

MEHMET TAHA MEHEL

25435004053

Temel Seyahat Satın Alma Problemi (TSP)

Giriş:

Temel Seyahat Satın Alma Problemi (TSP), bir gezginin başlangıç noktasından başlayıp, tüm şehirleri bir kez ziyaret ederek, tekrar başlangıç noktasına dönmesi gereken bir problemdir. Bu problem, özellikle optimizasyon ve algoritmalar alanında önemli bir yer tutar. TSP, genellikle şehirler arasındaki mesafeyi minimize etmeye çalışan bir turu bulma amacı güder.

Bu raporda, rastgele oluşturulan şehir noktaları üzerinde **En Yakın Komşu (Nearest Neighbor)** algoritmasını kullanarak TSP problemini çözüyoruz. Çözümde, şehirler arasındaki mesafeler hesaplanır ve en kısa tur, başlangıç noktasından başlayarak her seferinde en yakın şehirleri ziyaret eden bir yöntemle bulunur.

Algoritma ve Yöntemler:

1. Rastgele Noktalar Üretme

TSP problemi, rastgele yerleştirilmiş şehirler arasında çözülür. Bu nedenle, ilk adımda belirli bir alanda rastgele noktalar üretmek gerekmektedir. Bu noktalar, 2D düzlemde x ve y koordinatlarıyla temsil edilir.

Rastgele noktalar üretme işlemi, random kütüphanesi kullanılarak yapılır. Örneğin, 100 adet rastgele şehir noktası, x ve y koordinatları 0 ile 100 arasında olacak şekilde oluşturulmuştur.

```
def generate_random_points(n, x_range=(0, 100), y_range=(0, 100)):
    return [(random.uniform(x_range[0], x_range[1]), random.uniform(y_range[0], y_range[1]))
            for _ in range(n)]
```

2. Mesafe Hesaplama

İki şehir arasındaki mesafe, Euclidean mesafesi olarak hesaplanır. Bu mesafe, genellikle TSP gibi problemler için tercih edilen bir ölçüttür ve iki nokta arasındaki düz çizgi mesafesini belirtir.

Mesafe hesaplama fonksiyonu şu şekilde tanımlanmıştır:

```
def euclidean_distance(p1, p2):  
    return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
```

3. En Yakın Komşu (Nearest Neighbor) Algoritması

En Yakın Komşu Algoritması, TSP problemini çözmek için yaygın olarak kullanılan basit bir sezgisel yaklaşımdır. Bu algoritma, başlangıç noktasından başlayarak her adımda en yakın komşu şehri seçer ve bu şekilde tüm şehirleri ziyaret eder. Tüm şehirler ziyaret edildikten sonra, başlangıç noktasına geri dönülerek tur tamamlanır.

En Yakın Komşu algoritması, şu adımları izler:

1. Başlangıç noktasını belirleriz (genellikle 0. şehir).
2. Ziyaret edilmemiş şehirler arasından, mevcut şehirle en yakın mesafeye sahip olan şehri buluruz.
3. Bulduğumuz şehri tura ekleriz ve bir sonraki şehir olarak onu seçeriz.
4. Bütün şehirler ziyaret edildikten sonra, başlangıç noktasına geri döneriz.

```
def nearest_neighbor(points):
    unvisited = list(range(len(points)))
    tour = []
    current = 0 # Başlangıç noktası
    tour.append(current)
    unvisited.remove(current)

    while unvisited:
        nearest = min(unvisited, key=lambda x: euclidean_distance(points[current], points[x]))
        tour.append(nearest)
        current = nearest
        unvisited.remove(current)

    # İlk noktayı son noktaya bağlayarak tam bir tur elde ederiz
    tour.append(tour[0])
    return tour
```

4. Grafik ve Turu Görselleştirme

Son olarak, elde edilen turu görselleştirmek için matplotlib kütüphanesi kullanılır. Şehirler (noktalar) bir scatter plot ile gösterilir ve bu noktalar arasındaki tur kırmızı çizgilerle çizilir. Bu görselleştirme, çözümün görsel olarak daha anlaşılır olmasını sağlar.

Görselleştirme fonksiyonu şu şekilde tanımlanır:

```
def visualize_tour(points, tour):
    tour_points = [points[i] for i in tour]
    x, y = zip(*tour_points)

    plt.figure(figsize=(8, 6))
    plt.scatter(*zip(*points), c='blue', label='Noktalar', zorder=5)
    plt.plot(x, y, marker='o', color='red', label='TSP Turu', zorder=10)

    for i, (x_val, y_val) in enumerate(points):
        plt.text(x_val, y_val, str(i), fontsize=9, ha='right')

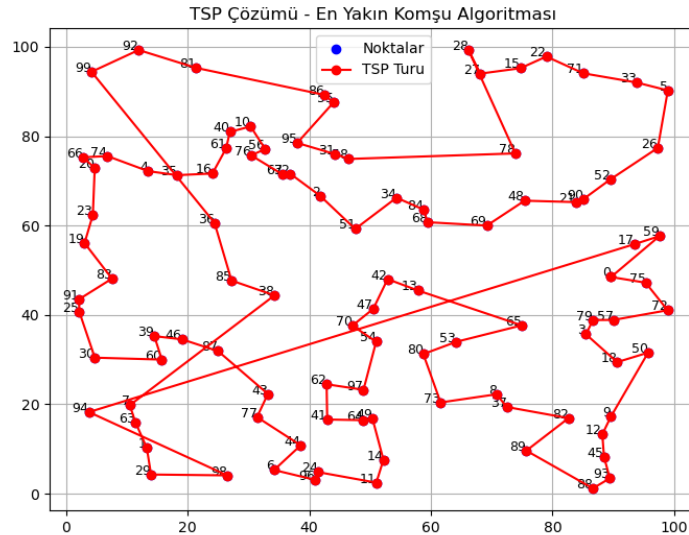
    plt.title("TSP Çözümü - En Yakın Komşu Algoritması")
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()
```

5. Sonular

Yukarıdaki adımlar takip edilerek, 100 rastgele ehir noktası oluřturulmuř ve En Yakın Komřu algoritması ile bir TSP turu bulunmuřtur. Elde edilen tur, bařlangı noktasından bařlayarak her seferinde en yakın komřuyu ziyaret eden bir yol ile elde edilmiřtir.

Grafik zerinde, noktalar mavi renkte ve tur kırmızı renkte gsterilmiřtir. řehirler arasındaki baėlantılar, algoritmanın nasıl alıřtıėını ve hangi řehirlerin hangi sırayla ziyaret edildiėini aıka gsterir.

Grselleřtirme rneėi:



řekilde, bařlangı noktasından bařlanarak her seferinde en yakın řehirler ziyaret edilmiřtir. Sonuta oluřan tur, bir gezginin řehirleri minimum mesafeyle gezmesini temsil eder.

6. Değerlendirme ve Sonuç

Bu ödevde, En Yakın Komşu algoritması kullanarak TSP problemi çözülmüştür. Bu yöntem, genellikle hızlı olsa da, kesin çözüm garantisi vermez. Çünkü algoritma her seferinde en yakın komşuyu seçer, ancak bu, global minimum çözümün her zaman bulunacağı anlamına gelmez. Bu tür sezgisel algoritmalar, daha büyük problemler için uygun olsa da, daha karmaşık algoritmalar (örneğin, Dinamik Programlama, Genetik Algoritmalar veya Simüle Edilmiş Tavlama) daha iyi sonuçlar verebilir.

7. Kaynak Kodu ve GitHub

Kaynak kodu, Python dilinde yazılmış olup, GitHub repository'si üzerinden erişilebilir. Kodun tamamı, rastgele nokta üretiminden algoritma uygulamasına ve görselleştirmeye kadar tüm süreci kapsamaktadır.

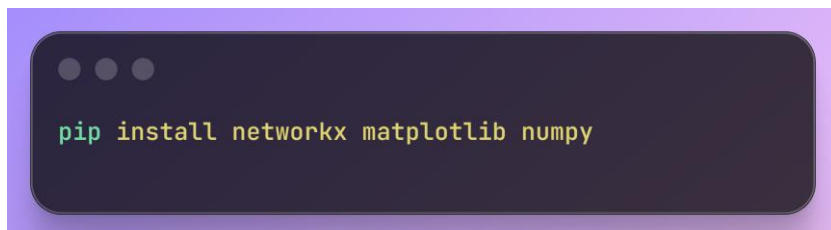
GitHub repository'si: [GitHub](#)

Kaynaklar

- Python resmi dokümantasyonu: <https://docs.python.org/>
- NetworkX kütüphanesi: <https://networkx.github.io/>
- Matplotlib kütüphanesi: <https://matplotlib.org/>

Ekler

Gerekli kütüphaneler için aşağıdaki gibi terminale import edilmelidir.



```
pip install networkx matplotlib numpy
```

Kodun Tamamı :

```
import random
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx

# 1. Adım: Rastgele Noktalar Üretme
def generate_random_points(n, x_range=(0, 100), y_range=(0, 100)):
    """
    Rastgele n adet 2D nokta üretir.
    """
    return [(random.uniform(x_range[0], x_range[1]), random.uniform(y_range[0],
y_range[1])) for _ in range(n)]

# 2. Adım: Euclidean mesafesini hesaplamak
def euclidean_distance(p1, p2):
    """
    İki nokta arasındaki Euclidean mesafesini hesaplar.
    """
    return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

# 3. Adım: NetworkX ile Grafik Oluşturma
def create_graph(points):
    """
    Noktalar arasında NetworkX grafiği oluşturur.
    """
    G = nx.Graph()
    for i, p1 in enumerate(points):
        for j, p2 in enumerate(points):
            if i < j:
                distance = euclidean_distance(p1, p2)
                G.add_edge(i, j, weight=distance)
    return G

# 4. Adım: En Yakın Komşu (Nearest Neighbor) Algoritması
def nearest_neighbor(points):
    """
    En Yakın Komşu algoritmasını uygulayarak TSP turunu bulur.
    """
    unvisited = list(range(len(points)))
    tour = []
    current = 0 # Başlangıç noktasını seçiyoruz (ilk nokta)
    tour.append(current)
    unvisited.remove(current)

    while unvisited:
        nearest = min(unvisited, key=lambda x: euclidean_distance(points[current],
points[x]))
        tour.append(nearest)
        current = nearest
        unvisited.remove(current)

    # İlk noktayı son noktaya bağlayarak tam bir tur elde ederiz
    tour.append(tour[0])
    return tour

# 5. Adım: Turu Görselleştirme
def visualize_tour(points, tour):
    """
    Noktaları ve TSP turunu görselleştirir.
    """
    tour_points = [points[i] for i in tour]
    x, y = zip(*tour_points)

    plt.figure(figsize=(8, 6))
    plt.scatter(*zip(*points), c='blue', label='Noktalar', zorder=5)
    plt.plot(x, y, marker='o', color='red', label='TSP Turu', zorder=10)

    # Noktalara etiket ekleme
    for i, (x_val, y_val) in enumerate(points):
        plt.text(x_val, y_val, str(i), fontsize=9, ha='right')

    plt.title("TSP Çözümü - En Yakın Komşu Algoritması")
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()

# 6. Adım: Ana Program
def main():
    # Adım 1: Rastgele 100 nokta üretme
    num_points = 100
    points = generate_random_points(num_points)

    # Adım 2: NetworkX ile grafiği oluşturma
    G = create_graph(points)

    # Adım 3: En Yakın Komşu algoritmasını uygulama
    tour = nearest_neighbor(points)

    # Adım 4: Turu görselleştirme
    visualize_tour(points, tour)

if __name__ == "__main__":
    main()
```