BM-469 Genetik Algoritmalar

2023-2024 Final Proje Raporu

Aycan Kaynakcı

191180052

1.Özet

Bu final proje raporu, Genetik Algoritmaların öğretmen atama ve çizelgeleme problemlerine uygulanması üzerine detaylı bir makale incelemesini ve problem çözümünü kapsamaktadır. İncelenen makelenin konusu özel bir dershanede farklı derslerin en uygun öğretmenlere atanması ve bu öğretmenlerin ilgili sınıflara yerleştirilmesi sorununa genetik algoritmalarla çözüm bulunmasıdır. Manuel olarak belirlenen iterasyon, mutasyon oranı ve çaprazlama oranlarını içeren bir arayüz kullanılmıştır. Öğretmen, sınıf ve ders bilgileri, txt dosyalarında tutularak bir kromozom yapısı oluşturulmuştur. Popülasyonun fitness değeri, atanamayan öğretmen veya sınıf sayısının toplamı olarak hesaplanmış ve düşük fitness değeri hedeflenmiştir. Küçük fitness değeri elde etmek için mutasyon ve tek noktalı çaprazlama kodları yazılmış. Kodların bir kısmına rapor içersinde yer verilmiştir.

2.Giriş

Genetik Algoritmalar Final Projesi çerçevesinde, öğretmen atama ve çizelgeleme problemlerini ele alan bir makale üzerine detaylı bir inceleme gerçekleştirdim. Bu çalışmada, özel bir dershanede farklı derslerin en uygun öğretmenlere atanması ve bu öğretmenlerin ilgili sınıflara yerleştirilmesi sorununa genetik algoritmalarla çözüm bulunmaya çalışılmıştır.

Çizelgeleme problemlerinin çözümü için genetik algoritmalar dışında Integer Programlama, Heuristik Modeller, Simulated Annealing ve Tabu Araştırması gibi teknikler de kullanılmaktadır. Bu tekniklerin incelenmesi, genetik algoritmaların projedeki rolünü daha iyi anlamamı sağladı.

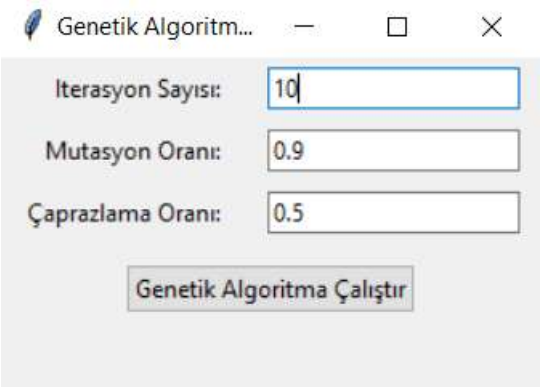
Projenin başlangıcında, öğretmen (teacher), sınıf (class) ve ders (lesson) bilgilerinin farklı tablolardan alınarak bir kromozom yapısı oluşturuldu. Popülasyonun fitness değeri, atanacak öğretmen veya sınıf bulunamadığında açılamayan grup sayısının toplamı olarak hesaplandı. Yani, çözümde fitness değerinin düşük olması hedeflendi. Aynı saat diliminde bir sınıf iki grup tarafından kullanılıyorsa veya aynı öğretmen aynı saat diliminde iki ders veriyorsa, ikinci grup açılamıyordu. Bu durumlar fitness değerini belirleyen unsurlardı.

Eğer fitness değeri 0 ise program durduruluyordu; değilse belirlenen iterasyon sayısı boyunca mutasyon ve tek noktalı çaprazlama işlemleri gerçekleştiriliyordu. En düşük fitness değerine sahip popülasyon, bir txt dosyasına kaydedilmekteydi. Projenin en önemli özelliklerinden biri , iterasyon sayısı, mutasyon oranı ve çaprazlama oranının manuel olarak belirlenmesidir.

Diğer yöntemlerle karşılaştırıldığında, genetik algoritmaların karmaşık problemlerde başarıyla kullanılabileceği ortaya çıkmaktadır.

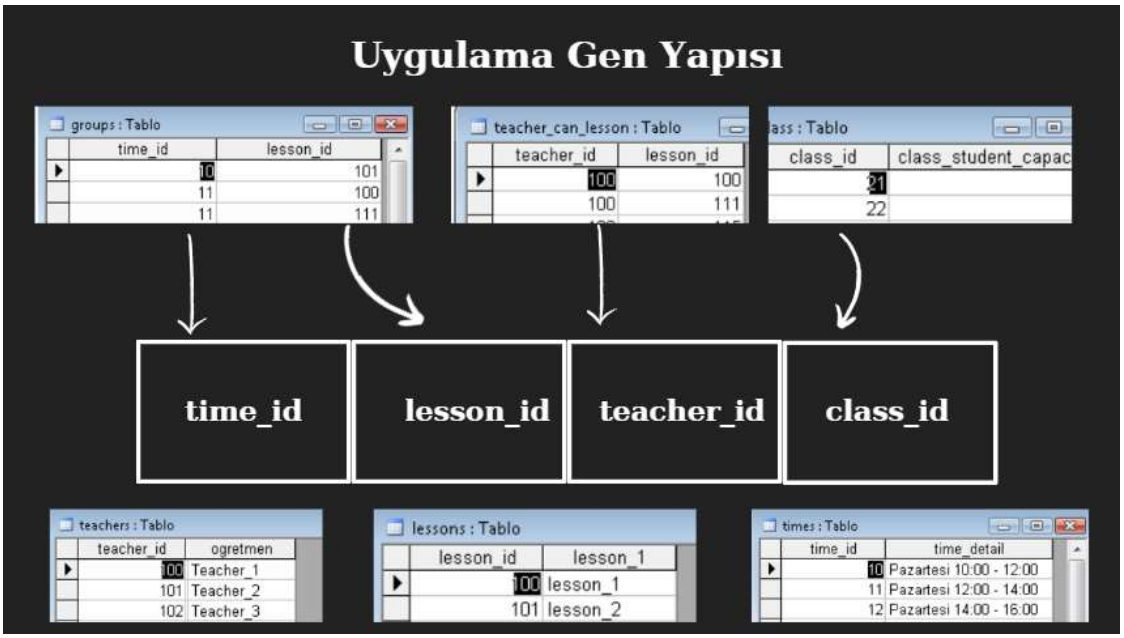
3.Yöntem

Öncelikle manuel olarak iterasyon, mutasyon oranı ve çaprazlama oranını girmek için tkinter kütüphanesini kullanarak bir arayüz oluşturdum.



Resim1.Yapmış olduğum arayüz

Makalede teacher, class, lessons gibi bilgiler görseldeki tablolarda tutulmaktaydı. Aynı zamanda kromozom yapısı aşağıda gösterildiği gibi oluşturulmuştur.



Resim2.Kullanılan tablolar

Ben bilgileri txt dosyalarında tutmayı tercih ettim. Öncelikle elimdeki txt dosyalarına verileri girdim. Verilerin tamamı makalede olmadığı için bazı değerleri tahmini olarak girmem gerekti.

Txt dosyalarındaki değerleri almak için iki adet fonksiyon tanımladım. İlk fonksiyon ile groups tablomdan ve class tablomdan veri çektim. Teacher_can_less tablosundaki lesson_id değeri ile groups tablosundaki lesson_id değerinin uyuşması gerektiğinden eşitliğini kontrol edebildiğim ikinci fonksiyon tanımladım.

Aldığım veriler ile kromozom yapımı oluşturdum.

```
# Kromozom oluşturma
chromosome = {
    "groups": selected_values_groups,
    "teacher_can_less": selected_values_teacher_can_less,
    "class": selected_values_class
}
```

Resim3. Kromozom yapısı

Daha sonra print_population fonksiyonu ile ekrana oluşturduğum popülasyonu bastırdım.

```
def print_population(population):
    for idx, individual in enumerate(population):
        teacher_id_value = individual["teacher_can_less"].get("teacher_id")
        lesson_id = individual["groups"].get("lesson_id")
        time_id = individual["groups"].get("time_id")
        class_id = individual["class"].get("class_id")
        print(f"Group ID:{lesson_id} Time ID:{time_id} Teacher ID: {teacher_id_value}  Class ID:{class_id}")
```

Resim4.print_population fonksiyonu

Değerlerimi ekrana bastırdım.

```
Group ID:125 Time ID:52 Teacher ID: 113  Class ID:23
Group ID:117 Time ID:53 Teacher ID: 107  Class ID:21
Group ID:122 Time ID:41 Teacher ID: 114  Class ID:34
Group ID:119 Time ID:62 Teacher ID: 108  Class ID:44
Group ID:122 Time ID:63 Teacher ID: 114  Class ID:23
Group ID:110 Time ID:73 Teacher ID: None  Class ID:63
Group ID:111 Time ID:73 Teacher ID: 110  Class ID:21
Group ID:112 Time ID:21 Teacher ID: 120  Class ID:63
Group ID:101 Time ID:54 Teacher ID: 103  Class ID:62
Group ID:102 Time ID:22 Teacher ID: 111  Class ID:52
```

Resim6.Popülasyon çıktısı

Daha sonra fitness değerim açılamayan grup sayısı olduğundan onu bulmak için teacher_id'si None olan değerleri tespit etmem gerekiyordu.

```
teacher_id_olmayan_kromozomlar = len([ind for ind, pop in enumerate(population) if
                                      not pop["teacher_can_less"].get("teacher_id")])
```

Resim6. teacher_id si olmayan kromozomların hesap edilme kodu

Yukarıdaki kod parçası ile popülasyon içindeki her bireyin "teacher_can_less" anahtarına sahip olup olmadığını kontrol edip ve eğer bu anahtar mevcut değilse ya da bu anahtarın "teacher_id" alt anahtarı mevcut değilse bunun yani öğretmenin olmadığı durumların sayılmasını sağladım. Ve aşağıdaki resimde görüldüğügibi fitness değerini yazdırdım.

```
Fitness: 4
Group ID:100 Time ID:12 Teacher ID: 100  Class ID:31
Group ID:112 Time ID:21 Teacher ID: 105  Class ID:54
Group ID:123 Time ID:64 Teacher ID: 108  Class ID:54
Group ID:110 Time ID:73 Teacher ID: None  Class ID:43
Group ID:102 Time ID:71 Teacher ID: 111  Class ID:62
```

Resim7. Çıktı

Daha sonra mutasyon işlemine başladım. Her birey için döngü oluşturarak, rastgele bir sayı üreterek mutasyonun gerçekleşip gerçekleşmeyeceğini belirledim. Mutasyon gerçekleşiyorsa, bireyin öğretmen bilgilerini güncellemek üzere dosyadan_deger_al_teacher_can_less fonksiyonunu kullandım. Eğer yeni öğretmen değerleri başarılı bir şekilde alınırsa ve bu değerlerin içinde "teacher_id" bulunuyorsa, bireyin "teacher_can_less" alanındaki "teacher_id" değeri bu yeni değerle güncellenmesini ancak yeni öğretmen değerleri alınamazsa veya alınan değerlerin içinde "teacher_id" bulunmuyorsa, tekrar öğretmen değerleri alınmaya çalışılmasını sağladım.

Bu mutasyon mekanizması, popülasyonun genetik çeşitliliğini artırarak adaptasyon yeteneklerini güçlendirmeye yönelik önemli bir mekanizmayı temsil eder. Bu sayede, genetik algoritmanın problem çözme sürecinde daha iyi sonuçlar elde etme olasılığı artar.

Her mutasyon adımı sonrasında tekrardan fitness değeri hesapladım ve eğer fitness değeri 0 olursa iterasyonların durdurulması için sys.exit() kullandım.

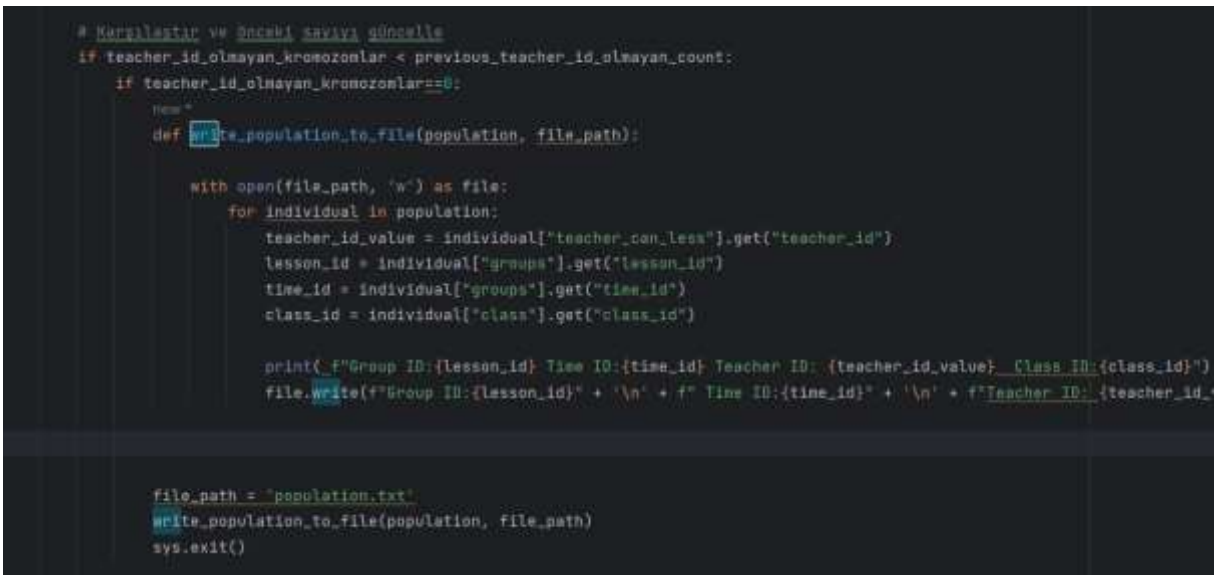
```
def mutation(population, mutation_rate):
    global previous_teacher_id_olmayan_count

    for individual in population:
        # Mutasyon oranına göre mutasyon gerçekleşip gerçekleşmeyeceğini kontrol et
        if random.random() < mutation_rate:
            new_teacher_values = dosyadan_deger_al_teacher_can_less(teacher_can_less_path,
                                                                    target_variables_teacher_can_less,
                                                                    individual["groups"]["lesson_id"])
            if new_teacher_values and "teacher_id" in new_teacher_values:
                # Yeni öğretmen ID'sini new_teacher_values'tan al
                new_teacher_id = new_teacher_values["teacher_id"]

                # Bireyin kromozomunu yeni öğretmen ID'si ile güncelle
                individual["teacher_can_less"]["teacher_id"] = new_teacher_id
```

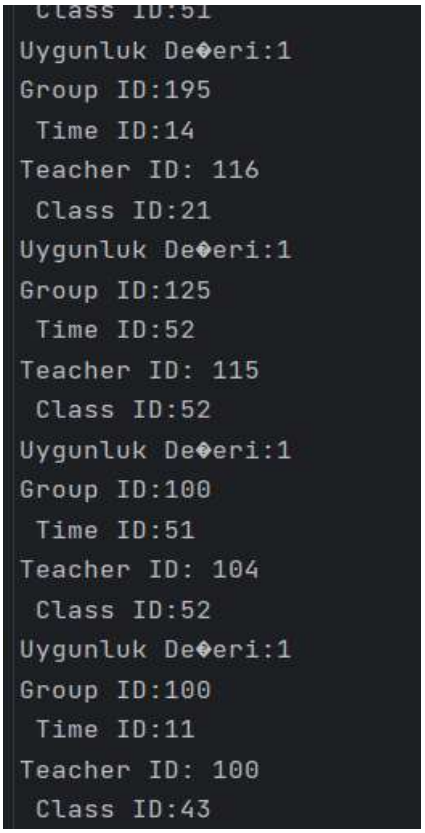
Resim8.Mutasyon Kodu

Mutasyon fonksiyonundan aldığım değerleri konsola yazdırdıktan sonra. Fitness değeri en düşün olanı dosyaya yazdırmak için hesaplanan fitness değerlerini karşılaştıran bir koşul satırı yazdım.



Resim9.Dosyaya yazdırma kodu

Yeni fitness değeri öncekinden küçük ise population.txt dosyasına yeni popülasyon yazdırılır. Eğer sıfıra eşit ise iterasyon sonlandırılır.



Resim10. Txt Dosyası Ekran Görüntüsü

Bir de tek noktalı çaprazlama yaptırmam gerekiyordu. Mutasyondan sonra yapılabilmesi için (eğer crossover_rate uygunsa) bir if else bloğu yazdım.

```
for iteration in range(num_iterations):
    if (iteration % 2 == 1):

      population=crossover(population, crossover_rate,previous_teacher_id_olmayan_count)
      print_population(population)

    else:

        mutationed = mutation(population, mutation_rate)
        print_population(population)
```

Resim11. Mutasyon ve Çaprazlama Fonksiyonları Çağırılması

Tek sayılı iterasyon numaralarında crossover çift numaralılarda mutasyon yapılmasını sağladım.

Crossover bölümünde kod, genetik algoritmanın çaprazlama aşamasını gerçekleştirmektedir. Çaprazlama, genetik materyalin iki ebeveyn birey arasında belirli bir olasılıkla değiştirilmesini ifade eder. İlk olarak, her bir birey için çaprazlama olup olmayacağı belirlenir. Eğer çaprazlama gerçekleşecekse, popülasyondan iki farklı ebeveyn seçilir. Ardından, belirli bir çaprazlama noktasına kadar olan genetik bilgiler birbirleriyle değiştirilerek iki yeni çocuk birey oluşturulur. Bu, genetik çeşitliliği artırarak potansiyel olarak daha iyi çözümlerin bulunmasına katkı sağlar. Çaprazlama işlemi gerçekleşmezse, yani belirli bir olasılık eşiğini geçmezse, ebeveyn bireyler doğrudan yeni popülasyona eklenir. Böylece, popülasyonun genetik materyali bir sonraki nesile taşınır.

Ayrıca, çaprazlama sonrasında mutasyon aşamasında yaptığım gibi mevcut popülasyondaki öğretmen ID'si olmayan kromozom sayısı hesapladım. Eğer önceki adımda belirlenen sayıdan daha düşük bir öğretmen ID'si olmayan kromozom sayısı elde edilirse, fitness değerini ve popülasyonun yeni halini txt dosyasına yazdırdım. Eğer öğretmeni None olan bir birey yoksa iterasyonları sonlandırdım.

```python
def crossover(population, crossover_rate,previous_teacher_id_olmayan_count):
    new_population = []  # Yeni popülasyonu tutacak liste

    for _ in range(len(population)):

        parent1_index = random.randint(0, len(population) - 1)
        parent2_index = random.randint(0, len(population) - 1)


        if random.random() < crossover_rate:

            child1, child2 = single_point_crossover(population[parent1_index], population[parent2_index])


            new_population.append(child1)
            new_population.append(child2)
        else:
            # Çaprazlama yapılmadıysa, ebeveynleri yeni popülasyona ekle
            new_population.append(population[parent1_index])
            new_population.append(population[parent2_index])
```

12.Crossover kod parçası

```python
crossover_point = 5

child1["groups"]["lesson_id"] = parent2["groups"]["lesson_id"][:crossover_point] + parent1["groups"]["lesson_id"][crossove
child2["groups"]["lesson_id"] = parent1["groups"]["lesson_id"][:crossover_point] + parent2["groups"]["lesson_id"][crossove
```

Resim13.Crossover kod parçası

Bu şekilde uygulamayı sonlandırmış oldum.

Farklı Mutasyon ve Çaprazlama Oranları Deneme Sonuçları

Deneme 1

İterasyon :10
Mutasyon Oranı:0.9
ÇaprazlamaOranı:0.1
Çözüm Kaçıncı İterasyonda Bulundu:4

Deneme 2

İterasyon :10
Mutasyon Oranı:0.9
ÇaprazlamaOranı:0.9
Çözüm Kaçıncı İterasyonda Bulundu:6

Deneme 3

İterasyon :10
Mutasyon Oranı:0.1
ÇaprazlamaOranı:0.9
Çözüm Kaçıncı İterasyonda Bulundu:5

Düşük iterasyon sayılarında sonuca ulaşıldığı için (datanında az olmasından kaynaklı) iterasyon sayılarını düşük tuttum.

4.Sonuçlar ve Tartışma

Genetik algoritmaların bu tür problemlerde kullanılması, esneklik sağlaması ve çözüm süresi açısından avantajlar sunmasıyla dikkat çekmektedir. Gerçekleştirilen testlerde, farklı popülasyon büyüklükleri ile çaprazlama ve mutasyon oranlarının çeşitli senaryoları test edilmiş ve bu oranların çizelge probleminin etkili bir şekilde çözülmesinde önemli bir rol oynadığı gözlemlenmiştir.

Projeyi değerlendirdiğimizde, çizelgeleme probleminin genetik algoritmalar için görece basit bir problem olduğu sonucuna ulaşılmıştır. Ancak, projede öğretmenlerin zaman tercihlerinin göz ardı edilmiş olması ve grupların zaman aralıklarının her zaman eşit olması gibi eksiklikler belirlenmiştir. Bu eksikliklerin, uygulamanın tam kapsamlı bir çözüm sunma potansiyelini kısıtladığı görülmüştür.

Bu çalışmanın gelecekteki geliştirmeler için bir temel oluşturabileceği düşünülmektedir. Özellikle öğretmenlerin zaman tercihleri ve grupların farklı zaman aralıklarında yer alabilmesi gibi faktörlerin dahil edilmesi, genetik algoritmaların çözüm kapasitesini artırabilir ve daha gerçekçi çözümler elde edilmesine olanak tanıyabilir.