KARINCA KOLONİSİ ALGORİTMASI

Karınca Koloni Algoritması

Bilim adamları, böcek davranışlarını inceleyerek başarılı optimizasyon algoritmaları geliştirmişlerdir.

Bu teknikler birçok bilimsel alanda ve mühendislik problemlerinde başarıyla uygulanmıştır.

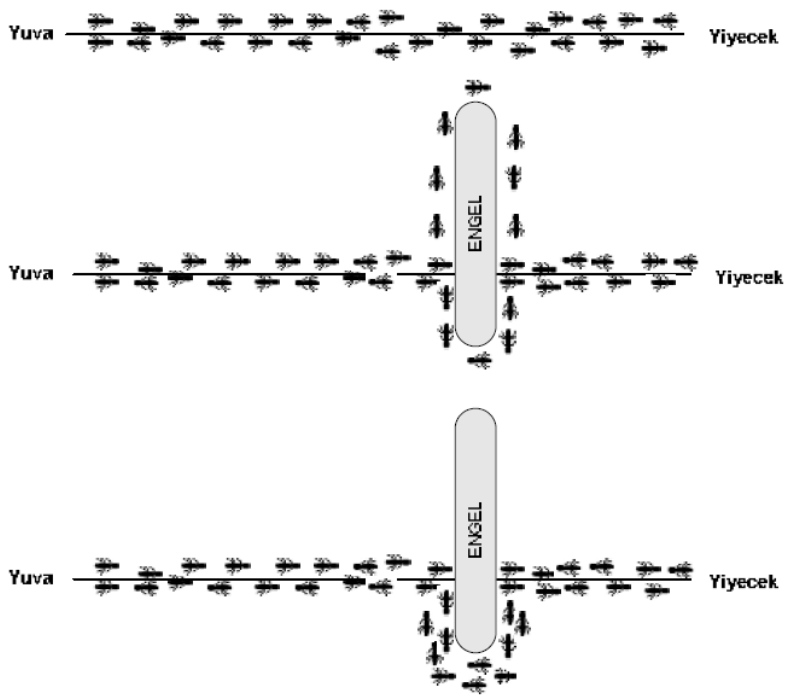
Teknikler, statik problemlerdeki yüksek performanslarına ilaveten, dinamik özellik gösteren problemlerde de yüksek derecede esnekliğe sahiptirler.

Karınca algoritmaları ilk olarak Dorigo ve meslektaşları tarafından; gezgin satıcı problemi (GSP) ve kuadratik atama (QAP) gibi zor optimizasyon problemlerinin çözümü için geliştirilmiştir.

Optimizasyon problemlerinin çözümü amacıyla karınca algoritmaları üzerine birçok çalışma devam etmektedir.

Bu çalışma alanlarından bazıları, komünikasyon ağlarının belirlenmesi, grafik renklendirme, iş çizelgeleme vs.dir

Karınca Koloni Algoritması



Karıncalar, yiyecek kaynaklarından yuvalarına en kısa yolu görme duyularını kullanmadan bulma yeteneğine sahiptirler. Aynı zamanda, çevredeki değişime adapte olma yetenekleri vardır. Dış etkenler sonucu takip ettikleri mevcut yol artık en kısa yol değilse, yeni en kısa yolu bulabilmektedirler.

Şekilde görüldüğü gibi karıncalar, başlangıçta düz bir hattı takip etmekte ve bu esnada feromon olarak adlandırılan bir maddeyi yol güzergahına bırakarak kendilerinden sonra gelen karıncaların yollarını bulmalarını kolaylaştırmaktadırlar.

Karınca Koloni Algoritması

- Önlerine bir engel konulduğunda feromonları takip edemediklerinden, karıncalar gidebilecekleri iki yoldan birini öncelikle rastsal olarak seçmektedirler.
- Kısa olan yoldan birim zamandaki geçiş daha fazla olacağından bırakılan feromon miktarı da daha fazla olur.
- Buna bağlı olarak, zaman içerisinde kısa olan yolu tercih eden karıncaların sayısında artış olur. Belli bir süre sonra tüm karıncalar kısa yolu tercih ederler.
- Başta rastsal hareket eden karıncaların izleri kontrol ederek yüksek olasılılıkla izlerin yoğun olduğu yönü takip etmesi otokatalitik bir davranış şeklidir ve karıncaların karşılıklı etkileşiminde sinerjik bir etki vardır.
- Algoritma, karınca kolonilerinden esinlenerek geliştirildiğinden sisteme, karınca sistemi (KS), algoritma ise karınca kolonileri algoritması (KKA) olarak adlandırılır.
- Karınca kolonileri optimizasyon problemlerinde kullanılır.
- Karınca sistemindeki karıncalar doğal karıncalardan farklıdır. Hafızaya sahiptirler, tamamen kör değildirler ve zamanın kesikli olduğu bir çevrede yaşarlar.

Karınca Koloni Algoritması

Yapay karıncalardan oluşan Karınca Kolonisi Algoritması, yapay feromon izlerinin güncelleştirilmesiyle tekrarlanan bir yapıya sahiptir.

Algoritmanın çalışma sürecinde, karıncalar tarafından güncellenen feromon izleriyle iyi bir çözümün bulunması için bilgi oluşturulmakta ve her iterasyonda bu bilgiler güncellenmektedir.

Şekilde görüldüğü gibi karıncalar, başlangıçta düz bir hattı takip etmekte ve bu esnada feromon olarak adlandırılan bir maddeyi yol güzergahına bırakarak kendilerinden sonra gelen karıncaların yollarını bulmalarını kolaylaştırmaktadırlar.

Adım 1 : Başlangıç feromon değerleri belirlenir.
Adım 2 : Karıncalar her düğüme rastsal olarak yerleştirilir.
Adım 3: Her karınca, sonraki şehri denklemde verilen lokal arama olasılığına bağlı olarak seçmek suretiyle turunu tamamlar.
Adım 4: Her karınca tarafından katedilen yolların uzunluğunu hesaplanır ve lokal feromon güncellemesi yapılır.
Adım 5: En iyi çözüm hesaplanır ve global feromon yenilemesinde kullanılır.
Adım 6: Maksimum iterasyon sayısı yada yeterlilik kriteri sağlanana kadar Adım 2' ye gidilir.

Karınca Parametrelerin belirlenmesi

- KKO'da ilk aşama, parametrelerin belirlenmesidir. Parametrelerin üzerinde çalışılan probleme uygun değerler alacak şekilde belirlenmesi sonuçları doğrudan etkilemektedir. Bu algoritma için bazı parametreler aşağıdaki gibidir.

α (alpha)	Sonraki şehri seçim işleminde feromon izi miktarının ne kadar dikkate alınacağını belirleyen parametredir.
β (beta)	Seçim işleminde şehirlerarası mesafe miktarının başka bir dikkate alınacağını belirleyen parametredir.
ρ (rho)	Her iterasyonda buharlaşacak olan feromon izi miktarını belirler.
m	Algoritmada kullanılan karınca sayısıdır.
Sonlandırma Koşulu	Algoritmanın hangi koşullarda sonlanacağını belirler. Örneğin: Belirli bir iterasyon sayısı, belirli süre, ulaşılmak istenen çözüm kalitesi olabilir.

Feromon izlerinin ilk değerlerinin oluşturulması

Feromon izlerinin ilk değerleri atanırken, tüm yollar için eşit atanır ve değerleri sıfıra yakındır.

Sonraki iterasyonlarda değerleri karıncalar tarafından güncellenerek farklılaşacak ve iyi çözümlerin üzerindeki izler yoğunlaşacaktır.

Çözümlerin oluşturulması

Belirli sayıda karınca, arama uzayında kendi çözümlerini oluştururlar.

Her bir karınca kendi çözümünü oluştururken sıradaki çözüm bileşenini rastsal bir hesaplamayla seçer. Bu seçime feromon izleri ve yol matrisleri doğrudan etki eder.

Alternatif çözüm bileşenleri arasında, yüksek feromon izi ve mesafe yönünden kısa olan yolların seçilme ihtimali artacaktır.

Geçiş Kuralı

- Her iterasyonda m adet karınca kendi çözümlerini oluşturup ardından feromon değerlerini bulunan çözüm kalitelerine göre güncellemektedir.
- Güncellenen bu feromon değerleri sonraki nesle (iterasyon) aktarılmakta ve böylece çözümler iyileştirilmektedir.
- Bu algoritmada yapay karıncalar, gezgin satıcı probleminin ağ yapısı üzerinde bir düğümden başka bir düğüme hareket etmek için olasılıklı bir geçiş kuralı kullanmaktadırlar.
- Başlangıçta her bir karınca ağ üzerinde rastgele bir düğüm üzerine yerleştirilir. Daha sonra her karınca daha önce ziyaret etmediği düğümlere hareket ederek bütün düğümleri ziyaret eder ve bütün düğümleri içeren bir sonuç içermiş olur.

Geçiş Kuralı

K karıncasının, i sonraki düğümü seçerken seçenekler arasında j düğümlerine ait geçiş olasılıkları aşağıdaki denklemle bulunur.

$$P_{ij}^{k}(t) = \frac{[\tau_{ij}t]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{u \in J_i^k}[\tau_{iu}t]^{\alpha}[\eta_{iu}]^{\beta}}$$

Burada *t* değeri döngü indeksidir (kesikli zaman değişkeni).

τij , feromon izi miktarıdır.

ηij ise (*i, j*) bağı üzerindeki sezgisel bilgidir başka bir değişle yol bilgisidir yani yapay karıncanın *i* düğümünden *j* düğümünü görebilme ölçüsüdür. Genellikle gezgin satıcı problemlerinde (*i, j*) yolu uzunluğunun tersi olarak hesaplanır. Bu bilgi probleme özgü bir kriterle değiştirilebilir. j_k^i ise *i* düğümü üzerinde bulunan *k* karıncasının gidebileceği, hala ziyaret etmediği düğümler kümesidir.

α ve β ise feromon iz miktarı ile sezgisel bilginin olasılık değeri belirlemedeki etkilerini belirleyen parametrelerdir.

Geçiş kuralı, bir sonraki düğümün yukarıdaki denklem ile hesaplanan olasılıklara uygun olarak seçimini sağlamaktadır.

Denkleme göre yol uzunluğu az ve feromon miktarının daha yoğun olduğu yolların seçilme olasılığı daha fazla olacaktır.

Feromon güncellemesi

Tüm karıncalar turlarını tamamladıktan sonra feromon miktarları güncellenmektedir.

İlk olarak tüm yollardaki feromonlar, belirlenen oranda (buharlaşma oranı) buharlaştırılmaktadır.

Daha sonra karıncaların geçiş yapmış oldukları yollardaki feromon miktarları, o yolu kullanan karıncanın toplam yol uzunluğuyla ters orantılı olarak arttırılmaktadır.

Böylelikle daha kısa yola sahip karıncaların kullandıkları yollardaki feromon miktarları daha fazla artış göstermektedir.

Feromon güncellemesi iki yolla yapılabilmektedir. Bunlar Lokal feromon güncellemesi ve Global Feromon güncellemesidir.

Lokal Feromon Güncellemesi

Tüm karıncalar turlarını tamamladıktan sonra, eski feromon miktarları belli bir oranda buharlaştırılır, her bir karıncanın turu boyunca geçiş yapmış olduğu yollarda belli bir miktarda feromon artışı sağlanır.

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t+1)$$

$\Delta\tau_{ij}^{k}(t+1)$, aşağıdaki formülle hesaplanır:

$\tau_{ij}(t)$, t iterasyonuna kadar biriken feromon düzeyi, $\Delta\tau_{ij}^{k}(t+1)$, t iterasyonundaki feromon düzeyi ve ρ $(0 \le \rho \le 1)$, feromon buharlaşma parametresi olmak üzere lokal feromon düzeyi aşağıdaki formülle hesaplanır:

$$\Delta\tau_{ij}^{k}(t+1) = \begin{cases} 1/L^{k}(t+1) & \text{k karıncası (i, j)} \\ & \text{yolunu kullanmışsa,} \\ 0 & \text{diğer durumlarda} \end{cases}$$

Lk(t+1) karıncanın toplam tur uzunluğudur. Lokal feromon güncellemesi, turları dinamik olarak değiştirerek geçiş yapılan yolları cazip hale getirir.

Karıncalar değişen feromon miktarlarına bağlı olarak her iterasyonda turlarını da değişmektedirler. Böylelikle sürekli olarak kısa turları bulmak amaçlanmaktadır.

Global feromon güncellemesi

KKA'da global feromon güncellemesi geçerli adımdaki en iyi sonuca sahip karıncanın izlediği yolun feromon düzeyinin arttırılmasından oluşur ve iterasyonlarda bulunan en iyi sonuçların belli bir oranda ileriki iterasyonlara aktarılmasını sağlar.

Global feromon güncellemesi lokal'e benzer. Aşağıdaki denkleme göre yapılır.

$$\tau_{ij}(t+1) = (1 - \rho)\,\tau_{ij}(t) + \Delta\tau_{ij}^{k}(t+1)$$

$$\Delta\tau_{ij}(t+1) = \begin{cases} \dfrac{1}{L_{best}(t+1)} & (i, j) \text{ en iyi tura ait ise} \\ 0 & \text{diğer durumlarda} \end{cases}$$

$L_{best}(t+1)$ geçerli iterasyonda bulunan en iyi turun uzunluğudur.

Optimum Karınca Sayısı

Karınca sayısının arttırılması çözümde iyileşmeye neden olur.

Fakat hesaplamaları arttırdığı için karınca sayısının fazla arttırılması işlem zamanlarının uzamasına neden olur.

GSP problemlerinde yapılan denemeler sonucunda karınca sayısının şehir sayısına eşit seçilmesinin uygun olacağı sonucuna varılmıştır.

Karınca sayısı, problem büyüklüğüne ve uygulama alanına bağlı olarak değişir.

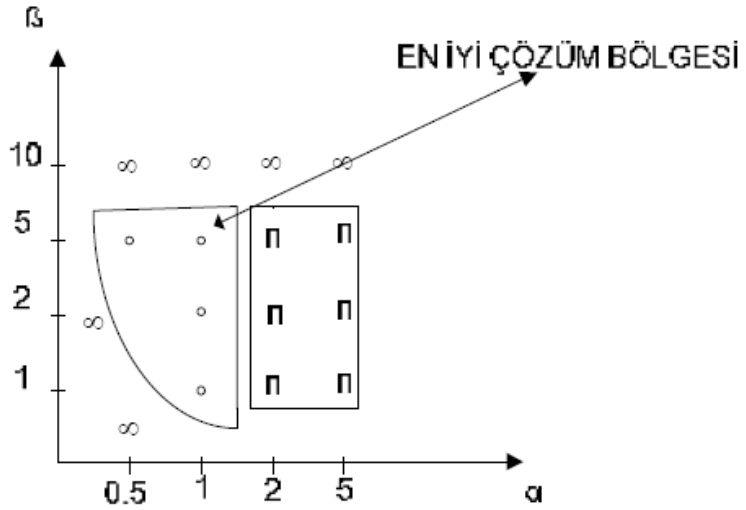
Parametre Değerleri

α değeri, ilgili yolun feromon miktarının önemini belirlemektedir.

α değerinin yüksek olması feromonun yoğun olduğu yolların seçilme olasılığını arttırmaktadır.

β değeri ise yol uzunluklarının, bir sonraki noktanın seçimindeki etkisini belirlemektedir. *β* değeri arttıkça tesadüfilik artmaktadır.

ŞEKİL'de α ve *β* parametrelerinin aldığı çeşitli değerler karşısında çözümün nasıl etkilendiği gösterilmiştir.



Örnek (python-kod)

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

# Şehir lokasyonlarının x ve y koordinatları
x = np.array([72, 81, 21, 82, 53, 8,  18, 45, 86, 95])
y = np.array([69, 33, 85, 84, 58, 66, 55, 49, 56, 27])

# Toplam şehir sayısı
sehirSayisi = len(x)

# Tüm şehirlerin birbiriyle olan mesafesi d matrisinde tutulur.
d = np.zeros([sehirSayisi,sehirSayisi])

# Şehirler arası mesafeler öklid mesafe kuralına göre hesaplanır
for i in range(sehirSayisi):
    for j in range(i+1,sehirSayisi):
        d[i,j] = math.sqrt(math.pow((x[i] - x[j]),2) + math.pow((y[i] - y[j]),2))
        d[j,i] = d[i,j]
```

$$d(A,B)=\sqrt{(X_1-X_2)^2+(Y_1-Y_2)^2}$$

öklit mesafesi

```python
# Tüm şehirlerin dolaşılmasını temsil eden bir rota için alınan toplam mesafe
# Şehirden şehire tüm uzaklıklar toplanır
def turMaliyeti(d,rota):
    L=0
    for i in range(len(rota)-1):
        L = L + d[rota[i], rota[i+1]]
    return L

# [0-1] aralığında rasgele bir sayı üretilir
# kümülatif oranının rasgele sayıyı içerdiği
# elemanlar seçilir.
# Rulet tekeri seçim yöntemi
def rulet(P):
    cprobs = P.copy()
    for i in range(1,len(P)):
        cprobs[i] = cprobs[i-1] + P[i]

    rs = np.random.ranf()
    secim = np.where(rs<cprobs)[0][0]
    return secim
```

```
iterasyon=100

# Karınca sayısı
Nant=20


# eta => mesafelerin çarpma işlemine göre tersi
# daha kısa mesafe => etanın değerinin daha yüksek olması
eta=1/d

# hiperparametreler
alpha=1
beta=1

# Feromen buharlaşmasını kontrol eden parametre
rho=0.3

# yollardaki feromen miktarı
tau=np.ones([sehirSayisi,sehirSayisi])

# sabit
Q=1

# Herbir karıncanın bir itarasyonda bir tur tamamladığında gezdiği şehirler
rotalar = np.zeros([Nant, sehirSayisi],dtype=int)

# Her bir rotanın maliyeti hesaplanım "maliyet" dizisinde tutulur
maliyet = np.zeros(Nant)

# En iyi maliyet yani en kısa rota uzunluğu "bestMaliyet" değişkeninde tutulur.
# Başlangıç değeri olarak "infity" değer almıştır.
bestMaliyet = math.inf

# Her iterasyonda en iyi maliyet değerine sahip rota tutulur
bestRota = []

# iterasyon boyunca hata oranları listeye alınır
objit = list()
```

```
83  for t in range(iterasyon):
84      for i in range(Nant):
85          gidilenSehirler = list()
86          gidilenSehirler.append(np.random.randint(0,sehirSayisi))
87          rotalar[i,0] = gidilenSehirler[0]
88
89          for j in range(1,sehirSayisi):
90              k = gidilenSehirler[-1]
91              P = np.power(tau[k,:],alpha) * np.power(eta[k,:],beta)
92              P[gidilenSehirler] = 0
93              P=P/sum(P)
94              s = rulet(P)
95              rotalar[i,j] = s
96              gidilenSehirler.append(s)
97
98          maliyet[i] = turMaliyeti(d, rotalar[i,:])
99          if maliyet[i] < bestMaliyet:
100             bestMaliyet = maliyet[i]
101             bestRota = rotalar[i,:]
102
103         for i in range(Nant):
104             rota = rotalar[i,:]
105             rota = np.append(rota,rota[0])
106             for j in range(sehirSayisi):
107                 m = rota[j]
108                 n = rota[j+1]
109                 tau[m,n]=tau[m,n]+Q/maliyet[i];
110
111     tau=(1-rho)*tau;
112     objit.append(bestMaliyet)
113     print('iterasyon : {}, BestMaliyet :{}'.format(t,bestMaliyet))
```

Bir karıncanın gittiği şehirlerin listesi

Rasgele bir şehir seçilip gidilenSehirler listesine ekleniyor. Aynı zamanda tüm karıncaların rota güzergahlarını tutan rota matrisine ekleniyor.

$$\left\{\left[\tau(i,u)\right]^{\alpha} \times \left[\eta(i,u)\right]^{\beta}\right\}$$

Bir sonraki gidilecek şehrin hangisi olduğunun tespiti için olasılık hesabı

Eğer daha önce o şehre gidilmişse olasılığı sıfır yapılır.

Bulunan olasılıklardan rulet tekeri seçim yöntemine göre bir rota belirlenerek gidilecek şehir seçilir.

En iyi maliyetli rota belirlenir.

feromen güncellemesi

feromen buharlaşması

```
iterasyon : 93, BestMaliyet :184.6233023929216
iterasyon : 94, BestMaliyet :184.6233023929216
iterasyon : 95, BestMaliyet :184.6233023929216
iterasyon : 96, BestMaliyet :184.6233023929216
iterasyon : 97, BestMaliyet :184.6233023929216
iterasyon : 98, BestMaliyet :184.6233023929216
iterasyon : 99, BestMaliyet :184.6233023929216
En iyi rota : [1 8 0 3 2 5 6 7 4 9], maliyet :184.6233023929216
```

```
116  print(bestRota)
117  data = np.append(bestRota,bestRota)
118  plt.plot(x[data], y[data],"o-")
119  plt.show()
```