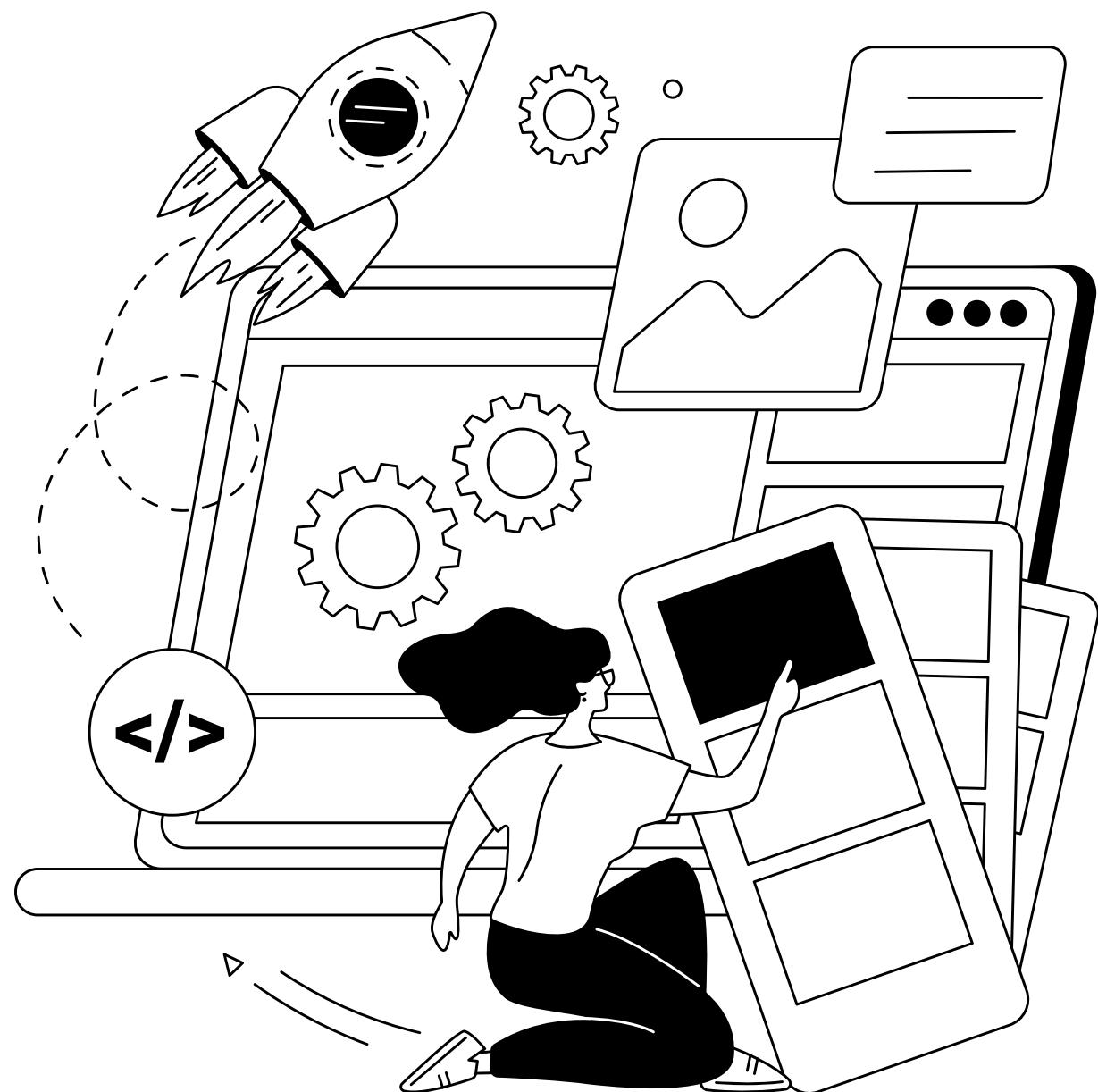


NATURE-INSPIRED  
OPTIMIZATION



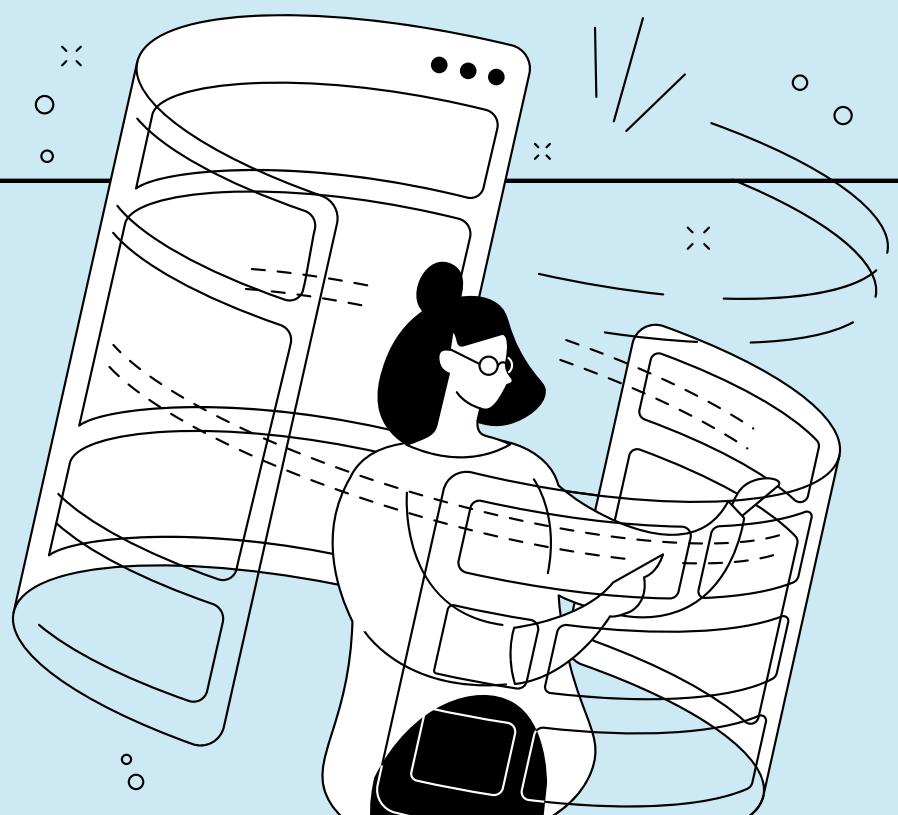
Frida Putriassa

# Implementasi Metode TSP pada Rute Pariwisata Menggunakan Algoritma Ant Colony Optimization

Studi Kasus :  
**Pariwisata Kota Cirebon**

# Content

Berikut merupakan beberapa bahasan terkait topik yang diambil.



## ↓ PENDAHULUAN

Berisi latar belakang dan penjelasan mengenai pariwisata dan destinasi wisata yang ada di Kota Cirebon.

## ↓ ALGORITMA ACO

Terdiri dari penentuan parameter dan matriks, implementasi menggunakan python dan representasi graph.

## ↓ KESIMPULAN

Berisi kesimpulan dan saran

# Pendahuluan

Pariwisata menjadi sektor yang semakin penting dalam pengembangan suatu daerah, tidak hanya dari segi ekonomi, tetapi juga dari aspek budaya dan sosial. Efisiensi dalam menentukan rute pariwisata menjadi krusial untuk memberi pengalaman terbaik bagi wisatawan. Salah satu tantangan dalam hal ini adalah Travelling Salesman Problem (TSP), yang mengharuskan penentuan rute terpendek untuk mengunjungi sejumlah titik tertentu.





# Potensi Pariwisata Kota Cirebon

Kota Cirebon kaya akan warisan budaya dan sejarah, menawarkan sejumlah destinasi pariwisata yang menarik. Dengan adanya Nasi Jamblang dan Empal Gentong Ibu Nur, Alun-alun Kota Cirebon, Keraton Kasepuhan, Taman Sari Goa Sunyaragi, Gedung BAT, dan Kawasan Batik Trusmi, potensi wisata Kota Cirebon perlu dioptimalkan. Penentuan rute yang efisien dapat meningkatkan daya tarik dan kenyamanan wisatawan.

# Wisata Kota Cirebon



NASI JAMBLANG DAN  
EMPAL GENTONG IBU NUR



ALUN-ALUN KOTA  
CIREBON



KERATON KASEPUHAN  
CIREBON

# Wisata Kota Cirebon



TAMAN SARI GOA  
SUNYARAGI



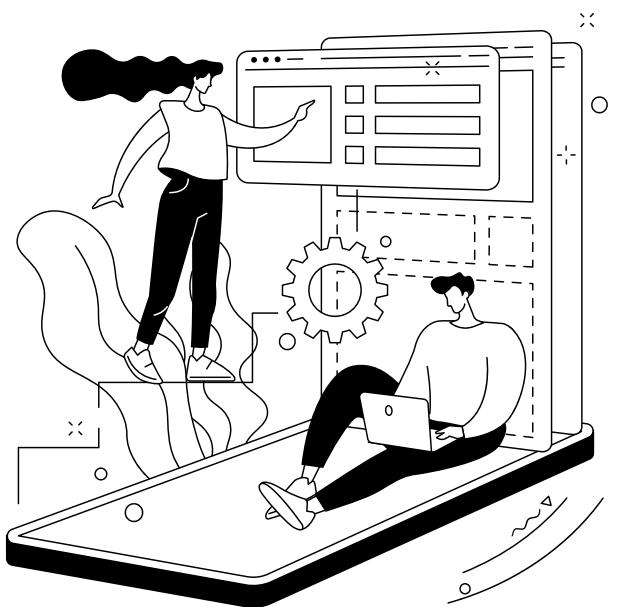
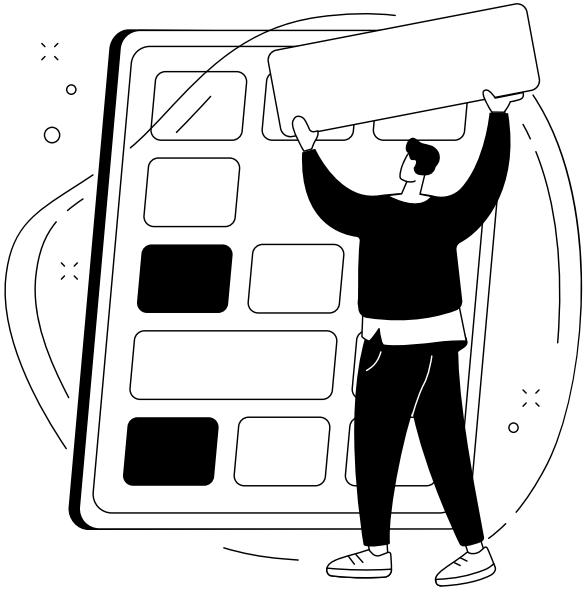
GEDUNG BRITISH  
AMERICAN TOBACCO



KERKAWASAN BATIK  
TRUSMIATON KASEPUHAN

# Apa itu Algoritma ACO?

Algoritma ACO, terinspirasi dari perilaku semut dalam mencari rute terpendek terbukti efektif dalam menyelesaikan TSP. Keunggulannya adalah menemukan solusi yang mendekati optimal dengan memanfaatkan jejak feromon dan kombinasi faktor probabilitas. Oleh karena itu, ACO menjadi pilihan yang tepat untuk mengatasi kompleksitas permasalahan penentuan rute pariwisata, seperti di Kota Cirebon.



# Penentuan Parameter

num\_iterations = 100

num\_ants = 10

alpha = 1

beta = 2

evaporation\_rate = 0.5



## ● ALPHA DAN BETA

Alpha menentukan pengaruh pheromone pada pemilihan rute. Beta Mengontrol seberapa besar pengaruh visibility (kebalikan jarak) pada pemilihan rute.

## ● EVAPORATION RATE

Mengontrol laju pengurangan pheromone pada jalur seiring waktu. Pengaturan evaporation rate memastikan pembaruan berkelanjutan dan menjaga keseimbangan antara eksplorasi dan eksplorasi rute terbaik.



# Matriks Jarak

## MATRIKS JARAK

[	0	1	3.4	4.1	2.6	6.2	]
[	1	0	3.3	4.7	1.9	6.4	]
[	3.4	3.3	0	5.5	1.6	9.7	]
[	4.1	4.7	5.5	0	7.5	6.9	]
[	2.6	1.9	1.6	7.5	0	8.3	]
[	6.2	6.4	9.7	6.9	8.3	0	]

## KETERANGAN

- A = Nasi Jamblang dan Empal Gentong Ibu Nur
- B = Alun-alun Kota Cirebon
- C = Keraton Kasepuhan
- D = Taman Sari Goa Sunyaragi
- E = Gedung BAT
- F = Kawasan Batik Trusmi

# IMPLEMENTASI ALGORITMA ACO

```
[1] import numpy as np
    import random

[2] def initialize_pheromone_matrix(num_wahana, initial_pheromone=1.0):
    return np.ones((num_wahana, num_wahana)) * initial_pheromone

[3] def calculate_probabilities(pheromones, visibility, alpha, beta):
    return (pheromones ** alpha) * (visibility ** beta)

[4] def select_next_wahana(pheromones, visibility, alpha, beta, visited_wahana):
    probabilities = calculate_probabilities(pheromones, visibility, alpha, beta)
    not_visited = [wahana for wahana in range(len(probabilities)) if wahana not in visited_wahana]

    # Membuat probabilities hanya untuk wahana yang belum dikunjungi
    probabilities_for_not_visited = probabilities[visited_wahana[-1], not_visited]

    # Normalisasi probabilitas hanya untuk wahana yang belum dikunjungi
    probabilities_for_not_visited /= probabilities_for_not_visited.sum()

    # Mengambil langkah berikutnya
    selected_wahana = np.random.choice(not_visited, p=probabilities_for_not_visited)

    return selected_wahana
```

# IMPLEMENTASI ALGORITMA ACO

```
[5]  def update_pheromones(pheromones, visited_wahana, path_length, evaporation_rate=0.5, pheromone_deposit=1.0):
0s    for i in range(len(visited_wahana) - 1):
        pheromones[visited_wahana[i], visited_wahana[i + 1]] *= (1.0 - evaporation_rate)
        pheromones[visited_wahana[i + 1], visited_wahana[i]] = pheromones[visited_wahana[i], visited_wahana[i + 1]]

        pheromones[visited_wahana[-1], visited_wahana[0]] *= (1.0 - evaporation_rate)
        pheromones[visited_wahana[0], visited_wahana[-1]] = pheromones[visited_wahana[-1], visited_wahana[0]]

    deposit = pheromone_deposit / path_length
    for i in range(len(visited_wahana) - 1):
        pheromones[visited_wahana[i], visited_wahana[i + 1]] += deposit
        pheromones[visited_wahana[i + 1], visited_wahana[i]] = pheromones[visited_wahana[i], visited_wahana[i + 1]]

    pheromones[visited_wahana[-1], visited_wahana[0]] += deposit
    pheromones[visited_wahana[0], visited_wahana[-1]] = pheromones[visited_wahana[-1], visited_wahana[0]]
```

# IMPLEMENTASI ALGORITMA ACO

```
[6]  def ant_colony_optimization(num_iterations, num_ants, pheromones, visibility, alpha, beta, evaporation_rate):
    num_wahana = len(pheromones)
    best_path = None
    best_path_length = float('inf')

    for iteration in range(num_iterations):
        for ant in range(num_ants):
            visited_wahana = [0] # Mulai dari wahana 0
            current_wahana = 0

            while len(visited_wahana) < num_wahana:
                next_wahana = select_next_wahana(pheromones, visibility, alpha, beta, visited_wahana)
                visited_wahana.append(next_wahana)
                current_wahana = next_wahana

            path_length = calculate_path_length(visited_wahana, pheromones)

            if path_length < best_path_length:
                best_path_length = path_length
                best_path = visited_wahana[:]

            update_pheromones(pheromones, visited_wahana, path_length, evaporation_rate)

    return best_path, best_path_length
```

# IMPLEMENTASI ALGORITMA ACO

```
✓ 0s [7] def calculate_path_length(path, pheromones):
      length = 0
      for i in range(len(path) - 1):
          length += pheromones[path[i], path[i + 1]]
      length += pheromones[path[-1], path[0]] # Kembali ke awal
      return length

✓ 0s [8] # Matriks Jarak Antar Wisata di Kota Cirebon
matriks_jarak = np.array([
    [0, 1, 3.4, 4.1, 2.6, 6.2],
    [1, 0, 3.3, 4.7, 1.9, 6.4],
    [3.4, 3.3, 0, 5.5, 1.6, 9.7],
    [4.1, 4.7, 5.5, 0, 7.5, 6.9],
    [2.6, 1.9, 1.6, 7.5, 0, 8.3],
    [6.2, 6.4, 9.7, 6.9, 8.3, 0]
])

✓ 0s [9] # Inisialisasi Pheromone Matrix dan Visibility Matrix
pheromones = initialize_pheromone_matrix(len(matriks_jarak))
visibility = 1 / matriks_jarak

<ipython-input-9-9ddd68d22ffd>:3: RuntimeWarning: divide by zero encountered in divide
    visibility = 1 / matriks_jarak
```

# IMPLEMENTASI ALGORITMA ACO

```
✓ 0s # Parameter ACO
    num_iterations = 100
    num_ants = 10
    alpha = 1
    beta = 2
    evaporation_rate = 0.5

✓ 0s [11] # Menjalankan Algoritma ACO
    best_path, best_path_length = ant_colony_optimization(num_iterations, num_ants, pheromones, visibility, alpha, beta, evaporation_rate)

✓ 0s [12] print("Best Path:", best_path)
    print("Best Path Length:", best_path_length)

    Best Path: [0, 1, 4, 2, 5, 3]
    Best Path Length: 3.3271479571304168
```

# IMPLEMENTASI ALGORITMA ACO

```
▶ import matplotlib.pyplot as plt

def plot_graph(matriks_jarak, best_path):
    # Menampilkan posisi wisata sebagai node
    for i, node_index in enumerate(best_path):
        node_name = chr(ord('A') + node_index) # Mengubah indeks node menjadi huruf A, B, C, ...
        plt.text(matriks_jarak[node_index, 1], matriks_jarak[node_index, 2], node_name,
                 fontsize=12, ha='center', va='center', color='white', fontweight='bold')
        plt.scatter(matriks_jarak[node_index, 1], matriks_jarak[node_index, 2],
                    s=300, color='blue', edgecolors='black', linewidth=2)

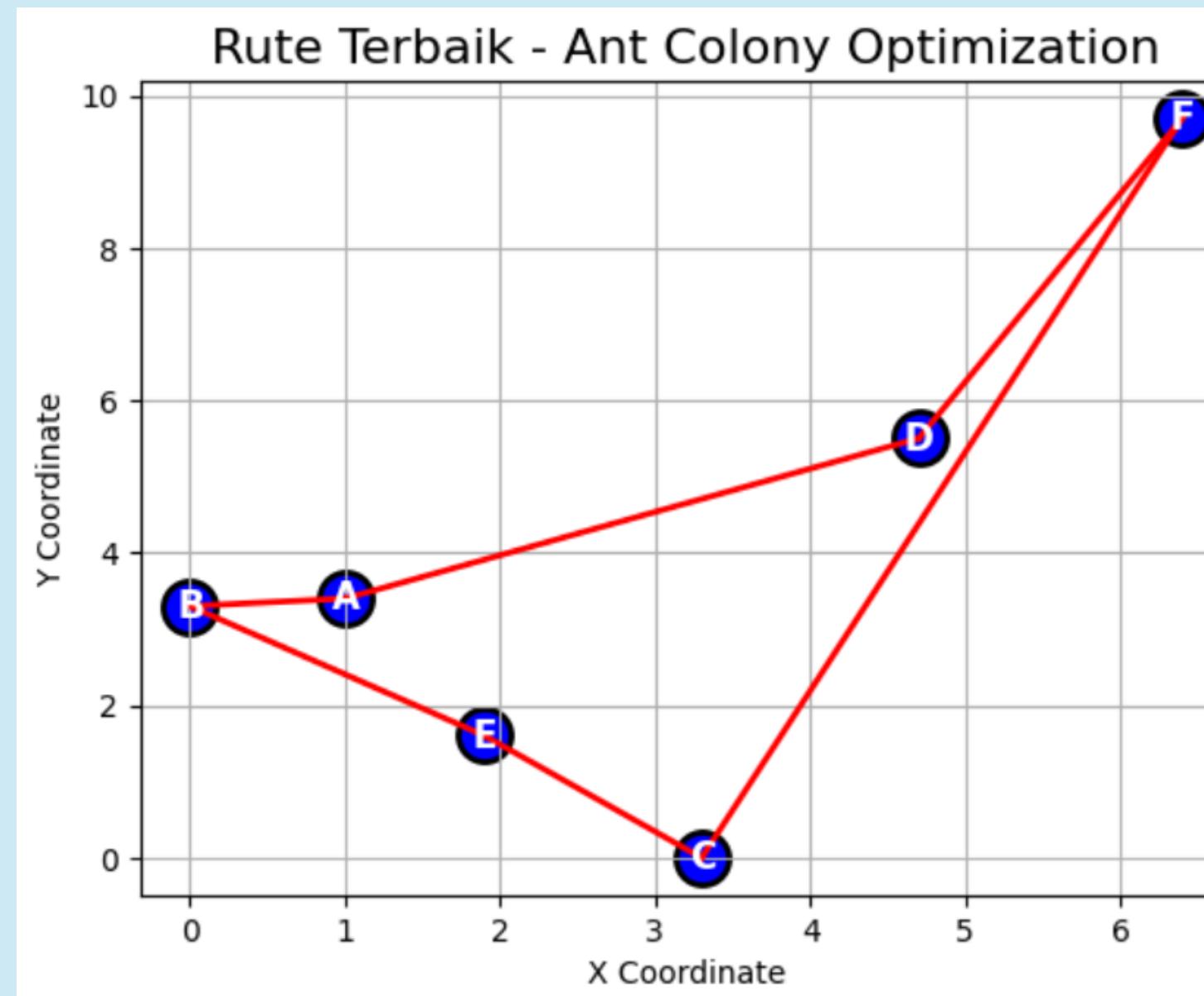
    # Menghubungkan node sesuai dengan rute terbaik
    for i in range(len(best_path) - 1):
        plt.plot([matriks_jarak[best_path[i], 1], matriks_jarak[best_path[i + 1], 1]],
                 [matriks_jarak[best_path[i], 2], matriks_jarak[best_path[i + 1], 2]],
                 color='red', linewidth=2)

    # Menghubungkan node terakhir dengan node pertama
    plt.plot([matriks_jarak[best_path[-1], 1], matriks_jarak[best_path[0], 1]],
             [matriks_jarak[best_path[-1], 2], matriks_jarak[best_path[0], 2]],
             color='red', linewidth=2)

plt.title('Rute Terbaik - Ant Colony Optimization', fontsize=16)
plt.xlabel('X Coordinate')
plt.ylabel('Y Coordinate')
plt.grid(True)
plt.show()
```

# REPRESENTASI GRAPH

```
✓ [15] # Menampilkan grafik rute terbaik  
plot_graph(matriks_jarak, best_path)
```



# Kesimpulan

Dalam implementasi Metode Traveling Salesman Problem (TSP) pada rute pariwisata Kota Cirebon menggunakan Algoritma Ant Colony Optimization (ACO), hasil terbaik yang ditemukan adalah rute [A, B, E, C, F, D] dengan panjang rute 3.33 unit. Algoritma ACO membuktikan kemampuannya dalam mencari rute terpendek dengan menjelajahi berbagai kemungkinan, memberikan solusi yang efisien untuk mengeksplorasi keindahan pariwisata Kota Cirebon. Hasil ini menggambarkan potensi ACO sebagai alat yang efektif untuk mengoptimalkan pengalaman wisata dengan menentukan rute terbaik secara otomatis.