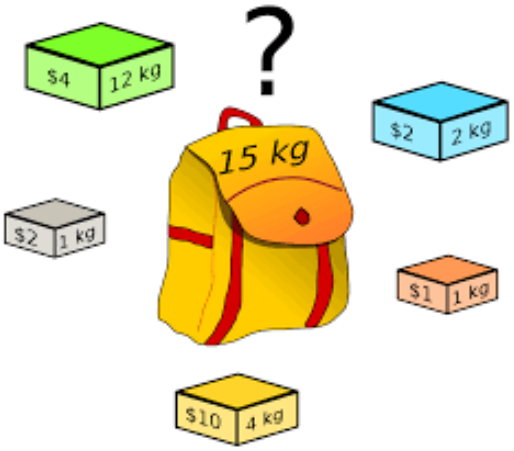
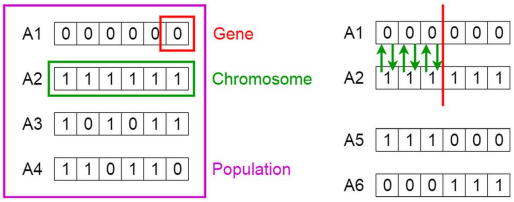




Genetic Algorithm

March 27, 2024

Knapsack and Genetic Algorithm by Umut Cindiloglu

Genetik algoritma (GA), doğal seleksiyon ve genetik mekanizmalardan esinlenerek geliştirilen bir arama ve optimizasyon yöntemidir. Genetik algoritmaların tarihçesi, 1960'ların sonlarına ve 1970'lerin başlarına dayanır. Bu yöntemin temeli, John Holland ve öğrencileri tarafından Michigan Üniversitesi'nde atılmıştır.

Charles Darwin'in evrim teorisinden ilham alan bu algoritma, en uygun çözümleri bulmak için bir popülasyon içindeki bireylerin (çözümlerin) "evrimleşmesini" simüle eder. Genetik algoritmalar, özellikle karmaşık ve çok boyutlu arama alanlarına sahip problemlerde etkili çözümler üretebilir.

0.1 Bir genetik algoritma genellikle aşağıdaki adımları içerir:

- **Başlangıç Popülasyonunun Oluşturulması:** Rastgele oluşturulan veya belirli bir yöntemle başlatılan bir dizi çözüm (birey) ile işleme başlar.
- **Uygunluk (Amaç) Fonksiyonu:** Her bir bireyin (çözümün) problemi ne kadar iyi çözdüğünü değerlendiren bir fonksiyondur. Bireylerin "uygunluğu", bu fonksiyonla belirlenir.
- **Seçim**: Uygunluk derecelerine göre bireyler seçilir. Yüksek uygunluk değerine sahip bireylerin seçilme olasılığı daha yüksektir.
- **Çaprazlama (Crossover):** Seçilen bireyler arasında, genetik bilginin yeni nesillere aktarılmasını sağlayan bir işlem gerçekleştirilir. Bu, genellikle iki bireyin genlerinin bir kısmının takas edilmesiyle yapılır.
- **Mutasyon:** Bireylerin genlerinde rastgele değişiklikler yapılır. Bu, arama alanındaki çeşitliliği artırarak algoritmanın yerel optimumlara takılıp kalmaktan kaçınmasına yardımcı olur.
- **Yeni Popülasyon:** Üretilen yeni bireylerle, eski popülasyon yer değiştirir. Bu yeni popülasyon, algoritmanın sonraki adımında kullanılır.
- **Durma Kriteri:** Belirli bir iterasyon sayısına ulaşma, belirli bir uygunluk değerine ulaşma veya iyileşmenin durması gibi koşullar sağlanana kadar algoritma tekrarlanır.

Genetik algoritmalar, mühendislikten finansa, yapay zekadan oyun teorisine kadar birçok alanda uygulanabilir ve genellikle NP-zor problemleri çözmek için kullanılır.

[68]:
```
import numpy as np
import pandas as pd
import random
import time
from collections import Counter
```

```
import matplotlib.pyplot as plt
```

[69]:
```
# Parametreler
ağırlıklar = np.array([20, 30, 35, 40, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95,␣
 ↪100])
# Bu kod satırı, sırt çantasına alınabilecek her bir nesnenin ağırlığını bir␣
 ↪numpy dizisi olarak tanımlar.

değerler = np.array([100, 120, 130, 145, 150, 160, 175, 180, 190, 200, 210, 220,␣
 ↪230, 240, 250])
# Bu kod satırı, sırt çantasına alınabilecek her bir nesnenin değerini bir numpy␣
 ↪dizisi olarak tanımlar.

kapasite = 250
# Bu kod satırı, sırt çantasının taşıyabileceği maksimum ağırlığı belirtir.
```

[70]:
```
popülasyon_boyutu = 100
# Bu kod satırı, genetik algoritmanın her iterasyonunda kullanılacak olan␣
 ↪popülasyonun büyüklüğünü belirtir.

mutasyon_oranı = 0.01
# Bu kod satırı, her bir genin mutasyona uğrama olasılığını belirtir.

çaprazlama_oranı = 0.7
# Bu kod satırı, iki bireyin çaprazlanma olasılığını belirtir.

iterasyon_sayısı = 100
# Bu kod satırı, genetik algoritmanın kaç iterasyon boyunca çalıştırılacağını␣
 ↪belirtir.
```

[71]:
```
def başlangıç_popülasyonu_oluştur(popülasyon_boyutu, n):
    return np.random.randint(2, size=(popülasyon_boyutu, n))
# Bu fonksiyon, rastgele bir başlangıç popülasyonu oluşturur.
# Her birey, sırt çantasına alınabilecek nesneleri temsil eden bir ikili dizi␣
 ↪ile ifade edilir.
```

[72]:
```
def uygunluk(birey, ağırlıklar, değerler, kapasite):
    toplam_değer = np.sum(birey * değerler)
    toplam_ağırlık = np.sum(birey * ağırlıklar)
    if toplam_ağırlık <= kapasite:
        return toplam_değer
    else:
        return 0
# Bu fonksiyon, bir bireyin (çözümün) uygunluğunu (fitness) hesaplar. Uygunluk,␣
 ↪bireyin toplam değeridir
 #eğer toplam ağırlık kapasiteyi aşmıyorsa.
# Aksi halde, uygunluk 0'dır.
```

[73]:
```
def turnuva_seçimi(popülasyon, uygunluklar, k=2):
    turnuva = np.random.choice(np.arange(len(popülasyon)), k)
    turnuva_uygunlukları = uygunluklar[turnuva]
    kazanan = turnuva[np.argmax(turnuva_uygunlukları)]
    return popülasyon[kazanan]
# Bu fonksiyon, turnuva seçimi yöntemini kullanarak yeni bireyler (çocuklar)␣
 ↪üretmek için iki ebeveyn seçer.
# Turnuva yöntemi seçim yöntemlerinden yalnızca birisidir, literatürde farklı␣
 ↪yöntemler de vardır.
```

[74]:
```
def çaprazla(anne, baba, çaprazlama_oranı):
    if random.random() < çaprazlama_oranı:
        kesim_noktası = np.random.randint(1, len(anne))
        çocuk1 = np.concatenate([anne[:kesim_noktası], baba[kesim_noktası:]])
        çocuk2 = np.concatenate([baba[:kesim_noktası], anne[kesim_noktası:]])
        return çocuk1, çocuk2
    else:
        return anne, baba
# Bu fonksiyon, belirli bir çaprazlama oranı ile iki bireyi (ebeveyni) çaprazlar.
#Çaprazlama işlemi, belirlenen bir kesim noktasından sonra iki bireyin genlerini␣
 ↪takas ederek
        #yeni çocuklar üretir.
#Eğer rastgele seçilen sayı çaprazlama oranından küçükse çaprazlama gerçekleşir,␣
 ↪aksi takdirde
        #ebeveynler değişmeden kalır.
```

[75]:
```
def mutasyon(birey, mutasyon_oranı):
    for i in range(len(birey)):
        if random.random() < mutasyon_oranı:
            birey[i] = 1 - birey[i]  # Genin değerini değiştir (0 ise 1 yap, 1␣
 ↪ise 0 yap)
    return birey
# Bu fonksiyon, belirli bir mutasyon oranı ile bireyin her bir genini rastgele␣
 ↪olarak mutasyona uğratır.
# Mutasyon, genetik çeşitliliği artırır ve yerel optimumlardan kaçınmaya␣
 ↪yardımcı olur.
```

[76]:
```
# Ana döngü: Genetik algoritmanın ana işlemlerini gerçekleştirir, aynı zamanda␣
 ↪süre ölçümü yapar
toplam_süre = 0  # Tüm çalışmaların toplam süresini tutacak
sonuçlar = []
x_time_run=100 # Algoritmayı X kez çalıştır
for çalıştırma in range(x_time_run):
    başlangıç_zamanı = time.time()  # Çalıştırmanın başlangıç zamanını kaydet

    popülasyon = başlangıç_popülasyonu_oluştur(popülasyon_boyutu,␣
 ↪len(ağırlıklar))
```

```
    for _ in range(iterasyon_sayısı):
        uygunluklar = np.array([uygunluk(b, ağırlıklar, değerler, kapasite) for b in popülasyon])
        yeni_popülasyon = []
        for _ in range(popülasyon_boyutu // 2):
            anne = turnuva_seçimi(popülasyon, uygunluklar)
            baba = turnuva_seçimi(popülasyon, uygunluklar)
            çocuk1, çocuk2 = çaprazla(anne, baba, çaprazlama_oranı)
            yeni_popülasyon.append(mutasyon(çocuk1, mutasyon_oranı))
            yeni_popülasyon.append(mutasyon(çocuk2, mutasyon_oranı))
        popülasyon = np.array(yeni_popülasyon)

    bitiş_zamanı = time.time()  # Çalıştırmanın bitiş zamanını kaydet
    çalışma_süresi = bitiş_zamanı - başlangıç_zamanı  # Bu çalışmanın süresini hesapla
    toplam_süre += çalışma_süresi  # Toplam süreye ekle

    # Her çalıştırmadaki en iyi çözümü bul ve kaydet
    en_iyi_uygunluk = max([uygunluk(b, ağırlıklar, değerler, kapasite) for b in popülasyon])
    sonuçlar.append(en_iyi_uygunluk)

# Ortalama çalışma süresini hesapla
ortalama_süre = toplam_süre / x_time_run
```

Gurobi ile amaç fonksiyonunun optimum değerini 850 bulmuştuk.

[77]:
```
# Sonuçların frekansını hesapla ve yazdır
sonuç_frekanları = Counter(sonuçlar)
print("Amaç Fonksiyonunun Değerleri ve Kaç Kere Bulunduğu:")
for değer, frekans in sonuç_frekanları.items():
    print(f"Değer: {değer}, Frekans: {frekans}")

# Ortalama süreyi yazdır
print(f"\nHer bir çalışmanın ortalama süresi: {ortalama_süre:.2f} saniye.")
```

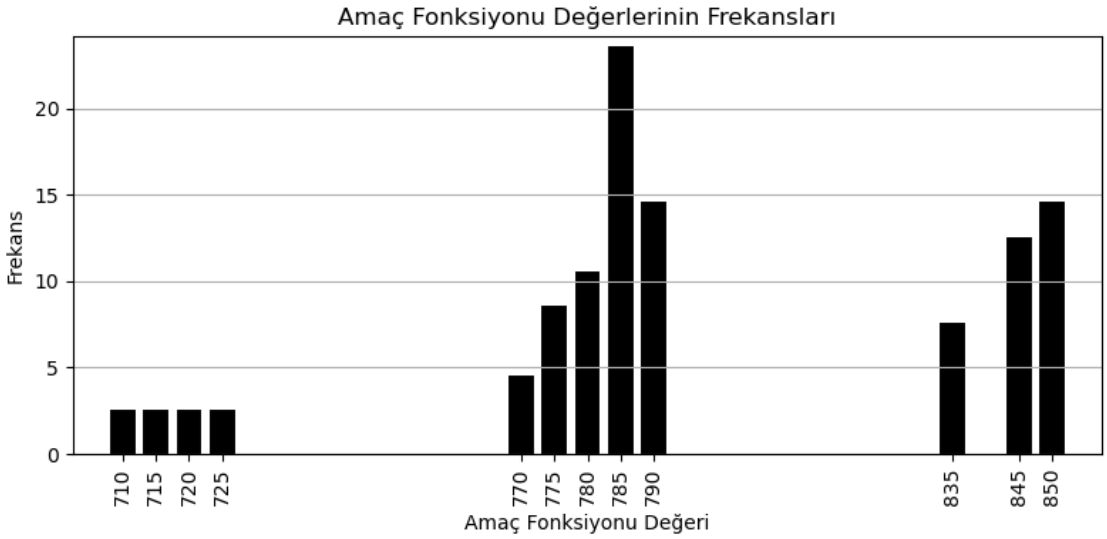
```
Amaç Fonksiyonunun Değerleri ve Kaç Kere Bulunduğu:
Değer: 780, Frekans: 10
Değer: 845, Frekans: 12
Değer: 850, Frekans: 14
Değer: 775, Frekans: 8
Değer: 785, Frekans: 23
Değer: 725, Frekans: 2
Değer: 770, Frekans: 4
Değer: 790, Frekans: 14
Değer: 715, Frekans: 2
Değer: 835, Frekans: 7
Değer: 720, Frekans: 2
```

```
Değer: 710, Frekans: 2

Her bir çalışmanın ortalama süresi: 0.42 saniye.
```

[78]:
```
sonuçlar_df = pd.DataFrame(list(sonuç_frekanları.items()), columns=['Amaç␣
↪Fonksiyonu Değeri', 'Frekans'])

# Plot çiz
plt.figure(figsize=(8, 4))  # Grafik boyutu
plt.bar(sonuçlar_df['Amaç Fonksiyonu Değeri'], sonuçlar_df['Frekans'],␣
↪color='blue', linewidth=10,
        edgecolor='black')
plt.xlabel('Amaç Fonksiyonu Değeri')
plt.ylabel('Frekans')
plt.title('Amaç Fonksiyonu Değerlerinin Frekansları')
plt.xticks(sonuçlar_df['Amaç Fonksiyonu Değeri'], rotation=90)
plt.grid(axis='y')
plt.tight_layout()  # Grafiği daha düzgün hale getir
plt.show()
```

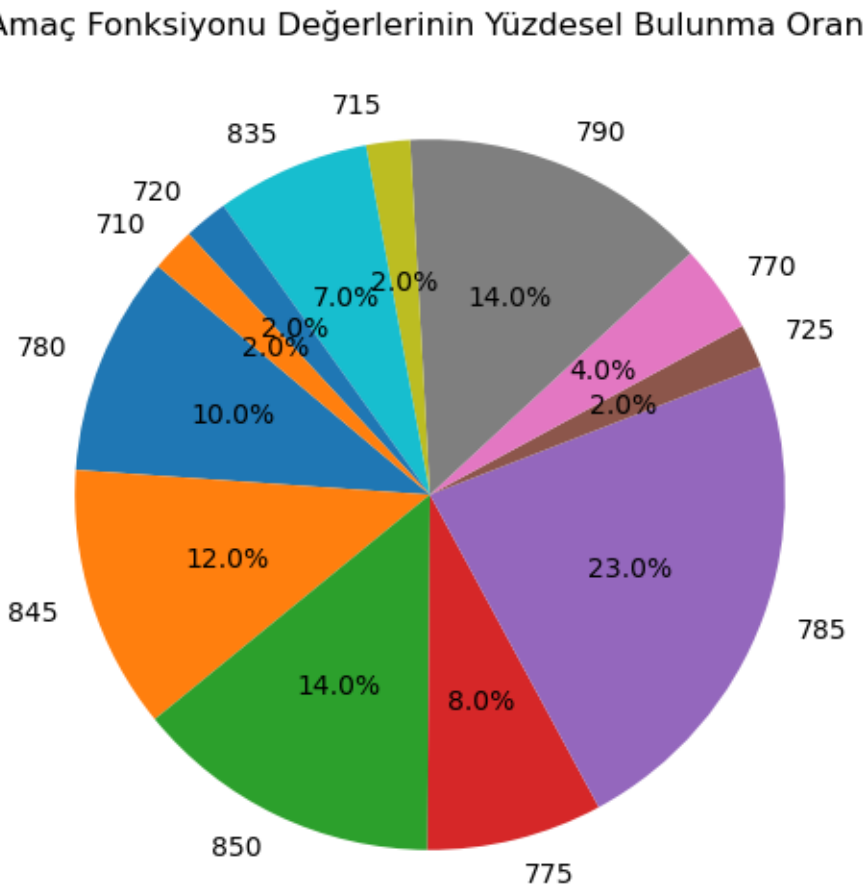


[79]:
```
# Sonuçların frekansını hesapla ve toplam değer sayısını bul
sonuç_frekanları = Counter(sonuçlar)
toplam_değer_sayısı = sum(sonuç_frekanları.values())

# Yüzdesel oranları hesapla ve DataFrame'e dönüştür
yüzdesel_oranlar = {değer: (frekans / toplam_değer_sayısı * 100) for değer,␣
↪frekans in sonuç_frekanları.items()}
```

```
yüzdesel_oranlar_df = pd.DataFrame(list(yüzdesel_oranlar.items()),␣
 ↪columns=['Amaç Fonksiyonu Değeri', 'Yüzdesel Oran'])

# Pasta grafiğini çiz
plt.figure(figsize=(6, 6))  # Grafik boyutu
plt.pie(yüzdesel_oranlar_df['Yüzdesel Oran'], labels=yüzdesel_oranlar_df['Amaç␣
 ↪Fonksiyonu Değeri'], autopct='%1.1f%%',
        startangle=140)
plt.title('Amaç Fonksiyonu Değerlerinin Yüzdesel Bulunma Oranı')
plt.show()
```



Genetik Algoritma Örnek Uygulama

Initial Population (Random)

Amaç Fonksiyonu

		f
1	2 2 2 2 2 0 1 2 1 0	802
2	2 2 1 2 2 2 0 2 1 0	-771
3	2 1 0 2 2 2 2 0 1 0	-799
4	1 1 1 0 1 2 1 1 0 2	707
5	0 0 1 0 1 2 1 1 0 0	281
6	0 2 1 1 0 0 1 1 2 1	-950
7	2 2 1 1 1 0 0 2 0 0	359
8	1 2 2 1 1 0 0 0 0 2	344
9	1 2 1 2 2 2 1 1 0 1	348
10	1 0 1 2 2 1 1 1 1 0	-2262

Turnuva selection

Selected

			f	
1	P1:	2 2 2 2 2 0 1 2 1 0	802	Crossover X
9	P2:	1 2 1 2 2 2 1 1 0 1	348	
9	P3:	1 2 1 2 2 2 1 1 0 1	348	Crossover ✓
4	P4:	1 1 1 0 1 2 1 1 0 2	707	
4	P5:	1 1 1 0 1 2 1 1 0 2	707	Crossover ✓
2	P6:	2 2 1 2 2 2 0 2 1 0	-771	
7	P7:	2 2 1 1 1 0 0 2 0 0	359	Crossover X
10	P8:	1 0 1 2 2 1 1 1 1 0	-2262	
3	P9:	2 1 0 2 2 2 2 0 1 0	-799	Crossover ✓
5	P10:	0 0 1 0 1 2 1 1 0 0	281	

en iyi 2'ye 15

CR = 0,6

Uniform

Değişken KA

		f
C1:	x x x x x x x x x x	xxx
C2:	x x x x x x x x x x	xxx
C3:	x x x x x x x x x x	xxx
C4:	x x x x x x x x x x	xxx
C5:	x x x x x x x x x x	xxx
C6:	x x x x x x x x x x	xxx
C7:	x x x x x x x x x x	xxx
C8:	x x x x x x x x x x	xxx
C9:	x x x x x x x x x x	xxx
C10:	x x x x x x x x x x	xxx

Mutation olasılığı

	f
1	xxx
2	xxx
3	xxx
4	xxx
5	xxx
6	xxx
7	xxx
8	xxx
9	xxx
10	xxx

i = 0

Bitirme koşulu

Yes

Print (Gen best)

Print (f_{best})