BLM5026 Bilgisayar Oyunlarında Yapay Zeka

Ekin Doğucan Akkaya
25435004005

Homework 3 Raporu

Ödevin kısaca hedefleri: Gelişmiş bir çözücü ekleyerek kendi algoritmamızla karşılaştırmak.

Ödevde istenilen anahtar konular

- İkinci bir TSP yaklaşımını uygulamak (örneğin, Google OR-Tools)
- Bilimsel standartlar kullanarak iki yöntemi karşılaştırmak (Kendi heuristicimiz vs OR-Tools):
    - Tekrarlanabilirlik için rastgele seed kullanmak.
    - En az 30 adet topoloji üretmek.
    - Ortalama tur uzunluğu ve çalışma süresini kıyaslamak.
- Sonuçları tablo/grafik kullanarak görselleştirmek.

Ödevin bulunduğu github linki: https://github.com/EkinAkkaya0/BLM5026_Homeworks

Ödev Kodu

```
import argparse
import random
import math
import time
import os

import networkx as nx
import osmnx as ox
import folium
import matplotlib.pyplot as plt

from ortools.constraint_solver import pywrapcp, routing_enums_pb2  # OR-Tools (pip
install ortools)

# Yol ağını indirdiğimiz fonksiyon.
def load_graph(place, network_type="drive", simplify=True):
    # Örnek: "Kadıköy, Istanbul, Turkey"
    # network_type: drive, walk, bike gibi olabilir.
    return ox.graph_from_place(place, network_type=network_type, simplify=simplify)

# Rastgele düğüm seçmek için fonksiyon
def pick_nodes(G, n, seed):
    rng = random.Random(seed)
    nodes = list(G.nodes())
    if n > len(nodes):
        raise ValueError("n, graf düğümlerinden büyük olamaz.")
    return rng.sample(nodes, k=n)

# Seçilen düğümlerin arasındaki en kısa mesafeyi hesapla
def shortest_lengths(G, nodes):
```

```
    dist = {}
    for i, u in enumerate(nodes):
        lengths = nx.single_source_dijkstra_path_length(G, u, weight="length")
        for v in nodes[i+1:]:
            d = lengths.get(v, float("inf"))
            dist[(u, v)] = d
            dist[(v, u)] = d
    return dist

# Problemimizin tam grafını oluşturan fonksiyon
def build_complete_graph(nodes, dist):
    H = nx.Graph()
    H.add_nodes_from(nodes)
    for i in range(len(nodes)):
        for j in range(i + 1, len(nodes)):
            u, v = nodes[i], nodes[j]
            d = dist[(u, v)]
            if math.isfinite(d):
                H.add_edge(u, v, weight=d)
    return H

# Greedy algoritması turumuz
def greedy_tour(H, start):
    unvisited = set(H.nodes())
    tour = [start]
    unvisited.remove(start)
    total = 0.0
    current = start

    # Her adımda şu anki düğüme en yakın olan düğüme gidiyoruz
    while unvisited:
        nxt = min(unvisited, key=lambda x: H[current][x]["weight"])
        total += H[current][nxt]["weight"]
        tour.append(nxt)
        unvisited.remove(nxt)
        current = nxt

    # En sonunda başlangıç noktasına tekrar dönüyoruz
    total += H[current][start]["weight"]
    tour.append(start)
    return tour, total

# Foliumda harita çizimi için fonksiyon
def draw_folium_map(G, tour, out_html="tsp_greedy_realmap.html"):

    # Merkez için (lat (latitude), lon (longitude)) ortalaması alıyoruz
    uniq = list(dict.fromkeys(tour))
    lat = sum(G.nodes[n]["y"] for n in uniq) / len(uniq)
    lon = sum(G.nodes[n]["x"] for n in uniq) / len(uniq)
    m = folium.Map(location=(lat, lon), zoom_start=13, control_scale=True)

    # Burada Nodelarımıza marker ekliyoruz
    for i, node in enumerate(tour[:-1], start=1):
        y, x = G.nodes[node]["y"], G.nodes[node]["x"]
        folium.Marker(
            location=(y, x),
            tooltip=f"P{i}",
            popup=f"P{i}<br>lat={y:.6f}, lon={x:.6f}"
```

```
        ).add_to(m)

    # Gerçek yolları tur olarak çiziyoruz. (kırmızı renkle)
    for u, v in zip(tour[:-1], tour[1:]):
        route = nx.shortest_path(G, u, v, weight="length")
        coords = [(G.nodes[n]["y"], G.nodes[n]["x"]) for n in route]
        folium.PolyLine(coords, color="red", weight=5, opacity=0.85).add_to(m)

    # Aynı isimde dosyaları üzerine yazmaması için dosya sonunda numaralandırma yapan
fonksiyon
    base, ext = os.path.splitext(out_html)
    i = 1
    new_name = out_html
    while os.path.exists(new_name):
        new_name = f"{base}_{i}{ext}"
        i += 1

    m.save(new_name)
    return new_name

# Problemimizin OR-Tools ile çözümünü yapan fonksiyon
def solve_tsp_ortools(H, start_node):
    # OR-Tools index tabanlı çalıştığı için düğümleri 0..N-1 şeklinde mapliyoruz.
    nodes = list(H.nodes())
    index_of = {node: i for i, node in enumerate(nodes)}
    n = len(nodes)

    # Mesafe matrisini oluşturan kısım
    dist_matrix = [[0] * n for _ in range(n)]
    for u, v, data in H.edges(data=True):
        i = index_of[u]
        j = index_of[v]
        # OR-Tools integer bir sayı beklediği için ağırlığı yuvarlayarak int
yapıyoruz.
        w = int(round(data["weight"]))
        dist_matrix[i][j] = w
        dist_matrix[j][i] = w

    start_index = index_of[start_node]

    manager = pywrapcp.RoutingIndexManager(n, 1, start_index)
    routing = pywrapcp.RoutingModel(manager)

    # OR-Tools her kenarın maliyetini bu fonksiyon ile sorgulatıyor.
    def distance_callback(from_index, to_index):
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return dist_matrix[from_node][to_node]

    transit_cb_idx = routing.RegisterTransitCallback(distance_callback)
    routing.SetArcCostEvaluatorOfAllVehicles(transit_cb_idx)

    search_params = pywrapcp.DefaultRoutingSearchParameters()
    # Başlangıç çözümünü hızlı olması için PATH_CHEAPEST_ARC ile üretiyoruz.
    search_params.first_solution_strategy =
routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC

    solution = routing.SolveWithParameters(search_params)
```

```
    if solution is None:
        raise RuntimeError("OR-Tools çözüm bulamadı.")

    # Çözümden turu çıkartan kısım
    index = routing.Start(0)
    route_indices = []
    while not routing.IsEnd(index):
        node_idx = manager.IndexToNode(index)
        route_indices.append(node_idx)
        index = solution.Value(routing.NextVar(index))
    # OR-Tools start_indexe dönerek bitiriyor

    route_nodes = [nodes[i] for i in route_indices]

    # Tur uzunluğunu H ağırlıklardan hesaplıyoruz.
    total = 0.0
    for u, v in zip(route_nodes[:-1], route_nodes[1:]):
        total += H[u][v]["weight"]

    return route_nodes, total

# çoklu topoloji için karşılaştırma deneyini yapan fonksiyon
def run_comparison_experiments(G, args):
    num_instances = args.instances
    base_seed = args.seed
    n = args.n
    start_index = args.start

    greedy_lengths = []
    ortools_lengths = []
    greedy_times = []
    ortools_times = []
    seeds_used = []

    # seed offset sayacı
    k = 0

    # İstediğimiz sayıda geçerli instance bulana kadar devam etmesini sağlayan kısım
    while len(greedy_lengths) < num_instances:
        seed = base_seed + k
        k += 1

        # Her deney için yeni bir topoloji seçmemizi sağlayan kısım
        tsp_nodes = pick_nodes(G, n, seed)
        if not (0 <= start_index < len(tsp_nodes)):
            raise ValueError("start index aralık dışında")

        dist = shortest_lengths(G, tsp_nodes)

        # Eğer bazı çiftler arasında hiç yol yoksa (inf mesafe), bu instance'ı atla
        if any(not math.isfinite(d) for d in dist.values()):
            print(f"Instance {len(greedy_lengths)+1:02d} | seed={seed} | SKIPPED
(disconnected nodes)")
            continue

        H = build_complete_graph(tsp_nodes, dist)
        start_node = tsp_nodes[start_index]
```

```
        # Greedy
        t0 = time.perf_counter()
        tour_greedy, len_greedy = greedy_tour(H, start_node)
        t1 = time.perf_counter()

        # OR-Tools
        t2 = time.perf_counter()
        tour_opt, len_opt = solve_tsp_ortools(H, start_node)
        t3 = time.perf_counter()

        greedy_lengths.append(len_greedy)
        ortools_lengths.append(len_opt)
        greedy_times.append(t1 - t0)
        ortools_times.append(t3 - t2)
        seeds_used.append(seed)

        print(
            f"Instance {len(greedy_lengths):02d} | seed={seed} | "
            f"Greedy={len_greedy/1000:.3f} km | OR-Tools={len_opt/1000:.3f} km"
        )

    # Ortalama değerleri hesaplayan kısım
    avg_g_len = sum(greedy_lengths) / num_instances
    avg_o_len = sum(ortools_lengths) / num_instances
    avg_g_time = sum(greedy_times) / num_instances
    avg_o_time = sum(ortools_times) / num_instances

    print("\n--- Ortalama Sonuçlar ---")
    print(f"Greedy ort. uzunluk:   {avg_g_len/1000:.3f} km")
    print(f"OR-Tools ort. uzunluk: {avg_o_len/1000:.3f} km")
    print(f"Greedy ort. süre:      {avg_g_time:.5f} s")
    print(f"OR-Tools ort. süre:    {avg_o_time:.5f} s")

    # Sonuçları hem tablo hem grafik olarak kaydeden kısım
    save_results_table(
        seeds_used,
        greedy_lengths,
        ortools_lengths,
        greedy_times,
        ortools_times,
        filename="tsp_comparison_table.png"
    )
    save_length_plot(
        greedy_lengths,
        ortools_lengths,
        filename="tsp_comparison_lengths.png"
    )
    save_time_plot(
        greedy_times,
        ortools_times,
        filename="tsp_comparison_times.png"
    )



# Matplotlib ile tabloyu görsel olarak kaydetmemize yarayan fonksiyon
def save_results_table(seeds, greedy_lengths, ortools_lengths, greedy_times,
ortools_times, filename):
```

```
    num_instances = len(seeds)
    fig, ax = plt.subplots(figsize=(10, min(0.4 * num_instances + 1, 12)))
    ax.axis('off')

    header = ["Inst", "Seed", "Greedy (km)", "OR-Tools (km)", "Greedy time (s)", "OR
time (s)", "G/OR oranı"]
    rows = []
    for i in range(num_instances):
        g_km = greedy_lengths[i] / 1000.0
        o_km = ortools_lengths[i] / 1000.0
        ratio = g_km / o_km if o_km > 0 else float("inf")
        rows.append([
            i + 1,
            seeds[i],
            f"{g_km:.3f}",
            f"{o_km:.3f}",
            f"{greedy_times[i]:.6f}",
            f"{ortools_times[i]:.6f}",
            f"{ratio:.3f}",
        ])

    table = ax.table(cellText=rows, colLabels=header, loc="center", cellLoc="center")
    table.auto_set_font_size(False)
    table.set_fontsize(8)
    table.scale(1, 1.2)

    fig.tight_layout()
    fig.savefig(filename, dpi=200)
    plt.close(fig)
    print(f"Tablo kaydedildi: {filename}")

# Tur uzunluklarını çizip kaydeden fonksiyon
def save_length_plot(greedy_lengths, ortools_lengths, filename):
    num_instances = len(greedy_lengths)
    x = list(range(1, num_instances + 1))

#  Her instance için Greedy ve OR-Tools tur uzunluklarını km cinsinden çizip PNG
olarak kaydediyoruz
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.plot(x, [g/1000 for g in greedy_lengths], marker="o", label="Greedy (km)")
    ax.plot(x, [o/1000 for o in ortools_lengths], marker="o", label="OR-Tools (km)")
    ax.set_xlabel("Instance")
    ax.set_ylabel("Tour length (km)")
    ax.set_title("Greedy vs OR-Tools Tour Lengths")
    ax.legend()
    ax.grid(True, linestyle="--", alpha=0.4)

    fig.tight_layout()
    fig.savefig(filename, dpi=200)
    plt.close(fig)
    print(f"Uzunluk grafiği kaydedildi: {filename}")

# Çalışma sürelerini çizip kaydeden fonksiyon
def save_time_plot(greedy_times, ortools_times, filename):
    num_instances = len(greedy_times)
    x = list(range(1, num_instances + 1))
```

```
# Bu sefer her instance için Greedy ve OR-Tools çalışma sürelerini saniye cinsinden
çizip PNG olarak kaydediyoruz
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.plot(x, greedy_times, marker="o", label="Greedy time (s)")
    ax.plot(x, ortools_times, marker="o", label="OR-Tools time (s)")
    ax.set_xlabel("Instance")
    ax.set_ylabel("Runtime (s)")
    ax.set_title("Greedy vs OR-Tools Runtimes")
    ax.legend()
    ax.grid(True, linestyle="--", alpha=0.4)

    fig.tight_layout()
    fig.savefig(filename, dpi=200)
    plt.close(fig)
    print(f"Zaman grafiği kaydedildi: {filename}")


def main():
    # Burası konsol üzerinden kodumuzu çalıştırırken düğüm sayısı, seed gibi değerleri
değiştirebilmek için olan kısım
    parser = argparse.ArgumentParser(description="Real Map TSP (Greedy vs OR-Tools)
with osmnx + folium")
    parser.add_argument("--place", default="Nilüfer, Bursa, Turkey", help="Şehir/bölge
adı (OSM)")
    parser.add_argument("--type", default="drive", choices=["drive", "walk", "bike"],
help="Ağ türü")
    parser.add_argument("--n", type=int, default=5, help="TSP düğüm sayısı")
    parser.add_argument("--seed", type=int, default=9, help="Seed")
    parser.add_argument("--start", type=int, default=0, help="Başlangıç düğümü indeksi
(0..n-1)")
    parser.add_argument("--out", default="tsp_greedy_realmap.html", help="Çıktı HTML
dosyası")
    # Assignment 3 için ek parametreler
    parser.add_argument("--compare", action="store_true",help="Greedy vs OR-Tools
karşılaştırmasını çalıştır")
    parser.add_argument("--instances", type=int, default=30,help="Karşılaştırma için
kaç farklı topoloji üretilecek")
    args = parser.parse_args()

    print(f"[+] '{args.place}' yol ağı indiriliyor ({args.type})...")
    G = load_graph(args.place, network_type=args.type)

    if args.compare:
        # Assignment 3: çoklu deney ve karşılaştırma
        run_comparison_experiments(G, args)
    else:
        # Düğümleri seç
        tsp_nodes = pick_nodes(G, args.n, args.seed)
        if not (0 <= args.start < len(tsp_nodes)):
            raise ValueError("start index aralık dışında")

        # İkili nodelarımız arasındaki en kısa mesafeyi kesapla
        dist = shortest_lengths(G, tsp_nodes)

        # Grafı oluştur
        H = build_complete_graph(tsp_nodes, dist)

        # Greedy turumuzu oluştur
```

```
        start_node = tsp_nodes[args.start]
        tour, total_m = greedy_tour(H, start_node)
        print(f"Tur uzunluğu ≈ {total_m/1000:.3f} km")
        print(f"Tur sırası (node id): {tour}")

        # Foliumda haritayı çiz
        out_file = draw_folium_map(G, tour, out_html=args.out)
        print(f"Kaydedildi: {out_file}  (tarayıcıda aç)")

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

Kodun Açıklaması

`def solve_tsp_ortools(H, start_node)` Fonksiyonu

Bu fonksiyon, Google OR-Tools kütüphanesini kullanarak TSP problemini çözer. OR-Tools, Greedy yaklaşımına göre çok daha gelişmiş bir optimizasyon yapısına sahiptir ve çoğu zaman daha kısa bir tur üretir. OR-Tools 0 ve N arasında olan tamsayı indexleriyle çalıştığı için, H grafındaki gerçek düğümler önce 0,1,2 gibi yeniden numaralandırılır. Böylece tüm hesaplamalar bu indexler üzerinden yapılır. H grafında bulunan tüm kenarların ağırlıkları (metre cinsinden) dist_matrix adlı 2 boyutlu bir matrise aktarılır. Bu matris OR-Tools'un temel girdi formatıdır. RoutingIndexManager ve RoutingModel sınıfları kullanılarak TSP modeli oluşturulur. Modelde hangi düğümün hangi sırada gezileceğine karar verecek arama algoritması çalıştırılır. OR-Tools her iki düğüm arasında geçişin maliyetini distance_callback fonksiyonuna sorar. Bu callback mesafe matrisinden uygun değeri döndürür. PATH_CHEAPEST_ARC ilk çözüm stratejisi kullanılarak hızlı bir başlangıç çözümü üretilir. Solver'ın bulduğu rota modellerden alınır, indexler orijinal düğüm ID'lerine geri çevrilir. Ayrıca tur uzunluğu metre cinsinden hesaplanarak döndürülür.

`def run_comparison_experiments(G, args)` Fonksiyonu

Bu fonksiyon, Greedy ve OR-Tools yöntemlerini bilimsel bir şekilde karşılaştırır. Burada amaç sadece tek bir örnek çözmek değil, çok sayıda rastgele topoloji üzerinden ortalama performans ölçütlerini (runtime, length gibi) çıkarmaktır. args.instances değeri kadar farklı TSP düğüm seti oluşturulur. Seed değerleri arttırılarak her instance için yeni bir rastgele seçim yapılır. Eğer seçilen düğümlerin bazıları birbirine ulaşamıyorsa (mesafe = inf), o instance kullanılmaz. Böylece hatalı deneyler sonuçları bozmamış olur. Her instance için Greedy algoritması çalıştırılır, tur uzunluğu ve çalışma süresi kaydedilir. Aynı instance için OR-Tools ile TSP çözülür, yine uzunluk ve süre ölçülür. OR-Tools'un ilk instance'da daha uzun sürmesinin nedeni "solver initialization" aşamasıdır. Her iki yöntemin tur uzunlukları ve çalışma süreleri ayrı listelerde tutulur. Sonrasında bu değerlerin ortalamaları hesaplanır. Deneylerden elde edilen tüm sonuçlar tablo (PNG) ve iki ayrı grafik (uzunluk ve runtime) olarak dosyaya kaydedilir.

`def save_results_table(seeds, greedy_lengths, ortools_lengths, greedy_times, ortools_times, filename)` Fonksiyonu

Bu fonksiyon, 30 instance boyunca elde edilen tüm verileri (Greedy vs OR-Tools uzunlukları ve süreleri) bir tablo halinde görsel dosyaya kaydeder. Instance başına sonuçları tablo satırlarına dönüştürür. Her satırda Instance numarası, Seed değeri, Greedy tur uzunluğu, OR-Tools tur uzunluğu, Her iki yöntemin çalışma süreleri ve İki algoritma arasındaki oran (karşılaştırma) verileri yer alır. Matplotlib kullanarak bir PNG tablo görseli oluşturur.

`def save_length_plot(greedy_lengths, ortools_lengths, filename)` Fonksiyonu

Bu fonksiyon, her instance için Greedy ve OR-Tools'un ürettiği tur uzunluklarını km cinsinden çizerek bir grafik oluşturur.

X ekseni: Instance numarası, Y ekseni: Tur uzunluğu (km);

Greedy algoritma = mavi çizgi, OR-Tools algoritması = turuncu çizgi ile temsil edilir. Grafik sonunda PNG dosyası olarak kaydedilir. Bu grafiği kullanarak algoritmaların çözüm kalitesini görsel olarak rahatlıkla karşılaştırabiliriz.

`def save_time_plot(greedy_times, ortools_times, filename)` Fonksiyonu

u fonksiyon, iki algoritmanın her instance için çalışma sürelerini çizer. Bu grafik sayesinde iki algoritma arasındaki çalışma süresi farkını kolayca gözlemleyebiliriz.

`def main()` Fonksiyonu

Bu kısım, programın ana çalışma bölümüdür. Komut satırından gelen parametreleri argparse kütüphanesiyle alır. Bu parametreler:

- --place: İndirilecek olan harita bölgesi (örneğin: “Nilüfer, Bursa, Turkey”)
- --type: Ağ türü (“drive”, “walk”, “bike”)
- --n: Rastgele seçilecek düğüm sayısı
- --seed: Rastgelelik tohumu (aynı sonuçları tekrar üretmek için)
- --start: Başlangıç düğümü indeksi
- --out: Oluşturulacak HTML dosyasının ismi
- --compare: Assignment 3 modunda çalıştırma ve Greedy–OR-Tools karşılaştırması
- --instances: Topoloji sayısı

Program sırasıyla şu işlemleri yapar:

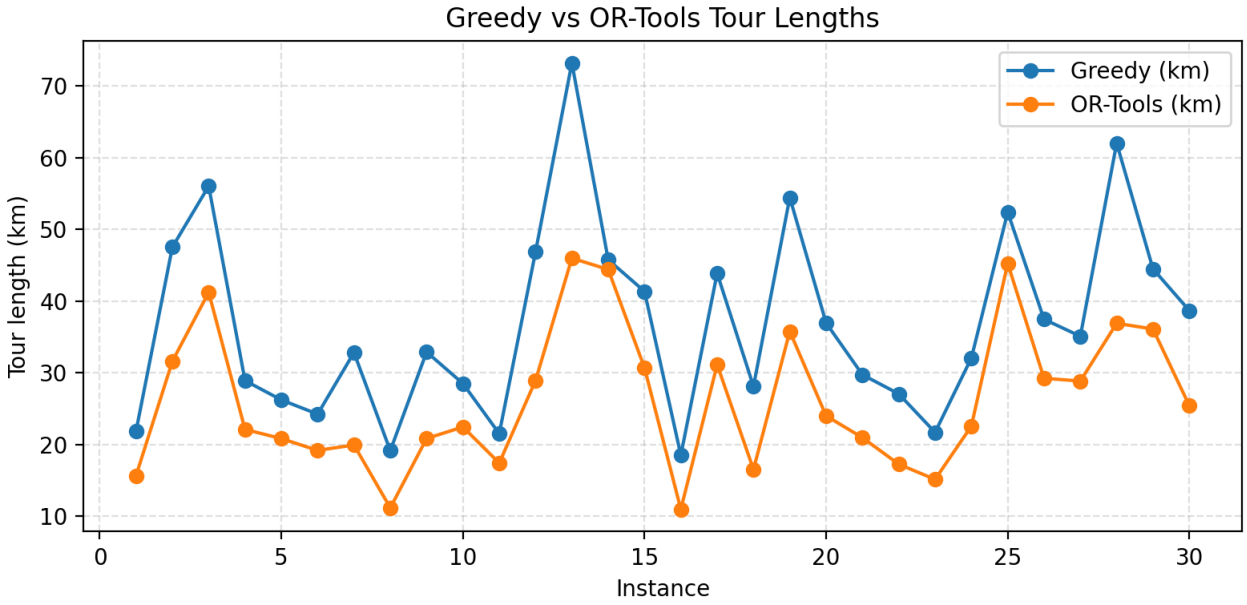
Eğer kullanıcı --compare parametresini verirse (Assignment 3 modu):

1. Seçilen bölgenin yol ağı load_graph() ile indirilir.
2. run_comparison_experiments() fonksiyonu çağrılır.
3. Kullanıcıya tüm deney sonuçlarının hangi dosyalara kaydedildiği terminalde bildirilir.

Eğer --compare verilmezse (Assignment 2 modu):

1. load_graph() fonksiyonuyla seçilen bölgenin yol ağını indirir.
2. pick_nodes() ile graf içinden rastgele n adet düğüm seçer.
3. shortest_lengths() fonksiyonuyla seçilen düğümler arasındaki en kısa mesafeleri hesaplar.
4. build_complete_graph() ile TSP’nin tam grafını oluşturur.
5. greedy_tour() ile en yakın komşu algoritmasını uygulayarak turu oluşturur.
6. draw_folium_map() fonksiyonunu çağırarak turu gerçek harita üzerinde görselleştirir.
7. Tur uzunluğunu kilometre cinsinden ekrana yazdırır ve oluşturduğu HTML dosyasının adını terminalde gösterir.

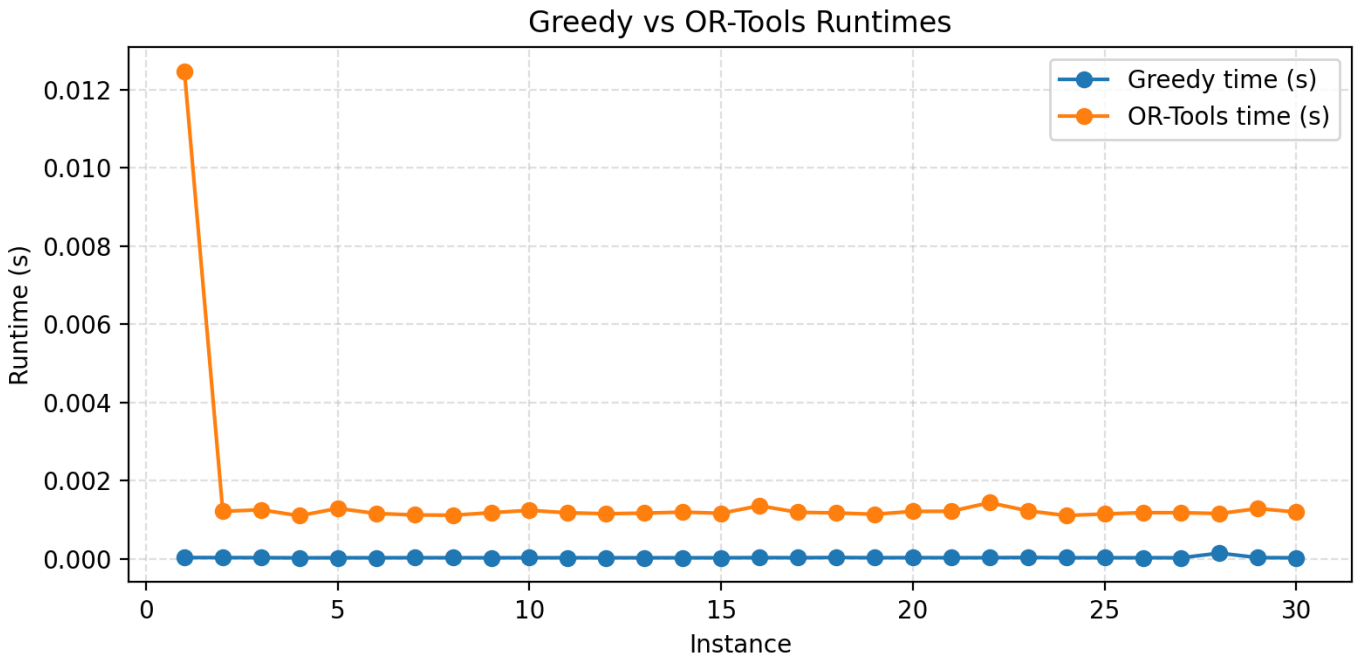
Karşılaştırma



Şekil 1: Tur Uzunlukları

Inst	Seed	Greedy (km)	OR-Tools (km)	Greedy time (s)	OR time (s)	G/OR oranı
1	9	21.866	15.647	0.000032	0.012471	1.397
2	10	47.535	31.606	0.000031	0.001214	1.504
3	11	56.050	41.219	0.000028	0.001253	1.360
4	12	28.900	22.088	0.000025	0.001100	1.308
5	13	26.208	20.817	0.000027	0.001290	1.259
6	14	24.224	19.161	0.000027	0.001158	1.264
7	15	32.820	19.922	0.000027	0.001122	1.647
8	16	19.126	11.142	0.000027	0.001114	1.717
9	17	32.899	20.826	0.000026	0.001179	1.580
10	18	28.491	22.431	0.000027	0.001240	1.270
11	20	21.500	17.402	0.000026	0.001176	1.235
12	21	46.885	28.925	0.000026	0.001152	1.621
13	22	73.233	45.989	0.000026	0.001167	1.592
14	23	45.771	44.411	0.000026	0.001194	1.031
15	24	41.348	30.678	0.000026	0.001164	1.348
16	25	18.496	10.944	0.000030	0.001357	1.690
17	26	43.879	31.127	0.000028	0.001189	1.410
18	27	28.095	16.504	0.000033	0.001173	1.702
19	28	54.443	35.776	0.000027	0.001138	1.522
20	29	36.944	23.979	0.000029	0.001212	1.541
21	30	29.722	20.944	0.000027	0.001216	1.419
22	31	27.052	17.234	0.000028	0.001435	1.570
23	32	21.679	15.116	0.000034	0.001227	1.434
24	33	32.059	22.549	0.000027	0.001107	1.422
25	34	52.416	45.264	0.000027	0.001146	1.158
26	35	37.465	29.229	0.000027	0.001180	1.282
27	36	35.086	28.841	0.000027	0.001178	1.217
28	37	62.033	36.898	0.000145	0.001157	1.681
29	38	44.494	36.101	0.000030	0.001286	1.232
30	39	38.673	25.415	0.000026	0.001195	1.522

Tablo 1: Karşılaştırma Tablosu



Şekil 2: Çalışma Hızı Karşılaştırması

Bu şekiller ve tabloya göre her topolojide Greedy algoritması daha uzun yollar izlemiştir. Bazı topolojilerde fark az da olsa birkaç topolojide fark çok büyüktür bu da Greedy algoritmasının zayıflıklarından birini göstermektedir. Ama konu Çalışma hızı(runtime) kıyaslamasına geldiğinde Greedy algoritmasının neredeyse anlık denebilecek kadar hızlı olduğunu gözlemleyebiliriz. Bu yönüyle hızlı bir algoritma olduğu çıkarımında bulunulabilir. OR-Tools ilk çalıştığında çalışma hızı(runtime) normalinden yüksektir. Bunun sebebi ilk çağırılışında yapılan içsel başlatma maliyetidir. Matematiksel olarak OR-Tools Greedy algoritmasından çok daha yavaş olmasına rağmen bu fark gerçek zaman açısından çok minimal ve neredeyse farkedilemez olduğu için yol optimizasyonu açısından OR-Tools tercihi hala açık bir şekilde daha mantıklıdır.