Ant Colony Optimization approach for TSP-D

Introduction - aspects related to TSP-D - the chosen variant:

We take into account the scenario when there is just one vehicle and one drone accessible. Our goal is to reduce the overall delivery job completion time, or the time it takes to serve every client, given a group of customers who will either be delivered by a truck or a drone. We assume that the drone can only be sent to a customer's location and that each location's service time is instantaneous. A workable method involves the truck route forming a tour that departs from and arrives at the depot, serving a subset of clients along the way. The drone is launched from a client location and returns to another (potentially different) location to service each remaining customer, and it only can go to one customer before having to return to the place where the truck is.

Problem formulation (from “A Study on the Traveling Salesman Problem with a Drone” Ziye Tang , Willem-Jan van Hoeve , and Paul Shaw)

We are given an undirected graph G = (V,E) with V = C ∪ {r} where r is the depot and C is the set of customers to be served by the truck or the drone. Let n = |C|. The travel time between a pair of nodes (i,j) is given by metric w(i,j). ρ ≥ 0 is the ratio between the truck’s and the drone’s travel time per unit distance. ρ is also called the speed differential. Every customer demands one parcel, which can be delivered by either a truck or a drone. A drone can only deliver one parcel at the time. We make the following assumptions about the behavior of the truck and the drone:

(a) The truck can dispatch and pick up a drone only at the depot or a customer location. The truck can continue serving customers after a drone is dispatched and reconnect with the drone at a possibly different node.

(b) The vehicle (truck or drone) that first arrives at the reconnection node has to wait for the other one, which we call synchronization.

(c) Upon returning to the truck, the time required to prepare the drone for another launch is negligible.

Our objective is to minimize the completion time, i.e. from the time the truck is dispatched from the depot with the drone to the time when the truck and the drone returns to the depot.

Ant Colony Optimization (ACO) is a metaheuristic optimization algorithm inspired by the foraging behavior of ants. The algorithm is used to find the shortest path or optimal solution in a graph or network. The Ant System variant is one of the most basic versions of ACO.

In the Ant System variant of ACO, the algorithm uses a set of artificial ants to explore a graph or network. Each ant starts at a **random node** and moves along the edges of the graph according to a set of rules.

What we've modified: **all ants start at the depot node** → details will follow below.

As the ants move, they deposit pheromone on the edges they traverse. The amount of pheromone deposited is proportional to the **quality** of the edge.

After each ant completes its tour, the amount of pheromone on the edges is updated based on the quality of the tour. Edges that were part of a good tour receive more pheromone than edges that were part of a bad tour. The pheromone is also evaporated over time, simulating the decay of the pheromone trails.

As the algorithm progresses, the pheromone on the edges guides the ants towards the optimal solution. Ants are more likely to choose edges with higher pheromone levels, thus reinforcing the paths that lead to the optimal solution. This reinforcement process is known as positive feedback.

What we have done: we have simply applied Ant Colony Optimization, and we have only adapted the problem specifics into the original AS implementation.

Chosen dataset - proposed by Bouman and used in the research paper - Publicly available here: https://github.com/pcbouman-eur/TSP-D-Instances

Why did we choose this dataset - it has many instances to choose from, and a description where we could understand exactly the format. Some of the instances have a solution available, some of them not, so we chose five solved instances (relatively small size, generally 9 nodes) and five (bigger) instances of a much bigger size.

More details and formulas used (basically the known ones):

At each step, a certain ant (k) applies a probabilistic action choice rule. In particular, the probability with which ant k, currently at location x, chooses to go to city y at the t th iteration of the algorithm is: (formulas below are referencing Wikipedia)

$$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{z \in \text{allowed}_x} (\tau_{xz}^\alpha)(\eta_{xz}^\beta)}$$

where τ_{xy} is the amount of pheromone deposited for transition from state x to y, $0 \leq \alpha$ is a parameter to control the influence of τ_{xy}, η_{xy} is the desirability of state transition xy (*a priori* knowledge, typically $1/d_{xy}$, where d is the distance) and $\beta \geq 1$ is a parameter to control the influence of η_{xy}. τ_{xz} and η_{xz} represent the trail level and attractiveness for the other possible state transitions.

“The other possible state transitions in our case” -> the set of locations which ant k has not yet visited -> this is our neighborhood

Therefore, α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information respectively.

We will use the same heuristic approach -> the inverse of the distance from location x to location y, so the same formula as above will be applied.

But - after choosing our best node, we need to decide if we use the drone or the truck at each point in our itinerary.

First of all, we take into consideration the restrictions - for example, we check if the drone is with the truck and if it can go to a city (the chosen one). If not -> we go by truck to that location. Else, if we can choose from both, we compute two probabilities, for truck and drone, and make a choice based on these probabilities. -> similar to the formula above,

Trails are usually updated when all ants have completed their solution, increasing or decreasing the level of trails corresponding to moves that were part of "good" or "bad" solutions, respectively. An example of a global pheromone updating rule is

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_k^m \Delta\tau_{xy}^k$$

where τ_{xy} is the amount of pheromone deposited for a state transition xy, ρ is the *pheromone evaporation coefficient*, m is the number of ants and $\Delta\tau_{xy}^k$ is the amount of pheromone deposited by kth ant, typically given for a TSP problem (with moves corresponding to arcs of the graph) by

$$\Delta\tau_{xy}^k = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

where L_k is the cost of the kth ant's tour (typically length) and Q is a constant.

and we will use these exact formulas in our implementation too, and we used Q = 5.0 and L will have the same meaning - the tour length.

Parameters:

colony size = 100

steps = 70

alpha = 1.2

beta = 3.1

rho = 0.1

pheromone deposit weight (Q) = 5

Results:

uniform-42-n9.txt

[197.2561102604338, 209.27028801164687, 208.5864418884052, 210.88929428912695, 214.11171567725262, 224.59280923479298, 209.52105599582842, 219.15313859207868, 197.6279029908963, 222.30150131755568, 221.89512930273142, 209.52105599582842, 209.52105599582842, 209.52105599582842, 197.6279029908963, 214.11171567725262, 209.52105599582842, 209.52105599582842, 219.15313859207868, 214.11171567725262, 208.58644188840526, 214.11171567725262, 214.11171567725262, 209.52105599582842, 209.52105599582842, 209.52105599582842, 202.81360608879254, 222.30150131755568, 214.11171567725262, 214.11171567725262]

mean: 211.55089068228736

min: 197.2561102604338

uniform-45-n9.txt

[235.75322390474912, 202.06953660116798, 223.70862048953194, 222.76108419779862, 222.81980752753338, 222.81980752753338, 225.6361361381243, 220.5432946420036, 233.05648003951393,

209.03646448826726, 231.77198936564258, 229.769597499364,
212.73627449210863, 212.73627449210863, 225.6361361381243,
212.73627449210863, 209.03646448826726, 234.64779323924085,
213.5662044439169, 225.6361361381243, 212.73627449210863,
213.36996015484783, 223.70862048953194, 225.6361361381243,
209.7817403196147, 225.6361361381243, 213.5662044439169,
220.54329464200356, 220.5432946420036, 229.769597499364]

mean: 220.72562864349564

min: 202.06953660116798

uniform-47-n9.txt

[217.93240447012016, 217.93240447012016, 217.10782909302043,
217.93240447012016, 220.91063258230517, 219.97658651268415,
217.93240447012016, 216.38105642607445, 217.93240447012016,
216.38105642607445, 217.93240447012016, 198.32303962499128,
219.97658651268415, 220.62472883288996, 198.32303962499128,
216.38105642607445, 222.03913428496645, 216.38105642607445,
216.38105642607445, 217.93240447012016, 228.93072678930386,
217.93240447012016, 198.32303962499128, 217.93240447012016,
220.04156694658914, 216.38105642607445, 216.38105642607445,
216.38105642607445, 217.93240447012016, 217.93240447012016]

mean:216.37892954844145

min: 198.32303962499128

uniform-48-n9.txt

[202.4906451181239, 203.13218878320689, 210.15427280860118,
232.70273410546517, 207.21246195796124, 224.0001647433821,
200.1004883211757, 196.35750071910311, 200.1004883211757,
193.10252943629112, 210.86906145947674, 223.07175817929277,
196.35750071910311, 207.21246195796124, 202.4906451181239,
202.4906451181239, 219.52756197616895, 200.1004883211757,
210.86906145947674, 207.21246195796124, 201.86586219978588,
202.4906451181239, 207.21246195796127, 196.35750071910311,

232.70273410546517, 210.15427280860118, 196.35750071910311, 196.35750071910311, 237.39602788438287, 202.4906451181239]

mean: 207.76467573103682

min: 193.10252943629112

uniform-80-n50.txt

[581.0253365112092, 575.3783911694311, 569.8657100237746, 538.3749538959406, 535.267280734005, 564.8502802338795, 567.2993889446449, 599.1310497785569, 554.2689603020946, 553.4824797540391, 545.6521978319707, 577.1749481926399, 570.1975478903448, 549.2071163532058, 521.6332211013292, 548.2444968039853, 525.0128767737275, 574.4096013819327, 557.0480301437848, 593.2809399075178, 538.4138186564704, 552.4416803158006, 578.2858659702432, 539.2238062188763, 541.1300235809143, 590.1180734306972, 546.4250712327223, 555.9038816593376, 568.6689653966138, 608.2251202163828]

mean: 560.6547038135358

min: 521.6332211013292

Genetic Algorithm

Because evolutionary algorithms may solve optimization issues and approximate a global minimum, this was our strategy. By performing selection, crossover, and mutation operations on a chromosomal representation, a genetic algorithm is defined.

Representation of chromosome:

After initially opting for different variants, we got inspired by this approach : https://arxiv.org/pdf/1812.09351.pdf - and separated the chromosome in two components: the list of locations (for the truck, starting and ending with the depot = 0) and a list for the drone - with zeros and ones. Intersections = 0 and the nodes where the drone goes = 1 -> after a value of 1 we need a zero and we cannot have two or more consecutive 1s.

We compute for each city the distances by other cities, so drone distances are computed as follows: truck distances * speed of drone. -> speed of drone is always 0.5 meaning double speed.

Utilizing truck intersections with the drone allows for the creation of a solution. We ensure that there are no longer any pairs of 0s next to one another and that 1 comes before 0. A population of 50 chromosomes and a max of of 50 iterations make up the algorithm setup.

-Roulette wheel selection
-Simple mutation

Results:

uniform-42-n9.txt

234.99960822253038

min 203.27708201882712

uniform-45-n9.txt

189.4407905952196

min 205.77632220929064

uniform-47-n9.txt

289.2192456681967

min 220.1351002464866

uniform-48-n9.txt

239.0933731700255

min 218.33708040548566

uniform-80-n50.txt

1474.1002961874096