מעבדה בבינה מלאכותית

1) לימדו את פורמט הקלט/פלט

אז אנחנו עשינו CLASS ל בעיה שלנו שבו יש מערך של הערים שלנו שבהם יש את המיקום ואת ה DEMAND לכל עיר

ובו גם יש את ה CAPACITY של כל מכונה וגם את הפתרון הכי טוב שהגענו אליו

```
class CVRP:

    def __init__(self, distanceMatrix,depot, cities, capacity, size):
        self.distanceMatrix = distanceMatrix
        self.depot = depot
        self.cities = cities
        self.capacity = capacity
        self.size = size
        self.best = []
        self.bestFitness = 0
```

וגם עשינו פונקציה לחישוב ה PATHCOST שלנו

```
    def calcPathCost(self, path):
        totalCost = 0
        capacity = self.capacity
        index = 0
        totalCost += self.distanceMatrix[path[index]][0]
        capacity -= self.cities[path[index] - 1].capacity
        vehicles = 1

        while index < (len(path) - 1):
            city1 = path[index]
            city2 = path[index + 1]
            if self.cities[city2 - 1].capacity <= capacity:
                cost = self.distanceMatrix[city1][city2]
                capacity -= self.cities[city2 - 1].capacity
                totalCost += cost
            else:
                totalCost += self.distanceMatrix[city1][0]
                capacity = self.capacity
                vehicles += 1
                totalCost += self.distanceMatrix[city2][0]
```

2)התאימו את בעית הדוגמא לעיל לפורמט הקלט ולכל אלגוריתם שאתם נדרשים לפתח

אחרי שהגדרנו את הקלט שלנו אנחנו שולחים לכל אחד מהאלגוריתמים שאנחנו צריכים לממש מופע של הבעיה ואז האלגוריתם מחשב את החלוקה האופטימלית (מימוש מפורט בסעיפים הבאים)

3) עבור האלגוריתמים השונים פתחו היוריסטיקות שונות שיכולות לסייע בפתרון

היוריסתיקה שפתחנו היא הבאה:

אנחנו מסתכלים על בעיית פרמוטציה של הערים ואז מסתכלים על כל השכנים ובוחרים את השכן שמקטין את אורך המסלול

אנחנו מסתכלים על השכנים שהוא פרמוטציה של הערים שאנחנו מייצגים בעזרת מערך

ואז מספר המשאיות הוא שלוקחים משאית עוברים על פרמוטציה ואז כל עוד המשאית יכולה לעבור לעיר הבאה במערך אנחנו ממשיכים בה אם לא מתחילים במשאית חדשה

סיבוכיות : o(n!)

4) קדדו את האלגוריתמים

Tabu search:

```python
def tabuSearch(problem, args):
    startTime = time.time()
    points = []
    best = initGreedySol(problem.size, problem)
    bestFitness, mypath = problem.calcPathCost(best)
    bestCandidate = best
    globalBest = best
    globalFitness = bestFitness
    tabuDict = {str(best): True}
    tabu = [best]
    local_counter = 0
    for _ in range(args.maxIter):
        iterTime = time.time()
        neighborhood = getNeighborhood(bestCandidate, args.numNeighbors)    # get neighborhood of current
        minimum, _ = problem.calcPathCost(neighborhood[0])
        bestCandidate = neighborhood[0]
        for neighbor in neighborhood:   # get the best neighbor and save it
            cost, _ = problem.calcPathCost(neighbor)
            if cost < minimum and not tabuDict.get(str(neighbor), False):
                minimum = cost
```

```python
        cost, _ = problem.calcPathCost(neighbor)
        if cost < minimum and not tabuDict.get(str(neighbor), False):
            minimum = cost
            bestCandidate = neighbor
    if minimum < bestFitness:   # update best (take a step towards the better neighbor)
        bestFitness = minimum
        best = bestCandidate
        local_counter = 0
    elif minimum == bestFitness:    # to detect local optimum
        local_counter += 1
    if bestFitness < globalFitness: # update the best solution found untill now
        globalBest = best
        globalFitness = bestFitness
    tabu.append(bestCandidate)
    tabuDict[str(bestCandidate)] = True
    if len(tabu) > args.maxTabu:
        tabuDict[str(tabu[0])] = False
        tabu.pop(0)
    if local_counter == args.localOptStop:  # if fallen into local optimum, reset and continue with the
        if bestFitness < globalFitness:
            globalBest = best
```

```python
    if local_counter == args.localOptStop:  # if fallen into local optimum, reset and
        if bestFitness < globalFitness:
            globalBest = best
            globalFitness = bestFitness
        bestCandidate = initGreedySol(problem.size, problem)
        best = bestCandidate
        bestFitness, _ = problem.calcPathCost(best)
        local_counter = 0
        tabuDict = {str(bestCandidate): True}
    points.append(bestFitness)
    print('Generation time: ', time.time() - iterTime)
    print('sol = ', best)
    print('cost = ', bestFitness)
    print()
print('Time elapsed: ', time.time() - startTime)
problem.best = globalBest   # save the solution and its fitness
Graph.draw(points)
problem.bestFitness = globalFitness
```

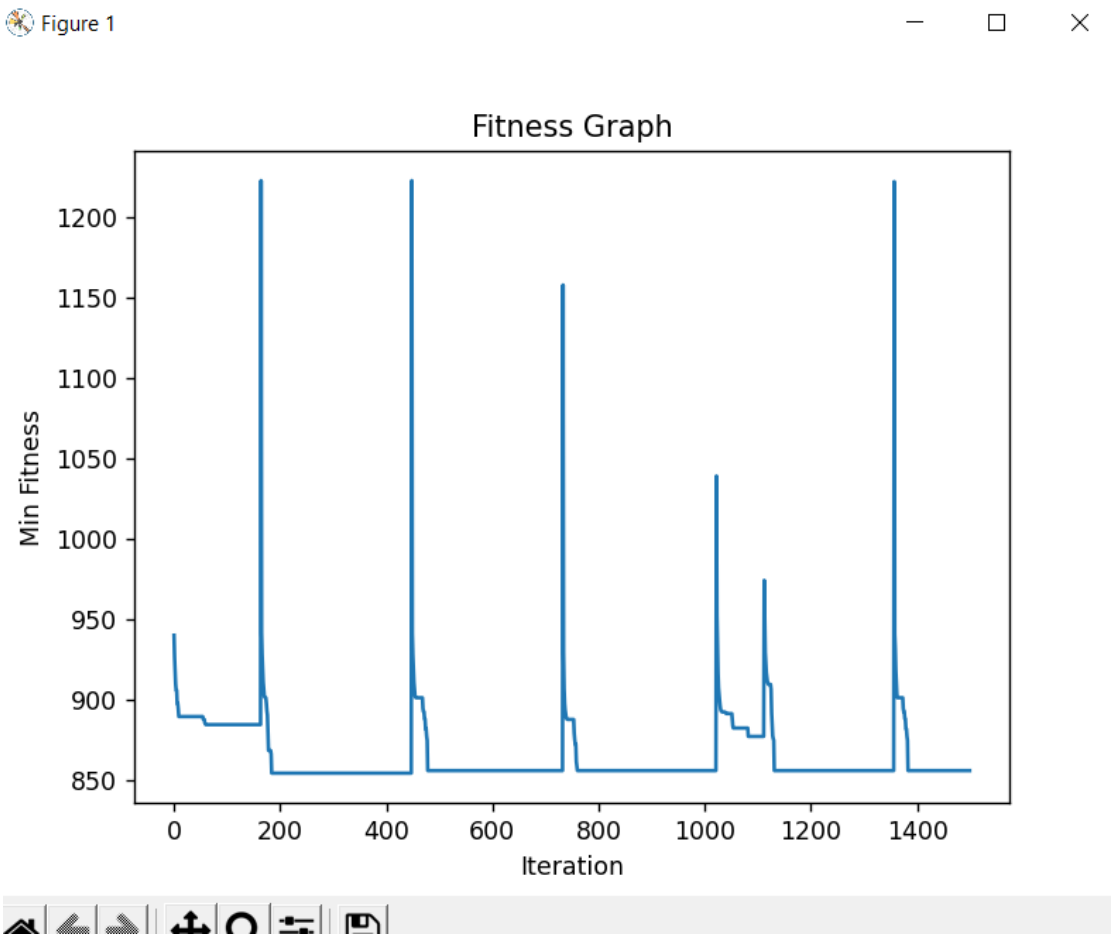
דוגמת ריצה: (תוצאה סופית עבור הדוגמה שבמשימה)

```
Generation time:  0.006982326507568359
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.00712895393371582
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Time elapsed:  10.8008713722229
80.6449510224598
0 1 2 3 0
0 4 0
```

דוגמה לגרף פיטנס: (הערה : אנחנו מראים גרף שמייצג המינימום פיטנס פיר GENERATION ולכן הגרף עולה ויורד הרבה אבל במהלך הריצה אנחנו שומרים על הפתרון האופטימלי ומדפיסים אותו!!!!!)



ACO:

```python
def ACO(problem, args):
    myarray=[]
    startTime = time.time()
    pheremonMatrix = [[float(1000) for _ in range(problem.size)] for _ in range(problem.size)]
    bestPath = []
    bestFitness = float('inf')
    currentBestPath = []
    currentBestFitness = float('inf')
    globalBest = []
    globalFitness = float('inf')
    local_counter = 0
    for _ in range(args.maxIter):
        iterTime = time.time()
        tempPath = getPath(problem, pheremonMatrix, args)
        tempFitness, _ = problem.calcPathCost(tempPath)
        if tempFitness < currentBestFitness:
            currentBestFitness = tempFitness
            currentBestPath = tempPath
        if currentBestFitness < bestFitness:  # update best (take a step towards the better neighbor)
            bestFitness = currentBestFitness
            bestPath = currentBestPath
            local_counter = 0
        if currentBestFitness == bestFitness:   # to detect local optimum
            local_counter += 1
        if bestFitness < globalFitness:# update the best solution found untill now
            globalBest = bestPath
            globalFitness = bestFitness
        myarray.append(globalFitness)
```

עם עוד פונקציות

דוגמת ריצה עבור הדוגמה הנתונה :

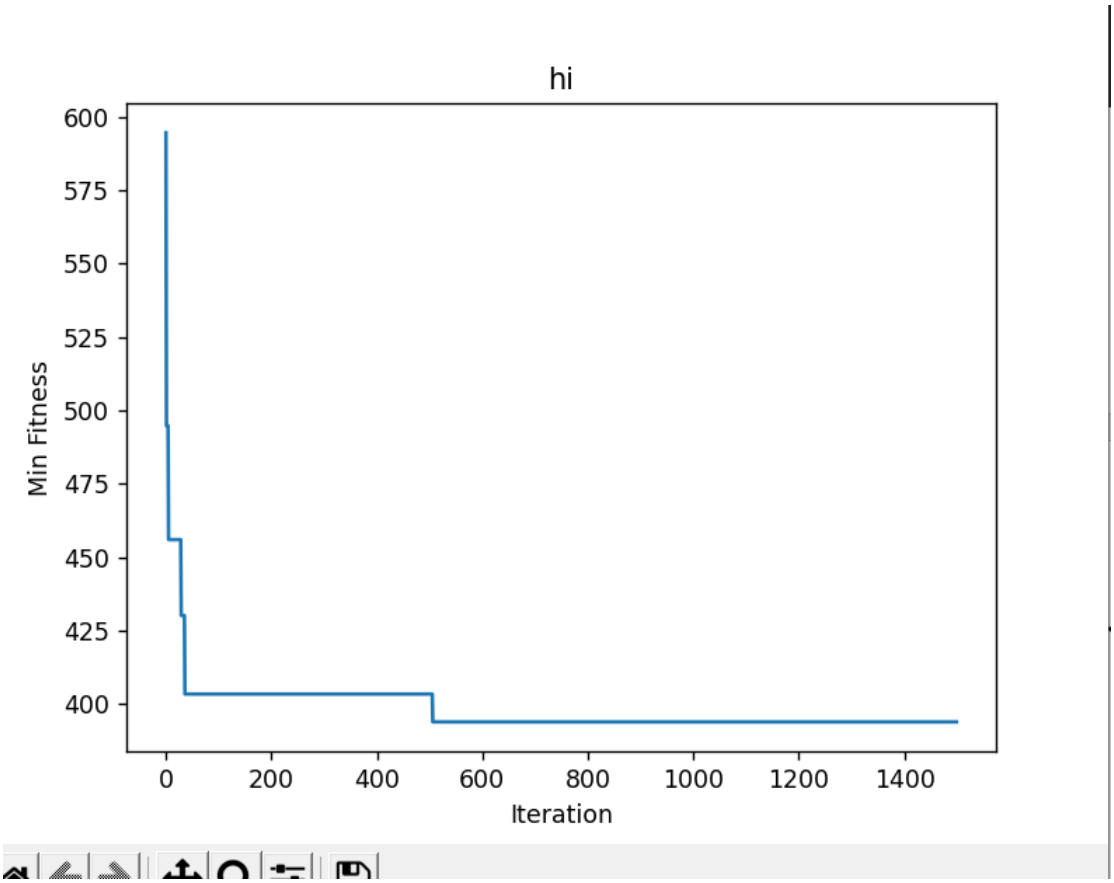
```
Generation time:  0.0009970664978027344
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.0
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.0
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Time elapsed:  0.420642614364624
80.6449510224598
0 4 1 2 0
0 3 0
```

דוגמה לגרף פיטנס



Simulated annealing :

```python
def simulatedAnnealing(problem, args):
    startTime = time.time()
    points = []
    best = initGreedySol(problem.size, problem)
    bestFitness, _ = problem.calcPathCost(best)
    globalBest = best
    globalFitness = bestFitness
    currentBest = best
    currentFitness = bestFitness
    temperature = float(args.temperature)
    local_counter = 0
    LK = 30
    for _ in range(args.maxIter):
        iterTime = time.time()
        neighborhood = getNeighborhood(best, args.numNeighbors)
        for _ in range(LK):  # pick 'LK' neighbors and get the best
            randNeighbor = neighborhood[randint(0, len(neighborhood) - 1)]
            neighborFitness, _ = problem.calcPathCost(randNeighbor)
            diff = neighborFitness - bestFitness
            metropolis = float(exp(float(-1 * diff) / temperature))
            if neighborFitness < currentFitness or rand() < metropolis:
                currentFitness = neighborFitness
```

```python
            if neighborFitness < currentFitness or rand() < metropolis:
                currentFitness = neighborFitness
                currentBest = randNeighbor
        if currentFitness < bestFitness:  # update best (take a step towards the better neighbor)
            best = currentBest
            bestFitness = currentFitness
            local_counter = 0
        if currentFitness == bestFitness:  # to detect local optimum
            local_counter += 1
        if bestFitness < globalFitness:  # update the best solution found untill now
            globalBest = best
            globalFitness = bestFitness
        if local_counter == args.localOptStop:  # if fallen into local optimum, reset and continue w
            if bestFitness < globalFitness:
                globalBest = best
                globalFitness = bestFitness
            best = initGreedySol(problem.size, problem)
            bestFitness, _ = problem.calcPathCost(best)
            currentBest = best
            currentFitness = bestFitness
            local_counter = 0
            temperature = float(args.temperature)
```

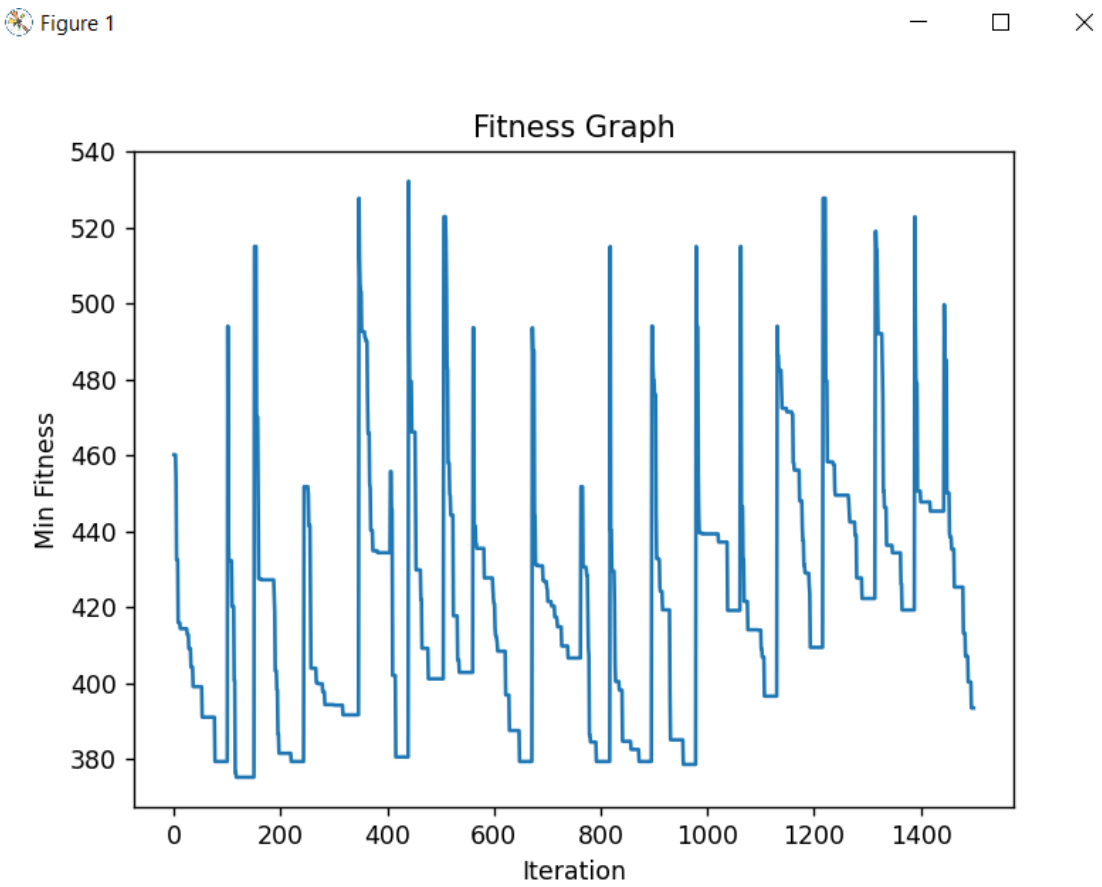
דוגמא לריצה:

```
Generation time:  0.003979682922363281
sol =  [1, 2, 3, 4]
cost =  80.6449510224598

Generation time:  0.003955364227294922
sol =  [1, 2, 3, 4]
cost =  80.6449510224598

Time elapsed:  6.836350679397583
80.6449510224598
0 1 2 3 0
0 4 0
```

דוגמא לגרף פיטנס:



GA:

לקחנו אותו מהמעבדות הקודמות דומה מאות ל NQUEENS בגלל שבשניהן עבדנו עם מערכים במימוש שלנו

דוגמת ריצה:

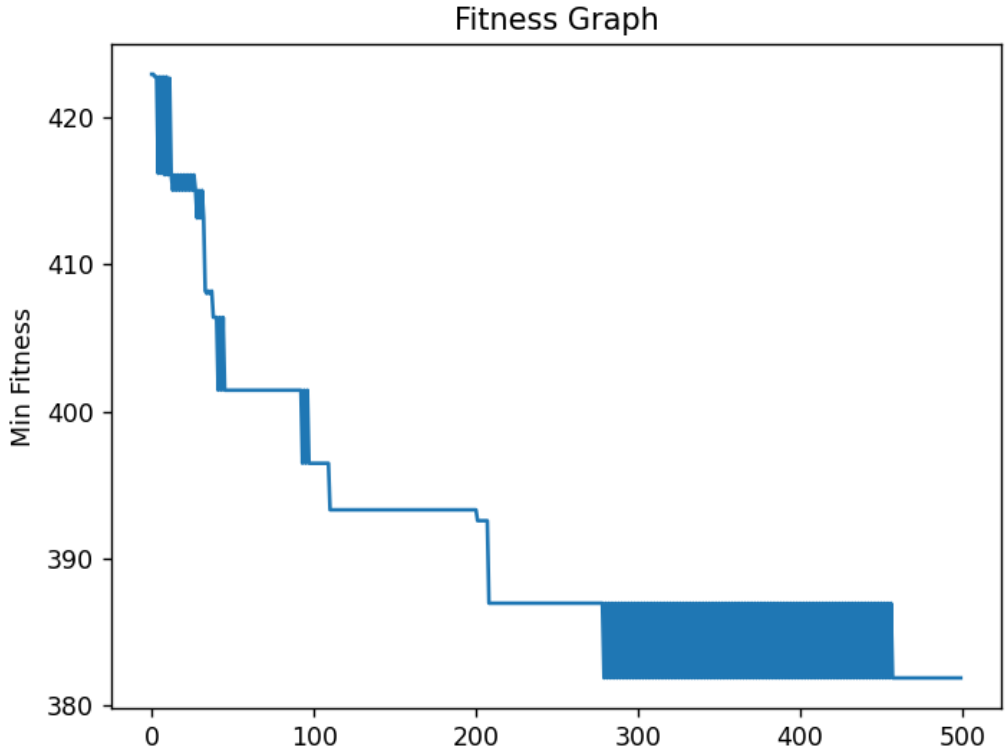
```
Generation time:  0.023935317993164062
sol =  [1, 2, 3, 4]
cost =  80.6449510224598

Generation time:  0.026896238327026367
sol =  [1, 2, 3, 4]
cost =  80.6449510224598

Generation time:  0.02496480941772461
sol =  [1, 2, 3, 4]
cost =  80.6449510224598

80.6449510224598
0 1 2 3 0
0 4 0
```

דוגמא לגרף פיטנס:



PSO :

גם הוא לקחנו ממעבדה שעברה(עבור חישובים שצריכים לבצע) למשל עבור חישוב החלקיקים עשינו כך:

```
for k in range(self.CVRP.size):
    # here we calculate the new string for each parcial
    # and update the velocity and position of it
    # using the formulas we saw in the lecture
    num1 = rand1 * self.C1 * (self.population[j].selfBest[k] - self.population[j].str[k])
    num2 = rand2 * self.C2 * (global_best[k] - self.population[j].str[k])
    num3 = self.W * (self.population[j].velocity[k])
    num4 = num1 + num2 + num3
    num6=(int(num4) % self.CVRP.size) +1
    #while num6 in arr1:
        #num6= randint(1,self.CVRP.size)
    arr1.append(num6)

    num5 = (arr1[k] + self.population[j].str[k])%self.CVRP.size+1
    #while num5 in arr2:
     ##   num5= randint(1,self.CVRP.size)
    arr2.append(num5)
```

ולמימוש CPSO עשינו ריצה לכמה ריצות לpso ואז לקחנו תשובות ועשינו מיזוג:

```
def cooperative_pso(self):
    global_best = float('inf')
    self.init_population()
    self.calc_fitness()
    self.sort_by_fitness()
    for i in range(iters):

        local_array = []
        for j in range(runs):
            local_array.append((self.pso_run()))
        self.update_parameters(iters,runs)

        local_best = self.cross_over(local_array)

        local_fitness= self.calc_fitness(local_best)
        if local_fitness < global_best:
            self.CVRP.best = copy.deepcopy(local_best)
            global_best = local_fitness
            self.CVRP.bestFitness=local_fitness
```

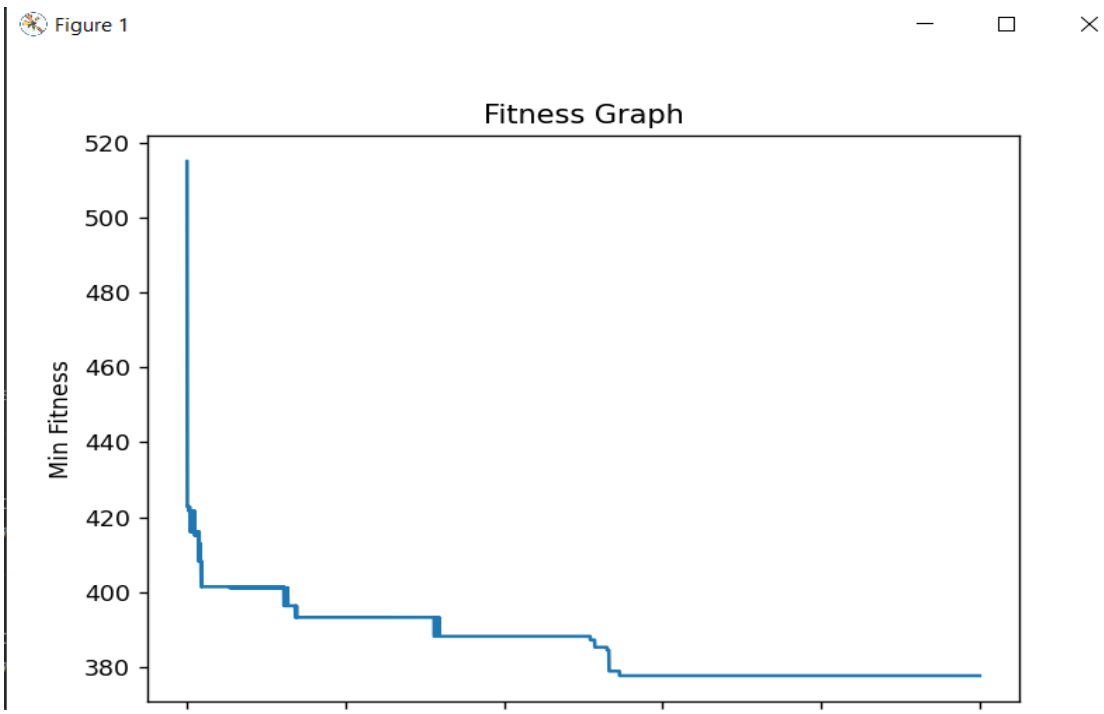
דוגמאת ריצה:

```
Generation time:  0.007141590118408203
sol =  [2, 1, 4, 3]
cost =  80.6449510224598

Generation time:  0.006981611251831055
sol =  [2, 1, 4, 3]
cost =  80.6449510224598

Time elapsed:  17.25322961807251
80.6449510224598
0 1 2 3 0
0 4 0
```

דוגמא לפיטנס:



5) עבור כל היוריסטיקה בה אתם עושים שימוש הסבירו את יתרונותיה ביחס לבעית הCVRP

יתרון היוריסטיקה הוא שאנחנו מחפשים מסלול שמשפר את העלות של מסלולים של המשאית ובגלל זה אנחנו נשאוף לפתרון אופטימלי עם מספיק זמן שהוא מסלול קצר ביותר שדרכו אנחנו מספקים את ה DEMAND לכל עיר כמו שנדרש

סעיף 7+6 עשינו עם 4

לסיכום יצא לנו ש ACO הוא הכי טוב אחרי זה יבוא SA והשאר דומים מבחינת ביצועים CPSO היה הכי כבד ו GA היה צריך הרבה יותר איטרציות להתכנס אצלנו