Laporan Akhir Soft Computing

PERBANDINGAN ALGORITMA *ANT COLONY OPTIMIZATION* DALAM OPTIMASI RUTE DISTRIBUSI TRUK PENGANGKUT SAMPAH DI KABUPATEN SIDOARJO

Disusun Oleh :

Muhammad Ali Fikri 5026201019

Andira Yulianengtias 5026211038

Maulidiya Meilani 5026211121

DEPARTEMEN SISTEM INFORMASI

FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS

INSTITUT TEKNOLOGI SEPULUH NOPEMBER SURABAYA

2024

Daftar Pustaka

I. Latar Belakang

Sampah kota mencakup semua jenis sampah, baik organik maupun anorganik, yang dihasilkan dan dibuang oleh masyarakat di berbagai area perkotaan. Kota-kota besar di Indonesia, seperti Jakarta, Semarang, Surabaya, Bandung, Palembang, dan Medan, menghadapi masalah serius terkait manajemen sampah. Diperkirakan, setiap orang menghasilkan rata-rata 0,5 kg sampah per hari.

Kabupaten Sidoarjo, berlokasi di Provinsi Jawa Timur, memiliki luas wilayah 714,24 km^2 dengan kepadatan penduduk yang tinggi, mencapai 3.026,39 jiwa per km^2. Kabupaten ini menghadapi tantangan besar dalam mengelola sampah, yang dipengaruhi oleh pertumbuhan penduduk yang cepat, kurangnya kesadaran masyarakat dalam menjaga kebersihan, dan kinerja petugas Dinas Kebersihan yang belum optimal. Selain itu, kondisi geografis Kabupaten Sidoarjo yang terdiri dari banyak desa dengan jarak tempuh yang jauh menjadi hambatan dalam pengangkutan sampah, menyebabkan penumpukan sampah yang mengganggu.

Kegiatan pengelolaan sampah di Kabupaten Sidoarjo melibatkan pemilahan, pengumpulan, pengangkutan, pengolahan, dan pemrosesan akhir sampah. Ada dua metode utama pengangkutan sampah: Stationary Container System (SCS) yang menggunakan dump truck atau compactor truck, dan Hauled Container System (HCS) yang menggunakan armroll truck. Proses pengangkutan sampah memegang peranan penting dalam biaya pengelolaan sampah, termasuk bensin, upah supir, dan tenaga kerja.

Penyelesaian masalah pengangkutan sampah dapat dilihat sebagai Travelling Salesman Problem (TSP), sebuah masalah optimasi yang bertujuan menemukan rute pengangkutan yang paling efisien. TSP termasuk dalam kelas masalah non-polynomial hard (NP-hard) dan bisa diselesaikan menggunakan metode heuristik dan metaheuristik seperti algoritma genetika, simulated annealing, dan optimasi Ant Colony Optimization.

Pada *final project* ini bertujuan untuk membandingkan tiga algoritma optimasi Ant Colony Optimization dalam menemukan rute pengangkutan sampah yang efisien, yaitu Ant Colony System (ACS), Elitist, dan Max-Min. Hasilnya diharapkan dapat memberikan pemahaman lebih mendalam mengenai metode mana yang paling sesuai untuk diterapkan dalam konteks ini, sehingga pengelolaan sampah dapat dilakukan secara lebih efisien dan berkelanjutan.

II. Manfaat dan Tujuan

Tujuan dari project ini adalah untuk mencari rute distribusi pengangkutan sampah yang paling optimal dengan membandingka tiga variasi dari algoritma Ant Colony Optimization (ACO). Secara khusus, manfaat dan tujuan dari project ini adalah :

1. Mengoptimalkan Rute Pengangkutan Sampah
   - Menerapkan tiga variasi algoritma Ant Colony Optimization (ACO), yaitu Ant Colony System (ACS), Elitist, dan Max-Min untuk mendapatkan rute pengangkutan sampah yang terpendek di Kabupaten Sidoarjo.
   - Membandingkan kinerja ketiga algoritma tersebut dalam hal pengelolaan rute yang efektif dan efisian.
2. Visualisasi Grafik Hasil Rute
   - Membuat visualisasi hasil rute pengangkutan sampah ke dalam grafik matplotlib untuk mempermudah pemahaman dalam perhitungan rutenya.
3. Memberikan Rekomendasi Implementasi
   - Memberikan rekomendasi praktis untuk implementasi hasil penelitian dalam operasional sehari-hari Dinas Lingkungan Hidup dan Kebersihan Kabupaten Sidoarjo
4. Meminimalkan Biaya Operasional Distribusi Pengangkutan Sampah
   - Mengurangi atau meminimalkan biaya operasional Dinas Lingkungan Hidup dan Kebersihan terkait pengangkutan sampah dalam wilayah yang luas seperti Kabupaten Sidoarjo.

III. Tinjauan Literatur

Studi yang diberi judul "Desain Sistem Optimasi Rute Distribusi Pengangkutan Sampah di Kabupaten Sidoarjo Menggunakan Algoritma Ant Colony Optimization (ACO)" oleh Kirana Gita Larasati menggambarkan secara detail masalah pengelolaan sampah di Kabupaten Sidoarjo. Dalam studi ini, ditemukan bahwa kesadaran masyarakat yang kurang dan kinerja Dinas Lingkungan Hidup dan Kebersihan yang tidak optimal menjadi penyebab utama masalah sampah di daerah tersebut.

Kabupaten Sidoarjo yang memiliki wilayah geografis yang luas memerlukan pengangkutan sampah yang harus menempuh jarak yang cukup jauh ke Tempat Pembuangan Akhir (TPA) di Jabon. Untuk mengatasi masalah ini, penelitian ini bertujuan untuk mengembangkan sistem optimasi rute distribusi pengangkutan sampah dengan memanfaatkan Algoritma Ant Colony Optimization (ACO). Pendekatan ini diharapkan dapat membantu Dinas Kebersihan dalam menentukan rute yang paling efisien yang melibatkan Tempat Pembuangan Sementara (TPS) di setiap kecamatan.

Solusi yang ditawarkan dalam penelitian ini menggunakan algoritma ACO untuk mencari rute terbaik. Selain itu, sebuah aplikasi berbasis web juga diajukan untuk memberikan visualisasi dari rute yang direkomendasikan, menunjukkan pendekatan yang mudah digunakan dan praktis.

Poin-poin kunci dalam penelitian:

- Identifikasi Masalah: Studi ini dengan jelas mengidentifikasi bahwa kurangnya kesadaran masyarakat dan kinerja Dinas Lingkungan Hidup dan Kebersihan adalah akar masalah sampah di Kabupaten Sidoarjo.
- Tujuan Studi: Penelitian ini bertujuan untuk mengembangkan sistem optimasi rute distribusi pengangkutan sampah dengan ACO, dengan fokus pada penjangkauan semua titik TPS di setiap kecamatan.
- Metode yang Digunakan: ACO dipilih sebagai algoritma kunci untuk mencari rute terbaik, dengan pengembangan aplikasi web menggunakan HTML5, CSS3, Javascript, dan PHP.

IV. Deskripsi Permasalahan

Adapun beberapa permasalahan yang ingin diselesaikan melalui pemanfaatan project ini dalam pengelolaan sampah di Kabupaten Sidoarjo, meliputi :

- Penumpukan Sampah : Terdapat tumpukan sampah di beberapa wilayah Sidoarjo. Hal ini seperti yang terjadi di belakang terminal Purabaya, dikutip dari JawaPos.com, 2024, terdapat gunungan sampah yang diperkirakan mencapai 160 sentimeter dan Panjang sekitar 30 meter. Fenomena ini jika tidak segera diatasi maka akan mengakibatkan dampat negatif terhadap lingkungan sekitar dan kesehatan masyarakat.

- Biaya Pengangkutan Sampah yang Tinggi : Biaya pengankutan sampah di Sidoarjo menjadi masalah yang signifikan, hal ini dinilai sangat memberatkan bagi warga Sidoarjo, tukang angkut sampah hingga para pengelolah TPST yang mengakibatkan pengelola TPST terpaksa menurunkan honor tukang angkut sampah demi menutup biaya operasional keseharian (Karim, 2023).
- Tantangan Rute Pengangkutan : Penentuan rute pengangkutan sampah yang optimal masih menjadi tantangan DLHK Kabupaten Sidoarjo, hal ini dikarenakan truk pengangkut sampah yang jumlahnya masih sangat terbatas (Arista, 2024), sehingga dibutuhkan rute pengangkutan sampah yang paling optimal atau tercepat.

V. Penjelasan Algoritma

Dalam final project ini, kami menggunakan algoritma Ant Colony Optimization (ACO) dan membandingkan 3 model atau variasi dari ACO yaitu Ant Colony System (ACS), Elitist, dan Max-Min.

A. Ant Colony System (ACS)

Algoritma Ant Colony System (ACS) dikembangkan oleh Dorigo, Maniezzo, dan Colorni pada tahun 1996 untuk menyelesaikan Traveling Salesman Problem (TSP). AS melibatkan sejumlah semut yang bekerja bersama melalui komunikasi pheromone. Setiap semut memulai tur mereka dari titik acak dan bergerak ke titik berikutnya berdasarkan probabilitas yang ditentukan oleh jarak dan konsentrasi pheromone. Semut memanfaatkan tabulist untuk menghindari mengunjungi titik yang sudah dikunjungi sebelumnya, mendekati solusi optimal. Setelah semua semut menyelesaikan tur mereka, pheromone diperbarui berdasarkan panjang tur. Pheromone baru ditinggalkan pada edge-edge dari tur terpendek, sementara pheromone lama menguap untuk mencegah stagnasi. Proses ini diulangi hingga solusi optimal ditemukan atau sistem menunjukkan tanda-tanda stagnasi.

B. Elitist

Pengembangan awal dari ACS mengarah pada strategi elitist yang dikenal sebagai Elitist Ant System (EAS), yang diusulkan oleh Dorigo, Maniezzo, dan Colorni dalam beberapa publikasi tahun 1991a, 1991b, dan 1996. Konsep dasar dari EAS adalah memberikan penguatan pheromone pada edge-edge yang membentuk tour terbaik yang ditemukan pada awal pelaksanaan algoritma. Tour terbaik ini sering disebut sebagai

TBS (tour terbaik yang pernah ditemukan). Dengan menerapkan strategi ini, EAS berupaya untuk mempertahankan informasi tentang solusi terbaik yang telah ditemukan sejauh ini, sehingga memberikan dorongan lebih kuat kepada semut untuk mengikuti jejak solusi yang telah terbukti efektif. Ini membantu meningkatkan kemungkinan pencapaian solusi optimal dalam iterasi berikutnya.

C. Max-Min

MAX-MIN Ant System (MMAS) adalah modifikasi dari algoritma ACS dengan beberapa perubahan kunci:

1. Penambahan Pheromone: MMAS memungkinkan penambahan pheromone pada edge-edge dari tour terbaik awal (best-so-far-tour) atau tour terbaik saat iterasi berjalan (iteration best-tour). Ini bisa menyebabkan stagnasi, namun.
2. Batasan Nilai Pheromone: Untuk menghindari stagnasi, MMAS menerapkan batasan pada nilai pheromone dalam interval [τ_min, τ_max].
3. Inisialisasi Pheromone: MMAS menginisialisasi nilai pheromone dengan batas atas, dengan evaporasi rendah untuk memulai eksplorasi lebih awal.
4. Penginisialisasian Ulang Pheromone: Dalam kasus stagnasi atau ketidaksesuaian iterasi, nilai pheromone direset untuk mendorong eksplorasi ulang.

VI. Penerapan Algoritma

Berikut adalah data yang akan digunakan dalam penerapan algoritma Ant Colony Optimization:

No	Nama TPS	Lat	Long
1	Depot dummy	-7.428	112.71
2	TPS Sidokerto Buduran	-7.427615	112.711109
3	TPS Sukorejo Buduran	-7.42004	112.717253
4	TPST Siwalanpanji Buduran	-7.4325987	112.7254143
5	TPS SDN Siwalanpanji Buduran	-7.432267	112.728402
6	SMA Antartika Buduran	-7.431704	112.726543
7	TPST PT Avian Buduran	-7.418909	112.725523
8	SMP PGRI 1 Buduran	-7.431318	112.724925
9	TPS Pondok Al-Ghozini Buduran	-7.4281	112.724118
10	TPS Perum Surya Asri (Dapurno) Buduran	-7.417853	112.757257

11	Jl. KH Khamdani Buduran	-7.428407	112.725056
12	TPS Perum Candiloka Candi	-7.494257	112.719477
13	TPS Candi Asri Candi	-7.493967	112.7168725
14	TPS Perum CSM Candi	-7.460478	112.687796
15	Terminal Larangan Candi	-7.466627	112.71302
16	TPS Tenggulunan Candi	-7.46971	112.71078
17	TPS Perum AL Candi	-7.477639	112.702337
18	TPS Ps. Krempyeng Sugihwaras Candi	-7.477367	112.701596
19	TPS Jl. Raya Bligo Candi	-7.475349	112.71509
20	TPS Kir Candi Candi	-7.4826914	112.7105095
21	TPS Jl Karang Tanjung Candi	-7.487192	112.690701
22	TPS Desa Ketajen Gedangan	-7.387379	112.732604
23	TPS Aloha Gedangan	-7.373912	112.728558
24	TPS Sungai Seruni Gedangan	-7.399051	112.724847
25	TPS Makro Gedangan	-7.367755	112.727134
26	TPS Hair Star Gedangan	-7.387904	112.744354
27	TPS Indomart Gedangan	-7.399895	112.727094
28	TPS Restoran Aloha Gedangan	-7.3742506	112.7282426
29	TPS Karangbong Gedangan	-7.403149	112.714927
30	TPST Kebonsikep Gedangan	-7.390629	112.723074
31	TPS Marinir Juanda Gedangan	-7.3798339	112.7390049
32	TPS Alloy Stell Gedangan	-7.3839035	112.7242795
33	TPS Sawotratap Gedangan	-7.367898	112.72927
34	TPS Gudang Garam Gedangan	-7.3543774	112.7210301
35	TPS Komplek AL Tebel Gedangan	-7.4042878	112.7248046
36	TPS SMA Negeri Gedangan	-7.387845	112.741398
37	TPS Perum Putri Juanda Waru	-7.3999753	112.7623732
38	TPS Taman Aloha Waru	-7.367158	112.704671
39	TPS Jl Raya Krian	-7.406027	112.585173
40	TPS Perum Citra Diamond Krian	-7.4462837	112.5687863
41	TPS Kemangsen Krian	-7.411169	112.569028
42	Sedati - TPS Angkasapura	-7.373831	112.800322
43	Sedati - TPS Bandara Juanda	-7.371533	112.780227
44	Sedati - TPS Jl. Raya Juanda	-7.380967	112.749007
45	Sedati - TPS Pasar Wisata Juanda	-7.373621	112.763504
46	Sedati - Miskan TPS Bea Cukai Juanda	-7.381391	112.758408
47	Sedati - TPS Jl. Raya Tropodo	-7.359124	112.764548
48	Sedati - Choirul Anan TPS Lesen	-7.390902	112.759091
49	Sedati - TPS AURI	-7.379465	112.750342
50	TPA Jabon	-7.548852908	112.7635909

a. Algoritma Ant Colony System

```
import numpy as np
import matplotlib.pyplot as plt
import time
```

```
class AntColony:
    def __init__(self, num_ants, num_iterations, pheromone_evaporation_rate,
alpha, beta, tps_names, coordinates):
        self.num_ants = num_ants
        self.num_iterations = num_iterations
        self.pheromone_evaporation_rate = pheromone_evaporation_rate
        self.alpha = alpha
        self.beta = beta
        self.tps_names = tps_names
        self.coordinates = coordinates
        self.num_nodes = len(coordinates)
        self.distances = self.calculate_distances()
        self.pheromone_matrix = np.ones((self.num_nodes, self.num_nodes))
        np.fill_diagonal(self.pheromone_matrix, 0)
        self.best_distance = float('inf')
        self.best_route = []

    def calculate_distances(self):
        distances = np.zeros((self.num_nodes, self.num_nodes))
        for i in range(self.num_nodes):
            for j in range(self.num_nodes):
                distances[i][j]    =    np.linalg.norm(self.coordinates[i]    -
self.coordinates[j])
        return distances

    def run(self):
        start_time = time.time()
        for iteration in range(self.num_iterations):
            ant_routes = []
            for ant in range(self.num_ants):
                route = self.generate_ant_route()
                ant_routes.append((route,
self.calculate_route_distance(route)))

            self.update_pheromone(ant_routes)

            if iteration == 0:
                self.plot_route(iteration, ant_routes)
            elif iteration == self.num_iterations - 1:
                self.plot_route(iteration, ant_routes)

            if iteration % 10 == 0:
                print("Iteration:",    iteration,    "|    Best    Distance:",
self.best_distance)

        print("Best Distance:", self.best_distance)
        print("Best Route:", self.best_route)
```

```
        end_time = time.time()
        print("Total Runtime:", end_time - start_time, "seconds")

    def plot_route(self, iteration, ant_routes):
        best_ant_route, _ = min(ant_routes, key=lambda x: x[1])
        best_route_coords = [self.coordinates[node] for node in best_ant_route]
        best_route_coords.append(self.coordinates[best_ant_route[0]])

        plt.figure(figsize=(8, 6))
        plt.scatter(self.coordinates[:, 1], self.coordinates[:, 0], c='blue',
label='TPS Locations')
        plt.plot([coord[1] for coord in best_route_coords], [coord[0] for coord
in best_route_coords],
                 c='red', linewidth=1.5, linestyle='-', label='Best Route')
        plt.title(f"Iteration {iteration}")
        plt.xlabel("Longitude")
        plt.ylabel("Latitude")
        plt.legend()
        plt.grid(True)
        plt.show()

    def generate_ant_route(self):
        start_node = 0  # Start from depot dummy
        unvisited_nodes = set(range(self.num_nodes))
        unvisited_nodes.remove(start_node)
        current_node = start_node
        route = [start_node]

        while unvisited_nodes:
            probabilities = self.calculate_probabilities(current_node,
unvisited_nodes)
            next_node = np.random.choice(list(unvisited_nodes),
p=probabilities)
            route.append(next_node)
            unvisited_nodes.remove(next_node)
            current_node = next_node

        route.append(start_node)
        return route

    def calculate_probabilities(self, current_node, unvisited_nodes):
        pheromone_values = np.array([self.pheromone_matrix[current_node][i]
for i in unvisited_nodes])
        distances = np.array([self.distances[current_node][i] for i in
unvisited_nodes])
        heuristic_values = 1 / (distances + 1e-10)  # Add a small value to avoid
division by zero
```

```
        probabilities = (pheromone_values ** self.alpha) * (heuristic_values **
self.beta)
        probabilities /= np.sum(probabilities)
        return probabilities

    def calculate_route_distance(self, route):
        distance = 0
        for i in range(len(route) - 1):
            distance += self.distances[route[i]][route[i + 1]]
        return distance

    def update_pheromone(self, ant_routes):
        for i in range(self.num_nodes):
            for j in range(self.num_nodes):
                if i != j:
                    self.pheromone_matrix[i][j]         *=         (1         -
self.pheromone_evaporation_rate)
                    for route, distance in ant_routes:
                        if (i in route) and (j in route):
                            self.pheromone_matrix[i][j] += 1 / distance

        best_ant_route, best_ant_distance = min(ant_routes, key=lambda x: x[1])
        if best_ant_distance < self.best_distance:
            self.best_distance = best_ant_distance
            self.best_route = [self.tps_names[node] for node in best_ant_route]

if __name__ == "__main__":
    tps_names = [
        "Depot dummy", "TPS Sidokerto Buduran", "TPS Sukorejo Buduran", "TPST
Siwalanpanji Buduran",
        "TPS SDN Siwalanpanji Buduran", "SMA Antartika Buduran", "TPST PT Avian
Buduran", "SMP PGRI 1 Buduran",
        "TPS Pondok Al-Ghozini Buduran", "TPS Perum Surya Asri (Dapurno)
Buduran", "Jl. KH Khamdani Buduran",
        "TPS Perum Candiloka Candi", "TPS Candi Asri Candi", "TPS Perum CSM
Candi", "Terminal Larangan Candi",
        "TPS Tenggulunan Candi", "TPS Perum AL Candi", "TPS Ps. Krempyeng
Sugihwaras Candi", "TPS Jl. Raya Bligo Candi",
        "TPS Kir Candi Candi", "TPS Jl Karang Tanjung Candi", "TPS Desa Ketajen
Gedangan", "TPS Aloha Gedangan",
        "TPS Sungai Seruni Gedangan", "TPS Makro Gedangan", "TPS Hair Star
Gedangan", "TPS Indomart Gedangan",
        "TPS Restoran Aloha Gedangan", "TPS Karangbong Gedangan", "TPST
Kebonsikep Gedangan", "TPS Marinir Juanda Gedangan",
        "TPS Alloy Stell Gedangan", "TPS Sawotratap Gedangan", "TPS Gudang Garam
Gedangan", "TPS Komplek AL Tebel Gedangan",
```

```
        "TPS SMA Negeri Gedangan", "TPS Perum Putri Juanda Waru", "TPS Taman Aloha Waru", "TPS Jl Raya Krian",
        "TPS Perum Citra Diamond Krian", "TPS Kemangsen Krian", "Sedati - TPS Angkasapura", "Sedati - TPS Bandara Juanda",
        "Sedati - TPS Jl. Raya Juanda", "Sedati - TPS Pasar Wisata Juanda", "Sedati - Miskan TPS Bea Cukai Juanda",
        "Sedati - TPS Jl. Raya Tropodo", "Sedati - Choirul Anan TPS Lesen", "Sedati - TPS AURI"
    ]

    coordinates = np.array([
        [-7.428, 112.710], [-7.427615, 112.711109], [-7.42004, 112.717253], [-7.4325987, 112.7254143],
        [-7.432267,   112.728402],   [-7.431704,   112.726543],   [-7.418909, 112.725523], [-7.431318, 112.724925],
        [-7.4281,    112.724118],    [-7.417853,    112.757257],    [-7.428407, 112.725056], [-7.494257, 112.719477],
        [-7.493967,   112.7168725],   [-7.460478,   112.687796],   [-7.466627, 112.71302], [-7.46971, 112.71078],
        [-7.477639,   112.702337],   [-7.477367,   112.701596],   [-7.475349, 112.71509], [-7.4826914, 112.7105095],
        [-7.487192,   112.690701],   [-7.387379,   112.732604],   [-7.373912, 112.728558], [-7.399051, 112.724847],
        [-7.367755,   112.727134],   [-7.387904,   112.744354],   [-7.399895, 112.727094], [-7.3742506, 112.7282426],
        [-7.403149,   112.714927],   [-7.390629,   112.723074],   [-7.3798339, 112.7390049], [-7.3839035, 112.7242795],
        [-7.367898,   112.72927],   [-7.3543774,   112.7210301],   [-7.4042878, 112.7248046], [-7.387845, 112.741398],
        [-7.3999753,   112.7623732],   [-7.367158,   112.704671],   [-7.406027, 112.585173], [-7.4462837, 112.5687863],
        [-7.411169,   112.569028],   [-7.373831,   112.800322],   [-7.371533, 112.780227], [-7.380967, 112.749007],
        [-7.373621,   112.763504],   [-7.381391,   112.758408],   [-7.359124, 112.764548], [-7.390902, 112.759091],
        [-7.379465, 112.750342]
    ])

    num_ants = 80
    num_iterations = 100
    pheromone_evaporation_rate = 0.05
    alpha = 1
    beta = 2

    ant_colony          =          AntColony(num_ants,          num_iterations, pheromone_evaporation_rate, alpha, beta, tps_names, coordinates)
    ant_colony.run()
```

1. **init__()**: Ini adalah konstruktor kelas AntColony yang menginisialisasi atribut-atribut objek, seperti jumlah semut (num_ants), jumlah iterasi (num_iterations), laju evaporasi feromon (pheromone_evaporation_rate), alpha, beta, nama-nama TPS (tps_names), koordinat TPS (coordinates), jumlah simpul (num_nodes), matriks jarak antara TPS (distances), dan matriks feromon (pheromone_matrix).

2. **calculate_distances()**: Fungsi ini menghitung jarak antara setiap pasangan titik TPS berdasarkan koordinat mereka menggunakan norma Euclidean.

3. **run()**: Fungsi ini adalah "jantung" dari algoritma ACO. Ini mengeksekusi serangkaian iterasi, di mana setiap iterasi terdiri dari beberapa langkah:

    a. Setiap semut membangkitkan rute baru (generate_ant_route()) dan menghitung panjang rute tersebut.

    b. Pheromon diperbarui (update_pheromone()) berdasarkan rute-rute semut.

    c. Pada iterasi pertama (iteration == 0) dan iterasi terakhir (iteration == self.num_iterations - 1), fungsi plot_route() dipanggil untuk menampilkan grafik rute terbaik pada saat itu.

    d. Pada setiap iterasi yang kelipatan 10, dilakukan pencetakan ke konsol untuk memantau perkembangan.

4. **plot_route():** Fungsi ini digunakan untuk menggambar grafik rute terbaik pada suatu iterasi. Ini memetakan lokasi TPS sebagai titik biru dan rute terbaik sebagai garis merah.

5. **generate_ant_route():** Fungsi ini digunakan oleh setiap semut untuk menghasilkan rute baru. Semut memilih node berikutnya untuk dikunjungi berdasarkan probabilitas yang dihitung berdasarkan feromon dan informasi heuristik.

6. **calculate_probabilities():** Fungsi ini menghitung probabilitas untuk memilih node selanjutnya berdasarkan feromon dan informasi heuristik.

7. **calculate_route_distance():** Fungsi ini menghitung panjang total suatu rute berdasarkan jarak antara node-node yang dikunjungi.

8. **update_pheromone():** Fungsi ini memperbarui matriks feromon berdasarkan rute-rute semut. Peningkatan feromon terjadi untuk setiap pasangan node yang dikunjungi oleh semut, dan jika semut menemukan rute yang lebih baik dari sebelumnya, maka nilai panjang rute tersebut digunakan untuk peningkatan feromon.

9. **num_ants:** Jumlah semut dalam koloni semut. Semakin banyak semut, semakin banyak solusi yang dieksplorasi secara bersamaan, tetapi juga meningkatkan waktu komputasi. Nilai ini mempengaruhi seberapa banyak rute yang dieksplorasi secara bersamaan pada setiap iterasi.

10. **num_iterations:** Jumlah iterasi atau siklus yang dilakukan oleh algoritma. Setiap iterasi biasanya mencakup seluruh populasi semut. Jumlah iterasi ini mempengaruhi berapa lama algoritma berjalan dan seberapa baik konvergensi solusi yang dicapai.
11. **pheromone_evaporation_rate:** Tingkat evaporasi feromon. Ini menentukan seberapa cepat feromon menguap dari jalur yang dilewati oleh semut. Tingkat yang lebih tinggi akan menyebabkan feromon menguap lebih cepat, yang mengarah pada eksplorasi yang lebih cepat ke berbagai rute. Namun, jika terlalu tinggi, ini dapat mengurangi kemungkinan konvergensi ke solusi yang baik.
12. **alpha:** Parameter yang mengatur pengaruh feromon pada pemilihan jalur oleh semut. Nilai alpha yang lebih tinggi menekankan lebih pada feromon daripada pada informasi heuristik (jarak antara dua titik). Nilai ini mempengaruhi seberapa besar semut mempertimbangkan feromon dalam pengambilan keputusan.
13. **beta:** Parameter yang mengatur pengaruh informasi heuristik (jarak antara dua titik) pada pemilihan jalur oleh semut. Nilai beta yang lebih tinggi menekankan lebih pada informasi heuristik daripada pada feromon. Nilai ini mempengaruhi seberapa besar semut mempertimbangkan jarak antara dua titik dalam pengambilan keputusan. Semakin besar nilai beta, semakin besar pengaruh heuristik, yang berarti semut cenderung memilih rute yang memiliki jarak atau biaya yang lebih kecil

b. Algoritma Elitist

```
.................................
def update_pheromone(self, ant_routes):
        # Evaporate pheromone
        self.pheromone_matrix *= (1 - self.pheromone_evaporation_rate)

        # Find best ant route
        best_ant_route, best_ant_distance = min(ant_routes, key=lambda x: x[1])
```

```
        # Update pheromone on the best route only
        for i in range(len(best_ant_route) - 1):
            current_node, next_node = best_ant_route[i], best_ant_route[i + 1]
            self.pheromone_matrix[current_node][next_node]     +=     1     /
best_ant_distance

        # Update best distance and route if a better solution is found
        if best_ant_distance < self.best_distance:
            self.best_distance = best_ant_distance
            self.best_route = [self.tps_names[node] for node in best_ant_route]
.........................
```

Fungsi **update_pheromone** dalam algoritma ACO elitist bertanggung jawab untuk memperbarui matriks feromon berdasarkan hasil rute terbaik yang ditemukan oleh koloni semut pada iterasi tertentu. Berikut adalah penjelasan langkah-langkahnya:

1. **Evaporasi Feromon:** Langkah pertama adalah mengurangkan nilai feromon pada semua jalur yang ada dalam matriks feromon. Hal ini dilakukan dengan mengalikan setiap elemen matriks feromon dengan nilai (1 - tingkat evaporasi feromon). Dengan demikian, feromon pada setiap jalur secara proporsional menguap seiring waktu, mengurangi pengaruhnya terhadap pemilihan rute di iterasi selanjutnya.
2. **Temukan Rute Terbaik:** Selanjutnya, dari semua rute yang telah ditempuh oleh semua semut, dicari rute terbaik yang memiliki jarak tempuh terpendek. Hal ini dilakukan dengan menggunakan fungsi min dengan parameter key=lambda x: x[1], yang mengambil rute dengan jarak terpendek sebagai rute terbaik.
3. **Update Feromon pada Rute Terbaik:** Setelah rute terbaik ditemukan, feromon pada setiap jalur dalam rute tersebut diperbarui. Untuk setiap pasangan node dalam rute terbaik, feromon pada jalur tersebut ditambah dengan nilai 1 dibagi oleh jarak rute terbaik. Dengan demikian, jalur-jalur dalam rute terbaik yang lebih pendek akan menerima peningkatan feromon yang lebih besar.
4. **Update Solusi Terbaik**: Terakhir, jika rute terbaik yang ditemukan pada iterasi saat ini memiliki jarak yang lebih pendek dari solusi terbaik yang telah ada sebelumnya, maka solusi terbaik dan rutenya diperbarui. Ini dilakukan dengan memperbarui self.best_distance dengan nilai jarak rute terbaik yang baru ditemukan, dan self.best_route dengan node-node yang membentuk rute terbaik tersebut.

Perbedaan:

1. Penanganan Pembaruan Feromon:
    a. **ACO Elitist:** Pada ACO elitist, hanya rute terbaik yang ditemukan oleh koloni semut pada iterasi tertentu yang digunakan untuk memperbarui feromon. Feromon hanya diperbarui pada jalur-jalur yang ada dalam rute terbaik.
    b. **ACS (Ant Colony System):** Pada ACS, semua semut yang berpartisipasi dalam pencarian solusi memperbarui feromon berdasarkan rute yang mereka tempuh. Ini dapat menyebabkan lebih banyak variasi dalam pembaruan feromon.
2. Penentuan Solusi Terbaik:
    a. **ACO Elitist:** Solusi terbaik didefinisikan sebagai rute terpendek yang pernah ditemukan oleh koloni semut sepanjang iterasi-algoritma berjalan. Solusi ini diperbarui hanya jika rute terbaik yang ditemukan pada iterasi saat ini lebih pendek dari solusi terbaik sebelumnya.
    b. **ACS (Ant Colony System):** Solusi terbaik didefinisikan sebagai rute terpendek yang pernah ditemukan oleh koloni semut sepanjang keseluruhan iterasi-algoritma berjalan. Dalam ACS, biasanya menggunakan mekanisme tour-memory untuk memilih solusi terbaik.

c. Algoritma Max-Min

```
import numpy as np
import matplotlib.pyplot as plt
import time

class AntColony:
    def __init__(self, num_ants, num_iterations, pheromone_evaporation_rate,
alpha, beta, tps_names, coordinates, min_pheromone, max_pheromone):
        self.num_ants = num_ants
        self.num_iterations = num_iterations
        self.pheromone_evaporation_rate = pheromone_evaporation_rate
        self.alpha = alpha
        self.beta = beta
        self.tps_names = tps_names
        self.coordinates = coordinates
        self.num_nodes = len(coordinates)
        self.distances = self.calculate_distances()
        self.pheromone_matrix = np.ones((self.num_nodes, self.num_nodes))
        np.fill_diagonal(self.pheromone_matrix, 0)
        self.best_distance = float('inf')
        self.best_route = []
        self.min_pheromone = min_pheromone
        self.max_pheromone = max_pheromone
```

```
    def calculate_distances(self):
        distances = np.zeros((self.num_nodes, self.num_nodes))
        for i in range(self.num_nodes):
            for j in range(self.num_nodes):
                distances[i][j]    =    np.linalg.norm(self.coordinates[i]    -
self.coordinates[j])
        return distances

    def run(self):
        start_time = time.time()
        for iteration in range(self.num_iterations):
            ant_routes = []
            for ant in range(self.num_ants):
                route = self.generate_ant_route()
                ant_routes.append((route,
self.calculate_route_distance(route)))

            self.update_pheromone(ant_routes,              self.min_pheromone,
self.max_pheromone)

            if iteration == 0 or iteration == self.num_iterations - 1:
                self.plot_route(iteration, ant_routes)

            if iteration % 10 == 0:
                print("Iteration:",    iteration,    "|    Best    Distance:",
self.best_distance)

        print("Best Distance:", self.best_distance)
        print("Best Route:", self.best_route)
        end_time = time.time()
        print("Total Runtime:", end_time - start_time, "seconds")

    def plot_route(self, iteration, ant_routes):
        best_ant_route, _ = min(ant_routes, key=lambda x: x[1])
        best_route_coords = [self.coordinates[node] for node in best_ant_route]
        best_route_coords.append(self.coordinates[best_ant_route[0]])

        plt.figure(figsize=(8, 6))
        plt.scatter(self.coordinates[:, 1], self.coordinates[:, 0], c='blue',
label='TPS Locations')
        plt.plot([coord[1] for coord in best_route_coords], [coord[0] for coord
in best_route_coords],
                 c='red', linewidth=1.5, linestyle='-', label='Best Route')
        plt.title(f"Iteration {iteration}")
        plt.xlabel("Longitude")
```

```
        plt.ylabel("Latitude")
        plt.legend()
        plt.grid(True)
        plt.show()

    def generate_ant_route(self):
        start_node = 0  # Start from depot dummy
        unvisited_nodes = set(range(self.num_nodes))
        unvisited_nodes.remove(start_node)
        current_node = start_node
        route = [start_node]

        while unvisited_nodes:
            probabilities     =     self.calculate_probabilities(current_node,
unvisited_nodes)
            next_node         =         np.random.choice(list(unvisited_nodes),
p=probabilities)
            route.append(next_node)
            unvisited_nodes.remove(next_node)
            current_node = next_node

        route.append(start_node)
        return route

    def calculate_probabilities(self, current_node, unvisited_nodes):
        pheromone_values   =   np.array([self.pheromone_matrix[current_node][i]
for i in unvisited_nodes])
        distances   =   np.array([self.distances[current_node][i]   for   i   in
unvisited_nodes])
        heuristic_values = 1 / (distances + 1e-10)  # Add a small value to avoid
division by zero
        probabilities = (pheromone_values ** self.alpha) * (heuristic_values **
self.beta)
        probabilities /= np.sum(probabilities)
        return probabilities

    def calculate_route_distance(self, route):
        distance = 0
        for i in range(len(route) - 1):
            distance += self.distances[route[i]][route[i + 1]]
        return distance

    def update_pheromone(self, ant_routes, min_pheromone, max_pheromone):
        # Evaporate pheromone
        self.pheromone_matrix *= (1 - self.pheromone_evaporation_rate)

        # Find best ant route
```

```
        best_ant_route, best_ant_distance = min(ant_routes, key=lambda x: x[1])

        # Update pheromone on the best route only
        for i in range(len(best_ant_route) - 1):
            current_node, next_node = best_ant_route[i], best_ant_route[i + 1]
            self.pheromone_matrix[current_node][next_node]     +=     1     /
best_ant_distance

        # Limit pheromone values to min_pheromone and max_pheromone
        self.pheromone_matrix[self.pheromone_matrix    <    min_pheromone]    =
min_pheromone
        self.pheromone_matrix[self.pheromone_matrix    >    max_pheromone]    =
max_pheromone

        # Update best distance and route if a better solution is found
        if best_ant_distance < self.best_distance:
            self.best_distance = best_ant_distance
            self.best_route = [self.tps_names[node] for node in best_ant_route
if node < len(self.tps_names)]

.....................................

    min_pheromone = 0.1
    max_pheromone = 10

    colony = AntColony(num_ants, num_iterations, pheromone_evaporation_rate,
alpha, beta, tps_names, coordinates, min_pheromone, max_pheromone)
    colony.run()
```

Kegunaan dari parameter-parameter ini adalah sebagai berikut:

1. **min_pheromone:** Parameter ini menentukan nilai minimum yang diizinkan untuk pheromone pada setiap jalur dalam matriks pheromone. Dengan mengatur nilai minimum ini, Anda dapat memastikan bahwa pheromone tidak akan menurun di bawah ambang tertentu selama proses evolusi algoritma. Hal ini membantu mencegah stagnasi dan memungkinkan menjaga keragaman dalam pencarian solusi.

2. **max_pheromone:** Parameter ini menentukan nilai maksimum yang diizinkan untuk pheromone pada setiap jalur dalam matriks pheromone. Dengan mengatur nilai maksimum ini, Anda dapat mencegah pheromone menjadi terlalu dominan, yang dapat mengarah pada konvergensi terlalu cepat ke solusi lokal dan mengurangi kemungkinan eksplorasi solusi alternatif yang lebih baik.

Algoritma MAX-MIN ant colony optimization (ACO) adalah variasi dari algoritma ACO yang menggunakan aturan peningkatan dan penguapan pheromone yang lebih ketat untuk meningkatkan konvergensi dan mencegah stagnasi dalam penyebaran pheromone. Perbedaan antara algoritma MAX-MIN ACO dan algoritma ACO biasa, seperti ACS (Ant Colony System) dan elitist ant system, terutama terletak pada cara pheromone diperbarui dan diatur.

1. **Pheromone Update Rule:**
    a. **MAX-MIN ACO:** Pada setiap iterasi, hanya jalur terbaik (biasanya hanya satu atau beberapa jalur terbaik) yang diperbarui pheromonnya. Pheromone diatur ke nilai maksimum atau minimum yang telah ditentukan sebelumnya (misalnya, min_pheromone dan max_pheromone).
    b. **ACS dan Elitist Ant System:** Pada setiap iterasi, semua semut yang berhasil menemukan solusi diperbarui pheromonnya. Pheromone diperbarui berdasarkan kinerja semua semut dalam menemukan solusi.
2. **Pheromone Evaporation:**
    a. **MAX-MIN ACO:** Evaporasi pheromone biasanya dilakukan secara global (untuk semua jalur) dengan tingkat penguapan yang tetap atau bervariasi tetapi terbatas pada nilai minimum tertentu.
    b. **ACS dan Elitist Ant System:** Evaporasi pheromone juga dilakukan secara global, tetapi tingkat penguapan mungkin berbeda dan dapat disesuaikan selama iterasi.
3. **Konvergensi dan Diversifikasi:**
    a. **MAX-MIN ACO:** Lebih cenderung menuju konvergensi pada solusi optimal karena hanya jalur terbaik yang diperbarui pheromonenya. Namun, risiko terjebak dalam minimum lokal juga lebih tinggi.
    b. **ACS dan Elitist Ant System:** Lebih mungkin untuk menjelajahi berbagai solusi karena semua jalur yang berhasil diperbarui pheromonenya, yang dapat membantu dalam menghindari minimum lokal.
4. **Kompleksitas dan Kinerja:**

a. **MAX-MIN ACO:** Lebih efisien dalam hal kinerja karena hanya jalur terbaik yang diperbarui pheromonenya, tetapi mungkin memerlukan penyesuaian parameter yang cermat.

b. **ACS dan Elitist Ant System:** Lebih kompleks dan memerlukan lebih banyak perhitungan karena semua jalur yang berhasil diperbarui pheromonenya, tetapi mungkin lebih tahan terhadap minimum lokal dan menawarkan lebih banyak fleksibilitas.

VII. Hasil dan Pembahasan

1. ACS

Iteration: 0 | Best Distance: 1.0961295969789506

Iteration: 10 | Best Distance: 1.0225294203196846

Iteration: 20 | Best Distance: 1.0090635879971808

Iteration: 30 | Best Distance: 1.0090635879971808

Iteration: 40 | Best Distance: 1.0090635879971808

Iteration: 50 | Best Distance: 1.0090635879971808

Iteration: 60 | Best Distance: 1.0090635879971808

Iteration: 70 | Best Distance: 1.0090635879971808

Iteration: 80 | Best Distance: 1.0090635879971808

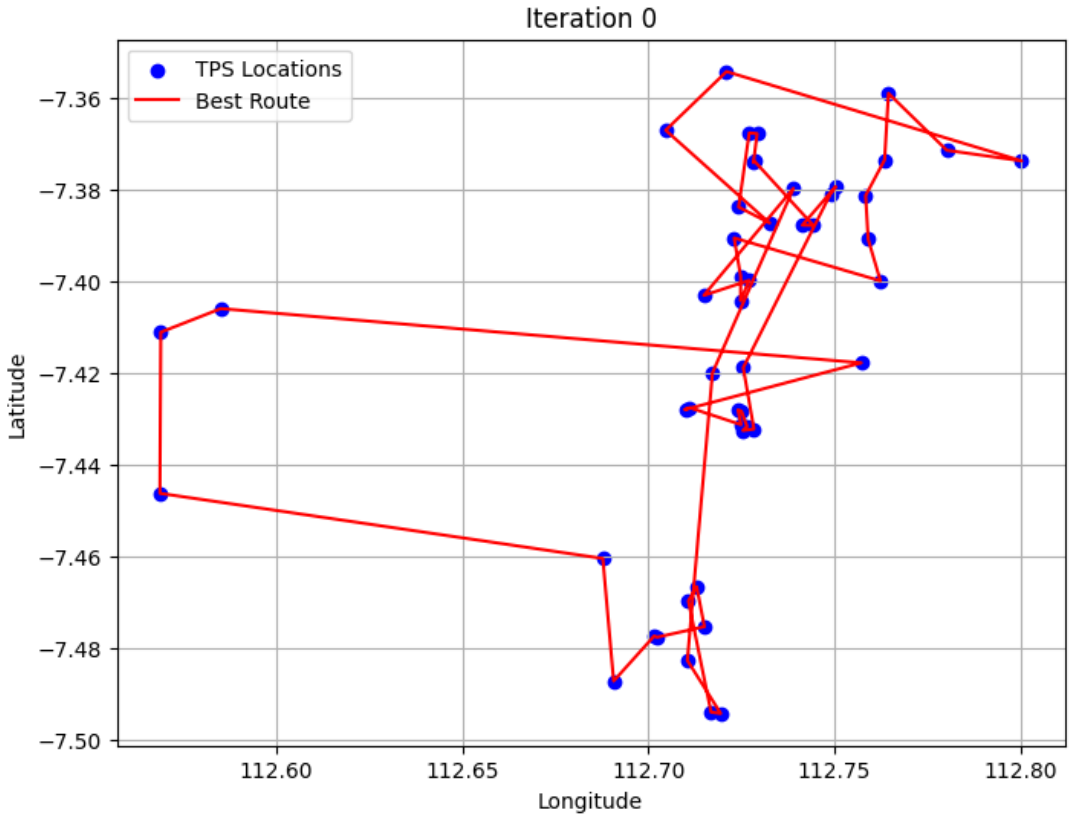
Iteration: 90 | Best Distance: 1.0090635879971808

Best Distance: 1.0090635879971808

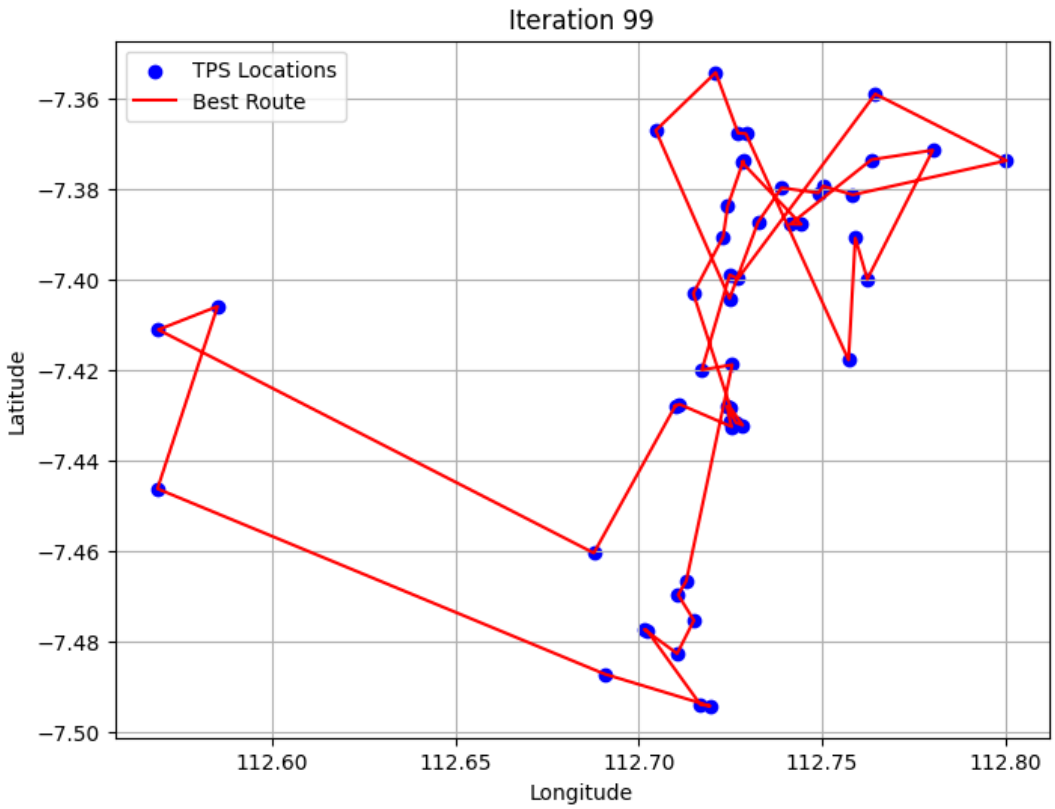
Best Route: ['Depot dummy', 'TPS Sidokerto Buduran', 'TPS Sukorejo Buduran', 'Jl. KH Khamdani Buduran', 'TPS Pondok Al-Ghozini Buduran', 'TPS SDN Siwalanpanji Buduran', 'SMA Antartika Buduran', 'SMP PGRI 1 Buduran', 'TPST Siwalanpanji Buduran', 'TPST PT Avian Buduran', 'TPS Sungai Seruni Gedangan', 'TPS Indomart Gedangan', 'TPST Kebonsikep Gedangan', 'TPS Alloy Stell Gedangan', 'TPS Restoran Aloha Gedangan', 'TPS Aloha Gedangan', 'TPS Sawotratap Gedangan', 'TPS Makro Gedangan', 'Sedati - TPS Jl. Raya Juanda', 'Sedati - TPS AURI', 'Sedati - Choirul Anan TPS Lesen', 'TPS Perum Putri Juanda Waru', 'TPS Komplek AL Tebel

Gedangan', 'TPS Karangbong Gedangan', 'Sedati - TPS Pasar Wisata Juanda', 'Sedati - TPS Jl. Raya Tropodo', 'Sedati - TPS Angkasapura', 'Sedati - TPS Bandara Juanda', 'Sedati - Miskan TPS Bea Cukai Juanda', 'TPS SMA Negeri Gedangan', 'TPS Hair Star Gedangan', 'TPS Desa Ketajen Gedangan', 'TPS Marinir Juanda Gedangan', 'TPS Gudang Garam Gedangan', 'TPS Taman Aloha Waru', 'TPS Perum Surya Asri (Dapurno) Buduran', 'Terminal Larangan Candi', 'TPS Tenggulunan Candi', 'TPS Jl. Raya Bligo Candi', 'TPS Kir Candi Candi', 'TPS Perum Candiloka Candi', 'TPS Candi Asri Candi', 'TPS Perum AL Candi', 'TPS Ps. Krempyeng Sugihwaras Candi', 'TPS Perum CSM Candi', 'TPS Jl Karang Tanjung Candi', 'TPS Perum Citra Diamond Krian', 'TPS Kemangsen Krian', 'TPS Jl Raya Krian', 'Depot dummy']

Total Runtime: 125.60040211677551 seconds



Gambar 7.1 Hasil Grafik ACS Iterasi ke 1



Gambar 7.2 Hasil Grafik ACS Iterasi ke 100

Algoritma ACS berhasil mencapai jarak terbaik sebesar 1.009, mengindikasikan bahwa rute yang dihasilkan memiliki panjang total sejauh 1.009 satuan (misalnya kilometer atau mil, tergantung pada konteks TSP). Proses pencarian solusi tersebut memakan waktu total 125.60 detik, yang mencakup waktu untuk menjalankan semua iterasi algoritma dan menemukan rute terbaik.

2. Elitist

Iteration: 0 | Best Distance: 1.1157601404386672

Iteration: 10 | Best Distance: 1.0024465529672666

Iteration: 20 | Best Distance: 0.9146273493982263

Iteration: 30 | Best Distance: 0.8305796530013776

Iteration: 40 | Best Distance: 0.8297662679154827

Iteration: 50 | Best Distance: 0.8273189695395502

Iteration: 60 | Best Distance: 0.8254254523676221

Iteration: 70 | Best Distance: 0.8254254523676221

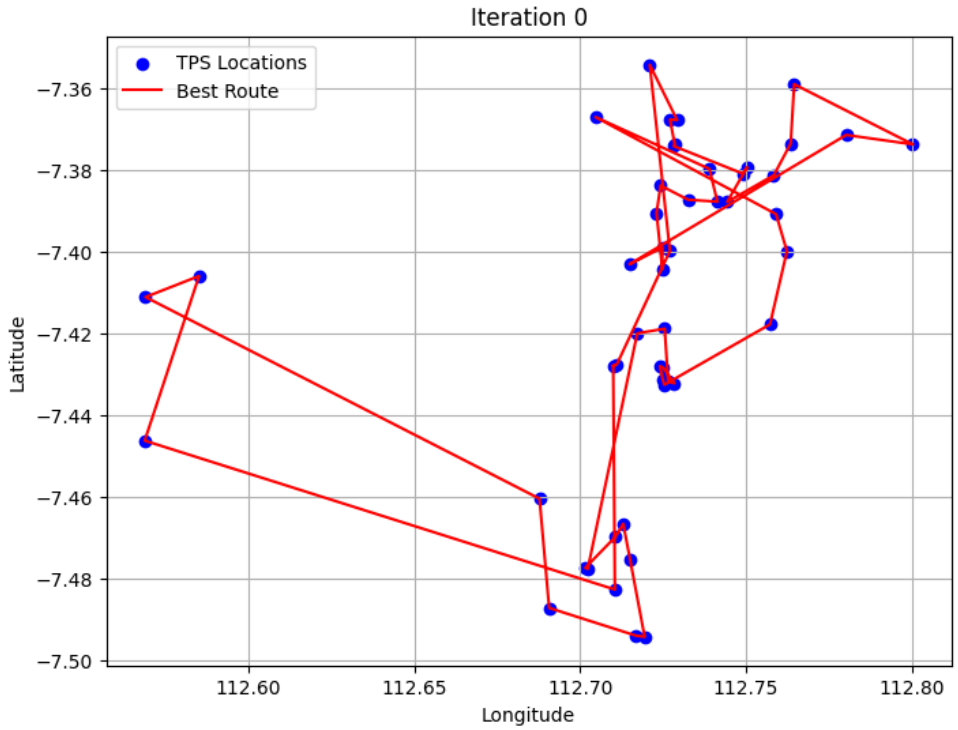
Iteration: 80 | Best Distance: 0.8254254523676221

Iteration: 90 | Best Distance: 0.8254254523676221

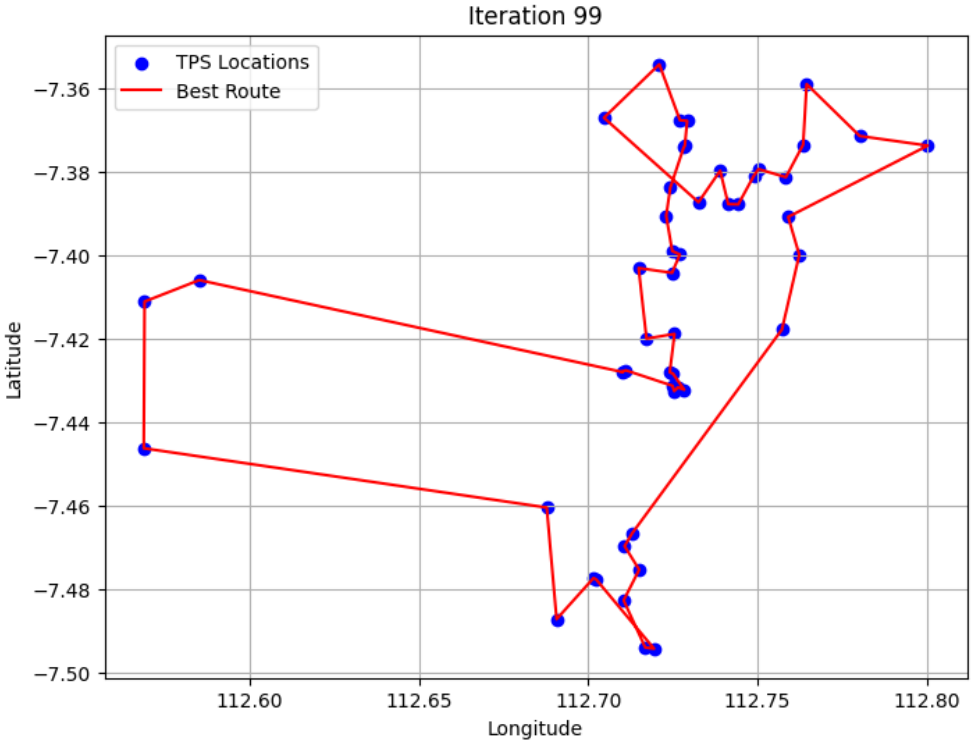
Best Distance: 0.8219164908651844

Best Route: ['Depot dummy', 'TPS Sidokerto Buduran', 'SMP PGRI 1 Buduran', 'TPST Siwalanpanji Buduran', 'SMA Antartika Buduran', 'TPS SDN Siwalanpanji Buduran', 'Jl. KH Khamdani Buduran', 'TPS Pondok Al-Ghozini Buduran', 'TPST PT Avian Buduran', 'TPS Sukorejo Buduran', 'TPS Karangbong Gedangan', 'TPS Komplek AL Tebel Gedangan', 'TPS Indomart Gedangan', 'TPS Sungai Seruni Gedangan', 'TPST Kebonsikep Gedangan', 'TPS Alloy Stell Gedangan', 'TPS Restoran Aloha Gedangan', 'TPS Aloha Gedangan', 'TPS Sawotratap Gedangan', 'TPS Makro Gedangan', 'TPS Gudang Garam Gedangan', 'TPS Taman Aloha Waru', 'TPS Desa Ketajen Gedangan', 'TPS Marinir Juanda Gedangan', 'TPS SMA Negeri Gedangan', 'TPS Hair Star Gedangan', 'Sedati - TPS Jl. Raya Juanda', 'Sedati - TPS AURI', 'Sedati - Miskan TPS Bea Cukai Juanda', 'Sedati - TPS Pasar Wisata Juanda', 'Sedati - TPS Jl. Raya Tropodo', 'Sedati - TPS Bandara Juanda', 'Sedati - TPS Angkasapura', 'Sedati - Choirul Anan TPS Lesen', 'TPS Perum Putri Juanda Waru', 'TPS Perum Surya Asri (Dapurno) Buduran', 'Terminal Larangan Candi', 'TPS Tenggulunan Candi', 'TPS Jl. Raya Bligo Candi', 'TPS Perum Candiloka Candi', 'TPS Candi Asri Candi', 'TPS Kir Candi Candi', 'TPS Perum AL Candi', 'TPS Ps. Krempyeng Sugihwaras Candi', 'TPS Jl Karang Tanjung Candi', 'TPS Perum CSM Candi', 'TPS Perum Citra Diamond Krian', 'TPS Kemangsen Krian', 'TPS Jl Raya Krian', 'Depot dummy']

Total Runtime: 52.16852831840515 seconds



Gambar 7.3 Hasil Grafik Elitist Iterasi ke 1



Gambar 7.4 Hasil Grafik Elitist Iterasi ke 100

Algoritma Elitist berhasil mencapai jarak terbaik sebesar 0.822, menunjukkan bahwa rute yang dihasilkan memiliki panjang total sejauh 0.822 satuan (misalnya kilometer atau mil, tergantung pada konteks TSP). Total waktu yang dibutuhkan untuk mencapai solusi tersebut adalah 52.17 detik, termasuk waktu yang diperlukan untuk menjalankan semua iterasi algoritma dan menemukan rute terbaik.

3. Max-Min

Iteration: 0 | Best Distance: 1.1587043986465213

Iteration: 10 | Best Distance: 0.8956586394744894

Iteration: 20 | Best Distance: 0.8955174345231711

Iteration: 30 | Best Distance: 0.8679816587603781

Iteration: 40 | Best Distance: 0.8492503230861834

Iteration: 50 | Best Distance: 0.8260953565478402

Iteration: 60 | Best Distance: 0.8073469670050171

Iteration: 70 | Best Distance: 0.8018586875377897

Iteration: 80 | Best Distance: 0.8012432829751602

Iteration: 90 | Best Distance: 0.8012432829751602

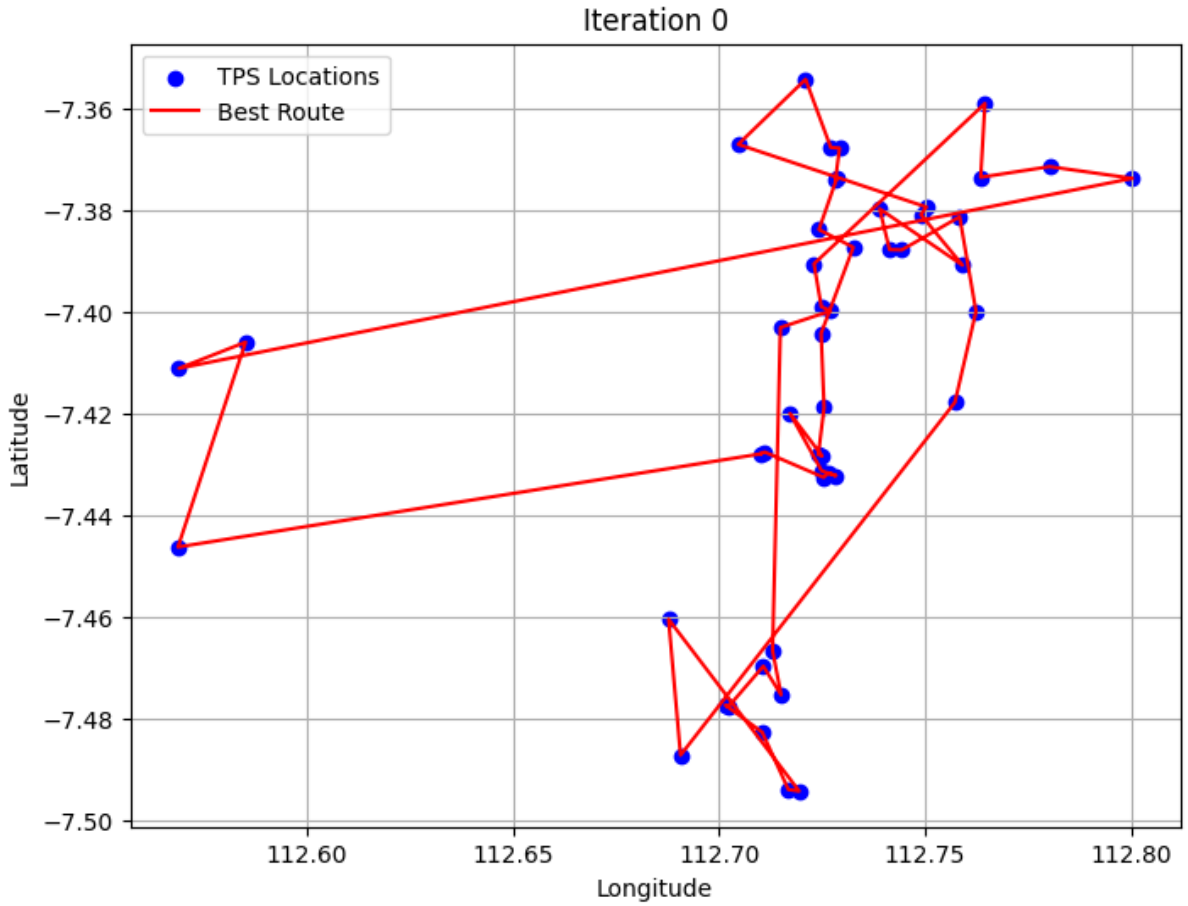
Best Distance: 0.8012432829751602

Best Route: ['Depot dummy', 'TPS Sidokerto Buduran', 'TPS Sukorejo Buduran', 'TPST PT Avian Buduran', 'TPS Pondok Al-Ghozini Buduran', 'Jl. KH Khamdani Buduran', 'SMP PGRI 1 Buduran', 'SMA Antartika Buduran', 'TPST Siwalanpanji Buduran', 'TPS SDN Siwalanpanji Buduran', 'Terminal Larangan Candi', 'TPS Tenggulunan Candi', 'TPS Jl. Raya Bligo Candi', 'TPS Kir Candi Candi', 'TPS Candi Asri Candi', 'TPS Perum Candiloka Candi', 'TPS Ps. Krempyeng Sugihwaras Candi', 'TPS Perum AL Candi', 'TPS Jl Karang Tanjung Candi', 'TPS Perum CSM Candi', 'TPS Petemon Waru', 'TPS Waru Sukodono', 'TPS Buduran 2', 'TPS Buduran', 'TPS Pandanrejo Waru', 'TPS Makro Gedangan', 'TPS Waru', 'TPS Aloha Gedangan', 'TPS DK Tanjung Gedangan', 'TPS Pasar Puri Candi', 'TPS Tanggulangin', 'TPS Jl. Raya Tropodo Gedangan', 'TPS Jl. Krembangan Dalam Gedangan', 'TPS Antasari Waru', 'TPS Perum Surya Asri (Dapurno) Buduran', 'TPS Hair Star Gedangan', 'TPS Candi II Waru', 'TPS Desa Ketajen Gedangan', 'TPS Punggul Candi', 'TPS Sangivara Candi', 'TPS Sungai Seruni

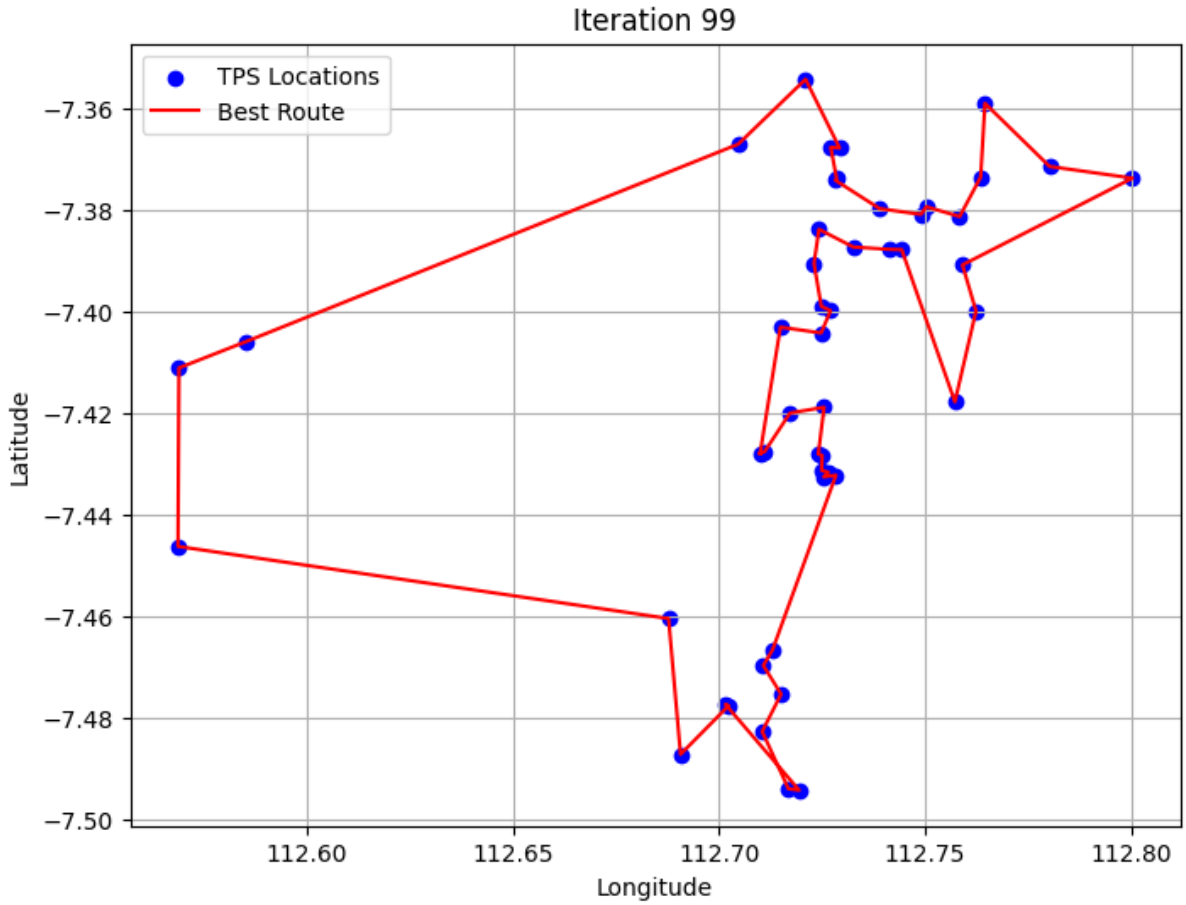
Gedangan', 'TPS Jl. Pahlawan Gedangan', 'TPS Brawijaya Waru', 'TPS Jl. Raya Manyar Gedangan', 'Depot dummy']

Total Runtime: 36.2403609752655 seconds

Algoritma Max-Min berhasil mencapai jarak terbaik sebesar 0.804, menunjukkan bahwa rute yang dihasilkan memiliki panjang total sejauh 0.804 satuan (misalnya kilometer atau mil, tergantung pada konteks TSP). Proses pencarian solusi tersebut memakan waktu total 34.15 detik, termasuk waktu yang diperlukan untuk menjalankan semua iterasi algoritma dan menemukan rute terbaik.



Gambar 7.5 Hasil Grafik MAX-MIN Iterasi ke 1



Gambar 7.6 Hasil Grafik MAX-MIN Iterasi ke 100

4. Tabel Perbandingan

Berikut perbandingan ketiga performa yang disajikan dalam bentuk tabel :

Nama	ACS	Elitist ACO	Max-Min ACO
Best Distance			
Total Runtime (s)			

VIII. Percobaan Algoritma MAXMIN ACO

Setelah mengetahui bahwa algoritma Max-min ACO menghasilkan jarak terpendek dan waktu tempuh tercepat, selanjutnya kami mencoba melakukan tiga kali percobaan dan tiap percobaan 10 kali *run.* Dengan detail variabel yang diubah sebagai berikut :

Percobaan 1 :

- num ants : 80

- num iteration : 100
- min pheromone : 0.1
- max pheromone : 10

Percobaan 2 :

- num ants : 50
- num interation : 100
- min pheromone : 0.1
- max pheromone : 10

Percobaan 3 :

- num ants : 80
- num iteration : 80
- min pheromone : 1
- max pheromone : 10

Didapatkan hasil sebagai beirkut :

	Percobaan 1		**Percobaan 2**		**Percobaan 3**	
	Best Distance	Runtime	Best Distance	Runtime	Best Distance	Runtime
Run ke-1	0,80	62,85	0,83	40,38	0,87	27,26
Run ke-2	0,81	39,52	0,84	21,48	0,92	26,85
Run ke-3	0,81	46,24	0,84	21,33	0,91	34,64
Run ke-4	0,81	41,46	0,82	22,03	0,89	27,42
Run ke-5	0,80	40,19	0,81	21,32	0,85	33,45
Run ke-6	0,81	41,47	0,82	21,56	0,89	27,72
Run ke-7	0,81	42,28	0,81	20,80	0,87	32,69
Run ke-8	0,81	41,39	0,81	23,34	0,93	28,50
Run ke-9	0,82	41,76	0,83	22,82	0,89	32,94
Run ke-10	0,81	40,94	0,81	21,72	0,89	27,08
Mean	**0,81**	**43,81**	**0,82**	**23,68**	**0,89**	**29,85**
SD	**0,01**	**6,92**	**0,01**	**5,92**	**0,02**	**3,15**

Dengan rata-rata *best distance* keseluruhan percobaan adalah 0,84 dan rata-rata *runtime* adalah 32,45. Hasil ini tidak jauh berbeda dengan hasil yang didapatkan sebelumnya.

IX. Kesimpulan

Dari hasil TSP ACO menggunakan algoritma ACS, Elitist, dan Max-Min, dapat ditarik beberapa kesimpulan:

1. Performa Algoritma:
   - **ACS (Ant Colony System):** Algoritma ini memperoleh jarak terbaik sebesar 1.009 pada iterasi ke-40. Namun, secara keseluruhan, algoritma ini memerlukan waktu eksekusi yang cukup lama, yaitu 125.60 detik.
   - **Elitist:** Algoritma ini mendemonstrasikan performa yang baik dengan jarak terbaik 0.821 pada iterasi ke-99. Selain itu, waktu eksekusi yang lebih singkat, yaitu 52.17 detik, menunjukkan efisiensi yang lebih tinggi dibandingkan dengan ACS.
   - **Max-Min**: Algoritma ini memperoleh jarak terbaik 0.801 pada iterasi ke-80, dengan waktu eksekusi yang paling singkat, yaitu 36.24 detik. Ini menunjukkan bahwa algoritma Max-Min memiliki keseimbangan antara efisiensi waktu dan kualitas solusi.

2. Efisiensi Waktu:
   Dilihat dari waktu eksekusi, algoritma Max-Min menonjol dengan waktu yang lebih singkat, diikuti oleh Elitist, dan terakhir ACS. Ini menunjukkan bahwa Max-Min lebih efisien dalam menghasilkan solusi dalam jangka waktu yang lebih singkat.
3. Kualitas Solusi:
   Meskipun waktu eksekusi lebih singkat, baik Elitist Ant System maupun Max-Min Ant System mampu memberikan solusi dengan kualitas yang kompetitif. Meskipun demikian, ACS juga mampu mencapai hasil yang kompetitif dalam hal kualitas solusi, meskipun memerlukan waktu eksekusi yang lebih lama.

Jika waktu eksekusi menjadi faktor kritis, Max-Min Ant System adalah pilihan terbaik karena mampu memberikan solusi yang baik dalam waktu eksekusi yang relatif lebih singkat.. Sementara itu, Elitist dan ACS mungkin tidak menjadi pilihan terbaik jika waktu eksekusi

menjadi faktor yang sangat penting, meskipun mampu memberikan solusi yang bersaing dalam hal kualitas.

X. Daftar Pustaka

Arista, V. D. (2024, Mei 1). *Sidoarjo Butuh Penambahan Armada Pengangkut Sampah*. Diambil kembali dari radarsidoarjo.id: https://radarsidoarjo.jawapos.com/kota-delta/85936168/sidoarjo-butuh-penambahan-armada-pengangkut-sampah

Illahi, N. P. (2024, Februari 23). *Tumpukan Sampah Menggunung di Belakang Terminal Purabaya, Begini Solusi dari DLHK Kabupaten Sidoarjo*. Diambil kembali dari jawapos.com: https://www.jawapos.com/surabaya-raya/014365816/tumpukan-sampah-menggunung-di-belakang-terminal-purabaya-begini-solusi-dari-dlhk-kabupaten-sidoarjo

Karim, R. I. (2023, Mei 17). *Retribusi Sampah Memberatkan, Bupati Sidoarjo Janji Revisi Perbup*. Diambil kembali dari liputanjatim.com: https://www.liputanjatim.com/retribusi-sampah-memberatkan-bupati-sidoarjo-janji-revisi-perbup/

Leksono, A., & Sarwadi, S. (2009). Algoritma Ant Colony Optimization (ACO) untuk menyelesaikan Traveling Salesman Problem (TSP) [Undergraduate thesis, Mathematics and Natural Science].