Course Scheduling Optimization

Module Code	Computational Intelligence Optimization (032024-IMK)
Intake Code	APDMF2310AI
Lecturer Name	Dr. Imran Medi
Hand in Date	16th of June 2024
Student Name	Amanov Nurlan
Student Number	TP077526

Table of Contents

Abstract **3**

1. Introduction **4**

1.1. Literature Review **5**

1.2. Methodology **7**

1.3. Experiment, Implementation & Simulation **11**

1.4. Results, Analysis & Discussion **14**

1.5. Conclusions & Future Research **23**

References **24**

ABSTRACT

This paper addresses the challenging problem of optimising course schedules for part-time postgraduate programmes at Asia Pacific University (APU) using computational intelligence (CI), in particular genetic algorithm (GA). The objective is to design an efficient schedule that minimises costs, ensures that students complete courses in a timely manner, and optimises resource allocation. Constraints such as limiting the number of modules available per set, not being able to schedule the same module sequentially, and ensuring that certain modules are only offered in certain sets make the task more challenging. The ability of GA to generate schedules that meet these constraints and minimise conflicts shows its effectiveness. Experimental results show that the GA-based method significantly improves the efficiency of scheduling, with the generated schedule satisfying all requirements and having minimal violations. The paper concludes by noting the possibility of further improving the algorithm to achieve zero violations and future research directions to improve the scheduling optimisation.

INTRODUCTION

Creating efficient schedules for universities is a complex and multifaceted optimisation problem, especially for part-time postgraduate programmes. Thus, this paper aims to create a solution based on Computational Intelligence (CI), using Genetic Algorithm (GA). The challenge of developing a schedule for the part-time programmes at Asia Pacific University (APU) involves minimising costs, ensuring timely completion of the programme for students and optimising resource allocation. The problem is complicated by specific requirements such as limiting the number of modules, avoiding sequential scheduling of the same module, and ensuring that certain modules are only offered at a certain intake time.

LITERATURE REVIEW

Ansari and Saubari (2020) describe the use of a Genetic Algorithm (GA) application to solve a scheduling problem in a university. The main goal is to generate a timetable that avoids conflicts such as lecture times, faculty availability and room allocation. In this paper, the components of Genetic Algorithm: chromosomes, which are schedules with genes; initial populations, which are randomly generated; fitness values, which are calculated based on scheduling conflicts, aiming for fewer violations; best chromosomes, which are selected for the next generation; crossover combines parts of two chromosomes and mutation introduces random changes to maintain diversity. As a result, the best schedule has fewer conflicts, but because of the small population and generations, some conflicts remain. The paper demonstrates the use of genetic algorithms for class scheduling and shows that they can automate and optimise the process with reduced conflicts. The methodology is well structured, and the results show that GA can significantly improve scheduling efficiency.

Another work by Wang (2018) discusses the same topic. This work aims to develop an improved adaptive Genetic Algorithm (GA) that can help solve problems. The growing requirements make traditional methods inefficient. However, the improved GA has soft and hard constraints to maximise its efficiency. Experimental results show that this algorithm outperforms the traditional GA by reducing the scheduling time and improving the suitability performance. Class scheduling is an optimisation problem with many constraints such as time, place and instructors that must be properly allocated. The experiments compare the improved adaptive GA with the traditional adaptive GA. The convergence metrics include:

- average suitability,
- maximum fitness,

- number of iterations for convergence.

The study shows that the shortest scheduling time can be obtained with an optimal population size of about 150 classes. The improved adaptive GA effectively optimises the course scheduling, handling complex constraints better than traditional methods. Future work should consider additional factors and explore new algorithms to further improve scheduling efficiency.

METHODOLOGY

Genetic Algorithm (GA) was chosen for this work. Genetic Algorithm are part of a class of evolutionary algorithms that mimics the process of natural selection. Being inspired by the process of natural selection, the Genetic Algorithm is used to generate solutions to search and optimisation problems. The algorithm starts with a randomly generated population of individuals that are candidate solutions. In this generated population, each individual is represented by a chromosome, where a binary string is used to encode it, but other encodings are also possible, for example permutations. To estimate the solution and its closest approximation to the optimal solution, fitness function is used and due to fitness score the probability of selection of an individual is determined. The genetic algorithm includes the following operators:

1) Individuals are selected using a selection operator, which helps to select individuals from the population. This selection is necessary to create a mating pool using their fitness as a basis. Some of the most common methods of selection are:

- Roulette wheel selection, where individuals are selected with a probability proportional to their fitness.
- Tournament selection is selection where a subset of individuals is selected at random, where the best individual is selected from this subset.
- Rank-based selection is selection where selection is based on a ranking based on suitability.

2) The crossover operator combines two parental chromosomes, where with their union, one or more offspring are produced. Common methods of crossover are as follows:

- One-point crossover is crossover where the crossing point is chosen randomly, after the parts of the parents change places, which helps to create offspring.

- Two-point crossover is crossover where two points of crossover are chosen and the segments between these points change places.
- Uniform crossover is crossover where each gene from the parent pair is chosen randomly for the offspring.

3) A mutation operator is an operator that by making random changes to individual chromosomes, helps maintain genetic diversity. Common methods of mutation are:

- Bit flip mutation is where a bit is randomly selected that switches between 0 and 1 in binary coding.
- A swap mutation is a mutation where two positions in a chromosome are swapped.
- Gaussian mutation, where a gene is randomly selected to which a value is added that is Gaussian distributed.

The resulting offspring are intergrown into the population, where less fit individuals are replaced. As a result, the following parameters are defined in the Genetic Algorithm code:

1) Population Size - number of individuals in the population.
2) Number of Generations - number of iterations the algorithm will run.
3) Crossover Rate - probability of crossover occurring between pairs of individuals.
4) Mutation Rate - probability of mutation occurring for everyone.

The dataset used for our work is also used for our work, where the following analyses are given below:

1) There are a total of 5 intakes which are January, March, May, August, and October.
2) There are two datasets with students and their choice of modules, where the datasets are

divided into programmes which are Artificial Intelligence (AI) and DSBA (Data Science and Business Analytics).

3) There are a total of 13 modules in the AI programme, which are Artificial Intelligence (AI), Image Processing and Computer Vision (IPCV), Business Intelligence Systems (BIS), Multivariate Methods for Data Analysis (MMDA), Fuzzy Logic (FL), Computational Intelligence Optimization (CIO), Applied Machine Learning (AML), Applied Robotics (AR), Pattern Recognition (PR), Research Methodology in Computing and Engineering (RMCE), Deep Learning (DL), Expert Systems and Knowledge Engineering (ESKE), and Natural Language Processing (NLP), where of those subjects 7 core subjects – AI, IPCV, FL, AML, CIO, NLP, RMCE, and the other 6 subjects are electives, where part-time students select 3 of them – AR, PR, ESKE, BIS, MMDA and DL.

4) There are a total of 18 modules in the DSBA programme, which are Big Data Analytics and Technologies (BDAT), Data Management (DM), Business Intelligence Systems (BIS), Multivariate Methods for Data Analysis (MMDA), Time Series Analysis and Forecasting (TSF), Behavioural Science, Social Media and Marketing Analytics (BSSMMA), Applied Machine Learning (AML), Advanced Business Analytics and Visualisation (ABAV), Data Analytical Programming (DAP), Research Methodology for Capstone Project (RMCP), Deep Learning (DL), Cloud Infrastructure and Services (CIS), Operational Research and Optimisation (ORO), Strategies in Emerging Markets (SEM), Multilevel Data Analysis (MDA), Data Protection and Management (DPM), Building IoT Application (BIA), Natural Language Processing (NLP) where 8 subjects are core – BDAT, DM, BIS, AML, RMCP, MMDA, DAP, ABAV. DSBA Students choose two pathways – Business Intelligence pathway, where 2 specialization core subjects – BSSMMA, TSF and 3 elective

– MDA, SEM, ORO, and the other Data Engineering pathway, where 2 specialization core subjects – CIS, DL, and 3 elective – NLP, DPM, BIA.

EXPERIMENT, IMPLEMENTATIONS & SIMULATION

The objectives of this paper are to provide a course schedule using the data provided, provide a recommended course schedule stating the modules to be offered each intake, as well as the number of violations present, and take into account key aspects:

- The University would like to limit the number of modules offered per intake to a maximum of 4 per programme – however, the fewer the better.
- A module should not run consecutively (i.e. if it is scheduled for January, it should not be scheduled for March).
- Research Methodology in Computing and Engineering (RMCE) and Research Methodology for Capstone Project (RMCP) must be scheduled only in January, May and October.
- The number of students who do not have a module to take for a given intake should be minimized (students who have taken the RM module are allowed to skip an intake).

Taking into account the first key aspect, each intake will have 4 modules, thus the following should be noted:

1) AI and DSBA programmes have common modules - BIS, MMDA, AML, DL, NLP.
2) Since there are two tracks in DSBA programme, based on the actual timetable developed by the university, the following subjects will be defined as one single subject - BSSMA and CIS and name it BSSMA/CIS; TSF and DL and name it DL; MDA and NLP and name it NLP; SEM and DPM and name it SEM/DPM; ORO and BIA and name it ORO/BIA. Because DL and NLP are common modules, the combinations TSF and DL are taken as DL, and MDA and NLP are taken as NLP for code operation.

3) Based on the actual timetable and the key aspect, solution development, RMCE and RMCP should be offered in the timetable three times in January, May, October.

4) Subject popularity will be used to determine the remaining places, where the most popular module will appear three times in the timetable, the other four after it will appear twice, with RMCE and RMCP not included in this top. The module with the lowest popularity will be discarded.

A 3-level chromosome was developed for this scenario (Figure 1), where each intakes schedule is built for each programme, so that means each programme for intake has their modules. Example, in January for AI program there is DL, AML, BIS, RMCE subjects.

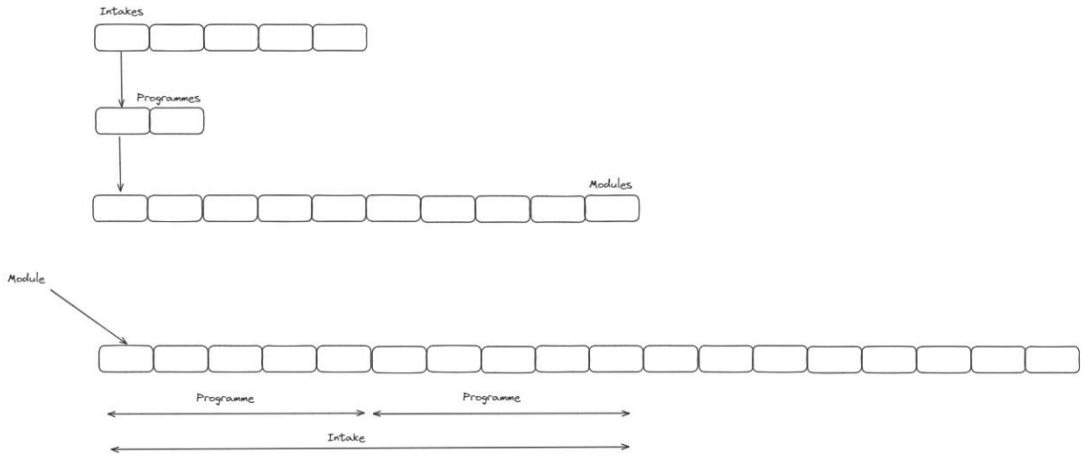

Figure 1. A 3-level Chromosome

The following function will evaluate the fitness of the above chromosome, where (Fitness = exactModulesPerIntakeViolations + consecutiveModuleViolation + consecutiveModuleAcrossIntakesViolation + sharedModuleViolation + rmModuleViolation + missingRequiredModuleViolation + duplicateModuleViolation + modulePreferenceViolation), where:

- exactModulesPerIntakeViolations – each intake has 4 modules, where violation is added

on violation.

- consecutiveModuleViolation – the same module cannot be in the same intake where a violation is added on violation.
- consecutiveModuleAcrossIntakesViolation – one and the same module cannot appear in two modules in a row, for example in January and March, where a violation is added in case of violation.
- sharedModuleViolation – common modules between two programmes must be offered in the same intake, where a violation is added in case of violation.
- rmModuleViolation – RMCE and RMCP should be in January, May and October intakes.
- missingRequiredModuleViolation – each module from the final list of modules must appear at least once, where a violation is added in case of violation.
- duplicateModuleViolation – no duplicate module proposals within the same intake, where a violation is added on infringement.
- modulePreferenceViolation – the frequency of occurrence of the module that was set to it must be respected, where violation is added in case of violation.

RESULTS, ANALYSIS & DISCUSSION

For this work, there are datasets ("MScAI.csv" and "DSBA.csv") that include information about student name, intake course, course name, study mode, module code and module name. However, to get the list of popularity of the courses, to get the frequency of their selection, the dataset will go through preprocessing:

- Import “pandas” library. “Pandas” is a powerful data manipulation and analysis library for Python.

```
import pandas as pd
```

- Load CSV files into DataFrames.

```
dsba_df = pd.read_csv('DSBA.csv')
ai_df = pd.read_csv('MScAI.csv')
```

- Map full module names to their short form.

```
dsba_mapping = { ...
}
ai_mapping = { ...
}
```

- Define list of all modules.

```
all_dsba_modules = ["BDAT", "DM", "BIS", "MMDA", "DL", "BSSMA/CIS", "AML",
"ABAV", "DAP", "RMCP", "ORO/BIA", "SEM/DPM", "NLP"]
all_ai_modules = ["AI", "IPCV", "BIS", "MMDA", "FL", "CIO", "AML", "AR",
"PR", "RMCE", "DL", "ESKE", "NLP"]
```

- Function to process a given DataFrame by keeping only name and module name columns, then mapping full module name to their short forms, then adding programme column, then count the occurrence of each module, then make sure that every module is included, even with zero count.

```
def process_dataframe(df, mapping, programme_name, all_modules):
    df = df.loc[:, ['NAME', 'MODULE NAME']]
    df.loc[:, 'MODULE NAME'] = df['MODULE NAME'].map(mapping)
    df.loc[:, 'PROGRAMME'] = programme_name
```

```
    count_df = df.groupby(['PROGRAMME', 'MODULE
NAME']).size().reset_index(name='COUNT')
    all_modules_df = pd.DataFrame({'MODULE NAME': all_modules, 'PROGRAMME':
programme_name})
    result_df = all_modules_df.merge(count_df, on=['MODULE NAME',
'PROGRAMME'], how='left').fillna(0)
    return result_df
```

- Process AI and DSBA DataFrames.

```
dsba_processed = process_dataframe(dsba_df, dsba_mapping, 'DSBA',
all_dsba_modules)
ai_processed = process_dataframe(ai_df, ai_mapping, 'AI', all_ai_modules)
```

- Combine the processed DataFrames.

```
final_result_df = pd.concat([dsba_processed, ai_processed])
```

- Save the final result into one CSV file.

```
final_result_df.to_csv('processed_modules_combined.csv', index=False)
```

The final CSV file contains information about counts of each module, where modules name is written in short form, also combining some of the modules, based on observations made before (Figure 2).

	A	B	C
1	MODULE N	PROGRAM	COUNT
2	BDAT	DSBA	84
3	DM	DSBA	63
4	BIS	DSBA	60
5	MMDA	DSBA	34
6	DL	DSBA	36
7	BSSMA/CI	DSBA	19
8	AML	DSBA	61
9	ABAV	DSBA	46
10	DAP	DSBA	33
11	RMCP	DSBA	22
12	ORO/BIA	DSBA	2
13	SEM/DPM	DSBA	2
14	NLP	DSBA	23
15	AI	AI	84
16	IPCV	AI	63
17	BIS	AI	60
18	MMDA	AI	34
19	FL	AI	19
20	CIO	AI	12
21	AML	AI	61
22	AR	AI	46
23	PR	AI	33
24	RMCE	AI	22
25	DL	AI	17
26	ESKE	AI	7
27	NLP	AI	19

Figure 2. Final CSV File

After the preprocessing of dataset, it is time for the main code. It is worth to mention that "Guarsd.py" from the second week in the Moodle was taken as a reference for the main code.

- First step, import libraries "pandas", "random", "deap", "numpy", "matplotlib".

```
import pandas as pd
import random
from deap import tools, creator, base, algorithms
import numpy as np
import matplotlib.pyplot as plt
```

- Load CSV file.

```
file_path = 'processed_modules_combined.csv'
df = pd.read_csv(file_path)
```

- Extract modules and sort them by count and exclude the last row as it is the least popular module.

```
required_modules_ai = df[df['PROGRAMME'] == 'AI'].sort_values(by='COUNT',
ascending=False)
required_modules_ai = required_modules_ai.iloc[:-1]
required_modules_dsba = df[df['PROGRAMME'] == 'DSBA'].sort_values(by='COUNT',
ascending=False)
required_modules_dsba = required_modules_dsba.iloc[:-1]
```

- Extract modules, ensure required modules are part of these.

```
modules_ai = df[df['PROGRAMME'] == 'AI'].sort_values(by='COUNT',
ascending=False)
modules_ai = modules_ai.iloc[:-1]
modules_dsba = df[df['PROGRAMME'] == 'DSBA'].sort_values(by='COUNT',
ascending=False)
modules_dsba = modules_dsba.iloc[:-1]

required_modules_ai = [module for module in required_modules_ai['MODULE
NAME'] if module in modules_ai['MODULE NAME']]
required_modules_dsba = [module for module in required_modules_dsba['MODULE
NAME'] if module in modules_dsba['MODULE NAME']]
```

- Create a list of modules of each programme.

```
all_ai_modules = modules_ai['MODULE NAME'].tolist()
```

```
all_dsba_modules = modules_dsba['MODULE NAME'].tolist()
```

- Determine top modules but exclude RMCE and RMCP.

```
top_modules_ai = modules_ai[~modules_ai['MODULE
NAME'].isin(['RMCE'])].head(5)
top_modules_dsba = modules_dsba[~modules_dsba['MODULE
NAME'].isin(['RMCP'])].head(5)
```

- Define module preferences by specifying how many times each module should appear.

```
module_preferences_ai = {}
module_preferences_dsba = {}

for i, module in enumerate(top_modules_ai['MODULE NAME']):
    if i < 1:
        module_preferences_ai[module] = 3
    else:
        module_preferences_ai[module] = 2

for i, module in enumerate(top_modules_dsba['MODULE NAME']):
    if i < 1:
        module_preferences_dsba[module] = 3
    else:
        module_preferences_dsba[module] = 2
module_preferences_ai['RMCE'] = 3
module_preferences_dsba['RMCP'] = 3

for module in all_ai_modules:
    if module not in module_preferences_ai:
        module_preferences_ai[module] = 1

for module in all_dsba_modules:
    if module not in module_preferences_dsba:
        module_preferences_dsba[module] = 1
```

- List of intakes and shared modules.

```
intakes = ['Jan', 'Mar', 'May', 'Aug', 'Oct']
shared_modules = ['BIS', 'MMDA', 'AML', 'DL', 'NLP']
```

- “Schedule” class defines how to handle the schedule, including constraints and penalties. It includes methods to calculate the cost of a schedule and print schedule information.

```
class Schedule:
    def __init__(self, modules_ai, modules_dsba, intakes,
required_modules_ai, required_modules_dsba, shared_modules,
                 hardConstraintPenalty, softConstraintPenalty,
module_preferences_ai, module_preferences_dsba):
        ...
    def __len__(self):
        return len(self.intakes) * 4 * 2  # Ensure exactly 4 modules per
intake for AI and DSBA

    def getCost(self, schedule):
        ...
        return (
        violations, exact_modules_per_intake_violations,
consecutive_module_violations, consecutive_module_across_intakes_violations,
shared_module_violations,
        rm_module_violations, missing_required_module_violations,
duplicate_module_violations,
        module_preferences_violations)

    def printScheduleInfo(self, schedule):
        ...
        print("\nViolations:")
        print(f"Exact Modules Per Intake Violations:
{exact_modules_per_intake_violations}")
        print(f"Consecutive Module Violations (within same intake):
{consecutive_module_violations}")
        print(f"Consecutive Module Violations (across intakes):
{consecutive_module_across_intakes_violations}")
        print(f"Shared Module Violations: {shared_module_violations}")
        print(f"RM Module Violations: {rm_module_violations}")
        print(f"Missing Required Module Violations:
{missing_required_module_violations}")
        print(f"Duplicate Module Violations: {duplicate_module_violations}")
        print(f"Module Preferences Violations:
```

```
{module_preferences_violations}")
        print(f"Total Violations: {violations}")
```

- Inside “Schedule” class, each violation defined.

```
exact_modules_per_intake_violations = 0
        for intake in self.intakes:
            if len(intake_modules_ai[intake]) != 4:
                exact_modules_per_intake_violations +=
abs(len(intake_modules_ai[intake]) - 4)
            if len(intake_modules_dsba[intake]) != 4:
                exact_modules_per_intake_violations +=
abs(len(intake_modules_dsba[intake]) - 4)
        violations += exact_modules_per_intake_violations *
self.hardConstraintPenalty
```

- Generic Algorithm setup using DEAP, which includes the population, fitness function, and genetic operators, running the GA, and plotting the results.

```
populationSize = 600
generations = 100
pCrossover = 0.9
pMutation = 0.4

sch = Schedule(modules_ai, modules_dsba, intakes, required_modules_ai,
required_modules_dsba, shared_modules, 100, 10,
               module_preferences_ai, module_preferences_dsba)

random.seed(42)
toolbox = base.Toolbox()
toolbox.register('gene', random.randint, 0, max(len(modules_ai),
len(modules_dsba)) - 1)
creator.create('FitnessMin', base.Fitness, weights=(-1.0,))
creator.create('Individual', list, fitness=creator.FitnessMin)
toolbox.register('individualCreator', tools.initRepeat, creator.Individual,
toolbox.gene, len(sch))
toolbox.register('populationCreator', tools.initRepeat, list,
toolbox.individualCreator)
```

```
def fitnessFunction(individual):
    return sch.getCost(individual)[0],

toolbox.register('evaluate', fitnessFunction)
toolbox.register('select', tools.selTournament, tournsize=3)
toolbox.register('mate', tools.cxOnePoint)
toolbox.register('mutate', tools.mutFlipBit, indpb=1 / len(sch))

startingPop = toolbox.populationCreator(populationSize)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register('min', np.min)
stats.register('avg', np.mean)
hof = tools.HallOfFame(5)
finalPop, logbook = algorithms.eaSimple(startingPop, toolbox, pCrossover,
pMutation, generations, stats, hof, True)
minValue, avgValue = logbook.select('min', 'avg')
plt.plot(minValue, color='red')
plt.plot(avgValue, color='green')
plt.xlabel('generations')
plt.ylabel('min/avg fitness per generation')
plt.show()
best = hof[0]
sch.printScheduleInfo(best)
```

As the result we got plot, total violations and schedule (Figure 3 and Figure 4).

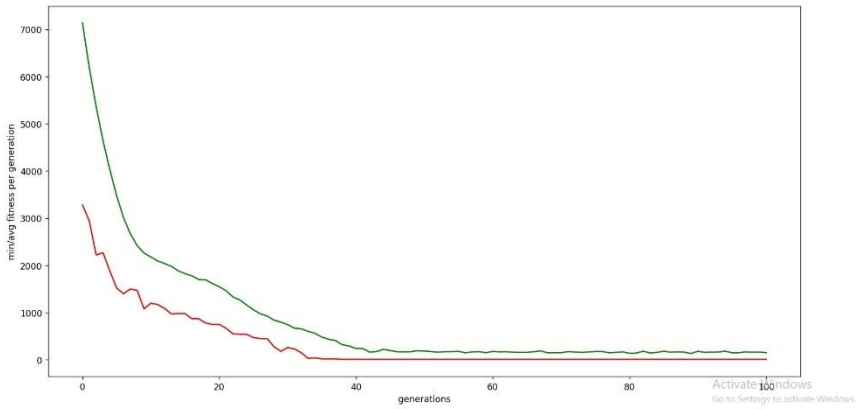

Figure 3. Plot with X-Label of Generations and Y-Label of MIN/AVG Fitness per Generation

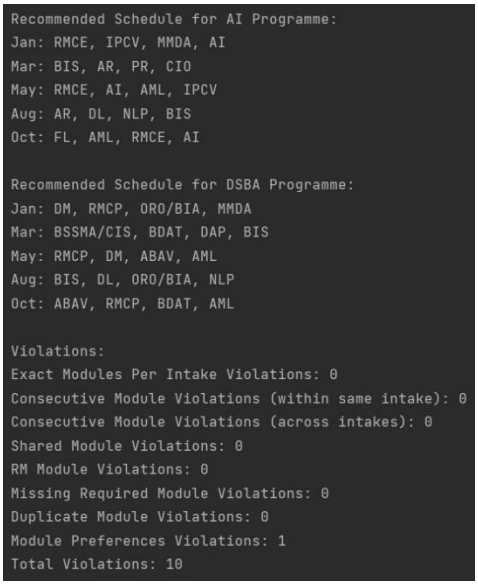

Figure 4. Recommended Schedule for Both Programmes and Violations Number

Table 1. Recommended Schedule for AI Programme

Intake	Module 1	Module 2	Module 3	Module 4
January	RMCE	IPCV	MMDA	AI
March	BIS	AR	PR	CIO
May	RMCE	AI	AML	IPCV
August	AR	DL	NLP	BIS
October	RMCE	AI	AML	FL

Table 2. Recommended Schedule for DSBA Programme

Intake	Module 1	Module 2	Module 3	Module 4
January	RMCP	ORO/BIA	MMDA	DM
March	BSSMA/CIS	BDAT	DAP	BIS
May	RMCP	DM	ABAV	AML
August	BIS	TSF/DL	MDA/NLP	ORO/BIA
October	RMCP	AI	AML	FL

Based on the violations, there is only one which is module preference violations, where ORO/BIA appears twice in DSBA schedule programme, instead it should appear once with ABAV appearing twice. As a result, the schedule meets all the given criteria by having 4 modules each intake, with the same module not occurring in back-to-back intakes and not duplicated twice in the same intake, with common modules occurring in the same intake for both programmes.

CONCLUSIONS & FUTURE RESEARCH

As a result, the developed code showed a good result by developing a schedule that met all the requirements with only 1 violation. It is possible to improve the code and achieve the result of 0 violation. Also, it is worth considering the system of selecting the number of appearances in the schedule by modules, thus improving it can achieve the use of three to four modules instead of just four. The fitness function considers all the criteria that have been stated in a given series. As for future research, this study will be useful as one example of using Genetic Algorithm to create for part-time students.

REFERENCES

Ansari, R. & Saubari, N., 2020. Application of genetic algorithm concept on course scheduling. IOP Conference Series: Materials Science and Engineering, 821(1), p.012043. IOP Publishing.

Emran, n.d. Guard scheduling optimization using a genetic algorithm. [online] Available at: https://lms2.apiit.edu.my/mod/resource/view.php?id=485819 [Accessed 22 June 2024].

Wen-jing, W., 2018. Improved adaptive genetic algorithm for course scheduling in colleges and universities. International Journal of Emerging Technologies in Learning, 13(6).