Experiment and Implementation

In this experiment, the GA was coded using Python programming on Jupyter notebook. The flowchart of the algorithm implementation is as shown in figure 4. DSBA, SE, AI class was created. Each class consists of relevant information such as the module names, function to retrieve student data, hard constraints and soft constraints as stated in section 4.5. Next, GA parameters as shown in Table 1 is initialized. An initial population was created with randomly generated chromosome in equal length. Each chromosome has a maximum length of 25 integers. The integer represents a module to be offered in an intake, with maximum five modules in one Intake. Next, the fitness of each individual is evaluated to identify the best individuals. Next, with the best individuals selected, selection, crossover and mutation were performed. The children produced from this process were then included to the current population to form the next generation. Elitism is applied in this process to replace the current population with the combination of new population and elites. Fitness value was evaluated.The best fitness score was determined and displayed for every 200 iterations. Upon reaching the maximum number of iterations, all three schedules are compared to include the common modules between programs in the same intake.

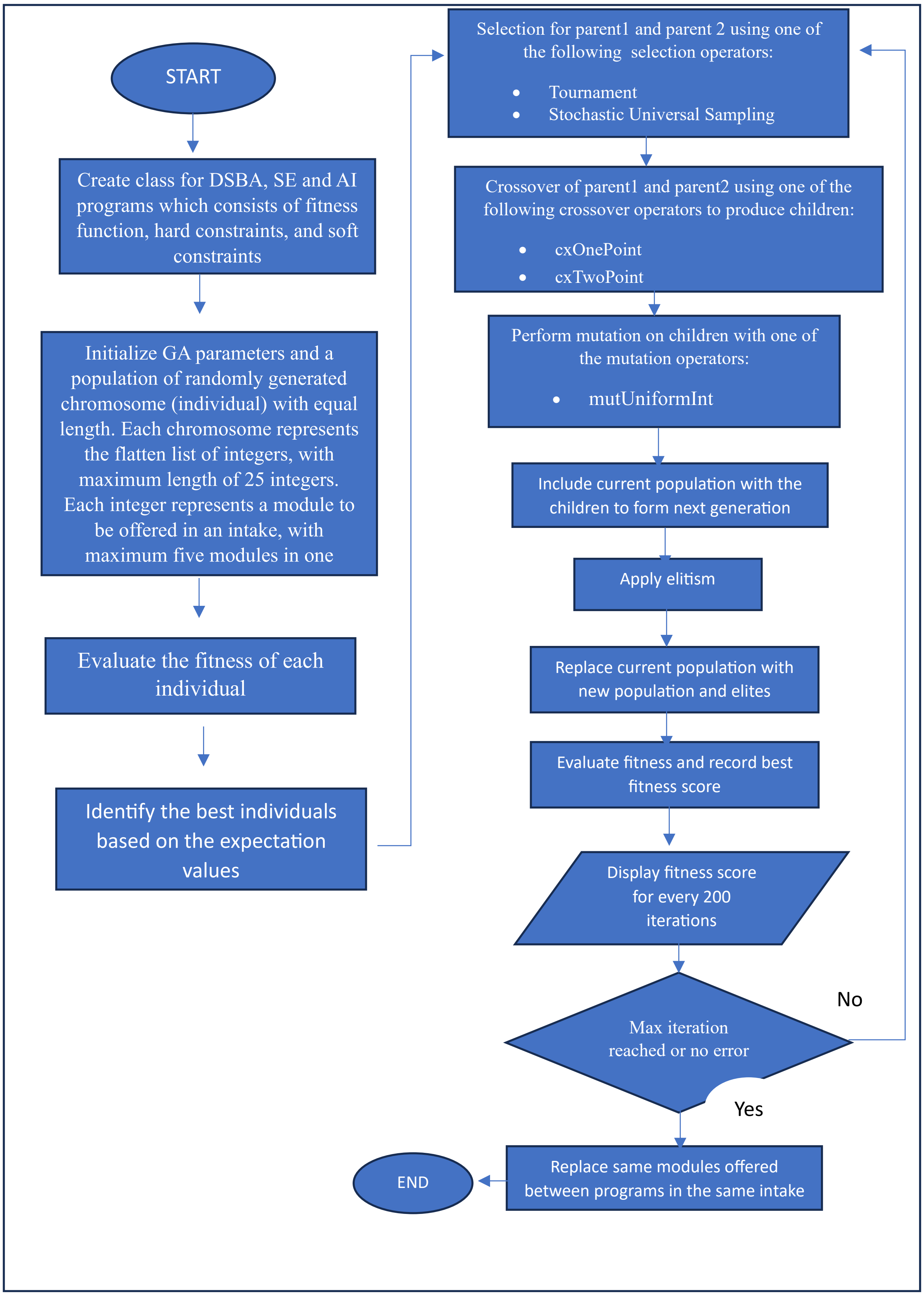

START
Create class for DSBA, SE and AI programs which consists of fitness function, hard constraints, and soft constraints
Initialize GA parameters and a population of randomly generated chromosome (individual) with equal length. Each chromosome represents the flatten list of integers, with maximum length of 25 integers. Each integer represents a module to be offered in an intake, with maximum five modules in one
Evaluate the fitness of each individual
Identify the best individuals based on the expectation values
Selection for parent1 and parent 2 using one of the following selection operators:
• Tournament
• Stochastic Universal Sampling
Crossover of parent1 and parent2 using one of the following crossover operators to produce children:
• cxOnePoint
• cxTwoPoint
Perform mutation on children with one of the mutation operators:
• mutUniformInt
Include current population with the children to form next generation
Apply elitism
Replace current population with new population and elites
Evaluate fitness and record best fitness score
Display fitness score for every 200 iterations
Max iteration reached or no error
No
Yes
Replace same modules offered between programs in the same intake
END

Figure 1 Detail Algorithm Flowchart

Several steps were taken in the code to ensure the required condition is met. Firstly, the module ‘RMCE’ and ‘RMCP’ are hard-coded when generating the individual. This is to ensure that it is offered in the first, third and last intakes.

```
def generateIndividual():
    lst=[[13],[14],[13],[14],[13]]
    for i in range(0,len(lst)):
        num= random.randint(2,3)
        while len(lst[i])<num:
            modulesInd= random.choice(range(0, len(ai.modules)+1))
            lst[i].append(modulesInd)
    chromosome=[mod for sublst in lst for mod in sublst]
    return chromosome
```

Second, the schedule of DSBA program is generated first. This is due to the high number of students enrol to this program compared to the other two. Meanwhile, DSBA program shares five common modules which are ‘AML’, ‘NLP’, ‘MMDA’, ‘DL’, and ‘BIS’ with AI program while shares two common modules (‘NLP’ and ‘BDAT’) with SE program. On the other hand, both AI and SE programs share only one common module, ‘NLP’.

Third, DSBA program consists of two pathways: Business Intelligence Pathway and Data Engineering Pathway. Two separate functions were created to retrieve the student information, as shown below. The modules offered for both pathways are coded as one module for example: 'BSSMA|CIS', 'TSF|DL', 'MDA|NLP', 'SEM|BIA', and 'ORO|DPM', this is to ensure that students from both pathways will have a module to take.

```
def getStuRecordDE(self):  #DE student

    stuDictDE={}
    df=pd.read_excel('DSBA.xlsx')
    df['PATH']= df['INTAKE CODE'].str[14:16]
    df['CODE']= df['MODULE CODE'].str.split('-').str[-1]
    df= pd.DataFrame(df, columns=['NAME', 'PATH','CODE'])
    de= pd.DataFrame(df[df['PATH']=='DE'])
    DE= de.groupby(['NAME','PATH'])['CODE'].apply(list).to_dict()
    stuDictDE.update(DE)

    return stuDictDE

def getStuRecordBI(self):  #BI student

    stuDictBI={}
    df=pd.read_excel('DSBA.xlsx')
    df['PATH']= df['INTAKE CODE'].str[14:16]
    df['CODE']= df['MODULE CODE'].str.split('-').str[-1]
    df= pd.DataFrame(df, columns=['NAME', 'PATH','CODE'])
    bi= pd.DataFrame(df[df['PATH']=='BI'])
    BI= bi.groupby(['NAME','PATH'])['CODE'].apply(list).to_dict()
    stuDictBI.update(BI)

    return stuDictBI
```

Moreover, a different function was coded to ensure that each student takes only one elective modules regardless of the pathway selected, relevant code is shown as follows:

```
# constraint 8: only 1 electives can be taken by each student depending on their pathway #soft constraint

def elecModViolation(self, chromosome):


    sc=self.appendMod(chromosome)
    violation=0
    stuDictDE= self.getStuRecordDE()
    electiveCount=0
    for modTaken in stuDictDE.values():
        for mod in modTaken:
            if mod in self.electiveDE:
                electiveCount+=1

    if electiveCount==1:
        for modTaken in stuDictDE.values():
            for s in sc:
                diff=[mod for mod in s if mod not in modTaken]
                for d in diff:
                    if len(diff)==1 and d in self.electiveDE:
                        violation+=5

    stuDictBI= self.getStuRecordBI()
    electiveCount=0
    for modTaken in stuDictBI.values():
        for mod in modTaken:
            if mod in self.electiveBI:
                electiveCount+=5

    if electiveCount==1:
        for modTaken in stuDictBI.values():
            for s in sc:
                diff=[mod for mod in s if mod not in modTaken]
                for d in diff:
                    if len(diff)==1 and d in self.electiveBI:
                        violation+=5
    return violation
```

Both SE and AI programs share the same set of hard constraints and soft constraints. Specifically, the common hard constraints of all programs are:

1. A module should not be offered consecutively. For example, if ‘AI’ module is offered in January intake, it should not be offered again in March intake.

```python
# Hard constraint 1: module should not run in consecutive intakes
def countConsecutiveModViolation(self, chromosome):

    sc=self.appendMod(chromosome)
    violation = 0

    for i in range(1, len(sc)):
        for mod in sc[i]:
            if mod in sc[i - 1]:
                violation += 5
    return violation
```

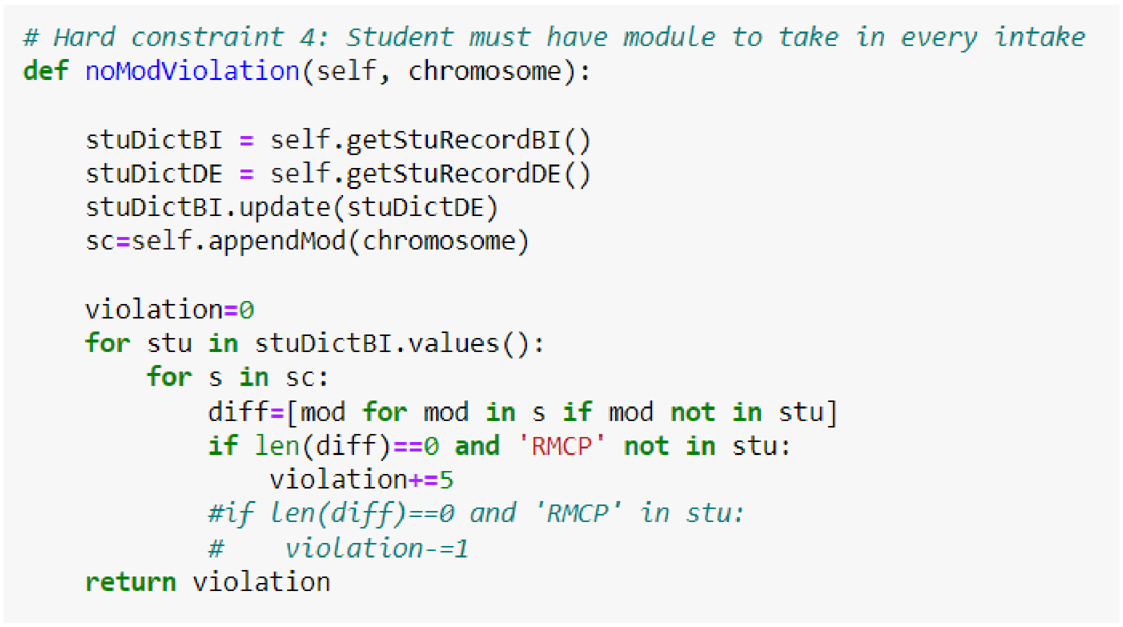
2. The modules offered in each intake should be different.

```python
# Hard constraint 2: modules offered in each intake should be different

def sameModViolation(self, chromosome):

    sc=self.appendMod(chromosome)
    violation=0
    for s in sc:
        for mod in s:
            if s.count(mod)>1:
                violation+=10
    return violation
```

3. The number of modules offered in each intake should be less than 5.

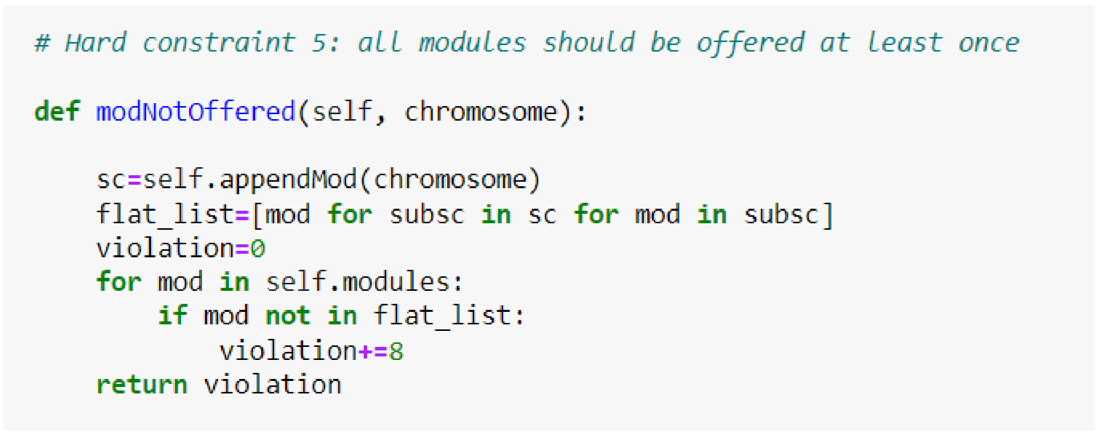
```python
def lessModOffered(self, chromosome):

    sc= self.appendMod(chromosome)
    violation=0

    for s in sc:
        if len(s)>4:
            violation+=10
    return violation
```

4. Every new and existing student must have a module to take in every intake. With the exception that if a student has taken the ‘RMCE’ or ‘RMCP modules, the student is permitted to skip an intake without taking any module.



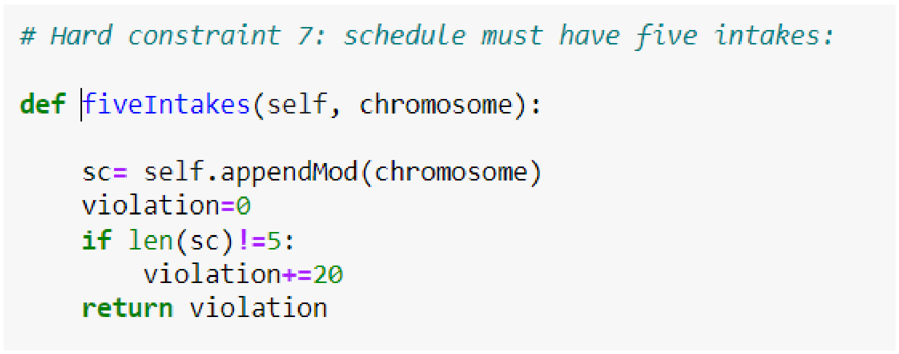
```
# Hard constraint 4: Student must have module to take in every intake
def noModViolation(self, chromosome):

    stuDictBI = self.getStuRecordBI()
    stuDictDE = self.getStuRecordDE()
    stuDictBI.update(stuDictDE)
    sc=self.appendMod(chromosome)

    violation=0
    for stu in stuDictBI.values():
        for s in sc:
            diff=[mod for mod in s if mod not in stu]
            if len(diff)==0 and 'RMCP' not in stu:
                violation+=5
            #if len(diff)==0 and 'RMCP' in stu:
            #    violation-=1
    return violation
```

5. All modules should be offered at least once throughout the five intakes.



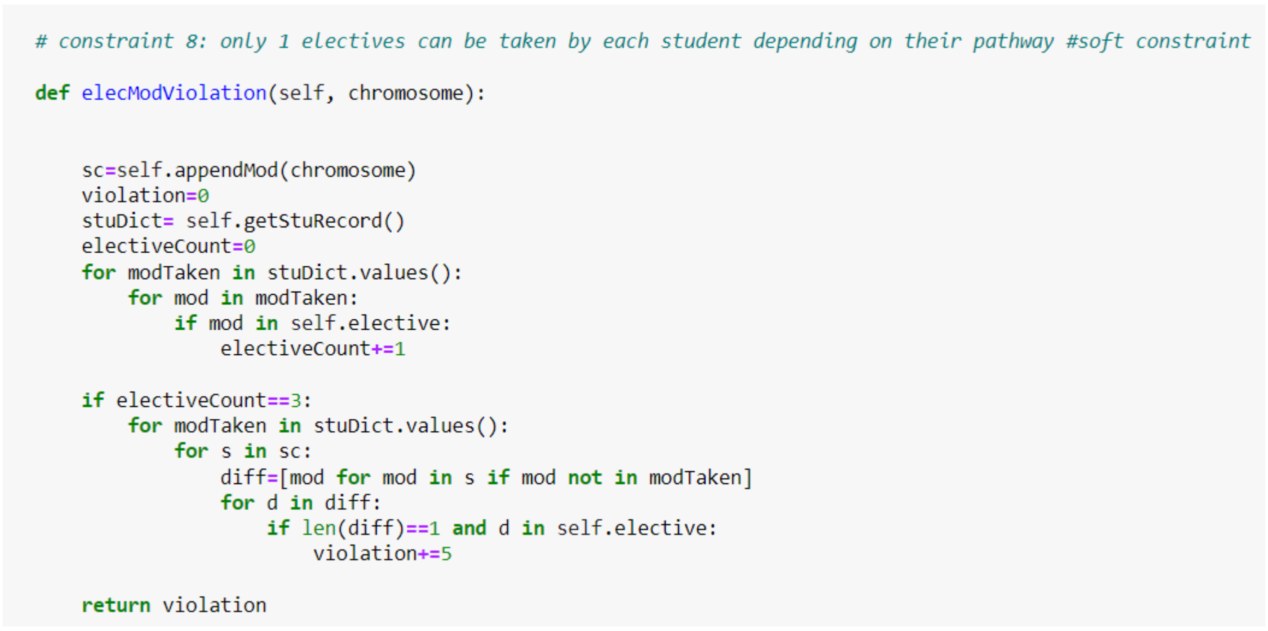
```
# Hard constraint 5: all modules should be offered at least once

def modNotOffered(self, chromosome):

    sc=self.appendMod(chromosome)
    flat_list=[mod for subsc in sc for mod in subsc]
    violation=0
    for mod in self.modules:
        if mod not in flat_list:
            violation+=8
    return violation
```

6. The schedule must have five intakes.



```
# Hard constraint 7: schedule must have five intakes:

def fiveIntakes(self, chromosome):

    sc= self.appendMod(chromosome)
    violation=0
    if len(sc)!=5:
        violation+=20
    return violation
```

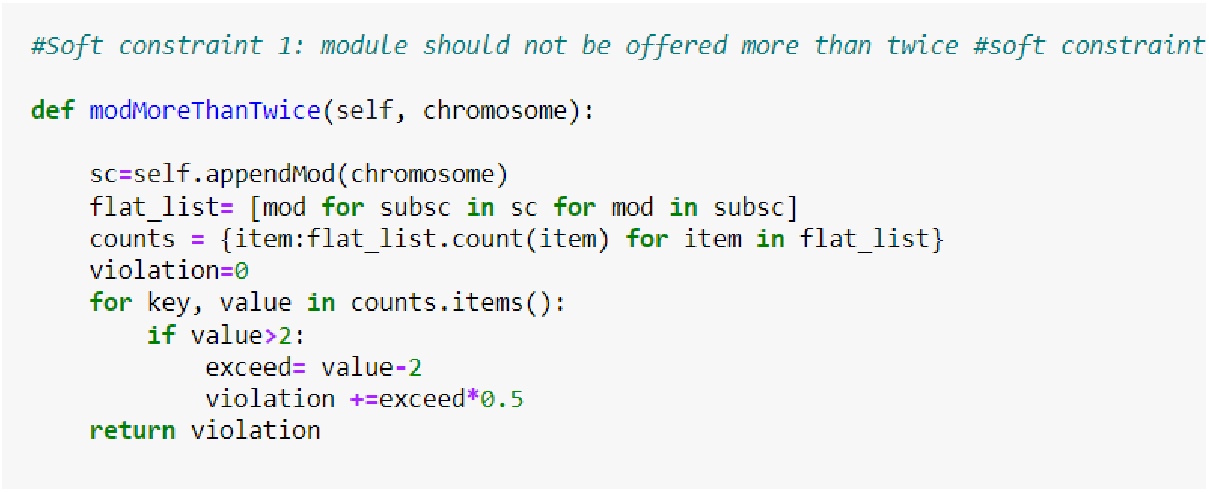
7. Student should not take more elective modules than required for their respective program.



```
# constraint 8: only 1 electives can be taken by each student depending on their pathway #soft constraint

def elecModViolation(self, chromosome):


    sc=self.appendMod(chromosome)
    violation=0
    stuDict= self.getStuRecord()
    electiveCount=0
    for modTaken in stuDict.values():
        for mod in modTaken:
            if mod in self.elective:
                electiveCount+=1

    if electiveCount==3:
        for modTaken in stuDict.values():
            for s in sc:
                diff=[mod for mod in s if mod not in modTaken]
                for d in diff:
                    if len(diff)==1 and d in self.elective:
                        violation+=5

    return violation
```

The soft constraint:

1. Each module should not be offered more than twice throughout the five intakes.



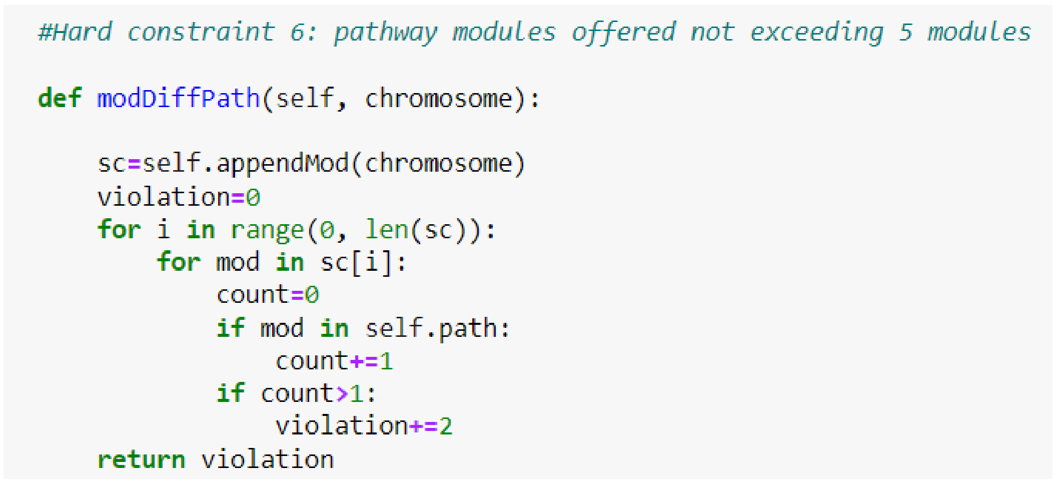
```
#Soft constraint 1: module should not be offered more than twice #soft constraint

def modMoreThanTwice(self, chromosome):

    sc=self.appendMod(chromosome)
    flat_list= [mod for subsc in sc for mod in subsc]
    counts = {item:flat_list.count(item) for item in flat_list}
    violation=0
    for key, value in counts.items():
        if value>2:
            exceed= value-2
            violation +=exceed*0.5
    return violation
```

In order to avoid the generated solution to violate the hard constraints, penalty was applied to each of the violations found.

For DSBA program, student is allowed to choose one elective module whereas students from SE and AI programs are allowed to take three elective modules.

The additional hard constraint for DSBA program is:

1. As this program offers two pathways: Business Intelligence and Data Engineering pathways The student should only take the respective modules for their chosen pathway and only one elective can be taken by each student regardless of the chosen pathway.



```python
#Hard constraint 6: pathway modules offered not exceeding 5 modules

def modDiffPath(self, chromosome):

    sc=self.appendMod(chromosome)
    violation=0
    for i in range(0, len(sc)):
        for mod in sc[i]:
            count=0
            if mod in self.path:
                count+=1
            if count>1:
                violation+=2
    return violation
```

```python
# Hard constraint 8: only 1 electives can be taken by each student depending on their pathway

def elecModViolation(self, chromosome):


    sc=self.appendMod(chromosome)
    violation=0
    stuDictDE= self.getStuRecordDE()
    electiveCount=0
    for modTaken in stuDictDE.values():
        for mod in modTaken:
            if mod in self.electiveDE:
                electiveCount+=1

    if electiveCount==1:
        for modTaken in stuDictDE.values():
            for s in sc:
                diff=[mod for mod in s if mod not in modTaken]
                for d in diff:
                    if len(diff)==1 and d in self.electiveDE:
                        violation+=5

    stuDictBI= self.getStuRecordBI()
    electiveCount=0
    for modTaken in stuDictBI.values():
        for mod in modTaken:
            if mod in self.electiveBI:
                electiveCount+=5

    if electiveCount==1:
        for modTaken in stuDictBI.values():
            for s in sc:
                diff=[mod for mod in s if mod not in modTaken]
                for d in diff:
                    if len(diff)==1 and d in self.electiveBI:
                        violation+=5
    return violation
```