



BERZIET UNIVERSITY
FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENCS3340
Project 1 Report

Course Browser (genetic algorithm)

Prepared by:
Mahmoud Shihab 1182143
Omar Daraghme 1200162

Supervised by:
Dr. Aziz Qaroush

Date: jan-2023

Table of Contents

Table of Figures:	3
Program implementation:	4
Program run:	9

Table of Figures:

Figure 1:example for the txt files.....	4
Figure 2:the table for the mapping for the slots.....	5
Figure 3:generate and check for doctors conlict func	5
Figure 4:fitness func.....	6
Figure 5:selection and crossover func's.....	6
Figure 6:genetic algorithm func.....	7
Figure 7:best chromosomes.....	7
Figure 8:mapping	7
Figure 9:program window.....	9
Figure 10:search for a course.....	9
Figure 11:the course was not found	10

Program implementation:

We used python to implement our project using VScode environment .
for GUI we used a library called tkinter.

Firstly , we made two classes :1-Courses.

2-Labs.

Each class have a string “code” , int “ sec number” , string “doctor name” ,
string “course name”.

We have two txt files contains the data of courses and labs .

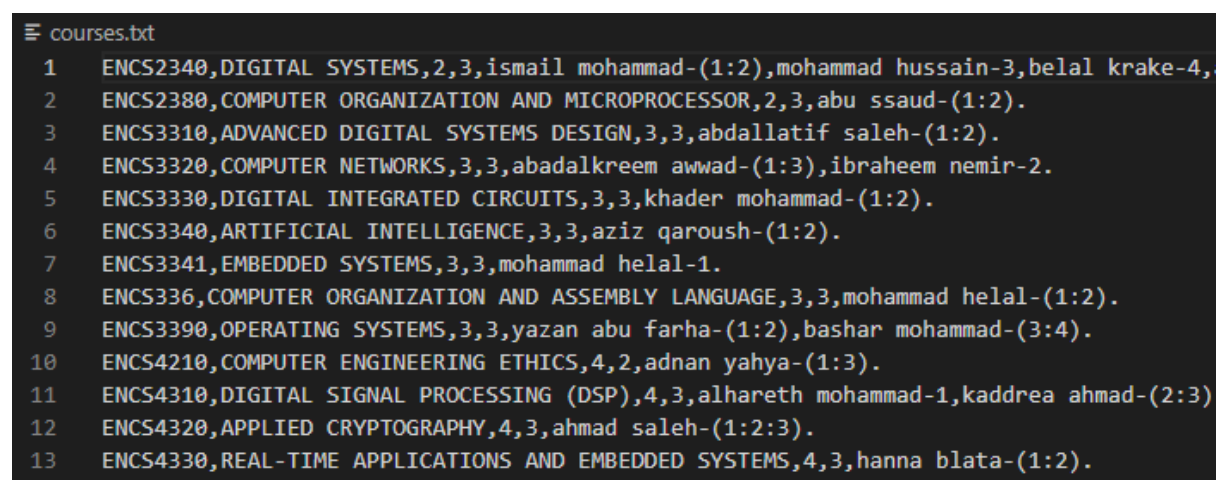
In the courses each line represents a course written as follows.

<course code>,<course name>,<year>,<hours>,n*<doctor name – sections>.

While the labs file each line present a course written as follows.

<labs code>,<labs name>,<hours>,n*<doctor name – sections>.

As shown in figure 1



```

courses.txt
1  ENCS2340,DIGITAL SYSTEMS,2,3,ismaail mohammad-(1:2),mohammad hussain-3,belal krake-4,
2  ENCS2380,COMPUTER ORGANIZATION AND MICROPROCESSOR,2,3,abu ssaoud-(1:2).
3  ENCS3310,ADVANCED DIGITAL SYSTEMS DESIGN,3,3,abdallatif saleh-(1:2).
4  ENCS3320,COMPUTER NETWORKS,3,3,abadalkreem awwad-(1:3),ibraheem nemir-2.
5  ENCS3330,DIGITAL INTEGRATED CIRCUITS,3,3,khader mohammad-(1:2).
6  ENCS3340,ARTIFICIAL INTELLIGENCE,3,3,aziz qaroush-(1:2).
7  ENCS3341,EMBEDDED SYSTEMS,3,3,mohammad helal-1.
8  ENCS336,COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE,3,3,mohammad helal-(1:2).
9  ENCS3390,OPERATING SYSTEMS,3,3,yazan abu farha-(1:2),bashar mohammad-(3:4).
10 ENCS4210,COMPUTER ENGINEERING ETHICS,4,2,adnan yahya-(1:3).
11 ENCS4310,DIGITAL SIGNAL PROCESSING (DSP),4,3,alhareth mohammad-1,kaddrea ahmad-(2:3)
12 ENCS4320,APPLIED CRYPTOGRAPHY,4,3,ahmad saleh-(1:2:3).
13 ENCS4330,REAL-TIME APPLICATIONS AND EMBEDDED SYSTEMS,4,3,hanna blata-(1:2).
  
```

Figure 1:example for the txt files

We have two list of objects: courses, labs. and we read the files that contains the data and make the objects.

The chromosome in the genetic algorithm represent the sections of all courses or labs ,and the data that each gene have is from 0-19 for the courses and 0-14 for the labs that represent the time slots in the weak for example : 0 for courses chromosome is MW 8:30-9:45. There will be mapping in the end to show the slots meaning using those tables as shown in figure 2.

```
mapC=[("MW 08:30-09:45"),("MW 10:00-11:15"),("MW 11:25-12:40"),("MW 12:50-02:05"),("MW 02:15-03:30"),
      ("TR 08:30-09:45"),("TR 10:00-11:15"),("TR 11:25-12:40"),("TR 12:50-02:05"),("TR 02:15-03:30"),
      ("SW 08:30-09:45"),("SW 10:00-11:15"),("SW 11:25-12:40"),("SW 12:50-02:05"),("SW 02:15-03:30"),
      ("SM 08:30-09:45"),("SM 10:00-11:15"),("SM 11:25-12:40"),("SM 12:50-02:05"),("SM 02:15-03:30")]

mapL=[("M 08:00-11:10"),("M 11:25-02:05"),("M 02:15-04:55"),
      ("T 08:00-11:10"),("T 11:25-02:05"),("T 02:15-04:55"),
      ("W 08:00-11:10"),("W 11:25-02:05"),("W 02:15-04:55"),
      ("R 08:00-11:10"),("R 11:25-02:05"),("R 02:15-04:55"),
      ("S 08:00-11:10"),("S 11:25-02:05"),("S 02:15-04:55"),]
```

Figure 2:the table for the mapping for the slots

We generate a number of chromosomes with a random values but for each chromosome we check that there is no conflict for the doctors times (in each time it can't has the same doctor 2 or more times)

```
def generate_chromosome(length, n):
    while(True):
        chromosome = [random.randint(0,length-1) for _ in range(n)] # initialize the list with length
        if(checkForDoctors(chromosome,length)):
            break
    return chromosome

def checkForDoctors(chromosome,length):
    flag=True
    lst=[{} for _ in range(length)]
    for i in range(len(chromosome)):
        if(int(length)==20):
            if LOC[i].doctor not in lst[chromosome[i]]:
                lst[chromosome[i]][LOC[i].doctor]=1
            else:
                lst[chromosome[i]][LOC[i].doctor]+=1
        else:
            if LOL[i].doctor not in lst[chromosome[i]]:
                lst[chromosome[i]][LOL[i].doctor]=1
            else:
                lst[chromosome[i]][LOL[i].doctor]+=1
    for doctors in lst:
        for doctor in doctors:
            if int(doctors[doctor])>1:
                flag = False
                break
    return flag
```

Figure 3:generate and check for doctors conflict func

the functions in the above figure generate a chromosome with random values and then goes to check by making a dictionary of doctors name for each time, for courses 0-19 and for labs 0-14 and for each doctor every time we find we add 1 in the dictionary under his name
if there is a doctor in a specific time have 2 or more then it returns false meaning that the chromosome is not valid yet if the flag returns true then the

chromosome we generated will be returned to the population and it do this process for n time in our code we make it do it 100 times (100 chromosome).

Then for each chromosome in the population we make fitness value that be estimated in the fitness function in the figure below. This function present the heuristic for our code. By calculate the maximum number for sections in every time slots and calculate the minimum then we find the difference between them we need the least value which means more diversity for the sections on the times and that is the goal.

```
def fitness(chromosome,length):
    lst=[0 for _ in range(length)]
    for i in range(len(chromosome)):
        lst[chromosome[i]]+=1
    return 10-(max(lst)-min(lst))
```

Figure 4:fitness func

Then we choose the best two chromosomes based on the fitness value and we make a crossover between these two and we replace the result of the cross over with the chromosome that have the worst fitness value we make this operation repeats for n times in our code we make it 5000 times. in each cross over we check for the doctors conflict and keep making cross over until have no conflicts and return the chromosome . the figure below shows the functions that do the operations that was explained before.

```
def selection(pop, fitnesses):
    # Select a chromosome from the population with probability proportional to its fitness
    idx = random.choices(range(len(pop)), weights=fitnesses, k=1)[0]
    return pop[idx]

def crossover(chromosome1, chromosome2,length):
    # Select a random crossover point and combine the two chromosomes
    while(True):
        idx = random.randint(1, len(chromosome1)-1)
        chromosome=chromosome1[:idx] + chromosome2[idx:]
        if(checkForDoctors(chromosome,length)):
            break
    return chromosome
```

Figure 5:selection and crossover func's

After we finish doing the genetic algorithm we return the chromosome in the population that have the best fitness value which will be the goal. The figure below is the algorithm function.

```
def genetic_algorithm(pop_size, length,n, n_generations):
    # Initialize the population with random chromosomes
    population = [generate_chromosome(length,n) for _ in range(pop_size)]

    # Evaluate the fitness of each chromosome
    fitnesses = [fitness(chrom,length) for chrom in population]

    for _ in range(n_generations):
        # Select the parents for crossover
        parent1 = selection(population, fitnesses)
        parent2 = selection(population, fitnesses)

        # Generate the offspring by crossover and mutation
        offspring = crossover(parent1, parent2,length)

        # Replace the least fit chromosome in the population with the offspring
        idx = fitnesses.index(min(fitnesses))
        population[idx] = offspring
        fitnesses[idx] = fitness(offspring,length)

    # Return the fittest chromosome
    return max(zip(population, fitnesses), key=lambda x: x[1])[0]
```

Figure 6:genetic algorithm func

We do this operation for the courses and for the labs to get the best chromosome for each as shown in the figure below

```
resultC=genetic_algorithm(100,20, len(LOC),5000)
resultL=genetic_algorithm(100,15, len(LOL),5000)
```

Figure 7:best chromosomes

The final chromosome we get, each location 0 - #of sections will be mapped to its object and save the data in dictionary based on the course name.as shown in the figure below.

```
lst={}
for i in range(len(resultC)):
    if LOC[i].code not in lst:
        lst[LOC[i].code]=[f"%1s  %s  %10s"%(LOC[i].sec, mapC[resultC[i]] ,LOC[i].doctor)]
    else :
        lst[LOC[i].code].append(f"%1s  %s  %10s"%(LOC[i].sec,mapC[resultC[i]] ,LOC[i].doctor) )
for i in range(len(resultL)):
    if LOL[i].code not in lst:
        lst[LOL[i].code]=[f"%1s  %s  %10s"%(LOL[i].sec,mapL[resultL[i]] ,LOL[i].doctor) ]
    else :
        lst[LOL[i].code].append(f"%1s  %s  %10s"%(LOL[i].sec,mapL[resultL[i]] ,LOL[i].doctor) )
```

Figure 8:mapping

For the GUI we make a window with the size of the screen, that contains a search bar to search for a specific course for the user to make. and under the search bar will be all the courses and the labs with their sections and times.

Program run:

When we run the program this window as in the figure below opens.

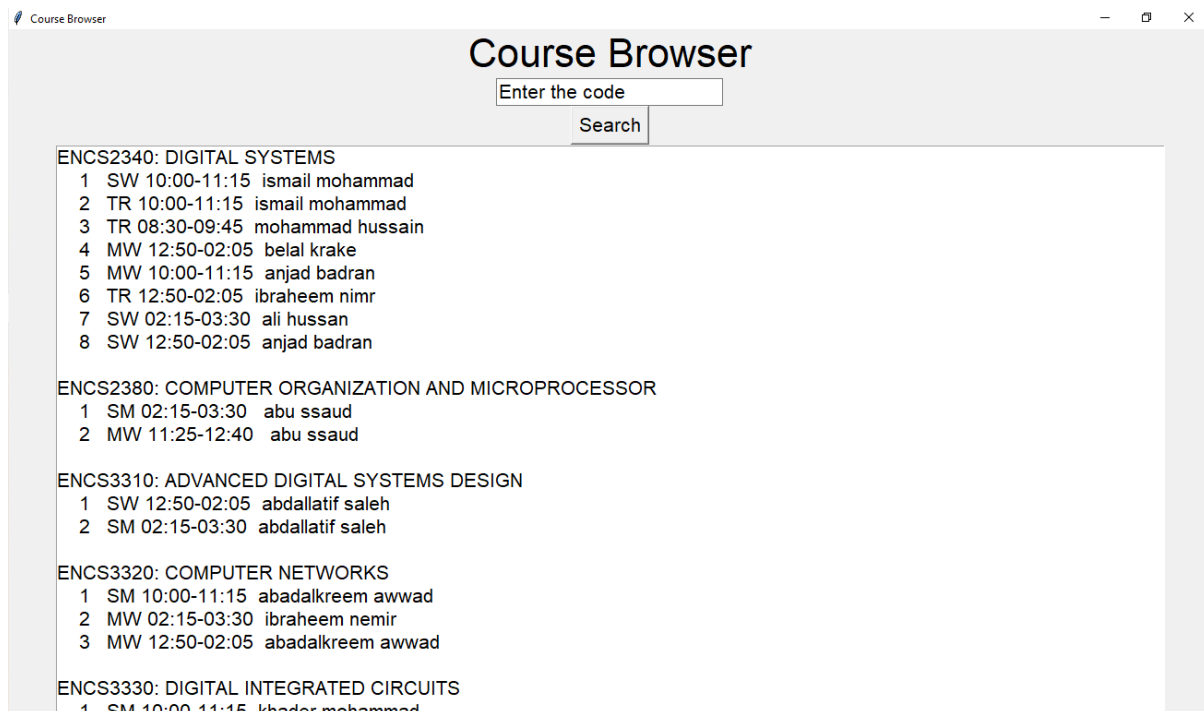


Figure 9:program window

If the user want to search for a specific course, the user will use the search bar and then press on the search button. a window will pop up that have the sections for the searched course. If the course is not there there will be a message that the course was not found.as shown in the figures below.

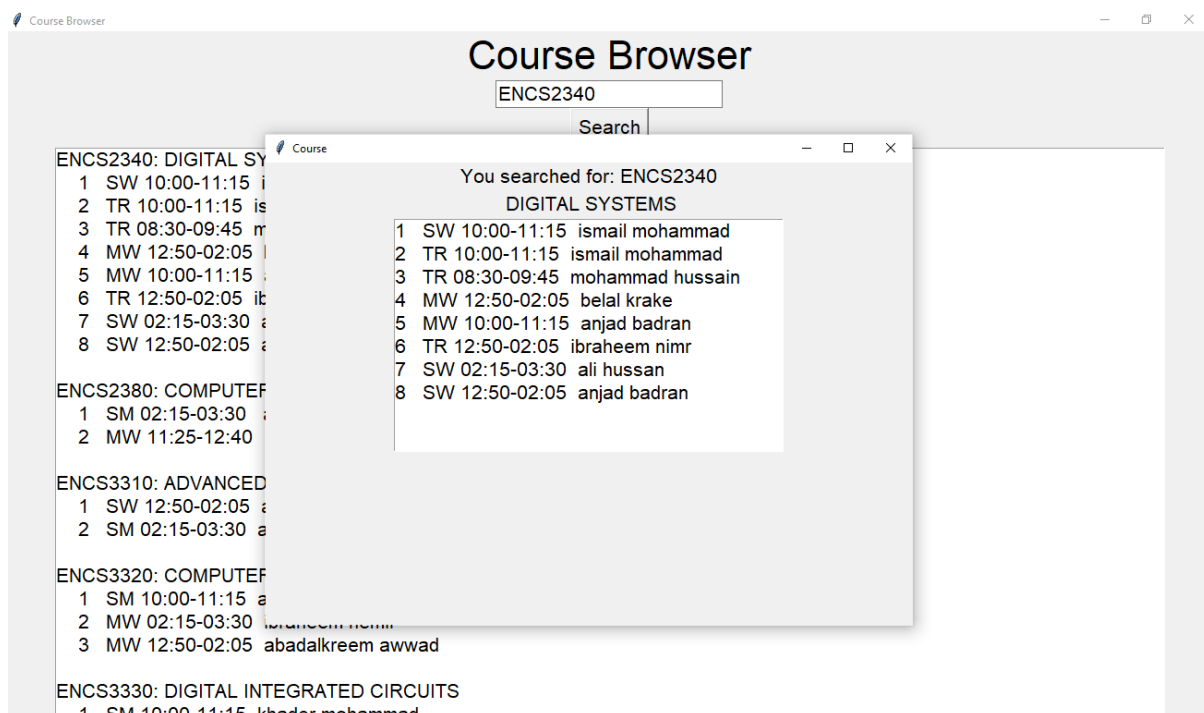


Figure 10:search for a course

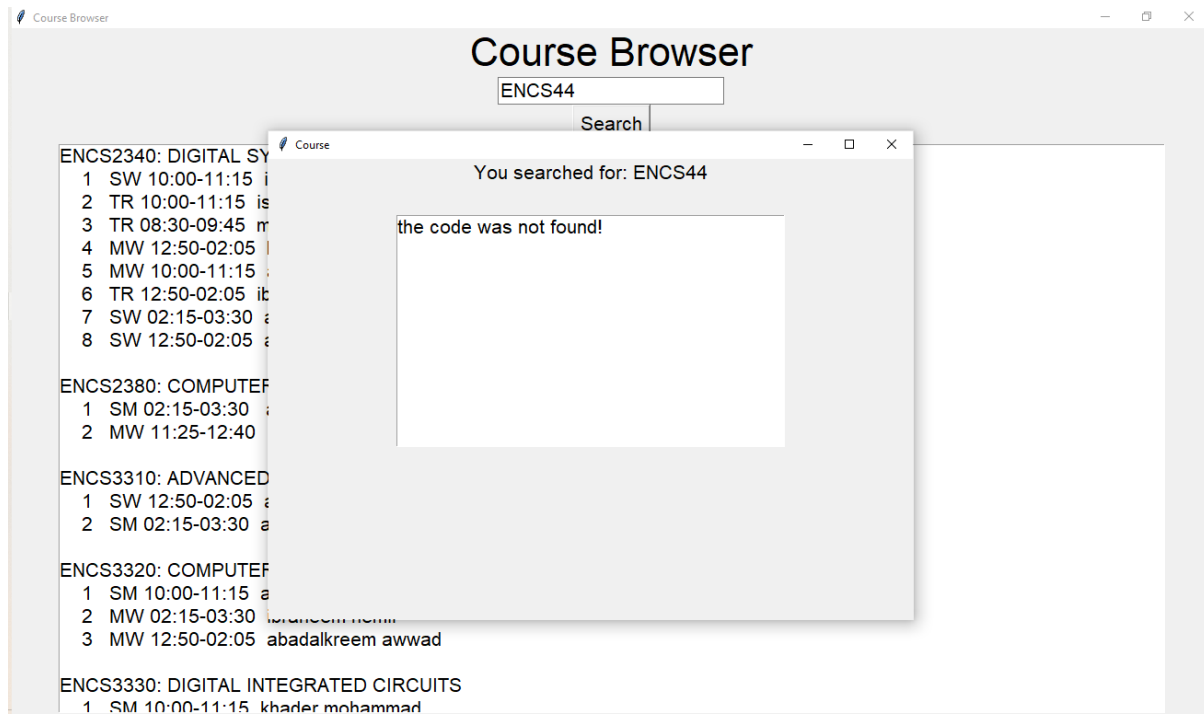


Figure 11: the course was not found