CMSC 170 | Introduction to Artificial Intelligence

Instructor: Prof. Jaderick Pabico

Scheduling Aircraft Landings Problem

John Rommel B. Octavo
CMSC 170 | Section ST-1L

Term Project Report

Aircraft Landing Scheduling Problem

The Ant Colony Optimization (ACO) was utilized to find the optimal solution for the Aircraft Landing Scheduling Problem (ALSP). Python programming language was used to code the algorithm. The problem presentation, solution construction, pheromone trail, local search, decision rule, pheromone update, iterations, and heuristics are all patterned to extract the near-optimal solution of the ALSP.

The problem presentation of the ALSP uses the list of dictionaries in Python to correctly represent the variables that are needed in the problem. Each dictionary has value 'A' for the appearance time, 'E' for the earliest landing time, 'T' for the target landing time, 'L' for the latest landing time, 'G' for the penalty per unit time if landing is scheduled before the target landing time, 'H' for the penalty per unit time if the landing is scheduled after the target landing time, and 'S' are for the list of separation time values of the aircraft to the other aircraft. All of the information stated is seen in Figure 1.

```
#=============================================================================================================
aircraft = [
    {'A': 54, 'E': 129, 'T': 155, 'L': 559, 'G': 10.00, 'H': 10.00, 'S': [99999, 3, 15, 15, 15, 15, 15, 15, 15, 15]},
    {'A': 120, 'E': 195, 'T': 258, 'L': 744, 'G': 10.00, 'H': 10.00, 'S': [3, 99999, 15, 15, 15, 15, 15, 15, 15, 15]},
    {'A': 14, 'E': 89, 'T': 98, 'L': 510, 'G': 30.00, 'H': 30.00, 'S': [15, 15, 99999, 8, 8, 8, 8, 8, 8, 8]},
    {'A': 21, 'E': 96, 'T': 106, 'L': 521, 'G': 30.00, 'H': 30.00, 'S': [15, 15, 8, 99999, 8, 8, 8, 8, 8, 8]},
    {'A': 35, 'E': 110, 'T': 123, 'L': 555, 'G': 30.00, 'H': 30.00, 'S': [15, 15, 8, 8, 99999, 8, 8, 8, 8, 8]},
    {'A': 45, 'E': 120, 'T': 135, 'L': 576, 'G': 30.00, 'H': 30.00, 'S': [15, 15, 8, 8, 8, 99999, 8, 8, 8, 8]},
    {'A': 49, 'E': 124, 'T': 138, 'L': 577, 'G': 30.00, 'H': 30.00, 'S': [15, 15, 8, 8, 8, 8, 99999, 8, 8, 8]},
    {'A': 51, 'E': 126, 'T': 140, 'L': 573, 'G': 30.00, 'H': 30.00, 'S': [15, 15, 8, 8, 8, 8, 8, 99999, 8, 8]},
    {'A': 60, 'E': 135, 'T': 150, 'L': 591, 'G': 30.00, 'H': 30.00, 'S': [15, 15, 8, 8, 8, 8, 8, 8, 99999, 8]},
    {'A': 85, 'E': 160, 'T': 180, 'L': 657, 'G': 30.00, 'H': 30.00, 'S': [15, 15, 8, 8, 8, 8, 8, 8, 8, 99999]}
]
#=============================================================================================================
```

Figure 1. Problem Presentation for Aircraft

In the solution construction, the algorithm is primarily based on the separation time and pheromone trail in determining the probability for each aircraft. These two factors affect the creation of solutions. Alpha and beta values are used to ensure reliable data from the calculation. The alpha controls the influence of the pheromone trail that is essential in the decision-making process on traversing the path to be taken. Beta on the other hand controls the weight in considering the separation time over the pheromone trail. Other variables that control the solution-making are rho and sigma. Rho represents the evaporation rate while sigma is the probability threshold for the ants to choose between exploitation and exploration. Figure 2 showcases the initial value of these control variables in the problem.

```
num_ants = 10
num_iterations = 100
alpha = 1
beta = 2
rho = 0.1
sigma = 0.9
```

Figure 2. Initial value of the control variables

The program imports a random module that will be used to generate random values for the program. It also uses the time module to measure the execution time of the program. In the program flow, the algorithm is composed of seven functions. The control variables are initialized in the main function of the program. They are passed as parameters to the primary function. The primary function that solves the optimal solution of ALSP is the solve_aircraft_landing(). Figure 3 shows the main function of the program and Figure 4 shows the primary function of the program.

```
num_ants = 10
num_iterations = 100
alpha = 1
beta = 2
rho = 0.1
sigma = 0.9

start_time = time.time()

best_solution, best_cost = solve_aircraft_landing(aircraft, num_ants, num_iterations, alpha, beta, rho, sigma)

end_time = time.time()
      You, 4 days ago • chore: added file for the term project
for i in range(len(best_solution)):
    best_solution[i] = best_solution[i] + 1

print('Best solution:', best_solution)
print('Best cost:', best_cost)

execution_time = end_time - start_time

print(f"Execution time: {execution_time} seconds")
```

Figure 3. Main function excluding the import statements and the aircraft initialization

```
def solve_aircraft_landing(aircrafts, num_ants, num_iterations, alpha, beta, rho, sigma):
    num_aircrafts = len(aircrafts)
    pheromone = [[1] * num_aircrafts for _ in range(num_aircrafts)]  # Initialize pheromone matrix
    separation_time = calculate_separation_time(aircrafts)  # Calculate separation time matrix
    best_solution = []  # Initialize variables to track best solution and cost
    best_cost = float('inf')

    for iteration in range(num_iterations):
        solutions = []  # Track solutions and costs for each ant in this iteration
        costs = []

        for ant in range(num_ants):
            solution = construct_solution(aircrafts, pheromone, separation_time, alpha, beta, sigma)  # Construct ant solution
            cost = calculate_cost(solution, aircrafts)  # Calculate cost of the solution
            solutions.append(solution)  # Add solution and cost to the respective lists
            costs.append(cost)

            if cost < best_cost:  # Update best solution if the current solution is better
                best_solution = solution
                best_cost = cost

        pheromone = update_pheromone(pheromone, solutions, costs, rho)  # Update pheromone matrix

    return best_solution, best_cost
```

Figure 4. The primary function of the program

The primary function of the program, solve_aircraft_landing() function, initializes the pheromone trail, separation time, best_solution, and best_cost. Inside the loop, the solutions and the cost are extracted. In each iteration, it makes sure that it stores the solution with the lowest cost as the best solution. Some lines in the code call another function to perform specific tasks. These were calculate_separation_time() which is responsible for extracting the separation time and storing it on a matrix, construct_solution() which is responsible for extracting solution based on the pheromone trail and the heuristic of the program, calculate_cost() which is responsible for calculating the cost of the possible solutions, and update_pheromomone() which is responsible for adding and pheromone for the taken path or solution and also responsible for the evaporation of the pheromone. The primary function returns the best solution and the best cost.

The second function is the construct_solution(). The function randomly selects an aircraft to begin the search for the best solution. Each unvisited aircraft are noted. While not all aircraft are visited, the probability of each unvisited aircraft being visited is calculated by calling the calculate_probabilities() function. The decision on who the next aircraft is to be visited is determined by calling the choose_next_aircraft() function. The function returns the calculated solution. Figure 5 shows the code for the construct_solution() function.

```
def construct_solution(aircrafts, pheromone, separation_time, alpha, beta, sigma):
    num_aircrafts = len(aircrafts)
    solution = []  # Initialize solution list
    unvisited = list(range(num_aircrafts))  # List of unvisited aircrafts

    start_aircraft = random.choice(unvisited)  # Randomly choose a start aircraft
    solution.append(start_aircraft)  # Add the start aircraft to the solution
    unvisited.remove(start_aircraft)  # Remove start aircraft from unvisited list

    while unvisited:  # Continue until all aircrafts are visited
        current_aircraft = solution[-1]  # Get the last visited aircraft
        probabilities = calculate_probabilities(current_aircraft, unvisited, pheromone, separation_time, alpha, beta)  # Calculate probabili
        next_aircraft = choose_next_aircraft(probabilities, unvisited)  # Choose the next aircraft based on the probabilities
        solution.append(next_aircraft)  # Add the next aircraft to the solution
        unvisited.remove(next_aircraft)  # Remove the next aircraft from the unvisited list

    return solution
```

Figure 5. Function for constructing a solution to the ALSP

The third function is the calculate_probabilities(). In general, it is the calculation of the probability of selection of each unvisited aircraft. Firstly, the pheromone_sum and the probabilities variables are initialized. The loop that traverses all unvisited aircraft is used to calculate the probability of selection. The probability of selection is based on separation time and pheromone trail. As discussed earlier the alpha and beta control variables play an important role to make the calculation more reliable. Lastly, the function returns the list of probabilities. Figure 6 shows the code for the calculate_probabilities() function.

```python
def calculate_probabilities(current_aircraft, unvisited, pheromone, separation_time, alpha, beta):
    pheromone_sum = 0.0
    probabilities = []  # List to store probabilities for each unvisited aircraft

    for aircraft in unvisited:
        st = separation_time[current_aircraft][aircraft]  # Separation time between current and unvisited aircraft
        # 0.0000001 is added to avoid division by zero error in case the st is equal to zero
        visibility = 1 / (st + 0.0000001)  # Calculate visibility (inverse of separation time)      You, 1 second ago • Uncommitted
        pheromone_value = pheromone[current_aircraft][aircraft] ** alpha  # Pheromone value between current and unvisited aircraft
        heuristic = visibility ** beta  # Heuristic value (visibility) raised to the power of beta
        probabilities.append(pheromone_value * heuristic)  # Calculate probability as product of pheromone and heuristic values
        pheromone_sum += pheromone_value * heuristic  # Calculate sum of pheromone values for normalization

    probabilities = [p / pheromone_sum for p in probabilities]  # Normalize probabilities
    return probabilities
```

Figure 6. Code for the function to calculate the probabilities

The fourth function is the choose_next_aircraft(). Here the value is sigma is essential in determining whether the current solution will be patterned to the calculated probabilities or not. A random number is generated. If this random number is less than the value of sigma, the current solution will take the path according to what the probability dictates. Otherwise, the current solution will randomly take a path regardless of the calculated probability of its aim to explore new paths. Utilizing sigma here is important to avoid the interpreting local optimum as the best solution. The choose_next_aircraft() function returns the next aircraft. Figure 7 shows the code for the function.

```python
def choose_next_aircraft(probabilities, unvisited):
    if random.random() < sigma:  # Exploitation: Choose aircraft with maximum probability
        max_probability = max(probabilities)
        max_index = probabilities.index(max_probability)
        next_aircraft = unvisited[max_index]
    else:  # Exploration: Randomly choose aircraft based on probabilities
        next_aircraft = random.choices(unvisited, probabilities)[0]

    return next_aircraft
```

Figure 7. Function in choosing the next aircraft

The fifth function is the calculate_cost(). This function in general is for calculating the cost of the solution. The cost is incremented based on the penalty that is given each time the scheduled landing is earlier or later than the target landing. The penalty_before and penalty_after are imposed if the scheduled landing is earlier and later than the target time respectively. These two penalties will be the sole identifier of incurred cost per solution presented. The function returns the calculated total cost. Figure 8 shows the code for calculating the cost.

```python
def calculate_cost(solution, aircrafts):
    total_cost = 0.0

    for i in range(len(solution)):
        current_aircraft = solution[i]  # Get current aircraft
        current_time = sum([aircrafts[solution[j]]['S'][i] for j in range(i)])  # Calculate current time based on separation times
        appearance_time = aircrafts[current_aircraft]['A']  # Get appearance time of current aircraft
        earliest_time = aircrafts[current_aircraft]['E']  # Get earliest landing time of current aircraft
        target_time = aircrafts[current_aircraft]['T']  # Get target landing time of current aircraft
        latest_time = aircrafts[current_aircraft]['L']  # Get latest landing time of current aircraft
        penalty_before = aircrafts[current_aircraft]['G']  # Get penalty per unit time if landing before target time
        penalty_after = aircrafts[current_aircraft]['H']  # Get penalty per unit time if landing after target time

        if current_time < target_time:
            total_cost += (target_time - current_time) * penalty_before  # Add penalty if landing before target time
        elif current_time > target_time:
            total_cost += (current_time - target_time) * penalty_after  # Add penalty if landing after the target time

        current_time += aircrafts[current_aircraft]['S'][i]  # Update current time based on separation time

    return total_cost
```

Figure 8. Function to calculate the cost of the solution.

The sixth function is the update_pheromone() function. Firstly, the solution extracted will be the basis of adding a pheromone to the path or order of aircraft in the solution. The greater the cost, the smaller the amount of pheromone that will be added. The lesser the solution cost, the greater the amount of pheromone that will be added to the solution. This means that lesser-cost solutions are given more advantages than others. Next, the evaporation rate controlled by the variable rho is also executed. All pheromones in the trail or solution will be uniformly decreased depending on the evaporation rate specified. The function returns the pheromone value. Figure 9, shows the code for the update_pheromone() function.

```python
def update_pheromone(pheromone, solutions, costs, rho):
    num_aircrafts = len(pheromone)
    updated_pheromone = [[0] * num_aircrafts for _ in range(num_aircrafts)]  # Initialize updated pheromone matrix
    for solution, cost in zip(solutions, costs):
        for i in range(len(solution) - 1):
            current_aircraft = solution[i]  # Get current aircraft
            next_aircraft = solution[i + 1]  # Get next aircraft
            updated_pheromone[current_aircraft][next_aircraft] += 1 / (cost+0.0000001)  # Update pheromone based on the solution cost

    for i in range(num_aircrafts):
        for j in range(num_aircrafts):
            pheromone[i][j] = (1 - rho) * pheromone[i][j] + rho * updated_pheromone[i][j]  # Update pheromone with evaporation and deposit

    return pheromone
```

Figure 9. Update pheromone function

The last function is the calculate_separation_time(). This function is responsible for initializing the value of the separation time variable in the primary function. It extracts the list of separation time ‘S’ and stores it on the matrix. This 2d matrix is then passed as the return value of the function. Figure 10 shows the code for calculating the separation time.

```python
def calculate_separation_time(aircraft):
    num_aircrafts = len(aircraft)
    separation_time = [[0] * num_aircrafts for _ in range(num_aircrafts)]  # Initialize separation time matrix

    for i in range(num_aircrafts):
        for j in range(num_aircrafts):
            separation_time[i][j] = aircraft[i]['S'][j]  # Set separation time between different aircrafts

    return separation_time
```

Figure 10. Function to calculate separation time

The outputs of the sample runs of the algorithm are shown below:

For 10 Aircraft:

```
Best solution: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Best cost: 22280.0
Execution time: 0.04473519325256348 seconds
```

For 15 Aircraft:

```
Best solution: [1, 2, 14, 3, 4, 5, 6, 7, 8, 9, 10, 13, 11, 12, 15]
Best cost: 2016180.0
Execution time: 0.08410286903381348 seconds
```

For 20 Aircraft:

```
Best solution: [11, 19, 12, 20, 1, 4, 6, 8, 9, 10, 16, 18, 2, 3, 15, 14, 13, 7, 17, 5]
Best cost: 6024900.0
Execution time: 0.13750743865966797 seconds
```