ROUTE OPTIMIZATION FOR DELIVERY DRONES

Sayed Fazal

Department of Computer Science and Engineering

Saveetha School of Engineering

Sayedfazal1291.sse@saveetha.com

Abstract

Recently, a new parcel delivery method has been emerged, which involves Unmanned Aerial Vehicles, also known as drones, assisting traditional trucks in last-mile delivery across logistic networks. This method generally combines a truck and one drone or more to handle the delivery processes to customers. And this paper tends to calculate the optimal route of this drone-assisted delivery system, the objective is to minimize the operational time and energy cost. I developed a heuristic solution approach which implements an effective Genetic Algorithm to solve and simulate this delivery system of practical size, this program can automatically generate the on-screen best route results and show the total operational time and energy cost.

Key words: Dynamic Programming, Drone, Greedy Methods, Genetic Algorithm, Delivery System

Introduction

During the past few years, the commercial use of drones has been more frequent, and the technology for drones and the relative delivery network has also increased rapidly. Therefore, the necessity of studying the effectiveness and result of this drone-assisted delivery system is existential. This new distribution model is the use of a regular delivery truck that collaborates with drones to support parcel delivery. The drones have the advantages such as flexibility and speed which the trucks do not have. In addition, the long-range transportation ability and high capacity of trucks are not available in drones. Hence, there are complementary advantages when using drones to assist parcel delivery. Some companies are experimenting and examining the new technology of drones to support the mails and parcels delivery. Amazon Prime Air has planned to use multirotor Miniature UAV to automatically carry packages from Amazon order fulfilment centre to customers' locations within 30 minutes of ordering. And China's biggest internet retailer—Alibaba, said it had begun testing drone-based deliveries to hundreds of customers.

I have researched and investigated some related literature, in every logistics activity, operational costs and time play important roles in the overall business cost. Hence, minimizing these costs by optimizing min-time and min-energy drone-assisted delivery system (DADS) is a vital objective of every company involved in transport and logistics activities.

Figure 1. Amazon's Prime Air UAV and Alibaba's drone

I have researched and investigated some related literature, in every logistics activity, operational costs and time play important roles in the overall business cost. Hence, minimizing these costs by optimizing min-time and min-energy drone-assisted delivery system (DADS) is a vital objective of every company involved in transport and logistics activities.

Related Literature

I am aware of several related literature on the Traveling Salesman Problem (TSP) and Vehicle Routing Problem (VRP), while these publications are not very suitable for this new truck-drone tandem delivery system. We call this new distribution method the Traveling Salesman Problem with Drone (TSP-D). Perhaps there are different names and for this kind of problem. But in general, this problem aims to find an optimal route for both trucks and drones, that minimize the total joint time to complete the delivery tasks of all packages.

Solution approaches	Literature
Integer linear programming	Agatz et al. (2016); Murray & Chu (2015)
Approximation algorithms	Agatz et al. (2016)
Dynamic programming	Bouman et al. (2017)
Simple heuristics	Murray & Chu (2015); Ha & Deville et al. (2015)
Simulated annealing	Ponza (2016)
Genetic algorithms	Fernandez & Harbison et al. (2016)

Table 1. Solution approaches proposed for different literature

Table 1 summarizes the present solution procedures and approaches that have been studied for several slightly variants of the TSP-D.

Murray & Chu (2015) called the problem "Flying Sidekick Traveling Salesman Problem", which developed a mixed integer linear programming (MILP) formulation and a simple heuristic solution to solve this TSP D problem. They first find the best route for the truck as a typical TSP. Then, they run a relocation procedure which will check each node's possibility for drone service. Ponza (2016) extended Murray & Chu (2015)'s work by proposing a Simulated Annealing approach to solve the problem.

Agatz et al. (2016) presented a MILP formulation and developed several Route first-cluster second approximation heuristic algorithms to solve the TSP-D. Bouman et al. (2017) extended Agatz et al. (2016)'s study and raised an exact solution which formed on dynamic programming. Their work can resolve the problem of practical size.

Fernandez & Harbison et al. (2016) used K-means clustering and Genetic algorithm to investigate the effectiveness and compare the difference between the truck-drone tandem delivery system and stand alone truck delivery network.

Tasks:

1. Analyze the Time Complexity of Brute-Force Route Optimization:

- Brute-force involves evaluating all possible routes to find the optimal one. Since this is often a Traveling Salesman Problem (TSP), the time complexity is O(n!)O(n!)O(n!), which becomes infeasible as the number of deliveries grows.
- To analyze, calculate expected time complexity based on package and delivery variations.

2. Prove the Correctness of Shortest-Path and Scheduling Algorithms:

- For the shortest-path, proof may rely on Dijkstra's algorithm or A* for weighted paths, ensuring the path with minimum distance or time is selected.
- Scheduling correctness, especially with constraints like priority, can be shown using induction to verify that algorithms (e.g., Earliest Deadline First) maintain the correct order.

3. Implement Dynamic Programming for Route Adjustment Based on Weather Updates:

- Use dynamic programming to adjust existing routes as weather conditions change. This could involve breaking down the main route into sub-routes that are adjusted dynamically, using recursive or memorization techniques.

4. Use Greedy Methods for Real-Time Scheduling During Peak Hours:

- Greedy algorithms prioritize high-priority or nearby deliveries to optimize efficiency during high-demand periods. Evaluate factors like delivery priority and distance to ensure that decisions minimize fuel consumption and travel time.

5. Compare Polynomial and Non-Polynomial Models for Large-Scale Routing:

- Polynomial models (e.g., Dijkstra's algorithm) tend to scale better than non-polynomial (like brute-force TSP).
- For large-scale routing, consider approximations such as the Nearest Neighbour or Genetic Algorithm to balance accuracy and efficiency.

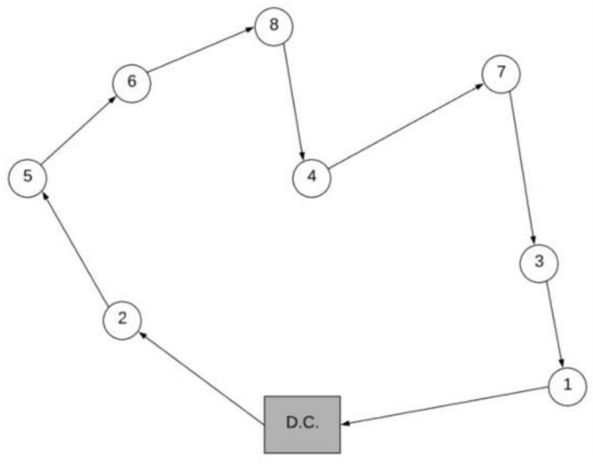

Figure 2.1. The optimal route for TSP

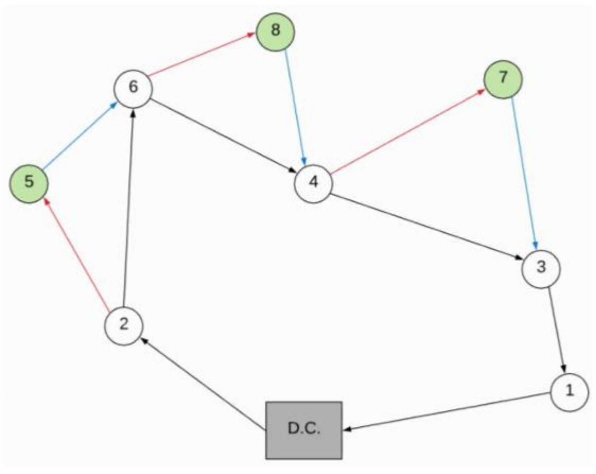

Figure 2.2. The optimal route for TSP-D

Deliverables:

Code for Brute-Force :

```
import itertools
# Define the distance matrix (symmetric matrix with distances between points)
# Replace this matrix with actual distances
distance_matrix = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]

# List of delivery points (0, 1, 2, ..., n-1)
n = len(distance_matrix)
delivery_points = list(range(n))

# Function to calculate the total distance of a given route
def calculate_route_distance(route):
    distance = 0
    for i in range(len(route) - 1):
        distance += distance_matrix[route[i]][route[i + 1]]
    # Return to the starting point
    distance += distance_matrix[route[-1]][route[0]]
    return distance

# Brute-force approach to find the minimum route
def find_optimal_route(points):
    min_distance = float('inf')
    optimal_route = None
    # Generate all permutations of delivery points (excluding the starting point for permutations)
    for perm in itertools.permutations(points[1:]):
        # Include the starting point (0) at the beginning of each route
        route = [points[0]] + list(perm)
        route_distance = calculate_route_distance(route)
        # Check if the current route is the shortest
        if route_distance < min_distance:
            min_distance = route_distance
            optimal_route = route
    return optimal_route, min_distance

# Run brute-force optimization
optimal_route, min_distance = find_optimal_route(delivery_points)

# Output the result
print(f"Optimal Route: {optimal_route}")
print(f"Minimum Distance: {min_distance}")
```

Code for Shortest Path :

```
import heapq

# Define the graph as an adjacency list (dictionary)
# Replace this graph with actual distances
graph = {
    0: {1: 10, 2: 15, 3: 20},
    1: {0: 10, 2: 35, 3: 25},
    2: {0: 15, 1: 35, 3: 30},
    3: {0: 20, 1: 25, 2: 30}
}

# Function to find the shortest path using Dijkstra's algorithm
def dijkstra(graph, start):
    # Initialize distances with infinity and set start node distance to 0
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    # Priority queue to keep track of the minimum distance node
    priority_queue = [(0, start)]  # (distance, node)

    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)

        # Skip if we found a shorter way to the node
        if current_distance > distances[current_node]:
            continue

        # Examine neighbors and update distances
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight

            # Only consider this path if it's better
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances

# Example usage
start_node = 0
shortest_paths = dijkstra(graph, start_node)

# Output the shortest paths from the starting node to each destination
for node, distance in shortest_paths.items():
    print(f"Shortest distance from node {start_node} to node {node}: {distance}")
```

Code for Priority Scheduling :

```
# Define a list of deliveries with (priority, delivery_id, duration)
# Higher number means higher priority
deliveries = [
    (3, 'Delivery 1', 2),
    (1, 'Delivery 2', 3),
    (4, 'Delivery 3', 1),
    (2, 'Delivery 4', 2)
]

# Function to perform Priority Scheduling
def priority_schedule(deliveries):
    # Sort deliveries by priority in descending order
    deliveries = sorted(deliveries, key=lambda x: -x[0])

    schedule = []
    current_time = 0

    # Process tasks based on priority
    for priority, delivery_id, duration in deliveries:
        schedule.append((delivery_id, current_time, current_time + duration))
        current_time += duration

    return schedule

# Schedule deliveries
schedule = priority_schedule(deliveries)

# Output the scheduling results
for delivery in schedule:
    print(f"{delivery[0]}: Start at {delivery[1]}h, Finish at {delivery[2]}h")
```

Results

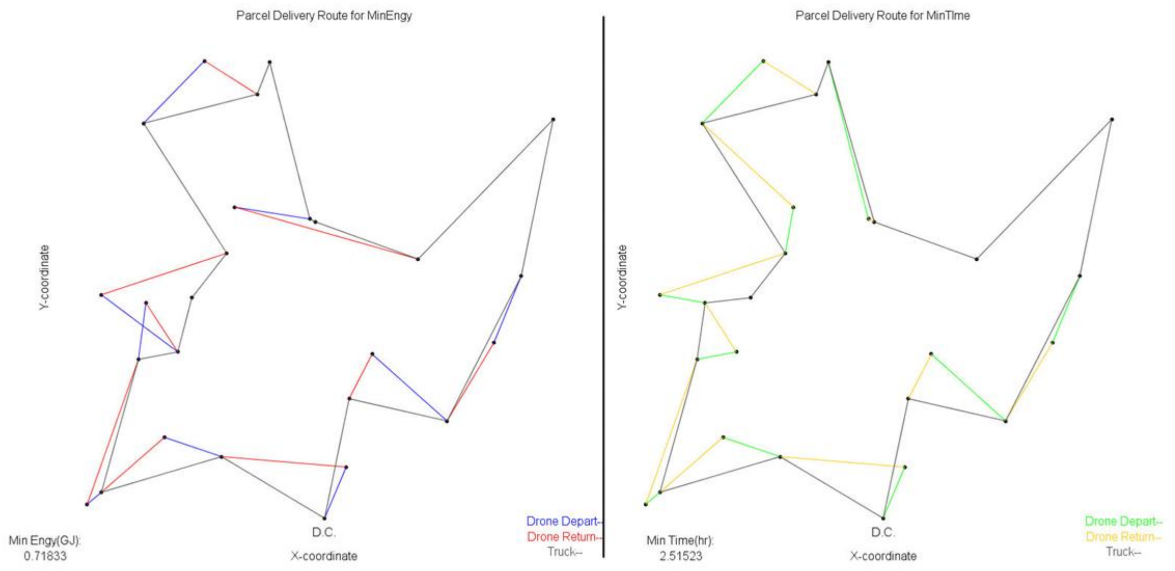

Figure 3.1. Optimal delivery route for 25 customers

Conclusion and Future Research

In this optimization of DADS project, I analyzed the scenario of a drone-assisted delivery system. Some past researches proved drone-assisted delivery is faster than truck standalone delivery, because the advantages of drones can offset the disadvantages of trucks. Hence, I developed a program to calculate the operational time and energy cost, and select the best solution of delivery route on the basis of MinTime and MinEngy.

Additionally, this program can optimize the delivery route based on the heuristic method-Genetic Algorithm, which can ensure the results are infinitely close to the global optimum. Owing to the deployment of effective GA, the program can solve the large-sized practical problem within a related short time, yet only has minimal influence on the quality of overall solutions.

Future research may aim to:

1. Simulate the logistics system using two or more drones;
2. Update the program to include the consideration of waiting time, and propose a reasonable model to evaluate the waiting penalties;
3. Improve this program to get a better performance when analyzing the larger practical size of scenario, and in the meanwhile reduce the runtime of the program by optimizing its structure.
4. Implement Google Map API to my program, to solve the realistic problem;