# Adaptive Ant Colony Optimization with Node Clustering for the Multi-Depot Mixed Fleet Capacitated Multiple TSP

Panagiotis Zachos

March 1, 2024

## 1 Variables

- $V = \{D \cup C\}$ : total vertices, Depots & Customers
- $VT$ : total Vehicle Types
- $D$ : total Depots
- $R$ : solution
- $K$ : cluster matrix
- $\tau$ : pheromone matrix
- $n_{ants}$ : number of ants in colonies
- $n_{freq}$ : frequency of the local optimization
- $n_{prim}$ : number of primary clusters
- $n_{size}$ : number of vertices in clusters
- $n_{sect}$ : number of sectors
- $T_{update}$ : temperature udpating coefficient
- $\alpha_{update}$ : temperature cooling coefficient
- $\rho_{min}, \rho_{max}$ : minimum and maximum limits of the pheromone evaporation coefficient
- $\delta$ : pheromone updating coefficient
- $\lambda$ : drones' pheromone updating coefficient
- $\alpha$ : distance probability coefficient
- $\beta$ : pheromone probability coefficient

At the initial phase of the algorithm, the pheromone matrix $\tau$ is initialized using (1). $\tau$ is a 4-dimensional matrix.

$\tau_{ij}^{(k)(h)} = 1$ for all $v_i, v_j \in V,\ vt_k \in VT$ and $d_h \in D$ (6)

Pheromone matrix update, evaporation coefficient $\rho$ and pheromone evaporation follow the same principles with the original AACO-NC found here.

Because of the addition of drones, which function in a star graph, moving only from depots to customers and back, adjustment was needed to the drone pheromone values to prevent them becoming so strong that they're the only type selected in each ant iteration. More specifically, we present a new update coefficient for drones in the pheromone update procedure:

**if** $vt = Drone$ **then**

$\tau_{dj}^{(vt)(d)} = \tau_{dj}^{(vt)(d)} + x_{dj} \cdot \frac{\lambda}{|v_d - v_j|} \cdot \frac{|R|}{|R^{update}|}$ , $\lambda = 2$

$x_{dj} = \begin{cases} 1 & \text{if edge from Depot d to customer j exists in } R^{update} \\ 0 & \text{otherwise} \end{cases}$

**else**

$\tau_{ij}^{(vt)(d)} = \tau_{ij}^{(vt)(d)} + x_{ij} \cdot \delta \cdot \frac{|R|}{|R^{update}|}$ , $\delta = 3$

$x_{ij} = \begin{cases} 1 & \text{if edge from } v_i \text{ to } v_j \text{ exists in } R^{update} \\ 0 & \text{otherwise} \end{cases}$

And in the evaporation procedure:

**if** $vt = Drone$ **then**

$\tau_{ij}^{(vt)(d)} = \tau_{ij}^{(vt)(d)} \cdot (1 - \rho \cdot 2)$ for all $v_i, v_j \in V$ , $t \in VT$ and $d \in D$

**else**

$\tau_{ij}^{(vt)(d)} = \tau_{ij}^{(vt)(d)} \cdot (1 - \rho)$ for all $v_i, v_j \in V$ , $t \in VT$ and $d \in D$

The node clustering technique also follows the original paper, which can be found here. Although in this algorithm, each vertex has more than one set of clusters, which is equal to the number of different vehicle types. This is because of the inability of certain vehicles to visit every vertex. Also, the drone clusters do not make use of the sectoring technique because that sometimes results in drones being assigned to too far away customers which is suboptimal and not realistic for large area instances.

# 2 Functions

In general, the functions used in this algorithm are slightly altered versions of the ones presented in the AACO-NC for the MDVRP by Stodola, with the purpose of adapting the AACO-NC algorithm to solve the MD-mfcmTSP.

The MD-mfcmTSP differs from the MDVRP in the following ways:

1. Supports multiple vehicle types (different capacities and speeds).
2. Each vehicle type has an assigned speed.
3. Considers realistic scenarios where big vehicles cannot access certain customers and drone safe landing spaces (not all customers can be serviced by drone).
4. Minimizes makespan instead of distance.
5. Vehicle capacity is dictated by the vehicle type and not by the vehicle's depot (depots are considered always stocked and as a reloading station for their vehicles).

6. Not every depot has to have the same number of vehicles or vehicle types assigned (e.g. Depot 1 has 4 trucks, Depot 2 has 10 drones and Depot 3 has 3 trucks, 5 motorcycles and 7 drones).

**Algorithm 1:** AACONC

**Function** *AACONC*($V = \{C \cup D\}, n_{ants}, n_{freq}, n_{size}, n_{sect}, n_{prim}, T_{update}, \alpha, \beta, \rho_{min}, \rho_{max}, \delta$)
    $|R| \leftarrow \infty$;
    $iter \leftarrow 0$;
    Initialize pheromone matrices $\tau$;
    **for** *each* $t_i \in VT$ **do**
        **if** $t_i = Drone$ **then**
            **for** *each* $v_i \in D^{(t_i)}$ **do**
                $K^{(t_i)(v_i)} \leftarrow$ createClustersDrone($C^{(t_i)}, v_i, n_{size}$);
        **else**
            **for** *each* $v_i \in V^{(t_i)}$ **do**
                $K^{(t_i)(v_i)} \leftarrow$ createClusters($V^{(t_i)}, v_i, n_{size}, n_{sect}, n_{prim}$);
    **while** *not terminated* **do**
        $|R_{\text{best}}| \leftarrow \infty$;
        $iter \leftarrow iter + 1$;
        **for** $a = 1$ *to* $n_{ants}$ **do**
            $R_a \leftarrow$ AntSolution($V, K, \tau, \alpha, \beta$);
            **if** $|R_a| < |R_{best}|$ **then**
                $R_{\text{best}} \leftarrow R_a$;
        **if** $iter \mod n_{freq} = 0$ **then**
            $R_{\text{best}} \leftarrow$ LocalOptimization($V, R_{\text{best}}$);
        **if** $|R_{best}| < |R|$ **then**
            $R \leftarrow R_{\text{best}}$;
        Update pheromone matrices $\tau$;
        Calculate evaporation coefficient $\rho$;
        Evaporate pheromone matrices $\tau$ using $\rho$;
    **return** $R$;

**Algorithm 2:** createClustersDrone

**Function** *createClustersDrone*($C^{(t_i)}, v_i, n_{size}$)
    $id = 1$
    $K_{id}^{(v_i)} = \emptyset$
    $V_{free} = C^{(t_i)}$
    **while** $V_{free} \neq \emptyset$ **do**
        **if** $|K_{id}^{(v_i)}| \geq n_{size}$ **then**
            $id = id + 1$
            $K_{id}^{(v_i)} = \emptyset$
        Find closest vertex $v \in V_{free}$ to $v_i$
        $K_{id}^{(v_i)} = K_{id}^{(v_i)} + \{v\}$
        $V_{free} = V_{free} - \{v\}$
    **return** $K^{(v_i)}$;

**Algorithm 3:** createClusters

**Function** *createClusters*($V^{(t_i)}, v_i, n_{size}, n_{sect}, n_{prim}$)
    $id = 1$
    $K_{id}^{(v_i)} = \emptyset$
    $V_{free} = V^{(t_i)} - v_i$
    **for** $j = 1$ *to* $n_{sect}$ **do**
        Find closest vertex $v \in V_{free}$ to $v_i$ in sector $j$
        $K_{id}^{(v_i)} = K_{id}^{(v_i)} + \{v\}$
        $V_{free} = V_{free} - \{v\}$
    **while** $V_{free} \neq \emptyset$ **do**
        **if** $|K_{id}^{(v_i)}| \geq n_{size}$ **then**
            $id = id + 1$
            $K_{id}^{(v_i)} = \emptyset$
        Find closest vertex $v \in V_{free}$ to $v_i$
        $K_{id}^{(v_i)} = K_{id}^{(v_i)} + \{v\}$
        $V_{free} = V_{free} - \{v\}$
    **return** $K^{(v_i)}$;

**Algorithm 4:** antSolution

**Function** *antSolution*($V = \{D, C\}, K, \tau, \alpha, \beta$)
  $V_{free} = C$;
  **while** $V_{free} \neq \emptyset$ **do**
    $vt = \text{selectVehicleType}(V_{free}, K, \tau)$
    $d = \text{selectDepot}(vt, V_{free}, K^{(vt)}, \tau)$
    $v = \text{selectVehicle}(vt, d, V_{free}, K^{(vt)}, \tau)$
    $pos \leftarrow$ vehicle's position
    $k = \text{selectCluster}(vt, d, v, V_{free}, K^{(pos)(vt)}, \tau, \alpha, \beta)$
    $V_{candidates} = V_{\text{free}} \cap K_k^{(pos)(vt)}$
    $c = \text{selectCustomer}(vt, d, pos, V_{candidates}, \tau, \alpha, \beta)$
    **if** $vt = Drone$ **then** //Drone serves customer and immediately returns to depot
      $R_d^{vt} = R_d^{vt} + \{c\}$
      $R_d^{(vt)} = R_d^{(vt)} + \{d\}$
    **else**
      **if** $v_{load} < c^{(demand)}$ **then**
        $R_d^{(vt)} = R_d^{(vt)} + \{d\}$
        $v_{load} = vt_{capacity}$
      $R_d^{vt} = R_d^{vt} + \{c\}$
      $v_{pos} = \{c\}$
      $v_{load} = v_{load} - c^{(demand)}$
    $V_{\text{free}} = V_{\text{free}} - \{c\}$
  **foreach** $d \in D$ *and* $vt \in VT$ **do** //Vehicles return to their depots
    $R_d^{vt} = R_d^{vt} + \{d\}$
  **return** $R = \{R_1^1, R_2^1, ..., R_2^3, R_3^3, ..., R_D^{VT}\}$

**Algorithm 5:** selectVehicleType

**Function** *selectVehicleType*($V_{free}, K, \tau$)
  **for** *each vehicle type* $t_i \in VT$ **do**
    $V_{\text{cand}} = \emptyset$
    **for** *each vehicle* **do**
      $pos \leftarrow$vehicle's current location
      $d \leftarrow$vehicle's depot
      **for** $k$ = *1 to* $n_{prim}$ **do**
        $V_{cand} = V_{cand} + V_{free} \cap K_k^{(t_i)(pos)}$
    $p(t_i) = \sum_{v_j \in V_{cand}} \tau_{v_{pos} v_j}^{(t_i)(d)} \div |vehicles^{(t_i)}|$
  $p_{sum} = \sum_{t_i \in VT} p(t_i)$
  $p(t_i) = p(t_i) \div p_{sum}$
  Select $t_i \in VT$ based on probabilities $p(t_i)$ using roulette wheel
  **return** $t_i$;

**Algorithm 6:** selectDepot

```
Function selectDepot(vt, V_free, K^(vt), τ)
    for each d_i ∈ D^(vt) do
        V_cand = ∅
        for each vehicle do
            pos ←vehicle's current location
            for k = 1 to n_prim do
                V_cand = V_cand + V_free ∩ K_k^(vt)(pos)
        p(d_i) = Σ_{v_j ∈ V_cand} τ_{v_pos v_j}^(vt)(d_i)
    p_sum = Σ_{d_i ∈ D^(vt)} p(d_i)
    p(d_i) = p(d_i) ÷ p_sum
    Select d_i ∈ D(vt) based on probabilities p(d_i) using roulette wheel
    return d_i;
```

**Algorithm 7:** selectVehicle

```
Function selectVehicle(vt, d, V_free, K^(vt), τ)
    for each v_i ∈ Vh^(vt)(d) do
        V_cand = ∅
        pos ←vehicle's current location
        for k = 1 to n_prim do
            V_cand = V_cand + V_free ∩ K_k^(vt)(pos)
        p(v_i) = Σ_{v_j ∈ V_cand} τ_{v_pos v_j}^(vt)(d)
    p_sum = Σ_{v_i ∈ V^(vt)(d)} p(v_i)
    p(v_i) = p(v_i) ÷ p_sum
    Select v_i ∈ Vh^(vt)(d) based on probabilities p(v_i) using roulette wheel
    return v_i;
```

**Algorithm 8:** selectCluster

**Function** *selectCluster*($vt, d, pos, V_{free}, K, \tau, \alpha, \beta$)
  **for** $k = 1$ *to* $n_{prim}$ **do**
    $V_{cand} = \emptyset$
    $V_{cand} = V_{free} \cap K_k^{(vt)(pos)}$
    **if** $V_{cand} = \emptyset$ **then**
      $\eta_k = \tau_k = 0$
    **else**
      $\eta_k = |V_{cand}| \cdot \sum_{v_j \in V_{cand}} |v_{pos} - v_j|^{-1}$
      $\tau_k = \frac{1}{|V_{cand}|} \cdot \sum_{v_j \in V_{cand}} \tau_{v_{pos} v_j}^{(vt)(d)}$
  $\eta_{sum} \leftarrow \sum_{k=1}^{n_{prim}} \eta_k^{\alpha}$
  $\tau_{sum} \leftarrow \sum_{k=1}^{n_{prim}} \tau_k^{\beta}$
  **if** $\eta_{sum} = 0$ **then**
    `// return first cluster with a free customer`
    **for** $k = n_{prim} + 1$ *to* $|K^{(vt)(pos)}|$ **do**
      $V_{cand} = V_{free} \cap K_k^{(vt)(pos)}$
      **if** $V_{cand} \neq \emptyset$ **then**
        return k
  **for** $k = 1$ *to* $n_{prim}$ **do**
    $p({K_k}^{(vt)(pos)}) = \frac{{\eta_k}^{\alpha} \cdot {\tau_k}^{\beta}}{\eta_{sum} \cdot \tau_{sum}}$
  $p_{sum} = \sum_{k \in n_{prim}} p({K_k}^{(vt)(pos)})$
  $p({K_k}^{(vt)(pos)}) = p({K_k}^{(vt)(pos)}) \div p_{sum}$
  Select $p({K_k}^{(vt)(pos)}) \in p(K^{(vt)(pos)})$ based on probabilities $p({K_k}^{(vt)(pos)})$
  **return** $k$;

**Algorithm 9:** selectCustomer

**Function** *selectCustomer*($vt, d, pos, V_{candidates}, \tau, \alpha, \beta$)
  **for** *each* $v_i \in V_{cand}$ **do**
    $p(v_i) = |v_{pos} - v_i|^{-\alpha} \cdot (\tau_{v_{pos} v_i}^{(vt)(d)})^{\beta}$
  $p_{sum} = \sum_{v_i \in V_{cand}} p(v_i)$
  $p(v_i) = p(v_i) \div p_{sum}$
  Select $v_i \in V_{cand}$ based on probabilities $p(v_i)$ using roulette wheel
  **return** $v_i$

**Algorithm 10:** LocalOptimization

1 **Function** *LocalOptimization*($V, R_{best}$)
2 **for** *each* $t_i \in VT$ **do**
3 **for** *each* $d_i \in D^{(t_i)}$ **do**
4 singleColonyOpt($R_{best}^{(t_i)(d_i)}, n_{max} = 1$)
5 singleColonyOpt($R_{best}^{(t_i)(d_i)}, n_{max} = 2$)
6 (Moves $n_{max}$ successive customer node(s) to different positions in the same route)
7 mutualColonyOpt($R_{best}^{(t_i)}, n_{max} = 1$)
8 **if** $t_i \neq Drone$ **then**
9 mutualColonyOpt($R_{best}^{(t_i)}, n_{max} = 2$)
10 (Moves $n_{max}$ successive customer node(s) from each $R_{best}^{(t_i)(d_i)}$ , $d_i \in D^{(t_i)}$ to different positions in each $R_{best}^{(t_i)(d_j)}$ , $d_j \neq d_i$)
11
**return** $R_{best}$

**Algorithm 11:** singleColonyOptimization

1 **Function** *singleColonyOptimization*($R_{best}^{(t_i)(d_i)}, n_{max}$)
2 **for** $n$ = *1 to* $n_{max}$ **do**
3 **foreach** *combination of n successive nodes in the route* **do**
4 move the nodes to a different place on the same route
5 evaluate the newly-created solution
6 **if** *this solution is better than the original and all constraints are satisfied* **then**
7 replace the original with the new solution
8 continue in point 4 **unless** all possible places in the route have been already evaluated
9 **return** $R_{best}$

**Algorithm 12:** mutualColonyOptimization

1 **Function** *mutualColonyOptimization*($R_{best}^{(t_i)}, n_{max}$)
2 **for** $n$ = *1 to* $n_{max}$ **do**
3 **foreach** *possible pair of depots d1 and d2* **do**
4 **foreach** *combination of n successive nodes in the route of d1* **do**
5 remove the nodes from the route of d1 and
6 insert them into the route of d2
7 evaluate the newly-created solution
8 **if** *this solution is better than the original and all constraints are satisfied* **then**
9 replace the original with the new solution
10 continue in point 6 **unless** all possible places in the route of d2 have been already evaluated
11 **return** $R_{best}$

# 3 Other

Because this is a new problem, there are no existing instances with which we can test the algorithm so we use Cordeau MDVRP instances while changing the demand of all customers to 1 and randomly setting each customer's accessibility with the probabilities:
85% to be accessible by drones
80% to be accessible by truck
while all customers can be accessed by motorbikes.