Laporan Soft Computing

OPTIMASI RUTE PENYEBARAN BROSUR BIMBINGAN BELAJAR MENGGUNAKAN ALGORITMA MAX-MIN ANT SYSTEM

Disusun Oleh :

Andira Yulianengtias 5026211038

DEPARTEMEN SISTEM INFORMASI

FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS

INSTITUT TEKNOLOGI SEPULUH NOPEMBER SURABAYA

2024

Data :

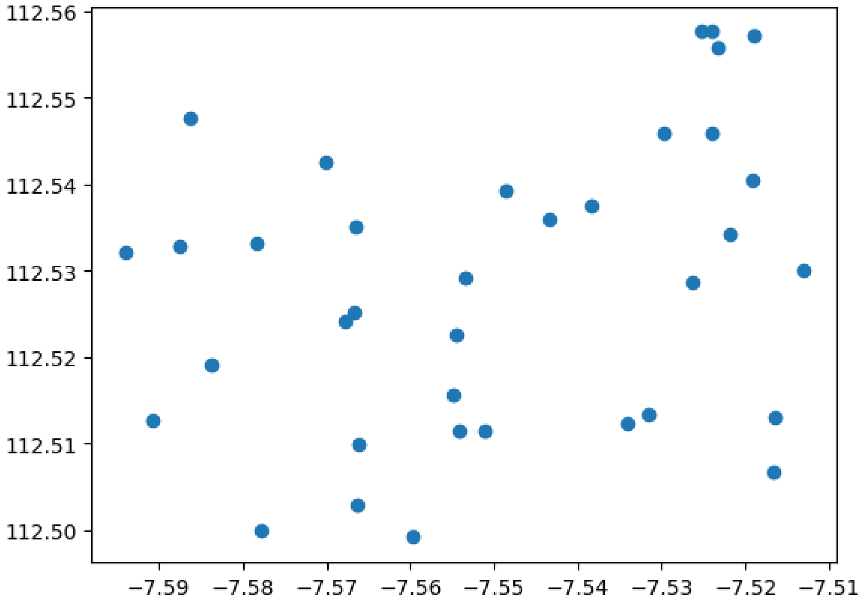
data latitude dan longitude SDN Gedangan 1 diganti karena tidak sesuai dengan yang ada di peta

Tempat Lokasi	Latitude	Longitude
SDN Gedangan 1	-7,5597	112,4992
SDN Gedangan 2	-7,5663	112,503
SDN Jiyu 1	-7,5863	112,5476
SDN Jiyu 2	-7,5701	112,5425
SDN Kaligoro	-7,5165	112,513
SDN Karangasem	-7,5166	112,5068
SDN Karangdiyeng 1	-7,5542	112,5114
SDN Karangdiyeng 2	-7,5511	112,5114
SDN Kepuharum	-7,5779	112,5
SDN Kepuhpandak 1	-7,5341	112,5124
SDN Kepuhpandak 2	-7,5315	112,5134
SDN Kertosari	-7,5837	112,5191
SDN Ketidur	-7,5665	112,5351
SDN Kutorejo	-7,5661	112,5099
SDN Payungrejo	-7,5908	112,5126
SDN Sampangagung 1	-7,5783	112,5332
SDN Sampangagung 2	-7,5875	112,5329
SDN Sawo 1 Kutorejo	-7,5545	112,5225
SDN Sawo 2 Kutorejo	-7,5534	112,5292
SDN Simbaringin	-7,594	112,5322
SDN Singowangi	-7,5384	112,5376
SDN Windurejo 1	-7,5666	112,5252
SDN Windurejo 2	-7,5677	112,5242
SDN Wonodadi 1	-7,5486	112,5392
SDN Wonodadi 2	-7,5434	112,536
SD Islam Roudlotul Qur’an	-7,5548	112,5156
SDN Belahantengah	-7,5298	112,5459
SDN Mojosulur 1	-7,5239	112,5458
SDN Mojosulur 2	-7,5232	112,5557
SDN Menanggal	-7,5191	112,5405

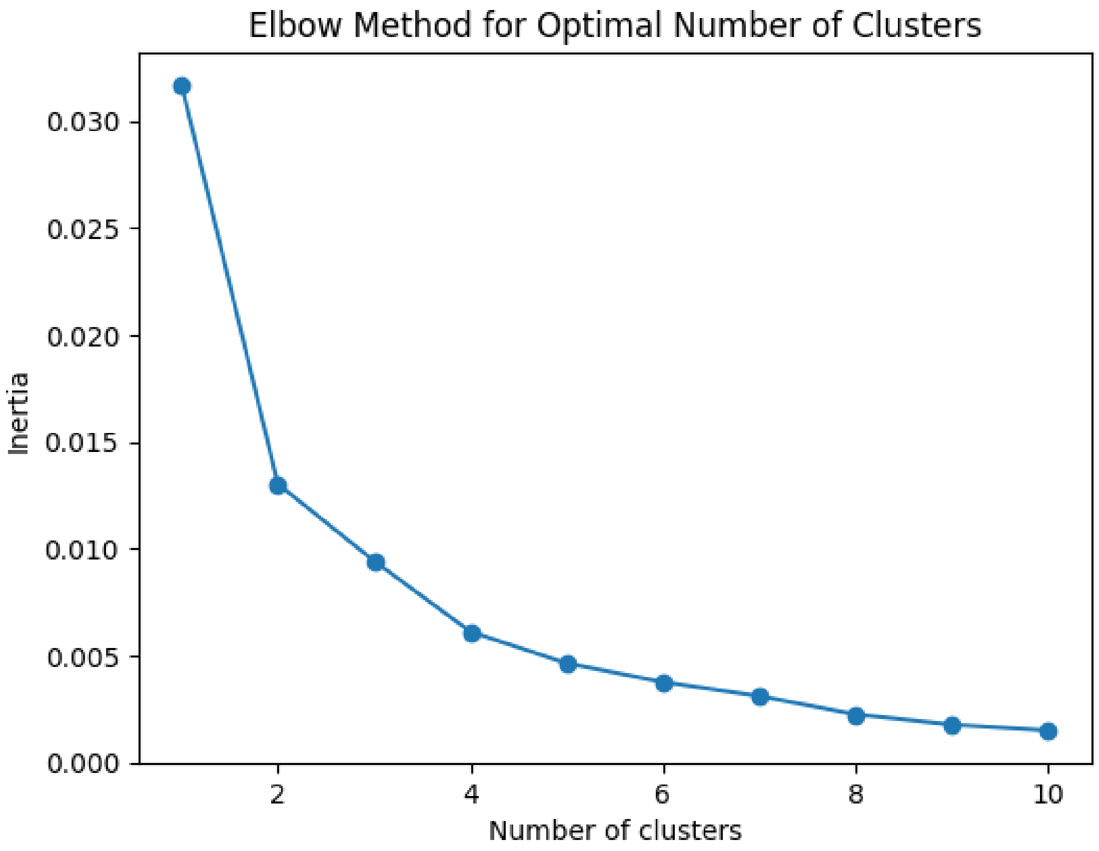
SDN Kauman	-7,5240	112,5576
SDN Awang Awang	-7,5253	112,5576
SDN Sumbertanggul 1	-7,5264	112,5286
SDN Sumbertanggul 2	-7,5219	112,5342
SDN Pekukuhan	-7,5131	112,5300
SDN Seduri 1	-7,5190	112,5572
BBC	-7,5389	112,5248

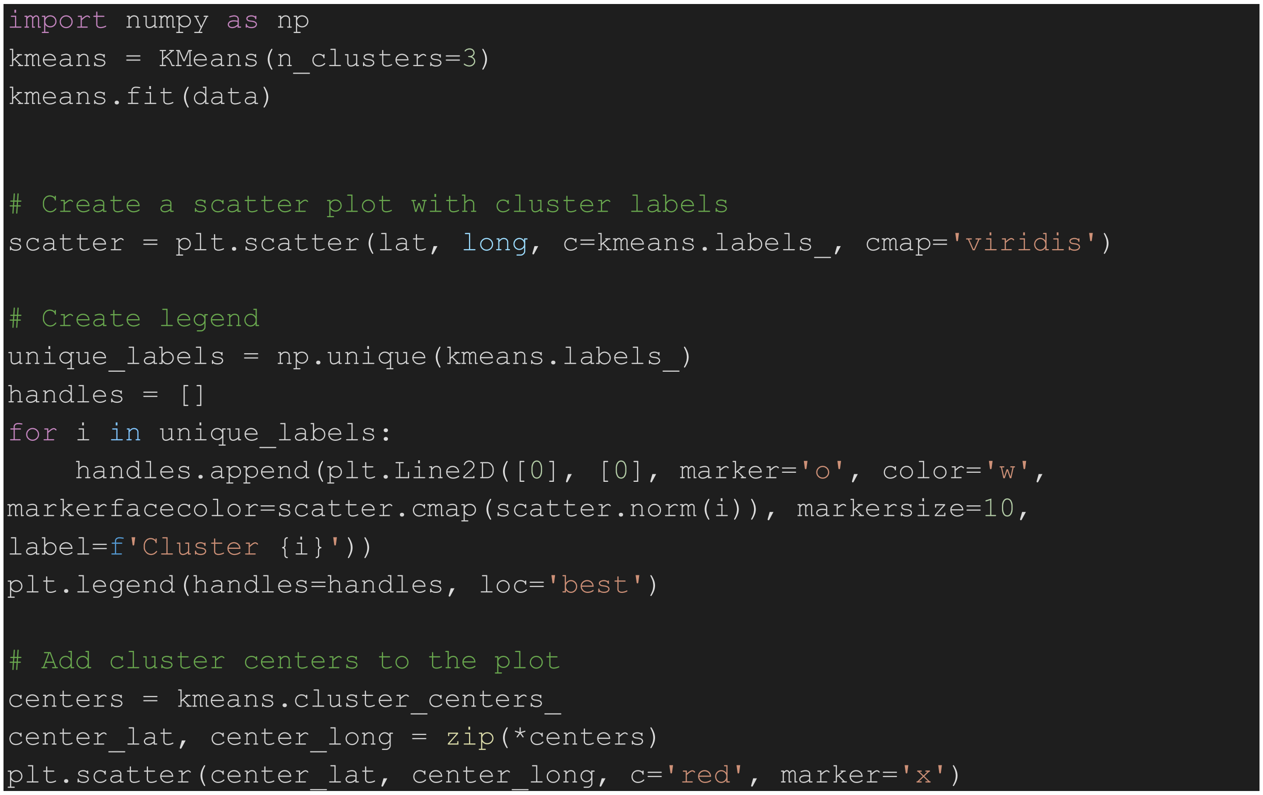
CLUSTERING

```
import matplotlib.pyplot as plt

# Data sekolah
sekolah = ["SDN Gedangan 1", "SDN Gedangan 2", "SDN Jiyu 1", "SDN Jiyu 2",
"SDN Kaligoro", "SDN Karangasem",
           "SDN Karangdiyeng 1", "SDN Karangdiyeng 2", "SDN Kepuharum",
"SDN Kepuhpandak 1", "SDN Kepuhpandak 2",
           "SDN Kertosari", "SDN Ketidur", "SDN Kutorejo", "SDN
Payungrejo", "SDN Sampangagung 1", "SDN Sampangagung 2",
           "SDN Sawo 1 Kutorejo", "SDN Sawo 2 Kutorejo", "SDN
Simbaringin", "SDN Singowangi", "SDN Windurejo 1",
           "SDN Windurejo 2", "SDN Wonodadi 1", "SDN Wonodadi 2", "SD
Islam Roudlotul Qur’an", "SDN Belahantengah",
           "SDN Mojosulur 1", "SDN Mojosulur 2", "SDN Menanggal", "SDN
Kauman", "SDN Awang Awang", "SDN Sumbertanggul 1",
           "SDN Sumbertanggul 2", "SDN Pekukuhan", "SDN Seduri 1"]

# Data koordinat
lat = [-7.5597, -7.5663, -7.5863, -7.5701, -7.5165, -7.5166, -7.5542, -
7.5511, -7.5779, -7.5341,
       -7.5315, -7.5837, -7.5665, -7.5661, -7.5908, -7.5783, -7.5875, -
7.5545, -7.5534, -7.594,
       -7.5384, -7.5666, -7.5677, -7.5486, -7.5434, -7.5548, -7.5298, -
7.5239, -7.5232, -7.5191,
       -7.5240, -7.5253, -7.5264, -7.5219, -7.5131, -7.5190]

long = [112.4992, 112.503, 112.5476, 112.5425, 112.513, 112.5068,
112.5114, 112.5114, 112.5, 112.5124,
        112.5134, 112.5191, 112.5351, 112.5099, 112.5126, 112.5332,
112.5329, 112.5225, 112.5292, 112.5322,
        112.5376, 112.5252, 112.5242, 112.5392, 112.536, 112.5156,
112.5459, 112.5458, 112.5557, 112.5405,
        112.5576, 112.5576, 112.5286, 112.5342, 112.5300, 112.5572]

plt.scatter(lat, long)
plt.show()
```

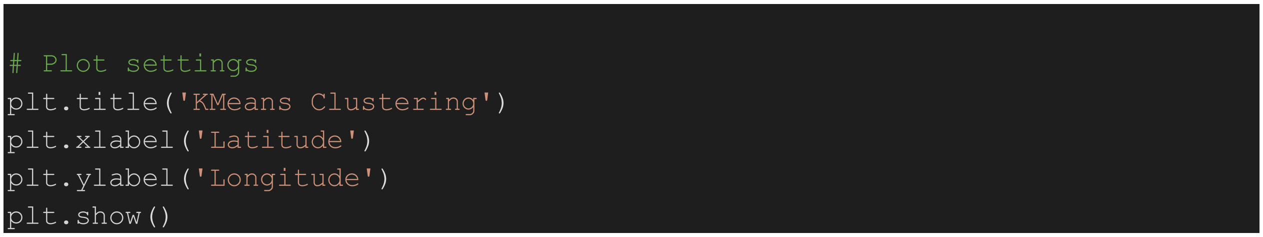


```
from sklearn.cluster import KMeans
# Combine lat and long into data points
data = list(zip(lat, long))
inertias = []

# KMeans clustering and inertia calculation
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

# Plotting the Elbow method
plt.plot(range(1, 11), inertias, marker='o')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

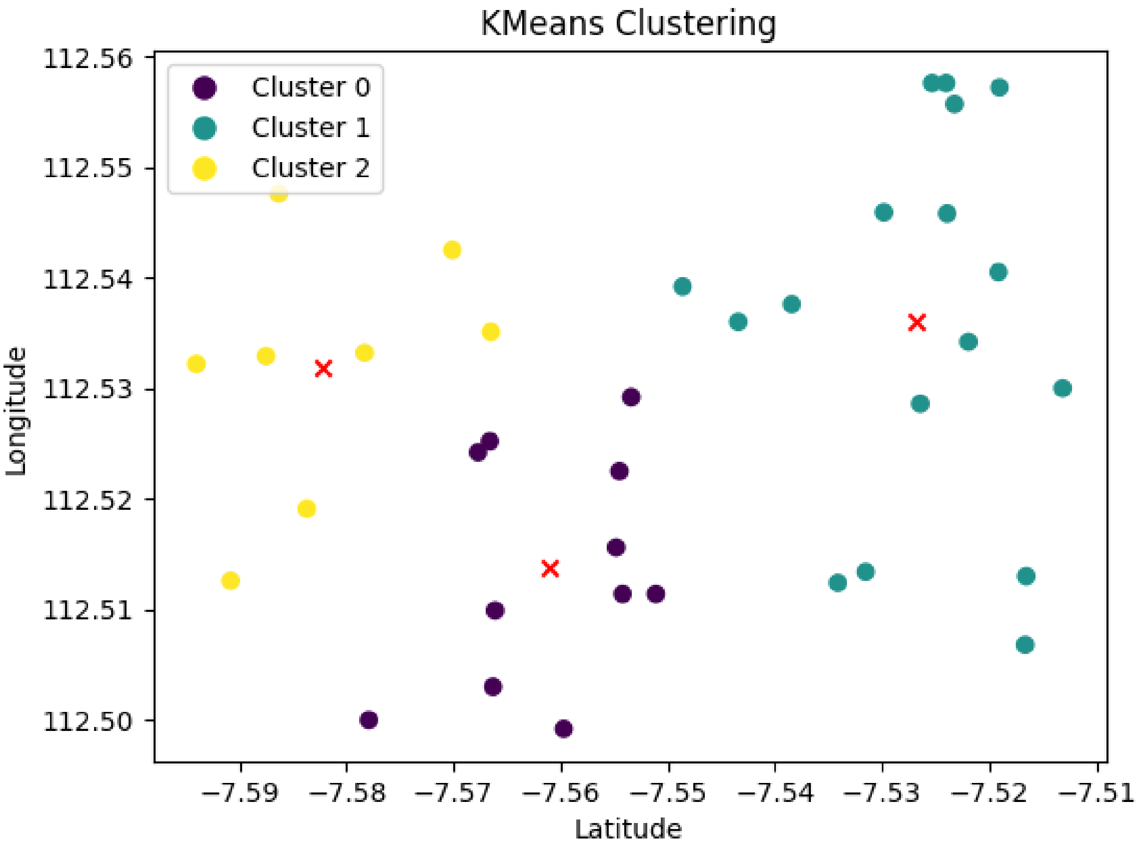


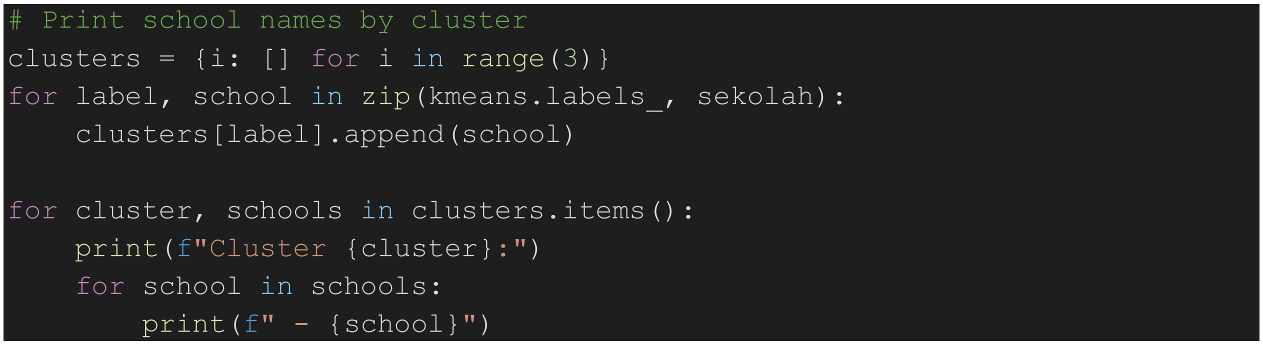


```
import numpy as np
kmeans = KMeans(n_clusters=3)
kmeans.fit(data)


# Create a scatter plot with cluster labels
scatter = plt.scatter(lat, long, c=kmeans.labels_, cmap='viridis')

# Create legend
unique_labels = np.unique(kmeans.labels_)
handles = []
for i in unique_labels:
    handles.append(plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=scatter.cmap(scatter.norm(i)), markersize=10,
label=f'Cluster {i}'))
plt.legend(handles=handles, loc='best')

# Add cluster centers to the plot
centers = kmeans.cluster_centers_
center_lat, center_long = zip(*centers)
plt.scatter(center_lat, center_long, c='red', marker='x')
```



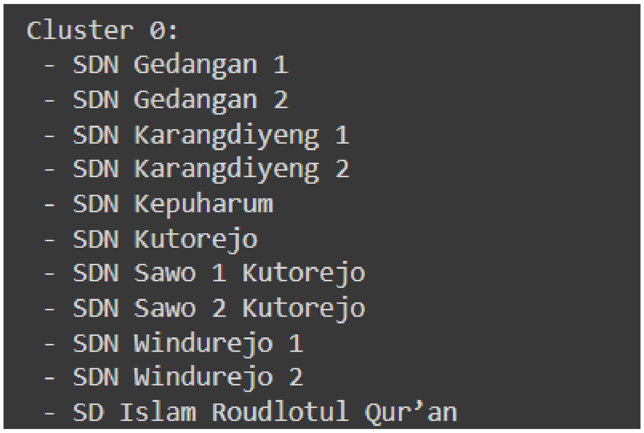
```
# Plot settings
plt.title('KMeans Clustering')
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.show()
```



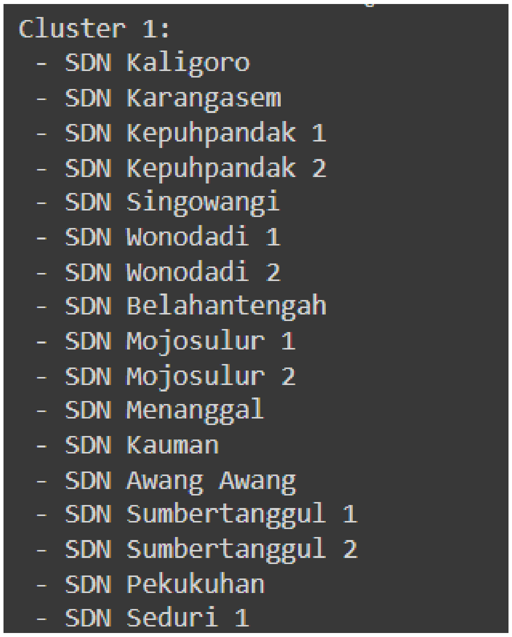


```
# Print school names by cluster
clusters = {i: [] for i in range(3)}
for label, school in zip(kmeans.labels_, sekolah):
    clusters[label].append(school)

for cluster, schools in clusters.items():
    print(f"Cluster {cluster}:")
    for school in schools:
        print(f" - {school}")
```

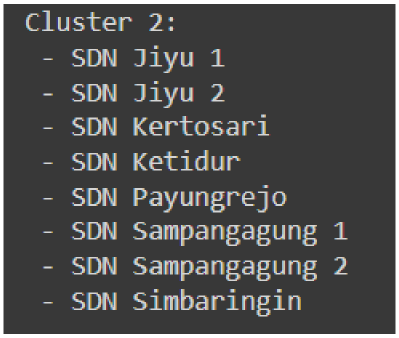
Didapatkan hasil



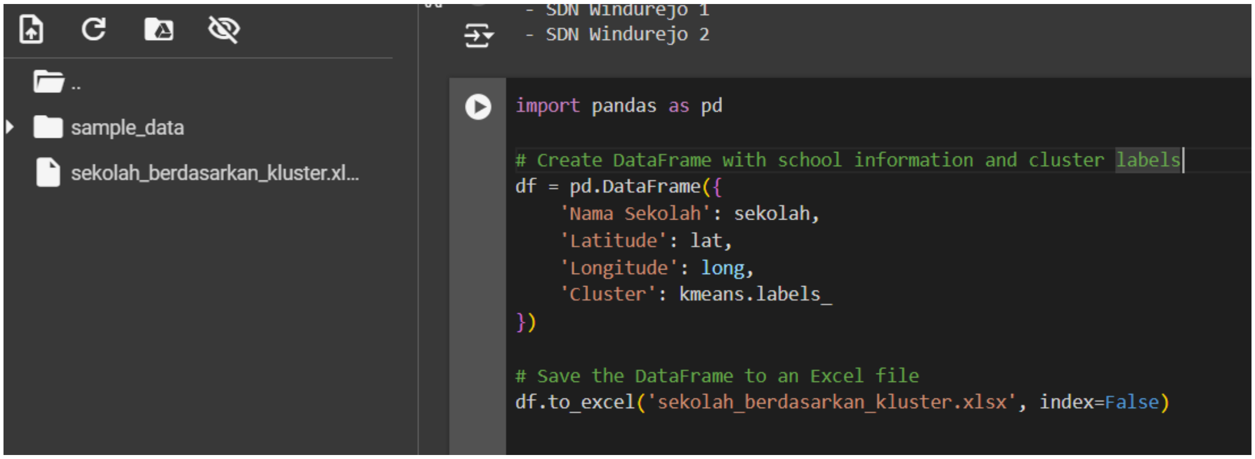
```
Cluster 0:
 - SDN Gedangan 1
 - SDN Gedangan 2
 - SDN Karangdiyeng 1
 - SDN Karangdiyeng 2
 - SDN Kepuharum
 - SDN Kutorejo
 - SDN Sawo 1 Kutorejo
 - SDN Sawo 2 Kutorejo
 - SDN Windurejo 1
 - SDN Windurejo 2
 - SD Islam Roudlotul Qur'an
```



```
Cluster 1:
 - SDN Kaligoro
 - SDN Karangasem
 - SDN Kepuhpandak 1
 - SDN Kepuhpandak 2
 - SDN Singowangi
 - SDN Wonodadi 1
 - SDN Wonodadi 2
 - SDN Belahantengah
 - SDN Mojosulur 1
 - SDN Mojosulur 2
 - SDN Menanggal
 - SDN Kauman
 - SDN Awang Awang
 - SDN Sumbertanggul 1
 - SDN Sumbertanggul 2
 - SDN Pekukuhan
 - SDN Seduri 1
```



```
Cluster 2:
 - SDN Jiyu 1
 - SDN Jiyu 2
 - SDN Kertosari
 - SDN Ketidur
 - SDN Payungrejo
 - SDN Sampangagung 1
 - SDN Sampangagung 2
 - SDN Simbaringin
```

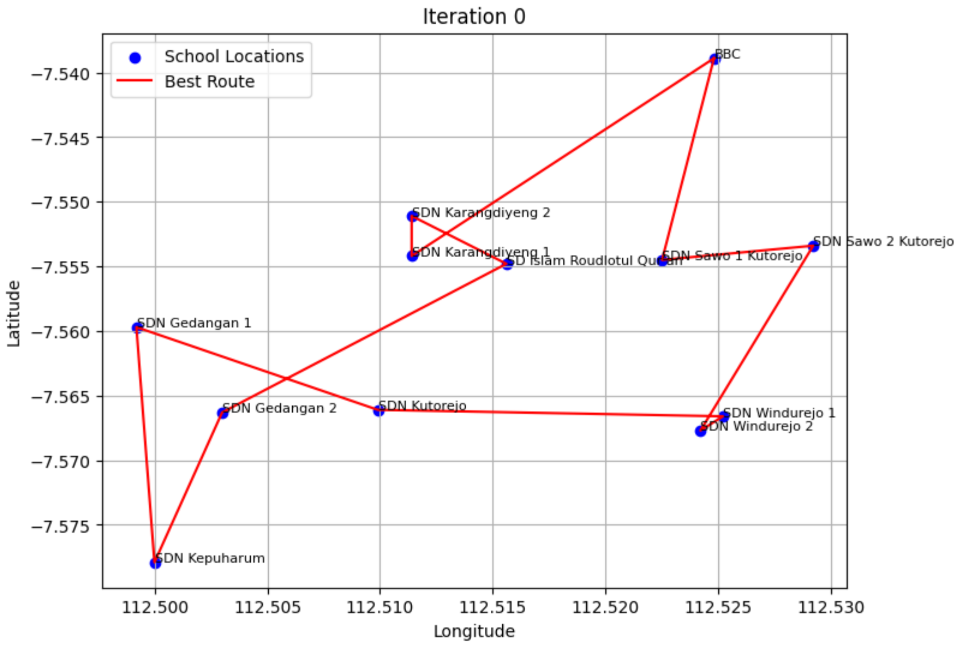


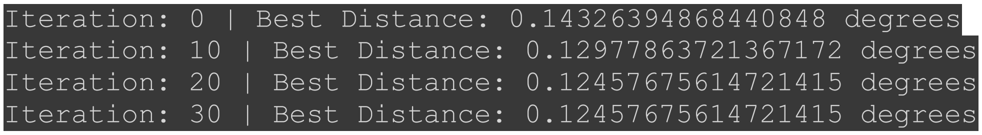
```
 - SDN Windurejo 1
 - SDN Windurejo 2
```

```python
import pandas as pd

# Create DataFrame with school information and cluster labels
df = pd.DataFrame({
    'Nama Sekolah': sekolah,
    'Latitude': lat,
    'Longitude': long,
    'Cluster': kmeans.labels_
})

# Save the DataFrame to an Excel file
df.to_excel('sekolah_berdasarkan_kluster.xlsx', index=False)
```

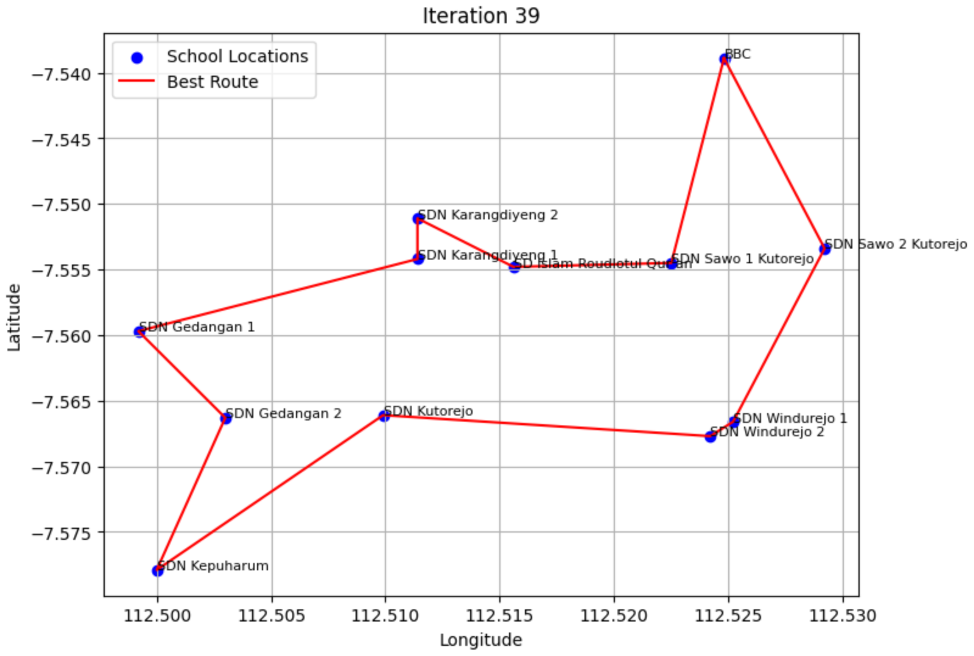
OPTIMASI MENGGUNAKAN MAX-MIN ANT SYSTEM

CLUSTER 0





```
Iteration: 0 | Best Distance: 0.14326394868440848 degrees
Iteration: 10 | Best Distance: 0.12977863721367172 degrees
Iteration: 20 | Best Distance: 0.12457675614721415 degrees
Iteration: 30 | Best Distance: 0.12457675614721415 degrees
```



Best Distance: 0.12457675614721415 degrees

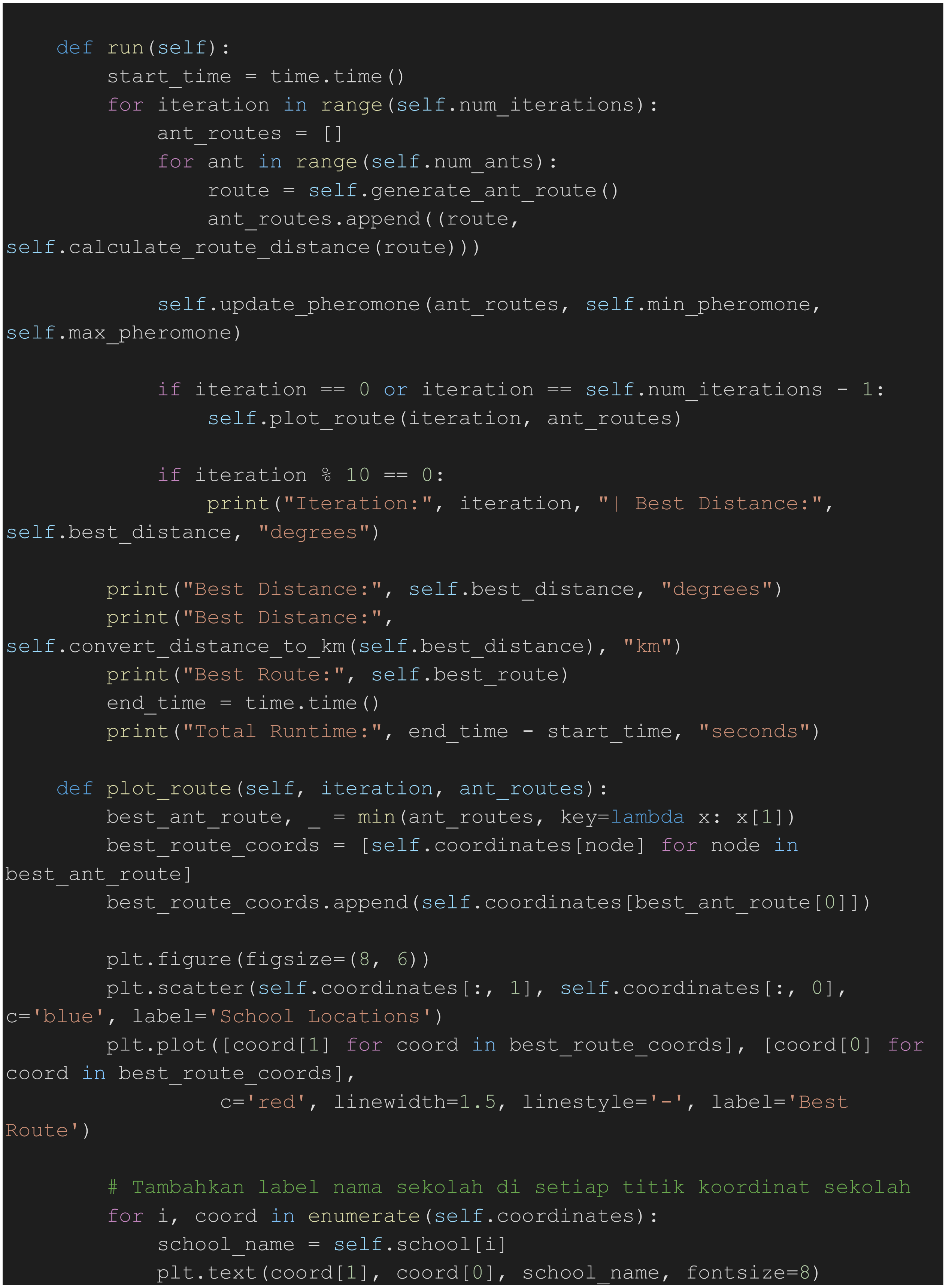
Best Distance: 13.867884494307878 km

Best Route: ['BBC', 'SDN Sawo 2 Kutorejo', 'SDN Windurejo 1', 'SDN Windurejo 2', 'SDN Kutorejo', 'SDN Kepuharum', 'SDN Gedangan 2', 'SDN Gedangan 1', 'SDN Karangdiyeng 1', 'SDN Karangdiyeng 2', 'SD Islam Roudlotul Qur'an', 'SDN Sawo 1 Kutorejo', 'BBC']

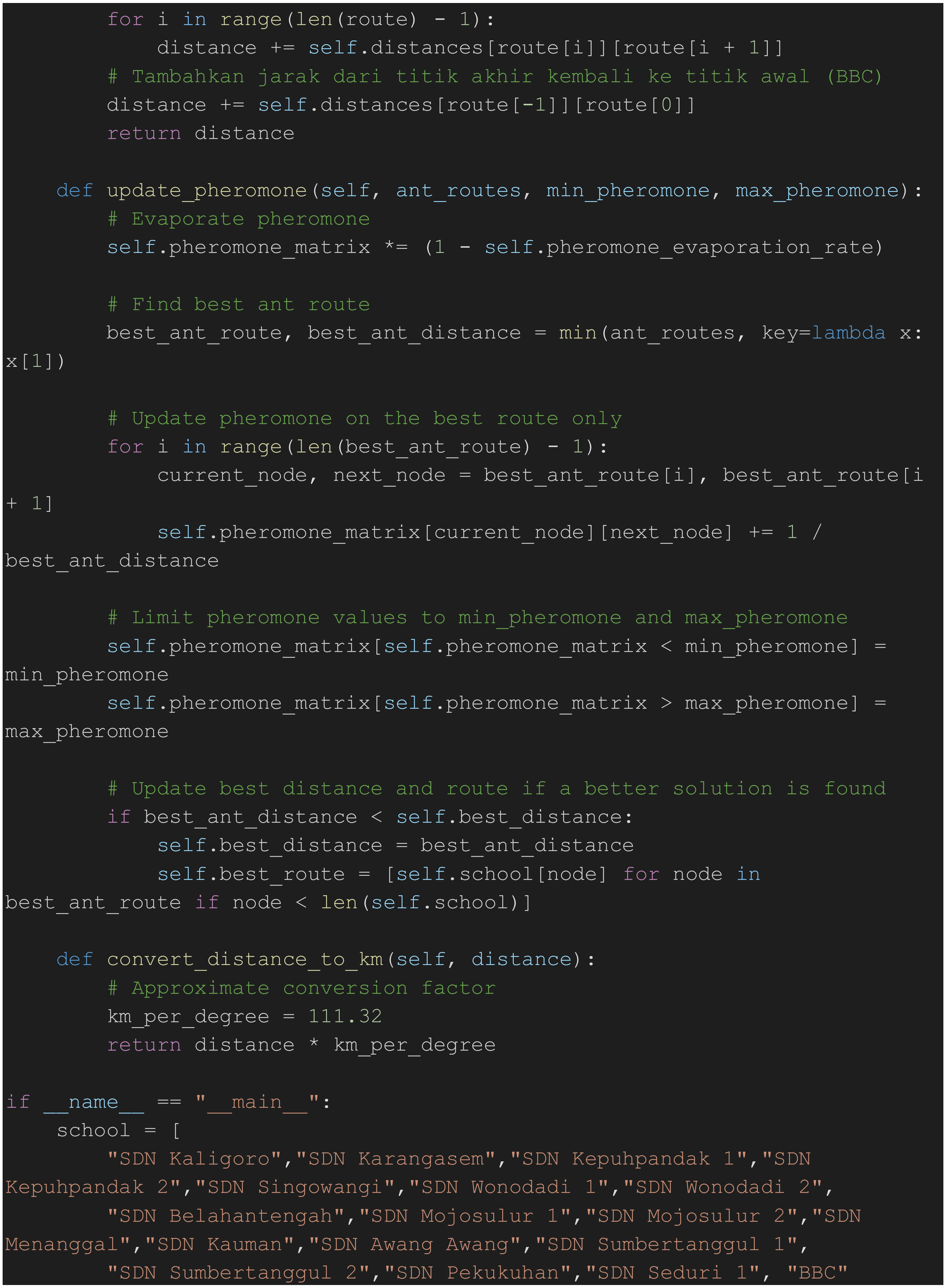
Total Runtime: 1.475968360900879 seconds

CLUSTER 1

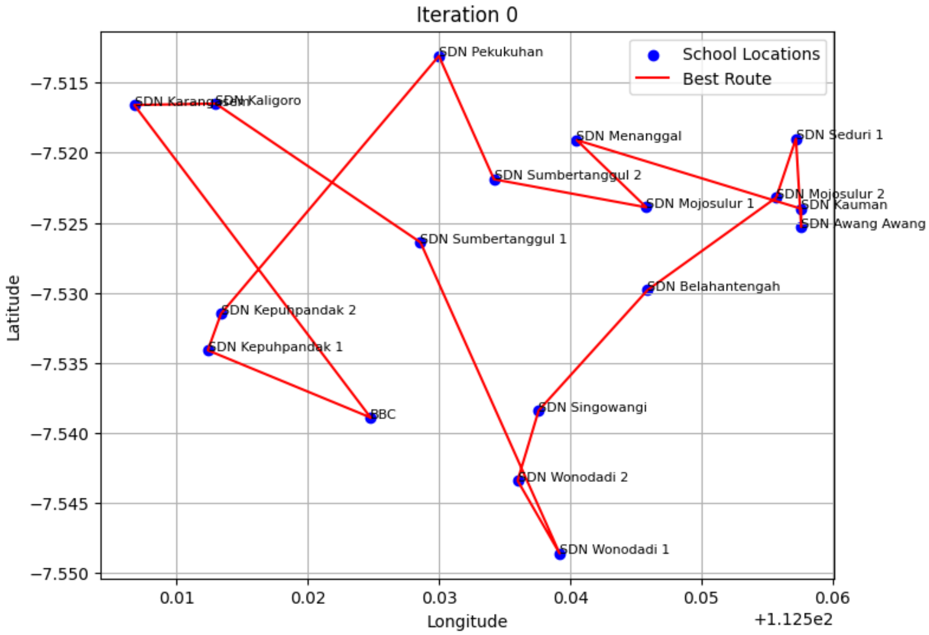
```
import numpy as np
import matplotlib.pyplot as plt
import time

class AntColony:
    def __init__(self, num_ants, num_iterations,
pheromone_evaporation_rate, alpha, beta, school, 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.school = school
        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
```

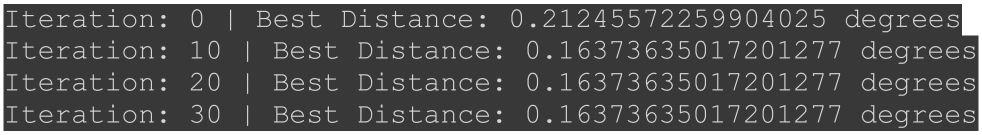


```
    def run(self):
        start_time = time.time()
        for iteration in range(self.num_iterations):
            ant_routes = []
            for ant in range(self.num_ants):
                route = self.generate_ant_route()
                ant_routes.append((route,
self.calculate_route_distance(route)))

            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, "degrees")

        print("Best Distance:", self.best_distance, "degrees")
        print("Best Distance:",
self.convert_distance_to_km(self.best_distance), "km")
        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='School 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')

        # Tambahkan label nama sekolah di setiap titik koordinat sekolah
        for i, coord in enumerate(self.coordinates):
            school_name = self.school[i]
            plt.text(coord[1], coord[0], school_name, fontsize=8)
        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 = self.coordinates.shape[0] - 1  # Index titik awal
(BBC)
        end_node = self.coordinates.shape[0] - 1    # Index titik akhir
(BBC)
        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(end_node)  # Tambahkan titik akhir (BBC) ke rute
        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
```

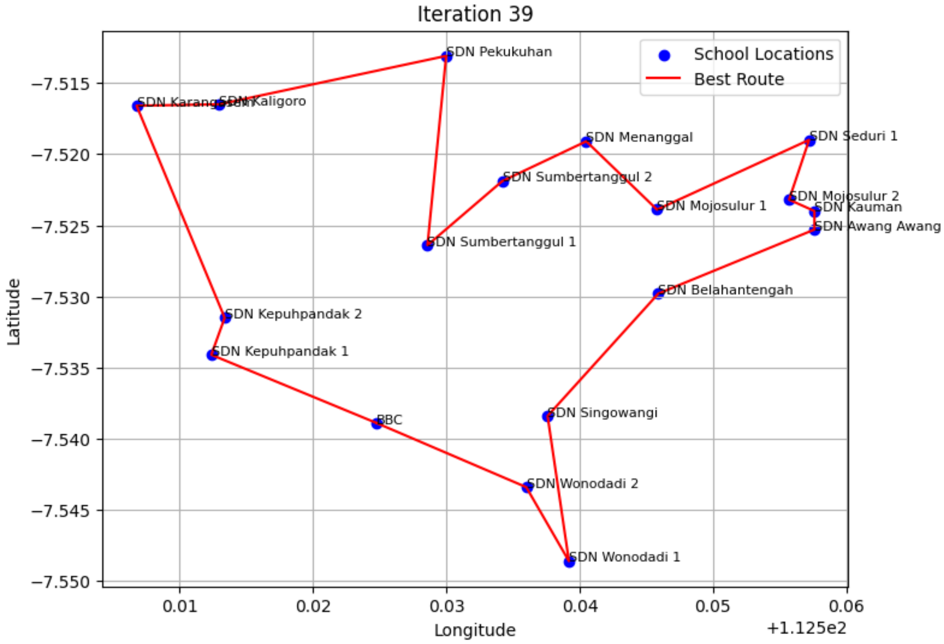


```
        for i in range(len(route) - 1):
            distance += self.distances[route[i]][route[i + 1]]
        # Tambahkan jarak dari titik akhir kembali ke titik awal (BBC)
        distance += self.distances[route[-1]][route[0]]
        return distance

    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.school[node] for node in
best_ant_route if node < len(self.school)]

    def convert_distance_to_km(self, distance):
        # Approximate conversion factor
        km_per_degree = 111.32
        return distance * km_per_degree

if __name__ == "__main__":
    school = [
        "SDN Kaligoro","SDN Karangasem","SDN Kepuhpandak 1","SDN
Kepuhpandak 2","SDN Singowangi","SDN Wonodadi 1","SDN Wonodadi 2",
        "SDN Belahantengah","SDN Mojosulur 1","SDN Mojosulur 2","SDN
Menanggal","SDN Kauman","SDN Awang Awang","SDN Sumbertanggul 1",
        "SDN Sumbertanggul 2","SDN Pekukuhan","SDN Seduri 1", "BBC"
    ]

    coordinates = np.array([
        [-7.5165, 112.513],[-7.5166, 112.5068],[-7.5341, 112.5124],[-
7.5315, 112.5134],[-7.5384, 112.5376],[-7.5486, 112.5392],[-7.5434,
112.536],
        [-7.5298, 112.5459],[-7.5239, 112.5458],[-7.5232, 112.5557],[-
7.5191, 112.5405],[-7.524, 112.5576],[-7.5253, 112.5576],[-7.5264,
112.5286],[-7.5219, 112.5342],
        [-7.5131, 112.53],[-7.519, 112.5572], [-7.5389, 112.5248]
    ])

    num_ants = 25
    num_iterations = 40
    pheromone_evaporation_rate = 0.05
    alpha = 1
    beta = 2
    min_pheromone = 0.1
    max_pheromone = 10

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





```
Iteration: 0 | Best Distance: 0.21245572259904025 degrees
Iteration: 10 | Best Distance: 0.16373635017201277 degrees
Iteration: 20 | Best Distance: 0.16373635017201277 degrees
Iteration: 30 | Best Distance: 0.16373635017201277 degrees
```



Best Distance: 0.16373635017201277 degrees

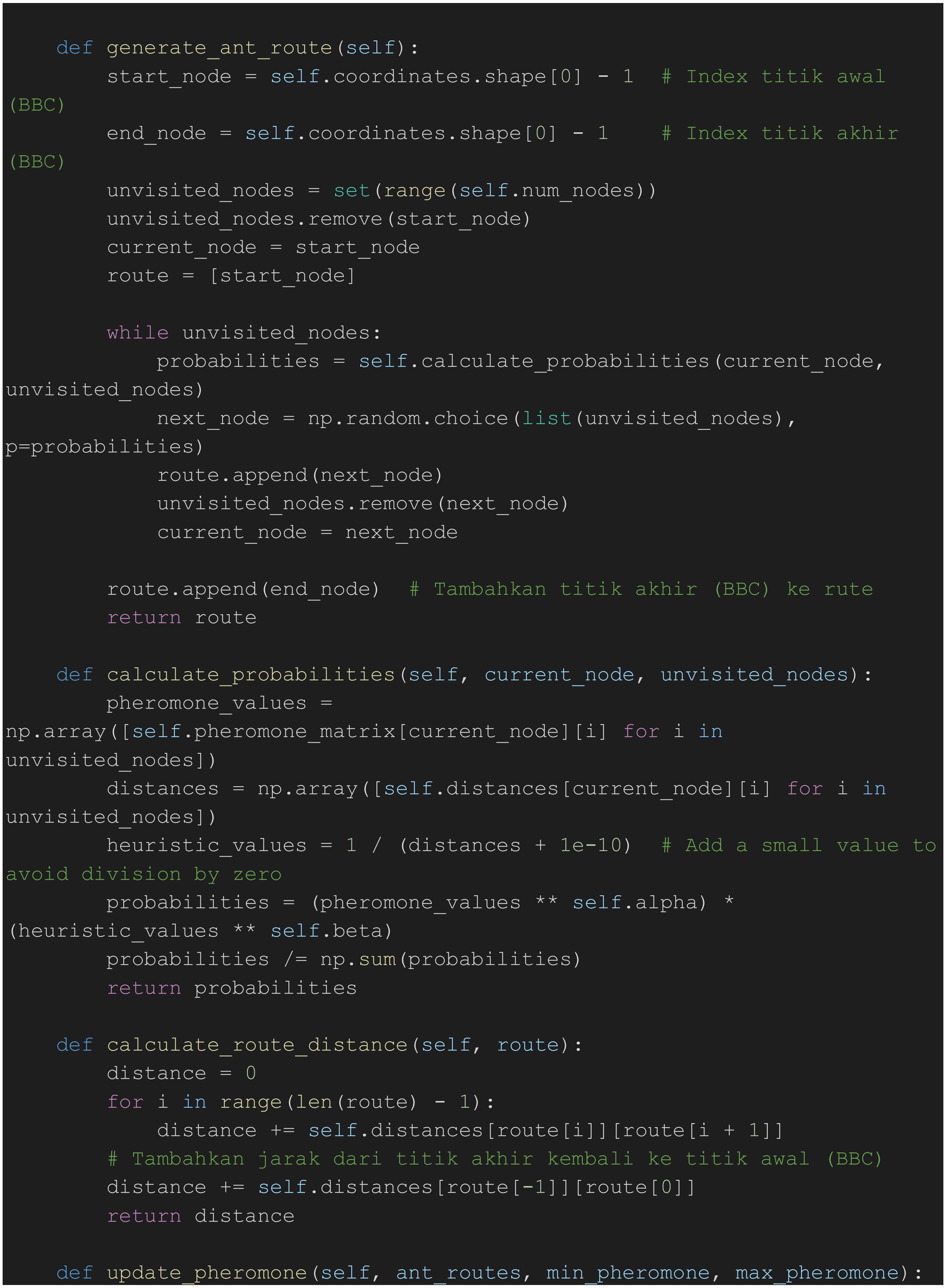
Best Distance: 18.22713050114846 km

Best Route: ['BBC', 'SDN Wonodadi 2', 'SDN Wonodadi 1', 'SDN Singowangi', 'SDN Belahantengah', 'SDN Awang Awang', 'SDN Kauman', 'SDN Mojosulur 2', 'SDN Seduri 1', 'SDN Mojosulur 1', 'SDN Menanggal', 'SDN Sumbertanggul 2', 'SDN Sumbertanggul 1', 'SDN Pekukuhan', 'SDN Kaligoro', 'SDN Karangasem', 'SDN Kepuhpandak 2', 'SDN Kepuhpandak 1', 'BBC']

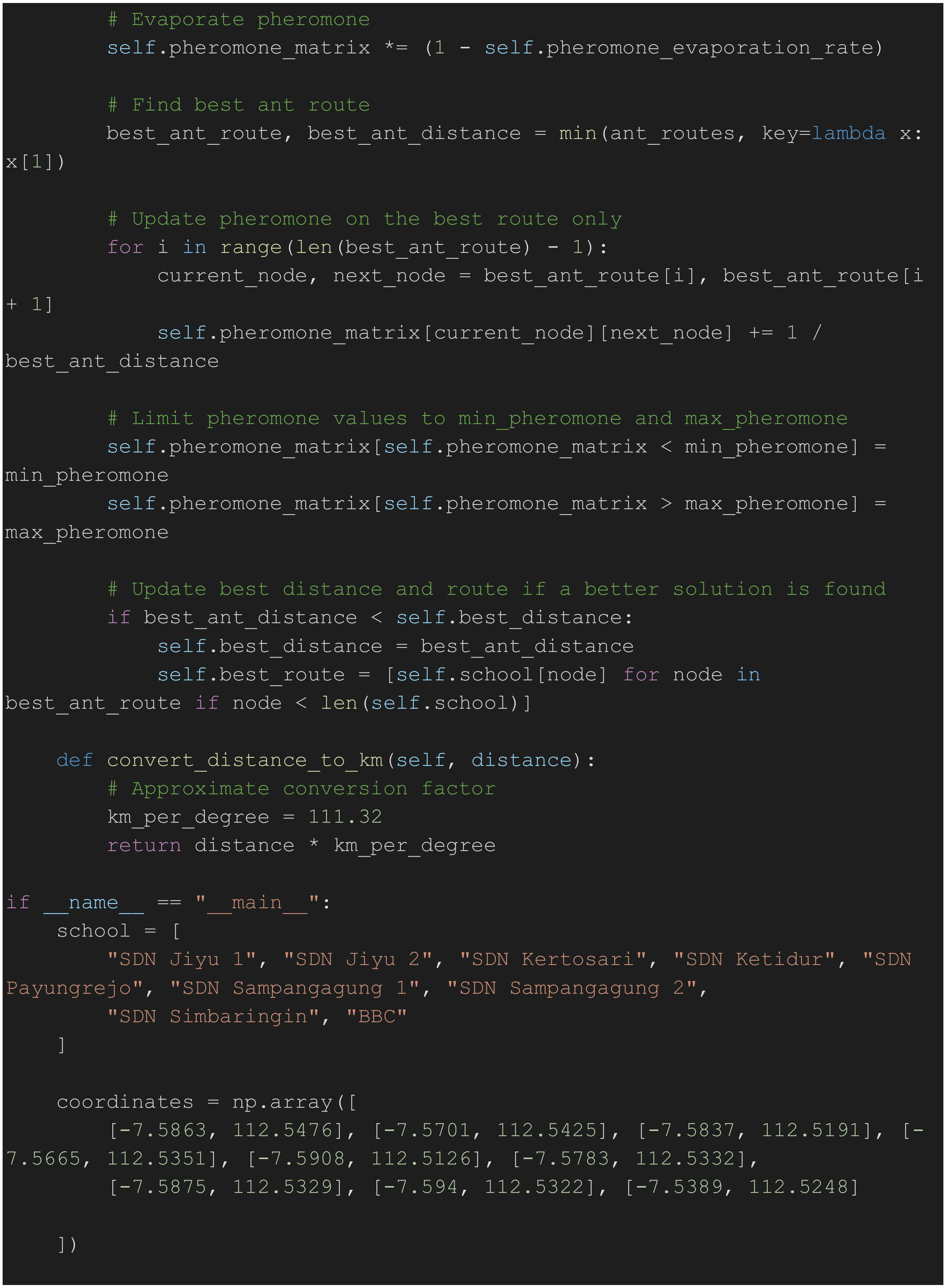
Total Runtime: 1.9592633247375488 seconds

CLUSTER 2

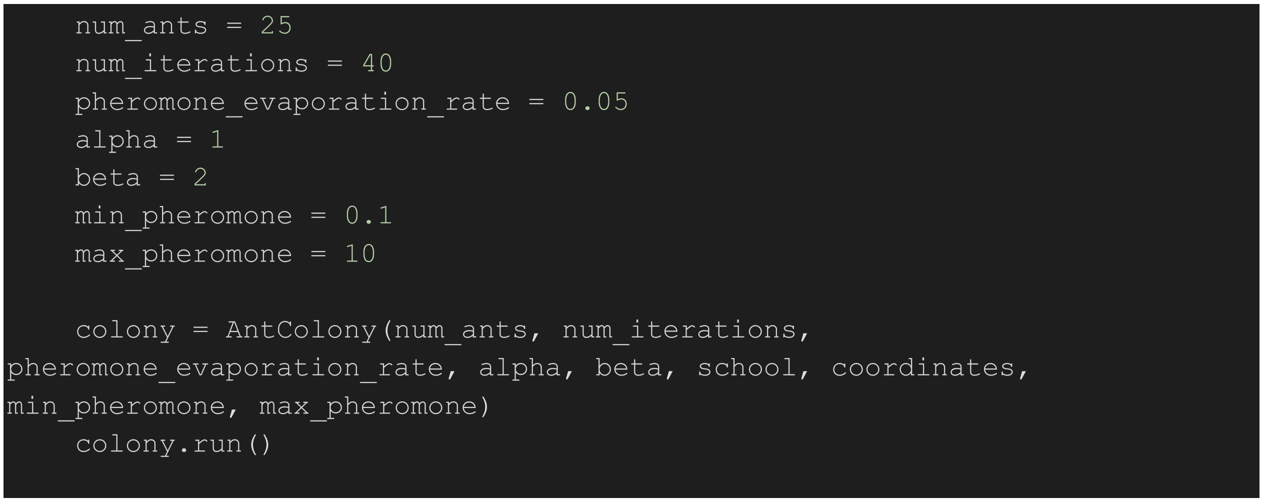
```
import numpy as np
import matplotlib.pyplot as plt
import time

class AntColony:
    def __init__(self, num_ants, num_iterations,
pheromone_evaporation_rate, alpha, beta, school, 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.school = school
        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, "degrees")

        print("Best Distance:", self.best_distance, "degrees")
        print("Best Distance:",
self.convert_distance_to_km(self.best_distance), "km")
        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='School 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')

        # Tambahkan label nama sekolah di setiap titik koordinat sekolah
        for i, coord in enumerate(self.coordinates):
            school_name = self.school[i]
            plt.text(coord[1], coord[0], school_name, fontsize=8)

        plt.title(f"Iteration {iteration}")
        plt.xlabel("Longitude")
        plt.ylabel("Latitude")
        plt.legend()
        plt.grid(True)
        plt.show()
```



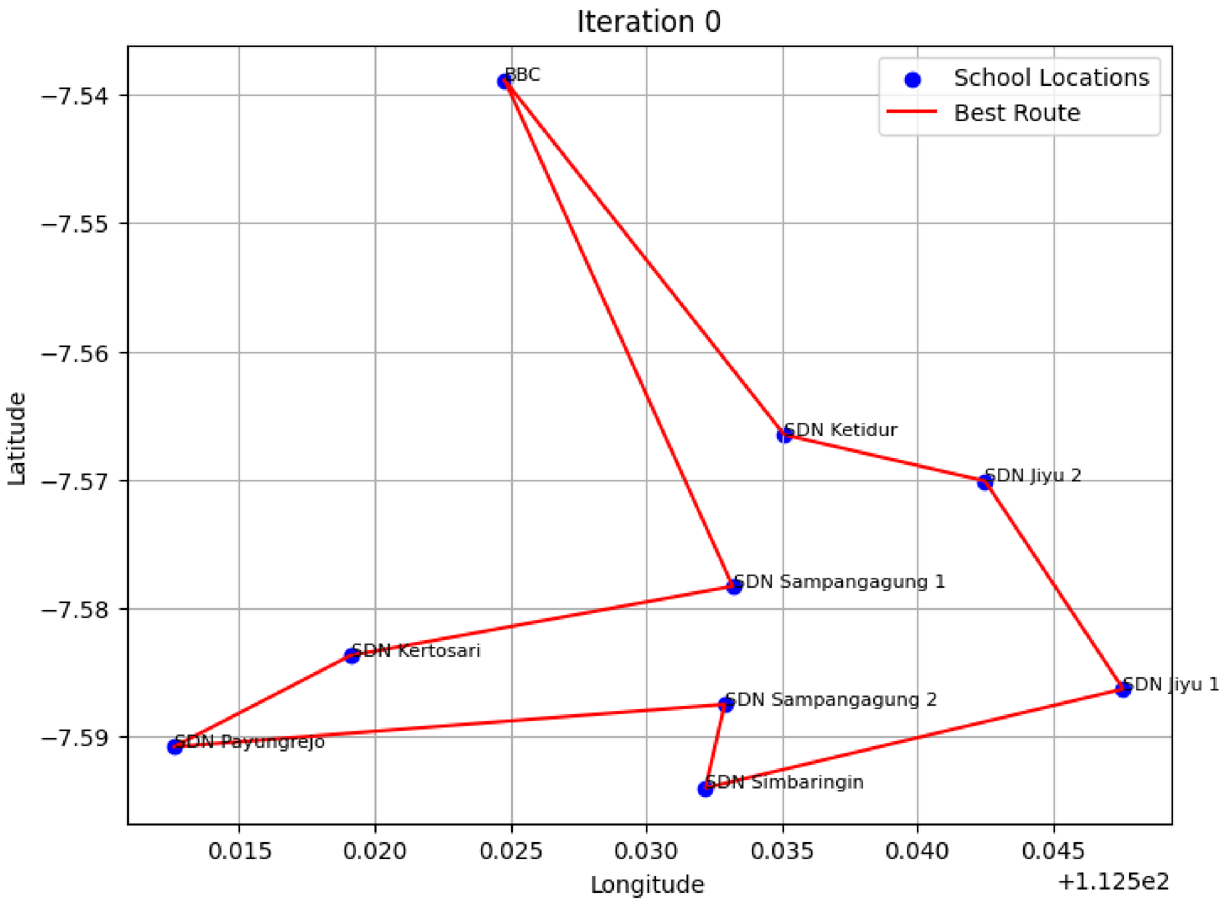
```
    def generate_ant_route(self):
        start_node = self.coordinates.shape[0] - 1  # Index titik awal
(BBC)
        end_node = self.coordinates.shape[0] - 1    # Index titik akhir
(BBC)
        unvisited_nodes = set(range(self.num_nodes))
        unvisited_nodes.remove(start_node)
        current_node = 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(end_node)  # Tambahkan titik akhir (BBC) ke rute
        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]]
        # Tambahkan jarak dari titik akhir kembali ke titik awal (BBC)
        distance += self.distances[route[-1]][route[0]]
        return distance

    def update_pheromone(self, ant_routes, min_pheromone, max_pheromone):
```

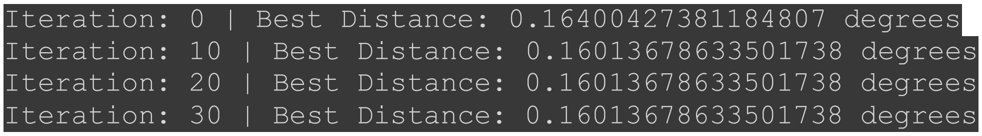


```
        # Evaporate pheromone
        self.pheromone_matrix *= (1 - self.pheromone_evaporation_rate)

        # 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.school[node] for node in best_ant_route if node < len(self.school)]

    def convert_distance_to_km(self, distance):
        # Approximate conversion factor
        km_per_degree = 111.32
        return distance * km_per_degree

if __name__ == "__main__":
    school = [
        "SDN Jiyu 1", "SDN Jiyu 2", "SDN Kertosari", "SDN Ketidur", "SDN Payungrejo", "SDN Sampangagung 1", "SDN Sampangagung 2",
        "SDN Simbaringin", "BBC"
    ]

    coordinates = np.array([
        [-7.5863, 112.5476], [-7.5701, 112.5425], [-7.5837, 112.5191], [-7.5665, 112.5351], [-7.5908, 112.5126], [-7.5783, 112.5332],
        [-7.5875, 112.5329], [-7.594, 112.5322], [-7.5389, 112.5248]

    ])
```

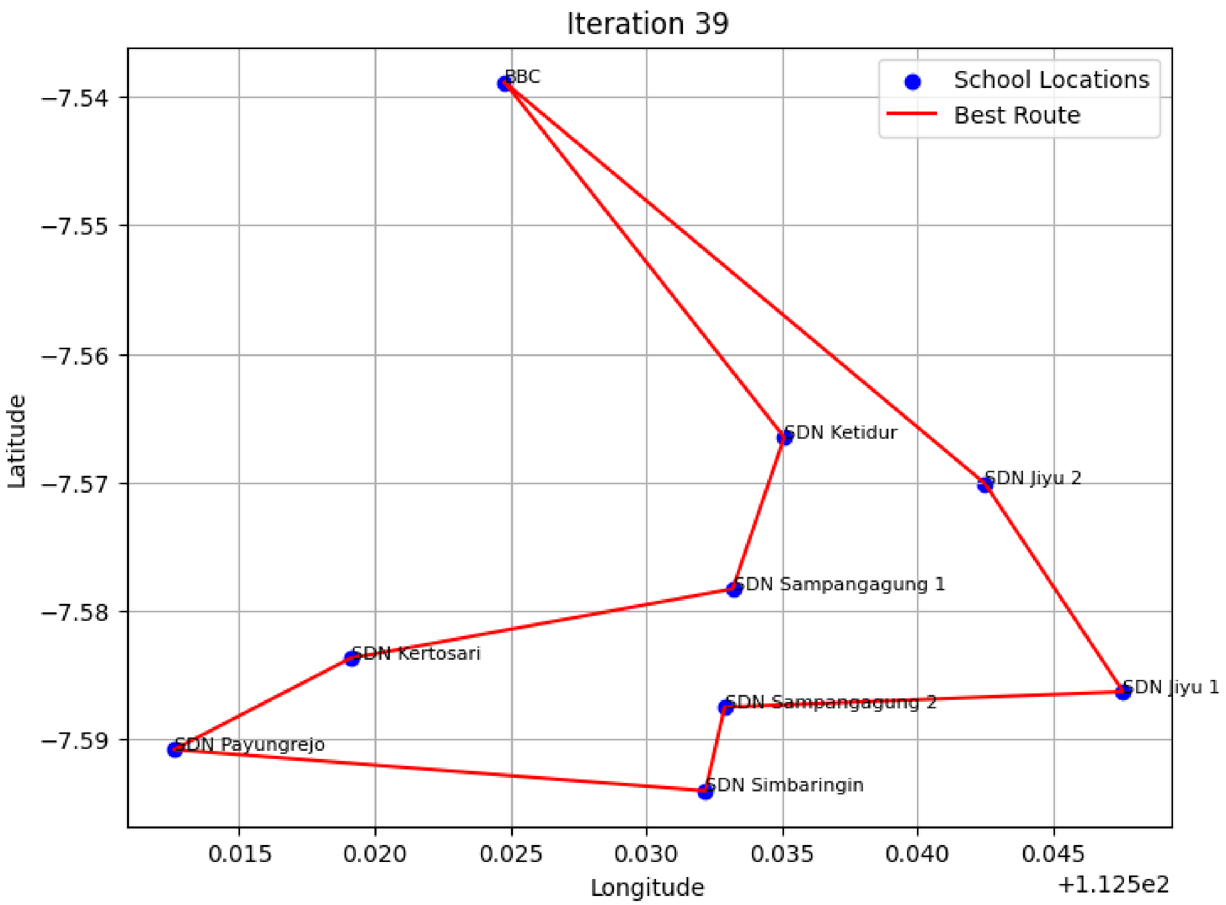


```
    num_ants = 25
    num_iterations = 40
    pheromone_evaporation_rate = 0.05
    alpha = 1
    beta = 2
    min_pheromone = 0.1
    max_pheromone = 10

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





```
Iteration: 0 | Best Distance: 0.16400427381184807 degrees
Iteration: 10 | Best Distance: 0.16013678633501738 degrees
Iteration: 20 | Best Distance: 0.16013678633501738 degrees
Iteration: 30 | Best Distance: 0.16013678633501738 degrees
```



Best Distance: 0.16013678633501738 degrees

Best Distance: 17.826427054814133 km

Best Route: ['BBC', 'SDN Jiyu 2', 'SDN Jiyu 1', 'SDN Sampangagung 2', 'SDN Simbaringin', 'SDN Payungrejo', 'SDN Kertosari', 'SDN Sampangagung 1', 'SDN Ketidur', 'BBC']

Total Runtime: 1.2620720863342285 seconds