Bilgisayar Oyunlarında Yapay Zeka

Homework Series: "Progressive TSP Research"

Assignment 4 – AI Technique Integration

Ödev Raporu

Şevval Uzar – 25435004013

1. Giriş

Gezgin Satıcı Problemi (TSP) için kullanılan üç farklı çözüm algoritması karşılaştırılmıştır. İncelenen yöntemler Nearest Neighbor (NN), OR-Tools TSP ve Genetic Algorithm'dir. Bu üç yaklaşım aynı problem örnekleri üzerinde test edilerek tur uzunluğu ve hesaplama süresi açısından karşılaştırılmıştır.

2. Yöntemler

Bu assignment'ta TSP problemini çözmek için üç farklı yaklaşım kullanılmıştır.

2.1 Nearest Neighbour (NN)

Nearest Neighbour TSP için kullanılan en basit ve en hızlı sezgisel yöntemlerden biridir. Bu algoritma seçilen bir başlangıç noktasından başlayarak her adımda henüz ziyaret edilmemiş şehirler arasından en yakın olanı tercih eder. Bu sayede çok kısa sürede bir tur oluşturur. Ancak her adımda yalnızca en yakın seçeneğe odaklandığı için daha iyi bir genel çözümü gözden kaçırabilir ve başlangıçta seçilen şehir elde edilen sonucun optimalliğini etkileyebilir.

2.2 Google OR-Tools

PATH_CHEAPEST_ARC her adımda maliyeti en düşük bağlantıları seçerek kısa sürede iyi kalitede çözümler üretmeyi amaçlar. OR-Tools çözüm yollarını daha planlı ve kapsamlı bir şekilde taradığı için Nearest Neighbour algoritmasına göre genellikle daha kısa turlar elde edilmesini sağlar.

2.3 Genetic Algorithm (GA)

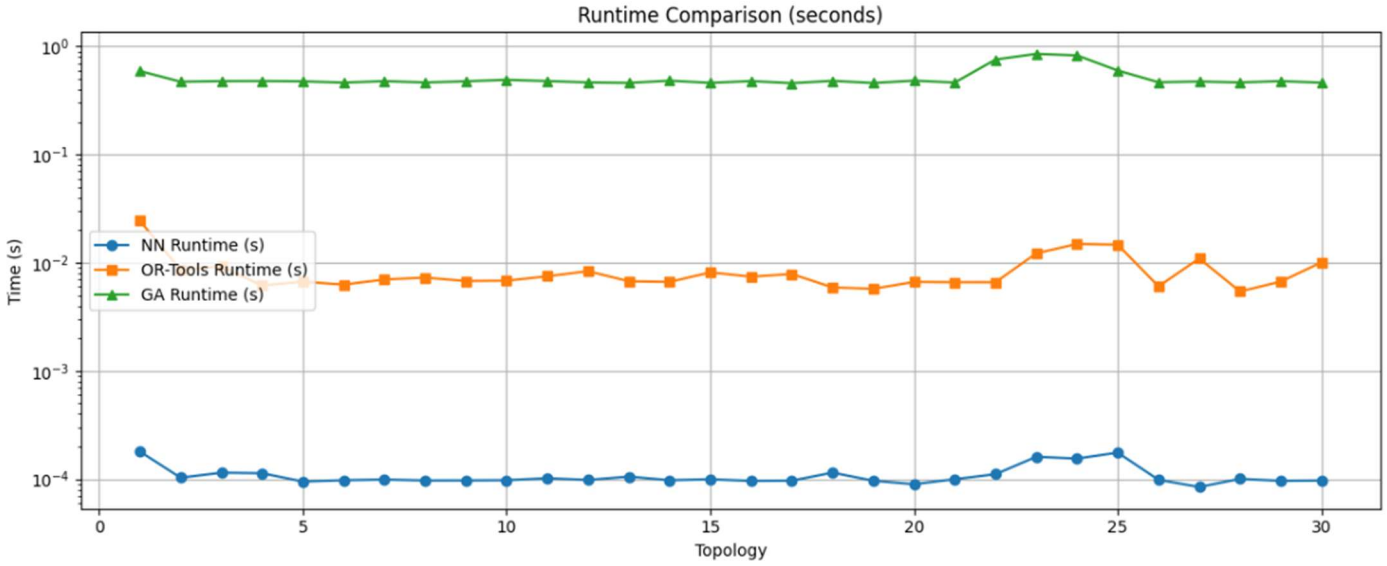
Genetic Algorith yapay zekâ temelli bir yöntemdir. Çözümler nesiller boyunca seçilip geliştirilerek zamanla daha iyi turlar elde edilmeye çalışılır. Bu sayede Nearest Neighbour'a göre genellikle daha kaliteli çoğu zaman da OR-Tools'a yakın sonuçlar üretir ancak bu iyileşme daha uzun çalışma süresi ve parametre ayarlarına duyarlılık gerektirir.

3. Sonuçlar

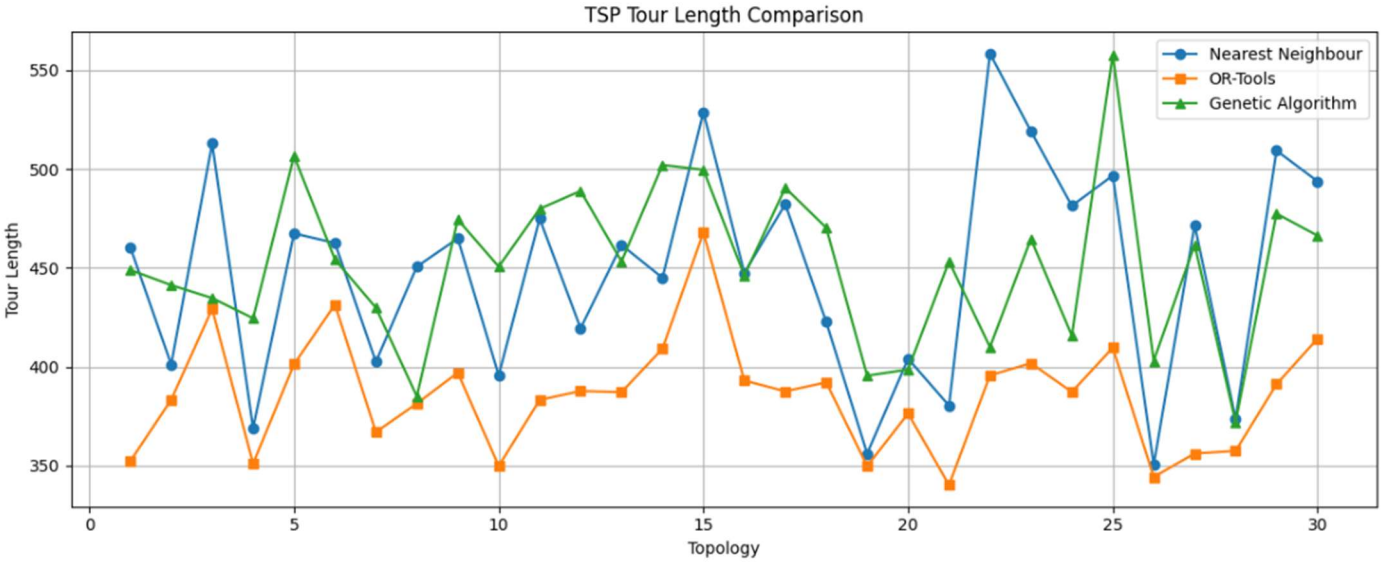
Bu çalışmada TSP problemini çözmek için kullanılan üç farklı yaklaşımın performanslarının birbirinden belirgin şekilde ayrıldığı elde edilmiştir. Nearest Neighbor algoritması çok hızlı sonuç üretmiş olsa da elde edilen tur uzunlukları çoğu durumda diğer yöntemlere kıyasla daha uzun kalmıştır. Bunun temel nedeni algoritmanın yalnızca bulunduğu noktaya en yakın seçeneği dikkate alması ve daha geniş çözüm uzayını göz ardı etmesi çıkarımı yapılmıştır.

Google OR-Tools daha gelişmiş arama mekanizmaları sayesinde en kısa ve en kaliteli turları üretmiş ve çözüm kalitesi açısından en başarılı yöntem olmuştur. Ancak bu diğer yöntemlere kıyasla daha uzun bir hesaplama süresi gerektirmiştir. Genetic Algorithm ise Nearest Neighbour ve OR-Tools arasında bir performans göstermiştir. Nearest Neighbour'a göre daha iyi sonuçlar üretirken birçok durumda OR-Tools'a yakın çözümler elde etmiş daha yüksek bir hesaplama maliyeti vermiştir.

4.) Plots



Grafikten yapılan çıkarımla çalışma süresi açısından Nearest Neighbour algoritması diğerlerine kıyasla en hızlı yöntemdir ve tüm topolojilerde neredeyse sabit çok düşük sürelerde çözüm üretmiştir. OR-Tools Nearest Neighbour'a göre daha yavaş olsa da çalışma süreleri düşük ve kararlıdır. Genetic Algorithm ise iteratif yapısından dolayı en uzun çalışma süresine sahip yöntem olduğu çıkarımı yapılmıştır.



Tur uzunluğu açısından Neraest Nighbour hızlı olmasına rağmen çoğu topolojide daha uzun ve dalgalı sonuçlar üretmiştir. OR-Tools neredeyse tüm örneklerde en kısa ya da ona çok yakın turlar bularak en istikrarlı ve kaliteli performansı göstermiştir. Genetic Algorithm ise genel olarak Nearest Neighbour'dan daha iyi OR-Tools'tan ise biraz daha uzun turlar üretmiş bazı topolojilerde OR-Tools'a oldukça yaklaşan sonuçlar elde etmiştir.

5.) Scripts

```
!pip install ortools

import math, random, time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp

def euclid(a, b):
    return math.hypot(a[0] - b[0], a[1] - b[1])

def tour_length(tour, coords):
    return sum(euclid(coords[tour[i]], coords[tour[(i+1) % len(tour)]])
               for i in range(len(tour)))

# 1.) NEAREST NEIGHBOUR
def nearest_neighbour(coords, start=0):
    n = len(coords)
    unvisited = set(range(n))
    tour = [start]
    unvisited.remove(start)

    while unvisited:
        last = tour[-1]
        nxt = min(unvisited, key=lambda v: euclid(coords[last], coords[v]))
        tour.append(nxt)
        unvisited.remove(nxt)

    return tour

# 2.) OR-TOOLS
def ortools_solve(coords, time_limit_s=1):
    n = len(coords)

    dist = {}
    for i in range(n):
        dist[i] = {}
        for j in range(n):
            dist[i][j] = int(euclid(coords[i], coords[j]) * 1000)

    manager = pywrapcp.RoutingIndexManager(n, 1, 0)
    routing = pywrapcp.RoutingModel(manager)

    def distance_callback(i, j):
        return dist[manager.IndexToNode(i)][manager.IndexToNode(j)]

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

    search_params = pywrapcp.DefaultRoutingSearchParameters()
    search_params.first_solution_strategy = routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
    search_params.time_limit.seconds = time_limit_s

    sol = routing.SolveWithParameters(search_params)
    if sol is None:
        return None, float("inf")

    index = routing.Start(0)
    route = []
    while not routing.IsEnd(index):
        route.append(manager.IndexToNode(index))
        index = sol.Value(routing.NextVar(index))

    return route, tour_length(route + [route[0]], coords)
```

```python
# 3.) GENETIC ALGORITHM
def genetic_tsp(coords, pop_size=100, generations=300, mutation_rate=0.2):
    n = len(coords)

    def create_individual():
        perm = list(range(1, n))
        random.shuffle(perm)
        return [0] + perm + [0]

    def crossover(p1, p2):
        a, b = sorted(random.sample(range(1, n), 2))
        child = [-1] * (n + 1)
        child[0] = child[-1] = 0
        child[a:b] = p1[a:b]
        fill = [x for x in p2[1:-1] if x not in child]
        idx = 0
        for i in range(1, n):
            if child[i] == -1:
                child[i] = fill[idx]
                idx += 1
        return child

    def mutate(ind):
        if random.random() < mutation_rate:
            i, j = random.sample(range(1, n), 2)
            ind[i], ind[j] = ind[j], ind[i]
        return ind

    population = [create_individual() for _ in range(pop_size)]

    for _ in range(generations):
        population = sorted(population, key=lambda x: tour_length(x, coords))
        elite = population[:int(0.2 * pop_size)]
        new_pop = elite.copy()

        while len(new_pop) < pop_size:
            p1, p2 = random.sample(elite, 2)
            child = crossover(p1, p2)
            new_pop.append(mutate(child))

        population = new_pop

    best = min(population, key=lambda x: tour_length(x, coords))
    return best, tour_length(best, coords)

NUM_TOPOLOGIES = 30
N_NODES = 20
BASE_SEED = 42

results = []

for i in range(NUM_TOPOLOGIES):

    random.seed(BASE_SEED + i)
    np.random.seed(BASE_SEED + i)

    coords = [(random.random() * 100, random.random() * 100)
              for _ in range(N_NODES)]

    # Nearest Neighbour
    t0 = time.perf_counter()
    nn_tour = nearest_neighbour(coords)
    nn_len = tour_length(nn_tour + [nn_tour[0]], coords)
    nn_time = time.perf_counter() - t0

    # OR-Tools
    t0 = time.perf_counter()
    ort_tour, ort_len = ortools_solve(coords)
    ort_time = time.perf_counter() - t0

    # Genetic Algorithm
    t0 = time.perf_counter()
    ga_tour, ga_len = genetic_tsp(coords)
    ga_time = time.perf_counter() - t0
```

```
    results.append({
        "topology": i + 1,
        "nn_length": nn_len,
        "nn_time": nn_time,
        "ortools_length": ort_len,
        "ortools_time": ort_time,
        "ga_length": ga_len,
        "ga_time": ga_time
    })

df = pd.DataFrame(results)

print(df.describe())

plt.figure(figsize=(12,5))
plt.plot(df["topology"], df["nn_length"], marker="o", label="Nearest Neighbour")
plt.plot(df["topology"], df["ortools_length"], marker="s", label="OR-Tools")
plt.plot(df["topology"], df["ga_length"], marker="^", label="Genetic Algorithm")
plt.title("TSP Tour Length Comparison")
plt.xlabel("Topology")
plt.ylabel("Tour Length")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12,5))
plt.plot(df["topology"], df["nn_time"], marker="o", label="NN Runtime (s)")
plt.plot(df["topology"], df["ortools_time"], marker="s", label="OR-Tools Runtime (s)")
plt.plot(df["topology"], df["ga_time"], marker="^", label="GA Runtime (s)")
plt.title("Runtime Comparison (seconds)")
plt.xlabel("Topology")
plt.ylabel("Time (s)")
plt.yscale("log")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

df.to_csv("tsp_results.csv", index=False)
```