An Intelligent Bio-Inspired Algorithm for the Faculty Scheduling Problem

Sarah Al-Negheimish[1], Fai Alnuhait[2], Hawazen Albrahim[3], Sarah Al-Mogherah[4], Maha Alrajhi[5], Manar Hosny[6]
College of Computer and Information and Sciences
King Saud University
Riyadh, Saudi Arabia

***Abstract*—All universities have faculty members who need to be assigned to teach courses. Those members have various specialties, preferences and different levels of experience. The manual assignment of courses is a very tedious and time-consuming task that the scheduling committee frequently faces in every department. To solve this timetabling problem, we proposed a novel approach using the Bees Algorithm (BA), which is inspired from bees' foraging behavior, hybridized with Demon algorithm and Hill Climbing for more extensive search. The scheduling process took into consideration all constraints and variables associated with scheduling courses, according to the requirements of the Computer Science department in our college. The results showed that the schedules produced from the algorithm outperformed the manual schedules in terms of achieving the objective function and satisfying the constraints. In addition, the hybridized version produced better results than the standard BA version without hybridization. The hybridized algorithm is designed for faculty scheduling, but can be further generalized to solve various timetabling problems.**

***Keywords*—Faculty scheduling; faculty assignment problem; Bees Algorithm; Demon algorithm; timetabling; scheduling**

I. Introduction

Assigning faculty members to teach courses is a tedious process that must be done by almost every university in the world each semester. Similar to other scheduling problems, faculty scheduling is an NP-hard problem [1] that is very difficult to solve optimally using conventional search methods. The reason behind this is the presence of many constraints that should be taken into consideration, such as the clash of times between courses, the maximum and minimum number of workload hours for each faculty, the preferences and specialties of the faculty, and many more hard requirements that can affect the quality of the solution. For such a difficult problem, the available variables and constraints play a significant role in choosing the method that will solve the problem.

Over the past few years, many different methods have been proposed to solve this problem, some of them are more efficient than others. Meta-heuristic optimization algorithms are among the most effective methods for this type of problem, because they keep improving the proposed solution until it reaches a certain satisfactory quality, although not necessarily the optimum [2].

Bio-inspired algorithms are meta-heuristic algorithms that are widely used in many different fields, because of their effectiveness in solving real life difficult problems that cannot be solved to optimality given the current computing resources. However, they are not widely used in the faculty scheduling problem, probably due to its complexity and difficulty of formulating its constraints that are involved in the solution method, which makes designing a bio-inspired solution method fairly complex. This paper presents a new intelligent bio-inspired meta-heuristic algorithm, namely the Bees Algorithm (BA), for solving the faculty scheduling problem.

The remainder of this paper is organized as follows: Section 2 provides a brief overview of some related work. Section 3 describes the problem in terms of variables, constraints, and the objective function. Section 4 presents the approach used to design and implement the algorithm. Section 5 demonstrates the results obtained from applying the algorithm using various evaluation techniques. The discussion is presented in Section 6. Finally, Section 7 provides the conclusion and some future work.

II. Related Work

The faculty scheduling problem has been studied for many years as an independent problem or combined with the more general university course scheduling problems [3]–[6]. In the literature, various methods were used by researchers to handle this problem. We will go through some of the different algorithms proposed to solve the problem below.

A. Gunawan and K. Ng [7] solved the teacher assignment problem using Simulated Annealing (SA) [8]–[10] and Tabu search (TS) [11]. The problem was divided into two phases. Phase one is concerned with finding a feasible schedule that satisfies all the hard constraints. Phase two aims to balance the credit hours between the faculty. Their algorithm starts by generating a random schedule, then applying SA as well as TS to improve the initial solution during both phases. The algorithm was tested using two real datasets, and yielded better results compared to the genetic algorithm [12]–[14], and manual allocation. E. Aycan and T. Ayav [15] also used SA to solve the course scheduling problem, which consisted of assigning courses to classrooms and timeslots, in addition to assigning their instructors. Their methodology focused purely on using SA as a strategy to find the best schedule. Based on the performance, that was measured by execution time and quality; they concluded that using SA gave better results compared to manual scheduling.

Parera et al. [16] used a Genetic Algorithm [13], [14] (GA) to solve a bigger problem which includes assigning faculty

members to sections and assigning sections to classrooms. Their goal was to produce a valid schedule free from time clashes, which the algorithm was able to provide. As specified earlier, A. Gunawan and K. Ng [12] used SA and Tabu search to solve the teacher assignment problem. In addition, they also developed a GA to tackle the mentioned problem. After assigning teachers to sections randomly, they used the GA in the reassignment process, where the crossover operator selects randomly two chromosomes (course with its list of teachers) and chooses crossover points to exchange the combination of the two chromosomes. In addition, the mutation operator only changes one gene (teacher) in a randomly chosen chromosome (course). The algorithm provided better schedules than the manual ones. Similarly, Y. OuYang and Y. Chen [17] used GA to solve the course scheduling problem, but with the addition of graph coloring algorithm to generate the initial population, and they used Tabu search in the mutation operation. They concluded that their algorithm performed better, in terms of timetabling speed, than the manually allocated one.

Gunawan et al. [18] proposed a solution for the course scheduling [19], and teacher scheduling problems using greedy heuristic and SA. It starts with assigning teachers to sections using a mathematical approach, then moves on to timetabling the sections into time periods using a greedy algorithm. Finally, SA is used to make improvements. The algorithm was able to solve the two problems simultaneously while providing feasible solutions.

Lastly, M. Hosny [20] proposed a heuristic approach to solve the faculty assignment problem. The designed algorithm can be divided into two phases, the first one starts by iterating over the list of teachers, assigning them sections, and reordering the list according to their assigned hours. The second phase is only needed if there are some sections that remained unassigned. After choosing the best schedule generated, a Hill Climbing Optimization algorithm is used to improve the chosen schedule. Although the algorithm was able to provide satisfying results, it was restricted to assigning labs to Teaching Assistants in the assignment process.

In the coming section, we will describe in detail the problem under consideration in this research, as it paves the first steps into solving the faculty scheduling problem.

III. Problem Description

The faculty assignment problem is defined as assigning teachers to courses, while adhering to a number of pre-specified constraints. In this paper, we define the problem of faculty scheduling in terms of the requirements provided by the Computer Science Department in King Saud University, as variables, constraints, and objective function.

A. Variables

There are only two variables under consideration in our problem: courses, and teachers. Starting with the courses, each course has a number of sections, each of which has a unique number, its specified hours, its type which can be: lecture, tutorial, or lab, and its time slots. Whereas for teachers, they are divided into PhD holders who teach only lectures, BSc holders who teach only tutorial and lab sections, and finally, MSc holders who can teach any type of sections. The final two categories are further divided into: students (i.e., those who are currently studying postgraduate degrees while working) and full-time.

B. Constraints

The constraints are divided into hard and soft constraints. Violating any of the hard constraints makes the schedule infeasible, whereas violating soft constraints affect the quality of the solution. In our problem, we only have one hard constraint which is the clash of courses' time for the same instructor. Having time conflicts in any teacher's schedule will make the whole solution infeasible. As for the soft constraints, we have five constraints. First, ensuring that each teacher's assigned hours are within the boundaries of a certain predefined minimum and maximum workload, which is stated for each instructor based on their rank. Second, fulfilling teachers' preferences for courses represented as a wish list. Third, balancing the workload amongst teachers within the same rank, so that there would be no significant differences. Forth, ensuring that each instructor gets at least a day off in their schedule for course preparation and research. Lastly, minimizing the number of instructors per course, as it can ease the teaching and coordination process.

C. Objective Function

It is crucial to define how to measure the quality of the generated schedules in order to evaluate the outcomes of the algorithm. First, we assume that any violation of our hard constraint (i.e., time clashes in a teacher's schedule) is not acceptable. Thus, we only measure the quality of feasible solutions. To measure the quality of the solution we count each soft constraint violation, multiplied by its weight, where the weight is determined based on the importance of the soft constraint. We also add to this, the number of unassigned sections in the current solution, since our main target is to assign all sections to instructors. The objective function is represented in (1) below. Intuitively, the closer the objective function is to zero, the better the solution would be.

$$Minimize \qquad u + \sum_{i=1}^{n} (w_i c_i). \tag{1}$$

Where, u is the number of unassigned sections in the solution, w_i is the penalty weight of the i^{th} (soft) constraint, c_i is the number of violations of the i^{th} (soft) constraint, n is the number of constraints.

To ensure the integrity of the objective function, the measurement of the violations for each constraint was normalized within the range [0, 1].

The following section demonstrates our proposed approach to develop a bio-inspired meta-heuristic for solving the faculty scheduling problem.

IV. Methodology

In this section, we propose a bio-inspired approach, based on the bees' foraging behavior, to solve the stated problem. The described algorithm hybridizes the Bees Algorithm [21] with the Demon Algorithm [2], and Hill Climbing [2]. The algorithm is inspired from N. Alhuwaishel and M. Hosny [22], where a hybrid Bees-Demon Algorithm was used to solve the University Course Timetabling problem. However, to the best

of our knowledge this kind of hybridization has not been attempted before for the faculty scheduling problem.

The Bees algorithm provides a variety of solutions as it starts with a population of solutions called the initial population. This approach is needed to explore the search space more; thus, gives a higher probability of finding near optimal solutions [23]. The basic idea of the Bees algorithm involves the generation of multiple initial solutions then focusing on exploiting the best and elite (best of the best) solutions, whilst performing modification to increase the chance of finding near optimal solutions. the steps used to apply the Bees algorithm, shown in Fig. 1, can be described as follows [21]:

a) Construct the initial population of schedules.

b) Evaluate fitness of the population using the objective function.

c) Selecting the best and elite solutions.

d) Perform neighborhood search on the selected sites (Hill Climbing), with more search to be done around elite solutions (Demon Algorithm).

e) Repopulate the region after removing discarded solutions.

The algorithm consists of two major parts: initial population construction, and neighborhood search. These subparts will be combined to design the overall algorithm.

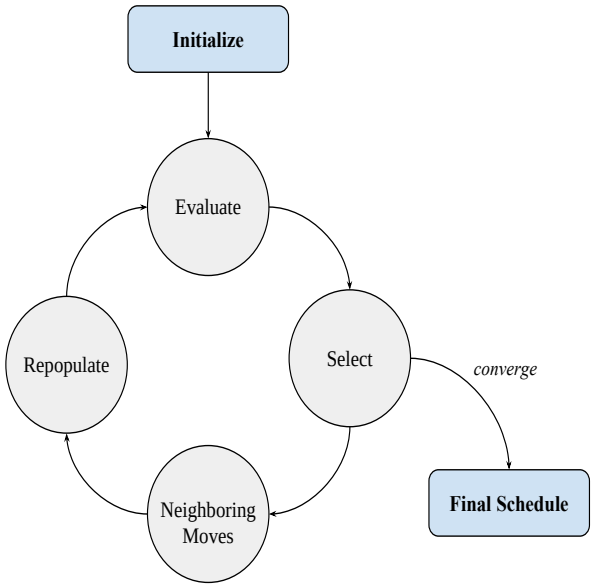

Fig. 1. Abstract representation of the hybrid algorithm.

A. Initial Population Algorithm

The first step in our proposed algorithm is to construct a population of feasible schedules that do not violate the hard constraint. To accomplish this, we designed a greedy randomized heuristic algorithm to populate the search space. After categorizing the sections into separate time slots to avoid time clashes during the assignment, the algorithm proceeds in three phases. The first phase tries to assign only the minimum workload for each faculty from their wish list, while the second phase tries to assign the remaining workload for the faculty, but this time not necessarily from their wish list. The third phase, on the other hand, is considered only if there are still sections not assigned to faculty after the first and second phases. In Phase three, assigning faculty hours above their maximum workload (within a certain predefined percentage) is attempted. In each of these phases, we order the list of faculty based on their rank, which is calculated by their job position, administrative posts, and seniority. Within the second and third phases, the number of assigned hours is taken into consideration. This is intended to give higher priority in assignment to those faculty on top of the list. For those whose ranking criteria are equivalent, the list is ordered randomly. Using these procedures, we obtain different schedules to create the population. The steps of the algorithm are described in Algorithm 1:

Algorithm 1: Initial population generation

Input: list of faculty *f*, list of sections *s*, the percentage *c* to increase workload, population size n_{SEP}

Output: population of feasible solutions *P*

```
P ← { }
repeat while | P | < n_SEP
categorize(s)
/* Phase 1: assigning minimum workload */
order(f )
for i = 1: | f |
    assign(f_i, s, min)
/* Phase 2: assigning maximum workload */
order(f )
for i = 1: | f |
    assign(f_i, s, max)
/* Phase 3: assigning above maximum workload */
if s ≠ { }
    for i = 1: | f |
        f_i(workload) ← f_i(workload) × c
        assign(f_i, s, max)
P ← P + f
```

The algorithm starts with an empty set of population *P*, then repeats the following steps until it reaches the desired number of schedules *n*. At the first step, the list of sections *s* is divided into buckets according to their timeslot using ***categorize***. This is done to ensure that any time clashing incident does not occur, thus preventing infeasible solutions. The ***order*** method takes the list of faculty *f* and sorts it according to the member's position, seniority, held admin post, and assigned hours. In cases where more than a member share the same criteria, their order will be chosen randomly. When it comes to the ***assign*** task, the method tries to assign f_i with a matching section according to either the maximum or minimum workload for f_i. It is important to note that in phase one's call of ***assign***, *s* is sorted according to f_i's preference. However, that is not taken into consideration within the the subsequent calls to ***assign*** in phases two and three. At the end, the algorithm checks whether *s* still has sections that need to be assigned. If so, every faculty's workload will be increased by percentage *c*. Fig. 2 represents these procedures.

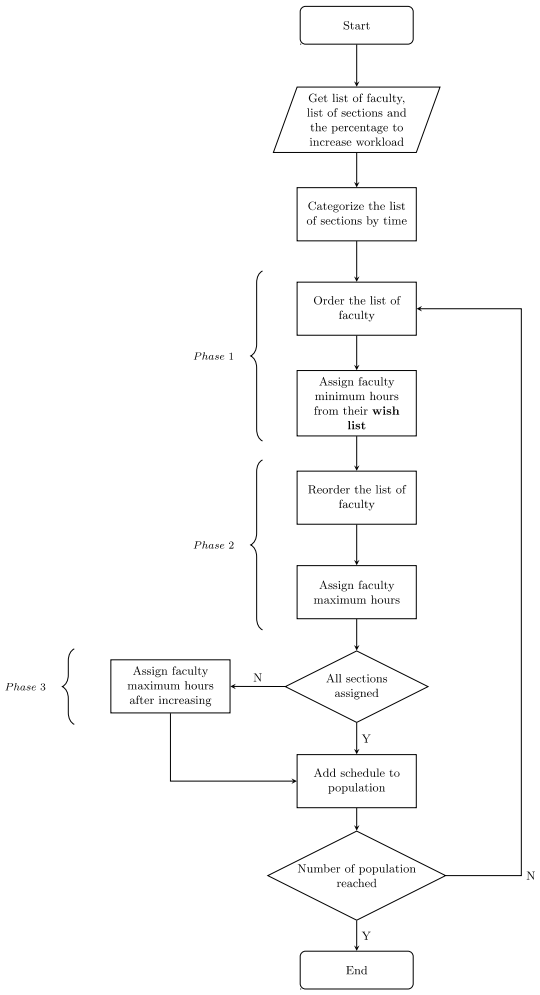

Fig. 2. Initial population generation flowchart.

B. Neighborhood Search

Neighboring moves are functions applied on already existing solutions to generate new slightly different ones, hoping to find better solutions in their vicinity. The most important characteristic of a neighboring move is locality, which is its effect on the original solution. Locality means that small changes applied to the original solution representation (e.g. changing the location of some variables) should correspond to small changes in the actual solution (i.e., real schedule). Otherwise, the new solution will have drastic changes different from the original solution, which, in extreme cases, could cause the search to converge towards a random solution [2]. Listed below are the proposed neighboring moves used to generate new solutions, which are intended to reduce the violations of the soft constraints in our problem.

1) Balancing the Workload

This procedure is applied when two different instructors, who belong to the same rank and position, are not equivalent in the number of hours assigned. This is accomplished by calculating the average hours of each group then segregating them into above-average and below-average. The algorithm then tries to transfer sections from the above-average group to the below-average group, if no violations in constraints occur.

2) One-day off

This function aims to increase the number of faculty members who have a day off, by swapping sections between faculty members if it ensures that both individuals enjoy a day off, have no conflicts in assigned courses, and can be assigned the type of section.

3) Max number of Faculty Members per Course

Each course has a limit on the number of different instructors that should not be exceeded. Therefore, this neighboring move tries to lessen the number of faculty members for these courses. This is achieved by setting a limit to each course, then it investigates the possibility of consolidating a course. The approach used in the algorithm explores potential faculty members that share more than one course together, then checks if the course sections could be swapped to reduce the number of instructors for either course.

C. Proposed Hybrid Algorithm

As previously mentioned, the main algorithm we use is inspired from the Bees Algorithm [21]. The algorithm starts by generating a collection of schedules named the population; then it selects the highest scoring schedules according to their fitness (objective function value). Then it divides the group into best schedules and elite schedules, which are the best of the best, creating two mutually exclusive sets. In our proposed method, we have also incorporated a Hill Climbing [2] approach for accepting the new schedule after applying the neighboring moves on the best schedules. In other words, if the new solution produced after applying a neighboring move on each of the best solutions is better in terms of the objective function, it replaces the previous solution. However, we took a different approach for searching around the elite schedules, where we apply the Demon Algorithm (DA). The DA is a variant of the famous Simulated Annealing (SA) approach but with a deterministic acceptance function [2]. The DA potentially accepts worse solutions, based on a certain credit value called the demon, in hopes of finding better solutions in the next iterations. Thus, the idea is to do a more intelligent search around elite solutions, in an attempt to discover even better solutions as the search progresses. The intensifying phase of the algorithm continues for each schedule until the schedule's fitness ceases to change for five consecutive iterations.

The first step in the proposed algorithm is to create a population of schedules using the heuristic algorithm described in Section 4.1. After the initialization of all the variables, including the demon credit *d*, we repeat the following process until convergence.

Firstly, the program evaluates every schedule in the population then selects the best and elite schedules from the population. Secondly, we intensify the search on the best and elite schedules using the neighboring moves for best and elite respectively. When applying neighboring search on the 'best' schedules, the new schedule is accepted if the overall fitness is better. However, when applying the search on 'elite' schedules, and based on the principle of the DA, the program calculates

the difference in fitness value of the two schedules, the old schedule and the new schedule. If the difference is positive (+), the new schedule is accepted and the difference value is credited to the demon value. On the other hand, if the difference is negative (-), the new schedule is accepted only if the demon credit can withstand the difference and then we update the demon value after subtracting the difference value. This process is considered as the intensification part of the algorithm as it intensifies the search on each individual schedule to improve it.

Finally, we repopulate the search space keeping the best and elite schedules from the previous iteration, and keep the current best schedule in hand. Diversification is incorporated in the algorithm as it helps to generate a population with various fitness values. It is worth mentioning that the selection of best set is done in a stochastic manner [24] using a tournament selection approach, as the algorithm chooses five schedules randomly from the population, and then selects the best schedule among them. This process is repeated until reaching the required number of best set. This approach gives better results in terms of diversification and producing different schedules, as deterministic methods tend to stick with the same best set regardless of the iterations [2].

Once the best schedule ceases to change for five consecutive iterations, the algorithm stops repetition and outputs the result. The details are described in Algorithm 2.

The algorithm starts by receiving a population of schedules from Algorithm 1 then evaluates every schedule in the population. Method ***eval*** does precisely that by taking a schedule and producing the objective function value for each corresponding schedule. After that, a set of *best* and *elite* schedules is selected respectively based on the fitness value in our case. Once the sets are established, the algorithm first iterates over the *best* set and attempts to improve the schedules using the method ***neighborhood*** then moves on to the *elite* schedules. ***Neighborhood*** implements the techniques discussed in Section 4.2 to improve the quality of schedules in both sets. However, improving criteria must be chosen for the schedules in both cases; this is selected to be 'Hill Climbing' for *best* and 'Demon' for *elite*, which are passed to the method ***neighborhood***. After these steps have been conducted, the algorithm repopulates the space with the inclusion of the *best* and *elite* sets, then proceeds with the next iteration.

Algorithm 2: Hybridized Algorithm

Input: demon credit *d*
Output: an optimal schedule *s*

```
p ← Initial Population (Algorithm 1)
repeat until convergence:
    for i = 1: | p |
        eval(p_i)
    b ← min(p) //best schedules
    e ← min(b) //elite schedules
    b ← b - e
    for i = 1: | b |
        repeat until convergence:
            neighborhood(b_i , hill climbing)
    for i = 1: | e |
        repeat until convergence:
            neighborhood(e_i , demon, d)
    p ← Initial Population + b + e
    for i = 1: | p |
        eval(p_i)
    s ← min(p)
return s
```

In the next section, details of how the results were evaluated will be illustrated to conduct analysis on the performance of the designed algorithm.

V. Experimental Results

In this section, we will discuss in details the dataset that we used to test our algorithm, and show our results and the different criteria used to evaluate these results.

A. Characteristics of the Dataset

To test our algorithm, we used a real dataset obtained from the Computer Science department at King Saud University. In addition, we a created another theoretical (i.e., synthesized) dataset inspired from the real dataset. We conducted the testing on one real dataset and two different theoretical datasets. Both types were saved on (.csv) files. Table I summarizes the details of the three datasets. Each dataset is split into two different parts:

a) The dataset of the teachers to be allocated: This includes: professors, associate professors, assistant professors, lecturers, and teaching assistants.

b) The dataset that contains the information about the courses and sections to be taught. This includes: lectures, labs, and tutorials.

B. Parameter Tuning

The objective function was designed to measure the quality of a schedule. A crucial part of it was to decide the weights used to penalize the violations of soft constraints. The weights were selected to be in the range [0, 1] and the totality of them would equal to one. We prioritized the soft constraints with their corresponding weights depending on the department's needs:

a) Minimum and maximum workload, $w_1 = 0.4$.
b) Balancing the workload, $w_2 = 0.3$.
c) One-day off, $w_3 = 0.2$.
d) Minimize the number of instructors per course, $w_4 = 0.1$.

After designing our algorithm, we had to test different values for each of the parameters: population size, best size, elite size, and demon credit.

For each of these parameters we used the same dataset. In parameter tuning [25], [26], we applied grid search technique to test a variety of values for one parameter (approximately 13 different values) keeping the rest of values constant. Each value was tested five times then we compared the resulting average fitness with that of the rest of the values.

TABLE I. DATASET SUMMARY

	Datasets		
Variable	**Real Dataset**	**Theoretical 1**	**Theoretical 2**
Courses	22	7	10
Sections	167	70	120
Professors	0	1	0
Associate Prof.	1	1	1
Assistant Prof.	8	1	6
Lecturers	16	9	13
Teaching Assist.	20	5	14

In addition, we chose to start our tuning with the population size, since it was observed to have the most effect on the fitness. Then we tested the number of best solutions, and then we tuned the number of elite solutions, since it is a subset of the best. Finally, we tuned the demon credit. The final parameter values we obtained were, population size = 50, best size = 8, elite size = 4, and demon credit = 0.25, which were deemed fit for our algorithm based on the quality of solutions obtained.

C. *Evaluating the Results by Analyzing the Objective Function*

We ran our algorithm twenty times on each of the three previously defined datasets, documenting the fitness and the execution time (in minutes). The program was executed on a Macbook Pro with OS X Yosemite, 2.9 GHz Intel Core i7 processor, 8 GB DDR3 memory.

In Fig. 3, a sample of the fitness (objective function) of the best schedule at hand is illustrated against the iteration sequence. It is clear that in every iteration, the intensification phase of the algorithm provides a major contribution to the fitness as well as finding another potential candidate to substitute the current schedule through the diversification phase.

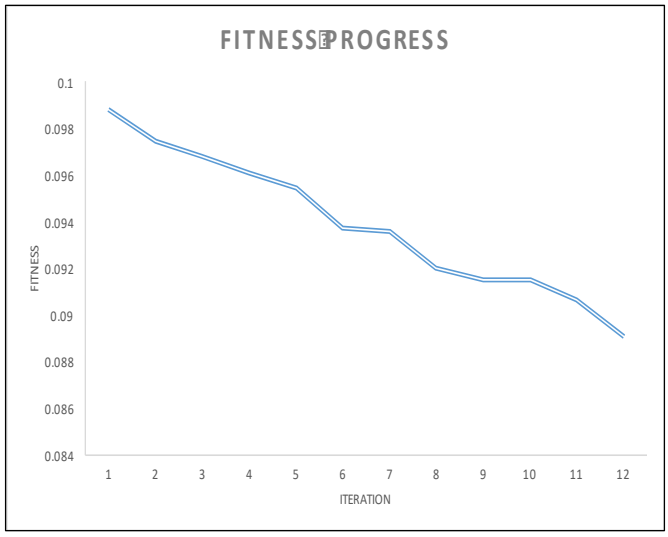

Fig. 3. Progress of best schedule over time.

TABLE II. DATASET RESULTS SUMMARY

	Datasets		
Measure	**Real Dataset**	**Theoretical 1**	**Theoretical 2**
Average Fitness	0.062	0.033	0.078
Standard Deviation	0.013	0.010	0.007
Minimum Fitness	0.029	0.002	0.059
Maximum Fitness	0.084	0.049	0.089

TABLE III. DATASET RUNTIME SUMMARY

	Datasets		
Criteria	**Real Dataset**	**Theoretical 1**	**Theoretical 2**
Number of Sections	167	70	120
Avg. Exec. Time (min)	5.17	1.775	4.35

As demonstrated by the results in Table II, all of the obtained values on all data sets are very close to zero, indicating very small number of violations of the soft constraints. Also, the values obtained fall into the same range of values, meaning that the algorithm produces a good result in each run. This is confirmed by measuring the standard deviation to ensure the stability of our algorithm. On the other hand, the maximum result shows the worst case, which is still very close to zero.

Understandably, the algorithm could not give a schedule with a zero-fitness value, as there are many factors that prevent the algorithm from reaching the ideal solution. For instance, different sections have different hours such as labs and lectures. Thus, it is nearly impossible to have the workload distributed perfectly between the faculty members. Nonetheless, considering all the constrains, we deemed important in the algorithm, the results shown are indeed very satisfactory.

Regarding the execution time, it is noticeable that the execution time differs from run to run, and that is understandable as well, since the time it takes for the algorithm to converge is distinct depending on the population in hand and the number of iterations it will take to improve various schedules in the dataset during the intensification phase. Table III above summarizes the relationship between the dataset size and execution time in minutes. Overall, the algorithm produces excellent results in a reasonable processing time.

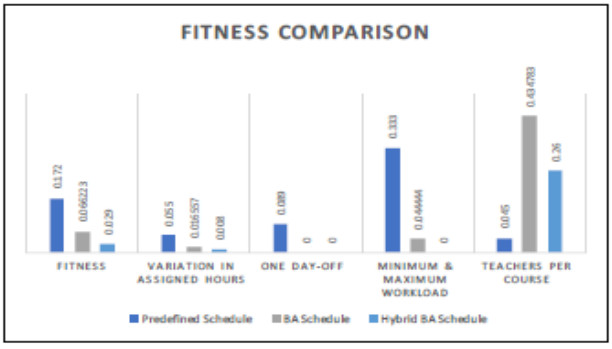

Fig. 4. Comparison between manual, BA and hybrid BA schedules.

D. Evaluating the Results by Comparing the Manual Schedule with our Algorithm's Schedule

The manually designed schedule was compared with the generated schedule by our algorithm on the real dataset to assess the latter. The assessment is based on fulfilling the soft constraints, since the hard constraint is guaranteed to be satisfied to make a valid schedule. Table V shows the results of the comparison in details.

As can be seen from Table V, our algorithm provided a schedule that eliminated any violations of 'min & max workload' constraint as well as the 'day off' constraint. In addition, it managed to balance the workload by a significant amount, since it provided an overall total variance from the average workload of just 0.008, compared to a total variance of 0.055 for the manually allocated schedule. Evaluating the overall fitness of the schedule, our schedule has a fitness of approximately 0.03 while the manual schedule has a fitness of approximately 0.17. This shows that our schedule improved the manual schedule by a remarkable 83.4% value. However, the only constraint that our algorithm was not able to provide satisfactory results for was minimizing the number of instructors per course shown evidently in Fig. 4. This is probably due to the small penalty weight that we assigned for this constraint, since we considered it less important in the schedule than the other constraints.

E. Evaluating the Results by Comparing the Proposed Algorithm with the Classic BA

To assess the effectiveness of hybridizing BA with Demon and Hill Climbing algorithms, we compared the results of our algorithm to the results produced from the classic BA using the real dataset following the same criteria used in the previous section. As shown in Table V, the hybrid algorithm outperforms the classic BA in generating the best schedule. The two algorithms performed equally well in avoiding the violation of the day off constraint. However, the hybrid algorithm produced considerably better results in fulfilling the rest of the soft constraints, with a 56% improvement in balancing the workload. Digging deeper into these schedules, we noticed a notable difference in the quality of the produced solutions at the instructors' level. While the schedules produced by the hybrid algorithm tend to limit the number of different courses and assign a consistent set of sections to each instructor, the classic BA generates more variant ones. An example to further explain this point is illustrated in Table IV below, where we took one of the instructors and compared her schedule generated by the two approaches. We can clearly see that the hybrid BA schedule is more practical and more convenient for the instructor.

TABLE IV. INSTRUCTOR ASSIGNED SECTIONS EXAMPLE

	BA Schedule			**Hybrid BA Schedule**		
	Course	Type	Hours	Course	Type	Hours
	CS09	Tutorial	1	CS17	Lab	6
	CS23	Lecture	3	CS23	Tutorial	2
	CS10	Tutorial	1			
	CS01	Tutorial	1			
Number of Sections	4			5		
Unique courses	4			2		
Total Workload	6			8		

TABLE V. MANUAL VS CLASSIC BA VS HYBRID BA RESULT SUMMARY

	Manual Schedule				**BA Schedule**				**Hybrid BA Schedule**			
Category	**Min and max workload violation**	**Balance workload violation**	**Day off violation**	**Number of instructors per course violation**	**Min and max workload violation**	**Balance workload violation**	**Day off violation**	**Number of instructors per course violation**	**Min and max workload violation**	**Balance workload violation**	**Day off violation**	**Number of instructors per course violation**
Students	2	0.003	0		0	0	0		0	0	0	
Professors	7	0.017	0		0	0.003	0		0	0.004	0	
Lecturers/TA	6	0.035	4		2	0.014	0		0	0.004	0	
All	15	0.055	4	1	2	0.017	0	10	0	0.008	0	6
Total Fitness	0.172				0.066				0.029			

TABLE VI. BA VS HYBRID BA AVERAGE COMPARISON

Criteria	**BA**	**Hybrid BA**
Avg. Fitness	0.076	0.062
Avg. Exec. Time	0.82	5.17

Taking the performance evaluation of the proposed algorithm a step further, we compared the two approaches in terms of the average fitness obtained along with the average time needed to generate the solutions, running each algorithm 20 times on the real dataset to get the average value.

As Table VI demonstrates, the hybrid algorithm generates better results in general. Although the improvement is not very significant on an average scale, our goal is to find the best fitted schedule which will be obtained through running the algorithm multiple times and adopting the best schedule. In other words, increasing the chance of finding a near optimal solution on a set of satisfactory solutions would be more beneficial than trying to ensure that all solutions in the set are optimal solutions. The difference in the average running time is not a concern as well, since the algorithm will only be run once each semester in practical situations. So, we can sacrifice the

increase in execution time for the sake of obtaining a much better schedule that will be adopted throughout the semester.

F. Evaluating the Results with the Assessment of the Schedule by the Scheduling Committee

Finally, our results were also assessed by members of the scheduling committee in the Computer Science Department, by answering an evaluation survey. Overall, we received a positive feedback from the scheduling committee. Generally, they strongly agreed that the algorithm fulfilled the requirement of assigning all courses to faculty members, as well as allowing each faculty member a day off per week. They also agreed that the algorithm managed to balance the workload among the faculty members. Moreover, the committee agreed that the quality of the schedule produced was satisfactory, and that they consider the resulting schedule reliable.

Finally, the scheduling committee strongly admits that our algorithm is needed and useful for the Computer Science department, and would use it if it was currently available.

VI. Discussion

After applying our proposed method to solving the faculty scheduling problem, it is evident that the Bees Algorithm proved its capability and suitability for this problem. Specifically, the diversification stage played a significant role in the exploration of many different solutions. This was achieved through the greedy-randomized population creation part of the algorithm. Whereas intensification further improved the solutions obtained that being the neighboring moves' role, focusing on minimizing the violations of the soft constraints.

Moreover, hybridizing the Bees Algorithm with another meta-heuristic immensely improved our algorithm's performance, leaping to a higher level of intelligence. We used both Hill Climbing and Demon algorithms as solution acceptance algorithms. The Hill Climbing algorithm was applied on the best and elite solutions, whilst the Demon algorithm was only applied on the elite solutions, with the intension of doing more intelligent search around the elite than the other selected best solutions.

VII. Conclusion

In this paper, we tackled the faculty scheduling problem, which is concerned with assigning faculty members to prescheduled courses. To solve the problem, firstly, we designed the construction of the initial population that is considered a primary factor in the Bees Algorithm. We used a specially designed greedy-randomized heuristic for this purpose. Secondly, we designed the neighboring moves that will be used to improve the solutions selected by the algorithm. We hybridized the Bees algorithm with the Demon algorithm and Hill Climbing, which is considered an innovative approach in this particular problem.

We used the dataset provided by the CS department to test our algorithm and chose to use this dataset to evaluate our algorithm, because it portrays a realistic environment. We also used two theoretical datasets to further test the algorithm. The algorithm showed superior results when compared to the manually allocated one, as it managed to eliminate 'min & max workload' constraint as well as the 'day off' constraint. Moreover, the scheduling committee in the department evaluated the schedules produced by the algorithm, and agreed that it satisfies their expectations.

Several areas of improvement arise, though, by enhancing the hybrid algorithm to solve some additional requirements, such as minimizing the number of courses assigned to each teacher. Further research could be conducted by broadening the problem and generalizing the algorithm to solve other variations of scheduling.

References

[1] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in Foundations of Computer Science, 1975., 16th Annual Symposium on, 1975, pp. 184–193.

[2] E.-G. Talbi, Metaheuristics: from design to implementation, vol. 74. John Wiley & Sons, 2009.

[3] D. Abramson, "Constructing school timetables using simulated annealing: sequential and parallel algorithms," Manag. Sci., vol. 37, no. 1, pp. 98–113, 1991.

[4] T. Ferdoushi, P. K. Das, and M. A. H. Akhand, "Highly constrained university course scheduling using modified hybrid particle swarm optimization," in Electrical Information and Communication Technology (EICT), 2013 International Conference on, 2014, pp. 1–5.

[5] F. Aloul, I. Zabalawi, and A. Wasfy, "A SAT-based approach to solve the faculty course scheduling problem," in AFRICON, 2013, 2013, pp. 1–5.

[6] R. Lewis and B. Paechter, "Finding feasible timetables using group-based operators," IEEE Trans. Evol. Comput., vol. 11, no. 3, pp. 397–413, 2007.

[7] A. Gunawan and K. M. Ng, "Solving the teacher assignment problem by two metaheuristics," Int. J. Inf. Manag. Sci., vol. 22, no. 2, pp. 73–86, 2011.

[8] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, and others, "Optimization by simmulated annealing," science, vol. 220, no. 4598, pp. 671–680, 1983.

[9] R. W. Eglese, "Simulated annealing: a tool for operational research," Eur. J. Oper. Res., vol. 46, no. 3, pp. 271–281, 1990.

[10] A. G. Nikolaev and S. H. Jacobson, "Simulated annealing," in Handbook of Metaheuristics, Springer, 2010, pp. 1–39.

[11] F. Glover and M. Laguna, Tabu Search∗. Springer, 2013.

[12] A. Gunawan, K. M. Ng, and H. L. Ong, "A genetic algorithm for the teacher assignment problem for a university in Indonesia," Inf. Manag. Sci., vol. 19, no. 1, pp. 1–16, 2008.

[13] J. H. Holland, "Genetic algorithms," Sci. Am., vol. 267, no. 1, pp. 66–72, 1992.

[14] M. Mitchell, "Genetic algorithms: An overview," Complexity, vol. 1, no. 1, pp. 31–39, 1995.

[15] E. Aycan and T. Ayav, "Solving the course scheduling problem using simulated annealing," in Advance Computing Conference, 2009. IACC 2009. IEEE International, 2009, pp. 462–466.

[16] S. Parera, H. T. Sukmana, and L. K. Wardhani, "Application of genetic algorithm for class scheduling (Case study: Faculty of science and technology UIN Jakarta)," in Cyber and IT Service Management, International Conference on, 2016, pp. 1–5.

[17] Y. OuYang and Y. Chen, "Design of automated Course Scheduling system based on hybrid genetic algorithm," in Computer Science & Education (ICCSE), 2011 6th International Conference on, 2011, pp. 256–259.

[18] A. Gunawan, K. M. Ng, and K. L. Poh, "Solving the teacher assignment-course scheduling problem by a hybrid algorithm," Int J Comput Inf. Engin, vol. 1, no. 2, pp. 137–142, 2007.

[19] M. W. Carter and G. Laporte, "Recent developments in practical course timetabling," in International Conference on the Practice and Theory of Automated Timetabling, 1997, pp. 3–19.

[20] M. I. Hosny, "A Heuristic Algorithm for Solving the Faculty Assignment Problem," in Proceedings of the International Conference on Frontiers in

Education: Computer Science and Computer Engineering (FECS), 2012, p. 1.

[21] D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi, "The bees algorithm–a novel tool for complex optimisation," in Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference (3-14 July 2006), 2011.

[22] N. ALHUWAISHEL and M. HOSNY, ""A Hybrid Bees/Demon Optimization Algorithm for Solving the University Course Timetabling Problem," in Proceedings of the 3rd NAUN International Conference on Mathematical, Computational and Statistical Sciences. Dubai, United Arab Emirates, February, 2015.

[23] X.-S. Yang, S. Deb, and S. Fong, "Metaheuristic algorithms: optimal balance of intensification and diversification," Appl. Math. Inf. Sci., vol. 8, no. 3, p. 977, 2014.

[24] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in Proceedings of the second international conference on genetic algorithms, 1987, pp. 14–21.

[25] Á. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," IEEE Trans. Evol. Comput., vol. 3, no. 2, pp. 124–141, 1999.

[26] A. E. Eiben and S. K. Smit, "Evolutionary algorithm parameters and methods to tune them," in Autonomous search, Springer, 2011, pp. 15–36.