AI LAB 3

פייסל סעדיה 208336321

אחמד גבארין 314722307

שאלה 1:

אנחנו חילצנו את הפרמטרים שאנחנו צריכים מקבצי ה txt הנמצאים באתר ע"י קריאת השורות ואז לחלץ מכל שורה את הפרמטרים הרלוונטים לפי הפורמט של כל קבצי ה txt שיש באתר , קודם כל המימד אחר כך ה capacity ולאחר מכן הקורדינאטות וכן הלאה , הקוד שבו אנו מחלצים את הפרמטרים הוא כך :

```python
names = ['0', 'E-n22-k4', 'E-n33-k4', 'E-n51-k5', 'E-n76-k10']
faisalsadi *
def get_input(input_file):
    file = open(names[int(input_file)] + '.txt', 'r')
    file.readline()# first three line contains no information
    file.readline()
    file.readline()

    dimension = file.readline()
    arr = [k for k in dimension.split(' ')]
    dimension = int(arr[2])

    file.readline()

    capacity = file.readline()
    arr = [k for k in capacity.split(' ')]# extract capacity
    capacity = int(arr[2])

    file.readline()

    wareHouse = file.readline()
    arr = [k for k in wareHouse.split(' ')]# extract cities coordinates
    wareHouse = Point.Point(int(arr[0]), int(arr[1]), int(arr[2]))# first city is the wareHouse

    cities = []
    for _ in range(dimension - 1):
        cityLine = file.readline()
        arr = [num for num in cityLine.split(' ')]
        city = Point.Point(int(arr[0]) - 1, int(arr[1]), int(arr[2]))# extract coordinantes
        cities.append(city)
```

שאלה 2:

לצורך תיאום הבעיה לאלגוריתמים השונים בנינו אובייקט שמתאר את הבעיה קראנו לו CVRP אובייקט זה מכיל את כל המידע הנדרש : מקומות הערים , מטריצת המרחקים ביו כל 2 ערים , capacity , demand ועוד.....

הינה הגדרת ה Class :

```python
class CVRP:
    faisalsadi *
    def __init__(self, distanceMatrix, cities, capacity, size):
        self.cities = cities
        self.capacity = capacity
        self.size = size
        self.best = []
        self.bestFitness = 0
        self.distanceMatrix = distanceMatrix
```

לאובייקט זה הוספנו גם מתודות שמחשבות את העלות של פתרון מסויים שייצגנו אותו בתור מערך ומחלצים ממנו את הסכום ומספר המכוניות הנדרשות כדי להשלים את המסלול

למשל עבור הבעיה בתרגיל המערך הזה [1,2,3,4] אז חישוב העלות עבור מערך זה יחזיר לנו 80.6 ו 2 רכבים , כאשר רוצים להדפיס את הפתרון אז עוברים על המערך משמאל לימין ומחסירים את ה demand מ capacity עד שנגיע למצב שבו אנחנו לא יכולים לספק את העיר הנוכחי ואז מוסיפים רכב חדש שממשיך את המסלול מהנקודה האחרונה .

אנחנו הסתכלנו בכל פעם על השכנים של פתרון מסויים (פרמוטציה) ובוחרים את השכן שמקטין את אורך המסלול

מימוש :

```
faisalsadi *
def calcPathCost(self, path):
    left = self.capacity
    overall = 0
    overall += self.distanceMatrix[path[0]][0]
    left -= self.cities[path[0] - 1].capacity
    for i in range(0,len(path)-1):
        city1 = path[i]     # looking to the current 2 cities in the permutuaion
        city2 = path[i + 1]
        if self.cities[city2 - 1].capacity <= left: # the veichle can still apply to the next demand so no need to a new one
            left -= self.cities[city2 - 1].capacity
            overall += self.distanceMatrix[city1][city2]
        else: # new veichle is needed
            overall += self.distanceMatrix[city1][0] #move the last veichle to the ware house from the current point
            left = self.capacity # refill capacity because we have a new empty veichle
            overall += self.distanceMatrix[city2][0] # add the first distance for the new veichle
            left -= self.cities[city2 - 1].capacity # subtract the first demand from capcity
    overall += self.distanceMatrix[path[len(path)-1]][0] # return last veichle to the ware house
    return overall
```

לאורך כל האיטרציות אנו שומרים את הפתרון הטוב ביותר ואז בסוף מחלצים את המסלולים באותה שיטה שבא אנחנו חישבנו את העלות של כל המסלולים בפרמוטציה מסויימת.

שאלה 3:

כמו שנאמר בסעיף הקודם אנו מסתכלים על סדרת ערים מסוימת (פרמוטציה של ערים

כמו שהיה לנו בבעיית ה N מלכות) מייצגים אותה בתור מערך ואז מפעילים את הטכניקות שלמדנו במעבדות קודמות כמו שחלוף ומוטציות ועוד כדי להסתכל על השכנים של פתרון נוכחי.

מבצעים את החיפוש עד שמחזירים את הפתרון עם הכי פחות עלות , נעזרים במטריצת המרחקים שאנו בונים באתחול הבעיה ובכל פעם עבור משאית מסויימת אנחנו מתקדמים לעיר הכי קרובה לפי מטריצת המרחקים TSP.

שאלה 4 :

מימוש האלגוריתמים :

GA with island Model :

```
faisalsadi *
class Island:
    faisalsadi
    def __init__(self, CVRP, population_size):
        self.population = []
        self.buffer = []
        self.CVRP = CVRP
        self.init_population(population_size)
        self.migration_rate = 0.1
        self.immigrant_rate = 0.1

    new *
    def init_islands(self):
         for i in range(5):  # Create 5 islands
            island = Island(self.CVRP, popsize)
            self.islands.append(island)

    new *
    def migrate_between_islands(self):
        num_islands = len(self.islands)
        for i, island in enumerate(self.islands):
            if i < num_islands - 1:
                next_island = self.islands[i + 1]
                migrants = int(island.migration_rate * popsize)
                for j in range(migrants):
                    index = randint(0, popsize - 1)
                    migrant = island.population[index]
                    next_island.receive_migrant(migrant)
    faisalsadi
```

```
faisalsadi
def receive_migrant(self, migrant):
    self.buffer.append(migrant)

faisalsadi
def immigrate(self):
    num_immigrants = int(self.immigrant_rate * popsize)
    immigrants = []
    for i in range(num_immigrants):
        rand_str = init_sol(self.CVRP)
        immigrant = GAstruct(rand_str, 0)
        immigrants.append(immigrant)
    self.population.extend(immigrants)
    self.population = self.population[:popsize]
```

```
faisalsadi
def run(self):
    gr = []
    for i in range(maxIter):
        iterTime = time.time()
        self.calc_fitness()
        self.sort_by_fitness()
        self.pmx()
        self.CVRP.best = self.population[0].str
        self.CVRP.bestFitness = self.population[0].fitness
        print('Generation time: ', time.time() - iterTime)
        self.print_best()

        gr.append(self.population[0].fitness)
    Graph.draw(gr)
```

Tabu Search :

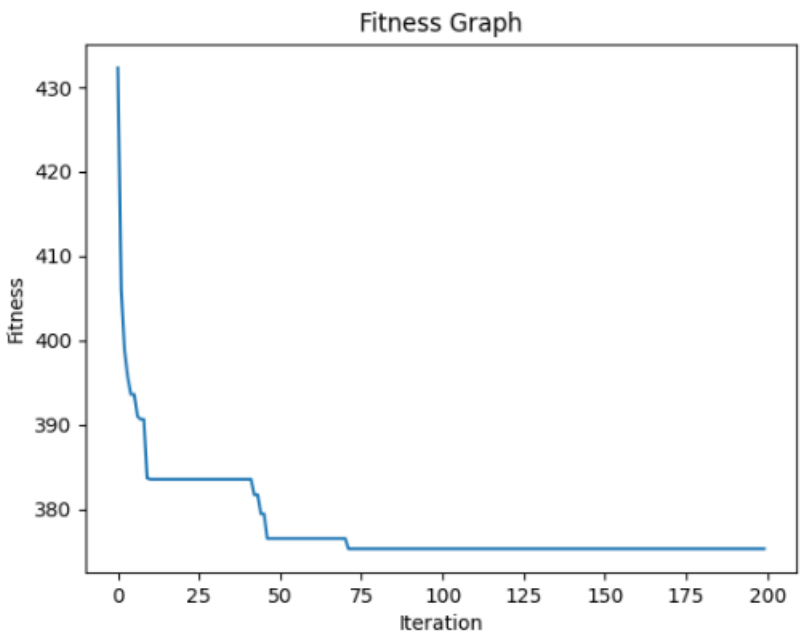
```
faisalsadi *
def tabuSearch_alg(problem, args):
    startTime = time.time()
    local_counter = 0
    ret = []
    best = init_solution(problem.size, problem)
    bestFitness, cars = problem.calcPathCost(best)
    bestCandidate = best
    globalBest = best
    globalFitness = bestFitness
    dict = {str(best): True}
    tabu = [best]
    gen=1
    for _ in range(args.maxIter):
        neighborhood = getNeighborhood(bestCandidate, args.numNeighbors)    # get neighborhood of current solution
        minimum, _ = problem.calcPathCost(neighborhood[0])
        bestCandidate = neighborhood[0]
        for neighbor in neighborhood:   # search for the best neighbor
            cost, _ = problem.calcPathCost(neighbor)
            if cost < minimum and not dict.get(str(neighbor), False):
                minimum = cost
                bestCandidate = neighbor
        if minimum < bestFitness:   # if found new best
            bestFitness = minimum
            best = bestCandidate
            local_counter = 0
        elif minimum == bestFitness:    # else incounter the local minimum number
            local_counter += 1
        if bestFitness < globalFitness: # update global minimum
```

```
        globalBest = best
        globalFitness = bestFitness
    tabu.append(bestCandidate)
    dict[str(bestCandidate)] = True
    if len(tabu) > args.maxTabu:
        dict[str(tabu[0])] = False
        tabu.pop(0)
    if local_counter == args.localOptStop:  # the case when stuck in local optimum ( reached the max number of local optimums)
        if bestFitness < globalFitness:
            globalBest = best
            globalFitness = bestFitness
        bestCandidate = init_solution(problem.size, problem)
        best = bestCandidate
        bestFitness, _ = problem.calcPathCost(best)
        local_counter = 0
        dict = {str(bestCandidate): True}
    ret.append(min(bestFitness,globalFitness) )
    print('********* Generation :#', gen, ' *************')
    print('best sol (permutation) = ', best)
    print('min cost = ', bestFitness)
    gen+=1
print('Elapsed Time : ', time.time() - startTime)
problem.best = globalBest
Graph.draw(ret)
problem.bestFitness = globalFitness
```

דוגמת הרצה : עבור הבעיה בתרגיל

```
********* Generation :# 199  *************
best sol (permutation) =  [3, 4, 1, 2]
min cost =  80.6449510224598
********* Generation :# 200  *************
best sol (permutation) =  [3, 4, 1, 2]
min cost =  80.6449510224598
Elapsed Time :  1.4697380065917969
route # 1 :   0 1 2 3 0
route # 2 :   0 4 0
Cost :  80.6449510224598
```

- גרף שיפור הפיטנס לבעיה הראשונה באתר E-n22-k4 :




```
Elapsed Time :  4.041363000869751
route # 1 :   0 14 21 19 16 0
route # 2 :   0 10 8 3 4 11 13 0
route # 3 :   0 17 20 18 15 12 0
route # 4 :   0 9 7 5 2 1 6 0
Cost :  375.2797871480124
```

Ant Colony :

```
faisalsadi *
def antColonyopt(problem, args):
    gr=[]
    bestPermutation, closest, currentBestPerm, currentBestclosest = reset()
    globalBest = []
    globalcloset = float('inf')
    local_counter = 0
    gen =1
    pheremonMatrix = [[float(1000) for _ in range(problem.size)] for _ in range(problem.size)]
    startTime = time.time()
    for _ in range(args.maxIter):
        tempPath = getPath(problem, pheremonMatrix, args)
        tempFitness, _ = problem.calcPathCost(tempPath)
        if tempFitness < currentBestclosest:
            currentBestclosest = tempFitness
            currentBestPerm = tempPath
        if currentBestclosest < closest:  # check neighborhood
            closest = currentBestclosest
            bestPermutation = currentBestPerm
            local_counter = 0
        if currentBestclosest == closest:   # incounter local optimum counter if still stuck in the same place
            local_counter += 1
        if closest < globalcloset:# best solution
            globalBest = bestPermutation
            globalcloset = closest
        gr.append(globalcloset)
        updatePheremons(pheremonMatrix, tempPath, tempFitness, args.Q, args.P)
        print('********* Generation :#',gen,' *************')
        print('sol = ', bestPermutation)
        print('cost = ', closest)
        gen+=1
        if local_counter == args.localOptStop:  # reset every parameter if we stuck for a long time
```

Cooperative Pso:

```python
def cooperative_pso(num_particles, num_iterations, num_customers, num_vehicles, depot, distance_matrix):
    omega = 0.5  # Inertia weight
    phi_p = 0.2  # Cognitive weight
    phi_g = 0.3  # Social weight

    particles = [Particle(num_customers, num_vehicles, depot, distance_matrix) for _ in range(num_particles)]
    gbest_position = np.copy(particles[0].position)
    gbest_fitness = float('inf')

    for _ in range(num_iterations):
        for particle in particles:
            particle.update_velocity(gbest_position, omega, phi_p, phi_g)
            particle.update_position()
            particle.evaluate_fitness()

            if particle.pbest_fitness < gbest_fitness:
                gbest_fitness = particle.pbest_fitness
                gbest_position = np.copy(particle.pbest_position)

    return gbest_position, gbest_fitness
```

Simulated Annealing :

```python
new *
def acceptance_probability(current_fitness, new_fitness, temperature):
    if new_fitness < current_fitness:
        return 1.0
    return np.exp((current_fitness - new_fitness) / temperature)


new *
def simulated_annealing(num_iterations, num_customers, num_vehicles, depot, distance_matrix, initial_temperature, cooling_rate):
    current_solution = generate_initial_solution(num_customers, num_vehicles, depot, distance_matrix)
    current_fitness = current_solution.total_distance
    best_solution = current_solution
    best_fitness = current_fitness
    temperature = initial_temperature

    for _ in range(num_iterations):
        neighborhood = get_neighborhood(current_solution)
        new_solution = random.choice(neighborhood)
        new_fitness = new_solution.total_distance

        if acceptance_probability(current_fitness, new_fitness, temperature) > random.random():
            current_solution = new_solution
            current_fitness = new_fitness

        if new_fitness < best_fitness:
            best_solution = new_solution
            best_fitness = new_fitness

        temperature *= cooling_rate

    return best_solution.routes, best_fitness
```

שאלה 5:

כיוון שאנו מסתכלים על השכנים של כל פתרון שמקטינים את אורך הסמלול הכולל אזי אחרי מספר של איטרציות אנחנו יכולים לשאוף לאורך האופטימלי הנדרש , כיוון שההיוריסטיקה משפרת את אורך המסלול באופן איטרטיבי וזה מתאים לאופי של בעיית ה CVRP .

לגבי יעילות ואיכות פתרון שמנו לב ש Tabu Search הניב את התוצאות הכי טובות מבחינת זמן ומבחינת אופטימליות והסיבות לכך הן:

1. גיוון הפתרון exploration : ברגע שנתקלים במינימום לוקלי אז אפשר לחלץ את התוכנית מהמצב הזה

2. חיפוש מבוסס על זכרון : אנחנו שומרים מצבים טובים כך שלא צריך לחזור לנקודת ה 0 בכל פעם שנתקעים

3. שומר על איזון בין exploration ל exploitation

כמובן שהבדלים משמעותיים ניתן לראות עבור הבעיות הגדולות יותר כמו הדוגמא 3 או 4 ששם המימדים הם יותר גדולים

שאלה 6:

לצורך זה הוספנו קובץ txt חדש בפורמט של הדוגמאות באתר שמתאר את הדוגמא בקובץ המשימה :

```
TYPE : CVRP
DIMENSION : 5
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 10
NODE_COORD_SECTION
1 0 0
2 0 10
3 -10 10
4 0 -10
5 10 -10
DEMAND_SECTION
1 0
2 3
3 3
4 3
5 3
DEPOT_SECTION
 1
 -1
EOF
```

תוצאה אחרי הרצת GA :

```
********* Generation :# 396  *************
best sol (permutation) =  [3, 4, 1, 2]
min cost =  80.6449510224598
********* Generation :# 397  *************
best sol (permutation) =  [3, 4, 1, 2]
min cost =  80.6449510224598
********* Generation :# 398  *************
best sol (permutation) =  [3, 4, 1, 2]
min cost =  80.6449510224598
********* Generation :# 399  *************
best sol (permutation) =  [3, 4, 1, 2]
min cost =  80.6449510224598
********* Generation :# 400  *************
best sol (permutation) =  [3, 4, 1, 2]
min cost =  80.6449510224598
Elapsed Time :  3.045062780380249
route # 1 :   0 1 2 3 0
route # 2 :   0 4 0
Cost :  80.6449510224598
```

<u>הערה :</u> ספיציפית בדוגמא שיש בקובץ המשימה יש יותר מפתרון אופטימלי אחד ולא רק זה שסופק לנו בקובץ המשימה , למשל הפתרון הבא הוא גם אופטימלי :

```
Time elapsed:  3.8929970264434814
route # 1 :   0 3 4 1 0
route # 2 :   0 2 0
Cost :  80.6449510224598
Choose one of these problems :
- 0 : exercise example
- 1 : E-n22-k4
- 2 : E-n33-k4
- 3 : E-n51-k5
```

שאלה 7:

גרף שמתאר את שיפור הפיטנס בבעיה 1: