Galatasaray Üniversitesi

DAĞITIK SİSTEMLER VE UYGULAMALARI QUIZ 5 RAPOR

17401455 - TUĞÇE ÇELİK

18401808 - SERKAN ŞAHİN

19401852 - DOĞA YAĞMUR YILMAZ

18401911 - MELİSSA HAŞİMBEGOVİÇ

İçindekiler:

1. Problem : Face Creation

Face Creation problemi, 15 elemanlı bir float arrayiyle gülen yüz çizdirmektir. Bu arrayin ilk 5 elemanı gözlerin konumuna 6-8 elemanları burnun konumuna ve 9-15. elemanları ise ağzın konumuna denk gelmektedir. Bu 15 uzunluğundaki array her bir agent'ın genomuna denk gelmektedir. Array'in içindeki her bir float eleman gendir.

Her bir agent oluşturulduğunda genomları random genlerle oluşturulur.

Her bir agent'ın ulaşmak istediği hedef gen dizlimi vardır.

Ulaşılmak istenilen hedef gen dizlimi: target_gene
eyes = [2.25, 3.75, 5, 5, 4] nose = [3, 3, 6] mouth = [2.5, 3, 3.5, 1.15, 1, 1.15, 8] target_gene = np.array(eyes + nose + mouth)

Populationumuz 10 farklı agent'dan oluşmaktadır. (10 olmasını gerekmez bu sayıyı değişebilir, ne kadar artarsa o kadar iyi sonuç elde eder.)

Agentların bu hedef gen dizlimine ulaşması ya da yaklaşması için Genetik Algoritma kullanarak population üzerinde Selection (parent selection) ve Cross-Over yaparak yeni agentlar (child) oluştururuz. Oluşturduğumuz agentlar için de Mutation işlemini gerçekleştirerek yeni genler ekleriz.

Hedef gen dizilimine ne kadar yaklaştığımızı child için kontrol ederiz ve eğer yeterince yaklaşmadıysak (early_stop) (ör: 0.95 oranından çok benzediyse genler, yeterince yaklaşmışız demektir.) Yukarıdaki işlemi istediğimiz kadar tekrar ettirebiliriz.

Ne kadar çok child oluşturursak hedef gen dizilimine o kadar yaklaşırız.

• Paralel GA Hakkında:

İlk generation (population) Master'da oluşturulup Fast ve Slow 'a gönderilir.

Mutation rate (Genetikte mutasyon oranı) : Zaman içinde tek bir gen veya organizmadaki yeni mutasyonların sıklığıdır.

Fast için mutation rate 0.90, Slow için 0.10 'dur. Fast ve Slow GA'yı çalıştırır ve 10 generation sonra çıktı vermektedir. Bu çıktıyı Master'a gönderir.

Master her iki çıktıyı alır. Master kendine gelen 20 yeni agent'tan fitnessları en iyi olan 10 agent'i seçer ve population haline getirir.

Bunu master 100 defa tekrar. Böylece 1000 generation gideriz. En iyi sonuca ulaştığımızda (fitness >= 0.95), early stop yaparız.

Paralel GA Kodunun Çalıştırılması ve Çıktılar:

NOT: Kodun çalıştırılması için FaceCreation.zip klasöründeki .py uzantılı dosyalar baz alınmıştır. Eğer üzerinde değişiklik yaptıysanız aşağıdaki adımlar işe yaramayabilir.

Paralel GA'yı çalıştırmak için:

Öncelikle Master.py dosyasındaki
`if __name__ == '__main__':`
bloğundaki tüm kodlar yorum satırındadır.
Aşağıdaki kodlar yorum satırından çıkarıp Master.py terminalden çalıştırılmalıdır.

```
generation = GA.evolution(population_size)
TCP_IP = 'localhost'
TCP_PORT = 6001
BUFFER_SIZE = 100000
tcpServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpServer.bind((TCP_IP, TCP_PORT))
threads = []
tcpServer.listen(2)
for i in range(2):
    (conn, (ip, port)) = tcpServer.accept()
    threadLock.acquire()
    connectedConns.append(conn)
    masterThread = Master(ip, port, conn, i)
    threadLock.release()
    masterThread.start()
    threads.append(masterThread)

for t in threads:
    t.join()
```

Master.py çalıştıktan sonra, Slow.py ve Fast.py'nin farklı terminallerden çalıştırılması gerekmektedir.

Paralel GA Sonucu:

```
0.96 is found in 72 iteration with 2 different mutation chance
0.978 is found in 59 iteration with 2 different mutation chance
0.951 is found in 49 iteration with 2 different mutation chance
0.975 is found in 70 iteration with 2 different mutation chance
0.951 is found in 47 iteration with 2 different mutation chance
0.952 is found in 42 iteration with 2 different mutation chance
0.95 is found in 57 iteration with 2 different mutation chance
0.956 is found in 89 iteration with 2 different mutation chance
0.953 is found in 51 iteration with 2 different mutation chance
0.95 is found in 67 iteration with 2 different mutation chance
0.951 is found in 76 iteration with 2 different mutation chance
0.954 is found in 72 iteration with 2 different mutation chance
0.96 is found in 63 iteration with 2 different mutation chance
0.952 is found in 56 iteration with 2 different mutation chance
0.963 is found in 68 iteration with 2 different mutation chance
0.96 is found in 57 iteration with 2 different mutation chance
0.955 is found in 57 iteration with 2 different mutation chance
0.961 is found in 57 iteration with 2 different mutation chance
0.954 is found in 80 iteration with 2 different mutation chance
0.95 is found in 63 iteration with 2 different mutation chance
0.95 is found in 53 iteration with 2 different mutation chance
The algorithm ran 20 times. The Average Evolution Count is: 62.6. iteration with 2 different mutation chance
```

• Standart GA Hakkında:

Doğrudan 1000 generation gideriz. En iyi sonuca ulaştığımızda (fitness >= 0.95) , early stop yaparız.

Farklı mutation rate'ler için sonuçlar oluşturmayı amaçlıyoruz. 0.05'i, 0.10'u, 0.15'i ..., 0.90'ı, 0.95i ve 1'i ayrı ayrı deneyelim.

Standart GA Kodunun Çalıştırılması ve Çıktılar:

```
tryArr = [0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.90, 0.95, 1]
```

tryArr dizisindeki mutasyon olasılığı değerleriyle standart GA'da algoritmayı 20 kez 1000 generation'a gidecek şekilde çalıştırmak için Master.py dosyasındaki ilgili kodu yorum satırından çıkarıp Master.py'yi öyle çalıştırmamız gerekmektedir.

NOT: Kodun çalıştırılması için FaceCreation.zip klasöründeki .py uzantılı dosyalar baz alınmıştır. Eğer üzerinde değişiklik yaptıysanız aşağıdaki adımlar işe yaramayabilir.

Standart GA'yı çalıştırmak için:

Master.py dosyasındaki
`if __name__ == '__main__':`
içerisindeki tüm kodlar yorum satırındadır.
Yalnızca, aşağıdaki kodları yorum satırından çıkarıp Master.py'yi çalıştırmamız yeterlidir.

```
tryArr = [0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.90, 0.95, 1]

for prob in range(len(tryArr)):
    totalEv = 0
    GA.p = tryArr[prob]
    for j in range(20):
        world = GA.evolution(10)
        for i in range(1000):
            best = world.evolve(G=1)
            totalEv = totalEv + 1
            if world.best_agent.fitness() >= 0.95:
                break
    print(f'For mutation probabilty : {GA.p}\nThe algorithm ran {j + 1} times.The Average Evolution Count is:
{totalEv / 20}\n')
```

Standart GA Sonuçları:

```
For mutation probabilty : 0.05
The algorithm ran 20 times.The Average Evolution Count is: 1000.0

For mutation probabilty : 0.1
The algorithm ran 20 times.The Average Evolution Count is: 1000.0

For mutation probabilty : 0.15
The algorithm ran 20 times.The Average Evolution Count is: 1000.0

For mutation probabilty : 0.2
The algorithm ran 20 times.The Average Evolution Count is: 1000.0

For mutation probabilty : 0.25
The algorithm ran 20 times.The Average Evolution Count is: 1000.0

For mutation probabilty : 0.3
The algorithm ran 20 times.The Average Evolution Count is: 982.3

For mutation probabilty : 0.35
The algorithm ran 20 times.The Average Evolution Count is: 974.1

For mutation probabilty : 0.4
The algorithm ran 20 times.The Average Evolution Count is: 873.1

For mutation probabilty : 0.45
The algorithm ran 20 times.The Average Evolution Count is: 867.0

For mutation probabilty : 0.5
The algorithm ran 20 times.The Average Evolution Count is: 779.9
```

```
For mutation probabilty : 0.55
The algorithm ran 20 times.The Average Evolution Count is: 823.7

For mutation probabilty : 0.6
The algorithm ran 20 times.The Average Evolution Count is: 818.2

For mutation probabilty : 0.65
The algorithm ran 20 times.The Average Evolution Count is: 734.85

For mutation probabilty : 0.7
The algorithm ran 20 times.The Average Evolution Count is: 686.4

For mutation probabilty : 0.75
The algorithm ran 20 times.The Average Evolution Count is: 715.6

For mutation probabilty : 0.8
The algorithm ran 20 times.The Average Evolution Count is: 697.15

For mutation probabilty : 0.9
The algorithm ran 20 times.The Average Evolution Count is: 712.3

For mutation probabilty : 0.95
The algorithm ran 20 times.The Average Evolution Count is: 782.95

For mutation probabilty : 1
The algorithm ran 20 times.The Average Evolution Count is: 858.3


Process finished with exit code 0
```

• Sonuç Değerlendirmeleri:

Paralel GA'da Slow 0.1 ve Fast 0.9 mutation rateleriyle paralel bir şekilde evolution yaptığımızda Master ortalama 62.2 kez Slow ve Fast'e algoritmayı çalıştırdı. Bunu 1000 generation'a giderken ortalama 622. denemede early stop yaptık diye yorumlayabiliriz.

Standart GA'da ise en hızlı sonuçlar 0.7 ve 0.8 mutation ratelerinde alındı. 0.7 için ortalam 686.4; 0.8 için ortalama 697.15 denemede early stop'a ulaştık. Genel olarak da mutation rate yükseldikçe, popülasyonda çeşitlilik ve gelişmenin ihtimalinin de artması demek olduğundan, daha az denemede iyi sonuçlara ulaşılabilmekte. Fakat geri kalan mutation ratelerde çok yüksek ortalamaların elde edilmesi güvenilirliği düşürebilir.

Standart GA'da, hem 0.1 ve 0.9 baktığımızda hem de en iyi sonucu veren 0.7 ve 0.8 'e baktığımızda Paralel GA'nın daha hızlı çalıştığını görmekteyiz.

2. Problem : Product Distribution

Bu problemimizde bir satıcı olarak elimizde belirli (sınırlı) stoklara sahip beş çeşit ürün bulunmaktadır.

Items	A	B	C	D	E
Stock (**s**)	30	40	20	40	20

İsteğimiz ise ürünlerimizi beş farklı şehirde satmak. Her şehirde her ürün için farklı alış fiyatı bulunmaktadır.

$\mathbf{P} = \{p_{i,j}\}$	x_1	x_2	x_3	x_4	x_5
A	1	4	6	4	4
B	3	8	2	5	15
C	3	12	3	5	5
D	2	6	10	2	4
E	10	5	12	6	3

Amacımız, açıklayacağımız bazı sınırlarla uyumlu olarak, ürün sayıları ve şehirler arasında toplam kazancımızı maksime edecek bir “match matrix” bulmaktır.

Satılan ürünlerden elde edilen kazancın temeli şu şekildedir:

$$f_{base} = \sum_{i} \sum_{j} p_{i,j} m_{i,j}$$

pi,j -> bir “i” ürününü bir “j” şehrinde satmakla elde edilen kazanç.

mi,j -> “i” ürününün “j” şehrinde satılma miktarı.

Sınırlamalarımız:

- “Match Matrix” doğal sayılardan oluşacak.

- Tam olarak 150 ürün satılacak.

- Satılan ürün miktarları stoklarına uygun olacak.

- Eğer tüm şehirler ziyaret edilirse satıcı bonus kazanabilir.

- Eğer satış miktarları dengeli olursa satıcı %20’ye kadar bonus kazanabilir.

Agentlarımız matrixlerden oluşmaktadır. Bu agentları arrayler içinde arrayler şeklinde kodumuza entegre etmekteyiz.

Agent

Agent'lar popülasyondaki bireylere denk gelir.

productA = [] # 30 adet stok

productB = [] # 40

productC = [] # 20

productD = [] # 40

productE = [] # 20

individual = [productA,....,productE] bireyimizin gen dizilimidir.

- “productX" için index sayısı elimizdeki ürün sayısına denktir.
- “productX” dizisinde 0, 1, 2, 3, 4 rakamları tutulmaktadır.Bu rakamlar ürünlerin satıldığı şehirleri temsil etmektedir.
- Örneğin, A ürünü elmalar olsun. 30 tane elmamız var demek bu. Her bir index 1 adet elmaya denk gelmektedir. productA[0] = 4 olsun, bu 0. indexteki elma 4. şehirde satıldı demek.

Problemi böyle kurgulamamızın nedeni elimizdeki tüm ürünü satmak istememiz ve satılacak ürünün depomuzdakinin asla ve asla fazla olamayacak olması.

Şehirlerin önemli olmasının nedeni, her bir ürünün fiyatının şehirden şehre değişmesidir. Bu fiyat listesini allPrice dizisinde tutarız.

priceA = [1, 4, 6, 4, 4]

priceB = [3, 8, 2, 5, 15]

priceC = [3, 12, 3, 5, 5]

priceD = [2, 6, 10, 2, 4]

priceE = [10, 5, 12, 6, 3]

allPrice = []

allPrice.append(priceA)

allPrice.append(priceB)

allPrice.append(priceC)

allPrice.append(priceD)

allPrice.append(priceE)

Fitness

Agent'ın ne kadar kar getirdiğini “def fitness(self):” fonksiyonu ile hesaplarız.

Fitness fonksiyonumuzda ana kazanç hesabı (Durum 0), tüm şehirlerin ziyaret edilmesi (Durum 1) her şehir için her üründen aynı miktarda satılması (Durum 2) için değişkenlerin belirlenmesi.

```
def fitness(self):
    global allPrice
    fbase = 0  # Durum 0: fbase hesab�

    f2 = 0  # Durum 2  de�i�kenleri
    x1 = []
    x2 = []
    x3 = []
    x4 = []
    x5 = []
    fx1base = 0
    fx2base = 0
    fx3base = 0
    fx4base = 0
    fx5base = 0  # Durum 2 de�i�kenleri

    # Durum 1: t�m �ehirlerin ziyaret edilmesi
    flag = [0, 0, 0, 0, 0]
    f1 = 0
    for i in range(len(self.genome)):
        x1.append(self.genome[i].count(0))  # durum 2 i�in gereken parametrelerin hesab�
        x2.append(self.genome[i].count(1))
        x3.append(self.genome[i].count(2))
        x4.append(self.genome[i].count(3))
        x5.append(self.genome[i].count(4))
        fx1base = fx1base + x1[i] * allPrice[i][0]
        fx2base = fx2base + x2[i] * allPrice[i][1]
```

Tüm şehirlerin ziyaret edilmesi (Durum 1) kontrolü ve her şehir için her üründen aynı miktarda satılması (Durum 2) parametreleri hesabı

```
    # Durum 1: t�m �ehirlerin ziyaret edilmesi
    flag = [0, 0, 0, 0, 0]
    f1 = 0
    for i in range(len(self.genome)):
        x1.append(self.genome[i].count(0))  # durum 2 i�in gereken parametrelerin hesab�
        x2.append(self.genome[i].count(1))
        x3.append(self.genome[i].count(2))
        x4.append(self.genome[i].count(3))
        x5.append(self.genome[i].count(4))
        fx1base = fx1base + x1[i] * allPrice[i][0]
        fx2base = fx2base + x2[i] * allPrice[i][1]
        fx3base = fx3base + x3[i] * allPrice[i][2]
        fx4base = fx4base + x4[i] * allPrice[i][3]
        fx5base = fx5base + x5[i] * allPrice[i][4]  # durum 2 sonu
        for j in range(5):
            p = allPrice[i][j]  # durum 0 fbase hesab�
            itemNum = self.genome[i].count(j)
            fbase = fbase + (p * itemNum)  # durum 0 fbase hesab�
            if (self.genome[i].count(j) > 0):  # durum 1 kontrolu
                flag[j] = 1

    if (flag.count(0) == 0):  # durum 1 kontrolu
        f1 = 100

    # Durum 2: Her bir �ehir i�in, Her ür�nden ayn� miktarda sat�lmas�

    f2 = f2 + fx1base * self.findf2or3Rate(x1)
```

Her şehir için her üründen aynı miktarda satılması (Durum 2) , Tüm şehirlerde dengeli ürün satışı ve toplam kazanç (score) hesabı

```
            if (self.genome[i].count(j) > 0):  # durum 1 kontrolu
                flag[j] = 1

    if (flag.count(0) == 0):  # durum 1 kontrolu
        f1 = 100

    # Durum 2: Her bir ♦ehir i♦in, Her ♦r♦nden ayn♦ miktarda sat♦lmas♦

    f2 = f2 + fx1base * self.findf2or3Rate(x1)
    f2 = f2 + fx2base * self.findf2or3Rate(x2)
    f2 = f2 + fx3base * self.findf2or3Rate(x3)
    f2 = f2 + fx4base * self.findf2or3Rate(x4)
    f2 = f2 + fx5base * self.findf2or3Rate(x5)
    # Durum 3: T♦m ♦ehirlerde dengeli miktarda ♦r♦n sat♦lmas♦ durumu (en iyi durum hepsinde 30 ♦r♦n sat♦lmas♦)
    # bunun i♦in t♦m ♦ehirlerde sat♦lan toplam ♦r♦n miktarlar♦n♦ tuttu♦umuz bir dizi olu♦tural♦m
    f3 = 0
    Cities = [sum(x1), sum(x2), sum(x3), sum(x4), sum(x5)]
    f3 = f3 + fbase * self.findf2or3Rate(Cities)
    """print("fbase:", fbase)
    print("f1:", f1)
    print("f2:", f2)
    print("f3:", f3)"""
    score = fbase + f1 + f2 + f3
    return score
```

Evolution

Evolution class içerisinde selection, cross-over ve mutation işlemleri yapılmaktadır.

Agentların hedef gen dizlimine yaklaşması (maksimum kazanç) için Genetik Algoritma kullanarak population üzerinde Selection (parent selection) ve Cross-Over yaparak yeni agentlar (child) oluştururuz. Oluşturduğumuz agentlar için de Mutation işlemini gerçekleştirerek yeni genler ekleriz.

Hedef gen dizilimine ne kadar yaklaştığımızı child için kontrol ederiz ve eğer yeterince yaklaşmadıysak (early_stop) (ör:1200 kazancına ulaştıysa/1200 kazancını geçtiyse, yeterince yaklaşmışız demektir.) Yukarıdaki işlemi istediğimiz kadar tekrar ettirebiliriz.

Ne kadar çok child oluşturursak hedef gen dizilimine o kadar yaklaşırız.

Popülasyon oluşturma ve cross-over

```
class evolution():
    def __init__(self, N):
        self.N = N
        self.population = {i: agent(i) for i in range(N)}
        self.update_probabilities()

    def update_probabilities(self):
        self.success = {i: self.population[i].fitness() for i in range(self.N)}
        total_success = sum(self.success.values())

        self.reproduction_probability = {i: self.success[i] / total_success for i in range(self.N)}

        sorted_by_success = sorted(self.success.items(), key=lambda kv: kv[1])
        self.best_agent = self.population[sorted_by_success[-1][0]]

    def selection(self):
        pr = [self.reproduction_probability[i] for i in range(self.N)]
        select = np.random.choice(self.N, 2, replace=False, p=pr)
        return select

    def crossover(self, selectedParents):
        parent0 = self.population[selectedParents[0]].genome
        parent1 = self.population[selectedParents[1]].genome
        child_gene = []
        cut = np.random.randint(len(parent1))
        for i in range(0, cut):
            child_gene.append(parent0[i])
        for i in range(cut, len(parent1)):
            child_gene.append(parent1[i])
```

Mutasyon ve yeni popülasyon yaratma

```
        #child_gene = np.hstack((parent0[:cut], parent1[cut:]))
        return child_gene

    def mutation(self, child_gene):
        global p
        mutation_point = np.random.randint(len(child_gene))
        if np.random.rand() < p:
            for i in range(15):
                mutation_point2 = np.random.randint(len(child_gene[mutation_point]))
                child_gene[mutation_point][mutation_point2] = np.random.randint(5)
        return child_gene

    def create_offspring(self):
        parents = self.selection()
        child_gene = self.crossover(parents)

        child_gene = self.mutation(child_gene)
        return child_gene

    def create_new_population(self):
        sorted_by_success = sorted(self.success.items(), key=lambda kv: kv[1])
        self.best_agent = self.population[sorted_by_success[-1][0]]

        for i in range(self.N // 2):
            child_gene = self.create_offspring()
            agent_id = sorted_by_success[i][0]
            self.population[agent_id].set_gene(child_gene)

        self.update_probabilities()
```

Yeni popülasyonu verilen parametrede (10) tekrarlama

```
    self.update_probabilities()

def evolve(self, G=10):
    for i in range(G):
        self.create_new_population()
    return self.population
```

• Paralel GA Hakkında:

İlk generation (population) Master'da oluşturulup Fast ve Slow 'a gönderilir.

Mutation rate (Mutasyon oranı): Zaman içinde tek bir gen veya organizmadaki yeni mutasyonların sıklığıdır.

Fast için mutation rate 0.90, Slow için 0.10 'dur. Fast ve Slow GA'yı çalıştırır ve 10 generation sonra çıktı vermektedir. Bu çıktıyı Master'a gönderir.

Master her iki çıktıyı alır. Master kendine gelen 20 yeni agent'tan fitnessları en iyi olan 10 agent'ı seçer ve population haline getirir.

Bunu master 100 defa tekrar. Böylece 1000 generation gideriz. En iyi sonuca ulaştığımızda (Kendimiz belirlemekteyiz, genellikle 1200 değerini tercih ettik. Bunun kontrolü için ise : generation.best_agent.fitness() >= fitness_value), early stop yaparız.

Paralel GA Kodunun Çalıştırılması ve Çıktılar:

Paralel GA'yı çalıştırmak için:

Öncelikle Master.py dosyasındaki
`if __name__ == '__main__':`
bloğundaki tüm kodlar yorum satırındadır.
Aşağıdaki kodlar yorum satırından çıkarıp Master.py terminalden çalıştırılmalıdır.

```
generation = GA.evolution(population_size)
TCP_IP = 'localhost'
TCP_PORT = 6001
BUFFER_SIZE = 100000
tcpServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpServer.bind((TCP_IP, TCP_PORT))
threads = []
tcpServer.listen(2)
for i in range(2):
    (conn, (ip, port)) = tcpServer.accept()
    threadLock.acquire()
    connectedConns.append(conn)
    masterThread = Master(ip, port, conn, i)
    threadLock.release()
    masterThread.start()
    threads.append(masterThread)

for t in threads:
    t.join()
```

Master.py çalıştıktan sonra, Slow.py ve Fast.py'nin farklı terminallerden çalıştırılması gerekmektedir.

Paralel GA Sonucu:

```
1207.76 is found in 6 iteration with 2 different mutation chance
1236.8 is found in 23 iteration with 2 different mutation chance
1249.84 is found in 19 iteration with 2 different mutation chance
1209.38 is found in 14 iteration with 2 different mutation chance
1215.25 is found in 15 iteration with 2 different mutation chance
1205.78 is found in 14 iteration with 2 different mutation chance
1272.77 is found in 1 iteration with 2 different mutation chance
1202.44 is found in 31 iteration with 2 different mutation chance
1213.56 is found in 1 iteration with 2 different mutation chance
1219.42 is found in 1 iteration with 2 different mutation chance
1207.07 is found in 3 iteration with 2 different mutation chance
1212.01 is found in 5 iteration with 2 different mutation chance
1252.16 is found in 34 iteration with 2 different mutation chance
1235.13 is found in 1 iteration with 2 different mutation chance
1213.98 is found in 3 iteration with 2 different mutation chance
1207.86 is found in 1 iteration with 2 different mutation chance
1246.66 is found in 12 iteration with 2 different mutation chance
1210.41 is found in 36 iteration with 2 different mutation chance
1304.76 is found in 2 iteration with 2 different mutation chance
1207.5 is found in 11 iteration with 2 different mutation chance
1200.74 is found in 6 iteration with 2 different mutation chance
The algorithm ran 20 times. The Average Evolution Count is: 11.65. iteration with 2 different mutation chance
```

• Standart GA Hakkında:

Doğrudan 1000 generation gideriz. En iyi sonuca ulaştığımızda (fitness >= 1200) , early stop yaparız.

Farklı mutation rate'ler için sonuçlar oluşturmayı amaçlıyoruz. 0.05'i, 0.10'u, 0.15'i ..., 0.90'ı, 0.95i ve 1'i ayrı ayrı deneyelim.

Standart GA Kodunun Çalıştırılması ve Çıktılar:

Standart GA'yı çalıştırmak için:

Master.py dosyasındaki
`if __name__ == '__main__':`
içerisindeki tüm kodlar yorum satırındadır.
Yalnızca, aşağıdaki kodları yorum satırından çıkarıp Master.py'yi çalıştırmamız yeterlidir.

```
tryArr = [0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.90, 0.95, 1]

for prob in range(len(tryArr)):
    totalEv = 0
    GA.p = tryArr[prob]
    for j in range(20):
        world = GA.evolution(10)
        for i in range(1000):
            best = world.evolve(G=1)
            totalEv = totalEv + 1
            if world.best_agent.fitness() >= fitness_value:
                break
        #print(j, i, world.best_agent.fitness())
    print(f'For probabilty : {GA.p}\nThe algorithm ran {j + 1} times.The Average Evolution Count is: {totalEv / 20}\n')
    #print(f'Final fitness Score: {world.best_agent.fitness()}\n')
```

Standart GA Sonuçları:

```
For probabilty : 0.05
The algorithm ran 20 times.The Average Evolution Count is: 222.2

For probabilty : 0.1
The algorithm ran 20 times.The Average Evolution Count is: 15.0

For probabilty : 0.15
The algorithm ran 20 times.The Average Evolution Count is: 86.3

For probabilty : 0.2
The algorithm ran 20 times.The Average Evolution Count is: 39.2

For probabilty : 0.25
The algorithm ran 20 times.The Average Evolution Count is: 67.65

For probabilty : 0.3
The algorithm ran 20 times.The Average Evolution Count is: 49.9

For probabilty : 0.35
The algorithm ran 20 times.The Average Evolution Count is: 19.75

For probabilty : 0.4
The algorithm ran 20 times.The Average Evolution Count is: 30.75

For probabilty : 0.45
The algorithm ran 20 times.The Average Evolution Count is: 32.9

For probabilty : 0.5
The algorithm ran 20 times.The Average Evolution Count is: 16.2
```

```
For probabilty : 0.55
The algorithm ran 20 times.The Average Evolution Count is: 12.55

For probabilty : 0.6
The algorithm ran 20 times.The Average Evolution Count is: 59.25

For probabilty : 0.65
The algorithm ran 20 times.The Average Evolution Count is: 24.55

For probabilty : 0.7
The algorithm ran 20 times.The Average Evolution Count is: 6.8

For probabilty : 0.75
The algorithm ran 20 times.The Average Evolution Count is: 31.85

For probabilty : 0.8
The algorithm ran 20 times.The Average Evolution Count is: 21.75

For probabilty : 0.9
The algorithm ran 20 times.The Average Evolution Count is: 8.5

For probabilty : 0.95
The algorithm ran 20 times.The Average Evolution Count is: 16.8

For probabilty : 1
The algorithm ran 20 times.The Average Evolution Count is: 8.05


Process finished with exit code 0
```

• Sonuç Değerlendirmeleri:

Paralel GA'da Slow 0.1 ve Fast 0.9 mutation rateleriyle paralel bir şekilde evolution yaptığımızda, algoritmamızı 20 kere çalıştırmamızın sonucunda ortalama 11.65 evolutionda, o popülasyonun, fitness değerine en uygun sonuca ulaştığını görüyoruz.

Standart GA'da ise en hızlı sonuçlar 0.7 mutation rate'inde alındı. 0.7 için ortalama 6.8 denemede early stop'a ulaştık. Genel olarak da mutation rate yükseldikçe, popülasyonda çeşitlilik ve gelişmenin ihtimalinin de artması demek olduğundan, daha az denemede iyi sonuçlara ulaşılabilmekte. Fakat geri kalan mutation ratelerde çok yüksek ortalamaların elde edilmesi güvenilirliği düşürebilir.

Genel olarak bakıldığında ise en iyi sonuca ulaşmak için gerekli evolution sayısının ortalamada paralel GA'da daha az olması ve Standartta elde edilen en iyi durumun paralelden daha iyi olmasına rağmen

o sonuca ulaşmanın şartının (0.7 mutation rate) önceden öngörülmesi zor olduğundan bu paralelin standart GA üzerine bir başarısı sayılabilir.

İki tip GA'ya bu problem kapsamında tekrar baktığımızda da, aynı bilgisayarda ve aynı şartlarda çalıştırılan paralel GA'nın standart GA'ya göre daha hızlı sonuç verdiğini gördük.