Bilgisayar Oyunlarında Yapay Zekâ Ödev-2 Raporu

Metehan SÖZENLİ 25435004037

Ödev-2 kapsamında, Gezgin Satıcı Problemi (TSP) gerçek harita verileri kullanılarak çözülmüştür. Bu amaçla, OpenStreetMap (OSM) platformundan elde edilen yol ağı üzerinde Nearest Neighbor (En Yakın Komşu) heuristiği uygulanmıştır.

Bu yöntemle, belirli sayıda konumun (şehir düğümünün) en kısa mesafeyle dolaşılması hedeflenmiştir. Gerçek yol verilerinin kullanılması sayesinde, önceki ödevdeki rastgele nokta tabanlı çözüm yerine şehir trafiğine uygun ve yönlü bir graf yapısı elde edilmiştir.

Kod Açıklamaları

create_osm_tsp_graph() fonksiyonu tarafından, OpenStreetMap verileri kullanılarak belirtilen şehir için yol ağı indirilmektedir. Elde edilen graf üzerinde en büyük bağlı bileşen seçilmekte, ardından belirlenen “n” sayıda düğüm rastgele örneklenmektedir.

```python
def create_osm_tsp_graph(place: str, n: int, seed: int, network_type: str = "drive"):
    """OSM verilerinden TSP grafu oluşturur"""
    # 1) Yolu indir
    G = ox.graph_from_place(place, network_type=network_type, simplify=True)

    # 2) En büyük bileşeni al
    if isinstance(G, (nx.DiGraph, nx.MultiDiGraph)):
        components = nx.weakly_connected_components(G)   # yönlü graf için "weak"
    else:
        components = nx.connected_components(G)          # yönsüz ise normal CC

    largest_nodes = max(components, key=len)
    G = G.subgraph(largest_nodes).copy()

    # Kullanmıyoruz çünkü tek yönlü yolları korumak istiyoruz
    # G.to_undirected()

    # 4) TSP için node'ları sample et
    all_nodes = list(G.nodes)
    random.Random(seed).shuffle(all_nodes)
    selected_nodes = all_nodes[:n]

    return G, selected_nodes
```

NearestNeighborSolver sınıfı, Nearest Neighbor algoritmasını graf üzerinde doğrudan uygulamaktadır. Başlangıç düğümünden itibaren, henüz ziyaret edilmemiş düğümler arasından en yakın olanı seçilmektedir. Tüm düğümler gezildikten sonra başlangıç

noktasına geri dönülmektedir. Sonuç olarak, tur sırası oluşturularak düğümler arasındaki rota belirlenmektedir.

```python
class NearestNeighborSolver:
    def solve(self, G: Any, nodes: List[Any], start_node: Any, weight: str = "length") -> List[Any]:
        """Nearest Neighbor TSP solver - directly on graph"""
        unvisited = set(nodes)
        tour = [start_node]
        unvisited.remove(start_node)

        while unvisited:
            current = tour[-1]
            # Find nearest unvisited node
            nearest = min(unvisited, key=lambda node: nx.shortest_path_length(G, current, node, weight=weight))
            tour.append(nearest)
            unvisited.remove(nearest)

        tour.append(start_node)  # Return to start
        return tour
```

FoliumPlotter sınıfı ise, elde edilen rotanın etkileşimli harita üzerinde görselleştirilmesinden sorumludur. ***"folium"*** kütüphanesi kullanılarak OSM tabanlı bir harita oluşturulmakta ve başlangıç düğümü yeşil, bitiş düğümü kırmızı, diğer düğümler mavi renkle gösterilmektedir.

Düğümlere ilişkin konum bilgileri ve tur sırası popup ile gösterilmektedir. Rota yolları mavi renkle çizilirken, başlangıç noktasına geri dönüş yolu kesikli kırmızı çizgi ile görselleştirilmektedir.

```python
class FoliumPlotter:
    def render(self, G: Any, tsp_nodes: Sequence[Any], tour_idx: Sequence[int], out_html: str = "tsp_map.html", weight: str = "length"):

        lats = [G.nodes[n]['y'] for n in tsp_nodes]
        lngs = [G.nodes[n]['x'] for n in tsp_nodes]
        center_lat = sum(lats) / len(lats)
        center_lng = sum(lngs) / len(lngs)

        # Create map with only OpenStreetMap
        m = folium.Map(
            location=[center_lat, center_lng],
            zoom_start=12,
            control_scale=True,
            tiles='OpenStreetMap'
        )

        # Calculate segment distances for display
        segment_distances = []
        for i in range(len(tour_idx) - 1):
            u_idx = tour_idx[i]
            v_idx = tour_idx[i + 1]
            u_node = tsp_nodes[u_idx]
            v_node = tsp_nodes[v_idx]
            # Get shortest path length
            try:
                dist_m = nx.shortest_path_length(G, u_node, v_node, weight=weight)
                segment_distances.append(dist_m)
            except:
                segment_distances.append(0)

        total_dist_km = sum(segment_distances) / 1000.0

        # Add nodes with custom markers
        for idx, n in enumerate(tsp_nodes):
            # Determine color based on position in tour
            tour_position = tour_idx.index(idx) if idx in tour_idx else -1

            if tour_position == 0:
                # Start node
                color = 'green'
                icon = 'play'
                prefix = 'fa'
            elif tour_position == len(tour_idx) - 2:  # Second to last is the actual finish (last is return to start)
                # End node
                color = 'red'
                icon = 'stop'
                prefix = 'fa'
            else:
                # Regular node
                color = 'blue'
                icon = 'circle'
                prefix = 'fa'
```

```
    # Create popup with detailed info
    popup_html = f"""
    <div style="font-family: Arial; width: 200px;">
        <h4 style="margin: 0; color: {color};">Node {idx}</h4>
        <hr style="margin: 5px 0;">
        <b>Tour Position:</b> {tour_position}<br>
        <b>Coordinates:</b><br>
          <b>Lat:</b> {G.nodes[n]['y']:.6f}<br>
          <b>Lon:</b> {G.nodes[n]['x']:.6f}
    """

    # Add distance to next node if not last
    if tour_position < len(tour_idx) - 1 and tour_position >= 0:
        next_dist_km = segment_distances[tour_position] / 1000.0
        popup_html += f"<br><b>Distance to next:</b> {next_dist_km:.2f} km"

    popup_html += "</div>"

    folium.Marker(
        location=(G.nodes[n]['y'], G.nodes[n]['x']),
        popup=folium.Popup(popup_html, max_width=250),
        tooltip=f"Node {idx} (Position {tour_position})",
        icon=folium.Icon(color=color, icon=icon, prefix=prefix)
    ).add_to(m)

    # Add a circle marker with number label
    folium.CircleMarker(
        location=(G.nodes[n]['y'], G.nodes[n]['x']),
        radius=15,
        color=color,
        fill=True,
        fillColor=color,
        fillOpacity=0.3,
        weight=2,
    ).add_to(m)

    # Add text label with tour position
    if tour_position >= 0:
        folium.Marker(
            location=(G.nodes[n]['y'], G.nodes[n]['x']),
            icon=folium.DivIcon(html=f"""
                <div style="
                    font-size: 12px;
                    font-weight: bold;
                    color: white;
                    text-align: center;
                    text-shadow: 1px 1px 2px black;
                    margin-left: -6px;
                    margin-top: -6px;
                ">{tour_position}</div>
            """)
        ).add_to(m)
# Draw each segment separately to color the last one differently
for i in range(len(tour_idx) - 1):
    u_idx = tour_idx[i]
    v_idx = tour_idx[i + 1]
    u_node = tsp_nodes[u_idx]
    v_node = tsp_nodes[v_idx]

    # Get the path nodes for this segment
    try:
        segment_path = nx.shortest_path(G, u_node, v_node, weight=weight)
        segment_coords = [(G.nodes[n]['y'], G.nodes[n]['x']) for n in segment_path]

        # Check if this is the last segment (return to start)
        is_return_segment = (i == len(tour_idx) - 2)

        if is_return_segment:
            # Return segment in red/orange with dashed line
            folium.PolyLine(
                locations=segment_coords,
                weight=5,
                opacity=0.7,
                color='#ff6b6b',  # Red/orange color
                dash_array='10, 10',  # Dashed line
                tooltip=f"Return to Start ({segment_distances[i] / 1000.0:.2f} km)"
            ).add_to(m)
        else:
            # Regular segment in blue
            folium.PolyLine(
                locations=segment_coords,
                weight=5,
                opacity=0.8,
                color='#3388ff',  # Blue color
                tooltip=f"Segment {i+1} ({segment_distances[i] / 1000.0:.2f} km)"
            ).add_to(m)
    except:
        pass
```

```
    # Add statistics panel
    stats_html = f"""
    <div style="
        position: fixed;
        top: 10px;
        right: 10px;
        width: 220px;
        background-color: white;
        border: 2px solid #3388ff;
        border-radius: 8px;
        padding: 15px;
        font-family: Arial;
        box-shadow: 0 0 15px rgba(0,0,0,0.3);
        z-index: 1000;
    ">
        <h3 style="margin: 0 0 10px 0; color: #3388ff; border-bottom: 2px solid #3388ff; padding-bottom: 5px;">
            TSP Tour
        </h3>
        <table style="width: 100%; font-size: 14px;">
            <tr>
                <td><b>Nodes:</b></td>
                <td style="text-align: right; font-weight: bold;">{len(tsp_nodes)}</td>
            </tr>
            <tr>
                <td><b>Distance:</b></td>
                <td style="text-align: right; color: #d95534f; font-weight: bold;">{total_dist_km:.2f} km</td>
            </tr>
        </table>
    </div>
    """
    m.get_root().html.add_child(folium.Element(stats_html))

    m.save(out_html)
    return out_html
```

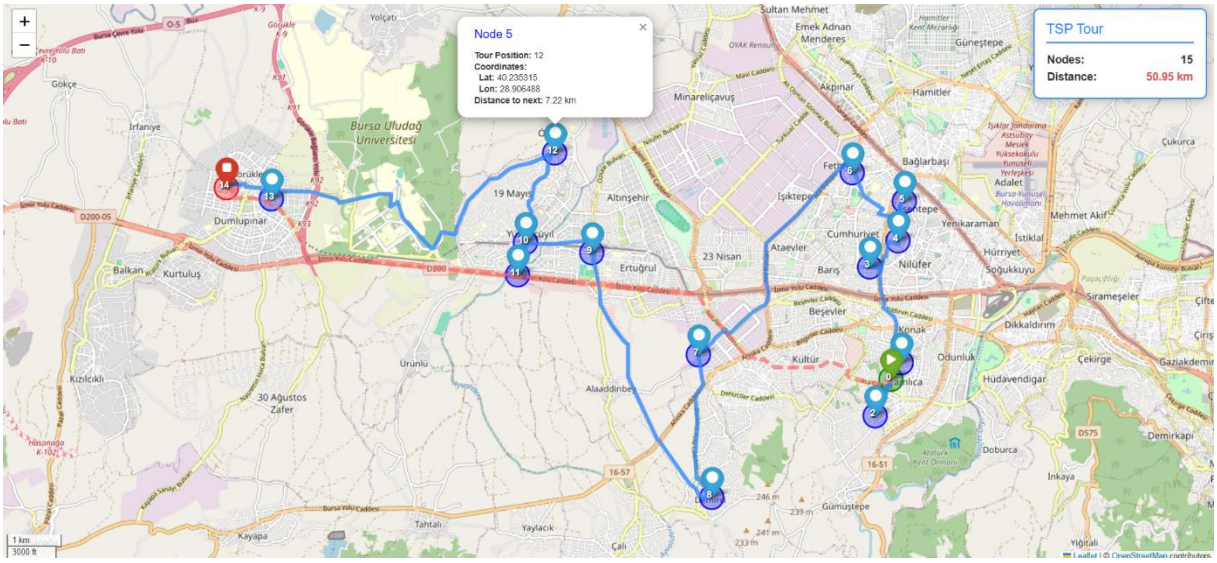
Programın çalıştırıldığı **main()** fonksiyonu, ilk olarak belirtilen şehir için yol ağı oluşturulmaktadır. Ardından rastgele seçilmiş düğümler arasında Nearest Neighbor yöntemi uygulanmakta ve turun toplam uzunluğu hesaplanmaktadır. Sonuçlar hem terminal üzerinde yazdırılmakta hem de oluşturulan rota *.html* dosyası olarak kaydedilmektedir.

```
def main():
    parser = argparse.ArgumentParser(description="Task 2: Real Map TSP with OSM Data")
    parser.add_argument("--place", type=str, default="Ankara, Turkey", help="OSM place string (e.g., 'Çankaya, Ankara, Turkey')")
    parser.add_argument("--n", type=int, default=12, help="Number of TSP nodes to sample")
    parser.add_argument("--seed", type=int, default=42, help="Random seed")
    parser.add_argument("--html", type=str, default="tsp_task2.html", help="Output HTML file")
    args = parser.parse_args()

    # Build real map graph with driving network only
    G, nodes = create_osm_tsp_graph(args.place, args.n, args.seed, network_type="drive")

    # Solve TSP using Nearest Neighbor directly on graph
    solver = NearestNeighborSolver()
    tour_nodes = solver.solve(G, nodes, start_node=nodes[0], weight="length")

    # Calculate total distance
    total_m = sum(nx.shortest_path_length(G, tour_nodes[i], tour_nodes[i+1], weight="length")
                  for i in range(len(tour_nodes)-1))
    print(f"Tour (node IDs): {tour_nodes}")
    print(f"Tour length ≈ {total_m/1000:.2f} km")

    # Create index mapping for visualization
    node_to_idx = {node: idx for idx, node in enumerate(nodes)}
    tour_idx = [node_to_idx[node] for node in tour_nodes]

    # Visualize on Folium
    plotter = FoliumPlotter()
    out = plotter.render(G, nodes, tour_idx, out_html=args.html, weight="length")
    print(f"Saved map: {out}")
```

Sonuç

Program çalıştırıldığında, Ankara haritası üzerinde 15 düğüm rastgele seçilmiş ve **Nearest Neighbor** heuristiği uygulanarak bir tur oluşturulmuştur. Elde edilen sonuçta, satıcının 0. düğümden başlayarak tüm düğümleri ziyaret ettiği ve yaklaşık **50,95 km** uzunluğunda bir rota kat ederek başlangıç noktasına geri döndüğü gözlemlenmiştir.

```
PS D:\Yüksek Lisans Dersler\Bilgisayar Oyunlarında Yapay Zeka\AI_in_Compter_Games_Assignments\Assignment-2> python task2.py --place "Nilufer,Bursa,Turkiye" --n 15 --seed 42 --html RealMap_tsp.html
Tour: [0, 3, 2, 8, 11, 9, 7, 4, 10, 6, 12, 1, 5, 14, 13, 0]
Tour length ≈ 51.82 km
Saved map: RealMap_tsp.html
```



Github Linki: https://github.com/BLM5026-AI-in-Computer-Games/hw2-metehansozenli