**Benzetim Tavlama (Simulated Annealing)(SA)**

**Dilaver ŞAHİN**

Fırat Üniversitesi Adli Bilişim Mühendisliği

* **Benzetim Tavlama Nedir?**
* **Tanım**: Benzetim tavlama, yüksek enerji durumundan düşük enerji durumuna geçiş yaparak en uygun çözümü bulmaya çalışan bir optimizasyon tekniğidir.
* **Kökeni**: Metalurji sürecinde bir metalin ısıtılması ve yavaşça soğutulması fikrine dayanır. Bu süreç, metalin iç yapısının en düşük enerji seviyesine ulaşmasını sağlar.
* **Benzetim Tavlamanın Temel Bileşenleri**
* **Enerji Fonksiyonu (Objective Function)**: Optimizasyon probleminin hedef fonksiyonudur. Minimize veya maksimize edilmek istenir.
* **Sıcaklık (Temperature)**: Çözüm arama sürecinin kontrol parametresidir. Yüksek sıcaklıklarda daha büyük adımlar atılabilirken, düşük sıcaklıklarda ince ayar yapılır.
* **Soğuma Programı (Cooling Schedule)**: Sıcaklığın zamanla nasıl düşürüleceğini belirleyen kurallardır. Genellikle geometrik veya logaritmik olarak azaltılır.
* **Geçiş Olasılığı (Transition Probability)**: Mevcut çözümden yeni bir çözüme geçişin kabul edilip edilmeyeceğini belirler. Yüksek sıcaklıklarda kötü çözümler de kabul edilebilirken, düşük sıcaklıklarda sadece iyi çözümler kabul edilir.
* **Algoritmanın Adımları**
* **Başlangıç Durumu**: Rastgele bir başlangıç çözümü seçilir ve başlangıç sıcaklığı belirlenir.
* **Iterasyon**:
* Yeni bir çözüm oluşturulur (komşu çözüm).
* Enerji farkı hesaplanır (ΔE).
* ΔE < 0 ise yeni çözüm kabul edilir; ΔE >= 0 ise kabul olasılığı hesaplanır (e^(-ΔE/T)).
* **Soğutma**: Sıcaklık, soğuma programına göre düşürülür.
* **Durdurma Kriteri**: Sıcaklık yeterince düşükse veya belirli bir iterasyon sayısına ulaşılmışsa algoritma durdurulur.
* **Benzetim Tavlamanın Uygulama Alanları**
* **Mühendislik**: Karmaşık sistemlerin optimizasyonu (örneğin, elektronik devre tasarımı).
* **Bilgisayar Bilimleri**: Gezgin Satıcı Problemi (TSP), çizelgeleme problemleri.
* **Fizik ve Kimya**: Molekül yapılandırmaları ve enerji minimizasyonu.
* **Ekonomi**: Portföy optimizasyonu.
* **Benzetim Tavlamanın Avantajları ve Dezavantajları**

**Avantajlar**:

* Küresel optimizasyon yapabilme yeteneği.
* Yerel minimumlardan kaçınma yeteneği.
* Basit ve geniş bir uygulama yelpazesi.

**Dezavantajlar**:

* Uygulama süresi genellikle uzundur.
* Parametre seçimleri (başlangıç sıcaklığı, soğuma programı) çok öneme sahiptir ve deneysel olarak belirlenmesi gerekebilir.

**Örnek Uygulama**

Bir örnekle açıklamak gerekirse, bir gezgin satıcı problemini çözmek için benzetim tavlama algoritması şu adımları takip edebilir:

* Rastgele bir şehir turu oluşturulur.
* Rastgele seçilen iki şehrin yerleri değiştirilir ve yeni turun toplam uzunluğu hesaplanır.
* Yeni tur, enerji fonksiyonuna göre değerlendirilir ve belirlenen kurallara göre kabul edilir veya reddedilir.
* Bu işlem, sıcaklık düşürülerek ve durdurma kriterine ulaşılana kadar tekrarlanır.

**Sonuç**

Benzetim tavlama, özellikle karmaşık optimizasyon problemleri için güçlü bir araçtır. Doğru uygulandığında, birçok farklı alanda başarılı çözümler üretebilir. Ancak, parametre ayarları ve algoritmanın doğru yapılandırılması önemlidir.

 

*Tablo 1 Tablo 2*

import numpy as np

import matplotlib.pyplot as plt

import random

import math

# Şehirlerin koordinatlarını rastgele oluşturma

*def* sehirler\_olustur(*n*):

    return np.random.rand(*n*, 2)

# İki şehir arasındaki mesafeyi hesaplama

*def* mesafe(*sehir1*, *sehir2*):

    return np.linalg.norm(*sehir1* - *sehir2*)

# Turun toplam uzunluğunu hesaplama

*def* toplam\_mesafe(*tur*, *sehirler*):

    mesafe\_toplami = 0

    for i in range(len(*tur*)):

        mesafe\_toplami += mesafe(*sehirler*[*tur*[i]], *sehirler*[*tur*[(i + 1) % len(*tur*)]])

    return mesafe\_toplami

# Rastgele iki şehir seçip yerlerini değiştirme (komşu çözüm)

*def* sehirleri\_degistir(*tur*):

    i, j = random.sample(range(len(*tur*)), 2)

    yeni\_tur = *tur*[:]

    yeni\_tur[i], yeni\_tur[j] = yeni\_tur[j], yeni\_tur[i]

    return yeni\_tur

# Benzetim tavlama algoritması

*def* benzetim\_tavlama(*sehirler*, *baslangic\_sicaklik*, *soguma\_orani*, *durdurma\_sicaklik*):

    n = len(*sehirler*)

    mevcut\_tur = list(range(n))

    mevcut\_mesafe = toplam\_mesafe(mevcut\_tur, *sehirler*)

    sicaklik = *baslangic\_sicaklik*

    en\_iyi\_tur = list(mevcut\_tur)

    en\_iyi\_mesafe = mevcut\_mesafe

    mesafeler = [mevcut\_mesafe]

    while sicaklik > *durdurma\_sicaklik*:

        yeni\_tur = sehirleri\_degistir(mevcut\_tur)

        yeni\_mesafe = toplam\_mesafe(yeni\_tur, *sehirler*)

        delta\_e = yeni\_mesafe - mevcut\_mesafe

        if delta\_e < 0 or random.random() < math.exp(-delta\_e / sicaklik):

            mevcut\_tur = yeni\_tur

            mevcut\_mesafe = yeni\_mesafe

            if mevcut\_mesafe < en\_iyi\_mesafe:

                en\_iyi\_tur = mevcut\_tur

                en\_iyi\_mesafe = mevcut\_mesafe

        mesafeler.append(mevcut\_mesafe)

        sicaklik \*= *soguma\_orani*

    return en\_iyi\_tur, en\_iyi\_mesafe, mesafeler

# Parametreler

sehir\_sayisi = 18

baslangic\_sicaklik = 1000

soguma\_orani = 0.995

durdurma\_sicaklik = 1e-3

# Şehirleri oluşturma

sehirler = sehirler\_olustur(sehir\_sayisi)

# Benzetim tavlama algoritmasını çalıştırma

en\_iyi\_tur, en\_iyi\_mesafe, mesafeler = benzetim\_tavlama(sehirler, baslangic\_sicaklik, soguma\_orani, durdurma\_sicaklik)

# Sonuçları çizme

plt.figure(*figsize*=(10, 5))

# Şehirleri ve en iyi turu gösterme

plt.subplot(1, 2, 1)

plt.scatter(sehirler[:, 0], sehirler[:, 1], *c*='red')

tur\_sehirler = np.array([sehirler[i] for i in en\_iyi\_tur + [en\_iyi\_tur[0]]])

plt.plot(tur\_sehirler[:, 0], tur\_sehirler[:, 1], 'b-')

plt.title('En İyi Tur')

# Mesafelerin değişimini gösterme

plt.subplot(1, 2, 2)

plt.plot(mesafeler)

plt.title('Tur Mesafesinin Değişimi')

plt.xlabel('Iterasyon')

plt.ylabel('Mesafe')

plt.tight\_layout()

plt.show()

print(*f*"En İyi Mesafe: {en\_iyi\_mesafe}")

sehirler\_olustur(n): Verilen bir n değeri için n adet rastgele koordinat üretir. Her bir koordinat bir şehri temsil eder.

mesafe(sehir1, sehir2): İki şehir arasındaki Euclidean mesafesini hesaplar.

toplam\_mesafe(tur, sehirler): Verilen bir tur için toplam mesafeyi hesaplar. Tur, şehirlerin bir listesidir ve her şehrin birbirini bir kez ziyaret ettiği kabul edilir.

sehirleri\_degistir(tur): Verilen bir turda rastgele iki şehri seçer ve yerlerini değiştirir. Bu işlem, mevcut turun komşu bir çözümünü oluşturur.

benzetim\_tavlama(sehirler, baslangic\_sicaklik, soguma\_orani, durdurma\_sicaklik): Simüle edilmiş tavlama algoritmasını uygular. Başlangıçta bir tur oluşturulur ve sıcaklık belirlenir. Daha sonra, belirli bir durdurma sıcaklığına kadar, sıcaklık azaltma oranına göre turdaki şehirlerin yerleri değiştirilir ve bu değişikliklerin kabul edilip edilmeyeceği belirlenir.

sehir\_sayisi = 18: 18 şehir içeren bir problem belirlenir.

baslangic\_sicaklik = 1000, soguma\_orani = 0.995, durdurma\_sicaklik = 1e-3: Tavlama algoritması için parametreler belirlenir.

sehirler = sehirler\_olustur(sehir\_sayisi): Belirlenen sayıda rastgele şehir oluşturulur.

en\_iyi\_tur, en\_iyi\_mesafe, mesafeler = benzetim\_tavlama(sehirler, baslangic\_sicaklik, soguma\_orani, durdurma\_sicaklik): Tavlama algoritması çalıştırılır ve en iyi tur, en iyi mesafe ve her iterasyondaki mesafeler kaydedilir.

Grafiğin sol tarafında, şehirler kırmızı noktalarla gösterilir ve en iyi tur mavi bir çizgi ile çizilir. Grafiğin sağ tarafında, her iterasyondaki mesafelerin değişimini gösteren bir grafik bulunur.