (Lines 2-5). All colors will be searched for an available slot (lines 6-12) such that exam scheduling constraints are not violated. Among the available slots, the chosen one for the section's exam is the one that keeps the number of students having two exams in the same day as minimum as possible. This is why we need variables 'minTwoExamsCount' (line 6) and 'selectedSlot' (line 7) to keep track of the slot that would currently generate the minimum number of students with two exams in the same day if the slot is chosen for the exam assignment. Notice that lines 9 and 11 make use of the previously mentioned indexes to access the days and time periods. This gives us the flexibility of accessing the days and time periods in the order we desire. Line 12 represents the color in which the algorithm tries to find a slot available for the section's exam. After that (Line 14), we call Algorithm 3 which goes through a set of scheduling constraints and returns a slot for the exam if possible. If the output of the aforementioned algorithm is 'null', then it means that no available slot was found in that color and hence we move on to the next color (Lines 15-18). Otherwise, we find how many students would have two exams in the same day if the exam is scheduled in the returned slot. We keep track of the slot that minimizes this number as much as possible(Lines 19-23). Eventually, a slot may be found such that it does not violate any constraint while at the same time it generates the greedy minimal number of students with two exams in the same day. This slot is used for the section's exam (line 26). If no slot is found, then the section stays uncolored. Now, we briefly explain each one of the constraints in the order they appear in line 2 of Algorithm 3.

- Constraint 1: We make sure the maximum number of exams for instructor per day is not reached.
- Constraint 2: Sometimes, there would be certain days and time periods during which the instructor is not available. For example, the instructor is leaving for a

Algorithm 2 Graph Coloring
Input: Previously mentioned lists
Output: A slot for each section if possible

```
for (each section V_i in VertexList) do
    Section sec=V_i;
    if (sec.color ≠ null) then
        continue;
    end if
    int minTwoExamsCount=Integer.MAXVALUE;
    Slot selectedSlot=null;
    for (each day D_j in DaysList) do
        int dayPosition=dayIndex[j];
        for (each TimePeriod T_k in TimePeriodList) do
            int periodPosition=timePeriodIndex[k];
            Color col=colorList[dayPosition,periodPosition];
            boolean validColor=true;
            Slot slot=ExamSchedulingConstraints(col,sec);
            if (slot == null) then
                validColor=false;
                continue;
            end if
            int examCount=twoExamsCount(col,sec);
            if (examCount<minTwoExamsCount) then
                minTwoExamsCount=examCount;
                selectedSlot=slot;
            end if
        end for
    end for
    sec.examTimeSlot=slot;
end for
```

Algorithm 3 Exam Scheduling Constraints
Input: Color col, Vertex sec
Output: Slot

```
Slot slot=null;
if (isDailyMaxExamsForInstructor(col,sec) or
Not(canColorBeUsedForInstructor(col)) or
Not(colorWithinLimit(col,sec)) or
isMaxExamsPerColorReached(col) or
isClashWithAnotherSectionClass(col,sec) or
isClashWithAnotherSectionExam(col,sec) or
studentsOfSectionHaveClassInColor(col,sec) or
studentsOfSectionHaveExamsInColor(col,sec) or
instructorOfSectionHaveClassInColor(col,sec) or
instructorOfSectionHaveExamsInColor(col,sec) or
(slot=findAvailableHallForExam(col,sec))==null or
studentsCountWithThreeExamsViolated(col,sec) then
    return null;
else
    return slot;
end if
```

conference trip. If this is one of those days and time periods, then this assignment is considered invalid.

- Constraint 3: Some academic institutions operate in a morning and evening system. In this case, a class held in the morning session cannot have its exam scheduled in the evening session.
- Constraint 4: We make sure the maximum number of exams per color is not reached; this is usually used to balance distribution of exams between colors.
- Constraint 5: An exam cannot be held in the same day, time, and hall in which a class is being held.
- Constraint 6: An exam cannot be held in the same day, time, and hall in which another exam is being held.
- Constraint 7: An exam cannot be held in a given day, time if there exists common students with another class held in the same day and time as those students can either attend the class or the exam.
- Constraint 8: An exam cannot be held in a given day, time if there exists common students with another exam held in the same day and time as those students cannot attend both exams simultaneously.
- Constraint 9: An exam cannot be held in a given day, time if there exists another class for the same instructor in the same day and time.
- Constraint 10: An exam cannot be held in a given day, time if there exists another exam for the same instructor in the same day and time.
- Constraint 11: We try to find an available hall for the exam in the given color.
- Constraint 12: We check sure that placing an exam here does not cause a student to undertake 3 exams in the same day.

D. The Hybrid Part

In the hybrid approach, we utilize graph coloring in a genetic algorithm for the purpose of generating high quality schedules.

1) Fitness Value: Fitness value is the criterion used for deciding which chromosome (solution) is better than the other. Basically, the answer to the question 'What is a good schedule?' governs the decision of defining and computing the fitness value. As we mentioned in Section I, a good exam schedule is something debatable. In this work, we compare two schedules based on two stages such that if the first stage failed to determine the best schedule among the two schedules, then we use the second stage for that purpose. In stage 1, we define a 'good schedule' as a schedule that has the minimum total number of students having 'two exams in the same day' throughout all exam schedule days. For example, if a given schedule has 30 students having 2 exams in the same day over the exam schedule period and another schedule has 20 students having the same issue, then the later schedule is better. This choice of fitness function ensures a more comfortable exam schedule for students because we minimize the chance for a student to undertake two exams in the same day. Suppose that $StudentList = \{U_1, U_2, ..., U_u\}$ represents a set of students. Based on that, the fitness function can be formulated as in Equation 1. The fitness function is the summation of the value TE for all students $\{U_1, ..., U_u\}$ in all days $\{D_1, ..., D_d\}$ such that TE equals 1 if the summation of exams for a given student U_g in all time periods $\{T_1, ..., T_t\}$ of a given day D_i equals two, otherwise TE equals zero. Here, EC_{ijg} represents the

exams count for student U_g in day D_i in time period T_j. We can see that our optimization problem is a minimization one where the goal is reaching a schedule with minimum fitness value.

$$F = \sum_{g=1}^{u} \sum_{i=1}^{d} TE : TE = \begin{cases} 1, & \text{if } \sum_{j=1}^{t} EC_{ijg} = 2 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

In addition, we aim to maximize the time between consecutive student exams. So, if two schedules are considered equal based on stage 1, then in stage 2 we find the average number of gaps between consecutive exams for all students. Consequently, the schedule with more gaps between consecutive exams is considered better. This value can be found by computing Equation 2. $S_{h1,w1}^{i1,j1}$ represents the slot assigned for the first exam and $S_{h2,w2}^{i2,j2}$ constitutes the slot assigned for the next consecutive exam for a given student. So, if $i_2 = i_1$, then the two exams are held in the same day and therefore the gap between the two exams is actually the gap between the time periods to which the two slots belong $(j_2 - j_1 - 1)$. However, if $i_2 \neq i_1$, this means that the two exams are held in different days. Accordingly, we calculate the gap between that two exams as follows. First, we compute the day difference $(i_2 - i_1)$ and then we add 1 to the result for the purpose of giving this case a higher weight than the case in which the two exams belong to the same day. Second, we multiply the result with the total number of time periods in a given day (t). The final value CE is divided by number of students (u) to find the average number of gaps between two consecutive student exams. The combination of stage 1 and stage 2 leads the algorithm to generate schedules that are more comfortable to students.

$$F = \frac{\sum_{g=1}^{u} \sum_{i=1}^{d} CE(S_{h1,w1}^{i1,j1}, S_{h2,w2}^{i2,j2})}{u}$$
$$: CE = \begin{cases} j_2 - j_1 - 1, & \text{if } i_2 = i_1 \\ (i_2 - i_1 + 1) \times t, & \text{if } i_2 \neq i_1 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

2) Population Generation: We start by creating a population (pool) of chromosomes (solutions). The population size is a parameter we control in the system. Instead of randomly choosing section/slot assignment, we use graph coloring to build each chromosome. The rationale behind this decision is as follows. The sections ordering step in the graph coloring algorithm increases the probability of generating good quality exam schedules because the sections with high degree and high weight are the toughest to schedule. Therefore it is important to schedule them first. This leaves various scheduling options for the rest of sections and consequently we end up with good quality exam schedules. Moreover, random Section/Slot assignment will generate invalid exam schedules because there are many constraints that need to be met before a given section's exam can be assigned to a specific slot. Random Section/Slot assignment is blind of these constraints and generates invalid schedules. Fixing invalid schedules is a time consuming inefficient process that we try to avoid as much as possible. In addition, even if random section/slot assignment happened to produce valid schedules, the fitness value of the schedules is going to be high (poor schedules) and that would increase the time needed for the evolution part of the hybrid approach to converge to an optimized solution. The population generation part of the algorithm is represented by lines 1-6 in Algorithm 4 .

Algorithm 4 Hybrid Approach
Input: Previously mentioned lists
Output: A slot for each section if possible

```
1: for (i = 0 to populationSize-1) do
2:     randomizeDaysListIndex();
3:     randomizeTimePeriodsListIndex();
4:     chromosome[i]=graphColoringPart();
5:     computeFitnessValue(chromosome[i]);
6: end for
7: for (i = 1 to numberOfGenerations) do
8:     Schedule[] parents=getTwoSchedulesForMating();
9:     Schedule[] offspring=performCrossover(parents);
10:    Schedule offspring0=performMutation(offspring[0]);
11:    Schedule offspring1=performMutation(offspring[1]);
12:    survivalSelection(offspring0);
13:    survivalSelection(offspring1);
14: end for
```

3) Parent Selection: The evolution process is performed through several generations. The number of generations is a parameter that we control in the system. The type of genetic algorithm we use is incremental in which two chromosomes are selected for mating (Crossover) in each generation. We use roulette wheel selection for choosing the two chromosomes. In roulette wheel selection, a slice in a pie (wheel) represents the probability that a chromosome will be selected for mating. Since our optimization problem is a minimization problem, then the slice area is inversely proportional to the fitness value of the chromosome. In other words, a chromosome with lower fitness value (Better Schedule) occupies a larger slice in the pie. Parent selection step is represented in line 8 of the code in Algorithm 4 .

4) Crossover Operator: The two chosen chromosomes (Table III and Table IV) will undergo a crossover (mating) process. The type of crossover we use is a multi point crossover where two points a_1 and b_1 are chosen randomly from the range [0,z] where z is the length of chromosome (number of colors) and $b_1 > a_1$. Next, the section assignments in the middle section of the two parents are swapped (Table V and Table VI). Here we make sure that $\{b_1 - a_1 \leq z\}$ so that the swapping process is not performed over more than half of the chromosome length. Otherwise, the crossover process will turn into producing totally different chromosomes than its parents. Also, emphasizing this restriction decreases the cost of the crossover process which in turn increases the efficiency of the system. The swapping process generates cases where a section is assigned to two different slots (Conflict). For example, Table IX shows two conflicts in offspring 1 and two conflicts in offspring 2. In the case of a conflict, we use two ways of conflict resolution. We try the first one

TABLE III. PARENT 1

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
V_1,V_2	V_7	V_5,V_8	V_3	V_4,V_6	V_9	V_{11}	V_{10}

TABLE IV. PARENT 2

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
V_7,V_{10}	V_1	V_3	V_2,V_4	V_9	V_5	V_6,V_{11}	V_8

TABLE V. OFFSPRING 1 BEFORE CONFLICT RESOLUTION

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
V_1,V_2	V_7	V_5,V_8	V_2,V_4	V_9	V_5	V_{11}	V_{10}

TABLE VI. OFFSPRING 2 BEFORE CONFLICT RESOLUTION

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
V_7,V_{10}	V_1	V_3	V_3	V_4,V_6	V_9	V_6,V_{11}	V_8

TABLE VII. OFFSPRING 1 AFTER CONFLICT RESOLUTION

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
V_1	V_2,V_7	V_8	V_3,V_4	V_9	V_5	V_6,V_{11}	V_{10}

TABLE VIII. OFFSPRING 2 AFTER CONFLICT RESOLUTION

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
V_7,V_{10}	V_1	V_6	V_3	V_4	V_2,V_9	V_{11}	V_5,V_8

and if it does not work then we go for the second one. The first conflict resolution way is that we cancel the Section/Slot assignment that came from the same parent and we allow the new Section/Slot assignment coming from the other parent via the swapping process. This type of conflict resolution might fail because forcing a section's exam to be scheduled in a specific slot may violate exam scheduling constraints such as a student having two exams in the same day and time or exceeding the number of exams allowed for a student in a given day. In this case, we follow the second conflict resolution way in which we cancel the Section/Slot assignment that came from the same parent and use the graph coloring algorithm to schedule the section's exam in an acceptable slot that does not violate exam scheduling constraints. Notice that a new problem results after crossover and before conflict resolution. The problem is that few sections are missing from the offspring chromosomes. V_3 and V_6 are missing from offspring 1 and V_2 and V_5 are missing from offspring 2. We resolve this issue by calling the graph coloring routine after crossover is concluded which assigns appropriate slots for the missing sections. The new places for the missing sections are marked in blue color. The crossover step is performed at line 9 of the code in Algorithm 4. Tables VII and VIII show the final chromosomes (schedules) after executing the conflict resolution process.

5) Mutation Operator: In the mutation step, we use a mutation rate such that for each section we generate a random number between 0 and 1 and if the number is less than or equal to the mutation rate, then we cancel Section/Slot assignment. After performing this step for all sections, we run the graph coloring Algorithm 2 to find a slot for each section affected by the previous step which results in creating new Section/Slot assignments. Lines 10 and 11 carry out the mutation step.

6) Survival Selection: After crossover and mutation steps are concluded, we have two offspring chromosomes that represent valid exam schedules. So, we calculate the fitness value for each of them. If the fitness value for a given offspring chromosome happened to be less than the least fitness value among population chromosomes, then the offspring chromosome replaces that chromosome with the least fitness value. In the cases where fitness value of the offspring does not outperform any chromosome in the population, then the offspring is discarded. Survival selection is realized at lines 12 and 13 of the code in Algorithm 4 .

TABLE IX. CONFLICT RESOLUTION OF THE CONFLICTS AT TABLES V, VI

Num	Conflict	Conflict Resolution 1	Conflict Resolution 2
Offspring 1	V_2 in C_4 and C_1	-unassign(V_2,C_1) -assign(V_2,C_4) -Result: Fail	-unassign(V_2,C_1) -graphcoloring() -Result: (V_2,C_2)
Offspring 1	V_5 in C_6 and C_3	-unassign(V_5,C_3) -assign(V_5,C_6) -Result: Success	Not Needed
Offspring 2	V_3 in C_4 and C_3	-unassign(V_3,C_3) -assign(V_3,C_4) -Result: Success	Not Needed
Offspring 2	V_6 in C_5 and C_7	-unassign(V_6,C_7) -assign(V_6,C_5) -Result: Fail	-unassign(V_6,C_7) -graphcoloring() -Result: (V_6,C_3)

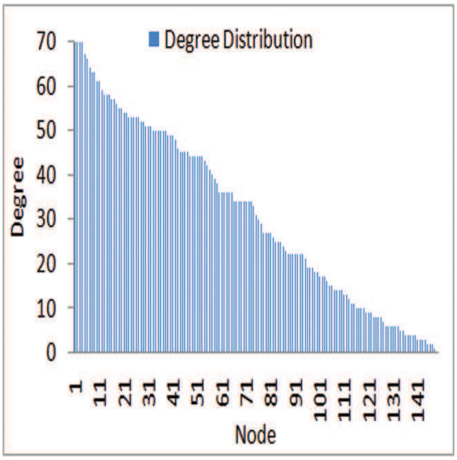

Fig. 3. Degree distribution of nodes.

V. EXPERIMENTS AND RESULTS

The algorithms we test in this section are as follows. The first one is the original hybrid approach we explained earlier (Original). The second one is a variation of the original algorithm in which we remove the crossover step and rely only on mutation (MutationOnly). The third one is the graph coloring algorithm (Graph). We implemented the aforementioned algorithms in an exam scheduling software developed using java language and MySQL database management system. The fitness value measured in the experiments refers to the number of students who have two exams in the same day throughout the examination period. For more accurate results, we perform 50 runs and take the average of the output numbers.

The dataset we employ in our experiments [24] is a new dataset that we propose for the research community. The dataset contains complete registration information that

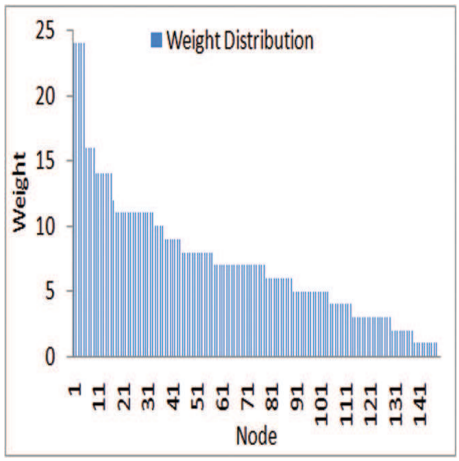

Fig. 4. Weight distribution for nodes.

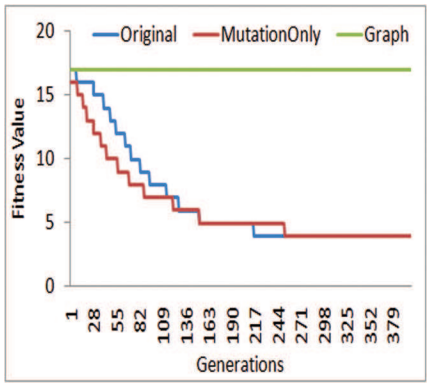

Fig. 5. Convergence of techniques.

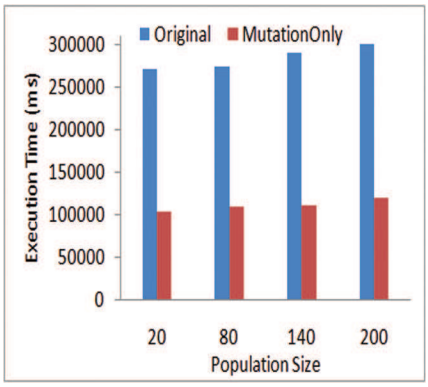

Fig. 6. Execution Time of techniques.

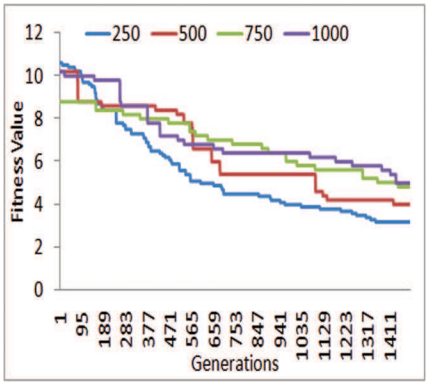

Fig. 7. Convergence speed for 'mutationOnly' when population size increases

reflects the real scenario during the 2017/2018 second semester in faculty of information technology at Isra University. Our dataset contains 89 courses, 143 sections, 24 instructors, 686 students, and 25 halls. Our experiments assume that exams take place during a span of 10 days such that each day contains 13 overlapping morning time periods and 3 evening time periods. In order to increase the degree of difficulty of finding valid exam schedules, we take into consideration the first and second exams and we exclude the final exams. This is because there are classes held during the first and the second exams, and therefore, the number of available halls in each time period is considerably less than the number of available halls during final exams which in turn makes scheduling harder. The process of finding valid exam schedules becomes more challenging when each section has common students with many other sections (Degree Increase). The challenge even escalates when the number of common students between each two sections is higher (Weight Increase). Therefore, in Fig. 3 and Fig. 4 we plot the degree distribution and weight distribution for sections. This helps in giving the reader a sense of the dataset under consideration. In Fig. 5, we aim to find which technique generates better schedules (lower fitness value). Here, we fix the number of generations at 500 and mutation rate at 0.5 and population size at 200. We found that the graph coloring technique is the worst because it depends on a greedy approach and hence it cannot explore the whole search space. On the other hand, both original and mutation only techniques are able to find more optimized schedules than graph coloring approach because of their genetic nature that helps to cover more area of the search space. However, looking back at the same Fig. 5, we notice that the mutation only solution converges faster than the original approach. In other words, the mutation only solution improves faster toward finding more optimized schedules. Consequently, we conclude that the mutation operator is more effective than the crossover operator in progressing toward more optimized schedules. In the second experiment illustrated in Fig. 6, we measure the execution time (milliseconds) of the original and mutation only techniques such that mutation rate is fixed at 0.5 and number of generations is fixed at 1500. We noticed that original approach is slower than the mutation only approach. This is due to the fact that crossover might generate conflicts and consequently time is spent to resolve those conflicts. This is why the crossover operator takes more time than the mutation operator that does not cause conflicts. Moreover, in the same figure we vary population size between 20 and 200 and the result shows that the execution time for both techniques increases when population size increases because more time is needed to generate the initial population. Also, both techniques require iterating through solutions of the population which takes more time as the population size increases. In the next experiment in Fig. 7 we investigate the effect of increasing population size (from 250 to 1000) on the speed of convergence for the mutation only algorithm such that we fix the mutation rate at 0.5 and number of generations at 1500. The figure shows that when population size decreases the algorithm converges

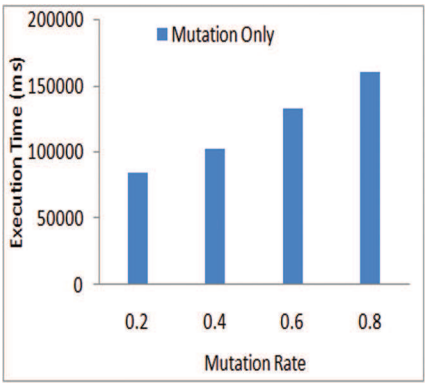

Fig. 8. Execution time for 'mutationOnly' when mutation rate increases.

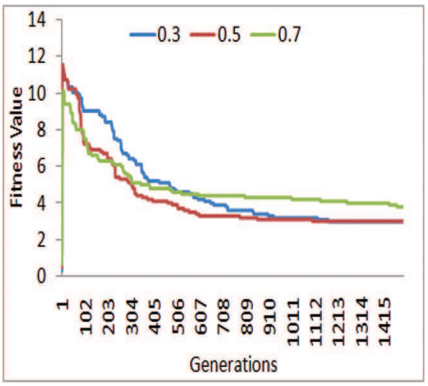

Fig. 9. Convergence speed for 'mutationOnly' when mutation rate increases.

faster. We noticed that when population size increases, the algorithm still manages to find the best possible exam schedule. However, it is interesting to point out that our experiments show that the fastest convergence occurred when population size equals to 1 which conforms with the result found in [19]. Next, we evaluate the effect of changing mutation rate on convergence and execution time. Fig. 8 shows that execution time increases when mutation rate increases which is the result of increasing number of operations that cancels the assignment of sections and makes scheduling decision for them again. In the previous experiment, population size is fixed at 200 and number of generations is fixed at 1500. We use the same fixed values in Fig. 9 in which we notice that increasing the mutation rate leads to faster convergence up to a point where increasing mutation rate generates reversible result (slower convergence). This actually makes sense because when the mutation rate increases the difference between the chromosomes before and after mutation becomes bigger. As a consequence, the chromosome after mutation becomes a totally new chromosome which would not preserve the quality genes in the parents which defies the concept of evolution. In our experiments, we obtained best results when mutation rate is 0.4. The previous set of experiments shows that combining graph coloring and genetic algorithms is effective in finding more optimized schedules. Also, the experiments indicate that the crossover operator is a costly operator and it is not as effective as the mutation operator.

VI. Conclusion and Future Work

In this paper, we proposed a hybrid approach that combines graph coloring and genetic algorithms for finding more optimized exam schedules. Our focus was to generate exam schedules in which we minimize the total number of students who have two exams in the same day. At the same time, we increased the gap between consecutive student exams. This helped in generating more comfortable exam schedule for students. Also, several types of realistic exam scheduling constraints were taken into consideration. Moreover, we evaluated our technique using realistic registration dataset. Our evaluation showed that relying on mutation instead of crossover leads to increased effectiveness and is not as computationally expensive. The hybrid approach generates promising results in terms of finding more optimized schedules that increase student comfort throughout the examination period. We relied on student comfort to measure the quality of schedules.by minimizing the number of students who have two exams in the same day. So, as a future work, we aim to expand student comfort by adding more details such as increasing the gap between exams for a given student. Moreover, we are planning to investigate more measures for deciding the quality of schedules and incorporate those measures in a realistic exam scheduling tool such that the user can pick the measure he needs.

References

[1] R. Ganguli and S. Roy, "A study on course timetable scheduling using graph coloring approach," *international journal of computational and applied mathematics*, vol. 12, no. 2, pp. 469–485, 2017.

[2] S. Saharan and R. Kumar, "Graph coloring based optimized algorithm for resource utilization in examination scheduling," *Applied Mathematics and Information Sciences*, vol. 10, no. 3, pp. 1193–1201, May 2016.

[3] N. G. N. Anand Nayyar, Dac-Nhuong Le, *Advances in Swarm Intelligence for Optimizing Problems in Computer Science*. Chapman and Hall/CRC, 2018.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[5] N. M. Gandhi and R. Misra, "Performance comparison of parallel graph coloring algorithms on bsp model using hadoop," in *2015 International Conference on Computing, Networking and Communications (ICNC)*, Feb 2015, pp. 110–116.

[6] H. T. Dao and S. Kim, "Vertex graph-coloring-based pilot assignment with location-based channel estimation for massive mimo systems," *IEEE Access*, vol. 6, pp. 4599–4607, 2018.

[7] B. Hardy, R. Lewis, and J. Thompson, "Tackling the edge dynamic graph colouring problem with and without future adjacency information," *Journal of Heuristics*, vol. 24, no. 3, pp. 321–343, Jun 2018.

[8] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Effective and efficient dynamic graph coloring," *Proceedings of the VLDB Endowment*, vol. 11, no. 3, pp. 338–351, Nov. 2017.

[9] H. Patidar and D. P. Chakrabarti, "A novel edge cover based graph coloring algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 5, 2017.

[10] I. Ashiru, C. Czarnecki, and T. Routen, "Characteristics of a genetic based approach to path planning for mobile robots," *Journal of Network and Computer Applications*, vol. 19, no. 2, pp. 149–169, 1996.

[11] J. Oh and C. Wu, "Genetic-algorithm-based real-time task scheduling with multiple goals," *Journal of Systems and Software*, vol. 71, no. 3, pp. 245–258, 2004, computer Systems.

[12] C. Kalayci and A. Gungor, "A genetic algorithm based examination timetabling model focusing on student success for the case of the college of engineering at pamukkale university, turkey," *Gazi University Journal of Science*, vol. 25, pp. 137 – 153, 2011.

[13] M. Saidi-Mehrabad and S. Bairamzadeh, "Design of a hybrid genetic algorithm for parallel machines scheduling to minimize job tardiness and machine deteriorating costs with deteriorating jobs in a batched delivery system," *Journal of Optimization in Industrial Engineering*, vol. 11, no. 1, pp. 35–50, 2018.

[14] R. L. Kadri and F. F. Boctor, "An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case," *European Journal of Operational Research*, vol. 265, no. 2, pp. 454 – 462, 2018.

[15] K. Kaur, N. Kaur, and K. Kaur, "A novel context and load-aware family genetic algorithm based task scheduling in cloud computing," in *Data Engineering and Intelligent Computing*. Singapore: Springer Singapore, 2018, pp. 521–531.

[16] M. A. Mohammed, M. K. A. Ghani, R. I. Hamed, S. A. Mostafa, M. S. Ahmad, and D. A. Ibrahim, "Solving vehicle routing problem by using improved genetic algorithm for optimal solution," *Journal of Computational Science*, vol. 21, pp. 255 – 262, 2017.

[17] M. R. Mirsaleh and M. R. Meybodi, "A michigan memetic algorithm for solving the vertex coloring problem," *Journal of Computational Science*, vol. 24, pp. 389 – 401, 2018.

[18] S. M. Douiri and S. Elbernoussi, "Solving the graph coloring problem via hybrid genetic algorithms," *Journal of King Saud University - Engineering Sciences*, vol. 27, no. 1, pp. 114 – 118, 2015.

[19] A. Eiben, J. van der Hauw, and J. van Hemert, "Graph coloring with adaptive evolutionary algorithms," *Journal of Heuristics*, vol. 4, no. 1, pp. 25–46, Jun 1998.

[20] C. A. Glass and A. Prügel-Bennett, "Genetic algorithm for graph coloring: Exploration of galinier and hao's algorithm," *Journal of Combinatorial Optimization*, vol. 7, no. 3, pp. 229–236, Sep 2003.

[21] Z. Kokosiński, M. Kołodziej, and K. Kwarciany, "Parallel genetic algorithm for graph coloring problem," in *Computational Science - ICCS 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 215–222.

[22] M. Hindi and R. V. Yampolskiy, "Genetic algorithm applied to the graph coloring problem," in *Proceedings of the 23rd Midwest Artificial Intelligence and Cognitive Science Conference*, 2012, pp. 60–66.

[23] W. Erben, "A grouping genetic algorithm for graph colouring and exam timetabling," in *Practice and Theory of Automated Timetabling III*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 132–156.

[24] [dataset] Osama Al-Haj Hassan, "Student registration dataset of faculty of information technology at isra university, amman, jordan, mendeley data, v1," 2018.