YAPAY ARI KOLONİSİ ALGORİTMASI

Dr. Öğr. Üyesi Mahir Kaya
TOKAT GAZİOSMANPAŞA ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
TOGU-CENG

Yapay Arı Koloni Algoritması

- Doğada var olan zeki davranışlar içeren süreçlerin incelenmesi araştırmacıları yeni optimizasyon metotları geliştirmeye sevk etmiştir. Arıların yiyecek arama davranışı modellenerek Yapay Arı Kolonisi (Artificial Bee Colony, ABC) algoritması geliştirilmiştir.
- Yapay arı kolonisi optimizasyon algoritması Derviş Karaboğa tarafından sürekli optimizasyon problemlerinin çözümü için önerilmiştir.

Yapay Arı Koloni Algoritması

- ABC algoritması bal arılarının yiyecek arama ve kaynakların pozisyon bilgilerini paylaşmak için kullandığı sallanım dansını temel alır.
- Optimizasyon probleminin olası çözümleri yiyecek kaynaklarının pozisyonları olarak düşünülür ve görevlerine göre üç tip arı yiyecek kaynaklarının keşfi ve iyileştirilmesi için çalışır.
- Bunlar işçi, gözcü ve kaşif arılardır.

Yapay Arı Koloni Algoritması

- Arama sürecinin başlangıcında arı popülasyonun yarısı kâşif arıdır. Bu kâşif arılara yiyecek kaynağı pozisyonları rastgele atandıktan sonra işçi arı adını almaktadırlar.
- İşçi arılar iterasyonlar boyunca daha iyi yiyecek kaynakları bulmaya çalışırlar.

Yapay Arı Koloni Algoritması

- Popülasyonun diğer yarısı ise gözcü arılardan oluşur. Bu arılar kovanda, işçi arıların yiyecek kaynakları hakkındaki pozisyon bilgisini paylaşmasını beklerler.
- Kaynakların bilgisini alan gözcü arılar yiyecek kaynağının kalitesine bağlı olarak bir işçi arıya atanmış olan kaynağı iyileştirmeye çalışır.
- Eğer herhangi bir yiyecek kaynağı belirli bir zaman süresince iyileştirilemezse bu kaynağın işçi arısı kâşif arı olur. Rastgele bir yiyecek aramasına çıkan kâşif arı yiyecek kaynağı atandıktan sonra tekrar işçi arı olur.

Yapay Arı Koloni Algoritması

- Yiyecek kaynaklarının yerleri optimizasyon problemine ait olası çözümlere ve kaynakların nektar miktarları da o kaynaklarla ilgili çözümlerin kalitesine (uygunluk) karşılık gelmektedir.
- Dolayısıyla ABC optimizasyon algoritması en fazla nektara sahip kaynağın yerini bulmaya çalışarak uzaydaki çözümlerden problemin minimumunu yada maksimumunu veren noktayı (çözümü) bulmaya çalışmaktadır.

ABC'nin Temel Özellikleri

1. Oldukça esnek ve basittir.
2. Gerçek yiyecek arayıcı arıların davranışlarına oldukça yakın şekilde simule eder.
3. Sürü zekâsına dayalı bir algoritmadır.
4. Nümerik problemler için geliştirilmiştir ama ayrık problemler içinde kullanılabilir.
5. Oldukça az kontrol parametresine sahiptir
6. Kâşif Arılar tarafından gerçekleştirilen küresel ve işçi ve gözcü arılar tarafından gerçekleştirilen global araştırma kabiliyetine sahiptir ve ikisi paralel yürütülmektedir.

ABC algoritmasının temel adımları:

Adım 1. Başlangıç yiyecek kaynağı bölgelerinin üretilmesi.

REPEAT

Adım 2. İşçi arıların yiyecek kaynağı bölgelerine gönderilmesi

Adım 3. Olasılıksal seleksiyonda kullanılacak olasılık değerlerinin işçi arılardan gelen bilgiye göre hesaplanması

Adım 4. Gözcü arıların olasılık değerlerine göre yiyecek kaynağı bölgesi seçmeleri

Adım 5. Bırakılacak kaynakların bırakılışı ve kaşif arı üretimi

UNTIL (çevrim sayısı=Maximum çevrim sayısı)

Başlangıç Yiyecek Kaynağı Bölgelerinin Üretilmesi

Arama uzayını yiyecek kaynaklarını içeren kovan çevresi olarak düşünebiliriz. Bu durumda algoritma arama uzayındaki çözümlere karşılık gelen rastgele yiyecek kaynağı yerleri üreterek çalışmaya başlamaktadır.

Rastgele yer üretme sureci her bir parametrelerinin alt ve üst sınırları arasında rastgele değer üreterek gerçekleşir.

$$x_{ij} = x_{\min j} + rand(0,1) * (x_{\max j} - x_{\min j})$$

Burada i=1… SN, J=1…D ve SN yiyecek kaynağı sayısı ve D is optimize edilecek parametre sayısıdır. Xmin j parametrelerin alt sınırıdır.

İşçi Arı Fazı

Her bir kaynağın bir işçi arısı vardır. Dolayısıyla yiyecek kaynakların sayısı işçi arıların sayısına eşittir.

İşçi arı çalıştığı yiyecek kaynağı komşuluğunda yeni bir yiyecek kaynağı belirler ve bunun kalitesini değerlendirir.

Yeni kaynak daha iyi ise bu yeni kaynağı hafızasına alır.

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$$

j : i. kaynağın rasgele seçilen tek bir parametresi

k: xi kaynağının rastgele seçilen bir komşu indis numarası

Vij: güncellenen kaynak

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$$

Eşitlikten de görüldüğü gibi $x_{i,j}$ ve $x_{k,j}$ arasındaki fark azaldıkça yani çözümler birbirine benzedikçe $x_{i,j}$ parametresindeki değişim miktarı da azalacaktır. Böylece bölgesel optimal çözüme yaklaştıkça değişim miktarı da adaptif olarak azalacaktır.

Güncellenme sonucu bulunan v_i parametre vektörü yeni bir kaynağa temsil etmekte ve bunun kalitesi hesaplanarak bir uygunluk değeri atanmaktadır.

$$fitness_i = \begin{cases} 1/(1+f_i) & f_i \geq 0 \\ 1/abs(f_i) & f_i < 0 \end{cases}$$

- Yeni Bulunan v_i çözümünün uygunluk değeri (yiyecek kaynağının kalitesi) eski çözümünkinden (x_i) daha iyi ise işçi arı hafızasından eski kaynağın yerini silerek v_i kaynağının yerini hafızaya alır.
- Aksi takdirde işçi arı x_i kaynağına gitmeye devam eder ve x_i çözümü geliştirilemediği için x_i kaynağı ile ilgili geliştirememe sayacı (trial) bir artar, geliştirdiği durumda ise sayaç sıfırlanır.

Gözcü Arı Fazı

- Tüm işçi arılar bir çevrimde araştırmalarını tamamladıktan sonra kovana dönüp buldukları kaynakların nektar miktarları (kalitesi) ile ilgili gözcü arılara bilgi aktarırlar.
- Bir gözcü arı dans aracılığıyla paylaşılan bilgiden faydalanılarak yiyecek kaynaklarının nektar miktarları ile orantılı bir olasılıkla bir bölge(kaynak) seçer.
- Olasılıksal seçme işlemi, algoritmada nektar miktarlarına karşılık gelen uygunluk değerleri uygulanarak yapılmaktadır.

Uygunluk değerine bağlı seçme işlemi rulet tekerliği seçme yöntemiyle gerçekleşir. Tekerlekteki her bir dilimin açısı uygunluk değeri toplamına oranı o kaynağın diğer kaynaklara göre seçilme olasılığı olduğunu vermektedir.

$$p_i = \frac{fitness_i}{\sum_{i=1}^{SN} fitness_i}$$

fitness kaynağın kalitesini SN işçi arı sayısını göstermektedir. Bu olasılık hesaplama işlemine göre bir kaynağın nektar miktarı arttıkça (uygunluk değeri arttıkça) bu kaynak bölgesini seçecek gözcü arı sayısı da artacaktır.

- Rulet tekerleğine göre secim işleminde her bir kaynak için [0.1] aralığında rastgele sayı üretilen ve p_i değeri bu üretilen sayıdan büyükse işçi arılar gibi gözcü arı da komşuluğunda bir yiyecek kaynağını kullanarak yeni çözüm üretir.
- Yeni çözüm daha iyi ise eski çözüm yerine bu çözüm alınır ve çözüm geliştirememe sayacı (trial) sıfırlanır. Eski çözümün uygunluğu daha iyi ise bu çözüm muhafaza edilir ve geliştirememe sayacı (trial) bir artırılır.
- Bu süreç,tüm gözcü arılar yiyecek kaynağı bölgelerine dağılana kadar devam eder.

Kaşif Arı Fazı

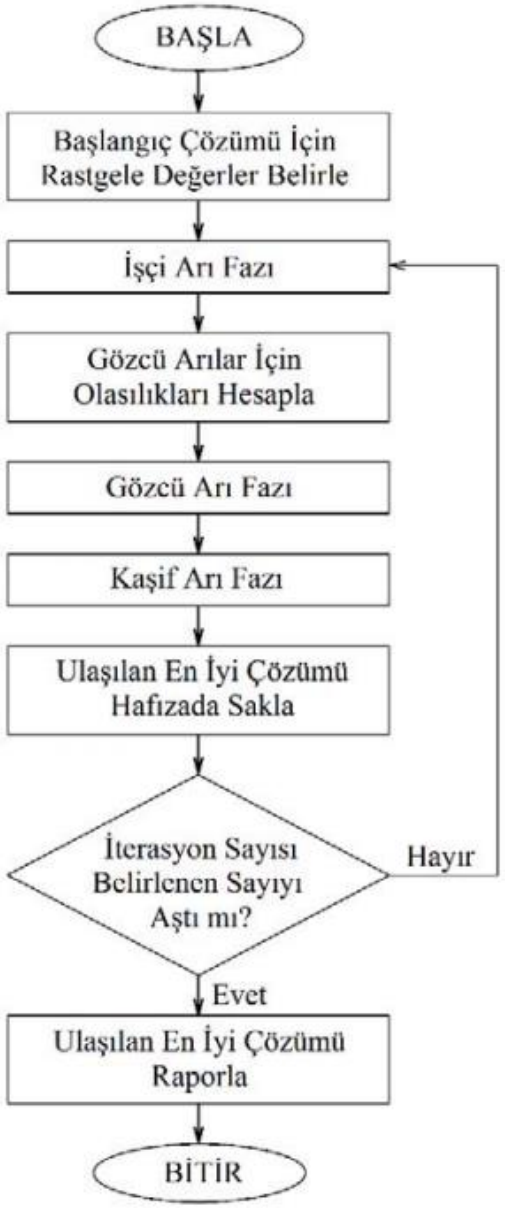
- Bir çevrim sonunda tüm işçi ve gözcü arılar arama süreçlerini tamamladıktan sonra çözüm geliştirememe sayaçları (trial) kontrol edilir.
- Bir arının bir kayaktan faydalanıp faydalanmadığı, yani gidip geldiği kaynağın nektarının tükenip tükenmediği çözüm geliştirememe sayaçları (trial) aracılığıyla bilinir.

Bir kaynak için çözüm geliştirememe sayacı (trial) belli bir limit değerinin üzerindeyse, artık bu kaynağın görevli arısının tükenmiş olan o çözümü bırakıp kendisi için başka bir çözüm araması gerekir.
Bu da biten kaynakla ilişkili olan işçi arının kâşif arı olması anlamına gelmektedir.

Kâşif arı haline gelen arı için rastgele çözüm arama sureci başlar.

Kaynağın terk ettiğinin belirlenmesi için kullanılan eşik değeri ABC algoritmasının önemli bir kontrol parametresidir ve “limit” olarak adlandırılmaktadır.

Temel ABC algoritmasında her çevrimde sadece kâşif arının çıkmasına izin verilir.


BAŞLA
Başlangıç Çözümü İçin Rastgele Değerler Belirle
İşçi Arı Fazı
Gözcü Arılar İçin Olasılıkları Hesapla
Gözcü Arı Fazı
Kaşif Arı Fazı
Ulaşılan En İyi Çözümü Hafızada Sakla
İterasyon Sayısı Belirlenen Sayıyı Aştı mı?
Hayır
Evet
Ulaşılan En İyi Çözümü Raporla
BİTİR

```python
import numpy as np
import matplotlib.pyplot as plt

#Sphere fonksiyonu
def fun(x):
    return np.array([sum(np.power(x,2))])

#Fitness değerini hesaplayan fonksiyon
#Yiyeceğin ne kadar kaliteli olduğunun bilgisi
def calculateFitness(fObjV):
    fFitness = np.zeros(len(fObjV))
    ind = np.where(fObjV>0)
    fFitness[ind] = 1 / (fObjV[ind] + 1)
    ind = np.where(fObjV < 0)
    fFitness[ind] = 1 + abs(fObjV[ind])
    return fFitness

#Problemin çözüm uzayının istenen aralıklarda olmasını sağlayan fonksiyon
def repair(x, lb, ub):
    for i in range(len(x)):
        if x[i] > ub:
            x[i] = ub
        if x[i] < lb:
            x[i] = lb
    return x
```

```
#Kolonideki toplam arı sayısı
NP = 20

#Yiyecek sayısı toplam koloninin yarısı,#yiyeccek sayısı kadar işçi arı var
FoodNumber = NP // 2

#Hangi kaynağın işçi arısının kaşif arı olacağına karar veren parametre
limit = 100

#Algoritma kaç iterasyon çalışacak
maxCycle = 2500

D = 100          #Problemin boyutu
ub = 100         #Çözüm uzayının max limiti
lb= -100         #Çözüm uzayının min limiti

#Yiyecek kaynağı [-100,100] aralığında rasgele olarak belirleniyor.
Foods = lb + np.random.ranf([FoodNumber,D]) * (ub-lb)

#Yiyeceklerin temsil ettiği çözümün amaç fonksiyonu değerleri
ObjVal = np.zeros(FoodNumber)
for i in range(FoodNumber):
    ObjVal[i]=fun(Foods[i,:])

#Fitness fonksiyonu yiyecek kaynağının kalitesi hesaplanır
#Amaç fonksiyonun minimum olduğu durumlar daha kaliteli sonuç üretir.
Fitness = calculateFitness(ObjVal)

#Hangi kaynağın işçi arısının kaşif arı olacağınının kontrolü
trial = np.zeros(FoodNumber)

#Rastgele oluşturulan çözümlerden en iyi çözüm uzayı ve değeri
#GlobalParams ve GlobalMin olarak tutuluyor
BestInd = np.where(ObjVal==min(ObjVal))
BestInd = BestInd[-1]
GlobalMin = ObjVal[BestInd]
GlobalParams = Foods[BestInd,:]

objit = list()
objit.append(GlobalMin)
```

```
iteration=1;
while (iteration <= maxCycle):
    #################
    # İşçi arı fazı #
    #################
    for i in range(FoodNumber):

        Param2Change = np.random.randint(0,D)

        neighbour = np.random.randint(0,FoodNumber)

        while neighbour == i:
            neighbour = np.random.randint(0,FoodNumber)

        sol = Foods[i,:].copy()
        sol[Param2Change] = Foods[i,Param2Change] \
            + (Foods[i,Param2Change]-Foods[neighbour,Param2Change]) \
            * (np.random.rand()-0.5)*2

        sol = repair(sol,lb,ub)
        ObjValSol = fun(sol)
        FitnessSol = calculateFitness(ObjValSol)

        if FitnessSol > Fitness[i]:
            Foods[i,:] = sol
            Fitness[i] = FitnessSol
            ObjVal[i] = ObjValSol
            trial[i] = 0
        else:
            trial[i]=trial[i] + 1
```

İşçi arı sayısı, besin kaynağına eşittir. Yani her işçi arı bir besin kaynağına gidiyor kabul edilir.

kaynağın rasgele seçilen tek bir parametresi

kaynağının rastgele seçilen bir komşu indis numarası

İşçi arı çalıştığı yiyecek kaynağı komşuluğunda yeni bir yiyecek kaynağı belirler.

Problemin çözüm uzayının max ve min limitler arasında olması sağlanır. Repair fonksiyonu

fitness : kaynağın kalitesi

İşçi arı yeni yiyecek kaynağının kalitesini değerlendirir. Yeni kaynak daha iyi ise bu yeni kaynağı hafızasına alır.

kaynakta iyileştirme yapıp yapılmadığının kontrolü,
Eğer iyileştirme olmuşsa ilgili kontrol (trial) 0 olur. Olmamışsa belirli bir limitten sonra ilgili işçi arı kaşif arıya dönüşerek rastgele yiyecek kaynağı belirler.

```
99  ##################
100 # Gözcü arı fazı #
101 ##################
102
103 prob= Fitness / sum(Fitness)
104
105 i=0
106 t=-1
107 while t < FoodNumber:
108     if np.random.rand()< prob[i]:
109         t = t + 1
110         Param2Change = np.random.randint(0,D)
111         neighbour = np.random.randint(0,FoodNumber)
112
113         while neighbour==i:
114             neighbour = np.random.randint(0,FoodNumber)
115
116         sol = Foods[i,:].copy()
117         sol[Param2Change]=Foods[i,Param2Change] \
118             + (Foods[i,Param2Change]-Foods[neighbour,Param2Change]) \
119             * (np.random.rand()-0.5)*2
120
121         sol = repair(sol,-100,100)
122         ObjValSol = fun(sol)
123         FitnessSol = calculateFitness(ObjValSol)
124
125         if FitnessSol > Fitness[i]:
126             Foods[i,:] = sol
127             Fitness[i] = FitnessSol
128             ObjVal[i] = ObjValSol
129             trial[i] = 0
130         else:
131             trial[i]=trial[i] + 1
132
133     i=i+1
134     if i==FoodNumber:
135         i=0;
```

$$p_i = \frac{fitness_i}{\sum_{i=1}^{SN} fitness_i}$$

her bir kaynak için [0.1] aralığında rastgele sayı üretilir, olasılık değeri (prob[i]) bu üretilen sayıdan büyükse işçi arılar gibi gözcü arı da komşuluğunda bir yiyecek kaynağını kullanarak yeni çözüm üretir.

kaynağın rasgele seçilen tek bir parametresi

kaynağının rastgele seçilen bir komşu indis numarası

Gözcü arı çalıştığı yiyecek kaynağı komşuluğunda yeni bir yiyecek kaynağı belirler.

Problemin çözüm uzayının max ve min limitler arasında olması sağlanır. Repair fonksiyonu

fitness : kaynağın kalitesi

Gözcü arı yeni yiyecek kaynağının kalitesini değerlendirir. Yeni kaynak daha iyi ise bu yeni kaynağı hafızasına alır.

kaynakta iyileştirme yapıp yapılmadığının kontrolü,
Eğer iyileştirme olmuşsa ilgili kontrol (trial) 0 olur. Olmamışsa belirli bir limitten sonra ilgili işçi arı kaşif arıya dönüşerek rastgele yiyecek kaynağı belirler.

```
137     ind=np.where(ObjVal==min(ObjVal))
138     ind=ind[-1]
139     if ObjVal[ind] < GlobalMin:
140         GlobalMin = ObjVal[ind]
141         GlobalParams = Foods[ind,:]
142         objit.append(GlobalMin)
143 
144     ###################
145     # Kaşif arı fazı #
146     ###################
147     ind = np.where(trial==max(trial))[0]
148     ind = ind[-1]
149     if trial[ind]>limit:
150         trial[ind]=0
151         sol = np.random.ranf([D]) * (ub-lb) + lb
152         ObjValSol = fun(sol)
153         FitnessSol = calculateFitness(ObjValSol)
154         Foods[ind,:]=sol
155         Fitness[ind]=FitnessSol
156         ObjVal[ind]=ObjValSol
157 
158     iteration = iteration + 1
159 
160     print("iterasyon :{}, obj :{}".format(iteration,GlobalMin))
161 
162 plt.plot(objit)
163 plt.xlabel("iterasyon")
164 plt.show()
```

En iyi yiyecek kaynağı, yani en optimum çözüm uzayı hafızaya alınır.

Bir çevrim sonunda tüm işçi ve gözcü arılar arama süreçlerini tamamladıktan sonra çözüm geliştirememe sayaçları (trial) kontrol edilir. Bir arının bir kayaktan faydalanıp faydalanmadığı, yani gidip geldiği kaynağın nektarının tükenip tükenmediği çözüm geliştirememe sayaçları (trial) aracılığıyla bilinir.

Bir kaynak için çözüm geliştirememe sayacı (trial) belli bir limit değerinin üzerindeyse, artık bu kaynağın görevli arısının tükenmiş olan o çözümü bırakıp kendisi için başka bir çözüm araması gerekir.

```
iterasyon :2482, obj :[1.12025492e-05]
iterasyon :2483, obj :[1.12025492e-05]
iterasyon :2484, obj :[1.1201817e-05]
iterasyon :2485, obj :[1.1201817e-05]
iterasyon :2486, obj :[1.12018058e-05]
iterasyon :2487, obj :[1.12018058e-05]
iterasyon :2488, obj :[1.12018058e-05]
iterasyon :2489, obj :[1.12018058e-05]
iterasyon :2490, obj :[1.12018058e-05]
iterasyon :2491, obj :[1.12018058e-05]
iterasyon :2492, obj :[1.12018058e-05]
iterasyon :2493, obj :[1.10287508e-05]
iterasyon :2494, obj :[1.10287508e-05]
iterasyon :2495, obj :[1.10287507e-05]
iterasyon :2496, obj :[1.10281941e-05]
iterasyon :2497, obj :[1.10281941e-05]
iterasyon :2498, obj :[1.10279412e-05]
iterasyon :2499, obj :[1.10279412e-05]
iterasyon :2500, obj :[1.102776e-05]
```