

**דוח מעבדה 1:**

תוכנית ה`ex0` מציגה UI כדי להכניס את הפרמטרים הרלוונטיים לכל בעיה. מה שמצוג על המסך אלה הערכים האופטימליים שנמצאו.

1. הוסיפו חישוב ודוחה בכל דור של ממוצע ה- FITNESS של האוכלוסייה  
ושל סטיית התקן מהממוצע

פונקציה המחשבת את ממוצע ה-FITNESS

```
def calcFitness(self):  
    self.sumFitness = 0  
    if self.fitnessType == FitnessType.REGULAR:  
        for person in self.population:  
            fitness = self.regularFitness(person.getString())  
            person.setFitness(fitness)  
            self.sumFitness += fitness  
    else:  
        for person in self.population:  
            fitness = self.bullseyeFitness(person.getString())  
            person.setFitness(fitness)  
            self.sumFitness += fitness
```

כמו שאנו רואים בקוד, אנחנו מחשבים את ה-FITNESS לפי סוג ההייריסטיקה ווכמיים אותו לתוך משתנה ששומר לנו את סכוםם (לא שמרנו אותו ישירות כממוצע כדי להשאיר אופציה להשתמש בסכום בכל מקום אחר אם נרצה).

פונקציה המחשבת את סטיית התקן מהממוצע:

```
def calcAvgSd(self):  
    avg = self.sumFitness / GA_POPSIZE  
  
    sum = 0  
    for person in self.population:  
        sum += (avg - person.getFitness()) ** 2  
    sd = abs(sum / GA_POPSIZE)  
    sd = math.sqrt(sd)  
  
    print("fitness data: avarage: ", avg, " || standard deviation:", sd, end=" ")
```

## 2. הוסיף חישוב ודוחה בכל דור של זמן ריצה CLOCK TICKS וזמן ריצה ABSOLUTE ELAPSED וכן עד להתקנות למינימום מקומי או גלובלי.

מצורף תמונה להרצת הקוד שמרת את זמן הריצה עבור כל איטרציה, ועובד זמן ריצת התוכנית עד להגיאן לפתרון:

```
Best: WUqsy9]rihXT ( 163 ) fitness data: avarage: 400.232421875 || standard deviation: 78.16443182130234 Generation time: 0.04687380790710449
Best: WUqsy9]rihX ( 113 ) fitness data: avarage: 298.0068359375 || standard deviation: 45.58272819303336 Generation time: 0.03889656066894531
Best: 4ulaj$Zrod** ( 88 ) fitness data: avarage: 228.40576171875 || standard deviation: 33.84659074535938 Generation time: 0.0449063777923584
Best: C\lbh!Uqdiv ( 72 ) fitness data: avarage: 176.81982421875 || standard deviation: 27.306604368844564 Generation time: 0.05781841278076172
Best: Je[tj$Zrod** ( 67 ) fitness data: avarage: 136.2529296875 || standard deviation: 21.537972717056757 Generation time: 0.06086087226867676
Best: Pfolx#^pqgd( ( 45 ) fitness data: avarage: 106.015625 || standard deviation: 17.824961375747073 Generation time: 0.053895851860046387
Best: C\lmn Wrosd! ( 29 ) fitness data: avarage: 83.27001953125 || standard deviation: 14.745160739510219 Generation time: 0.056821584701538086
Best: Ihmmn Wppsj ( 24 ) fitness data: avarage: 66.0791015625 || standard deviation: 13.375532727906954 Generation time: 0.0429081916809082
Best: Ihmmn Wrosd! ( 20 ) fitness data: avarage: 52.1484375 || standard deviation: 11.714642509316866 Generation time: 0.04086732864379883
Best: Ihmmn Wppmb! ( 13 ) fitness data: avarage: 40.54052734375 || standard deviation: 12.261725817942944 Generation time: 0.04188823699951172
< t: Icmlp Wppmb! ( 11 ) fitness data: avarage: 30.5678109375 || standard deviation: 11.581541877687668 Generation time: 0.03393650054931640 >
Best: Ihmmn Wpql! ( 9 ) fitness data: avarage: 23.53955078125 || standard deviation: 11.210494794976826 Generation time: 0.03388190269470215
Best: Hdmmn Wpql! ( 6 ) fitness data: avarage: 18.68701171875 || standard deviation: 11.01739512621065 Generation time: 0.03391075134277344
Best: Hdmmn Wpql! ( 6 ) fitness data: avarage: 14.80224609375 || standard deviation: 10.199736460385282 Generation time: 0.03294181823730469
Best: IflLn Wpql! ( 5 ) fitness data: avarage: 12.796875 || standard deviation: 11.40260001531997 Generation time: 0.03387761116027832
Best: Hdmln Woql! ( 4 ) fitness data: avarage: 10.154296875 || standard deviation: 10.190069705324161 Generation time: 0.035927772521972656
Best: HfllO Wpql! ( 3 ) fitness data: avarage: 8.583984375 || standard deviation: 10.667505847186392 Generation time: 0.034883975982666016
Best: Hello World! ( 0 ) fitness data: avarage: 7.599609375 || standard deviation: 11.407142081714074 Generation time: 0.01012077331542969
Time elapsed: 0.7380671501159668

Process finished with exit code 0
```

## 3. ממשו כך שאפשר יהיה לבחור בין שלושה אופרטורים לשיחולו SINGLE,TWO,UNIFORM

לנוחות הקראיה, אנו מצרפים תמונות לקטעי הקוד עבור כל פונקציה:

1-Single point:

```
def onePointCrossover(self):
    for i in range(self.esize, GA_POPSIZE):
        i1 = randint(0, int(GA_POPSIZE / 2) - 1)
        i2 = randint(0, int(GA_POPSIZE / 2) - 1)
        spos = randint(0, self.tsize - 1)

        str1 = self.population[i1].getString()
        str2 = self.population[i2].getString()
        string1 = str1[0: spos] + str2[spos:]
        string2 = str2[0: spos] + str1[spos:]
        self.nextPopulation[i].setString(self.calcStringFitness(string1, string2))

        if random() < GA_MUTATION:
            self.mutate(self.nextPopulation[i])
```

## 2-Two point:

```
def twoPointCrossover(self):
    for i in range(self.esize, GA_POPSIZE):
        i1 = randint(0, int(GA_POPSIZE / 2) - 1)
        i2 = randint(0, int(GA_POPSIZE / 2) - 1)
        spos1 = randint(0, self.tsize - 2)
        spos2 = randint(spos1 + 1, self.tsize - 1)

        str1 = self.population[i1].getString()
        str2 = self.population[i2].getString()
        string1 = str1[0: spos1] + str2[spos1: spos2] + str1[spos2:]
        string2 = str2[0: spos1] + str1[spos1: spos2] + str2[spos2:]
        self.nextPopulation[i].setString(self.calcStringFitness(string1, string2))

        if random() < GA_MUTATION:
            self.mutate(self.nextPopulation[i])
```

## 3-Uniform:

```
def uniformCrossover(self):
    for i in range(self.esize, GA_POPSIZE):
        i1 = randint(0, int(GA_POPSIZE / 2) - 1)
        i2 = randint(0, int(GA_POPSIZE / 2) - 1)
        string1 = ''
        string2 = ''

        for j in range(0, self.tsize):
            if (int(random()) % 2) == 0:
                string1 = string1 + self.population[i1].getString()[j]
                string2 = string2 + self.population[i2].getString()[j]
            else:
                string1 = string1 + self.population[i2].getString()[j]
                string2 = string2 + self.population[i1].getString()[j]

        self.nextPopulation[i].setString(self.calcStringFitness(string1, string2))

        if random() < GA_MUTATION:
            self.mutate(self.nextPopulation[i])
```

מימושו את שלושת הfonקציות, יש אפשרות לבחור את הפונקציה שרצים ולבוד לפיה.

4. הוסיף היוריסטייקה נוספת של "בול פגיעה" – פונקציה "המצ'פרת"  
ניחוש אותה במחuzeות ולו אם אינה במקומ הנכון וכן נותרת בונוס גדול  
על ניחוש אותה במקום הנכון.

מצורפת תמונה שמכילה את קוד ההיוריסטייקה:

```
def bullseyeFitness(self, string):  
    fitness = 0  
    for i in range(0, self.tsize):  
        char = string[i]  
        if char == GA_TARGET[i]:  
            fitness += 0  
        elif char in GA_TARGET:  
            fitness += 30  
        else:  
            fitness += 80  
    return fitness
```

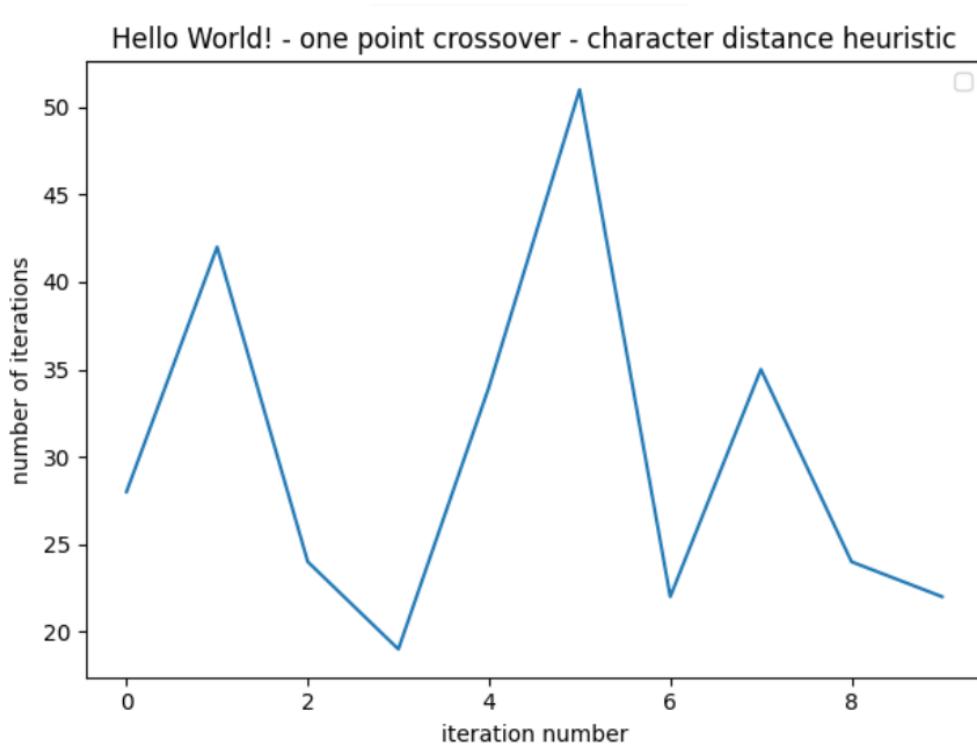
הסביר על היוריסטייקה:

- 1-אם חזינו אותה נכון במקומות הנכונים ← מעולה.
- 2-אם חזינו אותה נכון לא נכון במקומות הנכונים ← נשלם קנס בס"כ 30.
- 3-אם חזינו אותה לא נכון ← נשלם קנס כבד "80".

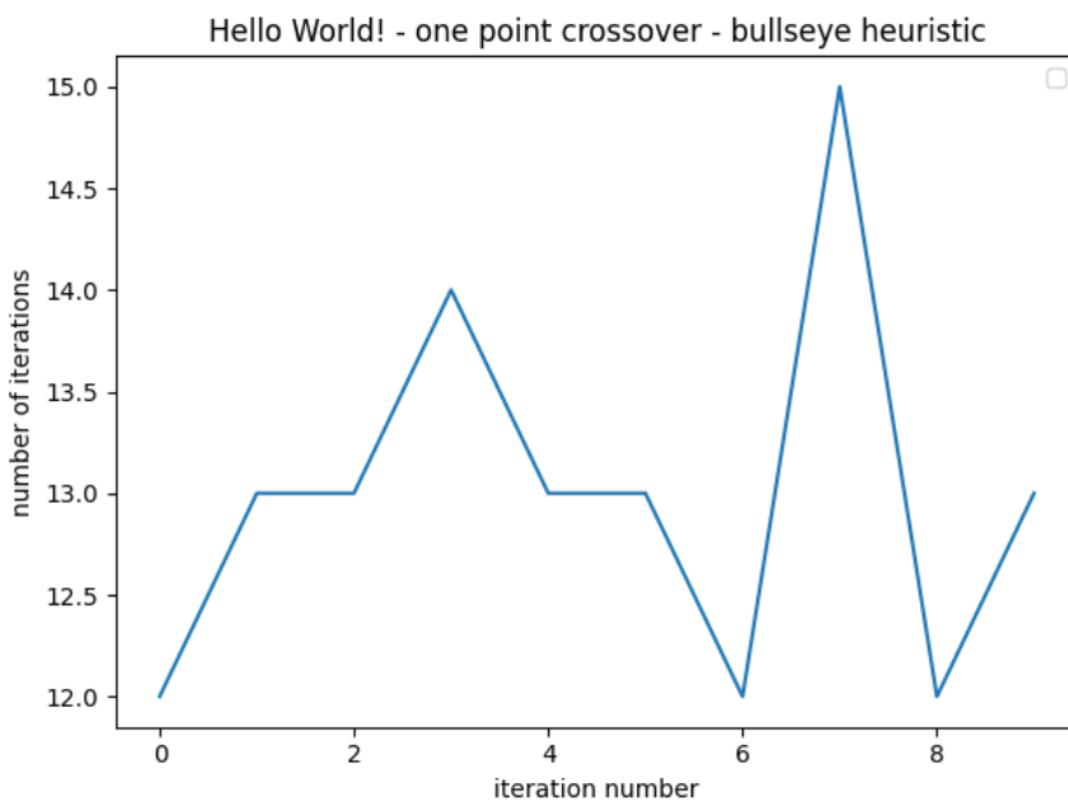
5. השוו את היוריסטייקה זו מול המקורית. (הסבירו כיצד היא קשורה  
לモוציאות במהלך האלגוריתם?) – האם היא משפרת את היוריסטייקה  
המקורית ואם לאו מדוע?

אנו מצרפים מספר גרפים שעשינו אותם עבור מספר איטרציות עבור כל היוריסטייקה עבור כל סוג  
(SINGLE\TWO\UNIFORM)

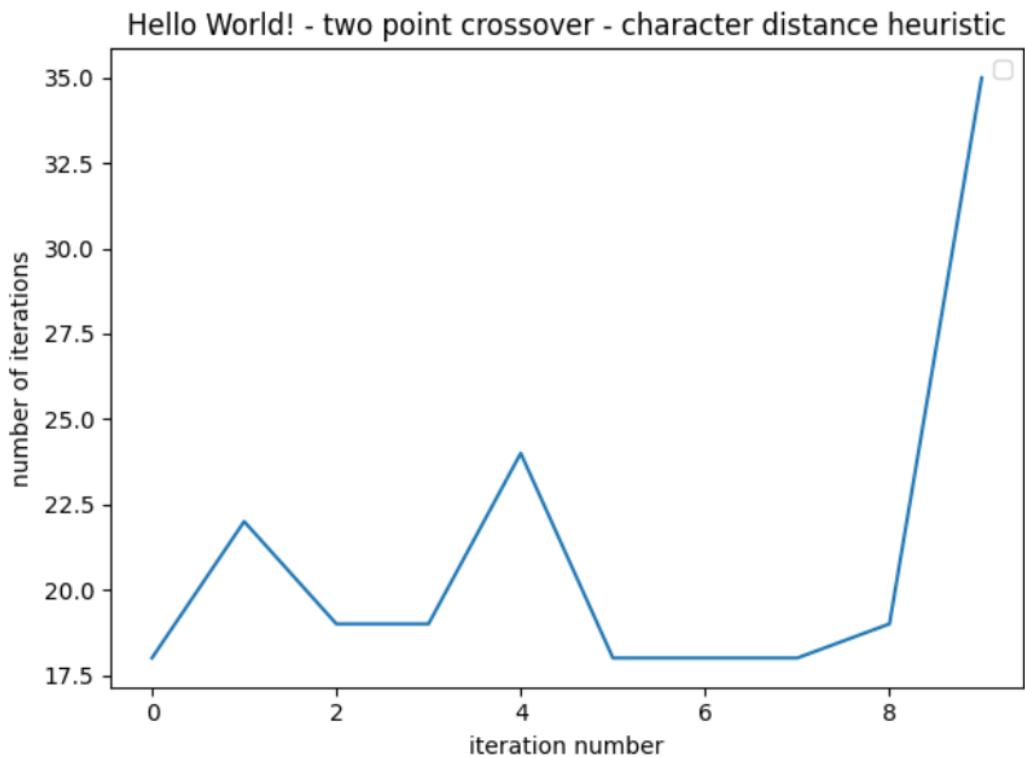
עובד היוריסטיקה מקורית **SINGLE-POINT**



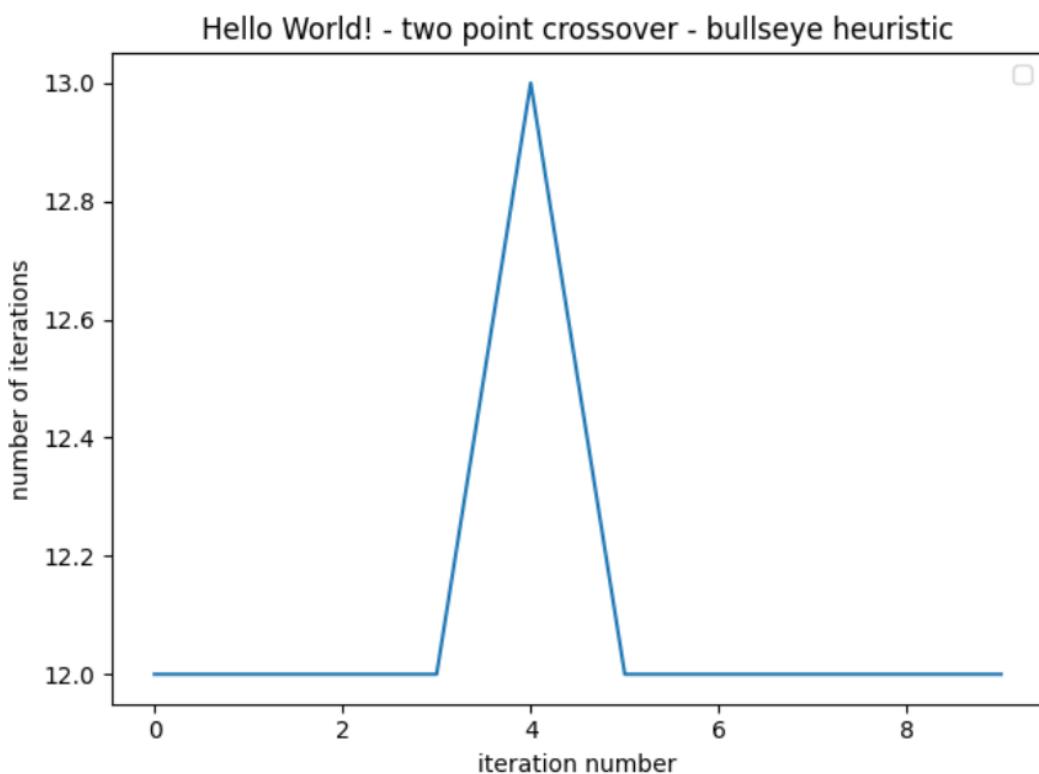
עובד היוריסטיקה "פול-פגיעה" **SINGLE-POINT**



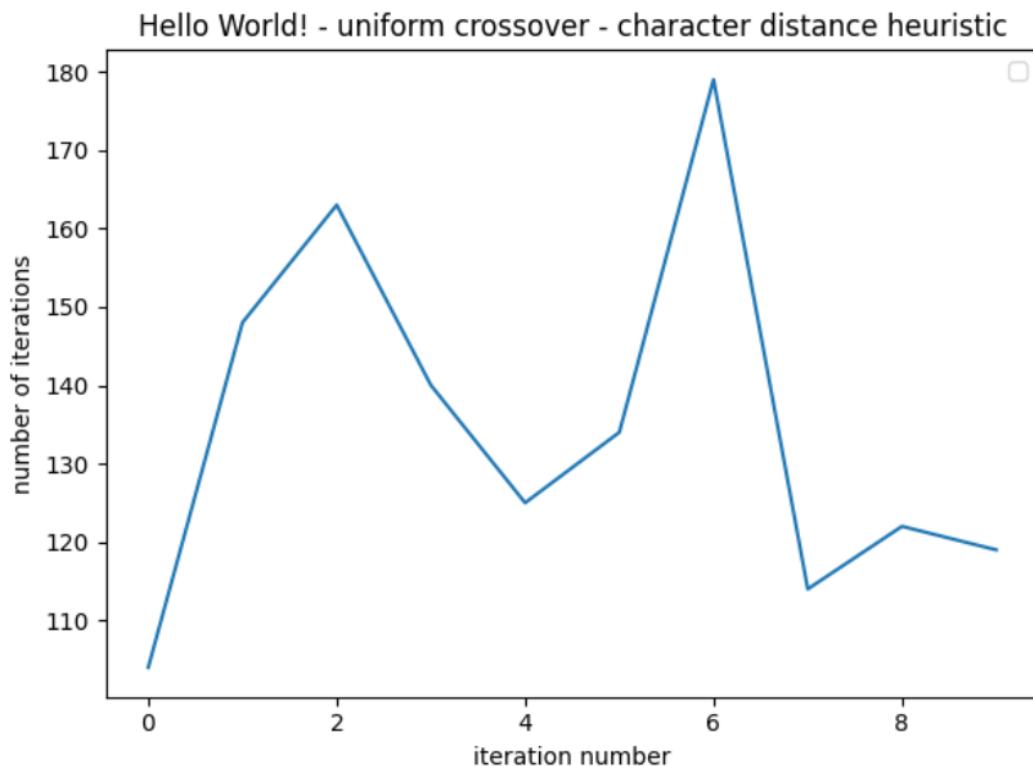
עובד היוריסטיקה מקורית :TWO-POINT



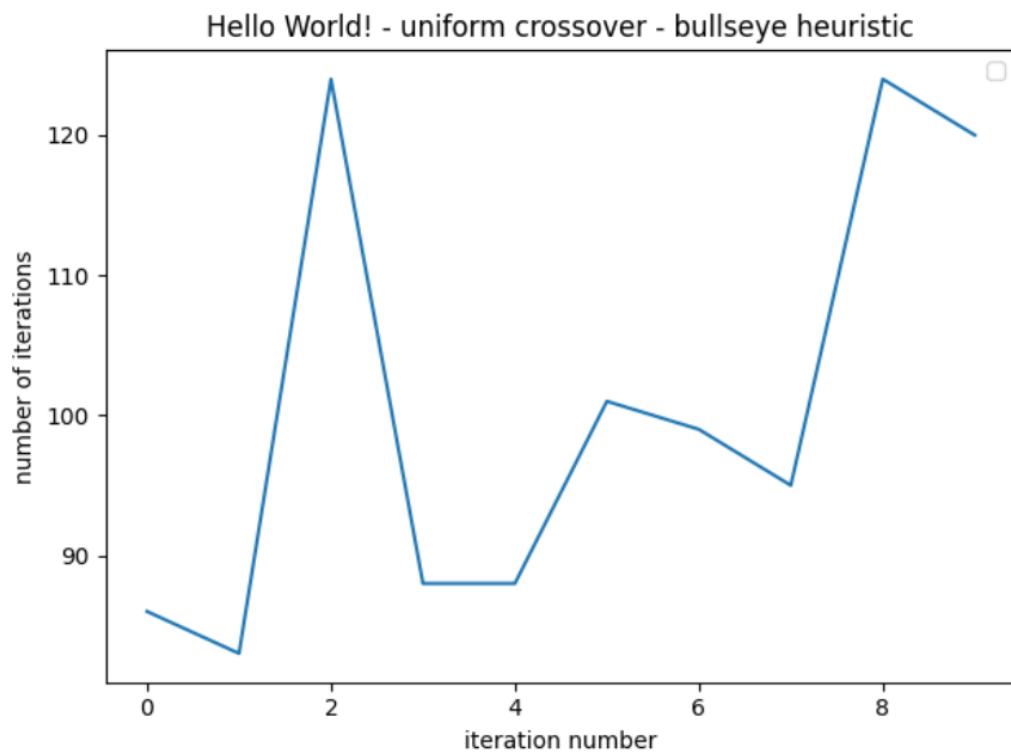
עובד היוריסטיקה "פול-פגיעה" :TWO-POINT



עובד היוריסטיקה מקורית UNIFORM:



עובד היוריסטיקה "פול-פגעה" UNIFORM:



יכולים לראות את ההבדל הגדול במספר האיטרציות עבור כל היוריסטיקה, כך שהיוריסטיקת "בול פגעה" צריכה הרבה פחות איטרציות בכל פעם עד שתמצא את הפתרון. (אנחנו יכולים לראות את זה בצייר (ז)

## 6. ציינו ונמקו אלו חלקים באלגוריתם אחראים ל EXPLORATION ואילו ?EXLOITATION

### EXPLORATION:

לפי מה שמצאנו, סוג ה- **Crossover** מואוד משפיע על SINGLE POINT/EXPLORATION/ כך למשל **SINGLE POINT** היא סוג של EXPLORATION מכיוון שהיא מייצרת בין חדש, מה שאומר שאנו מנסים לחפש מחירות חדשה. באותו אופן עבור שאר הסוגים שהם יותר מתוחכמים על ידם אנחנו מקבל בנים שהם יותר רחוקים מההורים שלהם מה שאומר שמחפשים משהו חדש. **mutate**: ה מכיוון שבפונקציה זו אנו מנסיםתו במחירות שמקבלים, זה אומר שהפונקציה מנסה לחפש מחירות גם כן חדשה.

### EXPLOITATION:

שלב ב- ח: **elitizel** מכיוון שבשלב הזה אנחנו שומרים את X% היכי טובים, אז אנחנו מנסים להתמקד במה שיש בידינו, מה שאומר שאנו נוטים להישאר בסביבה שאנו נמצא בה.

## 7. מימוש לבעה הנתונה את האלגוריתם האבולוציוני PSO שנלמד בקורס המבוא

מצורפת תמונה לקטע קוד שישיר לפונקציה האלגוריתם:

```
def run(self):
    target = [ord(char) for char in GA_TARGET]
    print(target)
    for t in range(GA_MAXITER):

        for particle in self.particles:
            particle.update(self.globalBest, self.C1, self.C2, self.W)
            fitness = self.calcFitness(particle.getString())
            particle.setFitness(fitness)
            self.updateGPBest(particle, fitness)
        self.sortByFitness()
        self.updateParameters(t, GA_MAXITER)
        if self.globalBestFitness == 0:
            print('the best is', self.globalBest, '(', self.globalBestFitness, ')')
            print('found')
            break
        print('the best is', self.globalBest, '(', self.globalBestFitness, ')')
```

הפונקציה ה兹ת היא לב האלגוריתם, מצורף עוד תמונה שמשלימות את הטכניקה, וגם תמונה לפסאודו קוד איך מימשנו אותה.

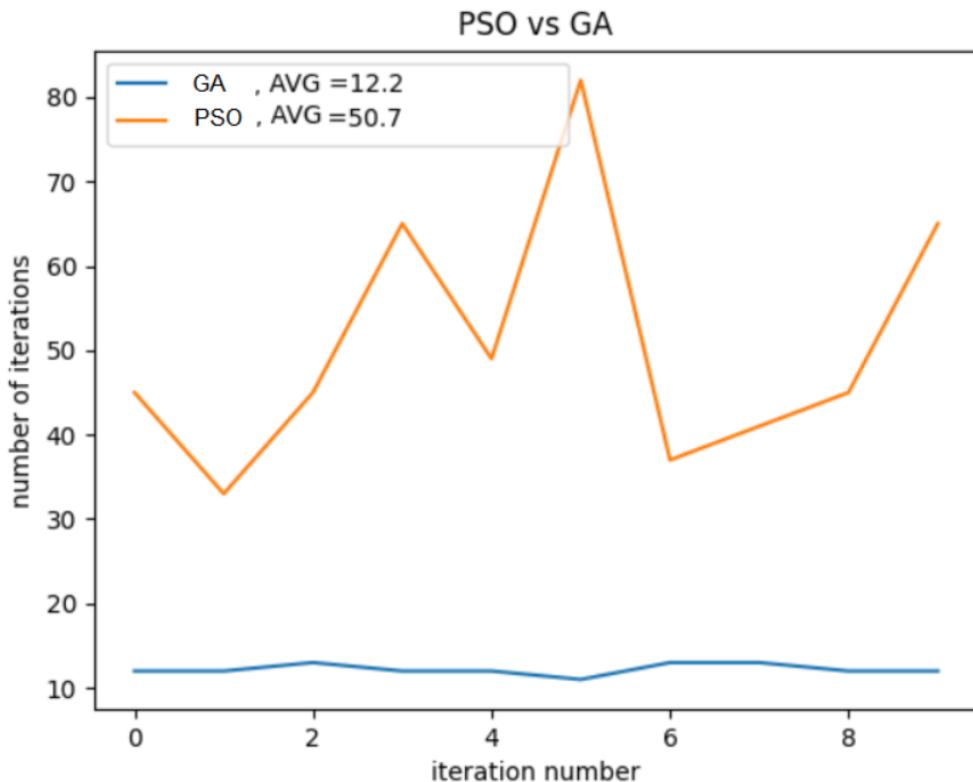
```
while a termination criterion is not met do:
    for each particle  $i = 1, \dots, S$  do
        for each dimension  $d = 1, \dots, n$  do
            Pick random numbers:  $r_p, r_g \sim U(0,1)$ 
            Update the particle's velocity:  $\mathbf{v}_{i,d} \leftarrow \omega \mathbf{v}_{i,d} + \varphi_p r_p (\mathbf{p}_{i,d} - \mathbf{x}_{i,d}) + \varphi_g r_g (\mathbf{g}_d - \mathbf{x}_{i,d})$ 
            Update the particle's position:  $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$ 
            if  $f(\mathbf{x}_i) < f(\mathbf{p}_i)$  then
                Update the particle's best known position:  $\mathbf{p}_i \leftarrow \mathbf{x}_i$ 
            if  $f(\mathbf{p}_i) < f(\mathbf{g})$  then
                Update the swarm's best known position:  $\mathbf{g} \leftarrow \mathbf{p}_i$ 

def updateGPBest(self, particle, fitness):
    if particle.getBestFitness() > fitness:
        particle.setBestString(particle.getString())
        particle.setBestFitness(fitness)
    if self.globalBestFitness > fitness:
        self.globalBest = particle.getString()
        self.globalBestFitness = fitness

def updateParameters(self, t, N):
    self.W = 0.4 * ((t - N) / N ** 2) + 0.4
    self.C1 = -3 * (t / N) + 3.5
    self.C2 = 3 * (t / N) + 0.5
```

8. השוו בסימולציה בין ביצועי שני האלגוריתמים לגבי הבעה הנתונה תחת  
ההיוריסטיקה המעודפת והפרמטריזציה המיטבית – התייחסו לאיכות  
פתרונותות ולמהירות ההגעה אליהם

אנחנו נבחר בהיוריסטיקה השנייה "פול-פגעה" מכיוון שהוא יותר טובה כפי שראינו לעילו +  
TOW-POINT crossover, נctrף מספר גרפים להשווא:



כמו שאנו

ראים בגרף לעיל, יש הבדל עצום במספר האיטרציות בין שני האלגוריתמים, מה שאומר שהאלגוריתם הגנטי מועדף על אלגוריתם PSO. ויכולם לראות שמספר האיטרציות קטן יותר גינתי, אומר שהוא יותר מהיר. אך האלגוריתמים הגינטי יותר יעל.

## חלק ב:

1. הוסיפו למנוע תמיכה במשיטות הבחירה שונות: ,SUS , RWS + scaling וטורנייר (דטרמיניסטי והסתברותי) עם פרמטר K

```
SUS(Population, N)
    F := total fitness of Population
    N := number of offspring to keep
    P := distance between the pointers (F/N)
    Start := random number between 0 and P
    Pointers := [Start + i*P | i in [0..(N-1)]]
    return RWS(Population,Pointers)

RWS(Population, Points)
    Keep = []
    for P in Points
        i := 0
        while fitness sum of Population[0..i] < P
            i++
        add Population[i] to Keep
    return Keep
```

זה פסאודו קוד לשני האלגוריתמים כך שאחד תלוי בשני, מכיוון שימוש RWS בלבד מחייב רק מועד אחד, אך מישנו אותו להיות גמיש להציג מועדים לפי ה- POINTS כדי להימנע משכפול. יתר על כן, מישנו אותו ע"י חיפוש בינארי למערך שמכיל בתא ה- *i* את סכום ה FITNESS מאפס עד *i* כדי לחסוך בזמן.

```
def RWS(problem, population, scaledFitness):  
    arr = [i for i in range(len(population))]  
    index = choice(arr, p=scaledFitness)  
    return population[index]  
  
def SUS(problem, population, scaledFitnesses):  
    indexArr = [i for i in range(int(len(population) / 2))]  
    scaled = []  
    sum = 0  
    for i in range(int(len(population) / 2)):  
        scaled.append(scaledFitnesses[i])  
        sum += scaledFitnesses[i]  
  
    for i in range(int(len(population) / 2)):  
        scaled[i] = float(scaled[i] / sum)  
  
    index1 = choice(indexArr, p=scaled)  
    index2 = 2 * index1  
    members = [population[index1], population[index2]]  
    return members
```

#### לגביה הטורניר:

מימושו אותו בדיק לפי הרצאה: בוחרים K מועמדים לטורניר ומנצח המועמד הכי טוב (שמבחןינו עם FITNESS אפס) וכך לחסוך בזמן ולא לחזור על פעולות מיותרות, אנחנו עשינו את זה תוך כדי בחירת המועמדים, ככלומר המועמד הראשון מבחרינו הוא הכי טוב, ואם מצאנו מועמד יותר טוב, אז נמשיך עמו.

```
def tournamentSelection(population):  
    best = None  
    for _ in range(TOURNAMENT_K):  
        person = population[randint(0, len(population) - 1)]  
        if best is None:  
            best = person  
        elif best.getFitness() > person.getFitness():  
            best = person  
    return best
```

#### לגביה scaling:

עשינו אותו לפי הרצאה:  $f = a*f + b$   
כך שזה יהיה יותר כללי, שמננו את B קבועים, ואת זה רואים בתמונה מתוחת בסעיף 2.

```
def scaleFitness(problem): # scales the fitness and uses it to make a probability vector
    scaled = []
    agePopulation = []
    sum = 0
    for person in problem.population:
        if not person.isGoodForMating():
            continue
        num = (problem.args.LINEAR_SCALING_A * person.getFitness()) + problem.args.LINEAR_SCALING_B
        num = float(1 / num)
        sum += num
        agePopulation.append(person)
        scaled.append(num)
    for i in range(len(scaled)):
        scaled[i] = float(scaled[i] / sum)
    return agePopulation, scaled, sum
```

בפונקציה הזאת אנחנו נחזיר מערך חדש ממנו משתמשים בו לפונקציית RWS.

## 2. הוסיפו שיטת שרידות נוספת לוגינית

```
class GAstruct:
    def __init__(self, string, fitness):
        self.string = string
        self.fitness = fitness
        self.age = randint(AGE_MIN, AGE_MAX)

    def getString(self):
        return self.string

    def getFitness(self):
        return self.fitness
```

```
3 # GA_POPSIZE = 2048
4 GA_POPSIZE = 100
5 GA_MAXITER = 16384
6 GA_ELITRATE = 0.10
7 GA_MUTATIONRATE = 0.25
8 GA_MUTATION = random.random
9 GA_TARGET = 'Hello World!'
10 TOURNAMENT_K = 10
11 AGE_MIN = 2
12 AGE_MAX = 30
13 NQUEENS = 15
14 LINEAR_SCALING_B = 10
15 LINEAR_SCALING_A = 1
16
```

הוסףנו למחלקה את הגיל, וקבענו DEFINE את התחום שבו המחרוזת חייה (יכולת להשתתף בזיווג).  
בכל דור, אנחנו מגדילים את הגיל.

```
def updateAge(self):
    self.age += 1
```

3. הוסיפו תמיינה למופע חדש של בעיה – בעית  $N$  המלכות על לוח שחמט  
– לצורך כך ממשו ייצוג מתאים לגן באורך  $N$ .

```
def initPopulation():
    tsize = NQUEENS
    population = []
    nextPopulation = []
    for _ in range(GA_POPSIZE):
        randomStr1 = [i for i in range(0, tsize)]
        randomStr2 = [i for i in range(0, tsize)]

        shuffle(randomStr1)
        shuffle(randomStr2)

        member1 = GAstruct(randomStr1, 0)
        member2 = GAstruct(randomStr2, 0)
        population.append(member1)
        nextPopulation.append(member2)

    return population, nextPopulation
```

מיצרים מערך באורך N המכיל את N המלכות, ומעריבבים אותם.  
בגלל שביעיות N-המלכות היא דומה מאוד לביעיות אחרות, אז יש להם משווה משותף שהוא RUN  
,NI-המלכות שונה מהם בפונקציית ה MATE ,FUNCTION ,LCN מצרך תמורה לה.

```
def run(problem):
    found = True
    startTime = time.time()
    problem.initPopulation()
    for _ in range(GA_MAXITER):
        iterTime = time.time()
        problem.calcFitness()
        problem.sortByFitness()
        problem.printBest()
        problem.calcAvgSd()

        if problem.population[0].getFitness() == 0:
            print('Generation time: ', time.time() - iterTime)
            print()
            break

    try:
        problem.mate()
    except:
        print('Generation time: ', time.time() - iterTime)
        found = False
        break
    problem.swap()
    problem.aging()
    print('Generation time: ', time.time() - iterTime)
    print()
    print('Time elapsed: ', time.time() - startTime)
    if not found:
        print('Solution not found. Reached local minimum.')


```

```
def mate(self):
    self.elitism()

    if self.crossoverType == CrossoverType.PMX:
        self.PMX()
    else:
        self.OX()
```

**דוגמת הרצה:**

```
C:\Users\Profhack\AppData\Local\Programs\Python\Python39\python.exe C:/Users/Profhack/Desktop/AI_Labi/lab1/main.py
Best: [2, 7, 11, 1, 8, 6, 9, 3, 5, 0, 4, 10] (1) fitness data: avarage: 7.60693359375 || standard deviation: 2.7493413223038914
Generation time: 0.17357158660888672

Best: [7, 11, 6, 8, 2, 0, 9, 4, 10, 1, 3, 5] (1) fitness data: avarage: 15.10693359375 || standard deviation: 11.444520557072279
Generation time: 0.24224090576171875

Best: [7, 11, 6, 8, 2, 0, 9, 4, 10, 1, 3, 5] (1) fitness data: avarage: 7.16943359375 || standard deviation: 3.0788947754931413
Generation time: 0.1898801326751709

Best: [7, 11, 6, 8, 2, 0, 9, 4, 10, 1, 3, 5] (1) fitness data: avarage: 5.6943359375 || standard deviation: 2.322471564604397
Generation time: 0.1721334457397461

Best: [7, 11, 6, 8, 2, 0, 9, 4, 10, 1, 3, 5] (1) fitness data: avarage: 5.72900390625 || standard deviation: 2.404483642265474
Generation time: 0.18347525596618652

Best: [7, 11, 6, 8, 2, 0, 9, 4, 10, 1, 3, 5] (1) fitness data: avarage: 5.49755859375 || standard deviation: 2.3756154417404183
Generation time: 0.16084027290344238

Best: [6, 3, 1, 8, 11, 2, 10, 5, 9, 4, 0, 7] (0) fitness data: avarage: 5.376953125 || standard deviation: 2.3812615021355246
Generation time: 0.06816363334655762
found

Time elapsed: 1.2144582271575928
```

.4. מימוש 2 אופרטורי שיחולף ו 2 אופרטורי מותציה חלופיים לתרומות.

את השיטות שבחרנו:

**שיטת שיחולוף מס' 1 : PMX – Partially Matched crossover**

Pick an arbitrary position in two parent permutations:

8	2	4	3	7	5	1	0	9	6
4	1	7	6	2	8	3	9	5	0

That choice means to interchange 5 with 8 in both parents.

5	2	4	3	7	8	1	0	9	6
4	1	7	6	2	5	3	9	8	0

Perform this operation several times, creating children with characteristics of both parents.

```
def PMX(population, nextPopulation, mutationType, selectionType):
    esize = int(GA_ELITRATE * GA_POPSIZE)
    agePopulation, scaledFitness, sumScaledFitness = scaleFitness(population)

    for i in range(esize, GA_POPSIZE):
        pmx = randint(0, NQUEENS - 1)
        person1, person2 = getCouple(agePopulation, scaledFitness, sumScaledFitness, selectionType)

        father = person1.getString()[0:]
        mother = person2.getString()[0:]

        num1 = father[pmx]
        num2 = mother[pmx]

        for j in range(len(father)):
            if father[i] == num2:
                father[i], father[pmx] = num1, father[i]
                break
        for j in range(len(mother)):
            if mother[i] == num1:
                mother[i], mother[pmx] = num2, mother[i]
                break
        string = calcStringFitness(father, mother)
        member = GAstruct(string, collosionFitness(string))
        member.zeroAge()

        if random() < GA_MUTATION:
            mutate(member, mutationType)
        nextPopulation.append(member)
```

במימוש הפונקציה: מגרילים שני אינדקסים [חושב לדעת שהם חיבים להיות צעירים כלומר גilm חוקי, זה מה שבודקת הפונקציה `getCouple`, ואז מגרילים את האינדקס המרכזית שאותו אנחנו רוצים לעבוד, ובודקים בהתאם ומחליפים].

## שיטת שיחולף מס' 2 :OX – Ordered crossover :

Pick about half of the elements of the first parent, (here, we choose 2, 4, 5, 1, and 6) and copy them to the child, preserving the positions.

Choose the remaining values (0, 3, 7, 8, and 9) from the second parent, and copy them to the child, preserving the order.

8	2	4	3	7	5	1	0	9	6
4	1	7	6	2	8	3	9	5	0
7	2	4	8	3	5	1	9	0	6

This preserves the some orderings of elements in both parents and position of some in the first parent.

```
def OX(population, nextPopulation, mutationType, selectionType):
    esize = int(GA_ELITRATE * GA_POPSIZE)
    agePopulation, scaledFitness, sumScaledFitness = scaleFitness(population)
    for i in range(esize, GA_POPSIZE):
        numbers_father = [i for i in range(0, NQUEENS - 1)]
        maxIndex = len(numbers_father) - 1
        index_father = []
        index_mother = []
        father1, mother1 = getCouple(agePopulation, scaledFitness, sumScaledFitness, selectionType)
        father = father1.getString()
        mother = mother1.getString()
        for _ in range(int(NQUEENS / 2)): # choose which indexes we want to take from father
            randII = randint(0, maxIndex)
            index_father.append(numbers_father[randII])
            numbers_father[randII], numbers_father[maxIndex] = numbers_father[maxIndex], numbers_father[randII]
            maxIndex -= 1

        index_father.sort()
        NonDuplicat_index = []

        for i in range(len(index_father)):
            NonDuplicat_index.append(father[index_father[i]])
        NonDuplicat_index.sort()

        k1 = 0
        for i in range(int(NQUEENS)):
            if k1 < len(NonDuplicat_index) and i == NonDuplicat_index[k1]:
                k1 += 1
            else:
                for l in range(len(mother)):
                    if mother[l] == i:
                        index_mother.append(l)
                        break
        index_mother.sort()
        child = []
        k1 = 0
        k2 = 0

        for i in range(NQUEENS):
            if k1 < len(index_father) and i == index_father[k1]:
                child.append(father[i])
                k1 += 1
            else:
                child.append(mother[index_mother[k2]])
                k2 += 1

        member = GAstruct(child, collosionFitness(child))
        member.zeroAge()

        if random() < GA_MUTATION:
            mutate(member, mutationType)
        nextPopulation.append(member)
```

הकשיי בפונקציה הוא שחייבים לדאוג לכך שהחיתוך המספרים שניקח מהאב עם המספרים שניקח מהאם הוא קבוצה ריקה "Ø" כדי לשמור על הבין להיות פרמוטציה ל- N המלכות. בלולאה השנייה מוצאים המספרים שאנו רוצים להימנע מלקיים מהאם, ובלולאה השלישי משלימים את המספרים שכן אפשר לקחת מהאם, כדי לשמור על פרמוטציה.

המוטציות שהשתמשו הפעם:

**2. מוטציה החלפה - exchange (swap) mutation**

**Example: (1 2 3 4 5 6 7 8 9)**

**3rd and 5th selected randomly**

**new tour becomes (1 2 5 4 3 6 7 8 9)**

```
def exchangeMutation(person):
    string = person.getString()
    i1 = randint(0, NQUEENS - 1)
    i2 = randint(0, NQUEENS - 1)
    string[i1], string[i2] = string[i2], string[i1]
    person.setString(string)
```

במימוש הפקציה אנחנו מגרילים שני אינדקסים, ומחליפים ביניהם

## 4. מוטציה היפוך פשוטה - simple inversion mutation

1

**Example:** (1 2 3 | 4 5 6 7 | 8 9)  
new tour becomes (1 2 3 7 6 5 4 8 9)

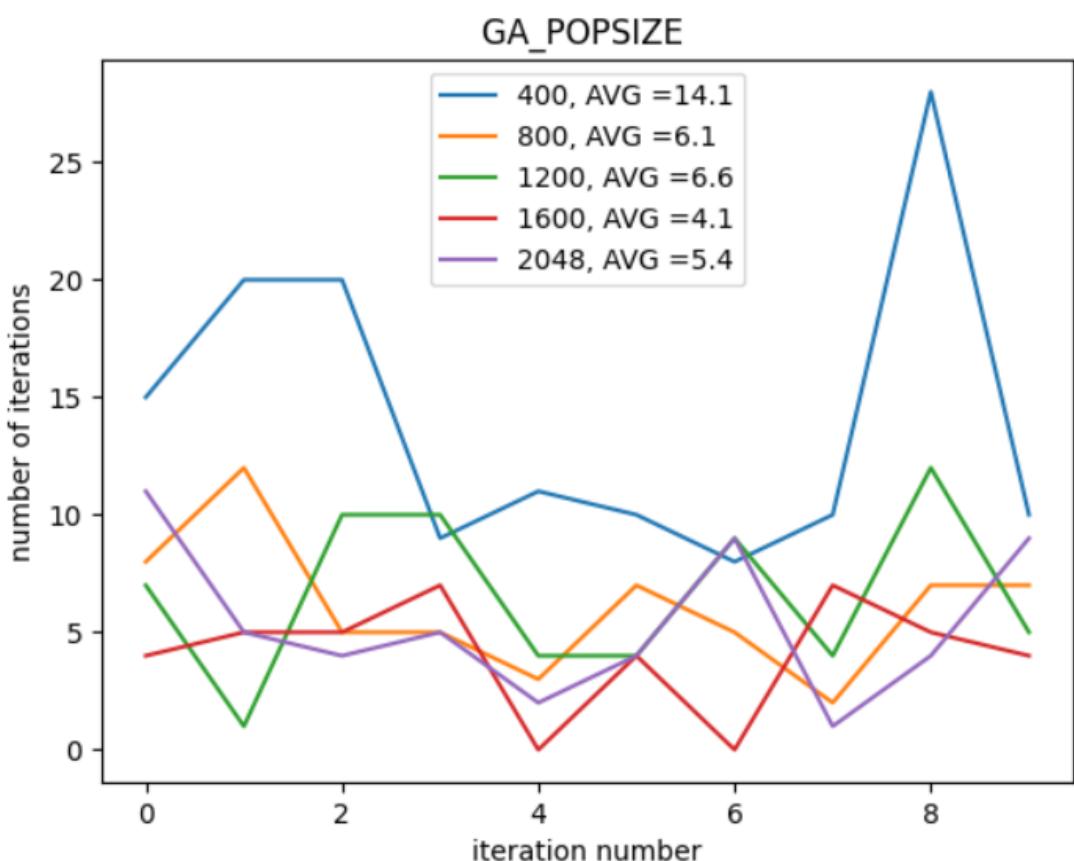
```
def simpleInversionMutation(person):
    string = person.getString()
    i1 = randint(0, NQUEENS - 1)
    i2 = randint(0, NQUEENS - 1)
    if i1 > i2:
        i1, i2 = i2, i1
    while i1 < i2:
        string[i1], string[i2] = string[i2], string[i1]
        i1 += 1
        i2 -= 1
    person.setString(string)
```

במימוש הפונקציה, אנחנו מגFAILים שני אינדקסים שמייצגים לנו את התחלת וסיום תת המחרוזת שרצויים להפוך, אז הופכים אותה.

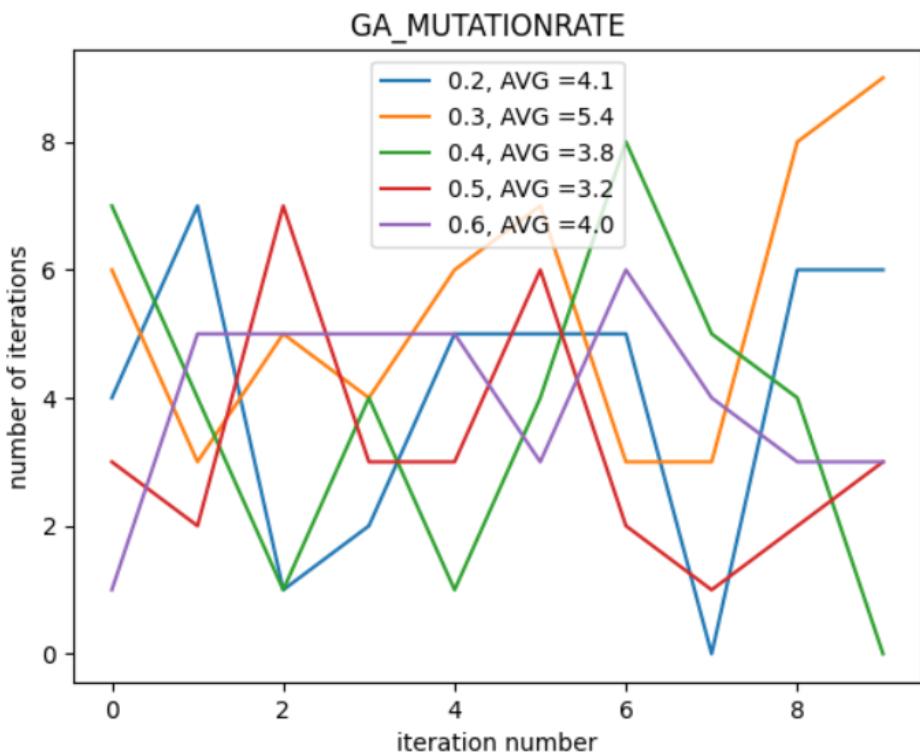
5. בדקו באמצעות סימולציות את רגישות פתרון שתי הביעות  $N$  המלכות ("בול פגעה") לפי הקритריונים של מהירות התכנסות, איקות הפתרון **זמן ריצה עפ"י הפרמטרים הבאים:**

עבור כל אחת מהבדיקות, השתמשו במשתנים הקבועים שקיבלו בתחילת המשימה ושינו בהתאם וهم:  
GA\_POPSIZE = 2048  
GA\_MAXITER = 16384  
GA\_ELITRATE = 0.10  
GA\_MUTATIONRATE = 0.25  
CrossoverType.OX  
SelectionType.RWS

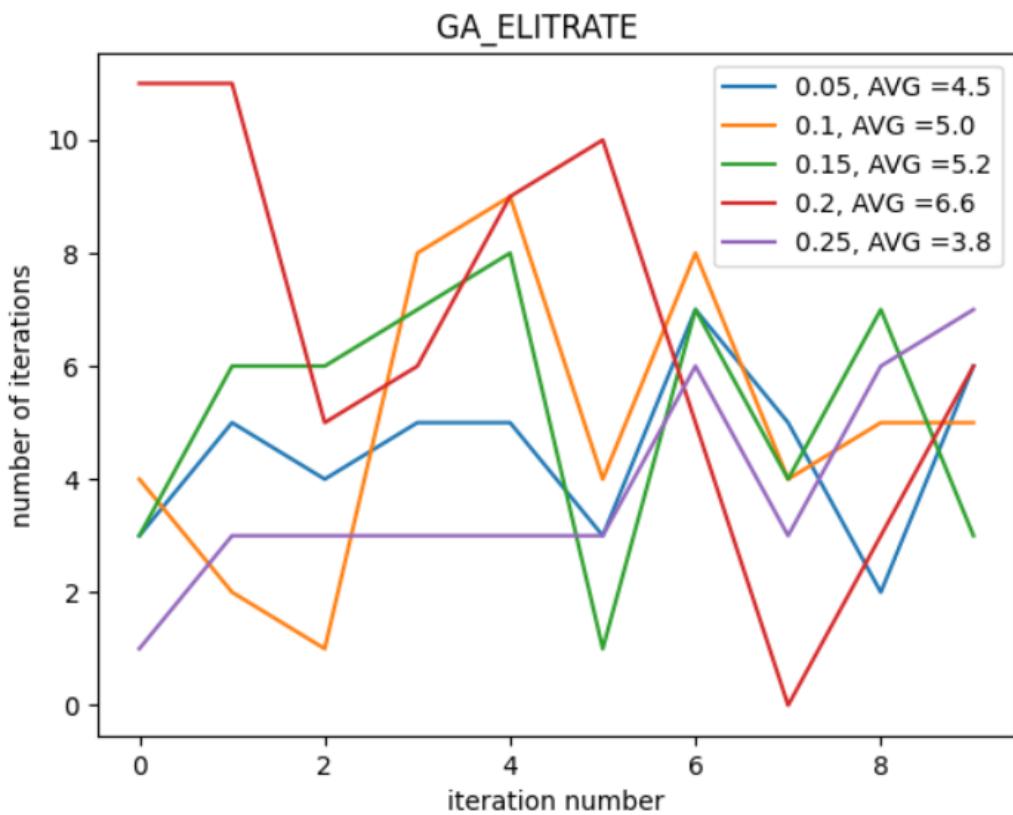
לגודל האוכלוסייה:



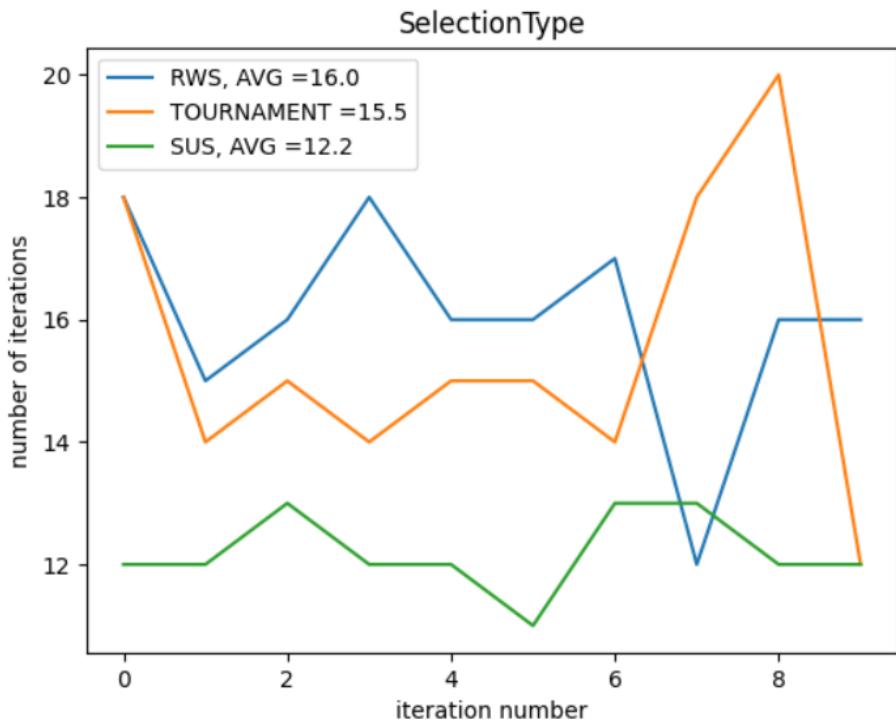
**הסתברות למוטציות:**



**לפרופורציה האוכלוסייה האליטיסטית :**

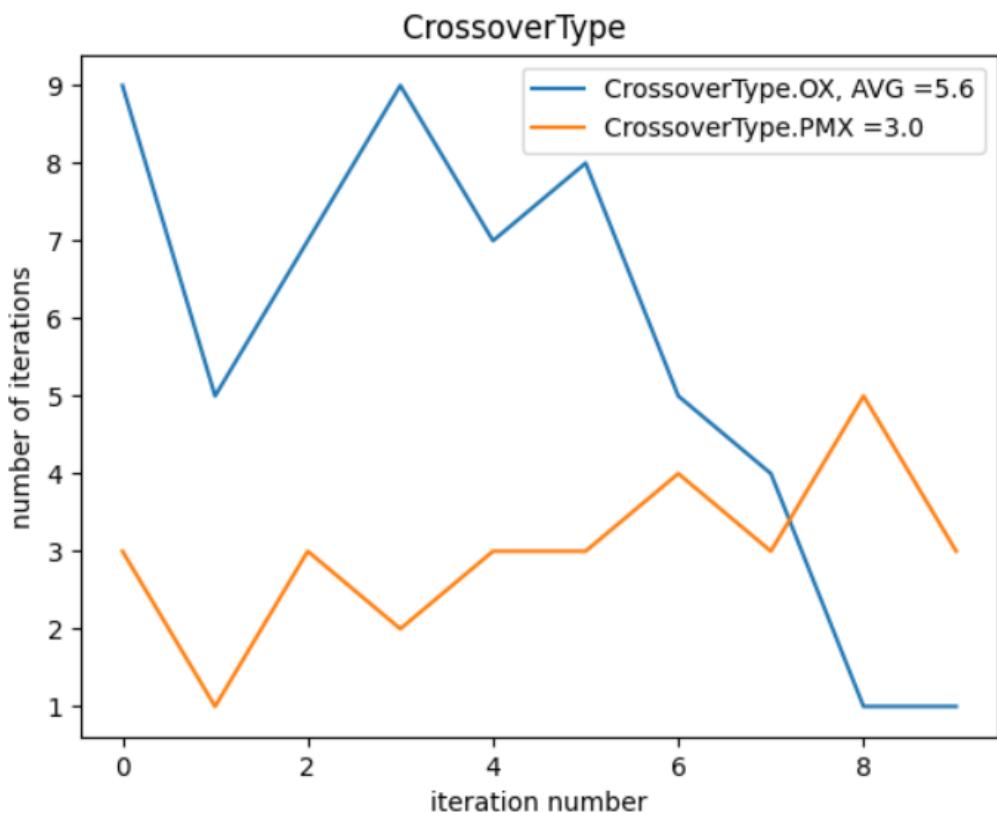


**אסטרטגיית הבחירה:**



**לאסטרטגיית השרידות:** אין לו להדגים עבוד הגיל, פשוט קיבלנו שעד גיל 50-2 זה הכי טוב.

**לאסטרטגיית השילוף:**



## 6. בהתאם לממצאי הסעיף הקודם בחרו את סט הפרמטרים האופטימלי עבור האלגוריתם שלכם והשתמשו בו לצורך פתרון בעיות הבאות

בהתאם לממצאי הסעיף הקודם סט הפרמטרים האופטימלי עבור האלגוריתם שלכם הם:

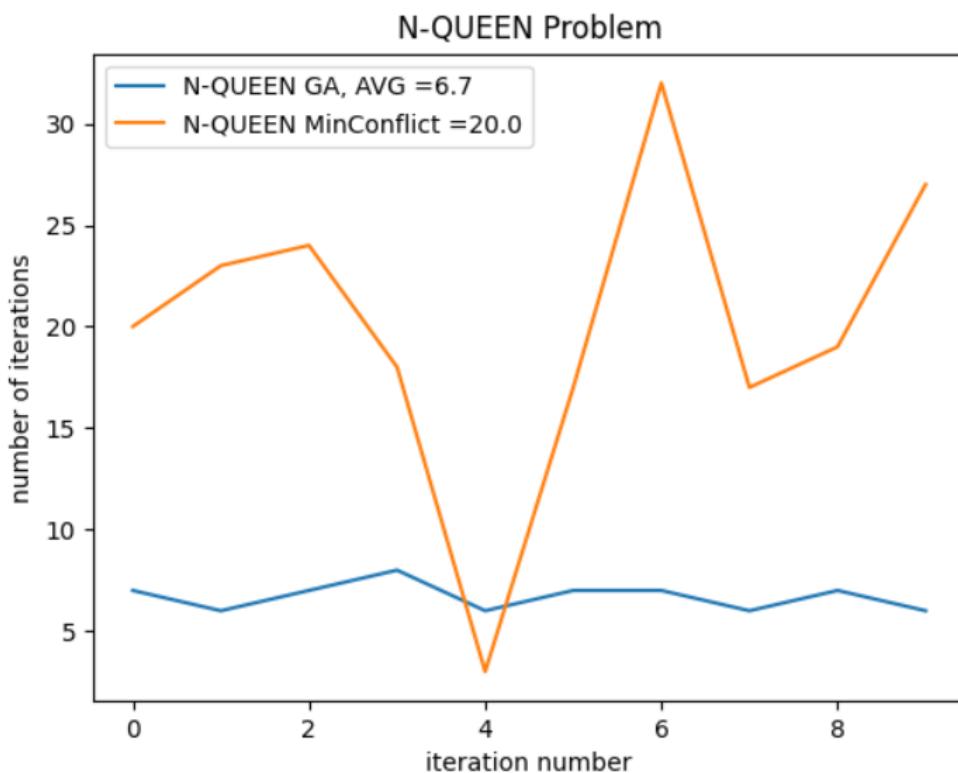
GA\_POPSIZE = 1600  
GA\_MAXITER = 16384  
GA\_ELITRATE = 0.10  
GA MUTATION RATE = 0.5  
CrossoverType.PMX  
SelectionType.SUS  
N-QUEEN = 15

והם ימשכו אותנו לסעיפים הבאים.

## 7. השוו את ביצועים האלגוריתם הגרפי עם הפרמטריזציה המיטבית מול אלגוריתם MINIMAL CONFLICTS עבור בעית N המלכות מבחינה מהירות ואיכות הפתרון. הסבירו כיצד ניתן להקליא בין שני האלגוריתמים?

מצורף חלק מקוד שמיימשנו עבור בעית MIN CONFLICTS

```
def run(self):
    startTime = time.time()
    for _ in range(GA_MAXITER):
        iterTime = time.time()
        print(self.board)
        max_conflict, index = self.getMaxConflictState()
        if max_conflict == 0:
            print('found')
            print('Generation time: ', time.time() - iterTime)
            print()
            break
        self.getNewPlace(index, max_conflict)
        print('Generation time: ', time.time() - iterTime)
        print()
    print('Time elapsed: ', time.time() - startTime)
```



לפי הגרף שאנו רואים, אלגוריתם פתרון בעיית N-QUEEN עבור האלגוריתם הגנטי יותר מהיר, יותרiesel ויותר איקוטי. חשוב לציין שאלגוריתם MIN CONFLICT מוצר ברוב האיטרציות אחריו כניסה ל local optimum, כלומר ברוב המקרים הוא לא מגע לפתרון אופטימלי.

8. קדדו והציגו כיצד ניתן להשתמש באלגוריתם הגנטי ב כדי לפתור את בעיית השק 0-1:

בעיית מקסימיזציה מהצורה הבאה – בנוסף יש שאותו הוא שואף למלא  $\sum_{i=1}^N P_i X_i$  ב  $\leq C$   $\sum_{i=1}^N W_i X_i$  עליך למצוא אילו פריטים  $X_i$  כדאי לו לחת (אחד בלבד מכל סוג) כדי למקסם את ערך השק מבלי לחרוג מńפח השק  $C$

$$\begin{aligned} & \text{maximize} \sum_{i=1}^N P_i X_i \\ & \text{s.t. } \sum_{i=1}^N W_i X_i \leq C, X_i \in \{0, 1\} \end{aligned}$$

הristol את האלגוריתם על 8 הבעיה המצוויות בקישור הבא והשו את התוצאות שקיבלתם עם הפתרון האופטימלי המופיע בצדן

[https://people.sc.fsu.edu/~jburkardt/datasets/knapsack\\_01/knapsack\\_01.html](https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html)

מצורף הריצות 8 הבעיה, וגם פונקציית חישוב הפיטנס, לא חושבים שציר להוסיף עוד פונקציות, כל המימוש מופיע בקובץ SOURCE.

```
def priceFitness(self, string):
    price = 0
    weight = 0
    for j in range(self.tsize):
        price += self.profits[j] * string[j]
        weight += self.weights[j] * string[j]

    if weight > self.capacity:
        price = 0

    return abs(self.maxProfit - price)
```

חשב לנו לצין תנאי העצירה הוא אם נשארנו באותו מצב 10 פעמים.

ו.מ טנוו: 207866328  
פארוק כריימן: 314656869

**בעיה 1:**

```
Best: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0] ( 370 ) fitness data: avarage: 374.134 || standard deviation: 33.99479436619672
Generation time: 0.23323774337768555

Best: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0] ( 370 ) fitness data: avarage: 374.954 || standard deviation: 36.80677769107206
Generation time: 0.1363050937652588

Best: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0] ( 370 ) fitness data: avarage: 374.385 || standard deviation: 34.26394278246448
Generation time: 0.13917160034179688

Best: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0] ( 370 ) fitness data: avarage: 375.2675 || standard deviation: 38.48645143099062
Generation time: 0.1600649356842041

Best: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0] ( 370 ) fitness data: avarage: 374.5075 || standard deviation: 35.379626110941295
Generation time: 0.1626415252685547

Best: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0] ( 370 ) fitness data: avarage: 373.703 || standard deviation: 30.1481805587004
Generation time: 0.1685481071472168

Best: [1, 1, 1, 1, 0, 1, 0, 0, 0, 0] ( 370 ) fitness data: avarage: 374.1195 || standard deviation: 32.20060899657024
Generation time: 0.011968612670898438

Time elapsed: 2.689363956451416
solution price: 309
solution weight: 165
optimal price: 309
optimal weight: 165

Process finished with exit code 0
```

**בעיה 2:**

```
Best: [0, 1, 1, 1, 0] ( 40 ) fitness data: avarage: 45.1515 || standard deviation: 13.046821365758065
Generation time: 0.10297393798828125

Best: [0, 1, 1, 1, 0] ( 40 ) fitness data: avarage: 45.3555 || standard deviation: 13.791632236613614
Generation time: 0.10154986381530762

Best: [0, 1, 1, 1, 0] ( 40 ) fitness data: avarage: 45.36 || standard deviation: 13.537665973128412
Generation time: 0.07827973365783691

Best: [0, 1, 1, 1, 0] ( 40 ) fitness data: avarage: 45.797 || standard deviation: 14.207912971298818
Generation time: 0.10152506828308105

Best: [0, 1, 1, 1, 0] ( 40 ) fitness data: avarage: 45.806 || standard deviation: 13.930447372572056
Generation time: 0.08876442909240723

Best: [0, 1, 1, 1, 0] ( 40 ) fitness data: avarage: 45.853 || standard deviation: 14.291339720264189
Generation time: 0.08842134475708008

Best: [0, 1, 1, 1, 0] ( 40 ) fitness data: avarage: 45.912 || standard deviation: 14.082444958173946
Generation time: 0.011988401412963867

Time elapsed: 1.6161425113677979
solution price: 51
solution weight: 26
optimal price: 51
optimal weight: 26

Process finished with exit code 0
```

**בעיה 3:**

```
Best: [1, 1, 0, 0, 1, 0] ( 115 ) fitness data: avarage: 129.475 || standard deviation: 40.487335982995944
Generation time: 0.0967400074005127

Best: [1, 1, 0, 0, 1, 0] ( 115 ) fitness data: avarage: 129.2 || standard deviation: 40.13552042767116
Generation time: 0.10272502899169922

Best: [1, 1, 0, 0, 1, 0] ( 115 ) fitness data: avarage: 129.375 || standard deviation: 40.18220221690195
Generation time: 0.08786892890930176

Best: [1, 1, 0, 0, 1, 0] ( 115 ) fitness data: avarage: 128.35 || standard deviation: 38.84942084510397
Generation time: 0.11819052696228027

Best: [1, 1, 0, 0, 1, 0] ( 115 ) fitness data: avarage: 129.125 || standard deviation: 39.26492550610532
Generation time: 0.1077122688293457

Best: [1, 1, 0, 0, 1, 0] ( 115 ) fitness data: avarage: 129.3 || standard deviation: 40.068815804812616
Generation time: 0.12324666976928711

Best: [1, 1, 0, 0, 1, 0] ( 115 ) fitness data: avarage: 128.775 || standard deviation: 39.070441192799336
Generation time: 0.010042190551757812

Time elapsed: 1.6957976818084717
solution price: 150
solution weight: 190
optimal price: 150
optimal weight: 190

Process finished with exit code 0
```

#### בעיה 4:

```
Best: [1, 0, 0, 1, 0, 0, 0] ( 81 ) fitness data: avarage: 81.7325 || standard deviation: 8.370838891652378
Generation time: 0.121644735333630371

Best: [1, 0, 0, 1, 0, 0, 0] ( 81 ) fitness data: avarage: 81.8395 || standard deviation: 9.019519929020814
Generation time: 0.09191656112670898

Best: [1, 0, 0, 1, 0, 0, 0] ( 81 ) fitness data: avarage: 81.854 || standard deviation: 9.174621736071737
Generation time: 0.10653233528137207

Best: [1, 0, 0, 1, 0, 0, 0] ( 81 ) fitness data: avarage: 81.5535 || standard deviation: 7.5909905644783935
Generation time: 0.1122586727142334

Best: [1, 0, 0, 1, 0, 0, 0] ( 81 ) fitness data: avarage: 81.644 || standard deviation: 8.037926598321237
Generation time: 0.1047208309173584

Best: [1, 0, 0, 1, 0, 0, 0] ( 81 ) fitness data: avarage: 81.644 || standard deviation: 7.875167553773064
Generation time: 0.10323476791381836

Best: [1, 0, 0, 1, 0, 0, 0] ( 81 ) fitness data: avarage: 81.5165 || standard deviation: 7.328282728579731
Generation time: 0.011968135833740234

Time elapsed: 1.9985814094543457
solution price: 107
solution weight: 50
optimal price: 107
optimal weight: 50

Process finished with exit code 0
```

ו.מ. טנווילס: 207866328  
פארוק כריימן: 314656869

### בעיה 5:

```
Best: [1, 0, 1, 1, 0, 1, 1] ( 408 ) fitness data: avarage: 510.9125 || standard deviation: 252.29675947928905
Generation time: 0.11246109008789062

Best: [1, 0, 1, 1, 0, 1, 1] ( 408 ) fitness data: avarage: 495.4975 || standard deviation: 235.67482893544178
Generation time: 0.11471438407897949

Best: [1, 0, 1, 1, 0, 1, 1] ( 408 ) fitness data: avarage: 499.725 || standard deviation: 239.21010508546698
Generation time: 0.11367249488830566

Best: [1, 0, 1, 1, 0, 1, 1] ( 408 ) fitness data: avarage: 503.56 || standard deviation: 245.06730993749446
Generation time: 0.12914776802062988

Best: [1, 0, 1, 1, 0, 1, 1] ( 408 ) fitness data: avarage: 510.7825 || standard deviation: 255.92707983671772
Generation time: 0.13164806365966797

Best: [1, 0, 1, 1, 0, 1, 1] ( 408 ) fitness data: avarage: 508.9575 || standard deviation: 253.39198032643083
Generation time: 0.14364218711853027

Best: [1, 0, 1, 1, 0, 1, 1] ( 408 ) fitness data: avarage: 503.38 || standard deviation: 246.21333351384547
Generation time: 0.013935565948486328

Time elapsed: 1.9734983444213867
solution price: 900
solution weight: 104
optimal price: 900
optimal weight: 104

Process finished with exit code 0
```

### בעיה 6:

```
Best: [0, 1, 0, 1, 0, 0, 1] ( 2063 ) fitness data: avarage: 2295.3235 || standard deviation: 550.729965452897
Generation time: 0.11228752136230469

Best: [0, 1, 0, 1, 0, 0, 1] ( 2063 ) fitness data: avarage: 2326.6205 || standard deviation: 578.5608779374488
Generation time: 0.10573816299438477

Best: [0, 1, 0, 1, 0, 0, 1] ( 2063 ) fitness data: avarage: 2299.4305 || standard deviation: 549.3718114080405
Generation time: 0.08910465240478516

Best: [0, 1, 0, 1, 0, 0, 1] ( 2063 ) fitness data: avarage: 2316.6765 || standard deviation: 565.98257468561
Generation time: 0.12221431732177734

Best: [0, 1, 0, 1, 0, 0, 1] ( 2063 ) fitness data: avarage: 2284.155 || standard deviation: 529.7251324743792
Generation time: 0.10688304901123047

Best: [0, 1, 0, 1, 0, 0, 1] ( 2063 ) fitness data: avarage: 2295.7625 || standard deviation: 548.8936737600001
Generation time: 0.10273861885070801

Best: [0, 1, 0, 1, 0, 0, 1] ( 2063 ) fitness data: avarage: 2279.176 || standard deviation: 529.1633604322958
Generation time: 0.012964248657226562

Time elapsed: 1.972198724746704
solution price: 1735
solution weight: 169
optimal price: 1735
optimal weight: 169

Process finished with exit code 0
```

ו.מ. טנווילס: 207866328  
פארוק כריימן: 314656869

### בעיה 7:

```
Best: [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1] ( 1298 ) fitness data: avarage: 1350.5385 || standard deviation: 259.36038733343605
Generation time: 0.15062451362609863

Best: [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1] ( 1298 ) fitness data: avarage: 1353.744 || standard deviation: 265.23338301201795
Generation time: 0.19345569610595703

Best: [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1] ( 1298 ) fitness data: avarage: 1353.6245 || standard deviation: 265.27137142886323
Generation time: 0.18650174140930176

Best: [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1] ( 1298 ) fitness data: avarage: 1334.9545 || standard deviation: 215.26724420995865
Generation time: 0.20345568656921387

Best: [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1] ( 1298 ) fitness data: avarage: 1336.123 || standard deviation: 215.57341874869465
Generation time: 0.14461350440979004

Best: [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1] ( 1298 ) fitness data: avarage: 1349.5105 || standard deviation: 255.62945818068368
Generation time: 0.1471402645111084

Best: [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1] ( 1298 ) fitness data: avarage: 1356.218 || standard deviation: 274.11583769640185
Generation time: 0.015933513641357422

Time elapsed: 3.120293378829956
solution price: 1458
solution weight: 749
optimal price: 1458
optimal weight: 749

Process finished with exit code 0
```

### בעיה 8:

```
Best: [1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1] ( 12367115 ) fitness data: avarage: 14948058.0305 || standard deviation: 5093518.971804413
Generation time: 0.26684069633483887

Best: [1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1] ( 12367115 ) fitness data: avarage: 14918795.8745 || standard deviation: 5071343.078591605
Generation time: 0.18317794799886688

Best: [1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1] ( 12367115 ) fitness data: avarage: 14898065.585 || standard deviation: 5062238.898713213
Generation time: 0.22698868377685547

Best: [1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1] ( 12367115 ) fitness data: avarage: 14884774.1865 || standard deviation: 4988871.092463168
Generation time: 0.18699145317077637

Best: [1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1] ( 12367115 ) fitness data: avarage: 14988119.9365 || standard deviation: 5118040.783436778
Generation time: 0.21728014945983887

Best: [1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1] ( 12367115 ) fitness data: avarage: 15009546.69 || standard deviation: 5127187.959541217
Generation time: 0.21304941177368164

Best: [1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1] ( 12367115 ) fitness data: avarage: 15007888.1425 || standard deviation: 5158643.834938309
Generation time: 0.034989725189288984

Time elapsed: 6.9401695728302
solution price: 13549894
solution weight: 6402568
optimal price: 13549894
optimal weight: 6402568

Process finished with exit code 0
```