A heterogeneous vehicle routing problem with drones and multiple depots

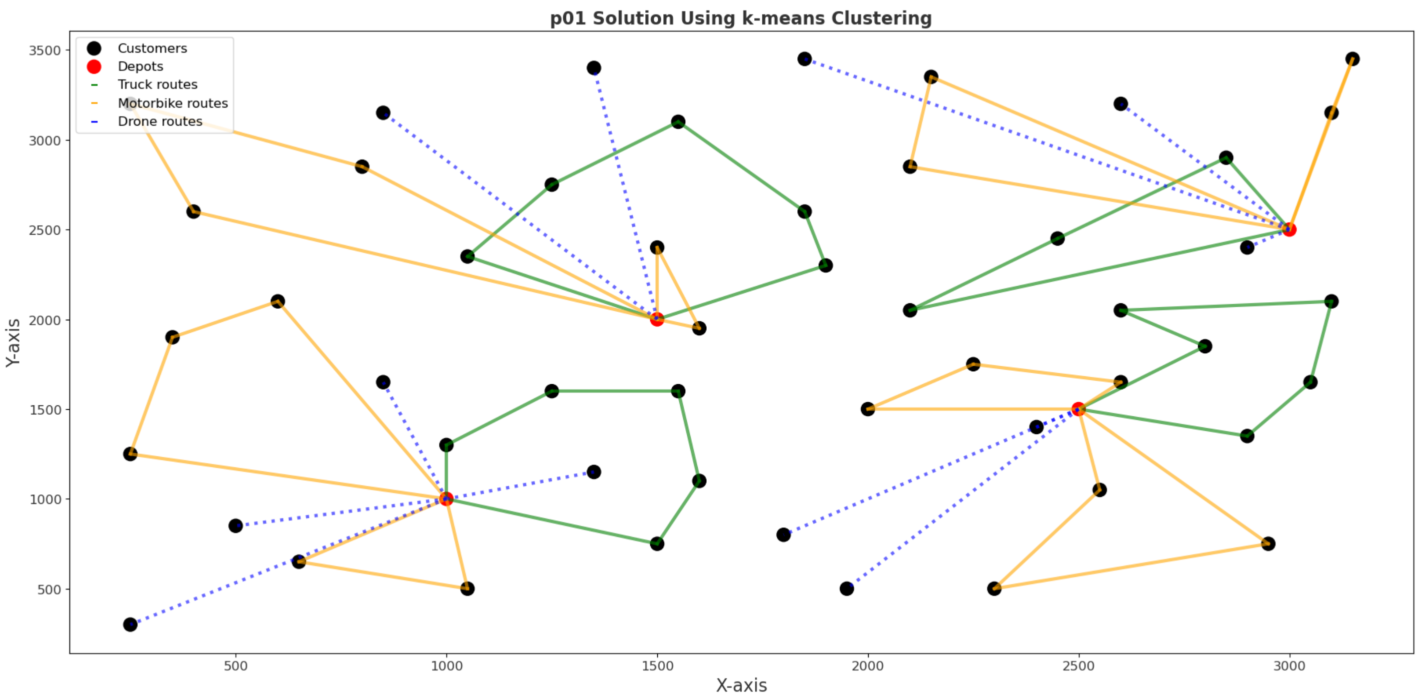
Panagiotis Zachos

June 2024

1 Multi-Depot mixed fleet capacitated multiple TSP

The problem takes as input a set of nodes, comprising of customer nodes and depot nodes. Each depot may be equipped with a heterogeneous fleet of vehicles: Trucks, Motorbikes, and Drones. Each vehicle type has a specific capacity k, representing the number of parcels it can carry. The vehicles initiate their routes from their respective depots, fully loaded with parcels, and visit customer nodes, each with a demand of one parcel. Upon completing a route, either due to no customers remaining or reaching capacity, the vehicle returns to its depot, where it is reloaded to full capacity and can be dispatched again if there are remaining customers. The aim is to visit all customers in the minimum possible time, with **no restriction on the total number of parcels available at each depot**, thus differing from traditional MDVRP constraints.

Figure 1: Solution example



1.1 Objective

The objective of the MD-mfcmTSP is to minimize the total makespan in which all customers have been served i.e min M_{total}.

Notation

- m : Number of depots
- D : Set of depots
- R_T^i : Truck route of depot $i \in D$
- R_M^i : Motorbike route of depot $i \in D$
- R_D^i : Drone route of depot $i \in D$
- M_T^i : Makespan of Truck route of depot $i \in D$
- M_M^i : Makespan of Motorbike route of depot $i \in D$
- M_D^i : Makespan of Drone route of depot $i \in D$
- $M_T = max(M_T^i, M_T^{i+1}, ..., M_T^m)$: Makespan of Trucks
- $M_M = max(M_M^i, M_M^{i+1}, ..., M_M^m)$: Makespan of Motorbikes
- $M_D = max(M_D^i, M_D^{i+1}, ..., M_D^m)$: Makespan of Drones
- $M_{total} = max(M_T, M_M, M_D)$: Total makespan

Table 1: Example

Depot	M_T	M_M	M_D	**Depot makespan**
1	3	2	1	3
2	4	3	2	4
3	1	5	2	5
4	2	6	1	**6**
Total	4	**6**	2	

2 Makespan calculations

In the case where a depot is equipped with more than 1 vehicle of either type, the makespan calculations are explained below. Assume 4 routes that need to be assigned to two trucks. Each route has a time cost associated with it. The route assignment needs to be done in such a way that the trucks' makespan is minimized. We first sort the routes based on their cost in a descending order. Then, starting from the route with the maximum cost, we assign it to the truck with the currently minimum makespan. This is done iteratively until all routes have been assigned to a vehicle.

Table 2: 4 routes that need to be assigned to a depot's 2 trucks

Route	Route cost
1	5
2	6
3	2
4	3

Table 3: Optimal split

Route	Route cost	Truck
1	5	1
2	6	2
3	2	2
4	3	1

Procedure for the assignment of 4 routes to 2 trucks of the same depot:

Table 4: Sort routes in descending order based on cost

Route	Route cost
2	6
1	5
4	3
3	2

Table 5: Iteratively assign routes to Trucks

Iteration	Route	Route cost	Truck	Truck 1 ms	Truck 2 ms
1	2	6	1	6	0
2	1	5	2	6	5
3	4	3	2	6	8
4	3	2	1	8	8
Total makespan = 8 (optimal)					

Table 6: Non-optimal assignment with routes sorted in increasing order

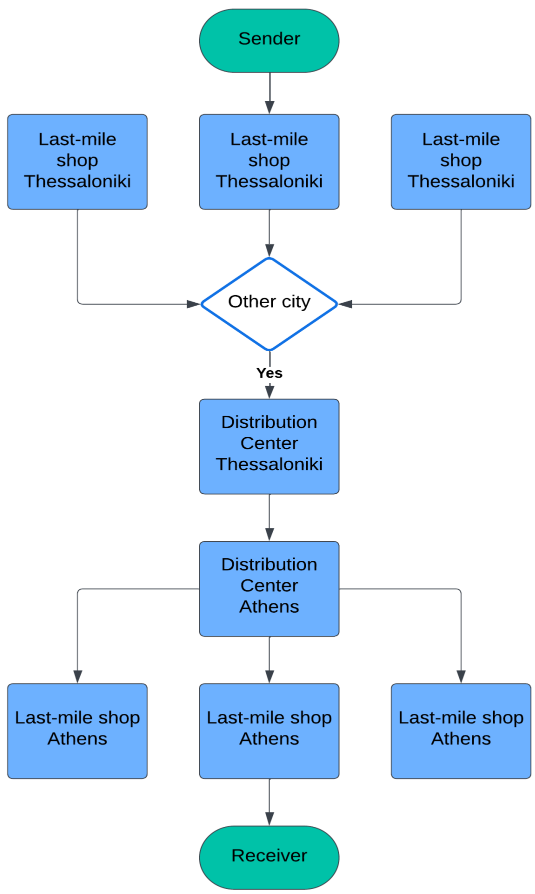
Iteration	Route	Route cost	Truck	Truck 1 ms	Truck 2 ms
1	3	2	1	2	0
2	4	3	2	2	3
3	1	5	2	7	3
4	2	6	1	7	9
Total makespan = 9 (non-optimal)					

2.1 Motivation / Use case

By introducing this problem, we contribute to the Travelling Salesman Problem and Vehicle Routing Problem research and the field of logistics and routing by introducing a realistic and challenging problem that considers the practical limitations and requirements of different vehicle types used in last-mile delivery. The proposed algorithms have the potential to improve efficiency and reduce costs for logistics companies operating in increasingly complex delivery environments.

Consider a parcel delivery company operating in Greece, with depots located in two major cities: Athens and Thessaloniki. The company manages numerous last-mile delivery shops in these cities, which serve as depots in the routing problem. For instance, a customer in Thessaloniki sends a parcel to a friend in Athens by visiting a nearby shop. After the shop stops receiving parcels for the day, a truck collects the parcels and delivers them to the Thessaloniki distribution center. Parcels destined for Athens are then transported overnight to the Athens distribution center. Upon arrival, parcels are sorted based on their delivery areas and sent to the appropriate last-mile shops in Athens. These shops, acting as depots in our problem, dispatch vehicles (Trucks, Motorbikes, Drones) to deliver the parcels to their final destinations. The MD-mfcmTSP addresses how these vehicles can be optimally routed to minimize delivery time, while simultaneously solving the problem of parcel allocation in each last-mile shop.

Figure 2: Life of a parcel



3 Problem description

- Each customer must be visited exactly once
- Each vehicle can serve at most as many customers per trip as its capacity
- Some customers may not be available for delivery via uav or larger vehicles
- Each depot must be equipped with at least one truck
- The composition of each depot's fleet is known
- Depots act as a start, finish and reload point for their fleet
- Each vehicle may perform unlimited routes (multi-trip)

4 Pseudocode for the MD-mfcmTSP

At the end of Algorithm 1, a local optimization function (3) is called which in addition to moving nodes in different places in the same route, also moves nodes between routes of different depots and different vehicle types.

Algorithm 1: MD-mfcmTSP heuristic

Input: $G_T, G_M, ..., G_D$
Output: M_{total}, $Sol = \{Sol^i = \{R^i_T, R^i_M, ..., R^i_D\}, Sol^{i+1}, ..., Sol^m\}$ for each $i \in D$
Create clusters K^i of customer nodes for each depot $d^i \in D$
by assigning each customer to the closest possible depot
for $each\ d^i \in D$ **do**
Call $Initialization(d^i, K^i)$
while $(M^i_T > M^i_M \parallel M^i_T > M^i_D)$ && $stop \neq true$ **do**
$diff_M = M^i_T - M^i_M$
$diff_D = M^i_T - M^i_D$
if $diff_M \geq diff_D$ **then**
$vt = M$
$cap =$ Motorbike's capacity
else
$vt = D$
$cap = 1$
end if
$M_{min} = M^i_T$
$r_{best} = \emptyset$
for $j = 1\ to\ |R^i_T| - cap$ **do**
$successive\_nodes = \emptyset$
$load = 0$
while $load + v_j^{demand} \leq cap$ && $v_j \in G_{vt}$ **do**
$successive\_nodes \mathrel{+}= v_j$
end while
if $|successive\_nodes| == cap$ **then**
$r_{new} = R^i_T[0] + \{successive\_nodes\} + R^i_T[0]$
$R'^i_{vt} = R^i_{vt} + r_{new}$
$M'_{vt} = R'^i_{vt}$'s makespan
$R'^i_T = R^i_T - \{successive\_nodes\}$
$M'_T = R'^i_T$'s makespan
$M_{new} = MAX(M'_T, M'_{vt})$
if $M_{new} < M_{min}$ **then**
$M_{min} = M_{new}$
$r_{best} = r_{new}$
end if
$r_{new} = \emptyset$
end if
$j \mathrel{+}= 1$
end for
if $r_{best} \neq \emptyset$ **then**
$R^i_T = R^i_T - \{r_{best}^{customers}\}$
$M_T = R^i_T$'s makespan
$R^i_{vt} \mathrel{+}= r_{best}$
$M_{vt} = R^i_{vt}$'s makespan
Call $local\_optimization(R^i_T, n_{max})$
Call $local\_optimization(R^i_{vt}, n_{max})$
else
$stop = true$
end if
end while
$Sol^i = \{R^i_T, R^i_M, ..., R^i_D\}$
end for
$M_T = MAX(M^i_T, M^{i+1}_T, ..., M^m_T)$
$M_M = MAX(M^i_M, M^{i+1}_M, ..., M^m_M)$
$M_D = MAX(M^i_D, M^{i+1}_D, ..., M^m_D)$
$M_{total} = MAX(M_T, M_M, ..., M_D)$
Call $optimization\_full(Sol, n_{max})$

Algorithm 2: Initialization(d^i, K^i)

1 **while** $\{K^i\} \cap \{G_T\} \neq \emptyset$ **do**
2     $R^i_T \mathrel{+}= NearestNeighbour(\{K^i\} \cap \{G_T\})$
3 **end while**
4 $M^i_T = R^i_T$'s makespan
5 $v_{free} = \{K^i\} - \{G_T\}$
6 **if** $v_{free} = \emptyset$ **then**
7   **return** R^i_T
8 **else**
9   **while** $v_{free} \neq \emptyset$ **do**
10   **if** $M_T - M_M \geq M_T - M_D \parallel G_D = \emptyset$ **then**
11             $R^i_M \mathrel{+}= NearestNeighbour(\{K^i\} \cap \{G_M\})$
12             $v_{free} = v_{free} - \{R^i_M\}$
13             $M^i_M = R^i_M$'s makespan
14   **else**
15             $R^i_D \mathrel{+}= closest(\{K^i\} \cap \{G_D\})$
16             $M^i_D = R^i_D$'s makespan
17   **end if**
18   **end while**
19 **end if**
20 **return** Sol^i

Algorithm 3: $optimization\_full(Sol, n_{max})$

1 Call $vt\_optimization(Sol, n_{max} = 2)$
2 **for** *each vt* **do**
3   **for** *each* $i \in D$ **do**
4   Call $local\_optimization(R^i_{vt}, n_{max} = 2)$
5   **end for**
6   Call $mutual\_depot\_optimization(R_{vt}, n_{max} = 2)$
7 **end for**
8 Call $vt\_optimization(Sol, n_{max} = 2)$

Algorithm 4: $local\_optimization(r, n_{max})$

1 **for** $n = 1$ *to* n_{max} **do**
2   **for each** *combination of n successive nodes on the route*
3   move the node(s) to a different place on the same route
4   evaluate the new route
5   **if** *this route is better than the original and all constraints are satisfied* **then**
6   replace the original route with the new one
7   continue in point 3 unless all possible places in the route have been evaluated
8   **end for**
9 **end for**
10 **return** r

Algorithm 5: $mutual\_depot\_optimization(R_{vt}, n_{max})$

1 **for** $n = 1$ *to* n_{max} **do**
2   **for each** *possible pair of depots* $c1$ *and* $c2$
3   **for each** *combination of n successive nodes in the route of c1*
4   remove the nodes from the route of c1 and insert them into c2
5   evaluate the newly-created routes
6   **if** $MAX(|R'^{c1}_{vt}|, |R'^{c2}_{vt}|) < MAX(|R^{c1}_{vt}|, |R^{c2}_{vt}|)$ *and all constraints are satisfied* **then**
7   replace the original routes with the new ones
8   continue in point 4 unless all possible places in c2 have been evaluated
9   **end for**
10   **end for**
11 **end for**
12 **return** R_{VT}

Algorithm 6: $vt\_optimization(Sol, n_{max})$

1 **for** $n = 1$ *to* n_{max} **do**
2   **for each** *depot* $i \in D$
3   **for each** *possible pair of vehicle types* $t1, t2 \in VT$
4   **for each** *combination of n successive nodes in* R^i_{t1}
5   remove the nodes from R^i_{t1} and insert them in R^i_{t2}
6   **if** $MAX(|R'^i_{t1}|, |R'^i_{t2}|) < MAX(|R^i_{t1}|, |R^i_{t2}|)$ *and all constraints are satisfied* **then**
7   replace the original routes with the new ones
8   continue in point 5 unless all possible places in R^i_{t2} have been evaluated
9   **end for**
10   **end for**
11   **end for**
12 **end for**
13 **return** Sol

Figure 3: p11 Initialization example using k-means clustering

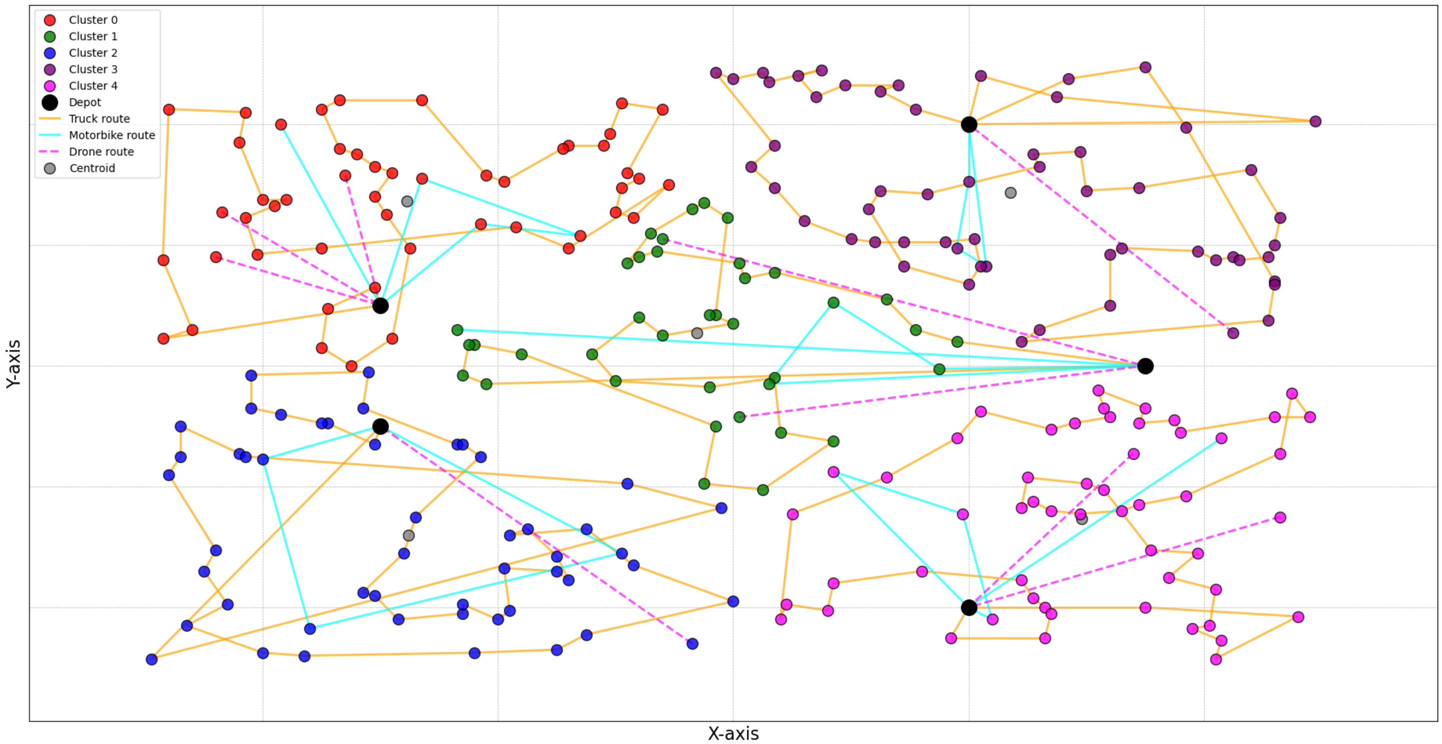


Figure 4: p01 initialization using k-means clustering

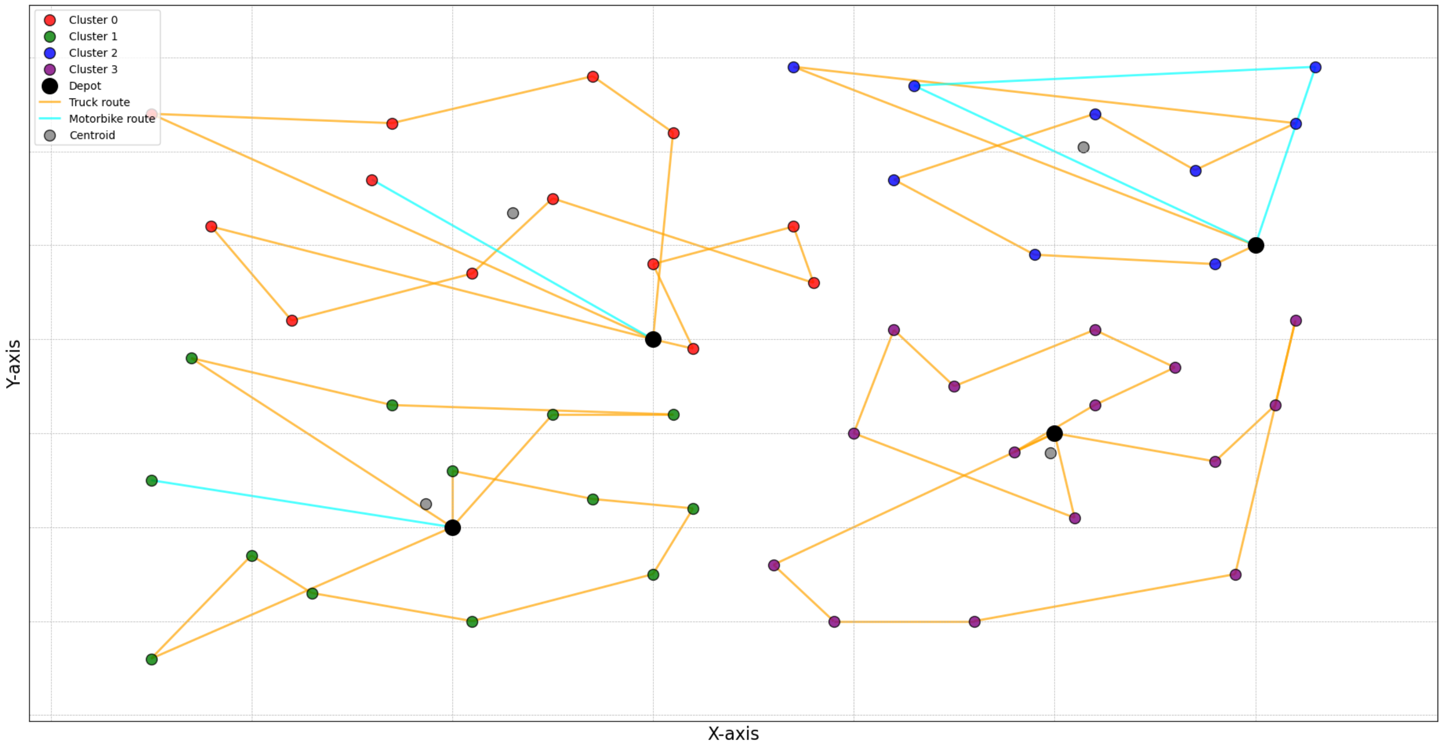
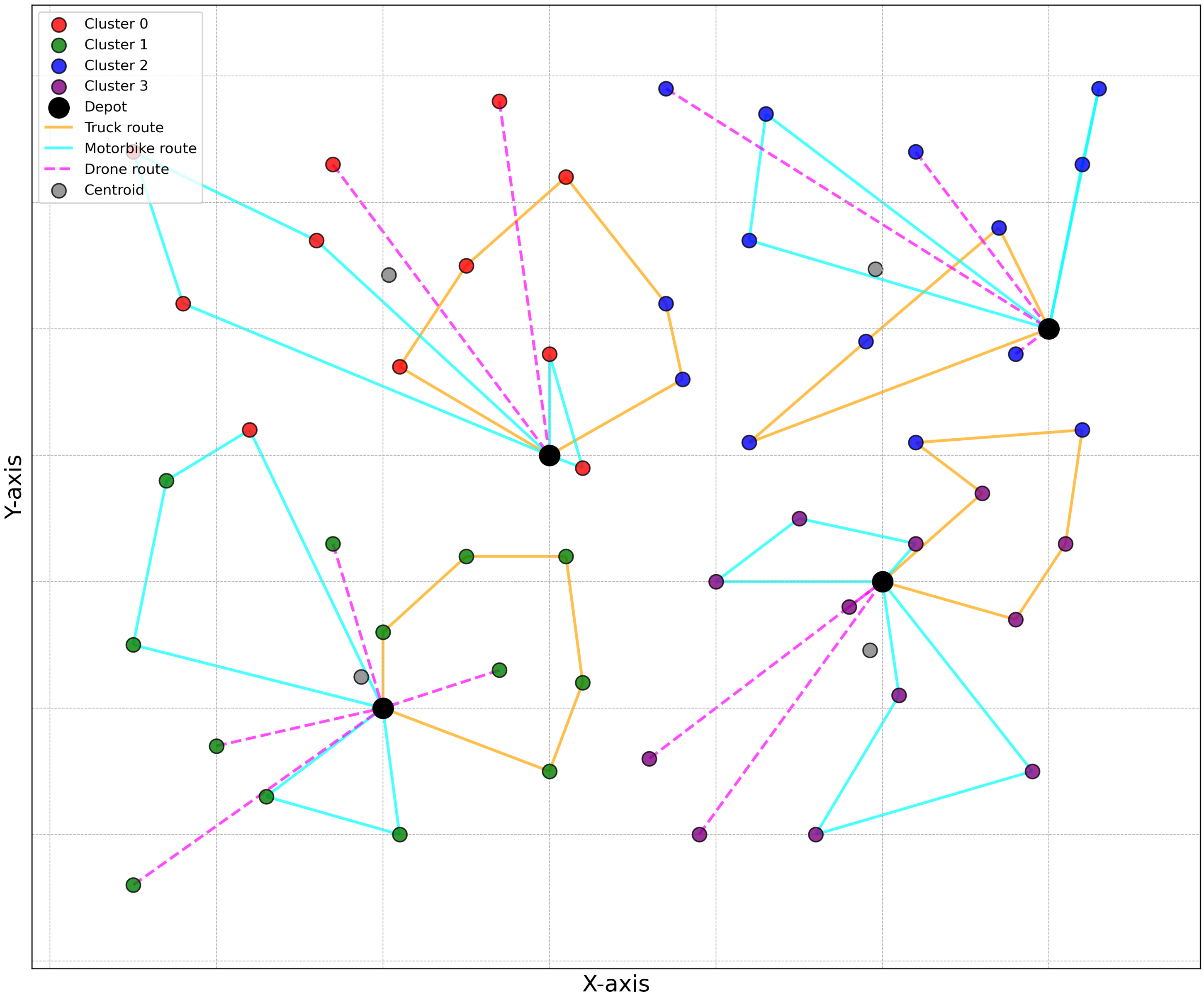


Figure 5: p01 solution using k-means clustering



5 AACONC+

Algorithm 7: AACONC+ Algorithm

Function *AACONC+*$(V, n_{ants}, n_{freq}, n_{size}, n_{sect}, n_{prim}, T_{update}, \alpha, \beta, \rho_{min}, \rho_{max}, \delta)$
  $|R| = \infty$
  $iter = 0$
  Initialize pheromone matrices τ
  for *each* $v_i \in V$ **do**
    $K(v_i) = \text{CreateClusters}(V, v_i, n_{\text{size}}, n_{\text{sect}})$
  end for
  while *not terminated* **do**
    $|R_{\text{best}}| = \infty$
    $iter = iter + 1$
    for $a = 1$ *to* n_{ants} **do**
      $R_a = \text{AntSolution}(V, K, \tau, \alpha, \beta)$
      if $|R_a| < |R_{best}|$ **then**
        $R_{\text{best}} = R_a$
      end if
    end for
    if $iter \mod n_{freq} = 0$ **then**
      $R_{\text{best}} = \text{LocalOptimization}(V, R_{\text{best}})$
    end if
    if $|R_{best}| < |R|$ **then**
      $R = R_{\text{best}}$
    end if
    Update pheromone matrices τ
    Calculate evaporation coefficient ρ
    Evaporate pheromone matrices τ using ρ
  end while
  return R

Algorithm 8: 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
    $K_{id}^{(v_i)} = K_{id}^{(v_i)} + \{v\}$
    $V_{free} = V_{free} - \{v\}$
  end for
  while $V_{free} \neq \emptyset$ **do**
    if $|K_{id}^{(v_i)}| \geq n_{size}$ **then**
      $id = id + 1$
      $K_{id}^{(v_i)} = \emptyset$
    end if
    Find closest vertex $v \in V_{free}$ to v_i
    $K_{id}^{(v_i)} = K_{id}^{(v_i)} + v$
    $V_{free} = V_{free} - v$
  end while
  return $K^{(v_i)}$

Algorithm 9: antSolution

Function *antSolution*$(V = \{D, C\}, K, \tau, \alpha, \beta)$
  $V_{free} = C$
  while $V_{free} \neq \emptyset$ **do**
    $d = \text{selectDepot}(vt, V_{free}, K^{(vt)}, \tau)$
    $vt = \text{selectVehicleType}(V_{free}, K, \tau)$
    pos = 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 $v_{load} < c^{(demand)}$ **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 $R = \{R_1^1, R_2^1, \ldots, R_2^3, R_3^3, R_D^{VT}\}$

Algorithm 10: selectVehicleType

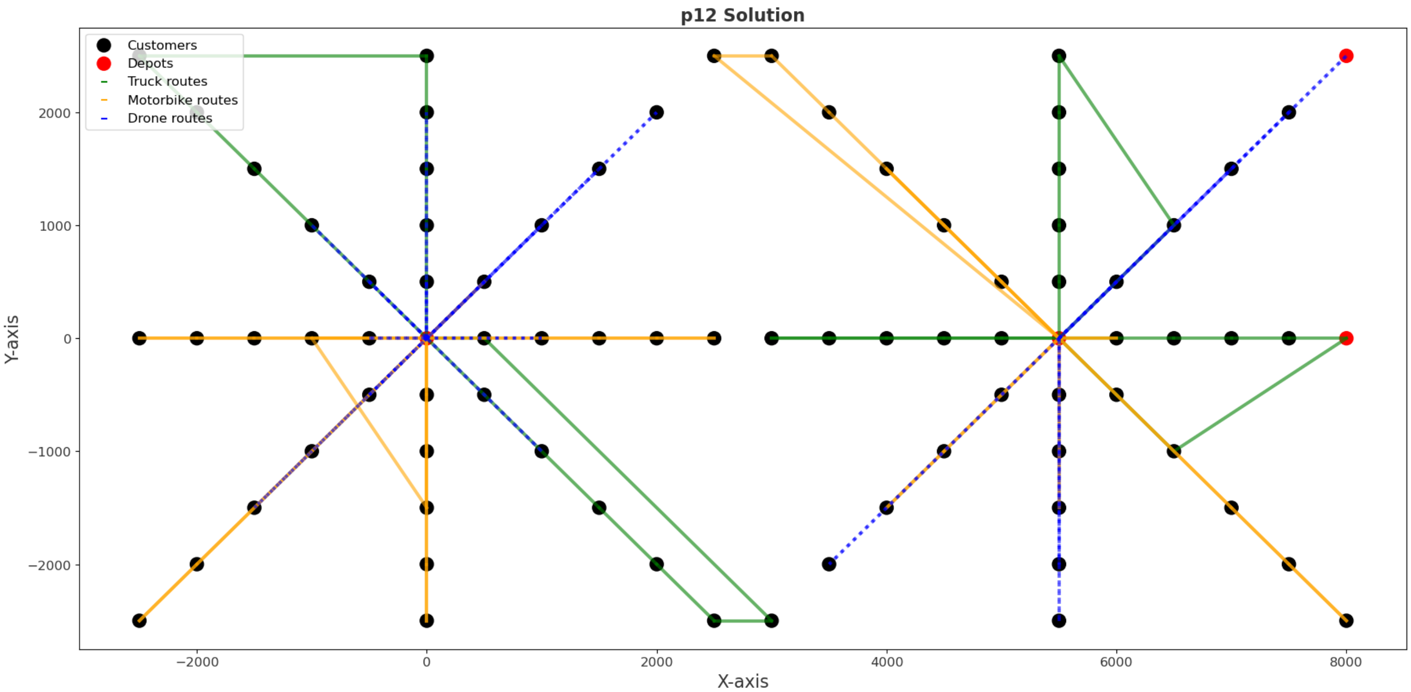
Function *selectVehicleType*($vt, d, V_{free}, K^{(vt)}, \tau$)
  for *each* $v_i \in V^{(vt)(d)}$ **do**
    $V_{\text{cand}} = \emptyset$
    pos ←vehicle's current location
    for k = *1 to* n_{prim} **do**
      $V_{cand} = V_{cand} + V_{free} \cap K_k^{(pos)(vt)}$
    end for
    $p(v_i) = \sum_{v_j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

6 Instances

Table 7: MD-mfcmTSP-C Instances

Instance	Customers	Depots	Dimensions	Truck capacity	Layout
p01	50	4	3150m x 3450m	8	Random
p02	50	4	3150m x 3450m	16	Random
p03	75	5	3500m x 3800m	14	Random
p04	100	2	3350m x 3850m	10	Random
p05	100	2	3350m x 3850m	20	Random
p06	100	3	3350m x 3850m	10	Random
p07	100	4	3350m x 3850m	10	Random
p08	249	2	9900m x 9800m	50	Random
p09	249	3	9900m x 9800m	50	Random
p10	249	4	9900m x 9800m	50	Random
p11	249	5	10500m x 5000m	50	Random
p12	80	2	10500m x 5000m	6	Regular
p13	80	2	10500m x 5000m	6	Regular
p14	80	2	10500m x 5000m	6	Regular
p15	160	4	10500m x 10500m	6	Regular
p16	160	4	10500m x 10500m	6	Regular
p17	160	4	10500m x 10500m	6	Regular
p18	240	6	16000m x 10500m	6	Regular
p19	240	6	16000m x 10500m	6	Regular
p20	240	6	16000m x 10500m	6	Regular
p21	360	9	16000m x 16000m	6	Regular
p22	360	9	16000m x 16000m	6	Regular
p23	360	9	16000m x 16000m	6	Regular

Figure 6: Regular Layout Instance



7 Results

7.1 New heuristic vs old heuristic

mfcmTSP heuristic (original) : Finds and swaps the minimum cost route in each iteration

MD-mfcmTSP heuristic (v1) : Finds and swaps the route which minimizes the depot's makespan in each iteration

Figure 7: Comparison between new and original heuristic

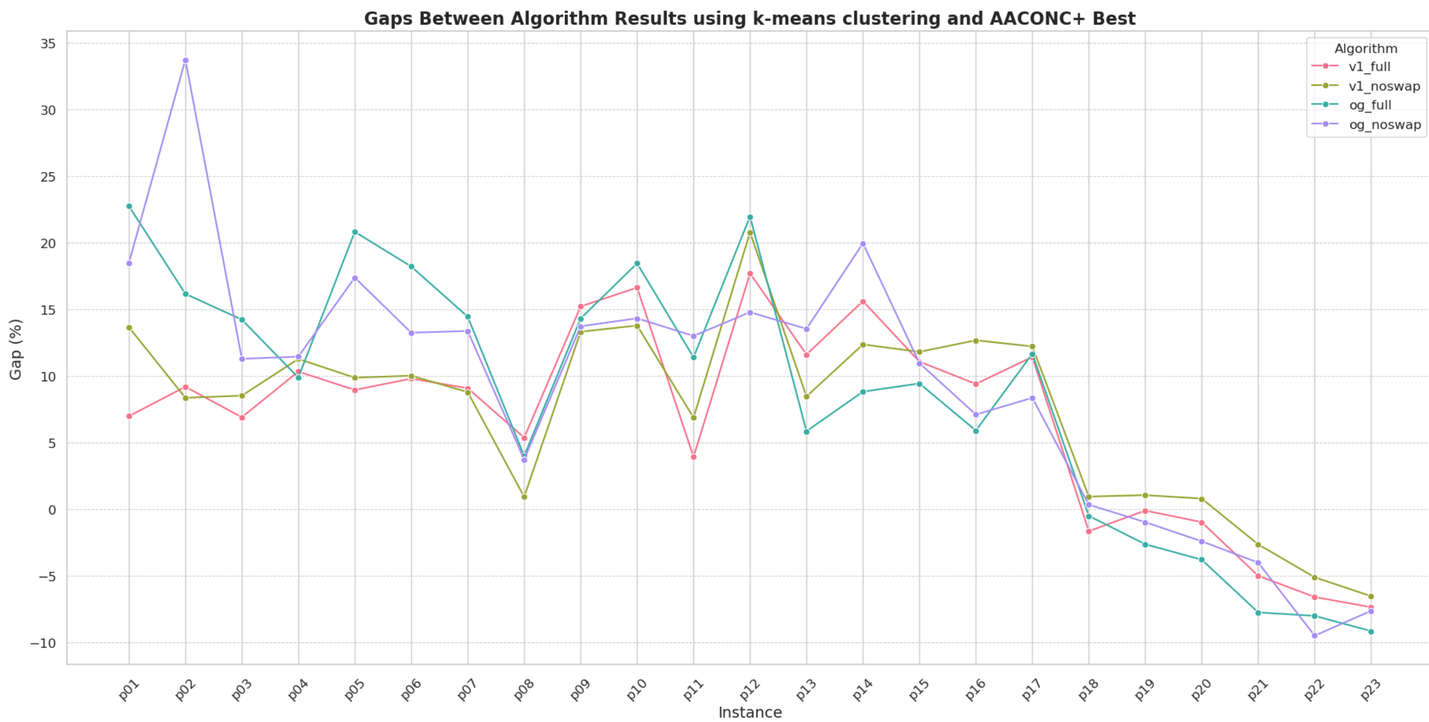


Figure 8: Comparison between new and original heuristic

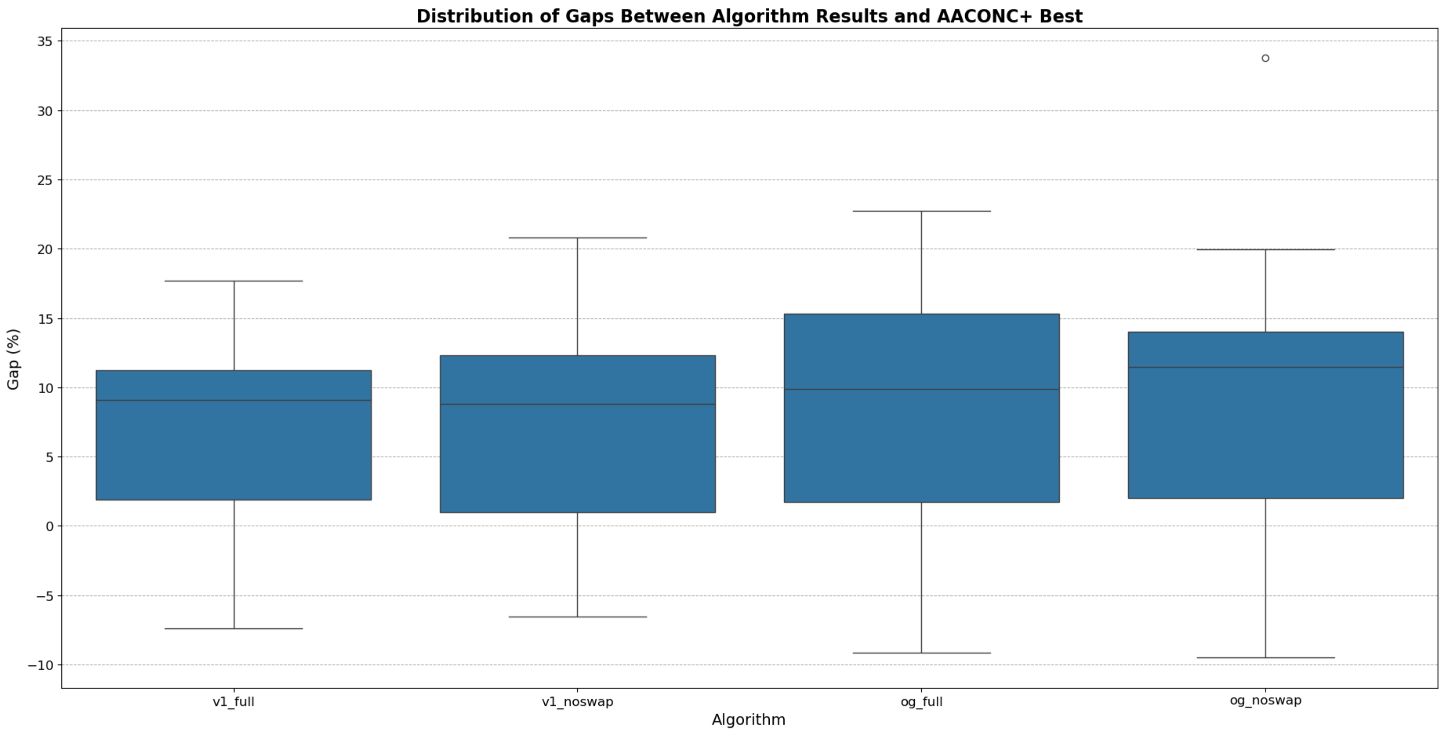
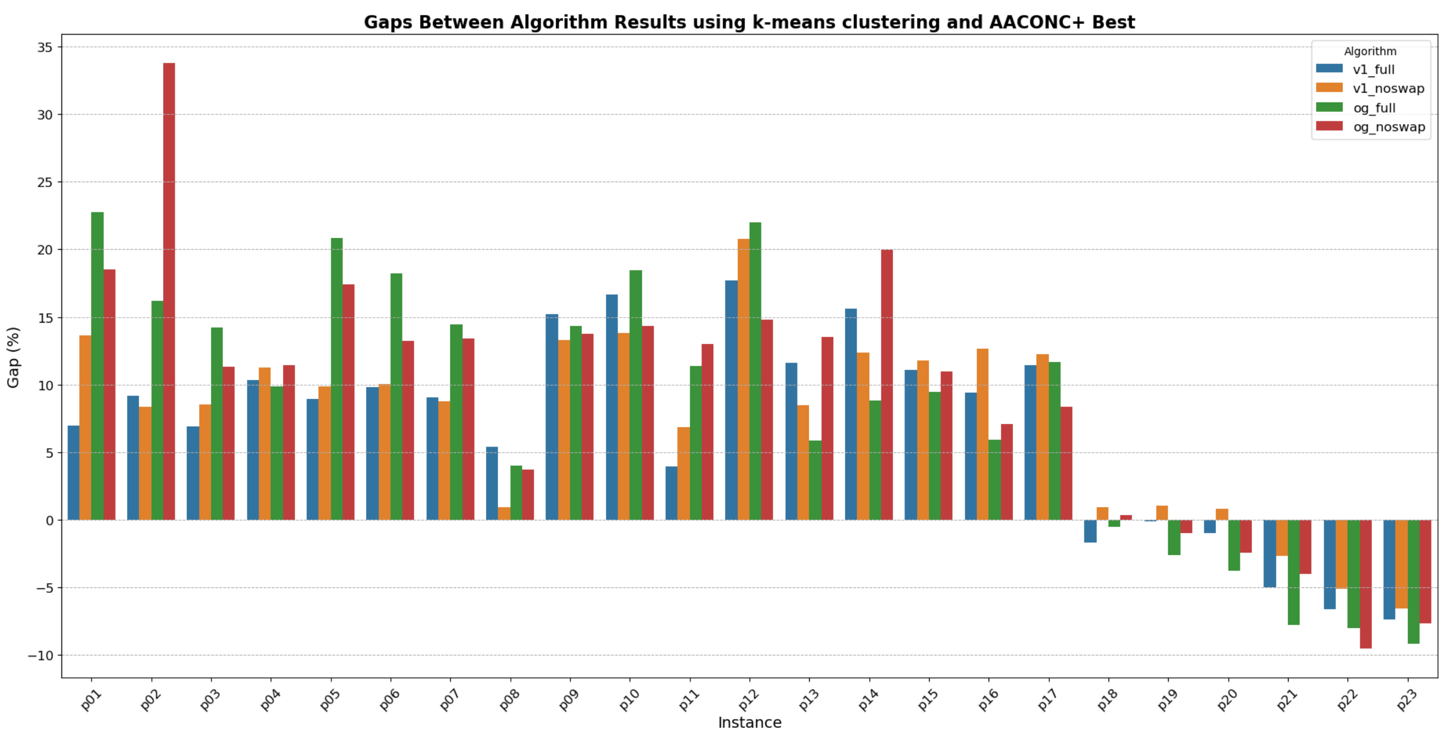


Figure 9: Comparison between new and original heuristic



8 MD-mfcmTSP Results Analysis

8.1 Local optimization impact on the heuristic

8.1.1 Proximity clustering

Table 8: Local optimization impact on heuristic using proximity clustering

Instance	Best	Full Local Opt	gap(%)	No Local Opt	gap(%)	Final Only	gap(%)	Swap Only	gap(%)
p01-C	215.15	217.42	1.06	334.04	55.26	**215.15**	0.00	173.96	27.33
p02-C	218.4	219.20	0.37	308.83	41.41	**218.40**	0.00	289.58	32.59
p03-C	202.43	214.84	6.13	253.18	25.07	**202.43**	0.00	255.52	26.23
p04-C	668.81	**668.81**	0.00	779.34	16.53	711.80	6.43	691.62	3.41
p05-C	630.78	**630.78**	0.00	726.62	15.19	655.4	3.90	697.86	10.63
p06-C	431.32	435.42	0.95	650.32	50.77	**431.32**	0.00	564.79	30.94
p07-C	309.48	**309.48**	0.00	350.77	13.34	349.19	12.83	366.51	18.43
p08-C	3079.58	3333.93	8.26	3321.63	7.86	**3079.58**	0.00	3428.15	11.32
p09-C	1917.82	**1917.82**	0.00	2429.24	26.67	2102.98	9.65	2303.70	20.12
p10-C	1493.13	**1493.13**	0.00	1864.23	24.85	1525.37	2.16	1706.14	14.27
p11-C	1101.33	1145.75	4.03	1239.36	12.53	**1101.33**	0.00	1276.59	15.91
p12-C	1273.77	**1273.77**	0.00	1356.09	6.46	1307.11	2.62	**1273.77**	0.00
p13-C	1191.96	1226.63	2.91	1455.87	22.14	**1191.96**	0.00	1259.97	5.71
p14-C	1248.53	1284.73	2.90	1413.87	13.24	**1248.53**	0.00	1302.34	4.31
p15-C	1347.67	**1347.67**	0.00	1508.50	11.93	1355.08	0.55	1375.95	2.10
p16-C	1289.29	**1289.29**	0.00	1448.53	12.35	1328.00	3.00	**1289.29**	0.00
p17-C	1282.38	1320.91	3.00	1375.55	7.27	**1282.38**	0.00	1320.91	3.00
p18-C	1260.24	**1260.24**	0.00	1446.92	14.81	1317.54	4.55	1339.83	6.32
p19-C	1266.92	**1266.92**	0.00	1406.13	10.99	1289.95	1.82	1301.62	2.74
p20-C	1323.1	**1323.10**	0.00	1429.00	8.00	1338.42	1.16	1351.39	2.14
p21-C	1309.14	1363.18	4.13	1470.03	12.29	**1309.14**	0.00	1363.18	4.13
p22-C	1295.67	**1295.67**	0.00	1465.74	13.13	1351.96	4.34	1306.6	0.84
p23-C	1252.36	**1252.36**	0.00	1396.11	11.48	1262.65	0.82	1254.42	0.16
AVG	1113.44	**1134.39**	**1.46**	1279.56	18.85	1138.07	2.34	1199.72	10.54

Figure 10: Gaps to full local optimization (proximity clustering)

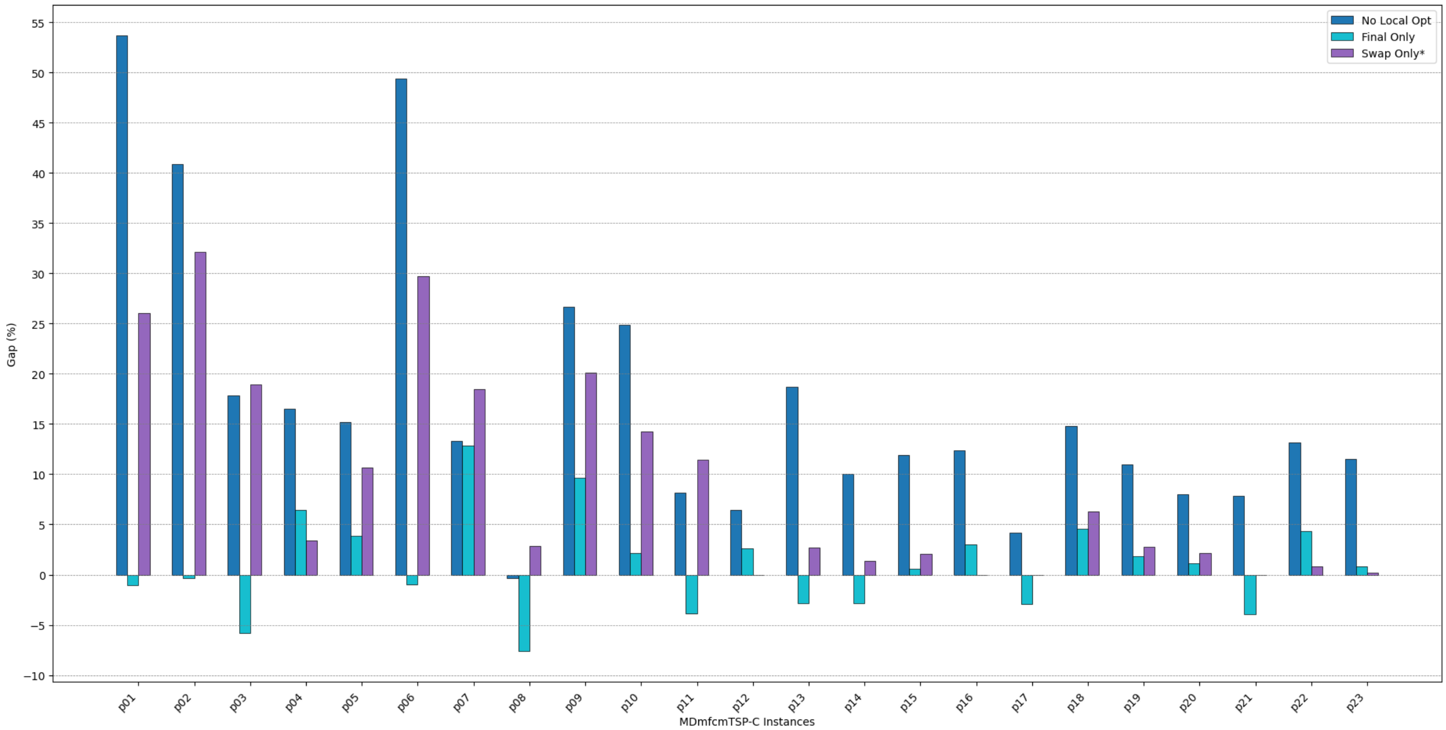
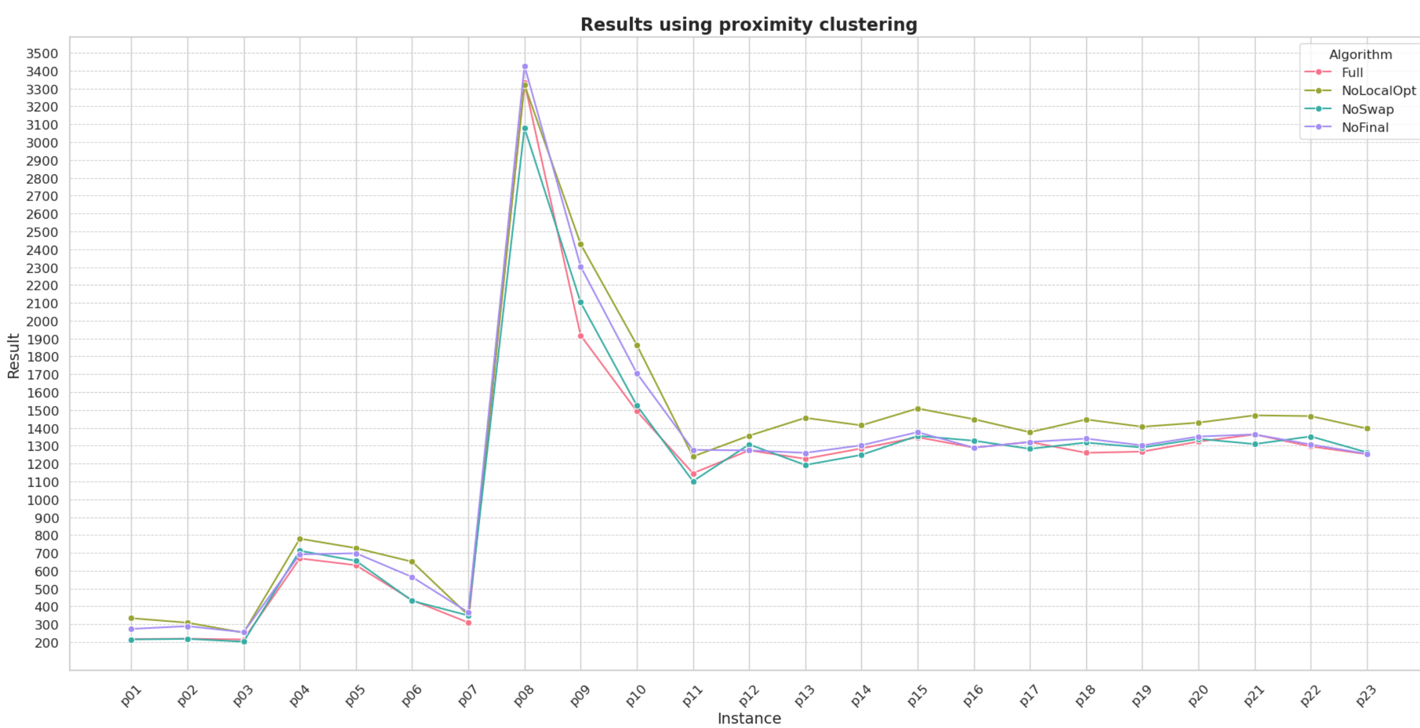


Figure 11: Results using proximity clustering



8.1.2 k-means clustering

Table 9: Local optimization impact on heuristic using k-means clustering

Instance	Best	Full Local Opt	gap(%)	No Local Opt	gap(%)	Final Only	gap(%)	Swap Only	gap(%)
p01	**191.03**	**191.03**	0.00	225.72	18.16	202.96	6.25	201.60	5.53
p02	**187.17**	188.60	0.76	209.39	11.87	**187.17**	0.00	198.02	5.80
p03	**185.82**	**185.82**	0.00	200.64	7.98	188.63	1.51	198.01	6.56
p04	**694.96**	**694.96**	0.00	807.71	16.22	700.95	0.86	**694.96**	0.00
p05	**624.33**	**624.33**	0.00	695.79	11.45	629.61	0.85	642.77	2.95
p06	**416.25**	**416.25**	0.00	492.31	18.27	417.09	0.20	456.81	9.74
p07	**313.57**	314.41	0.27	359.30	14.58	**313.57**	0.00	339.09	8.14
p08	**3051.63**	3186.02	4.40	3288.45	7.76	**3051.63**	0.00	3256.46	6.71
p09	**1932.21**	1964.49	1.67	2099.99	8.68	**1932.21**	0.00	2031.19	5.12
p10	**1463.63**	1500.38	2.51	1636.23	11.79	**1463.63**	0.00	1658.74	13.33
p11	**1031.09**	**1031.09**	0.00	1209.73	17.33	1060.37	2.84	1080.39	4.78
p12	**1273.77**	**1273.77**	0.00	1356.09	6.46	1307.11	2.62	**1273.77**	0.00
p13	**1191.96**	1226.63	2.91	1455.87	22.14	**1191.96**	0.00	1259.97	5.71
p14	**1248.53**	1284.73	2.90	1413.87	13.24	**1248.53**	0.00	1302.34	4.31
p15	**1295.23**	**1295.23**	0.00	1434.72	10.77	1303.76	0.66	**1295.23**	0.00
p16	**1289.29**	**1289.29**	0.00	1436.36	11.41	1328.00	3.00	**1289.29**	0.00
p17	**1273.39**	**1273.39**	0.00	1375.55	8.02	1282.38	0.71	1283.38	0.78
p18	**1211.56**	**1211.56**	0.00	1369.81	13.06	1243.53	2.64	1280.85	5.72
p19	**1248.93**	**1248.93**	0.00	1384.13	10.83	1263.35	1.15	1281.68	2.62
p20	**1234.12**	**1234.12**	0.00	1389.60	12.60	1256.25	1.79	1267.53	2.71
p21	**1231.23**	**1231.23**	0.00	1354.91	10.05	1261.89	2.49	1278.10	3.81
p22	**1230.99**	**1230.99**	0.00	1379.49	12.06	1250.49	1.58	1265.17	2.78
p23	**1222.63**	**1222.63**	0.00	1363.52	11.52	1233.67	0.90	1250.29	2.26
Average	**1088.84**	1100.86	0.67	1214.75	12.45	1100.81	1.31	1134.16	4.32

Figure 12: Comparison to full local optimization (k-means clustering)

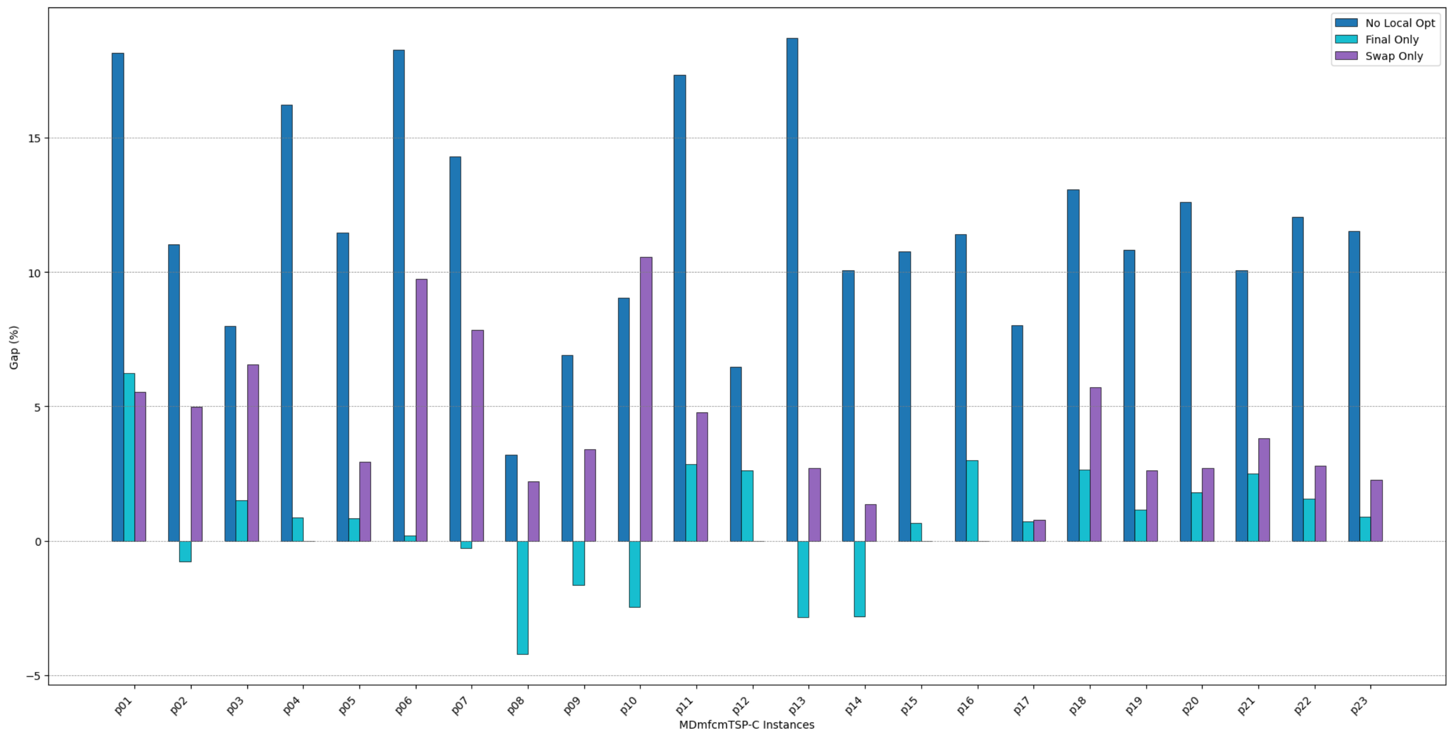


Figure 13: Results using k-means clustering

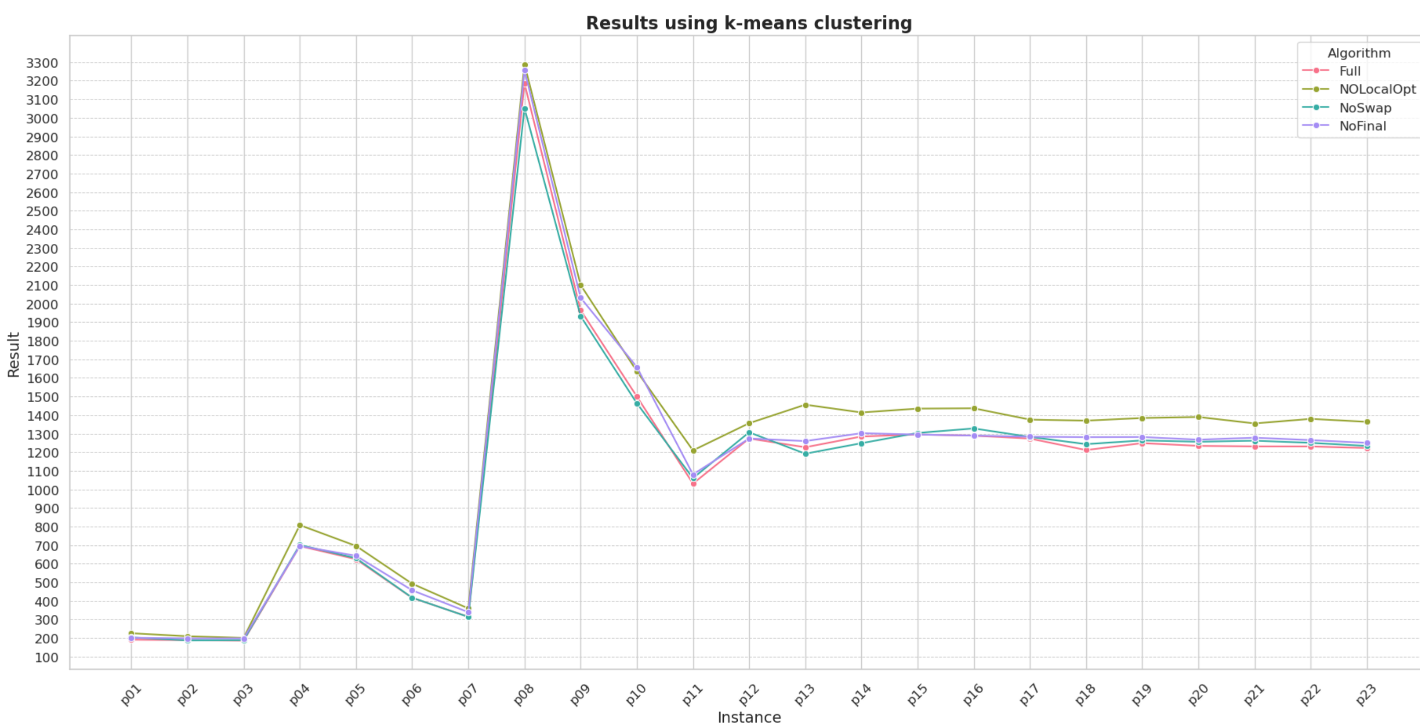
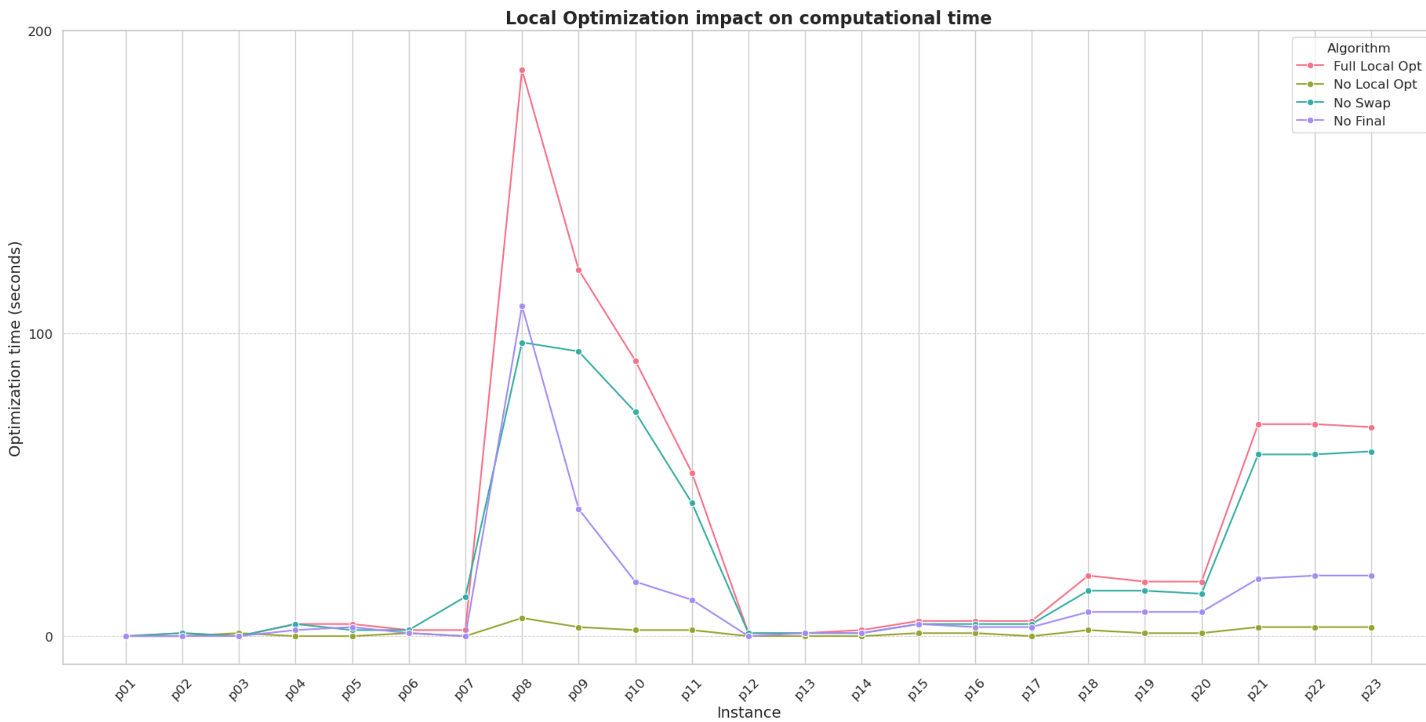


Figure 14: Local optimization impact on computational time (k-means)



8.2 AACONC+ Results and Comparison to heuristics

Table 10: AACONC+ Results

Instance	Best	Average	gap(%)	Worst	gap(%)	Average time(s)
p01-C	178.56	196.74	10.18	225.60	26.34	141
p02-C	172.73	196.57	13.8	234.78	35.92	123
p03-C	173.82	187.26	7.73	210.15	20.90	350
p04-C	629.88	644.84	2.37	675.63	7.26	1203
p05-C	573.03	613.47	7.06	692.44	20.84	884
p06-C	379.10	390.34	2.96	403.39	6.41	1316
p07-C	288.24	297.91	3.35	314.78	9.21	798
p08-C	3023.77	3100.23	2.53	3265.97	8.01	1635
p09-C	1705.03	1774.86	4.10	1846.92	8.32	3391
p10-C	1286.28	1319.78	2.60	1349.47	4.91	3549
p11-C	992.16	1035.63	4.38	1061.7	7.01	3600
p12-C	1082.12	1123.30	3.81	1175.39	8.62	446
p13-C	1099.02	1135.30	3.30	1191.96	8.46	491
p14-C	1111.12	1123.33	1.10	1170.17	5.31	423
p15-C	1166.08	1202.57	3.13	1228.27	5.33	2471
p16-C	1178.51	1209.60	2.64	1241.67	5.36	2146
p17-C	1142.74	1199.88	5.00	1253.47	9.69	2628
p18-C	1231.96	1263.90	2.59	1296.42	5.23	3368
p19-C	1250.26	1266.54	1.30	1284.23	2.72	3600
p20-C	1246.36	1266.70	1.63	1280.77	2.76	3445
p21-C	1296.19	1330.11	2.62	1347.09	3.93	3600
p22-C	1317.94	1332.07	1.07	1354.61	2.78	3600
p23-C	1319.93	1341.50	1.63	1356.62	2.78	3600
Average	1036.73	1067.50	3.95	1107.02	9.48	2035.13

Table 11: AACONC+ best and heuristic results

Instance	AACONC+	heuristic (prox.)	gap(%)	heuristic (k-means)	gap(%)
p01-C	**178.56**	217.42	21.76	191.03	6.98
p02-C	**172.73**	219.2	26.90	188.60	9.19
p03-C	**173.82**	214.84	23.60	185.82	6.90
p04-C	**629.88**	668.81	6.18	694.96	10.33
p05-C	**573.03**	630.78	10.08	624.33	8.95
p06-C	**379.10**	435.42	14.86	416.25	9.80
p07-C	**288.24**	309.48	7.37	314.41	9.08
p08-C	**3023.77**	3333.93	10.26	3186.02	5.37
p09-C	**1705.03**	1917.82	12.48	1964.49	15.22
p10-C	**1286.28**	1493.13	16.08	1500.38	16.64
p11-C	**992.16**	1145.75	15.48	1031.09	3.92
p12-C	**1082.12**	1273.77	17.71	1273.77	17.71
p13-C	**1099.02**	1226.63	11.61	1226.63	11.61
p14-C	**1111.12**	1284.73	15.62	1284.73	15.62
p15-C	**1166.08**	1347.67	15.57	1295.23	11.08
p16-C	**1178.51**	1289.29	9.40	1289.29	9.40
p17-C	**1142.74**	1320.91	15.59	1273.39	11.43
p18-C	1231.96	1260.24	2.30	**1211.56**	-1.66
p19-C	1250.26	1266.92	1.33	**1248.93**	-0.11
p20-C	1246.36	1323.10	6.16	**1234.12**	-0.98
p21-C	1296.19	1363.18	5.17	**1231.23**	-5.01
p22-C	1317.94	1295.67	-1.69	**1230.99**	-6.60
p23-C	1319.93	1252.36	-5.12	**1222.63**	-7.37
Average	**1037.52**	1134.39	11.24	1100.86	6.84

Figure 15: Comparison to the best solution found by AACONC+

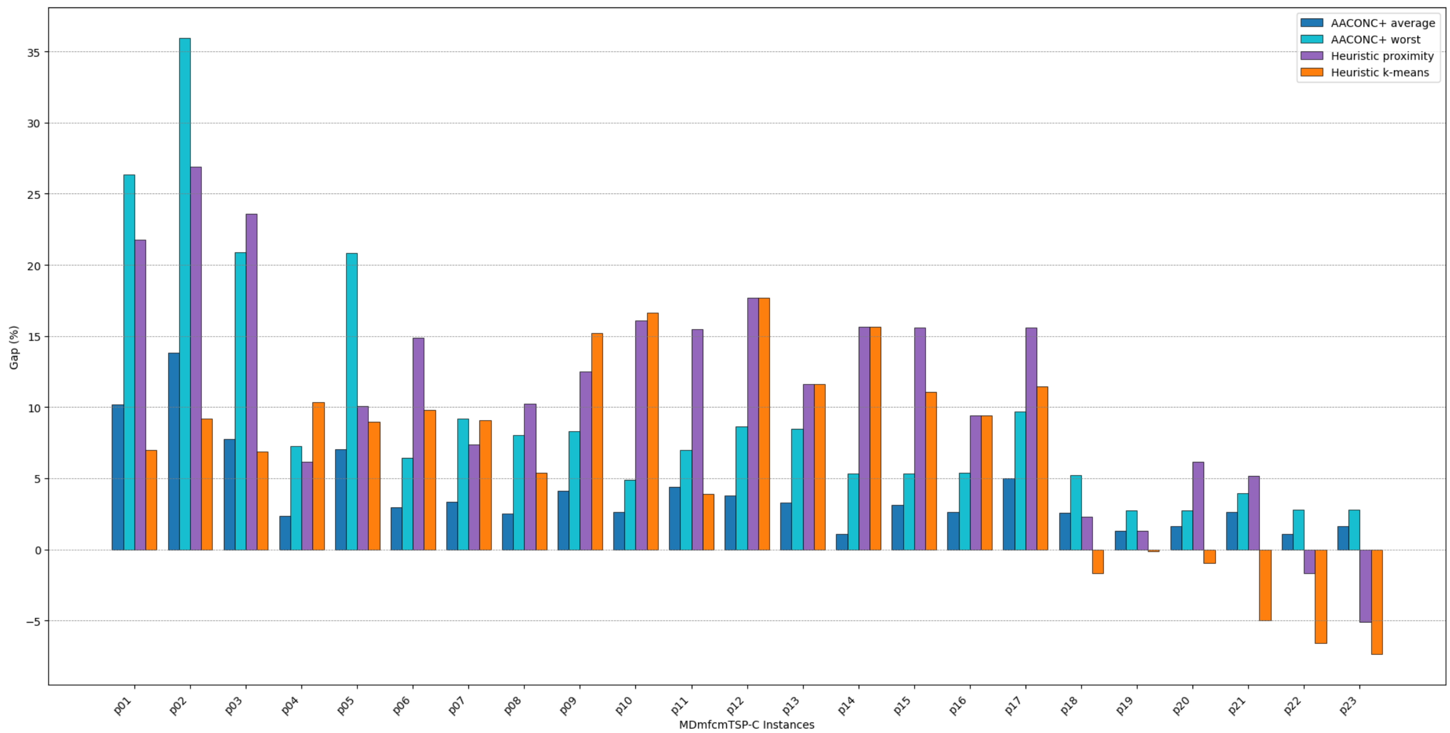


Figure 16: Comparison of best results found

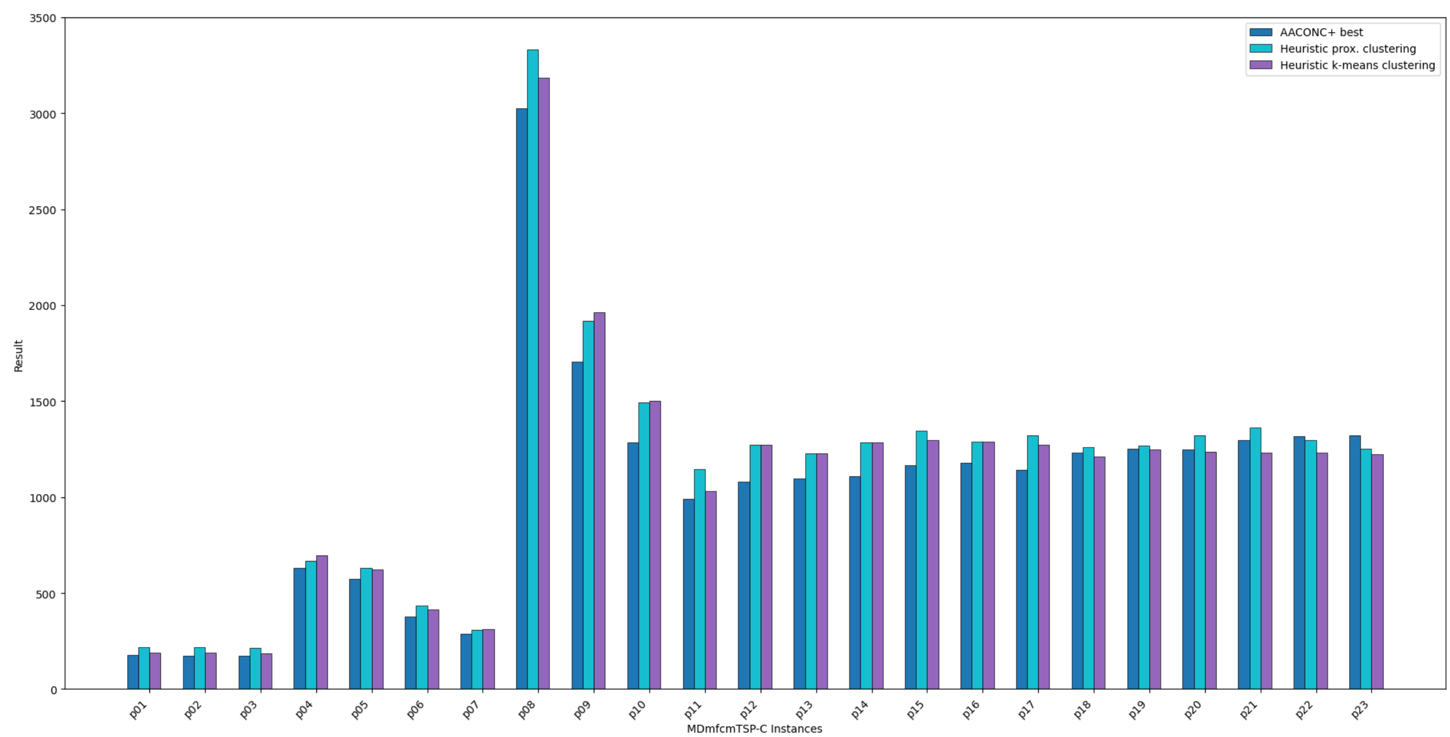


Figure 17: Comparison to the best solution found by AACONC+

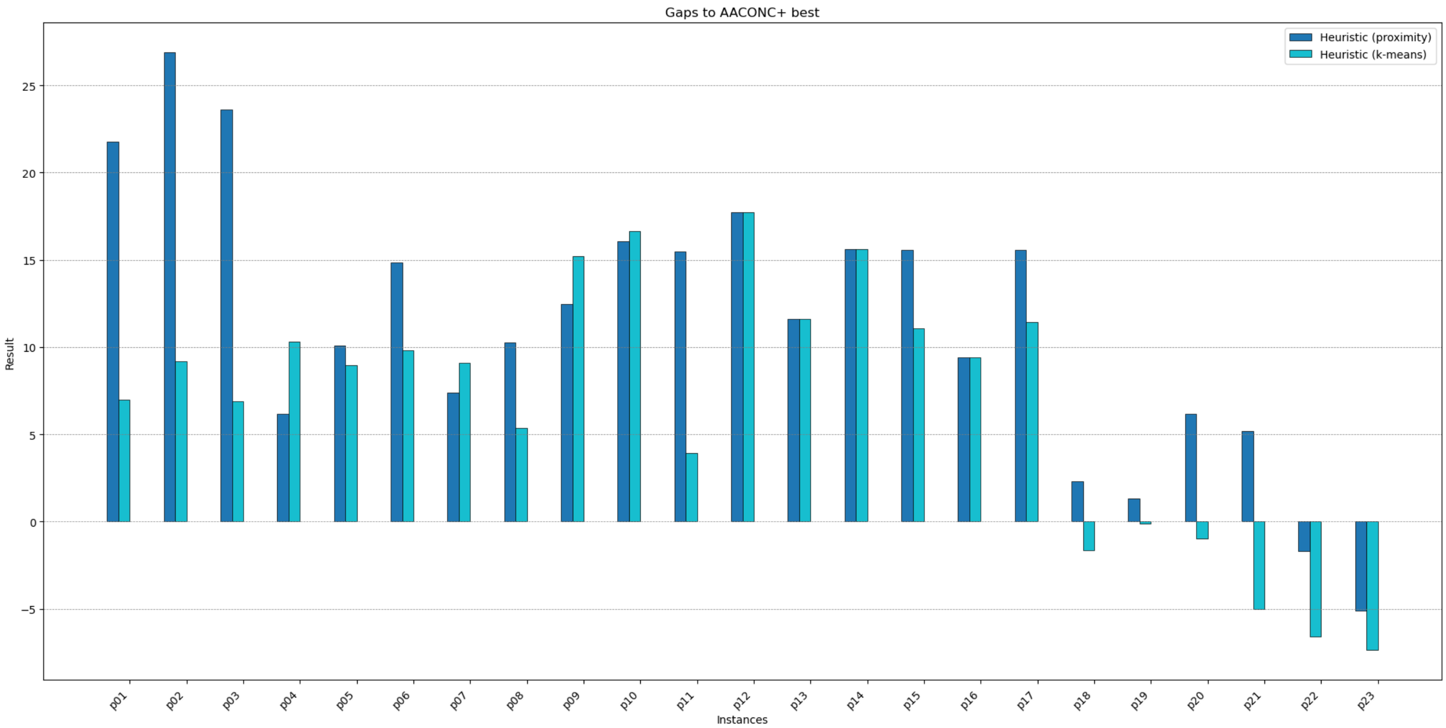


Figure 18: Comparison to the average solution found by AACONC+

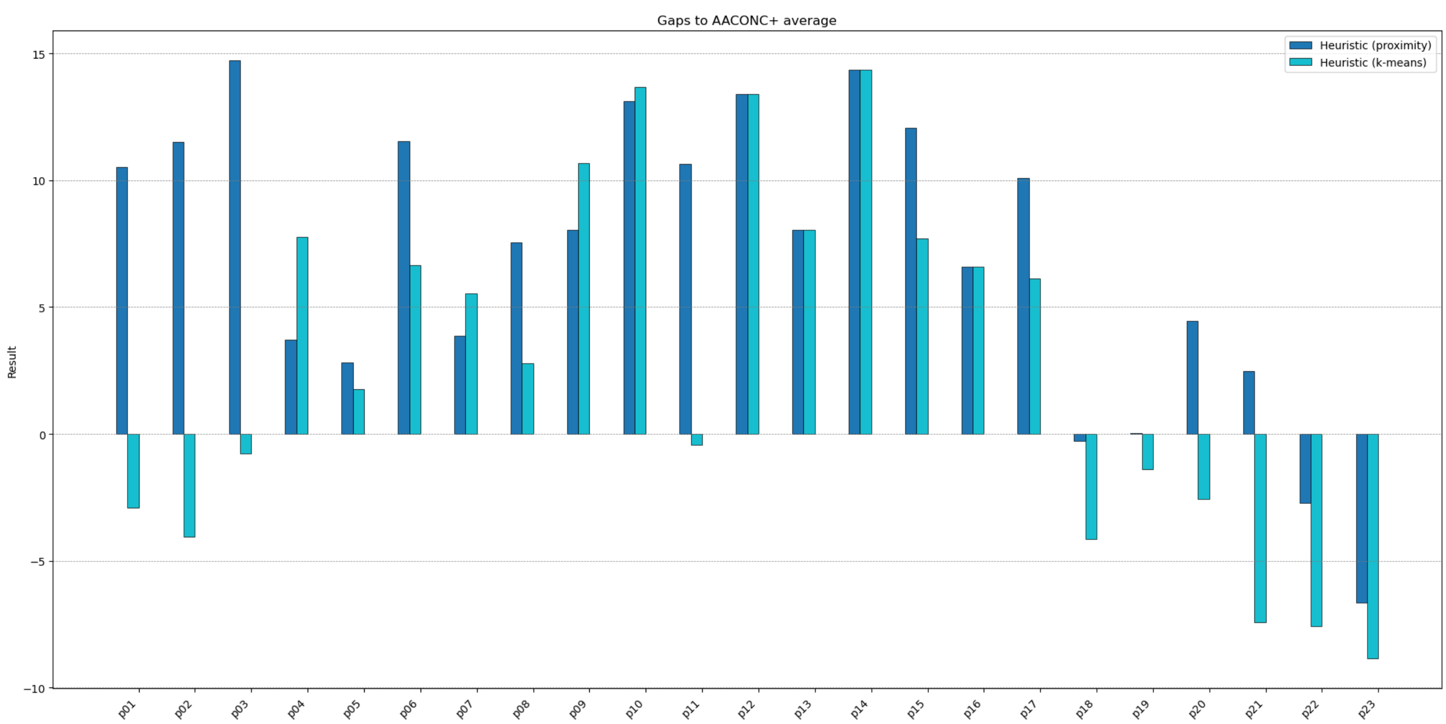


Figure 19: Comparison to the worst solution found by AACONC+

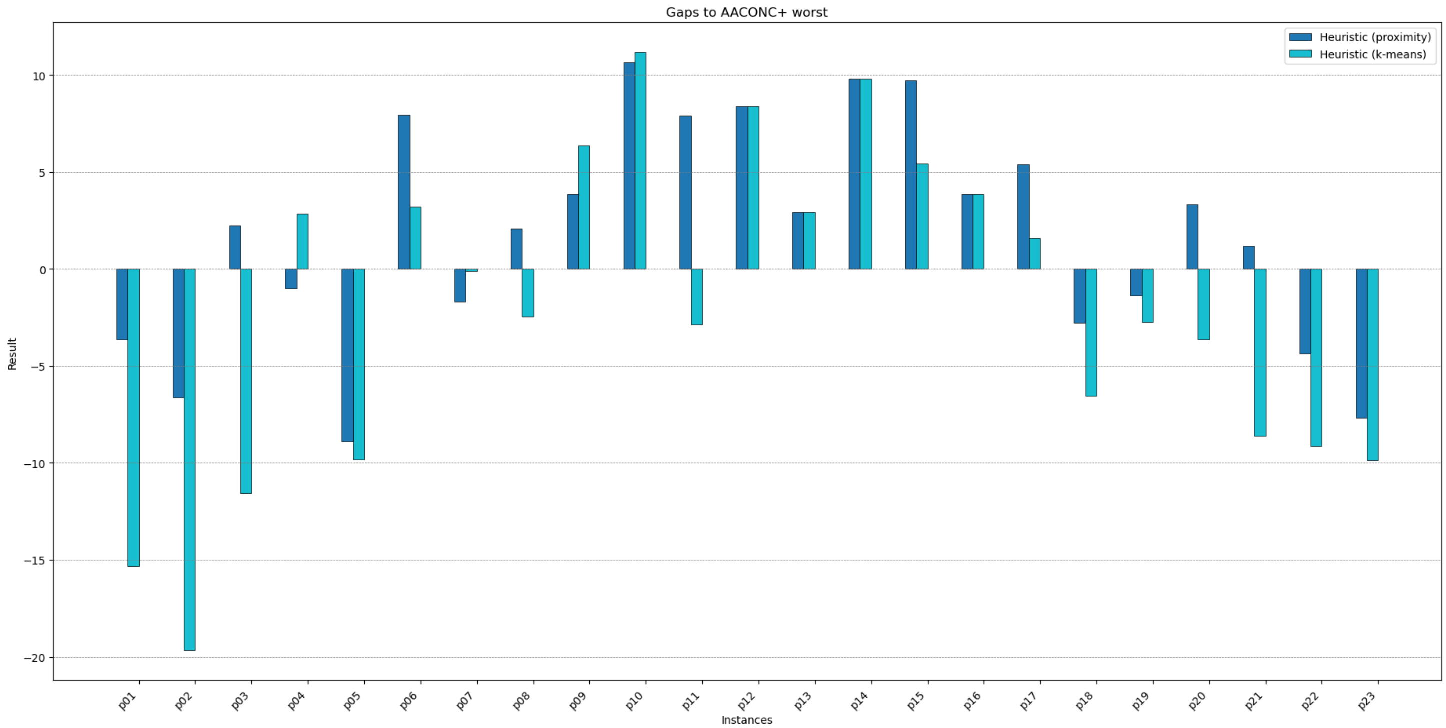
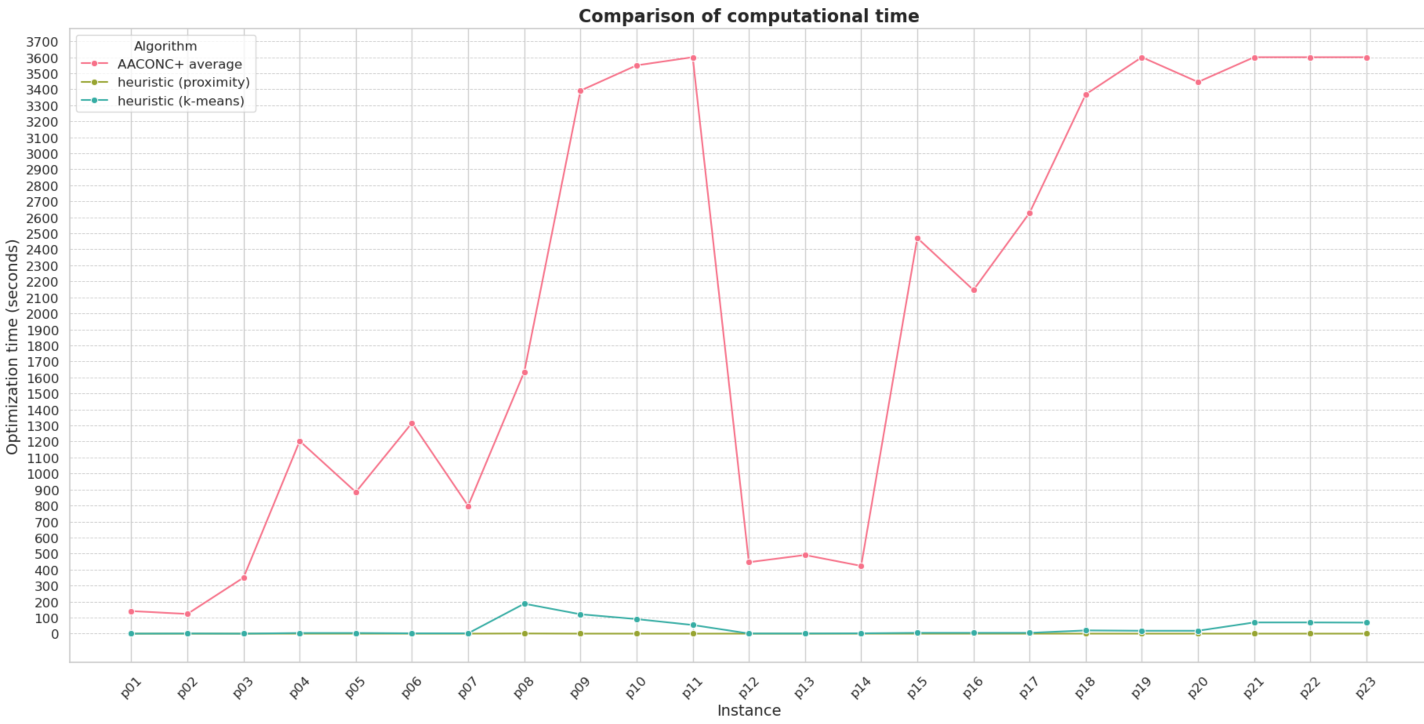


Figure 20: Comparison of best results found

Figure 21


Comparison of computational time
Algorithm
AACONC+ average
heuristic (proximity)
heuristic (k-means)
Optimization time (seconds)
Instance
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
2100
2200
2300
2400
2500
2600
2700
2800
2900
3000
3100
3200
3300
3400
3500
3600
3700
p01
p02
p03
p04
p05
p06
p07
p08
p09
p10
p11
p12
p13
p14
p15
p16
p17
p18
p19
p20
p21
p22
p23

9 Related Literature

At the same time, it is important that we address flight range limitations. Such limitations have important roles in much of the existing drone routing research. However, even he first drones developed by Amazon and JD.com ahad a flight range of 15 to 20 miles [33, 56]. Such a range is suitable to allow out and back travel in the medium-sized cities in which the authors live, Braunschweig, Germany, and Iowa City, United States. In fact, it is suitable for many larger cities such as Hamburg, Munich, and Paris. Thus, in contrast to other drone applications, in this work, we do not consider flight range as a limiting factor (Ulmer & Thomas (2018)).

Table 12: Drone-related Routing literature

Reference	Problem	Scale	Tandem	Flight endurance	Capacitated*	Solution Method
Murray & Chu (2015)	FSTSP	1-Truck 1-Drone 1-Depot	Yes	Yes	No	MILP, Heuristics
	PDSTSP	1-Truck n-Drone 1-Depot	No	Yes	No	MILP, Heuristics
Ha et al. (2015)	FSTSP	1-Truck 1-Drone 1-Depot	Yes	Yes	No	MIP, Heuristics
Freitas & Penna (2018)	FSTSP	1-Truck 1-Drone 1-Depot	Yes	Yes	No	IP, Heuristics
Boccia et al. (2021)	FSTSP/PDSTSP	1-Truck 1-Drone 1-Depot	Yes	Yes	No	MILP, Heuristics
Dell'Amico et al. (2021)	FSTSP	1-Truck 1-Drone 1-Depot	Yes	Yes	No	MILP
Dell'Amico et al. (2021)	FSTSP	1-Truck 1-Drone 1-Depot	Yes	Yes	No	Branch and bound, Heuristic
Kuroswiski et al. (2023)	FSTSP	1-Truck 1-Drone 1-Depot	Yes	Yes	No	MILP, Metaheuristics
Pilcher (2023)	FSTSP	1-Truck 1-Drone 1-Depot	Yes	Yes	No	Self-adaptive GA
Mbiadou Saleu et al. (2018)	PDSTSP	1-Truck n-Drone 1-Depot	No	Yes	No	MILP, Heuristics
Dinh et al. (2021)	PDSTSP	1-Truck n-Drone 1-Depot	No	Yes	No	Metaheuristics
Nguyen et al. (2022)	PDSTSP	1-Truck n-Drone 1-Depot	No	Yes	No	MILP
Mbiadou Saleu et al. (2022)	PDSMTSP	m-Truck n-Drone 1-Depot	No	Yes	No	MILP, Metaheuristics
Montemanni et al. (2023)	PDSTSP	1-Truck n-Drone 1-Depot	Yes	Yes	No	Constraint Programming
Nguyen et al. (2023)	PDSTSP-c	1-Truck n-Drone 1-Depot	No	Yes	No	MILP, Metaheuristics
Montemanni et al. (2024)	PDSTSP-c	1-Truck n-Drone 1-Depot	No	Yes	No	MILP, Constraint Programming
Ham (2018)	PDSTSP^{+DP}	m-Truck n-Drone 2-Depot	No	Yes	No	Constraint Programming
Agatz et al. (2018)	TSP-D	1-Truck 1-Drone 1-Depot	Yes	Yes	No	IP, Heuristics
Yurek et al. (2018)	TSP-D	1-Truck 1-Drone 1-Depot	Yes	Yes	No	MIP, Heuristics
Ha et al. (2018)	min-cost TSP-D	1-Truck 1-Drone 1-Depot	Yes	Yes	No	MILP, GRASP, TSP-LS
Dorling et al. (2016)	DDPs	0-Truck n-Drone 1-Depot	No	Yes	Yes	MILP, SA
Ulmer & Thomas (2018)	SDDPHF	m-Truck n-Drone 1-Depot	No	No	No	Adaptive dynamic programming
Salama & Srinivas (2020)	JOCR	1-Truck n-Drone 1-Depot	Yes	Yes	No	IP, MILP, Heuristics
Lu et al. (2022)	FDTSP	1-Truck n-Drone 1-Depot	Yes	Yes	No	Heuristics, Metaheuristics
Lan (2024)	TSPTWD	1-Truck 1-Drone 1-Depot	Yes	Yes	No	Metaheuristics
Wang et al. (2017)	VRPD	m-Truck n-Drone 1-Depot	Yes	Yes	Yes	Problem formulation, theoretical study
Schermer et al. (2018)	VRPD	m-Truck n-Drone 1-Depot	Yes	Yes	Yes	Heuristics
Sacramento et al. (2019)	VRP-D	m-Truck m-Drone 1-Depot	Yes	Yes	Yes*	MIP, ALNS
Schermer et al. (2019)	VRPDERO	m-Truck n-Drone 1-Depot	No	Yes	No	MILP, VNS, TS
Nguyen et al. (2022)	PDSVRP	m-Truck n-Drone 1-Depot	No	Yes	Yes	MILP, Metaheuristics
Schermer et al. (2019)	VRPD	m-Truck n-Drone 1-Depot	Yes	Yes	No	MILP, Matheuristic
Wang & Sheu (2019)	VRPD	m-Truck n-Drone 1-Depot	No	Yes	Yes	MIP, branch-and-price
Euchi & Sadok (2021)	VRP-D	m-Truck m-Drone 1-Depot	Yes	Yes	Yes	MILP, HGA
Lei et al. (2022)	VRPD	m-Truck m-Drone 1-Depot	Yes	Yes	Yes	Dynamical Artificial Bee Colony
Karak et al. (2019)	HVDRP	m-Truck n-Drone 1-Depot	Yes	Yes	Yes(Drones)	MIP, Heuristics
Kuo et al. (2022)	VRPDTW	m-Truck m-Drone 1-Depot	Yes	Yes	Yes	MIP, VNS
Stodola et al. (2024)	MDVRP-D	m-Truck m-Drone m-Depot	Yes	Yes	Yes	ACO
Oikonomou et al. (2019)	mfcmTSP	1-Truck 1-Drone 1-Depot	No	No	No	Heuristics
This paper	MD-mfcmTSP	m-Truck n-Drone k-Depot	No	No	Yes	Metaheuristics, Heuristics