Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 9, Issue. 6, June 2020, pg.82 – 91

TIMETABLE GENERATION USING GENETIC ALGORITHM FOR BATCHES UNDER A P J ABDUL KALAM TECHNOLOGICAL UNIVERSITY

Ghiridhar S[1]; Sachin A[2]; Edwin M.T[3]
Department of Computer Science and Engineering,
Adi Shankara Institute of Engineering and Technology,
(of Aff. A P J Abdul Kalam Technological University)
Ernakulam, India.
[1] ghiridhars@gmail.com
[2] sachinnp1998@gmail.com
[3] edwintensen07@gmail.com

Prof. Unnikrishnan K.N[4]
Department of Computer Science and Engineering,
Adi Shankara Institute of Engineering and Technology,
(of Aff. A P J Abdul Kalam Technological University)
Ernakulam, India.
[4] unnikrishnan.cs@adishankara.ac.in

Abstract— The problem of scheduling a timetable is a complex job that is considered as an NP-hard problem, i.e., not verifiable in polynomial time. This is a typical scheduling problem that appears to be a strenuous job in every academic institute. The Genetic Algorithm, being an adaptive algorithm, can improve its efficiency as it progresses. A timetable should satisfy a particular number of constraints that are specific to the organization that it has been subjected to. Here, the university that is associated with our college, Kerala Technological University (KTU), has just been established and it is essential to have a timetable scheduler of its own to ease the burden of colleges to specify its norms and amount of subjects it deals with every year.

Keywords— Genetic Algorithm, Chromosome, Mutation, Crossover, Selection, Fitness Function, Constraints.

I. INTRODUCTION

The process of creating a timetable involves filling the slots of timetable while fulfilling some criteria called constraints. There are two types of constraints, soft constraints, and hard constraints. Soft constraints are those that can be considered valid even when it is not completely satisfied, i.e., lenient. But, hard constraints cannot be valid once violated, i.e., strict. The search space of a timetabling problem is too large to find an optimum solution, that can be considered feasible. Feasible solutions here mean those solutions that completely satisfy hard constraints strictly as well as to comply with soft constraints.

We need to choose the most appropriate one from feasible solutions. There exist E*S ways of allocating E events to S slots when searching for an optimal solution. This implies that no definite algorithm exists to solve a problem that has such large instances of in reasonable time.

Genetic algorithms (GA) are optimization approaches inspired by biological evolution. They are very effective in solving complex problems. The process of GA starts with an initial population and generates a new population using selection, crossover, and mutation, which is constantly evaluated by a fitness function. The main goal behind GA is to improve the overall fitness of the population. The general genetic algorithm is:

```
population = randomly generated chromosomes;
while(terminating condition is not reached) {
runGA();}
// a single run of a genetic algorithm fu nction
runGA() {
parents = getParents();
child = crossover(parents);
child = mutate(child);
 population.add(child);
}
```

In this paper, we will be using the genetic algorithm to serve a metaheuristic approach for the University Timetable Problem. A generation is usually composed of a population or collection of solutions. We start from infeasible timetables and try to get feasible ones. The initial population is a random assignment of faculty and subject to a timeslot (day and hour). Solutions are supposed to go through a phase of evaluation and the fittest is to survive. Usually, the fitness value of the parents is an indicator of how probable it is to obtain offspring from them.

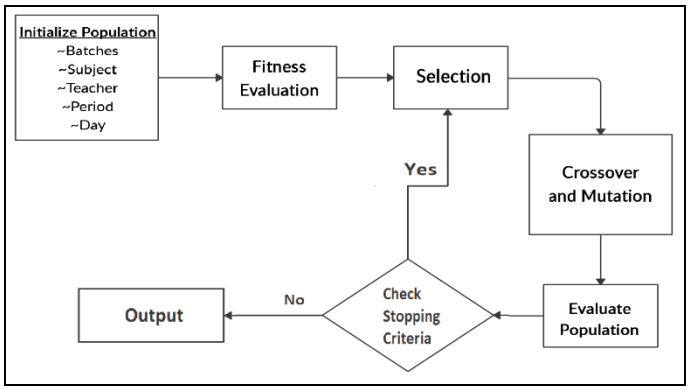

Fig.1. Genetic Algorithm Flowchart

The fitness function defined here is the inverse of the number of chromosomes (subject and faculty for a particular timeslot) with class conflicts. Lesser the number of students with class conflicts, the more fit the class is. The successive generation will become more suited to their environment. This can be done by letting them undergo crossover and mutation operations that will lead to better offspring. The above process is repeated until a termination condition is met or upon a certain number of generations reached. The whole process flowchart is shown in the figure (Fig. 1).

II. METHODOLOGY

A. REPRESENTATION

Different representations and data structures can be used to represent a timetable. One possible representation could be a matrix where each cell corresponds to a timeslot (day, period), and faculties and their subjects are assigned to each slot, also called a chromosome. To ensure no conflict, an entity called conflict operator is given to each chromosome. A sample is shown in the figure (Fig.2).

Batch name	Period 1	Period 2	Period 3	Period 4	Period 5	Period 6	Period 7
Day 1	C1 (Subject ID, Teacher ID, Conflict value)			C3 (S. ID, T. ID, C value)			
Day 2		C2 (S. ID, T.ID, C value)			C4 (S. ID, T. ID, C value)		C5 (S. ID, T. ID, C value)
Day 3				C8 (S. ID, T. ID, C value)			
Day 4			C6 (S. ID, T. ID, C value)				
Day 5							C7 (S. ID, T. ID, C value)

Fig.2. Sample Representation of the proposed timetable

Here, the figure is a sample unfilled population with each chromosome having 3 attributes, Subject ID, Teacher ID, and the conflict value.

The matrix is for each batch and any number of batches can be added depending on the scope of the solution the timetable needs to be generated for. In our proposed algorithm, the population matrix is a collection of batches, where each batch stores information about the teacher, the subject, and the corresponding timeslot.

B. CONSTRAINT IDENTIFICATION

Due to the complexity of the timetabling problems, a wide range of heuristics is used to find feasible solutions. The method is to formulate a genetic algorithm approach that allocates teaching events to timeslots. It uses some heuristics to find initial feasible solutions and it considers the following set of hard and soft constraints:

1. Hard constraints
    a. Lab periods should be of continuous hours for a specified period.
    b. There should not be following conflicts:
        i. A teacher cannot have more than one subject at the same time.
        ii. A subject should not have more than 1 faculty assigned to it at the same time.
        iii. A timeslot having a teacher and subject should not coincide between batches.
    c. The number of periods per week for the subjects should be within the limit specified by KTU.
2. Soft constraints
    a. There should be at least 7 periods for every teacher per week i.e., equal distribution of subjects within teachers.

The main focus is often directed toward avoiding the infeasibility or conflicts of any kind like having two subjects at the same time, etc. To avoid using complex functions and mining algorithms, this paper considers a simple to implement and a practical conflict operator to ensure no conflicts.

C. SELECTION

Genetic algorithms are population-based metaheuristics that start with a pool of solutions to select from. Creating an initial population starts with an empty timetable. Most of the solutions proposed in the literature[5] rely on random assignment of events to create an initial population. However, suggested that random generation methods do not, in many cases, guarantee to produce good quality or even feasible solutions. A set of feasible as well as infeasible solutions is obtained in this case. In some cases, the set of infeasible solutions is not discarded but repaired to turn them to feasible solutions using some heuristics, which requires more computational time.

Here, the idea is to control the entry of the chromosomes to the timeslots so that it satisfies the actual hour restriction specified by the university [6]. The chromosomes are randomly generated but are to be within the limits of the KTU norms. To do that we use two limits, minimum and maximum, which inputs the maximum and the minimum number of periods per week that a subject to be taken as per the University norms. The selection procedure is a two-way process.

At first, we fill the slots chromosome-by-chromosome these limits act as a funnel to allow only the subjects only up to the minimum limit, which forms a part of soft constraints.

Secondly, only after satisfying the minimum limits of all the subjects per week criteria, the remaining slots in the timetable are filled by not violating the maximum limit, which forms the other part of the soft constraints. Thereby laying the foundations of the constraints.

An advantage of starting with good initial feasible solutions is increasing the probability of directing the search toward better regions of the search space to help further convergence toward better solutions. Here, we introduce such a method to sort out to better the entry of the set of chromosomes.

D. FITNESS VALUE

Fitness functions are functions calculated for every candidate solution to measure how "fit" or "good" it is. This function is problem-specific and does not have a standard formula for calculation[7]. In university timetabling problems, common fitness functions are based on calculating the number of unscheduled events or conflicts. In this paper, a feasible solution is one that does not include any conflict.

To fully understand the function, let us consider the following:

The chromosome of the timetable's elements is constructed into a group of elements [8] (C). According to this paper, elements comprise two types of data: Subjects (S), and Teachers (T). A set of C will represent constituent elements of chromosomes. So, C = {S, T} will represent the elements of the chromosome. And batches B should be a collection of chromosomes. Each of S, B, and T has a subset of each own. It can be shown as S = {S0, S1, S2, …, Ss} which will represent a set of s subjects. In the same way for a set of g student groups writing the same exam will do as T = {T0, T1, T2, …, Tt} and for batches B = {B0, B1, B2, …, Bb}.

The chromosome will be examined by a Fitness Function (F). This function constructs from hard constraints and soft constraints. The objective of this function is to determine how well the chromosome meets requirements. The function can be represented as,

$$F(x)= \sum_{i=1}^{5} cost_i^{hard} + \sum_{i=1}^{1} cost_i^{soft} \quad \text{..... (1)}$$

Where,

F(x): Fitness function of x^{th} generation

w_i^{hard}: Weight of i^{th} hard constraint

w_i^{soft}: Weight of i^{th} soft constraint

From the equation Eq.(1), we can expand the following as,

$$cost_1^{hrd} = \sum_{s=1}^{s} ContinuousLabPeriods[(S_s,S_{s+1})] \quad ....(2)$$

$$cost_2^{hrd} = \sum_{t=1}^{t}\sum_{s=1}^{s} Teach.UniqueSubj.Prd\ [(S_s,T_t)] \quad ....(3)$$

$$cost_3^{hrd} = \sum_{t=1}^{t}\sum_{s=1}^{s} Subj.UniqueTeach.Prd\ [(S_s,T_t)] \quad ....(4)$$

$$cost_4^{hrd} = \sum_{b=1}^{b}\sum_{t=1}^{t}\sum_{s=1}^{s} UniqueSub.Teach.ForBatch\ [((S_s,T_t),B_b)] \quad ....(5)$$

$$cost_5^{hrd} = \sum_{b=1}^{b}\sum_{t=1}^{t}\sum_{s=1}^{s} Subj.Limitperweek[((S_s,T_t),B_b)] \quad ....(6)$$

$$cost_1^{sft} = \sum_{t=1}^{t}\sum_{s=1}^{s} Teach.Limit\ [(S_s,T_t)] \quad ....(7)$$

where the value of each function can be explained as seen below:

ContinuousLabPeriods[(S_s,S_{s+1})]

A step value is used for lab hours so that it is always allotted continuously. From Eq. (2)

Teach.UniqueSubj.Prd [(S_s,T_t)]

Equal 1: $C[T_t] == C[T_{t+1}]$

Equal -1: otherwise. From Eq. (3)

Subj.UniqueTeach.Prd [(S_s,T_t)]

Equal 1: $C[S_s] == C[S_{s+1}]$

Equal -1: otherwise. From Eq. (4)

UniqueSub.Teach.ForBatch [$((S_s,T_t),B_b)$]

Equal 1: $C[S_s, T_t] == C[S_s, T_t]$

Equal -1: otherwise. From Eq. (5)

Subj.Limitperweek[$((S_s,T_t),B_b)$]

The cost is not used since this constraint is used inside the selection part of GA and is used as an entry control in order to simplify this algorithm. From Eq. (6)

Teach.Limit [(S_s,T_t)]

Equal 1: $C[T_t] >= 7$

Equal -1: otherwise. From Eq. (7)

A reward-punishment system is being followed and is implemented on the conflict operator present in every chromosome. Every time a conflict has occurred, the conflict operator is adjusted. Inside the algorithm, every time the timetable as a whole is traversed, the number of conflicts is recorded and added to the operator. And whenever a conflict is resolved after crossover or mutation, the operator is decreased by the value of the number of conflicts. The process continues until the no conflicts occur i.e., the operator becomes zero for the whole timetable for all the batches.

E. CROSSOVER

Crossover is a method that exchanges an element of two chromosomes. There is no unified standard method of crossover. It has various types and forms and here we use a method based on the two-point crossover. This crossover focuses on the resolution of conflicts and the two points are defined using the conflict operator present in the chromosome. The operator[2] simply checks the conflict operator value and exchanges the chromosome altogether to produce a new set that after many iterations satisfies the constraints.

Consider this example to fully understand the concept. (Fig.3)

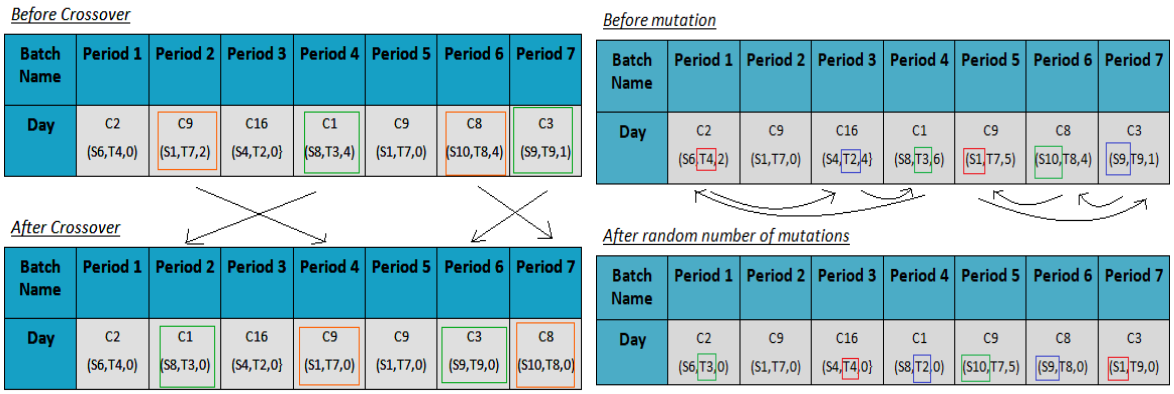

Fig.4. Proposed method for the working of mutation. Fig.3. Proposed method for the working of crossover.

Here, there are two crossovers taking place. First, C9 and C1 are being exchanged, resulting in no conflict. Similarly, the chromosomes C8 and C3 are exchanged to resolve the conflicts.

F. MUTATION

The mutation is the last step of improving a chromosome. The mutation is similar to crossover except that it makes changes in a chromosome itself. [5]Most of the mutation strategies follow a random selection approach (e.g. roulette wheel) to make modifications. But here a simple descent search heuristic is applied as a local search to improve the chromosome. The conflict value in the chromosome is checked and if it checks for conflict, the mutation is done until it resolves, in two ways.

Let us see an example to comprehend the method as in (Fig.4).

Here, within a chromosome, the 2 attributes Subject ID and Teacher ID can be mutated ie, interchanged from within the chromosome so that new varieties can be obtained. The example shows 7 chromosomes, out of which 6 have conflicts of various magnitude. After various iterations, it is found that C2, C16, and C1 interchanging teacher attributes resulted in no conflict. Similarly, C9, C8, and C3 changing the Subject attribute among itself resulted in the same.

III.IMPLEMENTATION

A. USER INTERFACE MODULE

This is the part where the user interacts with the app. The interface can be seen in the figure. (Fig.5)There are mainly 3 parts to this:

1. *Batches*: This is the part where a new batch can be added and under this module two sub-modules of Subjects and Teachers for the subject can be given.
2. *Subjects*: Here, the ID of the subject can be given as input along with the Subject per week limit specified by KTU. The inputs are typed in, instead of dropdowns because the nature of the criteria is dynamic and this method is better suited for that. One batch can have as many subjects as needed.
3. *Teachers*: Here, the ID of the teacher concerning the subjects can be given. One subject can have as many teachers as needed.

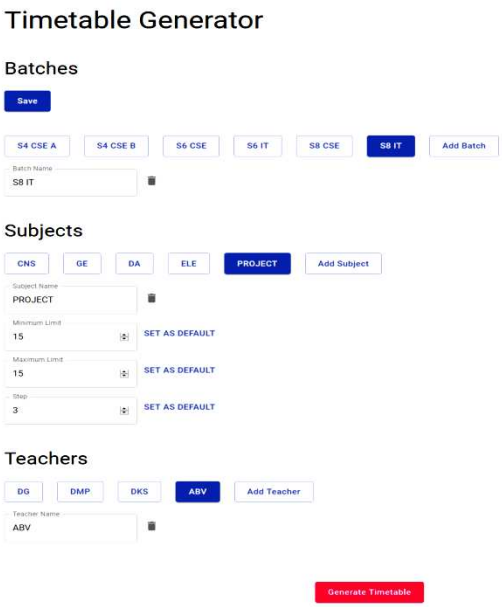

Figure:5 **Frontend interface of the Webapp**

There is an additional save button that when clicked, appends the input in a JSON file. It makes sure that data is not lost over time.

B. SERVER-SIDE MODULE

The algorithm (Fig.6) gives a full picture of the method.

Algorithm

1. Read input data i.e., batch details, subject details, and teachers' details.
2. Create the population matrix with dimensions [no of batches * no of days* no of hours per day].i.e., P= [B_1, B_2, B_3, ...] where B = [C_1, C_2,...] and C = [(S_1, T_1, c_1), (S_2, T_2, c_2), ...].
3. Repeat the following until no conflict occurs.
   - 3.1.Populate the matrix randomly with each element containing [Batch, Subject, Teacher] chromosomes, but within the minimum and maximum limits of the subjects-per-week.
   - 3.2. Check for conflicts.
   - 3.3. Repeat the following until no conflict occurs.
     - 3.3.1. Implement Crossover method by changing the swapping one subject with conflict with another, as explained in (*III.C*).
     - 3.3.2. Implement first way of Mutation method by exchanging the teachers of the subjects with conflicts with one another, as explained in (*III.C*).
     - 3.3.3. Implement second way of Mutation method by exchanging the subjects with conflicts with one another, as explained in (*III.C*).
4. Print the population matrix as a time table.

Fig.6. Modified GA used in this paper.

The main processing of the given inputs is done at this part. The main sub-modules of server-side programming are as follows.

First, the data from the frontend is loaded using get request. The data stored, is taken and processed using the modified version of the genetic algorithm. The steps progress in the order of:

The initiating of the population is taken place at first. Batch-wise selection of subject along with its faculties is chosen randomly but the subjects-per-week limit is ensured here itself to satisfy the constraint.

After filling the table, conflict resolution is done using crossover and mutation steps explained in the above sections. Conflict is continuously checked for and after many iterations, the output is generated. This is sent to the frontend again as a JSON and the result is displayed.

IV. TOOLS USED

The following software libraries have been used to implement a React-Flask Web App that could generate timetable based on the university norms.

1. React JS: A frontend library based on javascript that can create interactive user interfaces.
2. Flask: A Python-based micro-framework
3. Python: A high-level programming language that is used within Flask.
4. Heroku: A Platform as a Service that deploys the build in the cloud.

V. RESULTS AND DISCUSSION

A screenshot of the output produced using the method explained in the above sections, is shown in the figure.(Fig.7)

S4 CSE A	Hour 1	Hour 2	Hour 3	Hour 4	Hour 5	Hour 6	Hour 7
Day 1	LAB RRM	LAB RRM	LAB RRM	OODP RRM	PDD IB	OS RMV	COA TG
Day 2	COA TG	MATHS JVP	COA TG	PDD IB	OS RMV	COA TG	OS RMV
Day 3	OS RMV	MATHS JVP	OODP RRM	MATHS JVP	LAB GVM	LAB GVM	LAB GVM
Day 4	PDD IB	MATHS JVP	BE GTR	OODP RRM	BE GTR	OODP RRM	PDD IB
Day 5	BE GTR	BE GTR	COA TG	BE GTR	MATHS JVP	OS RMV	OODP RRM

S4 CSE B	Hour 1	Hour 2	Hour 3	Hour 4	Hour 5	Hour 6	Hour 7
Day 1	PDD DG	PDD DG	COA SMS	MATHS TGJ	LAB AN	LAB AN	LAB AN
Day 2	BE GTR	OS ART	LAB IB	LAB SMS	LAB IB	BE GTR	OS ART
Day 3	OODP RRM	COA SMS	BE GTR	OS ART	COA SMS	PDD DG	COA SMS
Day 4	OODP RRM	BE GTR	OODP RRM	MATHS TGJ	OODP RRM	OS ART	MATHS TGJ
Day 5	PDD DG	MATHS TGJ	COA SMS	MATHS TGJ	OS ART	OODP RRM	BE GTR

S6 CSE	Hour 1	Hour 2	Hour 3	Hour 4	Hour 5	Hour 6	Hour 7
Day 1	DAA SKM	POM LCL	CN AJ	SEPM MTI	WT VP	CN AJ	CD UKN
Day 2	CD UKN	DAA SKM	POM LCL	POM LCL	SEPM MTI	WT VP	POM LCL
Day 3	SEPM MTI	DAA SKM	DAA SKM	CD UKN	LAB DG	LAB ART	LAB DG
Day 4	WT VP	LAB ART	LAB ART	LAB DG	WT VP	CN AJ	CD UKN
Day 5	CN AJ	SEPM MTI	CN AJ	SEPM MTI	POM LCL	DAA SKM	CD UKN

S6 IT	Hour 1	Hour 2	Hour 3	Hour 4	Hour 5	Hour 6	Hour 7
Day 1	DS GVM	CD NKN	DM DKS	WT VP	LAB ST	LAB ST	LAB ST
Day 2	DM DKS	LAB JS	LAB JS	LAB JS	CD NKN	SPM AJ	WT VP
Day 3	POM GTR	SPM AJ	SPM AJ	CD NKN	WT VP	POM GTR	DM DKS
Day 4	DS GVM	DS GVM	SPM AJ	POM GTR	CD NKN	POM GTR	WT VP
Day 5	DS GVM	DM DKS	WT VP	DM DKS	DS GVM	POM GTR	SPM AJ

S8 CSE	Hour 1	Hour 2	Hour 3	Hour 4	Hour 5	Hour 6	Hour 7
Day 1	PROJECT RR	PROJECT RR	PROJECT RR	PROJECT RR	PROJECT RR	PROJECT RR	GE NA
Day 2	GE NA	ES SM	PROJECT RR	PROJECT RR	PROJECT RR	PROJECT MTI	PROJECT RR
Day 3	ELE JG	PROJECT RR	ES SM	ELE JG	GE NA	DMW ST	DMW ST
Day 4	PROJECT RR	PROJECT RR	PROJECT RR	DMW ST	ELE JG	ES SM	GE SOS
Day 5	ES SM	ELE SUV	DMW ST	ELE SUV	GE NA	DMW ST	ES SM

S8 IT	Hour 1	Hour 2	Hour 3	Hour 4	Hour 5	Hour 6	Hour 7
Day 1	ELE DKS	PROJECT ABV	PROJECT ABV	PROJECT ABV	GE HN	CNS DMP	DA RR
Day 2	PROJECT ABV	PROJECT ABV	PROJECT ABV	GE HN	CNS DMP	DA RR	CNS DMP
Day 3	ELE GVM	ELE DKS	DA RR	PROJECT DMP	PROJECT DMP	PROJECT DMP	DA RR
Day 4	PROJECT DG	PROJECT ABV	PROJECT ABV	GE HN	PROJECT DMP	PROJECT DMP	PROJECT DMP
Day 5	ELE DKS	CNS DMP	GE HN	DA RR	ELE DKS	CNS DMP	GE HN

Fig.7. Output Batch-wise Timetable of the Webapp

Here, the output obtained was batch-wise timetables of 6 different batches, according to the input given. Various batches can be given at the same time.

However, this algorithm was tested on the Computer Science and Engineering department of Adi Shankara Institute of Engineering and Technology, the crossover and mutation rate are as follows:

From the table (Table.1) we can infer that the rate of mutation exceeds the rate of crossover. This shows how this algorithm optimizes the output using higher chance of mutation.

Table.1

The rate(%) of crossover and mutation are found for 6 batches within the department	
Crossover% (upto 2 decimal places)	**Mutation% (upto 2 decimal places)**
20.00	80.00
17.31	82.69
33.33	66.67
21.05	78.95
23.52	76.48

VI. CITATIONS

To propose a successful solution to a difficult problem like np-hard timetable scheduling, it is required to look back at the effort that is done to solve the problem and to review it. University course timetabling problems were defined by Assi, and Halawi[1] as multi-dimensional problems, in which several students and professors are assigned to lectures and events.

Burke and Varley [9] made many efforts toward defining and discovering the different dimensions and requirements of the space allocation problem inside ninety-six universities and academic institutions in the United Kingdom.

Al-Jarrah, Ahmad [4] built a university course timetable using a hybrid algorithm. They used the genetic algorithm and iterated local search to avoid getting trapped in local optima.

Saptarini and Suasnawa [3], developed an adaptive tabu search, in which an initial timetable was constructed using a greedy search heuristic.

Genetic algorithms were chosen to solve the problem of this paper because they are known for their robustness in solving complex combinatorial problems and their flexibility and ability to search in complex, large spaces.

CONCLUSION AND FUTURE SCOPE

University timetabling is a hard problem to solve, especially, in a large university. In this work, we have identified the main constraints of the University and have hardcoded it to incorporate it into the genetic algorithm. By the use of simple operators to avoid random sampling and a weight-based fitness function, the algorithm was able to improve its output and an optimized form was obtained. The results show this genetic algorithm works well in creating a timetable that fits the criteria of the institution.

Timetabling has always been a field that found ways to imbibe new technology to create simpler methods. This paper used an alternative take on the traditional genetic algorithm. However, the popular fields of Data Science and Machine learning have made possible to use more complex algorithms and methods having higher efficiency.

Real-time data processing is another venue where this problem can be solved by the use of real-time data and dynamically process it to produce a unique timetable depending on the situation.

References

[1] Maram Assi, and Bahia Halawi, and Ramzi A. Haraty, Genetic Algorithm Analysis using the Graph Coloring Method for Solving the University Timetable Problem, Department of Computer Science, Lebanese American University,(2018)

[2] Esraa A. Abdelhalim, Ghada A. El Khayat, A Utilization-based Genetic Algorithm for Solving the University Timetabling Problem (UGA), Alexandria University,2017

[3] N G A P H Saptarini, I W Suasnawa, P I Ciptayani, Senior high school course scheduling using heuristic algorithm, The 2nd International Joint Conference on Science and Technology (IJCST), (2017)

[4] Mohammad A. Al-Jarrah, Ahmad A. Al-Sawalqah, and Sami F. Al-Hamdan, "DEVELOPING A COURSE TIMETABLE SYSTEM FOR ACADEMIC DEPARTMENTS USING HYBRID GENETIC ALGORITHM", Jordanian Journal of Computers and Information Technology (JJCIT), April (2017)

[5] Jumoke Soyemi, John Akinode, Samson Oloruntoba, "Electronic Lecture Timetable Scheduler using Genetic Algorithm", IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 2017

[6] U. M. Modibbo, I. Umar, M. Mijinyawa, R. Hafisu, "Genetic Algorithm for Solving University Timetabling Problem",(2017)

[7] A. Mahiba, C. Durai, Genetic algorithm with search bank strategies for university course timetabling problem, Int. Conf. Model Optim. Comput. (ICMOC 2012) 38 (2012) 253–263.

[8] S. Abdullah, H. Turabieh, B. McCollum, E. Burke, An investigation of a genetic algorithm and sequential local search approach for curriculum-based course timetabling problems, Multidisciplinary Int. Conf. Scheduling: Theory Appl. (2009)

[9] E. Burke, J. Kingston, K. Jackson, R. Weare, Automated university timetabling: the state of the art, Comput. J. 40 (9) (1997) 565–571.

