An exact method and an heuristic approach for the TSP-D with multiple drones problem

Anda Buinoschi MOC2, Bogdan Luncaşu MOC2

Abstract

The current paper presents two methods used for solving TSP with multiple drones. An exact method based on dynamic programming and an heuristic approach based on Genetic Algorithm and Hill Climbing show that using drones, we benefit of time-saving deliveries and reduce the costs.

1 Introduction

The problem states that a truck and a number of drones (or unmanned aerial vehicles, UAVs) deliver together a set of parcels for a set of customers geographically distributed in different places. The drone has the limit to deliver only one package and then it should return to the truck to take another and to make a new delivery. The final objective of the problem is to minimize the total time of the delivery process after successfully landed parcels to the customers.

2 Problem definition

Let C represent the set of customer parcels, such that $C = \{1, 2, ..., c\}$. Each customer must receive exactly one delivery by either the single truck or one of the drones available denoted by the set V. A particular customer $i \in C$ is "droneable" by UAV $v \in V$, belonging to the set $\hat{C}_v$ if the customer's parcel is eligible for drone delivery (it is not too heavy or any other restriction).

Each of the drone can only deliver one parcel at a time. It may be launched from any point on the map, either customer location or depot point. While it can be launched multiple times during the delivery process, it cannot leave the same location more than once. It can land at depot point, be retrieved by the truck at a customer's location, but it cannot land at the same customer location from which it was deployed. When a drone arrives to the truck, it may be loaded onto the truck and follow it for several locations or it can be launched with a new package. While the UAVs are airborne, the truck may deliver packages continuing its path.

The set of nodes are defined to be $N = \{0, 1, ..., c + 1\}$ where 0 and $c + 1$ represent the depot from which they start traveling and in which they have to return. The set of nodes from which the truck can depart are represented by

$N_0 = \{0, 1, ..., c\}$, while $N_+ = \{1, 2, ..., c+1\}$ are the set of nodes which the truck may visit.

The truck's travel time from a node $i \in N_0$ to $j \in N_+$ is given by τ_{ij}. Also, τ'_{vij} represents the time required for a UAV $v \in V$ to fly from node $i \in N_0$ to node $j \in N_+$.

When UAV $v \in V$ is launched from node $i \in N_0$, it requires $s^L_{v,i}$ units of time. The launch is indexed on v to keep track of differences in UAVs. The launch time is also indexed on the launch location; launching from the depot may require a different amount of time. The recovery time, $s^R_{v,k}$ is similarly defined for UAV $v \in V$ retrieved at node $k \in N_+$.

Truck deliveries require σ_k units of time for service at node $k \in N_+$, where $\sigma_{c+1} \equiv 0$ as there is no delivery at the depot. Similarly, UAV $v \in V$ requires σ'_{vk} units of time to perform the delivery service at node $k \in N_+$, where $\sigma'_{v,c+1} \equiv 0$.

3 Methods

3.1 Dynamic Programming

The approach for the Dynamic Programming method is formed of three phases during which we construct the path for the truck first, we compute the subproblems' cost by adding the drone on top of the truck's path and after that, we compute the subproblems' of efficient opperations for every subset combination. The approach used was first stated in the paper [3]. Below, I will do a resumé of these steps for solving an instance of TSP with drones.

3.1.1 Phase I

Phase I is a remodeled approach of the work of Held-Karp on TSP problem [4]. The optimal solution for the regular TSP problem can be considered as a path that start from an arbitary origin point, which we assume it is v_0, visits all locations in V and ends at the point where it started. We note with D_{TSP} the subproblem of finding the shortest path that start in origin point $v_0 \in V$, visits the locations in $S \subseteq V$ and ends in a point $w \in S$.

The subproblem $D_{TSP}(S, w)$ can be broken into two parts: the last arc that goes from a middle point $v \in S$ to $w \in V \backslash S$ and the shortest path from the origin point $v_0 \in S$ to the point $v \in S$. The recurrence formula is the following:

$$D_{TSP}(S, w) = \begin{cases} c(v_0, w) & \text{if } S = \{w\} \\ \infty & \text{if } w \notin S \\ \min_{v \in S \backslash \{w\}} D_{TSP}(S \backslash \{w\}, v) + c(v, w) & \text{otherwise} \end{cases},$$

where $c(x, y)$ represents the cost function between any two nodes $x, y \in V$.

The refined version of the referenced authors is that they solve this subproblems' costs for each depot point $v \in V$.

$$D_T(S,v,w) = \begin{cases} 0 & \text{if } v = w \wedge S = \{v\} \\ c(v,w) & \text{if } v \neq w \text{ and } S = \{v,w\} \\ \min_{u \in S \setminus \{v,w\}} D_T(S \setminus \{w\}, v, u) + c(u,w) & \text{otherwise} \end{cases},$$

where $c(x,y)$ represents the cost function between any two nodes $x, y \in V$.

3.1.2 Phase II

For the phase II, the authors added drone operations on top of the truck operations. Now, the problem of finding an efficient operation with included drone movement consists in two parts: for a droneable node $d \in S \setminus \{v,w\}$, we combine the flight of the drone via this middle point with the shortest truck path that starts in v, ends in w and visits all locations in $S \setminus \{d\}$. As the problem of finding shortest truck path is computed in the previous subsection, the table for D_{OP} for every triplet (S,v,w) is computed as:

$$D_{OP}(S,v,w) = \begin{cases} D_T(S,v,w) & \text{if } S = \{v,w\} \\ \infty & \text{if } v \notin S \vee w \notin S \\ \min_{d \in S \setminus \{v,w\}} \max\{c^d(v,d) + c^d(d,w), D_T(S \setminus \{d\}, v, w)\} & \text{otherwise} \end{cases}.$$

3.1.3 Phase III

In the third and final pass of their algorithm, it is computed the optimal table. In a subproblem $D(S,v)$ they look for a minimum cost sequence of efficient operations that starts at v_0, ends at v and visits all locations in S. The relation between the sub-problems can be expressed as:

$$D(S,v) = \begin{cases} 0 & \text{if } S = \{v_0\}, v = v_0 \\ \infty & \text{if } v \notin S \\ \min_{w \in S} \min_{T \subset S} D(S \setminus T, w) + D_{OP}(T,w,v) & \text{otherwise} \end{cases}.$$

3.2 Genetic Algorithm

3.2.1 Data modelling

- Chromosome = (tour_permutation, drone_visits) = (p, v)
- Permutation p: a permutation of the list [1,...,n-1]. The node 0 is appended at the beginning and end of p. The node 0 represents the depot.
- Nodes visited by vehicles = v
  - A list of vehicle ids, each vehicle is mapped to the corresponding node from p
  - The mapping is done by index (e.g. the node p_i is visited by vehicle v_i

3.2.2 Selection

For the parent selection we are using roulette selection where the best chromosomes get a higher chance of being selected.

The probability of selection is $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, where f represents the value of fitness function evaluated on chromosome c_i

3.2.3 Crossover

For the crossover operator we are using Partially Matched Crossover (PMX) to generate new permutations for tour and two point cut for drone assignments.

3.2.4 Mutation

For tour mutation we are using Shuffle Index Mutation which swaps two random cities from p(tour).

The mutation on vehicle assignments is done by replacing the current vehicle with a different one from existing pool.

3.2.5 Educate

The educate step is used to repair infeasible solutions.

An infeasible solution is a chromosome which breaks one or more of the imposed constraints(e.g. a non-dronable node is visited by a drone)

The educate step uses hill climbing to search for feasible solutions. This operation is done only on the vehicle array(v)

The neighbors are selected by replacing each node assignment with a different drone/truck. We keep trace of the drone's assignments so a drone will not visit two intermediary nodes between starting node i and rendezvous node k.

3.2.6 Objective function

The objective function represents the time spent to deliver all parcels by starting from depot and returning to depot.

We split the evaluation by computing the time spent from node i (launching node) to node k (rendezvous node). This is done by taking the maximum amount of time needed for all vehicles to reach node k.

3.2.7 Configuration optimization

The Genetic Algorithm has multiple configuration parameters used by the operators mentioned above. In order to find the most suitable configurations the F-race algorithm is used.

The search is done on the following configuration space:

```
configurations = {
    "population_size": [50, 70, 100],
    "crossover_probability": [0.1, 0.2, 0.3],
    "mutation_probability": [0.05, 0.1, 0.2],
    "chromosome_mutation_probability": [0.01, 0.05, 0.1],
    "tour_mutation_probability": [0.01, 0.05, 0.1],
    "drone_mutation_probability": [0.05, 0.1, 0.2]
}
```

Since this will create $3^6 = 729$ possible configurations, we are using Random Sampling to limit the number of configurations which will be checked as part of the F-race algorithm.

The number of instances used is 3, two of them having 8 nodes and the third one having 10 nodes.

We start the Friedman test when we have the results of at least 5 instances evaluated against all configurations got from the random sampling.

After we compute the configuration rankings on all the instances, we get the Friedman score and check against the critical value for n(number of instances) degrees of freedom and alpha = 0.05

If the Friedman score is higher than the critical value, we compute the posthoc_nemenyi_friedman between the best configuration and the other configurations to see if there is a major difference between them. If the $p_{value} <= 0.05$, the configuration checked against the best one is removed from the list.

The steps are repeated until we have only one configuration or the number of iterations is greater than 50.

3.2.8 Results

This section showcases the results of the algorithm on an instance with 10 nodes and 4 drones.

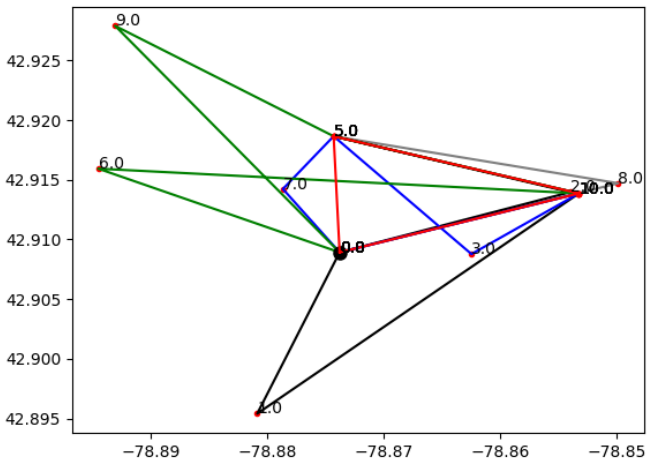
We used one of the selected configuration after the F-race step:

```
population_size = 100
chromosome_mutation_probability = 0.01
crossover_probability = 0.2
drone_mutation_probability = 0.05
mutation_probability = 0.05
tour_mutation_probability = 0.01
```

The following is the solution for the instance used in this section.

```
Tour                  0,6,1,10,3,5,8,9,7,2,0
Vehicle assignments   0,1,3,  0,2,0,4,1,2,3,0
```

and has a score of 520.44 units of time.



The red edges represents the truck path, and the other 4 colors represents the path of each of the drones.

4 Experiments

Algorithm	Number of nodes	Number of drones	Best score	Average score	Average execution time
Genetic Algorithm	10	4	520.44	598.91	7.84s
Genetic Algorithm	25	4	7872.68	8948.44	72.97s
Genetic Algorithm	50	4	16546.39	19053.60	308.15s

5 Conclusion

In conclusion, we state that not-free-lunch theorem applies in this case, too. While for low-dimensional data, the exact method can return the best solution in a reasonable time, for higher dimensions it is more worthy to try the heuristic approach which gives a solution very close to the optimal one.

References

[1] Chase C. Murray, Ritwik Raj, "The Multiple Flying Sidekicks Traveling Salesman Problem: Parcel Delivery with Multiple Drones", 2019
`https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3338436`

[2] Experimental data source
`https://github.com/optimatorlab/mFSTSP`

[3] Paul Bouman, Niels Agatz, Marie Schmidt, "Dynamic Programming Approaches for the Traveling Salesman Problem with Drone"
`https://core.ac.uk/download/pdf/154419746.pdf`

[4] Held-Karp algorithm for solving TSP
https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm

[5] Răschip Mădălina, Experimental Analysis of Algorithms class course, 2020
https://profs.info.uaic.ro/~mionita/aea/index.html

[6] A Hybrid Genetic Algorithm for the Traveling Salesman Problem with Drone
https://arxiv.org/pdf/1812.09351.pdf

[7] The Multiple Flying Sidekicks Traveling Salesman Problem (mFSTSP)
https://github.com/optimatorlab/mFSTSP