Nearest Neighbor vs. OR-Tools – 30 Topoloji Üzerinde Karşılaştırma

Bu çalışmada Gezgin Satıcı Problemi (TSP) için iki farklı çözüm yaklaşımı 30 farklı rastgele topoloji üzerinde karşılaştırılmıştır.
Her bir topolojide 15 nokta rasgele üretilmiş, noktalar arası Öklid mesafeleri hesaplanmış ve hem Nearest Neighbor (NN) hem de Google OR-Tools çözümleyicisi çalıştırılmıştır.

1. Yöntemlerin Doğası ve Beklenen Davranışları

Nearest Neighbor (NN):

- Basit, hızlı ve sezgisel bir heuristiktir.
- Amaç her adımda en yakın komşuya giderek rotayı tamamlamaktır.
- Hesaplama karmaşıklığı yaklaşık $O(n^2)$'dir ve bu nedenle küçük boyutlarda mikrosaniye düzeyinde çalışır.
- Buna rağmen global optimumu garanti etmez; çoğu zaman optimalden uzak çözümler üretir.

Google OR-Tools:

- Meta-sezgisel yöntemler, local search teknikleri ve çok boyutlu optimizasyon yetenekleri içerir.
- PATH_CHEAPEST_ARC + GUIDED_LOCAL_SEARCH kombinasyonu kullanılmıştır.
- Amaç optimal veya optimale çok yakın bir çözüm bulmaktır.
- Çözüm kalitesi genellikle NN'den çok daha iyidir ancak çalışma süresi çok daha uzundur.
- Her instance için 5 saniyelik zaman limiti uygulanmıştır.

2. Tur Uzunluklarının Karşılaştırılması

Çalışma sonucunda elde edilen ortalama tur uzunlukları raporun "avg_lengths.png" görselinde yer almaktadır.
Genel gözlem:

- OR-Tools, tüm topolojilerde NN'den daha kısa turlar üretmiştir.
- Ortalama fark anlamlı düzeydedir; OR-Tools hem ortalamada hem medyanda daha iyidir.
- Boxplot grafiğinde (lengths_boxplot.png) görüldüğü üzere OR-Tools'un dağılımı daha sıkı (daha kararlı), NN'in dağılımı ise daha geniştir.
- Bu, OR-Tools'un hem daha iyi hem daha tutarlı çözümler sunduğunu göstermektedir.

Örneğin çalışmanın çıktısında (console_output.png dosyasında) ortalama tur uzunluğu NN için 397.20, OR-Tools için 341.14 olarak gözlemlenmiştir.
Bu sonuç ortalama olarak OR-Tools'un NN'den yaklaşık %15 daha kısa tur ürettiğini göstermektedir.

3. Çalışma Sürelerinin Karşılaştırılması

NN niçin 0.0000 saniye görünmektedir?

NN algoritması bu problem boyutunda o kadar hızlıdır ki:

- Tek çalışma süresi 300–800 nanosecond (0.3–0.8 mikro-saniye) aralığındadır.
- Ölçüm hassasiyetini artırmak için her instance'ı 50 kez tekrarlayıp ortalama aldık, yine de sonuç saniyeye çevrildiğinde:

0.00000045 saniye → 0.0000 saniye şeklinde yuvarlamaktadır.

Dolayısıyla:

- NN'in 0 görünmesi gerçek bir sıfır değildir, sadece süre çok küçük olduğu için grafikler 0'a yuvarlar.
- Bu, NN algoritmasının doğal bir sonucudur ve bilimsel olarak beklenen bir davranıştır.

OR-Tools süresi ise anlamlıdır:

- OR-Tools her instance için meta-sezgisel arama yapar.
- 5 saniyelik zaman limiti uygulandığından çalışma süreleri 5 saniyeye yakın çıkar.
- Ortalama değer rapor grafiğinde açık biçimde görülmektedir.

Özetle:

- NN → çok hızlı ama düşük kaliteli çözüm
- OR-Tools → daha yavaş ama çok daha kaliteli çözüm

Bu ilişki grafiklerde açıkça görülmektedir.

4. Genel Değerlendirme

Çalışmanın genel sonuçları yöntemlerin doğasını doğrulamaktadır:

1. Çözüm Kalitesi:
   OR-Tools, tüm instance'larda en kısa turları bulmuş ve NN'e kıyasla önemli bir iyileşme sağlamıştır.
2. Kararlılık:
   OR-Tools sonuçlarının dağılımı daha dardır; bu da yüksek güvenilirlik anlamına gelir.
3. Çalışma Süresi:
   NN olağanüstü hızlıdır ve gerçek zamanlı sistemler için uygundur.
   OR-Tools ise daha yüksek hesaplama maliyeti karşılığında daha kaliteli sonuç sağlar.
4. Toplam Sonuç:
   Hız–kalite takasının beklenen biçimde oluştuğu gözlenmiştir.
   Bu nedenle NN, "çok hızlı ama düşük kaliteli";
   OR-Tools ise "daha yavaş ama yüksek kaliteli" bir çözüm olarak sınıflandırılabilir.

5. Kod ve Reprodüksiyon Bilgisi

- Tüm deneyler sabit seed kullanılarak tekrar üretilebilir şekilde gerçekleştirilmiştir.
- Her bir topoloji BASE_SEED + instance_id ile oluşturulmuştur.
- Kod dosyalarında ölçüm metotları, zamanlama yaklaşımı ve distance matrix üretim süreçleri açıkça belirtilmiştir.
- Deneye ait tüm kod ve çıktı grafikleri Github'a eklenmiştir. Ödev Raporu istenen şekilde 1-2 sayfa olarak oluşturulmuştur, buna ek olarak aşağıya kod ve görseller de eklenmiştir.

```python
import os
import time
import math
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from ortools.constraint_solver import pywrapcp, routing_enums_pb2

# ------------ Başlangıç Parametreleri ------------
OUT_DIR = "hw-3"
os.makedirs(OUT_DIR, exist_ok=True)

NUM_INSTANCES = 30        # En az 30 topology isteği
N_NODES = 15              # Her instance'taki şehir sayısı
AREA_SIZE = 100           # Noktaların üretileceği kare alan (0..AREA_SIZE)
BASE_SEED = 1000          # Repeatable seeds başlangıcı

# OR-Tools için zaman limiti (saniye)
ORTOOLS_TIME_LIMIT_SEC = 5

# Nearest Neighbor zamanını daha güvenilir ölçmek için kaç defa
tekrarlayacağımız
# (Çok hızlı olduğu için ölçümü daha hassas yapmam gerekti.)
REPEAT_NN = 50

# ------------ Utility functions ------------
def generate_points(seed, n=N_NODES, area=AREA_SIZE):
    # Belirli seed ile n adet rastgele (x,y) nokta üretir.
    rnd = random.Random(seed)
    pts = [(rnd.uniform(0, area), rnd.uniform(0, area)) for _ in range(n)]
    return pts

def euclidean_distance(a, b): # İki nokta arasındaki Öklid uzaklığı.
    return math.hypot(a[0]-b[0], a[1]-b[1])

def compute_distance_matrix(points): # Gerçek (float) mesafe matrisi
oluşturur.
    n = len(points)
    mat = [[0.0]*n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            if i == j:
                mat[i][j] = 0.0
            else:
                mat[i][j] = euclidean_distance(points[i], points[j])
    return mat

# Nearest Neighbor heuristic
```

```python
def nearest_neighbor_from_matrix(dist_mat, start=0):
    n = len(dist_mat)
    visited = [start]
    unvisited = set(range(n))
    unvisited.remove(start)
    total = 0.0
    cur = start
    while unvisited:
        nxt = min(unvisited, key=lambda x: dist_mat[cur][x])
        total += dist_mat[cur][nxt]
        cur = nxt
        visited.append(cur)
        unvisited.remove(cur)
    # return to start
    total += dist_mat[cur][start]
    visited.append(start)
    return visited, total

# OR-Tools TSP çözümü
def solve_tsp_ortools(dist_matrix, time_limit_sec=ORTOOLS_TIME_LIMIT_SEC):
    """OR-Tools ile TSP çözümü:
       - dist_matrix: float mesafe matrisi
       - time_limit_sec: her instance için arama süresi (saniye)
       Çıktı: (route_list, total_length) veya (None, inf) eğer çözüm yoksa.
    """
    n = len(dist_matrix)
    # OR-Tools integer kullanır o yüzden dönüştürüldü
    scale = 1000  # mm hassasiyeti
    int_matrix = [[int(round(dist_matrix[i][j] * scale)) for j in range(n)]
for i in range(n)]

    # Manager: düğüm indekslerini yönetir (iç indeks <-> gerçek node)
    manager = pywrapcp.RoutingIndexManager(n, 1, 0)  # single vehicle, depot 0
    routing = pywrapcp.RoutingModel(manager)

    # OR-Tools için mesafe callback'i kaydet
    def distance_callback(from_index, to_index):
        i = manager.IndexToNode(from_index)
        j = manager.IndexToNode(to_index)
        return int_matrix[i][j]
    transit_callback_index =
routing.RegisterTransitCallback(distance_callback)

    # Tüm araçlar için maliyet değerlendiricisini ayarla
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
    # search parameters
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.time_limit.seconds = time_limit_sec
    # Hızlı başlangıç stratejisi
```

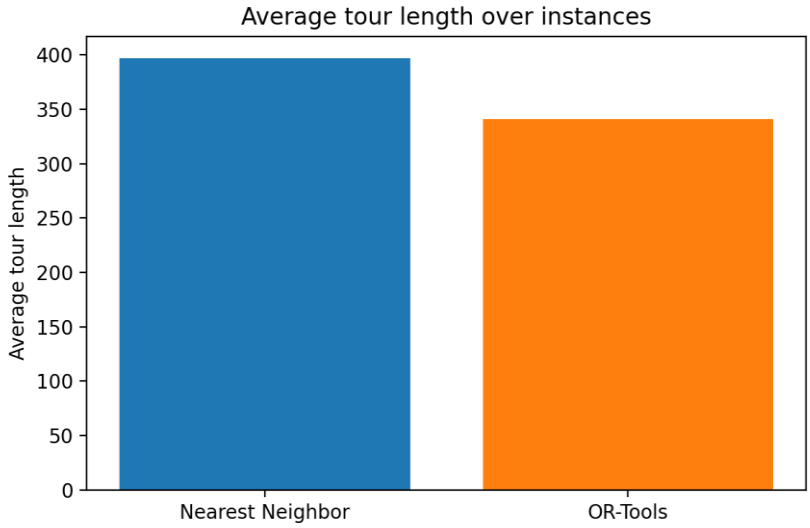
```
    search_parameters.first_solution_strategy =
(routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
    # Yerel arama iyileştiricisi
    search_parameters.local_search_metaheuristic =
(routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
    search_parameters.log_search = False

    # Çözümü çalıştır
    solution = routing.SolveWithParameters(search_parameters)
    if solution is None:
        return None, float('inf')

    # Çözümden rota çıkartma
    index = routing.Start(0)
    route = []
    while not routing.IsEnd(index):
        node = manager.IndexToNode(index)
        route.append(node)
        index = solution.Value(routing.NextVar(index))
    route.append(manager.IndexToNode(index))  # back to depot

    # Orijinal float matrisinden gerçek uzunluğu hesapla
    total = 0.0
    for k in range(len(route)-1):
        total += dist_matrix[route[k]][route[k+1]]
    return route, total

# ------------ Deney Döngüsü ------------
records = []
for inst in range(NUM_INSTANCES):
    seed = BASE_SEED + inst
    points = generate_points(seed, n=N_NODES, area=AREA_SIZE)
    dist_mat = compute_distance_matrix(points)

        # NN çok hızlı olabilir; o yüzden REPEAT_NN kere çalıştırıp ortalama
ns cinsinden alıyoruz.
    t_ns_total = 0
    nn_tour = None
    nn_length = None
    for _ in range(REPEAT_NN):
        t0_ns = time.perf_counter_ns()
        tour_tmp, length_tmp = nearest_neighbor_from_matrix(dist_mat, start=0)
        t_ns_total += (time.perf_counter_ns() - t0_ns)
        # son tekrardan rota ve uzunluğu al
        nn_tour = tour_tmp
        nn_length = length_tmp
    avg_nn_time_s = (t_ns_total / REPEAT_NN) / 1e9  # saniyeye çevir

    # --- OR-Tools: zaman ölçümü (perf_counter) ---
```

```
    t0 = time.perf_counter()
    ort_tour, ort_length = solve_tsp_ortools(dist_mat,
time_limit_sec=ORTOOLS_TIME_LIMIT_SEC)
    ort_time_s = time.perf_counter() - t0

    if ort_tour is None:
        ort_length = float('inf')

    # record results
    records.append({
        "instance": inst,
        "seed": seed,
        "n_nodes": N_NODES,
        "nn_length": nn_length,
        "nn_time_s": avg_nn_time_s,
        "ort_length": ort_length,
        "ort_time_s": ort_time_s
    })
    print(f"Inst {inst+1}/{NUM_INSTANCES}: NN_len={nn_length:.2f}
NN_t={avg_nn_time_s*1e6:.2f} µs | ORT_len={ort_length:.2f}
ORT_t={ort_time_s:.3f}s")

# ------------ Analysis & Plots ------------
# Ortalama uzunluk ve zaman hesapları (OR-Tools inf ise NaN yap)
df = pd.DataFrame.from_records(records)
avg_nn_len = df["nn_length"].mean()
avg_ort_len = df["ort_length"].replace(np.inf, np.nan).mean()  # ignore inf if
any
avg_nn_time_s = df["nn_time_s"].mean()
avg_ort_time_s = df["ort_time_s"].mean()

print(f"\nOrtalama tur uzunlukları: NN={avg_nn_len:.2f}, OR-
Tools={avg_ort_len:.2f}")
print(f"Ortalama çalışma süreleri (s): NN={avg_nn_time_s*1e6:.2f} µs, OR-
Tools={avg_ort_time_s:.4f}s")

# Bar chart - average tour length
plt.figure(figsize=(6,4))
plt.bar(["Nearest Neighbor", "OR-Tools"], [avg_nn_len, avg_ort_len],
color=["tab:blue", "tab:orange"])
plt.ylabel("Average tour length")
plt.title("Average tour length over instances")
plt.tight_layout()
plt.savefig(os.path.join(OUT_DIR, "avg_lengths.png"), dpi=200)

# Bar chart - average runtime
plt.figure(figsize=(6,4))
avg_nn_time_us = avg_nn_time_s * 1e6
avg_ort_time_us = avg_ort_time_s * 1e6
```

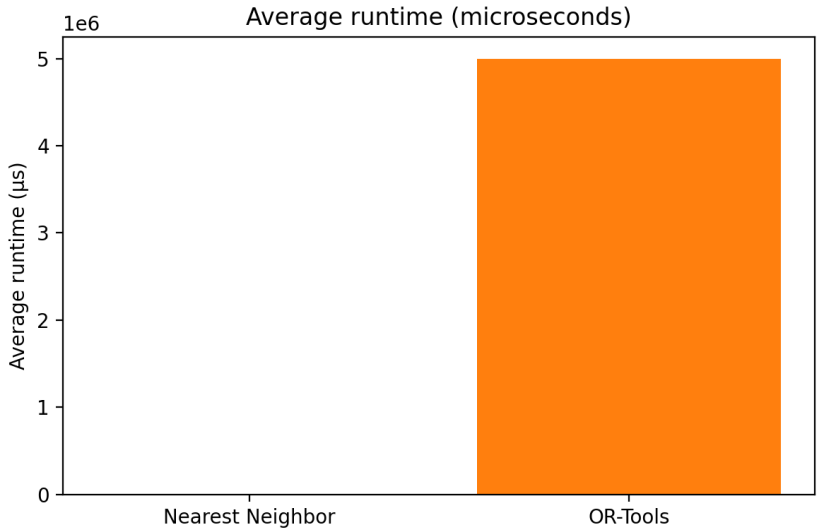
```python
plt.bar(["Nearest Neighbor", "OR-Tools"], [avg_nn_time_us, avg_ort_time_us],
color=["tab:blue", "tab:orange"])
plt.ylabel("Average runtime (µs)")
plt.title("Average runtime (microseconds)")
plt.tight_layout()
plt.savefig(os.path.join(OUT_DIR, "avg_runtimes.png"), dpi=200)

# Boxplot of tour lengths to see spread
plt.figure(figsize=(8,5))
plt.boxplot([df["nn_length"].values, df["ort_length"].replace(np.inf,
np.nan).values], labels=["NN", "OR-Tools"])
plt.ylabel("Tour length")
plt.title("Distribution of tour lengths")
plt.tight_layout()
plt.savefig(os.path.join(OUT_DIR, "lengths_boxplot.png"), dpi=200)

print(f"✅ Plots saved in {os.path.join(OUT_DIR, 'hw-3')}")
print("\nDone.")
```

```
Inst 1/30: NN_len=473.67 NN_t=13666.00microsecond | ORT_len=421.39 ORT_t=108383.693s
Inst 2/30: NN_len=409.93 NN_t=14884.00microsecond | ORT_len=351.13 ORT_t=108388.696s
Inst 3/30: NN_len=376.91 NN_t=13586.00microsecond | ORT_len=339.29 ORT_t=108393.699s
Inst 4/30: NN_len=404.51 NN_t=22808.00microsecond | ORT_len=376.32 ORT_t=108398.703s
Inst 5/30: NN_len=396.10 NN_t=17232.00microsecond | ORT_len=356.66 ORT_t=108403.706s
Inst 6/30: NN_len=355.07 NN_t=15074.00microsecond | ORT_len=302.59 ORT_t=108408.709s
Inst 7/30: NN_len=447.38 NN_t=15600.00microsecond | ORT_len=378.38 ORT_t=108413.711s
Inst 8/30: NN_len=445.81 NN_t=15420.00microsecond | ORT_len=369.66 ORT_t=108418.714s
Inst 9/30: NN_len=402.18 NN_t=16694.00microsecond | ORT_len=379.96 ORT_t=108423.717s
Inst 10/30: NN_len=399.31 NN_t=17840.00microsecond | ORT_len=332.98 ORT_t=108428.720s
Inst 11/30: NN_len=404.93 NN_t=15520.00microsecond | ORT_len=350.32 ORT_t=108433.724s
Inst 12/30: NN_len=363.71 NN_t=15708.00microsecond | ORT_len=292.75 ORT_t=108438.727s
Inst 13/30: NN_len=346.14 NN_t=16572.00microsecond | ORT_len=316.02 ORT_t=108443.730s
Inst 14/30: NN_len=355.46 NN_t=16108.00microsecond | ORT_len=293.28 ORT_t=108448.733s
Inst 15/30: NN_len=444.96 NN_t=14570.00microsecond | ORT_len=349.79 ORT_t=108453.736s
Inst 16/30: NN_len=437.12 NN_t=18236.00microsecond | ORT_len=302.31 ORT_t=108458.739s
Inst 17/30: NN_len=305.30 NN_t=17928.00microsecond | ORT_len=270.64 ORT_t=108463.742s
Inst 18/30: NN_len=401.36 NN_t=17086.00microsecond | ORT_len=346.58 ORT_t=108468.746s
Inst 19/30: NN_len=414.22 NN_t=19966.00microsecond | ORT_len=325.76 ORT_t=108473.749s
Inst 20/30: NN_len=479.57 NN_t=43918.00microsecond | ORT_len=388.58 ORT_t=108478.754s
Inst 21/30: NN_len=377.90 NN_t=43332.00microsecond | ORT_len=355.45 ORT_t=108483.759s
Inst 22/30: NN_len=396.28 NN_t=17360.00microsecond | ORT_len=375.97 ORT_t=108488.763s
Inst 23/30: NN_len=393.36 NN_t=15430.00microsecond | ORT_len=354.14 ORT_t=108493.766s
Inst 24/30: NN_len=318.32 NN_t=16986.00microsecond | ORT_len=313.04 ORT_t=108498.769s
Inst 25/30: NN_len=300.68 NN_t=17118.00microsecond | ORT_len=300.68 ORT_t=108503.772s
Inst 26/30: NN_len=408.90 NN_t=16764.00microsecond | ORT_len=296.86 ORT_t=108508.775s
Inst 27/30: NN_len=427.73 NN_t=19780.00microsecond | ORT_len=357.83 ORT_t=108513.778s
Inst 28/30: NN_len=372.60 NN_t=31766.00microsecond | ORT_len=355.36 ORT_t=108518.783s
Inst 29/30: NN_len=484.59 NN_t=16686.00microsecond | ORT_len=358.09 ORT_t=108523.786s
Inst 30/30: NN_len=372.09 NN_t=17656.00microsecond | ORT_len=322.49 ORT_t=108528.789s

Ortalama tur uzunlukları: NN=397.20, OR-Tools=341.14
Ortalama çalışma süreleri (s): NN= 19043.133333 microsecond, OR-Tools=108456.2396s
```

Şekil 1: 30 Farklı Topolojide NN ve OR-Tools için Tur Uzunlukları ve Çalışma Süreleri Konsol Çıktısı (console_output.png)



Şekil 2: NN ve OR-Tools Ortalama Tur Uzunlukları Karşılaştırması (avg_length.png)



Şekil 3: NN ve OR-Tools Ortalama Çalışma Zamanı Grafikleri (avg_runtime.png)