Çağla Ökmen

Homework 1 Dökümasyonu:

Ödevin ilk aşaması için bir graf oluşturuldu ve gezgin satıcı problemi (TPS) için basit bir sezgisel yöntemi uygulandı. Nearest Neighbor algoritmasıda denenmiş olup bazı graf tiplerinde daha düşük performans izlediği görülmüş ve devamında Insertion algoritması uygulanmıştır. Bu dokümanda İnsertion uygulaması anlatılmaktadır.

Gezgin satıcı problemi: Bir satıcı bir şehirden başlayarak belirli miktardaki şehirleri en az yol alacak şekilde giderek ilk şehre geri dönmesi problemini ele almaktadır.

Nearet Neigbor Greedy Algoritması: Başlangıç şehirden başlayarak her adımda en az yol veren gidilmemiş şehri seçerek ilerler ve başlangıç konumuna geri döner. Bu algoritma hızlı ama bazı iyi sonuçları kaçırmaktadır.

Insertion Greedy Algoritması: Farklı başlangıç şekilleri vardır örneğin en küçük üçgen seçilerek başlar. Bu dokümanda bu algoritma için başlama şekli rastgele üçgen şeklindedir. Rastgele 3 şehir belirlenir ve daha sonrasında her adımda ziyaret edilmeyen şehirlerden en az mesafe artışı sağlayan şehir araya eklenerek ilerler. Bu algoritma Nearet Neigbor algoritmasından performans olarak daha iyi sonucu vermektedir.

Kod Açıklaması

```python
import networkx as nx
import matplotlib.pyplot as plt
import random
import math

random.seed(7)
```

1. Eklenen kütüphaneler ve karşılaştırma ve değerlendirmeyi kolayca yapabilmek için seed eklendi.

```python
# oklid mesafesi hesapla
def oklid( point1,  point2):
    x1, y1 = point1
    x2, y2 = point2
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

# Graf olustur
def generate_graph(points, size):
    # Random Konum belirle
    for i in range(size):
        points[i] = (random.uniform(0, 50), random.uniform(0, 50))
    G = nx.Graph()
    # Node oluştur
    for index, point in points.items():
        G.add_node(index, pos=point)
    # Tüm nodelara Kenar ekle
    for i in range(len(points)):
        for y in range(i + 1, len(points)):
            wei = oklid(points[i], points[y])
            G.add_edge(i , y, weight=wei)
    return G
```

2. Generate_graph fonksiyonunda Networkx kütüphanesini kullanarak (x, y) düzlemi üzerinde oluşturulan rastgele noktalar alınarak Node'lar ve oklid fonksiyonu ile tüm

Node'lar arasına hesaplanan uzaklık kenarları (Edge) kenar ağırlığı (weight) olarak eklendi. İşlemler sonucu Graf'ı döndürmektedir.

```python
# Yolun Toplam uzunlugunu hesapla
def sum_distance(G, path):
    sum = 0
    for i in range(len(path) - 1):
        sum += G[path[i]][path[i + 1]]["weight"]
    return sum
```

3. İzlenen yolun (path) toplam uzunluğu kenarların ağırlıkları toplanarak sum_distance fonksiyonunda hesaplandı ve döndürüldü.

```python
# Ziyaret edilmemiş Nodlar arasından eklenmesi en ucuz Node u seçer ve yola ekleyerek dondurur.
def best_node_add(G, path, number):
    distance = None
    sm_dis = float('inf')
    for node_unvisited in G.nodes():
        if node_unvisited not in path:
            for i in range(len(path) - 1):
                n1 = path[i]
                n2 = path[i + 1]
                distance = G[n1][node_unvisited]["weight"] + G[node_unvisited][n2]["weight"] - G[n1][n2]["weight"]
                if distance < sm_dis:
                    sm_dis = distance
                    where = i + 1
                    add_node = node_unvisited
    number += 1
    path.insert(where, add_node)
    G.nodes[add_node]['Labels'] = number
    return path
```

4. Best_node_add fonksiyonunda, henüz ziyaret edilmemiş Node'lar arasından, mevcut yola eklendiğinde en az mesafe artışı sağlayan Node yola eklenerek yol (path) döndürüldü. Eklenme sırası takibini kolaylaştırmak için number değişkeni kullanılarak labeller eklendi.

```python
def tps_insertion(G):
    # Random 3 Node belirle
    nodes = list(G.nodes())
    path = random.sample(nodes, 3)
    # ilk secilen Node lara 0 labeli eklendi.
    for node in G.nodes():
        if node in path:
            G.nodes[node]['Labels'] = 0
    path.append(path[0])
    number = 0
    # Tum Nodelar ziyaret edilene kadar calistir. En iyi node veren yolu gunceller
    while len(path) - 1 < len(G.nodes()):
        path = best_node_add(G, path, number)
        number += 1
    #Toplam yolu hesapla
    sum = sum_distance(G, path)
    print(path)
    return sum, path
```

5. Tps_insertion fonksiyonunda grafı alır ve graf içerisinden rastgele 3 Node seçilerek yola ve takip etmek için aynı sayı labeli eklendi. Number değişkeni best_node_add fonksiyonunda yola eklenen Node'lar için burada tanımlanarak fonksiyona gönderildi. Tüm Node'lar yola eklenene kadar while döngüsü

çalıştırıldı. Tüm yollar eklendikten sonra sum_distance fonksiyonu ile toplam yol hesaplandı. Taplam gidilen mesafe ve izlenen yol döndürülür.

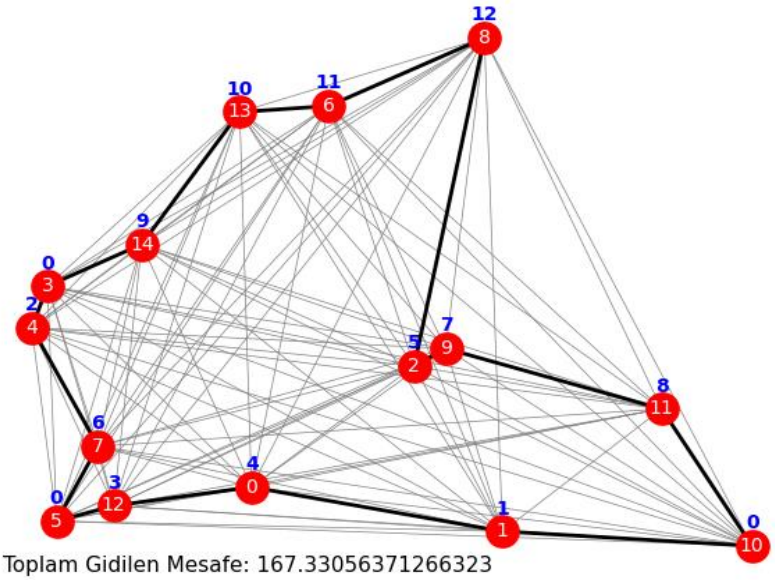
```python
# Grafı ve Secilen yolu çiz
def draw_graph(G, points, path_edge, tps_sum):
    nx.draw(G, pos=points , with_labels=True, node_color='red', edge_color="grey",
            font_color="white", node_size=280, font_size=10, width=0.5)
    nx.draw_networkx_edges(G, pos=points, edgelist=path_edge, width=2)
    label_points = {n: (x, y + 2) for n, (x, y) in points.items()}
    nx.draw_networkx_labels(G, pos=label_points, labels=nx.get_node_attributes(G, 'Labels'),
                            font_size=10, font_color="blue", font_weight="bold")
    plt.text(0, 0, f"Toplam Gidilen Mesafe: {tps_sum}", fontsize=11)
    plt.xlim(-3, 55)
    plt.ylim(-3, 55)
    plt.show()
```

6. draw_graph fonksiyonunda NetworkX ve Matplotlib ile grafiği görselleştirme yapıldı. Seçilen yollar belirginleştirildi. Node lara eklenen labeller ile algoritmanın ilerleyişi gözlemlendi. Ayrıca toplam mesafe aşağıya yazdırıldı.

```python
if __name__ == '__main__':
    points = {}
    path_edge = []
    G = generate_graph(points, 15) # 15 Node lu Graf uretir
    tps_sum, path = tps_insertion(G)
    # izlenen yol Kenarları belirtilmesi için
    path_edge = [(path[i], path[i+1]) for i in range(len(path)-1)]
    print(f"Greedy Insertion Algoritması ile TPS sonucu: {tps_sum}")
    draw_graph(G, points, path_edge, tps_sum)
```

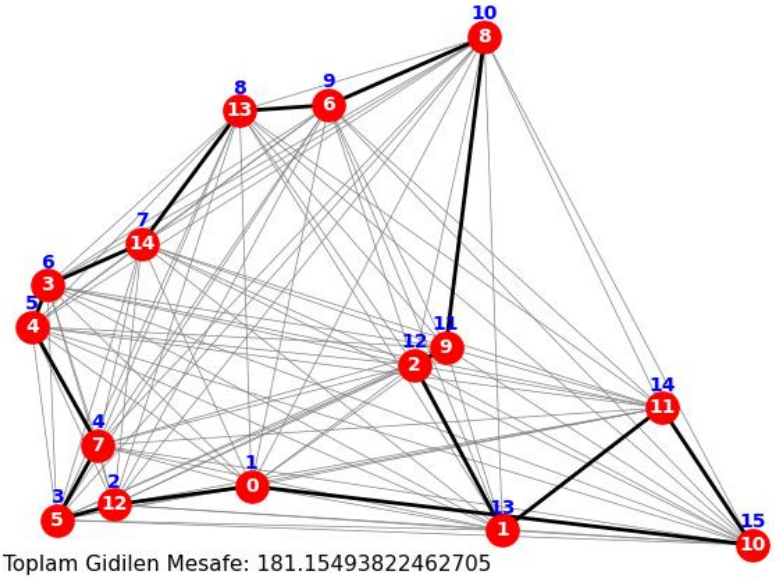
7. Ana program blogunda sırayla fonksiyonlar çalıştırıldı ve kısa yolun değeri yazdırıldı.

Sonuçlar



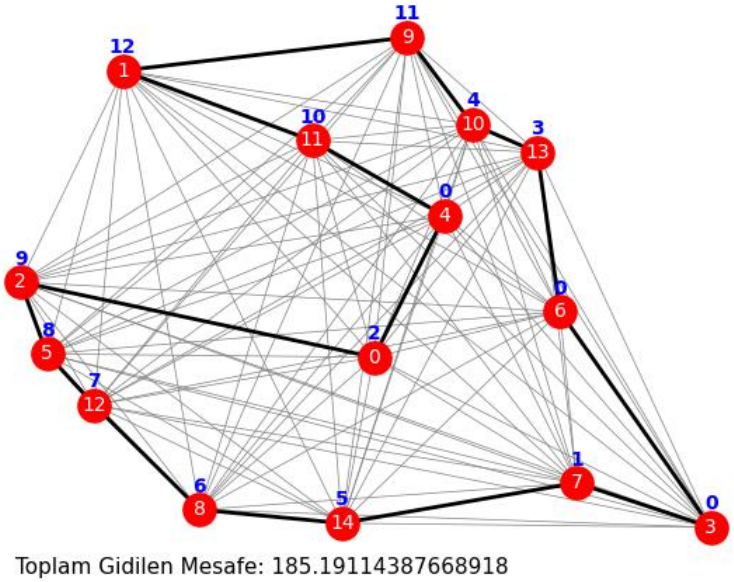
Şekil-1

Seed 7, oluşturulan Node sayısı 15, rastgele oluşturulan noktaların aralıkları 0 ile 50 arasında iken algoritma resimdeki gibi sonuç vermektedir. Başlangıç rastgele seçilen başlangıç Node'ları (3, 4, 10) olarak seçilip devamında en az mesafe artışı sağlayan Node eklenmesiyle devam etmiştir. Toplam uzaklık 167.33 olarak bulunmuştur.



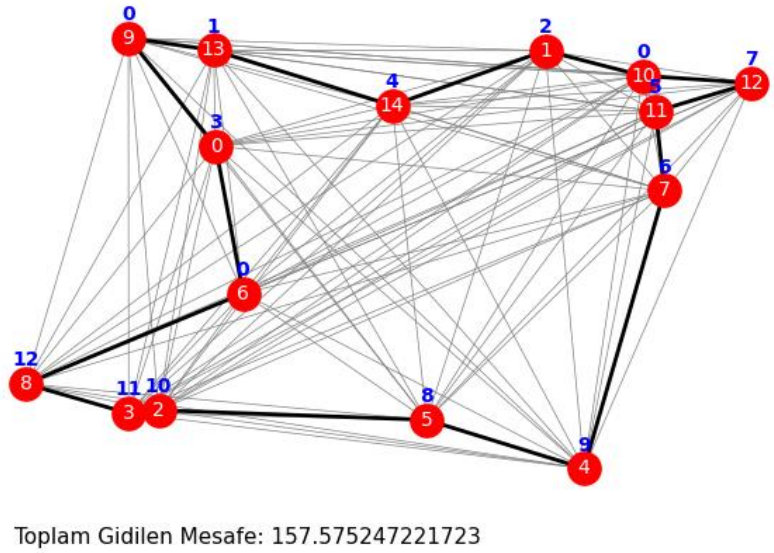
Şekil-2

Başlangıç Node u 0 ile başlayan ve diğer parametrelerde şekil-1 ile aynı olan uygulanan Nearest Neighbor algoritması ile ise 181.15 sonucunu bulunmuştur. Bu sonuçlar karşılaştırıldığında Insertion Algoritmasının daha doğru sonuçlar verdiği gözlemlenmiştir.



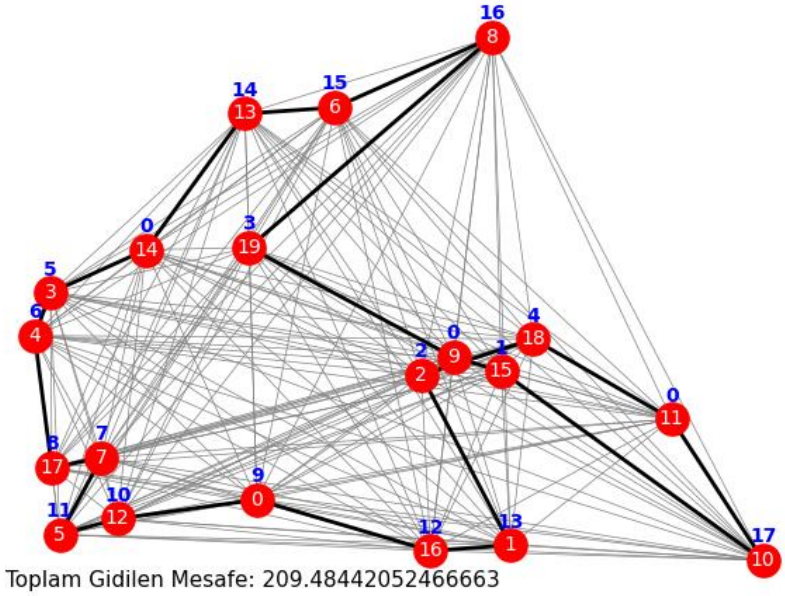
Şekil-3

Farklı Node dağılımlarını gözlemlemek için Şekil-1'de seed'i 9 yaparak şekil-3 sonuç alınmıştır.



Şekil-4

Seed 13 olduğunda ise şekil-4 sonucuna ulaşılmıştır.



Seed değeri Şekil-1'deki gibi 7 ve oluşturulacak Node sayısı 20 olduğunda ise Şekil-5 gözlemlenmektedir. Şekil-5'te gözle bakıldığında daha iyi sonuç görülmekte olup algoritmanın verimliliği düştüğü yorumu yapılabilmektedir.