Assignment 1 – Basic TSP on Random Points

Oluşturan: İbrahim Sezer 25435004019

Github repo link: https://github.com/ibrahimsezer/BLM5026_Bilgisayar_Oyunlarinda_Yapay_Zeka/tree/main/homework_1

Proje Amacı

Bu ödevin amacı, Gezgin Satıcı Problemi (Traveling Salesperson Problem - TSP) için temel bir **grafik soyutlaması** yapmak, rastgele bir örnek oluşturmak ve problemin çözümü için **En Yakın Komşu (Nearest Neighbor)** gibi basit bir sezgiseli uygulamaktır.

Gereksinimler

Bu projeyi çalıştırmak için aşağıdaki Python kütüphaneleri gereklidir:

pip install numpy networkx matplotlib

Çalıştırma

Projeye ulaşmak için github reposu linki: [Github Repo'su]

Projeyi klonladıktan sonra (`homework_1`) klasörüne geçtikten sonra Python script dosyasını (`tsp_nearest_neighbor.py`) çalıştırın:

cd homework_1
python tsp_nearest_neighbor.py

Terminal çıktısında tur sırasını ve maliyetini göreceksiniz. Ayrıca, bir `matplotlib` penceresinde, rastgele noktalar ve sezgisel ile bulunan tur görselleştirilecektir.

```
PS D:\projects\BLM5026_Bilgisayar_Oyunlarinda_Yapay_Zeka\homework_1> python .\tsp_nearest_neighbor.py
Grafik 15 düğüm ve 105 kenar içeriyor.
------------------------------
TSP Sezgisel Sonuçları:
Tur (Düğüm Sırası): [0, 9, 7, 2, 12, 1, 5, 3, 6, 8, 13, 11, 14, 10, 4, 0]
Tur Maliyeti (Toplam Uzunluk): 314.64
```

Uygulanan Yaklaşım ve Kriter Analizi

Rastgele Grafik Üretimi

Aşağıdaki tablo, rastgele grafik oluşturma sürecinin kriter analizini özetlemektedir:

Kriter	Açıklama
Doğru Grafik	`networkx.complete_graph` ile tam grafik oluşturulmuştur. Kenar ağırlıkları, noktalar arasındaki Öklid mesafesi ile doğru şekilde atanmıştır.
Parametreli	Grafik boyutu (`NUM_POINTS`) ve koordinat sınırı (`max_coord`) fonksiyon parametreleri aracılığıyla ayarlanabilir.
Tekrarlanabilirlik	Kodun başlangıcına `numpy.random.seed(16)` eklenerek, rastgele nokta üretimi tekrarlanabilir hale getirilmiş ve her çalıştırmada aynı grafik örneği garanti edilmiştir.

TSP Sezgiseli: En Yakın Komşu (Nearest Neighbor)

Yaklaşım: Bu, her adımda yerel olarak en iyi kararı (en kısa mesafeyi) vermeye çalışan **açgözlü (greedy)** bir sezgiseldir.

İşleyiş:

1. Tur, başlangıç düğümünden (0. düğüm) başlar.
2. Her adımda, mevcut düğümden henüz ziyaret edilmemiş düğümler arasında en kısa mesafedeki düğüm seçilir.
3. Tüm düğümler ziyaret edildikten sonra, turu kapatmak için son düğümden başlangıç düğümüne geri dönülür.

Verimlilik: Sezgisel, her düğümde geriye kalan düğümleri taradığı için $O(N^2)$ zaman karmaşıklığına sahiptir ve bu, basit bir sezgisel için verimli kabul edilir.

Görselleştirme

`matplotlib` kullanılarak net bir 2D görsel oluşturulmuştur.

- **Düğümler:** Tüm noktalar mavi noktalarla, başlangıç noktası ise kırmızı yıldızla vurgulanmıştır. Her noktanın etiketi (indeksi) gösterilmiştir.
- **Tur:** Tur, gri çizgilerle, turu kapatan son kenar ise netlik için yeşil renkte çizilmiştir.
- **Açıklık:** Görselin başlığında ve eksen etiketinde kullanılan `seed`, tur sırası ve hesaplanan toplam maliyet açıkça belirtilmiştir.

Kenar Sayısı Doğrulaması

Kullanılan $N = 15$ düğümlü grafik, bir Tam Grafik (Complete Graph) olduğu için, toplam kenar sayısı (E) aşağıdaki formülle hesaplanır:

$$E = \frac{N \times (N-1)}{2}$$

Hesaplama:

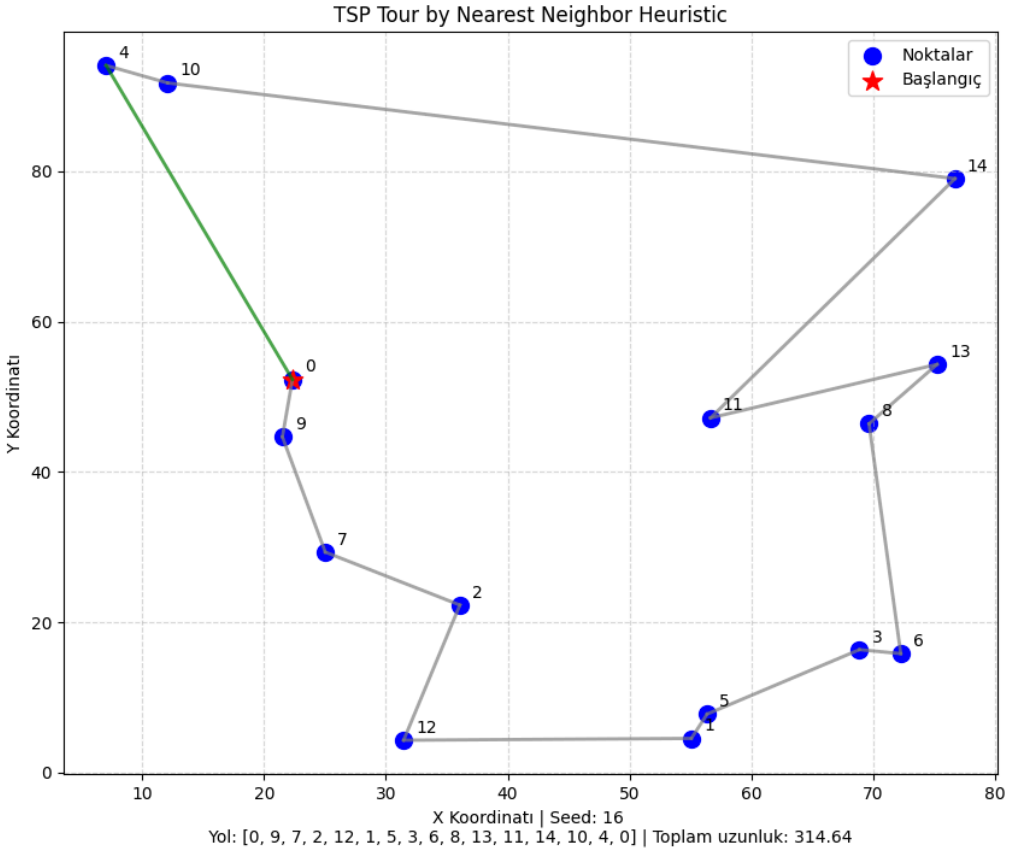
$N = 15$ için:

$$E = \frac{15 \times (15-1)}{2} = \frac{15 \times 14}{2} = 105$$

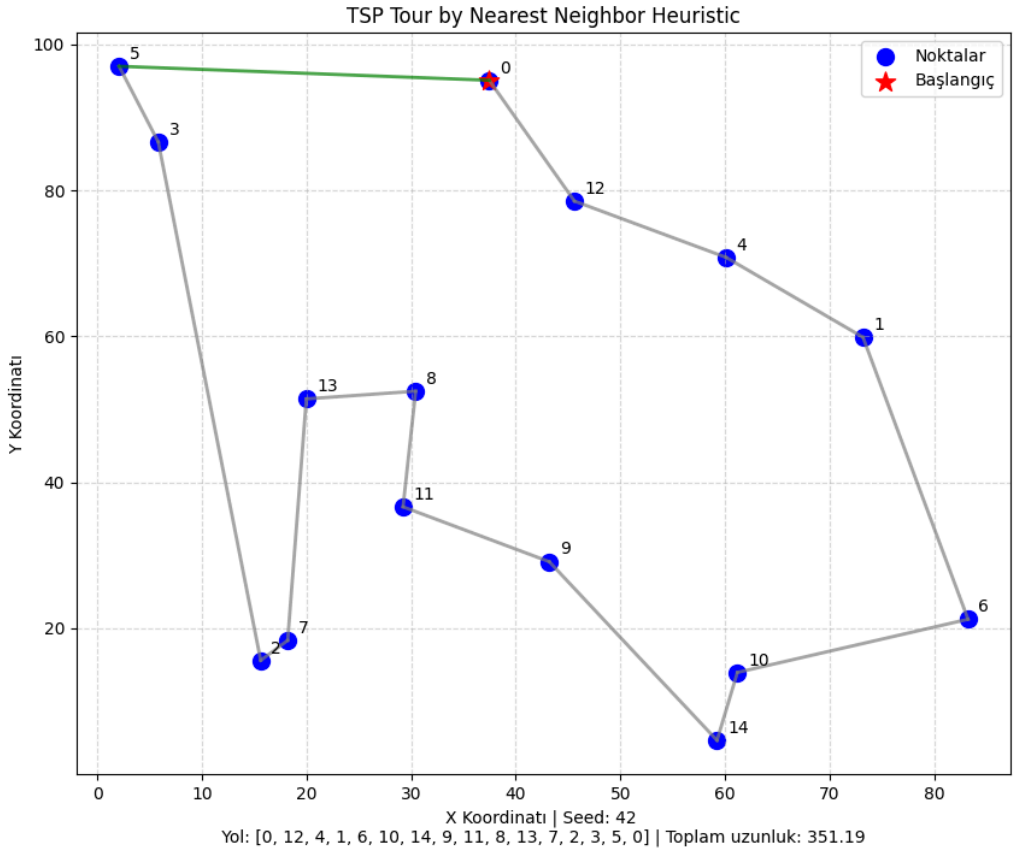
Bu sonuç, kod çıktısında belirtilen **105 kenar** sayısının doğruluğunu teyit etmektedir.

Grafik Sonuçları

Seed = 16 değeri için:



Seed = 42 değeri için:



Kod bloğu

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

#Tohum sabiti
SEED = 16
np.random.seed(SEED)

def generate_tsp_instance(num_points, max_coord=100):
    """
    Rastgele noktalar üretir ve tam grafik oluşturur.
    """
    points = {
        i: (np.random.uniform(0, max_coord), np.random.uniform(0, max_coord)) #
rastgele x,y koordinatları
        for i in range(num_points) # verilen num_points sayısı kadar nokta
    }
```

```
    G = nx.complete_graph(num_points)

    for i in range(num_points):
        for j in range(i + 1, num_points):
            dist = np.linalg.norm(np.array(points[i]) - np.array(points[j]))
            G.edges[i, j]['weight'] = dist

    return G, points

NUM_POINTS = 15
graph, coordinates = generate_tsp_instance(NUM_POINTS)

print(f"Grafik {graph.number_of_nodes()} düğüm ve {graph.number_of_edges()} kenar
içeriyor.")


def nearest_neighbor_tsp(graph):
    """
    Nearest Neighbor (En Yakın Komşu) sezgiselini uygular.
    """
    nodes = list(graph.nodes()) #node'ları bir listeye alıyoruz
    start_node = nodes[0]  #başlangıç node'unu belirliyoruz

    tour = [start_node] #tur listesi oluşturuyoruz
    visited = {start_node} #ziyaret edilen düğümleri tutan kümeyi tanımlıyoruz
    current_node = start_node #şu anki düğümü başlangıç düğümü olarak ayarlıyoruz
    total_cost = 0 #toplam maliyeti tanımlıyoruz

    # Tüm düğümleri ziyaret edene kadar döngü
    while len(visited) < len(nodes):
        min_dist = float('inf') #başlangıçta minimum mesafeyii sonsuz olarak
ayarlıyoruz
        next_node = None

        for neighbor in nodes:
            if neighbor not in visited:
                dist = graph.edges[current_node, neighbor]['weight'] #mevcut
düğümden komşuya olan mesafe

                if dist < min_dist: #eğer bu mesafe minimum mesafeden küçükse yer
değiştiriyoruz
                    min_dist = dist
                    next_node = neighbor #komşuyu sonraki düğüm olarak seciyoruz

        if next_node is not None:
            tour.append(next_node)
            visited.add(next_node)
            total_cost += min_dist
            current_node = next_node
```

```
        else:
            break

    if len(tour) == len(nodes):
        last_to_first_cost = graph.edges[current_node, start_node]['weight'] #son
düğümden başlangıç düğümüne olan mesafe hesabi
        tour.append(start_node)
        total_cost += last_to_first_cost

    return tour, total_cost

tsp_tour, tour_cost = nearest_neighbor_tsp(graph)


# Terminal sonuçları
print("-" * 30)
print("TSP Sezgisel Sonuçları:")
print(f"Tur (Düğüm Sırası): {tsp_tour}")
print(f"Tur Maliyeti (Toplam Uzunluk): {tour_cost:.2f}")
print("-" * 30)


def visualize_tsp_tour(coordinates, tour, title="TSP Tour by Nearest Neighbor
Heuristic"):
    """
    Noktaları ve bulunan turu görselleştirir.
    """
    plt.figure(figsize=(10, 8))

    x_coords = [coord[0] for coord in coordinates.values()]
    y_coords = [coord[1] for coord in coordinates.values()]

    plt.scatter(x_coords, y_coords, c='blue', s=100, label='Noktalar')

    for i, (x, y) in coordinates.items():
        plt.annotate(str(i), (x + 1, y + 1), fontsize=10)

    start_node = tour[0]
    plt.scatter(coordinates[start_node][0], coordinates[start_node][1], c='red',
s=150, marker='*', label='Başlangıç')

    for i in range(len(tour) - 1):
        n1 = tour[i]
        n2 = tour[i+1]

        x1, y1 = coordinates[n1]
        x2, y2 = coordinates[n2]

        color = 'green' if i == len(tour) - 2 else 'gray'
        linestyle = '-'
```

```
        linewidth = 2

        plt.plot([x1, x2], [y1, y2], color=color, linestyle=linestyle,
linewidth=linewidth, alpha=0.7)

    plt.title(title)
    plt.xlabel(f"X Koordinatı | Seed: {SEED} \n Yol: {tsp_tour} | Toplam uzunluk:
{tour_cost:.2f} ") #sonuçları plot ekranında yazdırma
    plt.ylabel("Y Koordinatı")
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.legend()
    plt.show()

# Görselleştirme
visualize_tsp_tour(coordinates, tsp_tour)
```