

Artificial intelligence

(Code description file)



*SIR:  
Haseeb Arshad*

Member name

19f-0240 -Abdullah Awan

“Exam Schedule Generator Using Local

Search Algorithms”

The project was scheduling of exams. Here we implemented simulated annealing to find out the best solution on the basis of probability which had a low cost and less conflicts in the timetable. Moreover, we used GUI graphical user interface of tkinter for better and unique output and interface.

Here’s the code

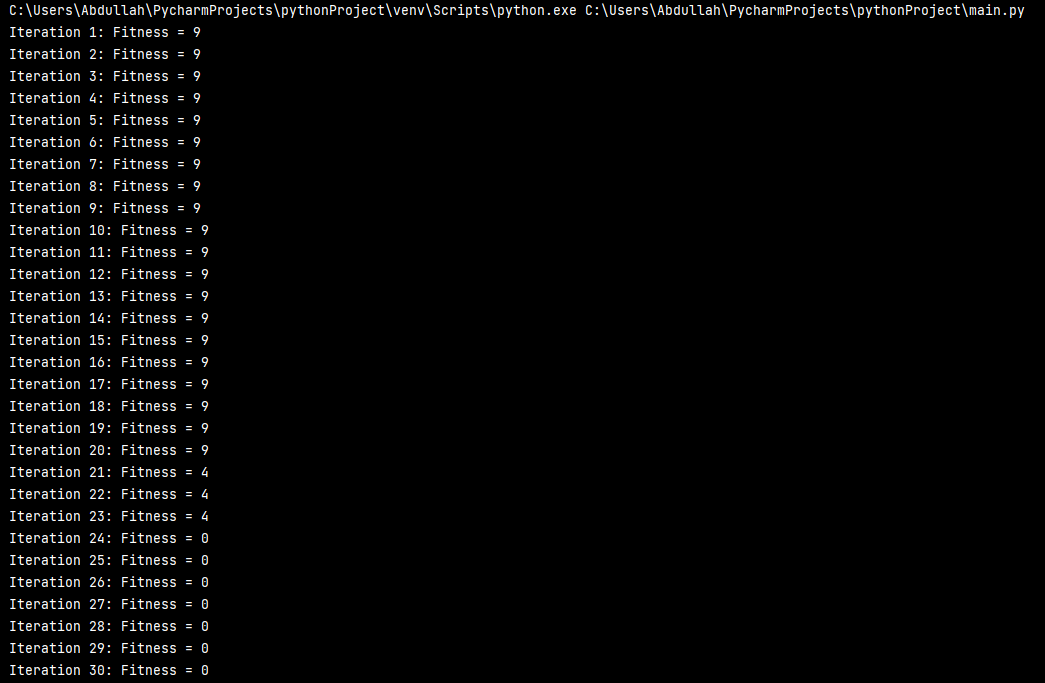
code:

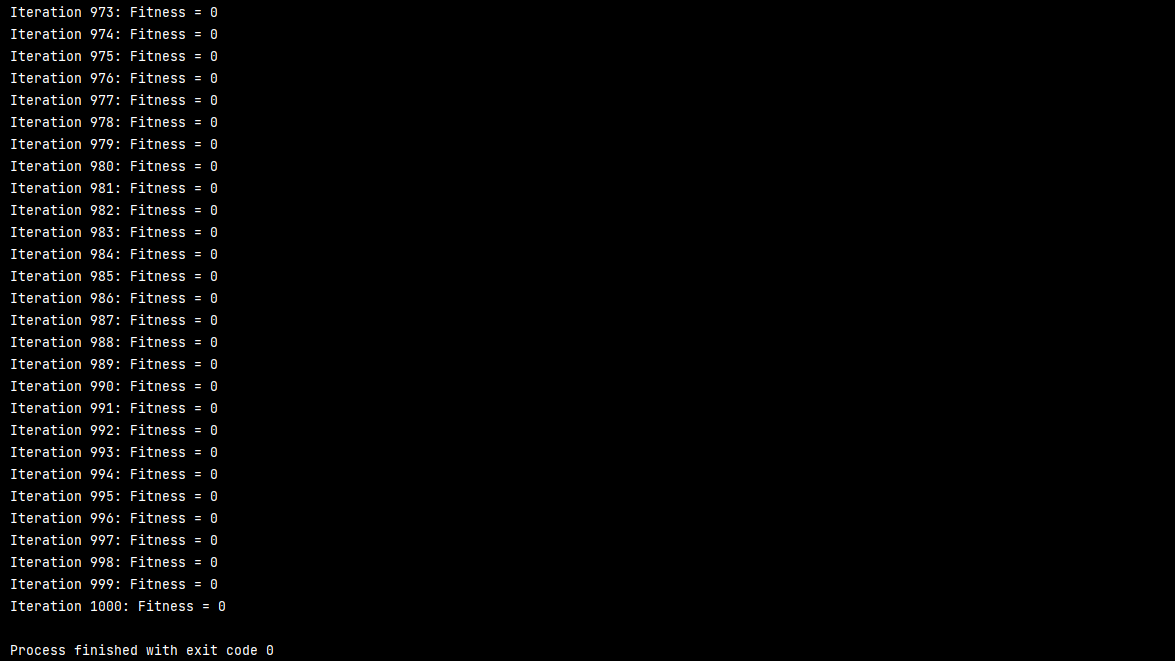
import numpy as np  
import pandas as pd  
import copy  
import math  
import random  
import tkinter as tk  
from tkinter import ttk  
from tkinter import messagebox  
  
def import\_data(fileDir):  
 teachers\_df = pd.read\_csv(fileDir + 'teachers.csv')  
 students\_df = pd.read\_csv(fileDir + 'studentNames.csv')  
 rooms\_df = pd.read\_csv(fileDir + 'rooms.csv')  
 courses\_df = pd.read\_csv(fileDir + 'courses.csv')  
 registered\_courses\_df = pd.read\_csv(fileDir + 'studentCourse.csv')  
  
 teachers = list(teachers\_df.iloc[:**, 0**])  
 students = list(students\_df.iloc[:**, 0**])  
 rooms = {i + **1**: capacity for i**,** capacity in enumerate(rooms\_df.iloc[:**, 1**])}  
 courses = {course\_code: course\_name for course\_code**,** course\_name in  
 zip(courses\_df.iloc[:**, 0**]**,** courses\_df.iloc[:**, 1**])}  
 student\_courses = {(student\_name**,** course\_code) for student\_name**,** course\_code in  
 zip(registered\_courses\_df.iloc[:**, 1**]**,** registered\_courses\_df.iloc[:**, 2**])}  
  
 return teachers**,** students**,** rooms**,** courses**,** student\_courses  
  
def random\_solution(courses**,** students**,** rooms):  
 exam\_schedule = {}  
 for course\_code**,** course\_name in courses.items():  
 exam\_schedule[course\_code] = {  
 'course': course\_name**,** 'room': random.sample(list(rooms.keys())**, 1**)**,** 'teacher': random.sample(teachers**, 1**)**,** 'time': random.choice([**9, 14**])**,** # 9 AM or 2 PM  
 'date': random.randint(**1,** num\_days) # Random date between 1 and num\_days  
 }  
 return exam\_schedule  
  
def calculate\_fitness(exam\_schedule**,** student\_courses**,** num\_days):  
  
 num\_violations = **0** for day in range(**1,** num\_days + **1**):  
 for time in [**9, 14**]:  
 exams\_at\_time = [course\_code for course\_code**,** exam in exam\_schedule.items() if exam['date'] == day and exam['time'] == time]  
 for student in students:  
 num\_exams = sum(**1** for course\_code in exams\_at\_time if (student**,** course\_code) in student\_courses)  
 if num\_exams > **1**:  
 num\_violations += **1** for day in range(**1,** num\_days + **1**):  
 for time in [**9, 14**]:  
 exams\_at\_time = [exam for exam in exam\_schedule.values() if exam['date'] == day and exam['time'] == time]  
 teachers\_at\_time = [exam['teacher'][**0**] for exam in exams\_at\_time]  
 if len(set(teachers\_at\_time)) != len(teachers\_at\_time):  
 num\_violations += **1** return num\_violations  
  
def generate\_neighbor\_solution(current\_solution**,** rooms):  
 new\_solution = copy.deepcopy(current\_solution)  
  
 exam\_code**,** exam\_info = random.choice(list(new\_solution.items()))  
  
  
 new\_date = random.randint(**1,** num\_days)  
  
  
 new\_solution[exam\_code]['date'] = new\_date  
 new\_solution[exam\_code]['room'] = random.sample(list(rooms.keys())**, 1**)  
  
 return new\_solution  
  
def simulated\_annealing(initial\_solution**,** student\_courses**,** num\_days**,** max\_iterations**,** initial\_temperature**,** cooling\_rate):  
 current\_solution = initial\_solution  
 current\_fitness = calculate\_fitness(current\_solution**,** student\_courses**,** num\_days)  
 best\_solution = current\_solution.copy()  
 best\_fitness = current\_fitness  
  
 temperature = initial\_temperature  
  
 for i in range(max\_iterations):  
 new\_solution = generate\_neighbor\_solution(current\_solution**,** rooms)  
 new\_fitness = calculate\_fitness(new\_solution**,** student\_courses**,** num\_days)  
  
 if new\_fitness < current\_fitness or random.uniform(**0, 1**) < math.exp((current\_fitness - new\_fitness) / temperature):  
 current\_solution = new\_solution  
 current\_fitness = new\_fitness  
  
 if new\_fitness < best\_fitness:  
 best\_solution = new\_solution  
 best\_fitness = new\_fitness  
  
 temperature \*= cooling\_rate  
  
 print(f"Iteration {i + **1**}: Fitness = {best\_fitness}")  
  
 return best\_solution  
  
def check\_break\_constraint(exam\_schedule**,** num\_days):  
 num\_violations = **0** for day in range(**1,** num\_days + **1**):  
 exams\_at\_time = [exam for exam in exam\_schedule.values() if exam['date'] == day and exam['time'] == **13**] # 1 PM  
 if exams\_at\_time:  
 num\_violations += **1** return num\_violations  
  
def check\_consecutive\_exams\_constraint(exam\_schedule**,** student\_courses**,** num\_days):  
 num\_violations = **0** for day in range(**1,** num\_days + **1**):  
 exams\_at\_day = [exam for exam in exam\_schedule.values() if exam['date'] == day]  
 exams\_at\_day.sort(key=lambda x: x['time']) # Sort exams by time on the day  
 for i in range(len(exams\_at\_day) - **1**):  
 current\_exam = exams\_at\_day[i]  
 next\_exam = exams\_at\_day[i + **1**]  
 if current\_exam['time'] == **9** and next\_exam['time'] == **14**: # Consecutive exams  
 for student in students:  
 if (student**,** current\_exam['course']) in student\_courses and (student**,** next\_exam['course']) in student\_courses:  
 num\_violations += **1** return num\_violations  
  
def check\_preferred\_order\_constraint(exam\_schedule**,** student\_courses):  
 num\_violations = **0** for student in students:  
 has\_mg\_course = any(course\_code.startswith('MG') for (\_**,** course\_code) in student\_courses if student == student)  
 has\_cs\_course = any(course\_code.startswith('CS') for (\_**,** course\_code) in student\_courses if student == student)  
 if has\_mg\_course and has\_cs\_course:  
 mg\_course\_exam = next((exam for exam in exam\_schedule.values() if exam['course'].startswith('MG') and (student**,** exam['course']) in student\_courses)**,** None)  
 cs\_course\_exam = next((exam for exam in exam\_schedule.values() if exam['course'].startswith('CS') and (student**,** exam['course']) in student\_courses)**,** None)  
 if mg\_course\_exam and cs\_course\_exam:  
 if mg\_course\_exam['date'] > cs\_course\_exam['date']:  
 num\_violations += **1** return num\_violations  
  
  
def check\_faculty\_meeting\_constraint(exam\_schedule**,** num\_days):  
 num\_violations = **0** half\_days = math.ceil(num\_days / **2**)  
 for day in range(**1,** half\_days + **1**):  
 exams\_at\_day = [exam for exam in exam\_schedule.values() if exam['date'] == day]  
 if len(exams\_at\_day) == **0**:  
 num\_violations += **1** return num\_violations  
  
def calculate\_soft\_constraints(exam\_schedule**,** student\_courses**,** num\_days):  
  
 soft\_constraints = {  
 'Break on Friday': check\_break\_constraint(exam\_schedule**,** num\_days)**,** 'Consecutive Exams': check\_consecutive\_exams\_constraint(exam\_schedule**,** student\_courses**,** num\_days)**,** 'Preferred Order of MG and CS Courses': check\_preferred\_order\_constraint(exam\_schedule**,** student\_courses)**,** 'Faculty Meeting Constraints': check\_faculty\_meeting\_constraint(exam\_schedule**,** num\_days)  
 }  
 return soft\_constraints  
  
def display\_schedule\_gui(exam\_schedule\_df**,** soft\_constraints\_df):  
 root = tk.Tk()  
 root.title("Exam Schedule and Soft Constraints")  
  
  
 exam\_schedule\_frame = ttk.Frame(root)  
 exam\_schedule\_frame.pack(pady=**10**)  
  
 exam\_schedule\_label = tk.Label(exam\_schedule\_frame**,** text="Exam Schedule for Two Weeks:"**,** font=("Arial"**, 14,** "bold"))  
 exam\_schedule\_label.grid(row=**0,** column=**0,** columnspan=**5,** pady=**5**)  
  
  
 exam\_schedule\_tree = ttk.Treeview(exam\_schedule\_frame**,** columns=('Course'**,** 'Room'**,** 'Day'**,** 'Time'**,** 'Teacher')**,** show='headings'**,** height=**15**)  
 exam\_schedule\_tree.grid(row=**1,** column=**0,** columnspan=**5,** padx=**5**)  
  
 exam\_schedule\_tree.heading('Course'**,** text='Course')  
 exam\_schedule\_tree.heading('Room'**,** text='Room')  
 exam\_schedule\_tree.heading('Day'**,** text='Day')  
 exam\_schedule\_tree.heading('Time'**,** text='Time')  
 exam\_schedule\_tree.heading('Teacher'**,** text='Teacher')  
  
 # Add data to the Treeview widget  
 for i**,** row in exam\_schedule\_df.iterrows():  
 exam\_schedule\_tree.insert(''**,** tk.END**,** values=(row['Course']**,** row['Room']**,** row['Day']**,** row['Time']**,** row['Teacher']))  
  
 # Create a Frame to contain the soft constraint violations data  
 soft\_constraints\_frame = ttk.Frame(root)  
 soft\_constraints\_frame.pack(pady=**10**)  
  
 # Create a Label for the soft constraint violations section  
 soft\_constraints\_label = tk.Label(soft\_constraints\_frame**,** text="Soft Constraint Violations:"**,** font=("Arial"**, 14,** "bold"))  
 soft\_constraints\_label.grid(row=**0,** column=**0,** columnspan=**5,** pady=**5**)  
  
 # Create a Treeview widget to display the soft constraint violations data in a table format  
 soft\_constraints\_tree = ttk.Treeview(soft\_constraints\_frame**,** columns=('Soft Constraint'**,** 'Violations')**,** show='headings'**,** height=**10**)  
 soft\_constraints\_tree.grid(row=**1,** column=**0,** columnspan=**5,** padx=**5**)  
  
 # Add columns to the Treeview widget  
 soft\_constraints\_tree.heading('Soft Constraint'**,** text='Soft Constraint')  
 soft\_constraints\_tree.heading('Violations'**,** text='Violations')  
  
 # Add data to the Treeview widget  
 for i**,** row in soft\_constraints\_df.iterrows():  
 soft\_constraints\_tree.insert(''**,** tk.END**,** values=(row['Soft Constraint']**,** row['Violations']))  
  
 # Run the GUI event loop  
 root.mainloop()  
  
# Constants  
fileDir = './test\_dataset/'  
num\_days = **14**max\_iterations = **1000**initial\_temperature = **1000.0**cooling\_rate = **0.95**# Load data  
teachers**,** students**,** rooms**,** courses**,** student\_courses = import\_data(fileDir)  
  
# Generate initial solution  
initial\_solution = random\_solution(courses**,** students**,** rooms)  
  
# Run simulated annealing algorithm to generate exam schedule for two weeks  
exam\_schedule\_2\_weeks = simulated\_annealing(initial\_solution**,** student\_courses**,** num\_days**,** max\_iterations**,** initial\_temperature**,** cooling\_rate)  
  
# Calculate soft constraints  
soft\_constraints = calculate\_soft\_constraints(exam\_schedule\_2\_weeks**,** student\_courses**,** num\_days)  
  
# Create a list of dictionaries to store the exam schedule data  
exam\_schedule\_data = []  
for course\_code**,** exam\_info in exam\_schedule\_2\_weeks.items():  
 day = exam\_info['date']  
 time = "9 AM" if exam\_info['time'] == **9** else "2 PM"  
 teacher = exam\_info['teacher'][**0**]  
 exam\_schedule\_data.append({  
 'Course': exam\_info['course']**,** 'Room': f"Room {exam\_info['room'][**0**]}"**,** 'Day': f"Day {day}"**,** 'Time': time**,** 'Teacher': teacher  
 })  
  
# Convert the list of dictionaries to a DataFrame  
exam\_schedule\_df = pd.DataFrame(exam\_schedule\_data)  
  
# Save exam schedule and soft constraint violations to CSV files  
exam\_schedule\_df.to\_csv('exam\_schedule.csv'**,** index=False)  
soft\_constraints\_df = pd.DataFrame(soft\_constraints.items()**,** columns=['Soft Constraint'**,** 'Violations'])  
soft\_constraints\_df.to\_csv('soft\_constraints.csv'**,** index=False)  
  
# Display the exam schedule and soft constraint violations in a GUI  
display\_schedule\_gui(exam\_schedule\_df**,** soft\_constraints\_df)

output

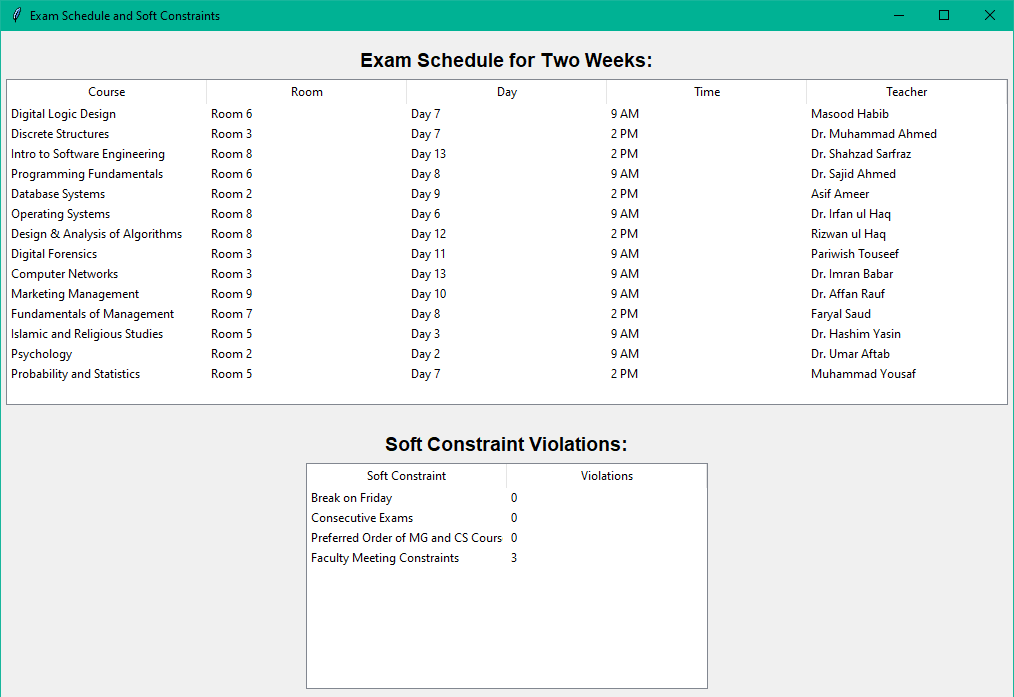
code generates iterations and fitness and two CSV files

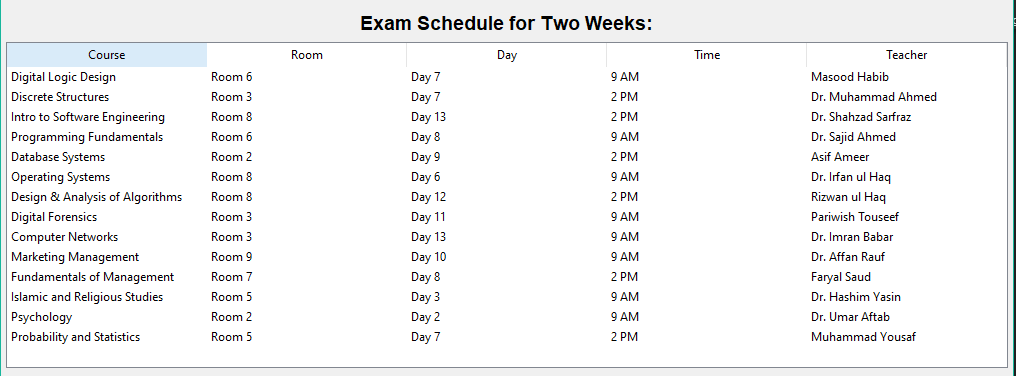
1. Exam\_schedule.csv
2. Soft\_constraints.csv
3. And also A GUI of timetable using tkinter

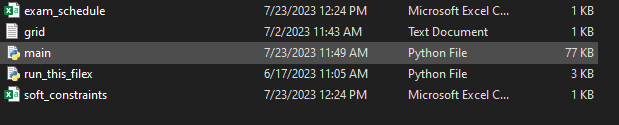




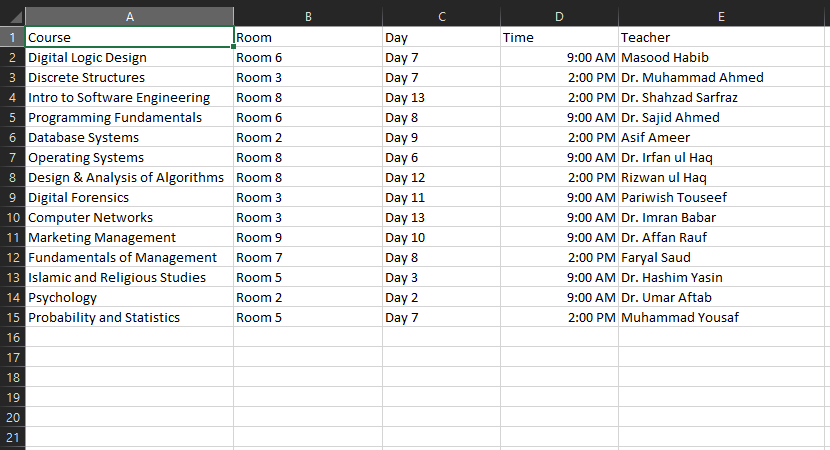
GUI output

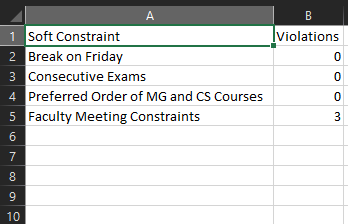




These two files will form  


Files Output





After performing some test on the code we updated the code which not only do simple two week also do three weeks exam schedule and also make two csv files and two soft constraints file with output of two GUI:

Here is updated code:

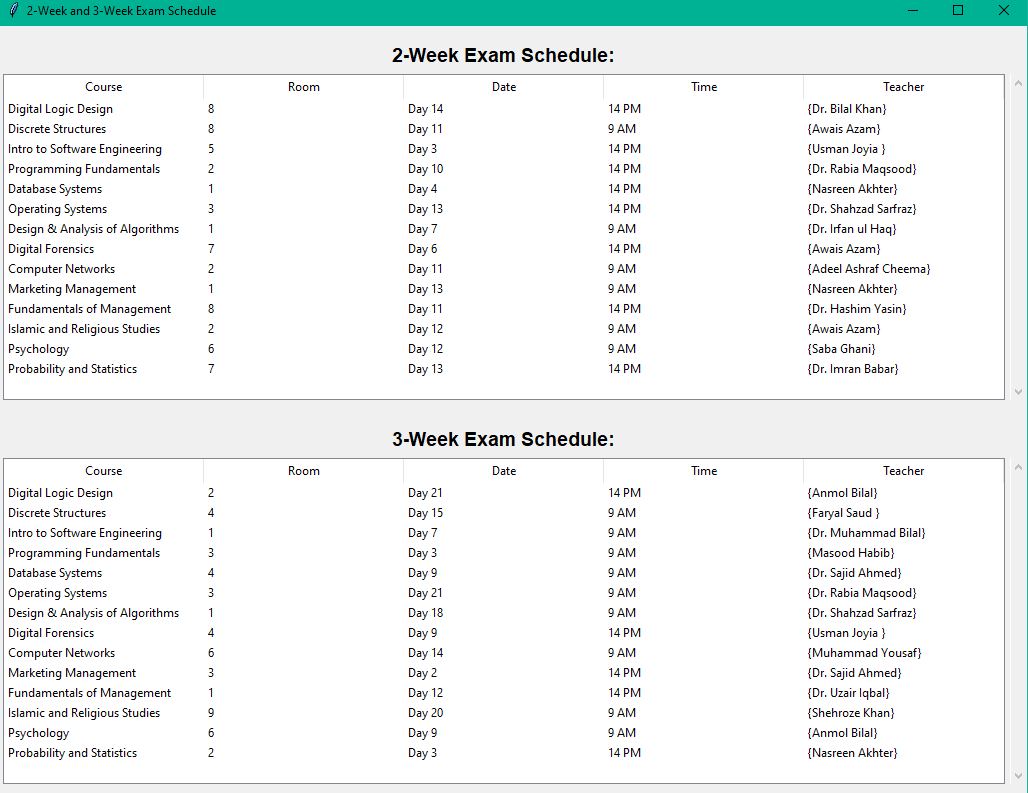
import numpy as np  
import pandas as pd  
import copy  
import math  
import random  
import tkinter as tk  
from tkinter import ttk  
from tkinter import messagebox  
  
def import\_data(fileDir):  
 teachers\_df = pd.read\_csv(fileDir + 'teachers.csv')  
 students\_df = pd.read\_csv(fileDir + 'studentNames.csv')  
 rooms\_df = pd.read\_csv(fileDir + 'rooms.csv')  
 courses\_df = pd.read\_csv(fileDir + 'courses.csv')  
 registered\_courses\_df = pd.read\_csv(fileDir + 'studentCourse.csv')  
  
 teachers = list(teachers\_df.iloc[:**, 0**])  
 students = list(students\_df.iloc[:**, 0**])  
 rooms = {i + **1**: capacity for i**,** capacity in enumerate(rooms\_df.iloc[:**, 1**])}  
 courses = {course\_code: course\_name for course\_code**,** course\_name in  
 zip(courses\_df.iloc[:**, 0**]**,** courses\_df.iloc[:**, 1**])}  
 student\_courses = {(student\_name**,** course\_code) for student\_name**,** course\_code in  
 zip(registered\_courses\_df.iloc[:**, 1**]**,** registered\_courses\_df.iloc[:**, 2**])}  
  
 return teachers**,** students**,** rooms**,** courses**,** student\_courses  
  
def random\_solution(courses**,** students**,** rooms**,** num\_days):  
 exam\_schedule = {}  
 for course\_code**,** course\_name in courses.items():  
 exam\_schedule[course\_code] = {  
 'course': course\_name**,** 'room': random.sample(list(rooms.keys())**, 1**)**,** 'teacher': random.sample(teachers**, 1**)**,** 'time': random.choice([**9, 14**])**,** # 9 AM or 2 PM  
 'date': random.randint(**1,** num\_days) # Random date between 1 and num\_days  
 }  
 return exam\_schedule  
  
def calculate\_fitness(exam\_schedule**,** student\_courses**,** num\_days):  
 num\_violations = **0** for day in range(**1,** num\_days + **1**):  
 for time in [**9, 14**]:  
 exams\_at\_time = [course\_code for course\_code**,** exam in exam\_schedule.items() if exam['date'] == day and exam['time'] == time]  
 for student in students:  
 num\_exams = sum(**1** for course\_code in exams\_at\_time if (student**,** course\_code) in student\_courses)  
 if num\_exams > **1**:  
 num\_violations += **1** for day in range(**1,** num\_days + **1**):  
 for time in [**9, 14**]:  
 exams\_at\_time = [exam for exam in exam\_schedule.values() if exam['date'] == day and exam['time'] == time]  
 teachers\_at\_time = [exam['teacher'][**0**] for exam in exams\_at\_time]  
 if len(set(teachers\_at\_time)) != len(teachers\_at\_time):  
 num\_violations += **1** return num\_violations  
  
def generate\_neighbor\_solution(current\_solution**,** rooms**,** num\_days):  
 new\_solution = copy.deepcopy(current\_solution)  
  
 exam\_code**,** exam\_info = random.choice(list(new\_solution.items()))  
  
 new\_date = random.randint(**1,** num\_days)  
  
 new\_solution[exam\_code]['date'] = new\_date  
 new\_solution[exam\_code]['room'] = random.sample(list(rooms.keys())**, 1**)  
  
 return new\_solution  
  
def simulated\_annealing(initial\_solution**,** student\_courses**,** num\_days**,** max\_iterations**,** initial\_temperature**,** cooling\_rate):  
 current\_solution = initial\_solution  
 current\_fitness = calculate\_fitness(current\_solution**,** student\_courses**,** num\_days)  
 best\_solution = current\_solution.copy()  
 best\_fitness = current\_fitness  
  
 temperature = initial\_temperature  
  
 for i in range(max\_iterations):  
 new\_solution = generate\_neighbor\_solution(current\_solution**,** rooms**,** num\_days)  
 new\_fitness = calculate\_fitness(new\_solution**,** student\_courses**,** num\_days)  
  
 if new\_fitness < current\_fitness or random.uniform(**0, 1**) < math.exp((current\_fitness - new\_fitness) / temperature):  
 current\_solution = new\_solution  
 current\_fitness = new\_fitness  
  
 if new\_fitness < best\_fitness:  
 best\_solution = new\_solution  
 best\_fitness = new\_fitness  
  
 temperature \*= cooling\_rate  
  
 print(f"Iteration {i + **1**}: Fitness = {best\_fitness}")  
  
 return best\_solution  
  
def check\_break\_constraint(exam\_schedule**,** num\_days):  
 num\_violations = **0** for day in range(**1,** num\_days + **1**):  
 exams\_at\_time = [exam for exam in exam\_schedule.values() if exam['date'] == day and exam['time'] == **13**] # 1 PM  
 if exams\_at\_time:  
 num\_violations += **1** return num\_violations  
  
def check\_consecutive\_exams\_constraint(exam\_schedule**,** student\_courses**,** num\_days):  
 num\_violations = **0** for day in range(**1,** num\_days + **1**):  
 exams\_at\_day = [exam for exam in exam\_schedule.values() if exam['date'] == day]  
 exams\_at\_day.sort(key=lambda x: x['time']) # Sort exams by time on the day  
 for i in range(len(exams\_at\_day) - **1**):  
 current\_exam = exams\_at\_day[i]  
 next\_exam = exams\_at\_day[i + **1**]  
 if current\_exam['time'] == **9** and next\_exam['time'] == **14**: # Consecutive exams  
 for student in students:  
 if (student**,** current\_exam['course']) in student\_courses and (student**,** next\_exam['course']) in student\_courses:  
 num\_violations += **1** return num\_violations  
  
def check\_preferred\_order\_constraint(exam\_schedule**,** student\_courses):  
 num\_violations = **0** for student in students:  
 has\_mg\_course = any(course\_code.startswith('MG') for (\_**,** course\_code) in student\_courses if student == student)  
 has\_cs\_course = any(course\_code.startswith('CS') for (\_**,** course\_code) in student\_courses if student == student)  
 if has\_mg\_course and has\_cs\_course:  
 mg\_course\_exam = next((exam for exam in exam\_schedule.values() if exam['course'].startswith('MG') and (student**,** exam['course']) in student\_courses)**,** None)  
 cs\_course\_exam = next((exam for exam in exam\_schedule.values() if exam['course'].startswith('CS') and (student**,** exam['course']) in student\_courses)**,** None)  
 if mg\_course\_exam and cs\_course\_exam:  
 if mg\_course\_exam['date'] > cs\_course\_exam['date']:  
 num\_violations += **1** return num\_violations  
  
  
def check\_faculty\_meeting\_constraint(exam\_schedule**,** num\_days):  
 num\_violations = **0** half\_days = math.ceil(num\_days / **2**)  
 for day in range(**1,** half\_days + **1**):  
 exams\_at\_day = [exam for exam in exam\_schedule.values() if exam['date'] == day]  
 if len(exams\_at\_day) == **0**:  
 num\_violations += **1** return num\_violations  
  
def calculate\_soft\_constraints(exam\_schedule**,** student\_courses**,** num\_days):  
  
 soft\_constraints = {  
 'Break on Friday': check\_break\_constraint(exam\_schedule**,** num\_days)**,** 'Consecutive Exams': check\_consecutive\_exams\_constraint(exam\_schedule**,** student\_courses**,** num\_days)**,** 'Preferred Order of MG and CS Courses': check\_preferred\_order\_constraint(exam\_schedule**,** student\_courses)**,** 'Faculty Meeting Constraints': check\_faculty\_meeting\_constraint(exam\_schedule**,** num\_days)  
 }  
 return soft\_constraints  
  
import tkinter as tk  
from tkinter import ttk  
  
def display\_schedule\_gui(exam\_schedule\_df\_two\_weeks**,** exam\_schedule\_df\_three\_weeks**,** soft\_constraints\_df\_two\_weeks**,** soft\_constraints\_df\_three\_weeks):  
 # Create the main window for exam schedule  
 exam\_schedule\_window = tk.Tk()  
 exam\_schedule\_window.title("2-Week and 3-Week Exam Schedule")  
  
 # Create a Frame for 2-week exam schedule  
 exam\_schedule\_frame\_two\_weeks = ttk.Frame(exam\_schedule\_window)  
 exam\_schedule\_frame\_two\_weeks.pack(pady=**10**)  
  
 exam\_schedule\_label\_two\_weeks = tk.Label(exam\_schedule\_frame\_two\_weeks**,** text="2-Week Exam Schedule:"**,** font=("Arial"**, 14,** "bold"))  
 exam\_schedule\_label\_two\_weeks.grid(row=**0,** column=**0,** columnspan=**5,** pady=**5**)  
  
 exam\_schedule\_tree\_two\_weeks = ttk.Treeview(exam\_schedule\_frame\_two\_weeks**,** columns=('Course'**,** 'Room'**,** 'Date'**,** 'Time'**,** 'Teacher')**,** show='headings'**,** height=**15**)  
 exam\_schedule\_tree\_two\_weeks.grid(row=**1,** column=**0,** columnspan=**5,** padx=**5**)  
  
 exam\_schedule\_tree\_two\_weeks.heading('Course'**,** text='Course')  
 exam\_schedule\_tree\_two\_weeks.heading('Room'**,** text='Room')  
 exam\_schedule\_tree\_two\_weeks.heading('Date'**,** text='Date')  
 exam\_schedule\_tree\_two\_weeks.heading('Time'**,** text='Time')  
 exam\_schedule\_tree\_two\_weeks.heading('Teacher'**,** text='Teacher')  
  
 # Add data to the Treeview widget for 2-week exam schedule  
 for i**,** row in exam\_schedule\_df\_two\_weeks.iterrows():  
 exam\_schedule\_tree\_two\_weeks.insert(''**,** tk.END**,** values=(row['course']**,** row['room']**,** f"Day {row['date']}"**,** f"{row['time']} {['AM'**,** 'PM'][row['time'] == **14**]}"**,** row['teacher']))  
  
 # Add a vertical scrollbar to the Treeview widget for 2-week exam schedule  
 scrollbar\_two\_weeks = ttk.Scrollbar(exam\_schedule\_frame\_two\_weeks**,** orient="vertical"**,** command=exam\_schedule\_tree\_two\_weeks.yview)  
 scrollbar\_two\_weeks.grid(row=**1,** column=**6,** sticky="ns")  
 exam\_schedule\_tree\_two\_weeks.configure(yscrollcommand=scrollbar\_two\_weeks.set)  
  
 # Create a Frame for 3-week exam schedule  
 exam\_schedule\_frame\_three\_weeks = ttk.Frame(exam\_schedule\_window)  
 exam\_schedule\_frame\_three\_weeks.pack(pady=**10**)  
  
 exam\_schedule\_label\_three\_weeks = tk.Label(exam\_schedule\_frame\_three\_weeks**,** text="3-Week Exam Schedule:"**,** font=("Arial"**, 14,** "bold"))  
 exam\_schedule\_label\_three\_weeks.grid(row=**0,** column=**0,** columnspan=**5,** pady=**5**)  
  
 exam\_schedule\_tree\_three\_weeks = ttk.Treeview(exam\_schedule\_frame\_three\_weeks**,** columns=('Course'**,** 'Room'**,** 'Date'**,** 'Time'**,** 'Teacher')**,** show='headings'**,** height=**15**)  
 exam\_schedule\_tree\_three\_weeks.grid(row=**1,** column=**0,** columnspan=**5,** padx=**5**)  
  
 exam\_schedule\_tree\_three\_weeks.heading('Course'**,** text='Course')  
 exam\_schedule\_tree\_three\_weeks.heading('Room'**,** text='Room')  
 exam\_schedule\_tree\_three\_weeks.heading('Date'**,** text='Date')  
 exam\_schedule\_tree\_three\_weeks.heading('Time'**,** text='Time')  
 exam\_schedule\_tree\_three\_weeks.heading('Teacher'**,** text='Teacher')  
  
 # Add data to the Treeview widget for 3-week exam schedule  
 for i**,** row in exam\_schedule\_df\_three\_weeks.iterrows():  
 exam\_schedule\_tree\_three\_weeks.insert(''**,** tk.END**,** values=(row['course']**,** row['room']**,** f"Day {row['date']}"**,** f"{row['time']} {['AM'**,** 'PM'][row['time'] == **14**]}"**,** row['teacher']))  
  
 # Add a vertical scrollbar to the Treeview widget for 3-week exam schedule  
 scrollbar\_three\_weeks = ttk.Scrollbar(exam\_schedule\_frame\_three\_weeks**,** orient="vertical"**,** command=exam\_schedule\_tree\_three\_weeks.yview)  
 scrollbar\_three\_weeks.grid(row=**1,** column=**6,** sticky="ns")  
 exam\_schedule\_tree\_three\_weeks.configure(yscrollcommand=scrollbar\_three\_weeks.set)  
  
 # Run the GUI event loop for exam schedule  
 exam\_schedule\_window.mainloop()  
  
 # Create a separate window for soft constraints  
 soft\_constraints\_window = tk.Tk()  
 soft\_constraints\_window.title("Soft Constraint Violations")  
  
 # Create a Frame to contain the soft constraint violations data for 2-week  
 soft\_constraints\_frame\_two\_weeks = ttk.Frame(soft\_constraints\_window)  
 soft\_constraints\_frame\_two\_weeks.pack(pady=**10**)  
  
 # Create a Label for the soft constraint violations section for 2-week  
 soft\_constraints\_label\_two\_weeks = tk.Label(soft\_constraints\_frame\_two\_weeks**,** text="Soft Constraint Violations (2-Week):"**,** font=("Arial"**, 14,** "bold"))  
 soft\_constraints\_label\_two\_weeks.grid(row=**0,** column=**0,** columnspan=**5,** pady=**5**)  
  
 # Create a Treeview widget to display the soft constraint violations data in a table format for 2-week  
 soft\_constraints\_tree\_two\_weeks = ttk.Treeview(soft\_constraints\_frame\_two\_weeks**,** columns=('Soft Constraint'**,** 'Violations')**,** show='headings'**,** height=**10**)  
 soft\_constraints\_tree\_two\_weeks.grid(row=**1,** column=**0,** columnspan=**5,** padx=**5**)  
  
 soft\_constraints\_tree\_two\_weeks.heading('Soft Constraint'**,** text='Soft Constraint')  
 soft\_constraints\_tree\_two\_weeks.heading('Violations'**,** text='Violations')  
  
 # Add data to the Treeview widget for 2-week  
 for i**,** row in soft\_constraints\_df\_two\_weeks.iterrows():  
 soft\_constraints\_tree\_two\_weeks.insert(''**,** tk.END**,** values=(row['Soft Constraint']**,** row['Violations']))  
  
 # Create a Frame to contain the soft constraint violations data for 3-week  
 soft\_constraints\_frame\_three\_weeks = ttk.Frame(soft\_constraints\_window)  
 soft\_constraints\_frame\_three\_weeks.pack(pady=**10**)  
  
 # Create a Label for the soft constraint violations section for 3-week  
 soft\_constraints\_label\_three\_weeks = tk.Label(soft\_constraints\_frame\_three\_weeks**,** text="Soft Constraint Violations (3-Week):"**,** font=("Arial"**, 14,** "bold"))  
 soft\_constraints\_label\_three\_weeks.grid(row=**0,** column=**0,** columnspan=**5,** pady=**5**)  
  
 # Create a Treeview widget to display the soft constraint violations data in a table format for 3-week  
 soft\_constraints\_tree\_three\_weeks = ttk.Treeview(soft\_constraints\_frame\_three\_weeks**,** columns=('Soft Constraint'**,** 'Violations')**,** show='headings'**,** height=**10**)  
 soft\_constraints\_tree\_three\_weeks.grid(row=**1,** column=**0,** columnspan=**5,** padx=**5**)  
  
 soft\_constraints\_tree\_three\_weeks.heading('Soft Constraint'**,** text='Soft Constraint')  
 soft\_constraints\_tree\_three\_weeks.heading('Violations'**,** text='Violations')  
  
 # Add data to the Treeview widget for 3-week  
 for i**,** row in soft\_constraints\_df\_three\_weeks.iterrows():  
 soft\_constraints\_tree\_three\_weeks.insert(''**,** tk.END**,** values=(row['Soft Constraint']**,** row['Violations']))  
  
 # Run the GUI event loop for soft constraints  
 soft\_constraints\_window.mainloop()  
  
# Constants  
fileDir = './test\_dataset/'  
num\_days\_two\_weeks = **14**num\_days\_three\_weeks = **21**max\_iterations = **1000**initial\_temperature = **1000.0**cooling\_rate = **0.95**# Load data  
teachers**,** students**,** rooms**,** courses**,** student\_courses = import\_data(fileDir)  
  
# Generate initial solution for two weeks  
initial\_solution\_two\_weeks = random\_solution(courses**,** students**,** rooms**,** num\_days\_two\_weeks)  
  
# Run simulated annealing algorithm to generate exam schedule for two weeks  
exam\_schedule\_2\_weeks = simulated\_annealing(initial\_solution\_two\_weeks**,** student\_courses**,** num\_days\_two\_weeks**,** max\_iterations**,** initial\_temperature**,** cooling\_rate)  
  
# Calculate soft constraints for two weeks  
soft\_constraints\_two\_weeks = calculate\_soft\_constraints(exam\_schedule\_2\_weeks**,** student\_courses**,** num\_days\_two\_weeks)  
  
# Generate initial solution for three weeks  
initial\_solution\_three\_weeks = random\_solution(courses**,** students**,** rooms**,** num\_days\_three\_weeks)  
  
# Run simulated annealing algorithm to generate exam schedule for three weeks  
exam\_schedule\_3\_weeks = simulated\_annealing(initial\_solution\_three\_weeks**,** student\_courses**,** num\_days\_three\_weeks**,** max\_iterations**,** initial\_temperature**,** cooling\_rate)  
  
# Calculate soft constraints for three weeks  
soft\_constraints\_three\_weeks = calculate\_soft\_constraints(exam\_schedule\_3\_weeks**,** student\_courses**,** num\_days\_three\_weeks)  
  
# Convert the exam schedules to DataFrames  
exam\_schedule\_df\_two\_weeks = pd.DataFrame(exam\_schedule\_2\_weeks.values())  
exam\_schedule\_df\_three\_weeks = pd.DataFrame(exam\_schedule\_3\_weeks.values())  
  
# Save exam schedules and soft constraint violations to CSV files  
exam\_schedule\_df\_two\_weeks.to\_csv('exam\_schedule\_two\_weeks.csv'**,** index=False)  
exam\_schedule\_df\_three\_weeks.to\_csv('exam\_schedule\_three\_weeks.csv'**,** index=False)  
  
soft\_constraints\_df\_two\_weeks = pd.DataFrame(soft\_constraints\_two\_weeks.items()**,** columns=['Soft Constraint'**,** 'Violations'])  
soft\_constraints\_df\_three\_weeks = pd.DataFrame(soft\_constraints\_three\_weeks.items()**,** columns=['Soft Constraint'**,** 'Violations'])  
  
soft\_constraints\_df\_two\_weeks.to\_csv('soft\_constraints\_two\_weeks.csv'**,** index=False)  
soft\_constraints\_df\_three\_weeks.to\_csv('soft\_constraints\_three\_weeks.csv'**,** index=False)  
  
# Display the exam schedules and soft constraint violations in GUIs  
display\_schedule\_gui(exam\_schedule\_df\_two\_weeks**,** exam\_schedule\_df\_three\_weeks**,** soft\_constraints\_df\_two\_weeks**,** soft\_constraints\_df\_three\_weeks)

output:

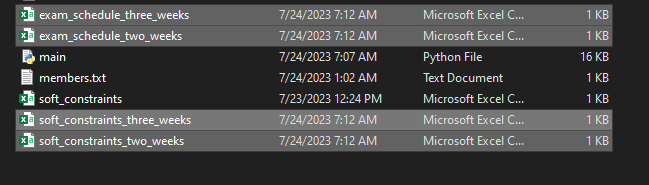
code generates iterations and fitness and four(4) CSV files

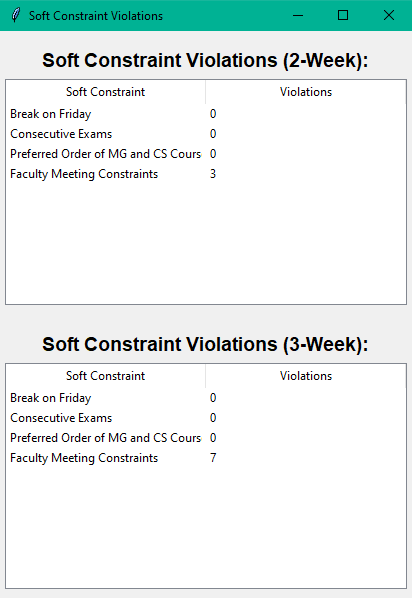
1. 2- Exam\_schedule.csv
2. 2- Soft\_constraints.csv
3. And also A GUI of timetable using tkinter with two interfaces

Output of latest code:



Files:





Files output:

