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

Panagiotis Zachos

May 9, 2024

1 Variables

- $V = \{D \cup C\}$: total vertices, Depots & Customers
- VT : total Vehicle Types
- D : total Depots
- R : solution
- K : cluster matrix
- τ : 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
- α_{update} : temperature cooling coefficient
- ρ_{min}, ρ_{max} : minimum and maximum limits of the pheromone evaporation coefficient
- δ : pheromone updating coefficient
- α : distance probability coefficient
- β : pheromone probability coefficient

At the initial phase of the algorithm, the pheromone matrix τ is initialized using (1). τ 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 ρ and pheromone evaporation follow the same principles with the original AACO-NC found here.

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 restrictions which forbid some vehicles from visiting certain customers.

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

Input: $V, n_{\text{ants}}, n_{\text{freq}}, n_{\text{size}}, n_{\text{sect}}, n_{\text{prim}}, T_{\text{update}}, \alpha, \beta, \rho_{\min}, \rho_{\max}, \delta$

$|R| \leftarrow \infty$
iter $\leftarrow 0$
Initialize pheromone matrices τ
for each $t \in VT$
  for each $v_i \in V^{(t)}$
    $K^{(t)(v_i)} \leftarrow$ CreateClusters$(V^{(t)}, v_i, n_{size}, n_{sect}, n_{prim})$
  end for
end for
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**
      $R_{\text{best}} \leftarrow R_a$
    end if
  end for
  if $iter \mod n_{freq} = 0$ **then**
    $R_{\text{best}} \leftarrow$ LocalOptimization(V, R_{best})
  end if
  if $|R_{best}| < |R|$ **then**
    $R \leftarrow R_{\text{best}}$
  end if
  Update pheromone matrices τ
  Calculate evaporation coefficient ρ
  Evaporate pheromone matrices τ using ρ
end while
return R

Algorithm 2: antSolution

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

Algorithm 3: selectVehicleType

Function *selectVehicleType*(V_{free}, K, τ)
  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**
        $V_{cand} = V_{cand} + V_{free} \cap K_k^{(pos)(t_i)}$
      end for
    end for
    $p(t_i) = \sum_{v_j \in V_{cand}} \tau_{v_{pos} v_j}^{(t_i)(d)}$
  end for
  $p_{\text{sum}} = \sum_{t_i \in VT} p(t_i)$
  return $rouletteWheel(p(VT), p_{sum})$

Algorithm 4: selectDepot

Function *selectDepot*$(vt, V_{free}, K^{(vt)}, \tau)$
  for *each* $d_i \in D^{(vt)}$ **do**
    $V_{cand} = \emptyset$
    for *each vehicle* **do**
      $pos \leftarrow$vehicle's current location
      for k = *1 to* n_{prim} **do**
        $V_{cand} = V_{cand} + V_{\text{free}} \cap K_k^{(pos)(vt)}$
      end for
    end for
    $p(d_i) = \sum_{v_j \in V_{cand}} \tau_{v_{pos} v_j}^{(vt)(d_i)}$
  end for
  $p_{sum} = \sum_{d_i \in D^{(vt)}} p(d_i)$
  return $rouletteWheel(p(D^{(vt)}), p_{sum})$

Algorithm 5: selectVehicle

Function *selectVehicle*$(vt, d, V_{free}, K^{(vt)}, \tau)$
  for *each* $v_i \in V^{(vt)(d)}$ **do**
    $V_{\text{cand}} = \emptyset$
    $pos \leftarrow$vehicle's current location
    for k = *1 to* n_{prim} **do**
      $V_{cand} = V_{cand} + V_{free} \cap K_k^{(pos)(vt)}$
    end for
    $p(v_i) = \sum_{v_j \in V_{cand}} \tau_{v_{pos} v_j}^{(vt)(d)}$
  end for
  $p_{sum} = \sum_{v_i \in V^{(vt)(d)}} p(v_i)$
  return $rouletteWheel(p(V^{(vt)(d)}), p_{sum})$

Algorithm 6: selectCluster

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

Algorithm 7: selectCustomer

Function *selectCustomer*($vt, d, pos, V_{candidates}, \tau, \alpha, \beta$)
  for *each* $v_i \in V_{cand}$ **do**
    $p(v_i) = |v_{pos} - v_i|^{-\alpha} \cdot (\tau_{v_{pos} v_i}^{(vt)(d)})^{\beta}$
  end for
  $p_{sum} = \sum_{v_i \in V_{cand}} p(v_i)$
  return $rouletteWheel(p(v), p_{sum})$

Algorithm 8: LocalOptimization

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

Algorithm 9: singleColonyOptimization

1 **Function** *singleColonyOptimization*($R_{best}^{(t_i)(d_i)}, n_{max}$)
2   **for** n = *1 to* n_{max} **do**
3   **for each** *combination of n successive nodes in the route*
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   **end if**
9   continue in point 4 **unless** all possible places in the route have been already evaluated
10   **end for**
11   **end for**
12   **return** R_{best}

Algorithm 10: mutualColonyOptimization

1 **Function** *mutualColonyOptimization*($R_{best}^{(t_i)}, n_{max}$)
2   **for** n = *1 to* n_{max} **do**
3   **for each** *possible pair of depots d1 and d2*
4   **for each** *combination of* n *successive nodes in the route of d1*
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   **end if**
11   continue in point 6 **unless** all possible places in the route of d2 have been already evaluated
12   **end for**
13   **end for**
14   **end for**
15   **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 will create new random ones with the following structure:

{number of customers} {number of depots} {number of different vehicle types}

$\{\text{capacity}_{vt_1}\}, \{\text{capacity}_{vt_2}\}, \{...\}, \{\text{capacity}_{vt_n}\}$

$\{\text{speed}_{vt_1}\}, \{\text{speed}_{vt_2}\}, \{...\}, \{\text{speed}_{vt_n}\}$

$\{\text{customer1}_{id}\}\{\text{customer1}_{x}\}\{\text{customer1}_{y}\}\{\text{customer1}_{demand}\}\{\text{customer1}_{accessibility}\}$

$\{\text{customer2}_{id}\}\{\text{customer2}_{x}\}\{\text{customer2}_{y}\}\{\text{customer2}_{demand}\}\{\text{customer2}_{accessibility}\}$
.
.
.
$\{\text{customerN}_{id}\}\{\text{customerN}_{x}\}\{\text{customerN}_{y}\}\{\text{customerN}_{demand}\}\{\text{customerN}_{accessibility}\}$

$\{\text{depot1}_{id}\}, \{\text{depot1}_{x}\}, \{\text{depot1}_{y}\}$, {number of vehicles of type 1}, {n of vehicles of type 2}, ..., {n of vehicles of type N}

$\{\text{depot2}_{id}\}, \{\text{depot2}_{x}\}, \{\text{depot2}_{y}\}$, {number of vehicles of type 1}, {n of vehicles of type 2}, ..., {n of vehicles of type N}
.
.
.
$\{\text{depotN}_{id}\}, \{\text{depotN}_{x}\}, \{\text{depotN}_{y}\}$, {number of vehicles of type 1}, {n of vehicles of type 2}, ..., {n of vehicles of type N}

$\text{customer}_{accessibility}$ is an integer ≥ 1 which shows what vehicle type(s) can serve the customer.
Examples:
"123" means that the customer can be served by vehicle types 1, 2 and 3
"3" means that the customer can only be served by vehicle type 3