Enhancing The Performance of the Greedy Algorithm Using Chicken Swarm Optimization: An Application to Exam Scheduling Problem

Taha M. Mohamed

Information Technology Dept., Faculty of Computers and Information,
Helwan University, Egypt
Tahamahdy3000@yahoo.com

Abstract

Exam scheduling, or exam timetabling, is one of the most important challenges of credit hour systems. Credit hours system are characterized by their large number of courses, students, and professors. However, there are many constraints regarding exam time tabling. The time tabling problem is a well known graph coloring problem. In this paper, we present a novel approach for solving the exam time tabling problem by enhancing the performance of the greedy algorithm using by modifying the chicken swarm optimization (CSO). In the proposed approach, the modified CSO is used to optimize the ordering of vertices fed into the greedy algorithm. Results show that, statistically wise, using the proposed approach, significant enhancements are obtained compared to using manual timetabling or even using the greedy algorithm individually. Moreover, the soft constraints are considered on the proposed approach using a proposed objective function. From time perspective, the execution time of the algorithm is approximately 1.5 minutes on the average, even in case of dense graphs. Furthermore, the cost function of the proposed approach is significantly less than that of using greedy algorithm only.

Keywords: *Exam Scheduling, Exam Time Tabling, Chicken Swarm Optimization, Greedy Algorithm.*

1.Introduction

Credit hour systems are currently implemented in most universities and institutions. In Egypt, many Egyptian universities recently implement this system in the faculties. One important problem of the credit hour system is the problem of exam time tabling, or exam scheduling. The timetabling problem is always a difficult task which comes up every calendar year in educational institutions, especially if it has to be done manually [1]. Often, there is a large number of students and courses that have to be scheduled against existing limited resources [1]. In practice, universities will often have predefined number of timeslots in their timetable and their task will be to determine a feasible time table using fewer or equal timeslots [2]. The course and exam timetabling are relatively close problems, however, significant differences exist between them [3].

In Egypt, for example, lot of institutions perform the time table manually. The process of manual scheduling of exam time tabling consumes a lot of time and efforts due to the constraints involved in time table scheduling. In general, the importance of these constraints are not the same for all institutions [1], [2]. For example, in our faculty we are interested in designing a feasible time table having the minimum number of time slots. An advantage to the time table is that, the time table can consider students preferences. Studies show that, the exam time table may affect students performance. The authors in [4], made a survey of

student views finding that, over half of the students were unhappy with their examination timetables [4]. Furthermore, about 30% of respondents believed that, their examination timetable negatively affected their academic achievement [4]. The quality of the timetabling has a great impact on different stakeholders including lecturers, students and administrators [3].

When designing exam time tabling, there are many constraints that should be taken into consideration. One and perhaps the most important constraint is the avoidance of conflicts between exams. Another constraint is the limitation of the available exam slots. Another limitation is related to the availability of exam rooms and human resource. The previous constraints are called hard constraints that must be satisfied and cannot be violated [2]. Additionally, there are some soft constraints that are preferable to be satisfied such as students and institution preferences. For example, it is preferred to maximize the time interval between two consecutive exams for the majority of the students. Actually, the quality of an examination timetable is measured by the extent of the soft constraint satisfaction [4]. The problem of university exam timetabling could be modeled as a graph coloring problem [2]. The problem involves assigning a set of events (exams) to a fixed number of colors (timeslots). The conflicting events need to be assigned to different timeslots. In graph coloring problems, a course is considered as a vertex, with edges connecting two courses are the conflicts between them. Each color represents a timeslot. A feasible coloring corresponds to a complete timetable with no conflict violations [2].

A well known solution to the graph coloring problem is the greedy Algorithm. The greedy algorithm is one of the constructive algorithms that guarantees satisfying one important constraint of the hard constraint (the conflicts) [2]. One important problem of the greedy algorithm is; the greedy algorithm doesn't consider the satisfaction of soft constraints. So, the exam time tabling problem is a typical optimization problem.

In this paper, we propose a new approach for enhancing the performance of the greedy algorithm using a modified version of the chicken swarm optimization (CSO). In this new approach, we are satisfying the hard constraints using the greedy algorithm, and hence, avoiding the large searching domain. On the same time, the main problem of the greedy algorithm is solved. A new objective function is proposed to measure the quality of the generated exam time table. This paper is organized as follows; in section 2, we present the necessary background. In section 3, we present the related work. In section 4, the proposed algorithm is introduced. In section 5, the experimental results are introduced and discussed. The paper is concluded in section 6.

2. Background

2.1 Exam Time Tabling As a Graph Coloring Problem

Exam timetable is a difficult task in most faculties. The staff involved in the scheduling process have a lot of difficulties due to the large number of students and courses that are required to be scheduled against limited recourses [1]. There are multiple constraints involved in timetable design. These constraints could be classified into hard constraints and soft constraints. The main hard constraint is that, no exams with common resources could be carried out simultaneously. The other hard constraints may be related to the limitation of time slots and exam rooms [3]. Hard constraints have more higher priority over soft constraints. Timetables are considered as "feasible" if all of the hard constraints are satisfied [2]. However, the soft constraints are desirable but are not absolutely critical. In practice, it is

usually impossible to find feasible solutions that satisfy all of the soft constraints. Soft constraints vary between various institutions in terms of both types and importance [3]. The quality of timetables is usually measured by soft constraints satisfaction [3].

The graph coloring problem could model exam time tables. The vertices of the graph represent courses, while graph edges represent the conflicts [2]. Let $G = (V, E)$ be a graph, consisting of a set of *n* vertices *V* and a set of m edges *E*. Given such a graph, the graph coloring problem tries to assign each vertex $v \in V$ an integer $c(v) \in \{1,2, \ldots, k\}$ such that: $c(v) \neq c(u) \; \forall \{v, u\} \in E$; and *k* is minimal [2].

The adjacency matrix, of the graph, is a rectangular matrix *A* with dimension *n x n* for which $A_{ij} = 1$ if and only if vertices v_i and v_j are adjacent, and $A_{ij} = 0$ otherwise [2]. In exam timetabling problem, the adjacency matrix may be considered as the a binary version of the conflict matrix *C*. We define the conflict matrix *C* as the matrix containing all conflicts between courses. That is *C* is an *n x n* matrix such that C_{ij} represents the total number of students registering both courses *i* and *j*. the conflict matrix is a very important term in the proposed objective function in satisfying both hard and soft constraints. The chromatic number of a graph *G*, denoted by $\chi(G)$, is the minimum number of colors required to get a feasible coloring of *G* [2], [3].

The density of graph *G* is the ratio between the number of edges to the number of vertices. Graphs having low densities are referred to as sparse graphs. Graphs having high densities are called dense graphs. The density of *G* could be calculated as:

$$D(G) = m/((n(n-1))/2). \quad (1)$$

The constructive algorithms, such as the Greedy algorithm, are the classical solutions to graph coloring problems. These algorithms ensure the satisfaction of hard constraints. In these algorithms, the exams are ordered by some heuristic. The exams are then assigned, one by one, into the timeslots [3]. These algorithms are simple and powerful. They could provide a reasonable solution within a small amount of time and they are easy to implement. They are often used to construct initial solutions to be further enhanced [3].

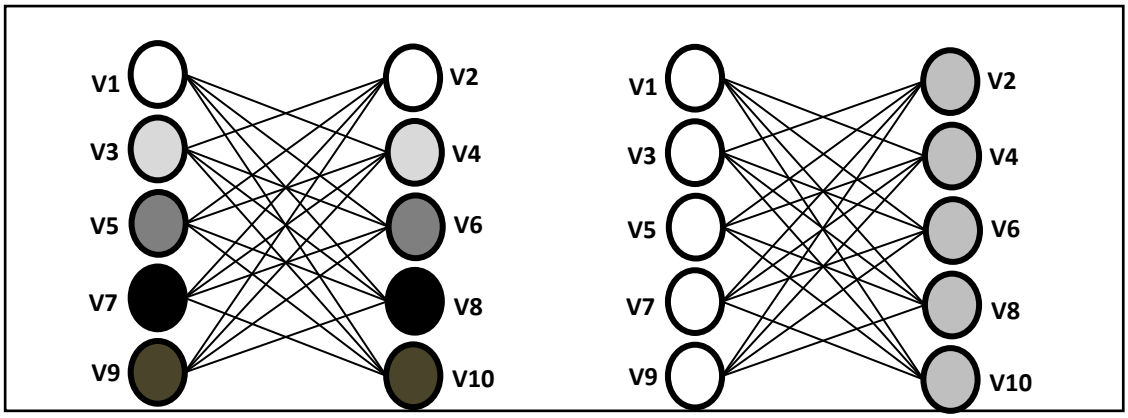
The meta-heuristic optimization approaches, proposed to solve graph coloring problems, are classified into two categories: one or two stage algorithms. In the one stage algorithms, the satisfaction of both hard constraints and soft constraints is attempted simultaneously. Then, the violations of hard constraints are penalized more heavily than violations of soft constraints [2]. The two-stage algorithms, however, satisfy hard constraints first forming a feasible solution. Next, the soft constraints are optimally being satisfied by navigating the space of feasible solutions [2]. Our proposed approach is a two stage algorithm that use the greedy algorithm first to get a feasible solution. Next, a modified CSO is used to enhance this initial feasible solution.

2.2 The Greedy Algorithm

The greedy algorithm is one of the simplest, but most fundamental, heuristic algorithms for graph coloring. The algorithm takes vertices one by one according to some ordering and assigns each vertex its first available color. The greedy algorithm can produce an optimal solution for any graph given the correct sequence of vertices [2]. In time tabling problems, the greedy algorithm works as follows; the courses are fed into the greedy in some ordering (permutation) of the courses one by one in the first available time slot. If there is a conflict

between the current course and any other course in the current time slot, then, a new slot is generated, and the current course is added to this slot. Otherwise, the current course is added to the current time slot. The process is repeated until finishing all courses.

It is interesting to note that, the courses ordering, fed to the greedy algorithm, is very important that affects the quality of the generated time table. For example, figure 1 shows two different ordering (permutations) of vertices [2]. In figure 1 (a), the permutation $\pi = (v_1, v_2, v_3, \ldots, v_{10})$ leads to five colors solution. However, a permutation of the form $\pi = (v_1, v_3, \ldots, v_9, v_2, v_4, \ldots, v_{10})$ produces only two colors, on the same graph, shown in Figure 1(b).



(a) five different colors (b) Two different colors

Figure 1. Different Order of Vertices in the Greedy Algorithm [2]

In the proposed approach, we propose a modified CSO to select the optimal ordering of vertices (exams) that fed to the greedy algorithm. A complete description of the greedy algorithm is shown in algorithm1 listing [2]. In the algorithm π represents the permutation fed to the algorithm, *S* refers to the solution [2].

$Algorithm1: GREEDY\ (S \leftarrow \emptyset, \pi)$

(1) $for\ i \leftarrow 1\ to\ |\pi|\ do$
(2) $\quad for\ j \leftarrow 1\ to\ |S|$
(3) $\quad\quad if\ (S_j \cup \{\pi_i\})\ is\ an\ independent\ set\ then$
(4) $\quad\quad\quad S_j \leftarrow S_j \cup \{\pi_i\}$
(5) $\quad\quad\quad break$
(6) $\quad\quad else$
(7) $\quad\quad\quad j \leftarrow j + 1$
(8) $\quad if\ j > |S|\ then$
(9) $\quad\quad S_j \leftarrow \{\pi_i\}$
(10) $\quad\quad S \leftarrow S \cup S_j$

2.3 Chicken Swarm Optimization (CSO)

Chicken swarm optimization is an optimization technique that mimics chickens swarm behaviors. It is proposed by Meng. et. al. [5]. CSO performance is compared with other optimization techniques on twelve benchmark problems. CSO outperforms these techniques in terms of accuracy and robustness. The chicken swarm is divided into several groups. Each group consists of a rooster, hens, and chicks. Different chickens follow different laws of motions. The identity of the chickens (roosters, hens and chicks) are depending on the groups.

The chickens with best several fitness values would be acted as roosters. The chickens with worst several fitness values would be designated as chicks. The others would be the hens. The hens randomly choose groups to live in. Chickens follow their group-mate rooster to search for food. They may prevent the ones from eating their own food [5]. The equations from 2 to 6 describe the motion of the chickens [5].

$$X_{i,j}^{t+1} = X_{i,j}^{t} * (1 + Randn(0, \sigma^2)) \quad (2)$$

$$\sigma^2 = \begin{cases} 1, & if\ f_i \le f_k \\ \exp(\frac{f_k - f_i}{|f_i| + \varepsilon}), & otherwise \end{cases} \qquad k \in [1, N], k \neq i \quad (3)$$

$$X_{i,j}^{t+1} = X_{i,j}^{t} + S1 * Rand * \left(X_{r1,j}^{t} - X_{i,j}^{t}\right) + S2 * Rand * \left(X_{r2,j}^{t} - X_{i,j}^{t}\right) \quad (4)$$

$$S1 = \exp((f_i - f_{r1})/abs(f_i) + \varepsilon)) \quad (5)$$

$$S2 = \exp((f_{r2} - f_i)) \quad (6)$$

$$X_{i,j}^{t+1} = X_{i,j}^{t} + FL * (X_{m,j}^{t} - X_{i,j}^{t}) \quad (7)$$

The N virtual chickens, illustrated by their positions $X_{i,j}^{t+1}$, such that $i \in [1, 2, \ldots, N]$, $j \in [1, 2, \ldots, D]$ at time t. The $Randn$ is a Gaussian random generator. f is the fitness value of the corresponding x [5]. Algorithm 2 listings describes the original CSO algorithm [5].

***Algorithm 2: Original CSO Algorithm* [5]**

```
Initialize a population of N chickens and define the related parameters
Evaluate the N chickens' fitness values, t = 0;
While (t < MAx_Generation)
   If (t%G == 0)
      Rank the chickens' fitness values and establish a hierarchal order in the swarm;
      Divide the swarm into different groups, and determine the relationship between chicks hens in a group;
   End if
   For i = 1: N
      If i == rooster  Update its solution/location using equation(2); End if
      If i == hen      Update its solution/location using equation(4); End if
      If i == chick    Update its solution/location using equation(7); End if
      Evaluate the new solution;
      If the new solution is better than its previous one, update it;
   End for
End while
```

3.Related Work

In [3], the authors present an excellent survey of the research on exam timetabling in the last decade. In [4], the authors present the results of a survey amongst students concerning their own preferences for particular properties of examination timetables. The survey shows that fairness is indeed a concern for them. In [2], the authors discuss the graph coloring problem and its constructive algorithms, the greedy, DSATUR, and RLF algorithms.

Recently, several meta-heuristic methods are used to solve the time tabling problem. For example, in [1], the authors propose a genetic algorithm technique to develop a timetable. In [6], the authors present a meta-heuristic technique to solve the exam timetabling problem. They propose a non-linear formulation used to find an initial feasible solution. In [7], the

authors present a hybridization of the ants colony algorithm and local search heuristic. The hybridization is used to maximize the free time between consecutive exams for each student. In [8], a multi-objective optimization program was proposed to handle spreading of exams criteria and a Tabu Search was implemented to find a good feasible solution. In [9], the authors construct a starting feasible timetable using a simple graph coloring heuristic. Simulated Annealing is then used as an iterative heuristic for improving the spreading of the conflicting exams. In [10], the authors present the regulatory algorithm (RGA) which is a two dimensional extension of standard evolutionary algorithms. The algorithm is applied to exam timetabling in the University of Duisburg-Essen. The genetic Algorithm is used to analyze conflicts only of the time table. In [11], the authors treat the timetable problem as a single objective optimization problem. The space of both valid and invalid solutions is explored. Simulated annealing is then used to optimize this objective function.

Regarding chicken swarm optimization (CSO), Meng et. al. [5], present the recent CSO that mimics chickens behaviors. The authors compare the performance of CSO with that of particle swarm (PSO), Differential evolution (DE) and bat algorithm (BA) on twelve benchmark problems. Their experiments show that CSO outperforms these algorithms in terms of optimization accuracy and robustness. In [12], the authors propose a CSO based algorithm for feature selection. The results outperform other optimization methods such as particle swarm optimization (PSO) and genetic algorithm. In [13], the authors use CSO to solve extremely challenging non-convex economic load dispatch problem. In [14], the authors use CSO to solve the clustering routing protocol of wireless sensor networks. Results show that, CSO is better than PSO in solving this problem. Some improvements on the original CSO have been proposed. For example, in [15], the authors proposed an improvement to CSO to overcome the local minimum problem. Again, CSO outperforms particle swarm optimization, bat algorithm, and original chicken swarm optimization. In [16], a modified CSO algorithm is proposed for global optimization. This modification reduces CSO steps. In [17], the authors present an improved chicken swarm optimization algorithm based on elite opposition-based learning to avoid the possibility of local minimum.

It is clear from the previous literature survey that, till now, CSO is not used in the graph coloring problems. Considering the novelty and superiority of CSO over the other optimization algorithms, we use CSO to enhance the performance of the greedy algorithm in graph coloring problems.

4. The Proposed Approach

In this section, we will explain the proposed approach that efficiently solves the graph coloring problem focusing on one of the most important applications of graph coloring which is the automatic exam timetabling problem. As we explain before, the greedy algorithm could guarantee an optimal solution(s), to the graph coloring problem if a proper order of the vertices is given. The main advantage of using the greedy algorithm is the guaranteeing of satisfying the main hard constraint of the timetabling problem regarding exams conflicts. However, improper ordering (permutation) of vertices leads to bad solutions from the perspective of soft constraints. The random ordering (permutation) of vertices is a possible solution to the greedy problem. This process may be conducted several times to choose the most optimal solution. However, there is no guarantee of reaching the global optimal solution in this case. In fact, our results prove this concept as we will show in the experimental results section.

To overcome this problem, in the proposed approach, we represent a more better solution for guarantying reaching the global minimum (the optimal solution) by using CSO to find an optimal ordering of vertices that is fed to the greedy algorithm. Additionally, the usage of CSO guaranteeing also satisfying the second hard constraint regarding the minimum number of time slots generated. Another advantage of using CSO is the ability of satisfying one of the important soft constraint (the students preferences).

Unfortunately, the original implementation of CSO generates a vector of real numbers rather than integers [5]. So, we propose a modification to the original CSO. Consider S_i is a solution vector of size *n*, generated from the original CSO. That is $S_i = [s_1, \ldots, s_n]$. Such that, S_i is a real number. We need a mapping function *MAP* to convert *S* to a new modified integer vector S_M of the same size such that, S_M (i) ≠ S_M (j) ∀ {i, j}, $i, j \in \{1, \ldots, n\}$.

S_M (i) ≤ n ∀ i. In fact, There are infinite number of mapping functions that satisfy the previous requirements. However, we propose a simple mapping function that described in algorithm 3 listings.

Algorithm 3**: **Mapping

Input: *Real vector of S of size n*
Output: *Integer vector SM of size n*

1. $I = 0$
2. *While not end of array*
3. $Min = Minimum\ (S)$
4. $Min\_index = index\ of\ Minimum\ of\ (S)$
5. $SM(Min\_index) = i$
6. $S\ (Min\_index) = very\ large\ number;$
7. $I = i + 1$
8. *End while*

The proposed mapping algorithm searches for the minimum element of *S* and gives it index *i*; $0 \leq i < n$. Then this element is converted to a very large number to avoid choosing it again. The process is repeated until all values in *S* are mapped to integers. This simple mapping function guarantees generating a semi random permutation from 0 to *n-1* (the number of courses) without repeating. This cost of this permutation is measured by the objective function. In the next CSO iteration, a new permutation is used to reach the optimal solution. A numerical example of the map function is shown in the table below.

Old vector (real values)				
3.2	0	0	1.2	8

New vector (integer values)				
4	1	2	3	5

To measure the cost of a solution, we propose a new objective function. The proposed objective function is a minimization function aims to consider three constraints; the first is the hard constraint regarding scheduling of conflicted courses. The second semi-hard constraint is the number of time slots generated by the time table. The final constraint is the students preferences regarding maximization of the interval between consecutive exams with respect to all students. This proposed objective (cost) function, of a solution *sol*, is given by equation 8.

$$Minimize\ Cost(sol) = f1.|sol| + \Sigma(f2.D(s_i, s_j) - f3.abs(i-j)) \quad (8)$$

The first term of equation 8 represents the length of the time table in slots $|sol|$. This term is weighted by a weighting factor f_1. The second term of the equation have two sub terms weighted by factors f_2, f_3. The first sub-term $D(s_i, s_j)$ measures the distance between any two slots in the time table s_i, s_j. the second sub-term is the absolute difference in slots between the measured slots. The second term is repeated for all time slots generated in the time table, and the total summation is computed.

To measure the cost of a time table, let the generated time table, *sol*, has *n* slots that is $sol=\{s_1, \ldots, s_n\}$. Each slot s_i may contains q scheduled courses, that is $s(i) =\{c_1, \ldots c_q\}$. We define the distance D between two time slots s_i, s_j as: $D = \Sigma\, Con(c_x, c_y)$, such that c_x is a course ϵ to slot s_i, and c_y belongs to slot s_j $\forall$ c_i, c_k. Here, *Con* is the conflict matrix which contains the number of student registering the two courses c_x and c_y.

The objective of computing the distance between time slots is to find the total number of students common on two slots. If these two slots are near to each other, then they are penalized, and the cost function is increased. However, if these two slots are far away, from time perspective, then, the cost is decreased. This is controlled by the second sub-term $abs(i-j)$. In general, these distances are largely depending on the number of courses and the conflict matrix. So, weighting factors are required to balance the cost function. The main advantage of this objective function, compared to other methods, is the flexibility in adding, or removing, constraints according to university requirements without affecting the main hard constraint (the conflicts between time slots). The proposed approach is shown in Figure 2, and the algorithm is shown in algorithm 4 listings.

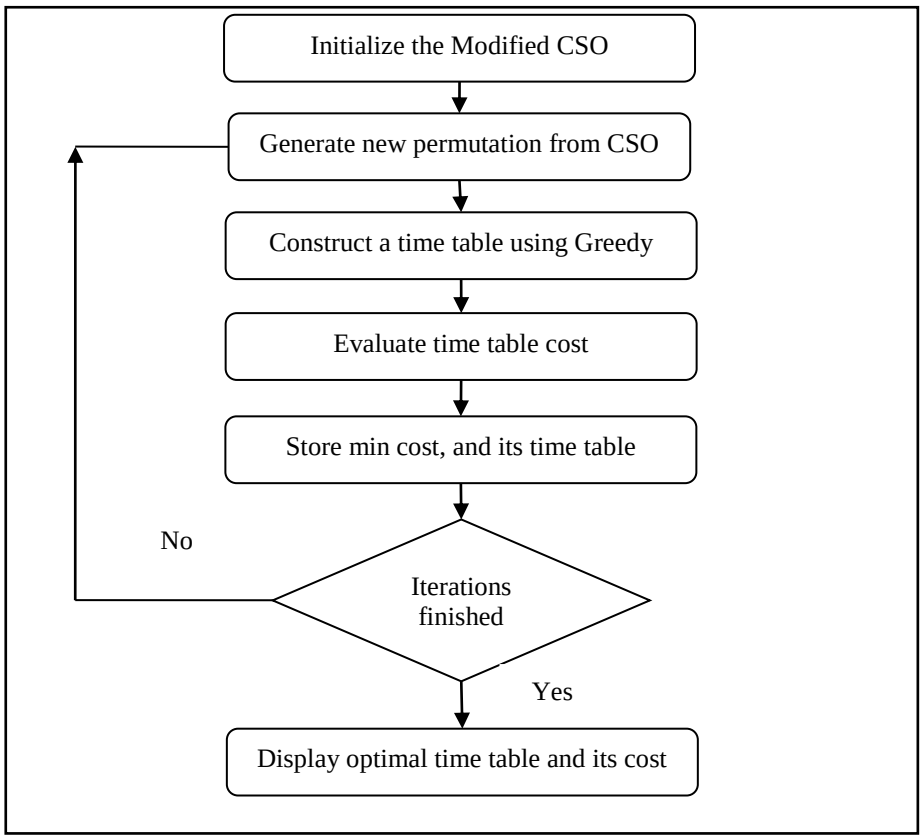


Figure 2. The Flowchart of the Proposed Approach

Algorithm 4: $CSO - Greedy\ Approach$

$Input$: $conflict\ matrix$
$Output$: $optimal\ time\ table$

1. $Min\_cost\ =\ very\ large\ number$
2. Do
3. $RND\ =\ random\ permutation\ using\ the\ new\ map\ function\ from\ CSO$
4. $Solution\ =\ greedy\ (RND)$
5. $c\ =\ cost\ (Solution)$
6. $if\ c < Min\_cost\ then$
7. $Min\_cost = c\ and\ optimal\_time\_table\ =\ solution$
8. $End\ if$
9. $Enhance\ the\ solution\ by\ generating\ new\ permutation$
10. $While\ number\ of\ iterations\ finished$

5. Experimental Results and Discussion

In this section, we will present the experimental results of the proposed approach. We use MS Excel 2007, and Matlab R2012a for testing. The data set used consists of 12 different exam time tables. The dataset include two different academic programs in Faculty of computers and information, Helwan University; the general program (*Gen*), and the software engineering program (*SWE*). Table 1 describes the part of the dataset regarding the general program. Table 2 shows the second part of the dataset regarding the SWE program. For each program of the two programs, three consecutive academic years are considered. Each academic year include two consecutive semesters; the fall semester and the spring semester. The first column, of the two tables, show the semesters included in the experiments. The second column shows the number of courses taught in the semester. The third column shows the graph density, computed by using equation 1. The average number of courses, and the average of the graph density, are also shown in the table. Moreover, the correlation between the number of courses and the densities are also shown.

The data set, of the two programs, are initially separated into two tables to test the variation between them. It is noted that from table 1, the average number of courses is 43 course, while the average graph density is 0.72. Meaning that, the graphs are too dense although of the small number of courses. It is also noted that, the correlation between the number of courses and the graph density is a strong negative correlation. That means, as the number of courses decreased, the links, and the conflicts, between these courses are increased.

Table 1. Datasets Description for General Program

Semester	Number of courses	Graph Density
Gen Fall 14	44	0.70
Gen Spring 15	48	0.70
Gen Fall 15	43	0.75
Gen Spring 16	48	0.67
Gen Fall 16	40	0.80
Gen Spring 17	39	0.75
Avg.	**43.66**	**0.72**
Corr.	**-0.8467**	

Table 2 shows the same information of table 1, but now for SWE program. Here, the average number of courses is 33.5. That is, the average number of courses is less than the average number of courses in the general program. The average graph density is 0.53, which is also less than that of the general program. Again, the correlation in Table 2 is a strong negative correlation, meaning that, as the number of courses increased, the graph density decreased strongly.

Table 2. Datasets Description for SWE Program

Semester	Number of courses	Graph Density
SWE Fall 14	32	0.58
SWE Spring 15	33	0.55
SWE Fall 15	34	0.52
SWE Spring 16	44	0.40
SWE Fall 16	31	0.64
SWE Spring 17	27	0.54
Avg.	**33.5**	**0.53**
Corr.	**-0.79**	

Table 3 shows the values of the various parameters used in the experiments. The first column of the table shows the number of population, the number of chickens, used in the chicken swarm optimization (CSO) that represents the number of roosters, hens, and chicks. The second column shows the number of iterations used.F_1, F_2, F_3 are the three scaling factors used in the proposed objective function. The population number and the number of iterations are randomly chosen, whereas the objective function parameters are chosen by trial and error. it should be noted that, the parameters of the objective function are highly depending on the dataset itself and the values of the conflict matrix *C*. Changing these values highly affects the amount of soft constraints satisfaction, and also the number of slots produced.

Table 3. Parameters for Soft Constraints and Pop Parameters

POP.	ITERATIONS	F_1	F_2	F_3
50	50	20	0.01	10

Table 4 shows the difference in performance between various methods that used in time table scheduling. In this table, we compare the quality of time tables produced by the proposed approach, manual, and greedy algorithm alone, in scheduling time tables. The first column shows the semester used. The second column counts the number of already produced time slots obtained from the manual time table scheduling. The third column shows the time slots obtained, from the proposed approach, considering the hard constraints only. The fourth column of the table shows the results of the proposed approach considering both hard and soft constraints. The last column of the table shows the performance measured from using the greedy algorithm individually by testing the greedy algorithm performance using 100 trials with different permutation. In this case, the result of the best one solution, produced by the greedy algorithm, are chosen.

Analysis of the table shows that, the manual scheduling, of the time tabling, is the worst technique used as the number of scheduled time slots is always greater than, or equal to, the number of slots of the other techniques. This is clear from the average number of time slots which equals 20.75, which exceeds all other tested techniques. On the other hand, the best average number of slots is obtained from using the proposed approach considering hard constraints only. The average number of slots in this case is 17.66. So, the proposed approach outperforms the other tested techniques regarding the number of generated time slots. Additionally, the proposed approach outperforms 100 trials of permutations fed to the greedy algorithm. As expected, adding more constraints (the soft constraints) will maximize the objective function value, causing more increase on the average number of generated time slots. This is shown in the fourth column. Here, the average number of generated time slots is 18.33, which is also better than the manual scheduling of the table. However, it is slightly larger than using greedy algorithm only, taking into consideration that, the greedy algorithm doesn't consider the soft constraints, as shown in Table 5. It is also clear that, the number of time slots could be decreased, in the case of soft constraints, by changing the objective function parameters giving more importance to the number of time slots.

Table 4. Number of Slots Obtained from Various Techniques

Semester	*Manual*	*CSO (Hard)*	*CSO (Hard and Soft)*	*greedy (100)*
Gen Fall 14	23	21	23	22
Gen Spring 15	21	21	21	21
Gen Fall 15	27	27	28	27
Gen Spring 16	23	22	24	23
Gen Fall 16	28	22	22	22
Gen Spring 17	26	20	22	20
SWE Fall 14	17	14	14	14
SWE Spring 15	17	14	14	15
SWE Fall 15	15	13	13	13
SWE Spring 16	16	14	15	15
SWE Fall 16	24	13	13	14
SWE Spring 17	12	11	11	12
Avg.	**20.75**	**17.66**	**18.33**	**18.16**

The results of table 4 are statistically analyzed using three statistical tests [18]. All tests are based on the following two hypotheses:

$$\boldsymbol{H0}: \mu 1 = \mu 2 \ , \ \boldsymbol{Ha}: \mu 1 \neq \mu 2$$

The null hypotheses H_0 states that, there is no statistical differences between the two population means. The alternative hypothesis, H_a, states that, the population means are different. The first statistical test, we conducted, is a one tail paired t-test, with confidence 95% [18], between the manual scheduling of the time table and the proposed approach using hard constraints only. This test shows that, t_{stat} equals 3.3, $t_{critical}$ equals 1.7. Here, t_{stat} is greater than $t_{Critical}$. So, the decision is to reject the null hypothesis [18]. That is, the time table, scheduled using the proposed approach, is better than the manual scheduling of the time table taking into consideration the number of generated time slots.

The second conducted test is also the paired t-test between the hard constraints and the soft constraints of the algorithm. The test shows that, t_{stat} equals -2.6 and $t_{critical}$ equals 1.7. So,

t_{stat} is less than $t_{Critical}$. So, the decision is to accept the null hypothesis. This means, there is no statistical differences between the time tables generated, using the proposed approach, either when considering hard constraints or soft constraints, regarding the number of generated time slots. In this case, we benefit from satisfying the soft constraints without a significant increase in the generated time slots.

The third conducted experiment is the nonparametric Wilcoxon signed rank test [18]. This test is used to measure the statistical difference between the output of the proposed approach, and the output of the greedy algorithm using 100 random iterations. A first look at the results obtained from Table 4 shows that, six semesters are enhanced, using the proposed approach, over using the greedy algorithm. The output of the other six semesters are the same using the two algorithms. This result is supported by using the Wilcoxon signed rank test. In the test, $w_{computed}$ equals 0 and $w_{critical}$ equals 2. So, the decision is to reject the null hypothesis. That is, the $w_{computed}$ of the signed rank test Wilcoxon is less than $w_{critical}$. This means, using the proposed approach is better than using 100 trials of the greedy algorithm. Furthermore, a graphical plotting of Table 4 is shown in Figure 3. It is very clear from the figure that, the manual scheduling of the time table is the worst option regarding the number of generated time slots. The figure also shows that, the greedy algorithm is better than the manual scheduling. Finally, the proposed approach outperforms the two other techniques.

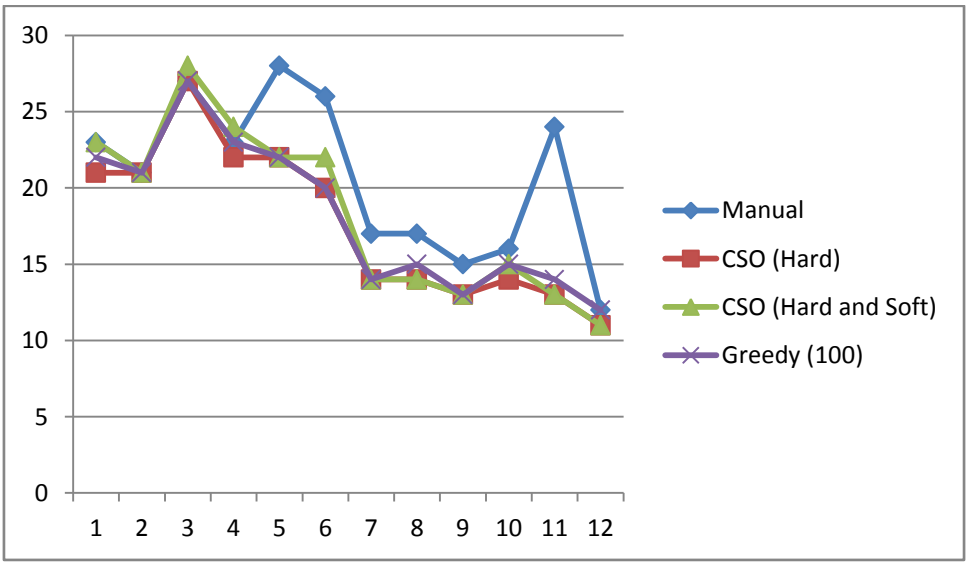


Figure 3. Comparison of Generated Time Slots

Table 5 shows the total cost, of the objective function, measured between the proposed approach considering the soft constraints and the minimum cost of using greedy algorithm individually. Here, the cost of the greedy algorithm is the minimum cost obtained in the 100 tested trials. From the table it is clear that, the average cost function of the proposed algorithm is 65.27 whereas the average cost function of the greedy algorithm is 89.83. Meaning that, a large reduction of the cost function is obtained using the proposed approach. Additionally, in all cases, the cost function of the proposed algorithm is lower than that of the greedy algorithm. The cost function is measured by using the objective function shown in equation 8.

An important conclusion from the table is that, in all cases, the satisfaction of the proposed algorithm to the soft constraints is more better than using the greedy algorithm individually.

Table 5. Total Cost of the Objective Function

Semester	CSO (soft const.)	greedy Min cost (100 trial)
Gen Fall 14	91.61	148
Gen Spring 15	84.63	117
Gen Fall 15	139.33	162
Gen Spring 16	103.95	151
Gen Fall 16	103.48	140
Gen Spring 17	92.31	167
SWE Fall 14	29.25	32
SWE Spring 15	29.37	35
SWE Fall 15	25.75	29
SWE Spring 16	29.64	34
SWE Fall 16	27.87	33
SWE Spring 17	26.11	30
Avg.	**65.27**	**89.83**

Furthermore, a graphical plotting of Table 5 is shown in Figure 4. Again, the proposed approach outperforms the 100 trials of the greedy algorithm. Here, we consider both soft and hard constraints. It is clear from the figure that, in all cases the proposed approach outperforms the 100 trials of the greedy algorithm.

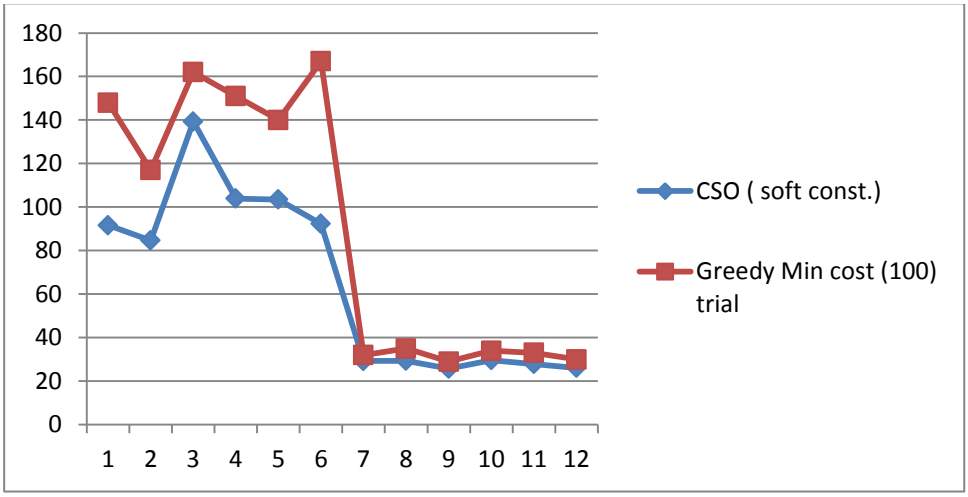


Figure 4. Cost Comparison

Moreover, the time analysis of the proposed algorithm is interesting. The average execution time of the proposed algorithm is a approximately 100 seconds. In fact, this is depending on the number of courses and graph density. The average execution time of the greedy algorithm for 100 trials is 3 seconds. However, the average scheduling of manual time tabling is hours or perhaps days or weeks. Although, the proposed approach is not the best in execution time, some important benefits are obtained from the proposed approach in terms of soft constraints satisfaction, in which, the greedy algorithm cannot satisfy as shown in Table 5. Moreover, 100 seconds are not significantly large compared to the advantages gained by

generating optimal time table, taking into consideration that, time table generation is performed only once at each semester. Comparing to [10], the authors state that their proposed algorithm takes several minutes in execution. Considering this, our proposed approach outperform their algorithm.

6. Discussion

An important challenge of credit hours systems is the exam scheduling, or exam timetabling. The process of manual scheduling of exam time tabling consumes a lot of time and efforts due to the constraints involved in time table scheduling. The problem is a typical graph coloring problem. In our faculty, we are interested in designing feasible time table having the minimum number of time slots without conflicts. Currently, the time table is prepared manually. The greedy algorithm could solve the problem of avoiding exams conflicts, and the running of the algorithm is $O(n^2)$. However, the ordering of the vertices (courses) highly affects the number of generated colors (time slots). Also, the greedy algorithm doesn't consider the soft constraints. So, we propose a new approach for enhancing the performance of the greedy algorithm using the recent chicken swarm optimization (CSO). In this new approach, we are satisfying the hard constraints first using the greedy algorithm avoiding courses conflicts, and hence avoiding searching in a large search domain that the optimization algorithms often face. Using CSO proves superior performance compared to other meta heuristic algorithms.

We propose a new objective function to measure the quality of the generated exam time table. The proposed approach is a two stage algorithm that use the greedy algorithm first to reach a feasible solution, and then use a modified CSO to enhance this initial feasible solution. In the first stage, the avoidance of conflicts is guaranteed by using the greedy algorithm. Next, the proposed objective function explores the domain of valid solutions. The objective function, is a minimization function, mainly considers three terms; the time table length in slots, the differences between slots, and the number of students who are common in different courses. The main advantage of this objective function, compared to other methods, is the flexibility of adding, or removing, constraints according to university requirements without affecting the main hard constraint (the conflict between slots). The weights of the objective function can guarantee the priority of one constraint over another. The proposed approach is more better than those of one stage meta-heuristic. The proposed approach dramatically decreases the search space, and then a small search space is optimized using CSO. Actually, we test the one stage approach using genetic algorithm, and we cannot find a feasible solution in a reasonable amount of time.

In our experiments, the data set used consists of 12 exam timetables representing the dataset of two academic programs in faculty of computers and information; the general program (GEN), and the software engineering program (SWE). The two programs have different number of courses. The data set used is for three consecutive years including two semesters. Three performance of three techniques are compared; the proposed approach, the greedy algorithm with 100 random permutation of the input, and the manual time table (which is used till now in time table scheduling). The performance measured is the number of time slots obtained by each timetabling technique. Results show that, the manual scheduling is the worst one as the number of slots is always greater than or equal to the generated number of slots using the other two techniques. On the other hand, the best average number of slots is

obtained from using the proposed approach. The greedy algorithm has a moderate performance between manual scheduling and the proposed approach. Moreover, the proposed approach considers the soft constraints by using the proposed objective function. The proposed approach could control the balance between the number of time slots and soft constraints satisfaction. The statistical tests performed shows a significant improvements occurred by using the proposed approach over both manual scheduling and even 100 trial of the greedy algorithm.

While the objective function parameters are chosen by trial and error in our experiments, they could be easily adjusted and tuned to suit other data sets. Moreover, changing the objective function parameters reflects the importance of certain soft constraints. The combination of the objective function could consider other constraints such as rooms capacity and many other constraints.

Obviously, the proposed approach produces the optimal time table regarding the number of time slots. However, this may causes more courses to be scheduled on one time slot. Sometimes, this is not desirable as there are some other constraints such as rooms capacity and staff occupancy. It is possible to manually dividing the time slots having more students into one or more slots at the cost of increasing the number of time slots produced. The produced time table may be further enhanced by swapping generated time slots considering the overall institution preferences. In fact, our experiments show that, there are lot of optimal solutions obtained having the same cost function considering the hard constraints. The institution is free to use any of them based on their preferences.

The cost of the proposed approach is less than the cost of the greedy algorithm (100 trials). From the execution time point of view, the average execution time of the proposed approach is a approximately 100 seconds depending on the number of courses and graph density. Comparing to the current manual scheduling of the table, manual scheduling is spending days or even weeks to finalize the time table scheduling. Moreover, as the results says, the produced time slots, of the manual scheduling, are the largest among the other compared methods. Moreover, there is no guarantee of satisfying the soft constraints by manual scheduling or even the greedy algorithm. Compared to other methods [10], the authors state that, their proposed algorithm takes several minutes in execution. So, our proposed approach outperform their algorithm.

7. Conclusions

An important challenge of credit hours systems is the exam scheduling, or exam timetabling. There are many constraints regarding exam table scheduling such as conflicts of exams, availability of time slots, and students preferences. The problem of exam timetabling is a well known graph coloring problem that could be solved using the greedy algorithm. However, there are some drawbacks of the greedy algorithm regarding the quality of the resulted solution. In this paper, we present a novel approach for solving exam time tabling problem by enhancing the performance of the greedy algorithm using modified chicken swarm optimization (CSO). In the proposed approach, CSO is used to choose the optimal order of vertices (courses) to produce the minimum number of colors (time slot). Results show that, the proposed approach significantly enhances the efficiency of the time tabling solution compared to using manual timetabling or even using 100 random iterations of the greedy algorithm when being used individually. Moreover, the proposed algorithm considers

the soft constraints of students preferences in which other algorithms do not. The importance of each soft constraint could be determined using a proposed weighted objective function. From time point of view, the execution time of the algorithm is approximately 1.5 minutes on the average, even in case of higher graph densities. Furthermore, the cost function of the proposed algorithm is significantly less than that of multiple iterations of the greedy algorithm.

References

[1]. Sani, H.M. and Yabo, M.M., 2016. Solving Timetabling problems using Genetic Algorithm Technique. In International Journal of Computer Applications, Vol. 134, Issue. 15, 2016.

[2]. R. Lewis, "A Guide to Graph Coloring Algorithms and Applications", Cardiff School of Mathematics, Cardiff University, Springer International Publishing Switzerland, 2016.

[3]. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T. and Lee, S.Y. "A survey of search methodologies and automated system development for examination timetabling". In Journal of scheduling, Vol. 12, Issue. 1, pp.55-89, 2009.

[4]. Muklason, A., Parkes, A.J., Özcan, E., McCollum, B. and McMullan, P. "Fairness in examination timetabling: Student preferences and extended formulations". In Applied Soft Computing, ELSEVEIR, Vol. 55, pp.302-318, 2017.

[5]. Meng X., Liu Y., Gao X., Zhang H, "A New Bio-inspired Algorithm: Chicken Swarm Optimization". In: Tan Y et al., Coello C.A.C. (eds) Advances in Swarm Intelligence. ICSI, Part I, LNCS 8794, pp. 86–94, 2014.

[6]. Arbaoui, T., Boufflet, J.P. and Moukrim, A. "A matheuristic for exam timetabling", In IFAC-PapersOnLine journal, Vol. 49, Issue. 12, pp. 1289-1294, 2016.

[7]. El Hilali Alaoui, A., Dkhissi, B., Boukachour, J. and Abounacer, R., 2016. "A hybrid Ant Colony Algorithm for the exam timetabling problem". Revue Africaine De La Recherche En Informatique Et Mathématiques Appliquées, Vol. 12, 2016.

[8]. Amaral, P. and Pais, T.C. "Compromise ratio with weighting functions in a Tabu Search multi-criteria approach to examination timetabling", In Computers & Operations Research journal, Vol. 72, pp.160-174, 2016.

[9]. Cheraitia, M. and Haddadi, S. "Simulated annealing for the uncapacitated exam scheduling problem". In International Journal of Metaheuristics, Vol. 5, Issue. 2, pp.156-170, 2016.

[10]. Klüver, C. and Klüver, J.. "A regulatory algorithm (RGA) for optimizing examination timetabling". In Computational Intelligence (SSCI), IEEE Symposium Series on, pp. 1-8, 2016.

[11]. [11] Ceschia, S., Di Gaspero, L. and Schaerf, A. "Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem", In Computers & Operations Research journal, ELSEVIER, Vol. 39, Issue. 7, pp.1615-1624, 2012.

[12]. Hafez, A.I., Zawbaa, H.M., Emary, E., Mahmoud, H.A. and Hassanien, A.E.,. "An innovative approach for feature selection based on chicken swarm optimization". In 7th International Conference of Soft Computing and Pattern Recognition (SoCPaR), IEEE, pp. 19-24, 2015.

[13]. Sivasakthi, S. and Muralikrishnan, N. "Chicken Swarm Optimization for Economic Dispatch with Disjoint Prohibited Zones Considering Network Losses". In Journal of Applied Science and Engineering Methodologies, Vol. 2, Issue. 2, pp.255-259, 2016.

[14]. Wang, Q., Zhu, L. "Optimization of wireless sensor networks based on chicken swarm optimization algorithm". In AIP Conference Proceedings, Vol. 1839, Issue. 1, 2017.

[15]. Wu, D., Kong, F., Gao, W., Shen, Y. and Ji, Z. "Improved chicken swarm optimization", In Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), IEEE International Conference, pp. 681-686, 2015.

[16]. Nursyiva, I., Aris, T., Dian, W. "Chicken Swarm as a Multi Step Algorithm for Global Optimization", In International Journal of Engineering Science Invention, Vol. 6 Issue. 1, PP. 08-14, 2017.

[17]. Qu, C., Zhao, S.A., Fu, Y. and He, W. "Chicken Swarm Optimization Based on Elite Opposition-Based Learning", In Mathematical Problems in Engineering, Hindawi, Vol. 2017, Article ID 2734362, 2017.

[18]. Gopal, K., Kanji, "100 Statistical Tests", 3rd Edition, Sage Publications, India, 2006.