# 1. PARÇACIK SÜRÜ OPTİMİZASYONU

PSO, bir sürü içindeki her parçacığın, mevcut pozisyonu, hızı ve en iyi konumu gibi bilgileri tuttuğu bir algoritmadır. Parçacıklar, her iterasyonda hedef fonksiyonu hesaplamak için pozisyonlarını kullanırlar ve daha sonra hızlarını ve konumlarını güncellemek için en iyi pozisyonlarını ve sürü içindeki diğer parçacıkların bilgilerini kullanırlar.

Bu şekilde, PSO, sürü içindeki her bir parçacığın konumlarını ve hızlarını zaman içinde optimize ederek hedef fonksiyonunu eniyilemeye çalışır. Bu süreç, PSO'nun birçok optimizasyon probleminde başarılı bir şekilde kullanılmasını sağlamaktadır.

PSO'nun ana avantajı, diğer optimizasyon algoritmalarına kıyasla oldukça basit bir yapısının olması ve çok sayıda değişken içeren karmaşık problemlerde bile iyi sonuçlar vermesidir. PSO, ayrıca, özellikle işlem süresi ve hızlı çözüm gerektiren gerçek zamanlı problemler gibi dinamik problemlerde de kullanışlıdır.

Sonuç olarak, PSO doğal metaheuristik bir optimizasyon algoritmasıdır ve birçok farklı alanda başarılı bir şekilde kullanılmaktadır. PSO'nun basit yapısı ve iyi sonuçlar vermesi, özellikle karmaşık problemler için bir çözüm arayan araştırmacılar için ilgi çekicidir.

# 2. PROBLEM

## PARÇACIK SÜRÜ OPTİMİZASYONU (PSO) ALGORİTMASI KULLANARAK BİR BİTKİ YETİŞTİRME PROBLEMİ İÇİN EN UYGUN SULAMA PARAMETRELERİ BULMA:

Bitki yetiştirme, tarım sektörü için önemli bir konudur ve verimli bir bitki yetiştirme süreci için uygun sulama parametrelerinin belirlenmesi gereklidir. PSO algoritması, bitki yetiştirme problemleri için uygun bir optimizasyon aracıdır.

PSO algoritması, bitkilerin büyümesi için gerekli olan su miktarını optimize etmek ve bitki verimini artırmak için kullanılabilir. Bu amaçla, sulama süresi, sulama sıklığı, toprak nemi, sıcaklık ve nem gibi çeşitli parametreler, PSO algoritması ile optimize edilebilir.

PSO algoritması, bitki yetiştirme problemi için en uygun sulama parametrelerini belirleyerek, tarım sektöründe verimliliği artırmaya yardımcı olabilir. Ayrıca, PSO algoritması, bitki yetiştirme sürecinde sulama parametrelerinin manuel olarak ayarlanmasından kaynaklanan hataları da azaltabilir.

Bununla birlikte, PSO algoritmasının kullanılması, bitki yetiştirme sürecinde ölçülen verilerin doğruluğuna ve güvenilirliğine bağlıdır. Doğru olmayan veriler, PSO algoritmasının yanlış sonuçlar üretmesine neden olabilir. Bu nedenle, bitki yetiştirme problemleri için PSO algoritmasının kullanılması, doğru ölçümlerin yapılmasını gerektirir.

Sonuç olarak, PSO algoritması, bitki yetiştirme problemlerinde uygun sulama parametrelerini optimize etmek için kullanılabilir. PSO algoritmasının kullanımı, tarım sektöründe verimliliği artırabilir ve bitki yetiştirme sürecinde manuel ayarlamalardan kaynaklanan hataları azaltabilir.

## 3. YAPILAN ÇALIŞMA

Bu çalışmada, bir bahçedeki bitki büyümesini maksimize etmek ve su tüketimini minimize etmek için bir PSO (Parçacık Sürü Optimizasyonu) algoritması kullanarak optimal sulama parametrelerini bulmak için tasarlanmıştır.

NumPy kütüphanesini kullanarak bir dizi matematiksel işlem yapar. Kod, bahçenin alanı, toprak türü ve bitki türü gibi çeşitli değişkenleri tanımlar ve hedef fonksiyonu, PSO algoritması ve uygunluk değeri hesaplama gibi çeşitli fonksiyonları içerir.

PSO algoritması, belirli bir sayıda partikül oluşturur ve her bir partikülün belirli bir hızı ve konumu vardır. Her bir partikül, uygunluk fonksiyonunu hesaplamak için kullanılan bir dizi parametreyi temsil eder. Algoritma, her bir partikülün uygunluk fonksiyonu değerlerini hesaplar ve en iyi konumları ve uygunluk değerlerini kaydeder. Algoritma daha sonra, partiküllerin hızını ve konumunu güncelleyerek daha iyi konumlara doğru hareket eder. Bu işlem, belirli bir sayıda iterasyon boyunca devam eder ve sonunda en iyi konum ve uygunluk değerleri bulunur.

Bu kod örneği, belirli bir bitki türü ve toprak türü için en uygun sulama parametrelerini bulmak için kullanılabilir. Ancak, farklı bitki türleri, toprak türleri ve bahçe boyutları için farklı parametreler gerekebilir.

### 4. KOD İNCELEMESİ

Bu satırlarda, bahçenin alanı, toprak türü ve bitki türü gibi değişkenler tanımlanmıştır.

```
# Toprak ve bitki özelliklerinin tanımlanması
area = 20  # Bahçenin alanı (metrekare)
soil_type = "sandy"  # Toprak türü
plant_type = "tomato"  # Bitki türü
```

Bu satırlarda, hedef fonksiyon tanımlanmıştır. Bu fonksiyon, algoritmanın optimize etmeye çalışacağı ana fonksiyondur. Burada, su tüketimini minimize ederek bitki büyümesini maksimize etmeye çalışıyoruz. Fonksiyon, x parametresi olarak bir dizi değer alır ve bu değerleri kullanarak su tüketimi ve bitki büyümesi gibi faktörlere dayalı bir uygunluk değeri hesaplar.

plant_growth ve water_consumption değerleri, bitki büyümesi ve su tüketimi için hesaplanan değerlerdir. Fonksiyon, maksimize edilmesi gereken değerin negatif olarak verildiğine dikkat edin, bu nedenle PSO algoritmasının minimize edilmesi gereken bir fonksiyon olarak kullanması gerekiyor.

```
# Hedef fonksiyonun tanımlanması
def objective_function(x):
    # x[0] - su kaynağı
    # x[1] - sulama miktarı
    # x[2] - sulama sıklığı
    # x[3] - sulama süresi
    # x[4] - sulama bölgesi
    # x[5] - sulama yöntemi

    # Hedef fonksiyonu - Su tüketimini minimize ederek bitki büyümesini maksimize etmek
    water_consumption = x[1] * x[2] * x[3] * x[4] * x[5]
    plant_growth = x[1] * x[3] * x[4] * x[5]
    objective = -(plant_growth / water_consumption)
    return objective
```

Bu kod parçası, PSO algoritmasını uygulamak için gerekli olan parametreleri alır. Bu parametreler şunlardır:

objective_function: optimize edilecek hedef fonksiyon

bounds: optimize edilecek değişkenlerin alt ve üst sınırları

num_particles: partikül sayısı

maxiter: maksimum iterasyon sayısı

c1: bireysel öğrenme katsayısı

c2: topluluk öğrenme katsayısı

Bu parametreler kullanılarak, PSO algoritması uygulanır. Bu işlem sırasında, partiküllerin başlangıç konumları, başlangıç hızları ve uygunluk değerleri belirlenir. En iyi konumların ve

uygunluk değerlerinin kaydedilmesi, algoritmanın ilerleyen aşamalarında kullanılmak üzere yapılır.

```
# PSO algoritmasının tanımlanması
def pso(objective_function, bounds, num_particles, maxiter, c1, c2):
    # Partiküllerin başlangıç pozisyonlarının belirlenmesi
    dimensions = len(bounds)
    x_min, x_max = np.asarray(bounds).T
    x_0 = np.random.rand(num_particles, dimensions)
    x_0 = x_min + x_0 * (x_max - x_min)

    # Başlangıç hızlarının belirlenmesi
    v_min = -abs(x_max - x_min)
    v_max = abs(x_max - x_min)
    v_0 = np.random.uniform(low=v_min, high=v_max, size=(num_particles, dimensions))

    # En iyi konumların ve uygunluk değerlerinin kaydedilmesi
    best_position = x_0.copy()
    best_fitness = [objective_function(x) for x in x_0]
```

**PSO FOKSİYONU:**

Pso fonksiyonu altı argüman alır: optimize edilecek fonksiyon olan objective_function, her parametre için alt ve üst sınırları içeren bir tuple listesi olan bounds, optimizasyon algoritmasında kullanılacak tanecik sayısı olan num_particles, algoritmanın maksimum iterasyon sayısı olan maxiter ve ivme katsayıları olan c1 ve c2.

Fonksiyonun, boyut sayısını bounds listesinin uzunluğunu alarak hesaplar daha sonra bounds listesini her boyut için minimum ve maksimum değerleri içeren iki dizilime ayırır. Num_particles satırı ve boyutlar sütunları olan rastgele değerler matrisi oluşturur.

Bounds'dan alınan minimum ve maksimum değerler kullanılarak rastgele değerleri uygun aralıklara ölçeklendirir.

```
def pso(objective_function, bounds, num_particles, maxiter, c1, c2):
    # En iyi konumların ve uygunluk değerlerinin kaydedilmesi
    dimensions = len(bounds)
    x_min, x_max = np.asarray(bounds).T
    x_0 = np.random.rand(num_particles, dimensions)
    x_0 = x_min + x_0 * (x_max - x_min)
```

Her boyut için sınır aralığının mutlak değerini alarak minimum ve maksimum hız değerlerini hesaplar ve minimum için sonucun negatifini alır ve num_particles satırı ve dimensions sütunu olan v_min ve v_max arasında rastgele değerler içeren bir matris oluşturur.

Her parçacık için en iyi pozisyonunu mevcut pozisyonu olarak ve en iyi uygunluğu o pozisyondaki amaç fonksiyonu değeri olarak başlatır.

Küçük uygunluk değeri olan tüm parçacıkların en iyi pozisyonunu ve uygunluğunu global_best_position ve global_best_fitness olarak başlatır.

```
v_min = -abs(x_max - x_min)
v_max = abs(x_max - x_min)
v_0 = np.random.uniform(low=v_min, high=v_max, size=(num_particles, dimensions))

best_position = x_0.copy()
best_fitness = [objective_function(x) for x in x_0]

global_best_position = best_position.copy()
global_best_fitness = min(best_fitness)
```

Dıştaki döngüde, algoritma maxiter iterasyonu için çalışır.

İçteki döngüde, her biri num_particles parçacığı için:

Parçacığın hızı, önceki hızı, şimdiye kadar en iyi konumuna olan mesafesi (best_position - x_0) ve küresel en iyi konumuna olan mesafesi (global_best_position - x_0) kullanılarak güncellenir ve buna bir miktar rasgelelik (np.random.rand()) eklenir.

Parçacığın konumu güncellenir, güncellenmiş hız kullanılarak.

Parçacığın konumu, bounds tarafından belirlenen sınırlar içinde kalmak üzere sınırlanır.

Yeni pozisyonun uygunluğu (objective function value) hesaplanır.

Yeni pozisyonun uygunluğu, parçacığın mevcut en iyi pozisyonunun uygunluğundan daha iyi ise, parçacığın en iyi pozisyonu ve uygunluğu güncellenir.

Yeni pozisyonun uygunluğu, küresel en iyi uygunluktan daha iyi ise, küresel en iyi konumu ve uygunluğu güncellenir.

Son olarak, fonksiyon küresel en iyi konumu ve uygunluğu döndürür.

```
for i in range(maxiter):
    for j in range(num_particles):
        # Partikülün yeni hızı
        v_0 = v_0 + c1 * np.random.rand(dimensions) * (best_position - x_0) + c2 * np.random.rand(dimensions) * (global_best_position - x_0)

        # Partikülün yeni konumu
        x_0[j] = x_0[j] + v_0[j]

        # Sınırların dışına çıkmaması sağlanır
        x_0[j] = np.clip(x_0[j], x_min, x_max)

        # Partikülün uygunluk değeri
        fitness = objective_function(x_0[j])

        # En iyi konumların ve uygunluk değerlerinin güncellenmesi
        if fitness < best_fitness[j]:
            best_fitness[j] = fitness
            best_position[j] = x_0[j]

            if fitness < global_best_fitness:
                global_best_fitness = fitness
                global_best_position = x_0[j]

return global_best_position, global_best_fitness
```

Bu kod, optimize edilecek bir amaç fonksiyonu (objective_function) ve optimize edilecek değişkenlerin sınırlandırılmış değerlerinin (bounds) bir listesi verilerek, PSO algoritması kullanarak bu amaç fonksiyonunun global minimumunu bulmayı amaçlar.

Algoritma, 50 tane parçacık kullanarak, en fazla 100 iterasyon boyunca çalışacak şekilde ayarlanmıştır. Hızlanma katsayıları (c1 ve c2) her ikisi de 2 olarak ayarlanmıştır.

PSO algoritması, parçacıkların rastgele pozisyonlarla başlayarak amaç fonksiyonunun minimumunu bulmak için iteratif bir yaklaşım kullanır. Her iterasyonda, her parçacığın pozisyonu ve hızı güncellenir ve her bir parçacık kendi en iyi pozisyonunu ve global en iyi pozisyonunu takip eder.

Kodun çıktısı, algoritmanın global en iyi pozisyonunu ve uygunluk değerini ekrana yazdırır.

```
bounds = [(0, 100), (0, 50), (1, 7), (1, 24), (1, 10), (0, 1)]
num_particles = 50
maxiter = 100
c1 = 2
c2 = 2
global_best_position, global_best_fitness = pso(objective_function, bounds, num_particles, maxiter, c1, c2)
print("En iyi pozisyon: ", global_best_position)
print("En iyi uygunluk değeri: ", global_best_fitness)
```

**ALINAN ÇIKTI:**

```
En iyi pozisyon:  [ 0. 50.  1.  1. 10.  1.]
En iyi uygunluk değeri:  -1.0
PS C:\Users\ayşegül\Desktop\PythonOdevler>
```

Algoritma, bir dizi parametre (su kaynağı, sulama miktarı, sulama sıklığı, sulama süresi, sulama bölgesi ve sulama yöntemi) için olası değerler arasında rastgele gezinerek, her bir değer kombinasyonu için hedef fonksiyonun değerini hesaplar. Bu işlemi birkaç kez (maxiter) tekrarlayarak, en iyi uygunluk değeri ve pozisyonunu belirler.

Çıktıda, "En iyi pozisyon" dizesi, en iyi sonucu veren parametre kombinasyonunu gösterir. Bu kombinasyon, 6 parametrenin her biri için bir sayı içerir. "En iyi uygunluk değeri" ise, bu parametrelerin kullanıldığı hedef fonksiyonunun en iyi sonucunu gösterir. Burada, negatif bir değer gösterildiği için, hedef fonksiyonu maksimize etmek istediğimiz anlaşılır.

Bu durumda, en iyi çözümde su kaynağı 0, sulama miktarı 50, sulama sıklığı 1, sulama süresi 1, sulama bölgesi 10 ve sulama yöntemi 1 olarak seçilmiştir. "En iyi uygunluk değeri" satırı, en iyi çözümün hedef fonksiyonunun değerini gösterir. Bu durumda, en iyi çözüm hedef fonksiyonunu en çok maksimize eden çözümdür ve hedef fonksiyonunun değeri -1.0'dır.