מעבדה בבינה מלאכותית

חלק א:

סעיף 1:

עבור בעיית בול פגיעה

```python
def distance( firstIndex, secondIndex):
    distances = numpy.zeros((len(firstIndex) + 1, len(secondIndex) + 1))

    for t1 in range(len(firstIndex) + 1):
        distances[t1][0] = t1

    for t2 in range(len(secondIndex) + 1):
        distances[0][t2] = t2

    for t1 in range(1, len(firstIndex) + 1):
        for t2 in range(1, len(secondIndex) + 1):
            if firstIndex[t1 - 1] == secondIndex[t2 - 1]:
                distances[t1][t2] = distances[t1 - 1][t2 - 1]
            else:
                a = distances[t1][t2 - 1]
                b = distances[t1 - 1][t2]
                c = distances[t1 - 1][t2 - 1]

                if a <= b and a <= c:
                    distances[t1][t2] = a + 1
                elif b <= a and b <= c:
                    distances[t1][t2] = b + 1
                else:
                    distances[t1][t2] = c + 1

    print(distances[len(firstIndex), len(secondIndex)])
```

עבור בעיית QUEENS-N

```
def distance(self, firstIndex, secondIndex):
    #values1 = self.population[firstIndex]
    #values2 = self.population[secondIndex]
    """Compute the Kendall tau distance."""
    values1 = firstIndex
    values2 = secondIndex
    n = len(values1)
    assert len(values2) == n, "Both lists have to be of equal length"
    i, j = numpy.meshgrid(numpy.arange(n), numpy.arange(n))
    a = numpy.argsort(values1)
    b = numpy.argsort(values2)
    ndisordered = numpy.logical_or(numpy.logical_and(a[i] < a[j], b[i] > b[j]),
                                   numpy.logical_and(a[i] > a[j], b[i] < b[j])).sum()
    print(ndisordered / (n * (n - 1)))
    return ndisordered / (n * (n - 1))
```

עבור בעיית BINPACKING

```
def distance(self, first, second):
    values1 = first.arr
    values2 = second.arr
    """Compute the Kendall tau distance."""
    n = len(values1)
    assert len(values2) == n, "Both lists have to be of equal length"
    i, j = numpy.meshgrid(numpy.arange(n), numpy.arange(n))
    a = numpy.argsort(values1)
    b = numpy.argsort(values2)
    ndisordered = numpy.logical_or(numpy.logical_and(a[i] < a[j], b[i] > b[j]),
                                   numpy.logical_and(a[i] > a[j], b[i] < b[j])).sum()
    # print("the distance is: ",ndisordered / (n * (n - 1)),"for indeces:",firstIndex,secondIndex)
    return ndisordered / (n * (n - 1))
```

סעיף 2:

Selection pressure code:

```
def selection_pressure(self):
    best_fit_number = 0
    best_fitness = self.population[0].fitness
    for i in range(self.GA_POPSIZE):
        if self.population[i].fitness == best_fitness:
            best_fit_number += 1
    print("selection pressure is: ", best_fit_number / self.GA_POPSIZE)

    return best_fit_number / self.GA_POPSIZE
```

Genetic diversity code:

```
def genetic_diversity(self, index):
    l = 0
    for j in range(self.GA_POPSIZE):
        l += self.distance(self.population[index], self.population[j])
    self.population[index].diversity = l
```

מצאנו שה selection pressure עולה ככל שאנחנו מתקדמים באיטירציות
למשל הריצה הזו התחיל כך

```
selection pressure is:  0.008
C:\Users\win10\PycharmProjects\AI_LAB1\BinPacking.py:196: R
  adaptive = 2 * self.pmax * self.pmax * pow(numpy.exp(1),
1 / 20
selection pressure is:  0.008
2 / 20
selection pressure is:  0.014
3 / 20
selection pressure is:  0.02
4 / 20
```

ואז עלה לכך

```
selection pressure is:  0.558
19 / 20
selection pressure is:  0.59
20 / 20
```

וה DIVERSITY ירד עם הזמו למשל התחיל כך:

```
C:\Users\win10\AppData\Local\Programs\Python\Python310\python.exe C:/U
Diversity is:  240.47040000000027
C:\Users\win10\PycharmProjects\AI_LAB1\BinPacking.py:196: RuntimeWarni
  adaptive = 2 * self.pmax * self.pmax * pow(numpy.exp(1), self.genera
1 / 20
Diversity is:  243.9722666666666
2 / 20
Diversity is:  240.6314666666668
3 / 20
Diversity is:  245.60239999999976
4 / 20
```

ואז ירד לכך:

```
Diversity is:  140.53039999999964
18 / 20
Diversity is:  126.2328000000004
19 / 20
Diversity is:  81.67039999999966
20 / 20

BEST FOUND IN 20 STEPS:
 [3, 1, 3, 1, 1, 3]  ( 257 )
num of packs :  2
```

סעיף 3:

מצאנו שהכי טוב שהיה הוא מעבר הדרגתי מ EXPLORATION ל EXPLOTATION הקבוע היה נתקע הכי הרבה בלוקאל אופטימום

קוד הadaptive שלנו:

```
self.pmax = self.pmax * self.pmax
adaptive = 2 * self.pmax * self.pmax * pow(numpy.exp(1), self.generation_number * 0.5) / (
        self.pmax + self.pmax * pow(numpy.exp(1), self.generation_number * 0.5))
if random() < adaptive * self.GA_MUTATION:
    self.inversion_mutation(i)
```

הוא היה הכי כבד ולקח הרב ה זמן בגלל הנוסחה שלו כלומר זמן של כל איטירציה עלה הרבה

סעיף 4:

קוד של THRESHHOLD SPECIATION :

```
def Threshhold_spec(self):
    numberof_species = -1
    for i in range(self.GA_POPSIZE):
        flag = 0
        for j in range(len(self.species)):
            counter = 0
            for k in range(len(self.species[j])):
                destence = self.distance(self.population[i], self.species[j][k])
                if destence < self.threshhold:
                    counter += 1
            if counter == len(self.species[j]):
                flag = 1
                self.population[i].specynum = j
                self.species[j].append(self.population[i])
                break
        if flag == 0:
            new_array = []
            self.population[i].specynum = len(self.species)
            new_array.append(self.population[i])
            self.species.append(new_array)
            numberof_species += 1
```

קוד של CLUSTRING

```
def clustering_spec(self):
    self.classifier = []
    self.centers = []
    for i in range(self.k):
        self.classifier.append([])
        self.centers.append(self.population[i])
    for i in range(self.maxSteps):
        for j in range(self.GA_POPSIZE):
            dis = []
            for citizen in self.centers:
                dis.append(self.distance(self.population[j], citizen))
            index = dis.index(min(dis))
            self.classifier[index].append(self.population[j])
            self.population[j].classy = index
        self.calc_new_centers()
```

אלגוריתם של THRESHHOLD הוא מאוד כבד ודורש הרבה זמן לבצע כי הוא עושה הרבה השוואות ה CLUSTERING אלגוריתם מתבצע ביותר מהירות כאשר ה K הוא לא גדול

ולפעמים מצאנו ש CLUSTERING דורש יותר איטירציות לצאת מ לוקאל אופטימום

סעיף 5:

```
def random_immigrant(self):
    esize = self.GA_POPSIZE * 0.7
    tsize = self.N
    for i in range(int(esize), self.GA_POPSIZE):
        i1 = randint(0, int(esize))
        self.population[i] = self.population[i1]
        if random() < self.GA_MUTATION:
            self.inversion_mutation(i)
```

בחרנו להחליף את 30% הגנים הכי גרועים

סעיף 6:

א) שלושת האלגוריתמים בשתי הגרסאות מוצאים פתרון כלשהו כלומר הם שלימים

ב) איצולינו עבור בעיית בול פגיעה ו NQUEENS במעבדה הזו תמיד מוצא פתרון אופטימאלי וגם במעבדה הישנה אם משתמשים ב MINIMAL CONFLICTS עבור NQUEENS בסוף מגיעים לתוצאה הנדרשת אבל ב BINPACKING לא תמיד מוצא את האופטימלי אבל במעבדה הזו הגיע לפתרונות יותר טובים

ג) בגרסה של המעבדה שעברה כל האלגוריתמים התכנסו יותר לאט מהגרסה החדשה הגרסה החדשה מצאה פתרון בפחות איטירציות

ד) בגרסה הישנה היה יותר מהיר כי לא היה צריך לחשב קבוצות ו DIVERSITY שהוא כבד מאוד בגלל זה היה יותר מהר עבור כל הבעיות