DM872 – Mathematical Optimization at Work

Answers to Obligatory Assignment 2, Spring 2025

Domonkos Kertész

29.05.2025.

All files are available on my GitHub

1 Determining Solution

Problem Description A delivery company must satisfy orders for 200 retail stores. Each store has a demand that must be served by exactly one truck, and trucks have uniform capacity $Q = 12\,600$ kg. Stores specify *time windows* during which they can receive shipments; some have multiple disjoint intervals. Travel distances are Euclidean based on provided coordinates, and trucks travel at 1000 distance-units per hour. Service times at each customer are also given in 24h format, and trucks may wait if they arrive early.

Problem Formulation Formulating the VRPTW as a single-commodity flow MIP with:

- Binary variables $x_{ij} \in \{0, 1\}$ indicating whether a vehicle travels from node i to j.
- Continuous variables t_i for service start times at node i.
- Continuous variables l_i for cumulative load after servicing node i.
- Continuous ordering variables u_i (MTZ) to eliminate subtours.
- Binary variables y_i indicating if customer i is served (allows skipping with penalty).

Objective

$$\min \sum_{i \neq j} d_{ij}\, x_{ij} \;+\; P \sum_{i=1}^{n} (1 - y_i)$$

where d_{ij} is Euclidean distance and P is a large penalty for skipping.

Constraints

1. **Routing continuity:** $\sum_j x_{ij} = \sum_j x_{ji} = y_i$ for all customers i.
2. **Depot:** At least one departure and return arc; no self-loop at depot.
3. **Time windows:** $e_i \leq t_i \leq \ell_i$ for consolidated interval $[e_i, \ell_i]$.
4. **Service propagation:** If $x_{ij} = 1$, then

   $$t_j \geq t_i + s_i + \frac{d_{ij}}{v} - M\,(2 - x_{ij} - y_i - y_j),$$

   where s_i is service time and $v = 1000/60$ units/minute.
5. **Capacity:** $l_0 = 0$, $l_i \geq q_i\, y_i$, $l_i \leq Q$, and if $x_{ij} = 1$:

   $$l_j \geq l_i + q_j - Q\,(2 - x_{ij} - y_i - y_j).$$
6. **Subtour elimination (MTZ):** $1 \leq u_i \leq n - 1$ and

   $$u_i - u_j + (n - 1)\, x_{ij} \leq n - 2 + n\,(2 - y_i - y_j).$$

Solution Methodology I've adopted the Miller–Tucker–Zemlin formulation for subtour elimination, augmented with big-M constraints to enforce time-window and load propagation only when an arc is used and both endpoints are served.

2 Implementing solution

The implementation is written in Python using Gurobi 12.0.1 as a solver.

- `load_data()`, `merge_data()` and `compute_matrices()` parse and preprocess coordinate, demand, service-time, and raw time-window files into consolidated DataFrames and distance/travel-time matrices.
- `consolidate_time_windows()` merges potentially multiple intervals per customer into a single $[e_i, \ell_i]$ by taking the global minimum start and maximum end.
- `check_capacity()` asserts no single demand exceeds $Q = 12\,600$.
- `build_model()` creates Gurobi variables, objective, and all constraints as outlined above.
- `solve_and_extract()` runs with a 300 s time limit, prints the total penalized distance, reconstructs routes by following variables x_{ij}, and outputs per-vehicle routes.

This code is fully contained in `asg2.py`, it is not included in the report itself because I couldn't get TeXworks to render it.

Pseudo code for each function.

Algorithm 1: `time_to_minutes`

Data: time_str: string formatted as "HH:MM"
Result: m: integer minutes since midnight

initialization;
split time_str on ":" into h, m';
$h \leftarrow \text{int}(h)$; $m' \leftarrow \text{int}(m')$;
$m \leftarrow h \times 60 + m'$;
return m;

Algorithm 2: `load_data`

Data: Paths: `coord_path`, `demand_path`, `service_path`, `tw_path`
Result: Tuple of DataFrames (df_coordinates, df_demands, df_service_times, df_time_windows_raw)

initialization;
`// Define file paths and load each into a DataFrame`
foreach *path in* {*coord_path, demand_path, service_path, tw_path*} **do**
    load text file at `path` into a DataFrame with appropriate column names;
apply `time_to_minutes` to the `start` and `end` columns of `df_time_windows_raw`;
return the four DataFrames;

Algorithm 3: `merge_data`

Data: DataFrames `df_coordinates`, `df_demands`, `df_service_times`
Result: Merged DataFrame `df` sorted by `id`

initialization;
$df \leftarrow$ `df_coordinates`;
merge df with `df_demands` on `id`;
merge result with `df_service_times` on `id`;
sort df by `id` ascending;
reset row indices of df;
return df;

Algorithm 4: `compute_matrices`

Data: DataFrame `df_instances` with columns `x`, `y`
Result: Matrices (D, T): distances and travel times

initialization;
extract coordinates array $C \leftarrow$ `df_instances[['x','y']].to_numpy()`;
for $i \leftarrow 0$ **to** $n-1$ **do**
    for $j \leftarrow 0$ **to** $n-1$ **do**
        $D[i][j] \leftarrow \text{EuclideanDistance}(C[i], C[j])$;
$v \leftarrow 1000/60$;
`// speed in units/minute`
forall i, j **do**
    $T[i][j] \leftarrow D[i][j]/v$;
return (D, T);

Algorithm 5: `consolidate_time_windows`

Data: Raw time windows `df_time_windows_raw`, array of node IDs `ids`
Result: List `TW` of consolidated (start, end) per node

initialization;
foreach *i in* `ids` **do**
    $raw[i] \leftarrow$ empty list;
`// Group intervals by node`
foreach *row in* `df_time_windows_raw` **do**
    append (row.start, row.end) to raw[row.id];
foreach *i in* `ids` **do**
    $S \leftarrow \min\{s : (s, e) \in raw[i]\}$;
    $E \leftarrow \max\{e : (s, e) \in raw[i]\}$;
    append (S, E) to `TW`;
return `TW`;

Algorithm 6: `check_capacity`

Data: DataFrame `df_instances` with column `demand`, capacity C
Result: Assertion that no single demand exceeds C

initialization;
if *there exists a demand d in* `df_instances.demand` *such that* $d > C$ **then**
    raise AssertionError("Customer demand exceeds capacity");

Algorithm 7: `build_model`

Data: Parameters: n, D, T, `df_instances`, `TW`, C, M, P
Result: Gurobi Model $\mathcal{M}$ and variable dict `arc_used`

initialization;
create Gurobi model $\mathcal{M}$ named "VRPTW";
add binary vars `arc_used`$[i, j]$ for all $i, j \in [0..n)$;
add continuous vars `arrival`$[i]$, `load`$[i]$, `mtz`$[i]$;
add binary vars `served`$[i]$ for $i \in [1..n)$;
set objective: minimize $\sum D[i][j] \cdot$ `arc_used`$[i, j] + P \sum(1 -$ `served`$[i])$;
// Routing continuity
for $i = 1$ **to** $n - 1$ **do**
    add constraint $\sum_j$ `arc_used`$[j, i] =$ `served`$[i]$;
    add constraint $\sum_j$ `arc_used`$[i, j] =$ `served`$[i]$;
// Depot constraints
add constraint $\sum_{j=1}^{n-1}$ `arc_used`$[0, j] \geq 1$;
add constraint $\sum_{i=1}^{n-1}$ `arc_used`$[i, 0] \geq 1$;
set `arc_used`$[0, 0].ub \leftarrow 0$;
// Time windows
for $i = 0$ **to** $n - 1$ **do**
    $(s, e) \leftarrow$ `TW`$[i]$;
    add constraint `arrival`$[i] \leq e$;
add constraint $0 \leq$ `arrival`$[0] \leq M$;
// Capacity
add constraint `load`$[0] = 0$;
for $i = 1$ **to** $n - 1$ **do**
    add constraint `load`$[i] \geq$ `df_instances.demand`$[i] \cdot$ `served`$[i]$;
    add constraint `load`$[i] \leq C$;
// Propagation along arcs
for $i = 1$ **to** $n - 1$ **do**
    for $j = 1$ **to** $n - 1$, $j \neq i$ **do**
        add time constraint with big-M condition;
        add load constraint with big-M condition;
// MTZ subtour elimination
for $i = 1$ **to** $n - 1$ **do**
    add constraints $1 \leq$ `mtz`$[i] \leq n - 1$;
    for $j = 1$ **to** $n - 1$ **do**
        add MTZ constraint with arc and served variables;
return $(\mathcal{M},$ `arc_used`$)$;

Algorithm 8: `solve_and_extract`

Data: Gurobi Model $\mathcal{M}$, `arc_used`, distance matrix D
Result: Printed routes or infeasibility report

initialization;
set solver time limit to 300 seconds;
$\mathcal{M}$.optimize();
if $\mathcal{M}.status \in \{OPTIMAL, TIME\_LIMIT\}$ **then**
    print total distance;
    identify all routes by tracing `arc_used`$[0, j]$ from depot;
    foreach *route found* **do**
        compute route length using D;
        print vehicle route and length;
else
    if $\mathcal{M}.status = INFEASIBLE$ **then**
        print "Model infeasible"; compute and write IIS file;
    else
        print "No solution found.";

Computational Results Running on a 4-thread Intel i7-6500U, the solver explored 1 635 nodes in 300 s, producing:

- **Total distance (with penalties):** 142 479.83
- **Number of vehicles:** 51
- **Optimality gap:** 85.98%

Vehicle routes (distance in units):

Table 1: Vehicle distances and routes

Vehicle	Distance	Route sequence
1	612.96	[0,4,76,41,0]
2	2 649.06	[0,6,1,9,23,0]
3	6 359.65	[0,8,15,28,71,86,0]
4	1 346.87	[0,13,52,64,62,0]
5	607.83	[0,17,54,55,0]
6	2 085.12	[0,21,34,5,2,0]
7	480.13	[0,22,38,27,10,0]
8	898.94	[0,24,11,0]
9	1 691.11	[0,25,20,3,14,16,43,0]
10	604.76	[0,31,48,33,0]
11	1 801.11	[0,44,35,37,56,0]
12	1 129.63	[0,51,42,57,0]
13	2 575.77	[0,53,40,45,0]
14	6 502.11	[0,59,26,29,18,0]
15	2 467.99	[0,61,46,36,39,47,0]
16	3 890.60	[0,63,30,32,0]
17	2 744.21	[0,68,65,88,112,0]
18	1 687.12	[0,78,60,50,0]
19	3 276.73	[0,79,70,87,129,0]
20	2 272.79	[0,81,69,73,100,0]
21	5 763.38	[0,83,80,82,170,0]
22	1 040.41	[0,89,84,77,58,0]
23	1 718.63	[0,90,111,94,85,0]
24	1 317.74	[0,95,103,91,66,7,0]
25	2 960.83	[0,97,96,147,138,0]
26	1 164.03	[0,99,105,92,74,0]
27	1 879.24	[0,102,107,93,0]
28	2 624.73	[0,104,110,116,0]
29	4 515.66	[0,109,67,19,106,0]
30	1 509.56	[0,113,121,120,98,0]
31	2 497.87	[0,114,124,141,145,146,125,0]
32	3 555.21	[0,122,123,117,135,0]
33	2 981.01	[0,128,134,126,0]
34	2 879.64	[0,137,130,133,139,0]
35	1 543.70	[0,144,154,148,0]
36	1 594.31	[0,149,140,142,72,0]
37	2 598.91	[0,155,156,152,0]
38	3 258.30	[0,157,132,101,115,162,0]
39	2 855.07	[0,158,160,150,108,0]
40	5 460.43	[0,159,136,119,49,12,0]
41	1 914.80	[0,164,175,153,75,0]
42	4 922.77	[0,167,151,118,131,0]
43	5 083.99	[0,169,163,143,127,182,0]
44	4 485.79	[0,176,168,161,179,0]
45	4 208.37	[0,177,174,165,0]

Continued on next page

Table 1 – continued from previous page

Vehicle	Distance	Route sequence
46	3 225.93	[0,178,192,181,171,0]
47	4 343.46	[0,185,173,187,0]
48	3 643.99	[0,186,193,184,172,166,0]
49	3 982.62	[0,188,183,180,0]
50	3 627.95	[0,197,190,195,196,198,0]
51	3 636.99	[0,199,189,191,194,0]

3 Visualization of Results

Figure 1 shows the combined solution. Figures for each route can be found bellow.

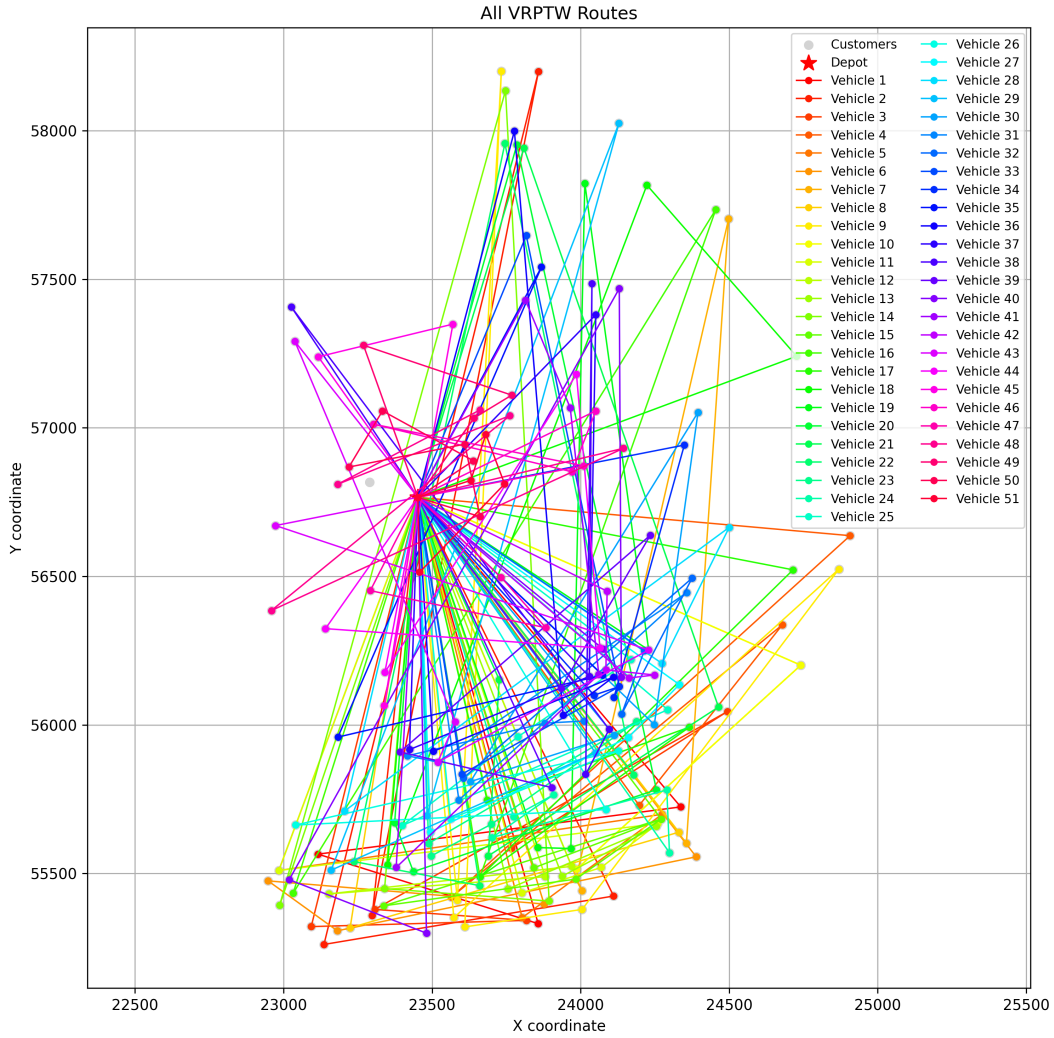

Figure 1: Combined VRPTW solution routes

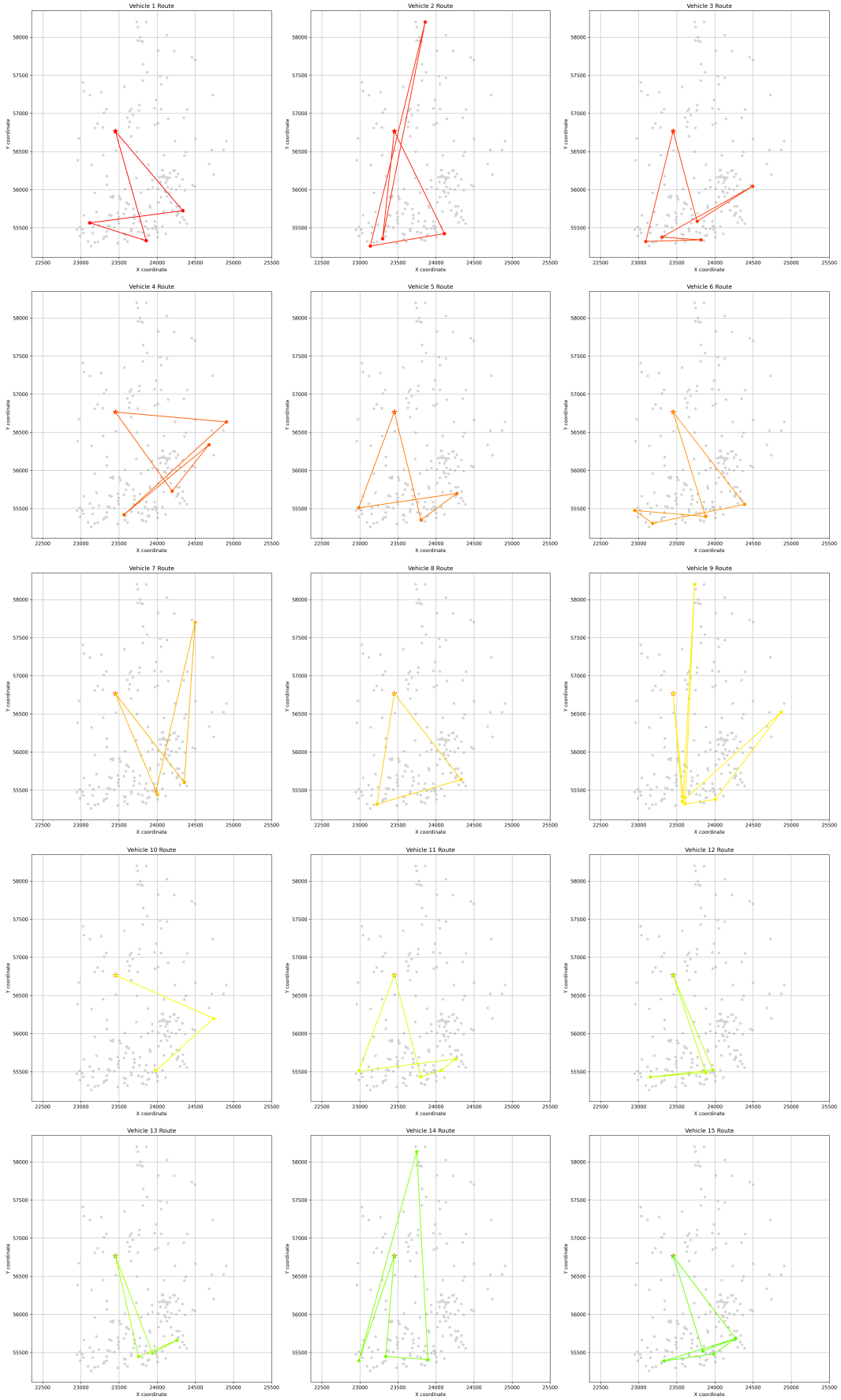

Figure 2: Individual routes for each vehicle (1–15)

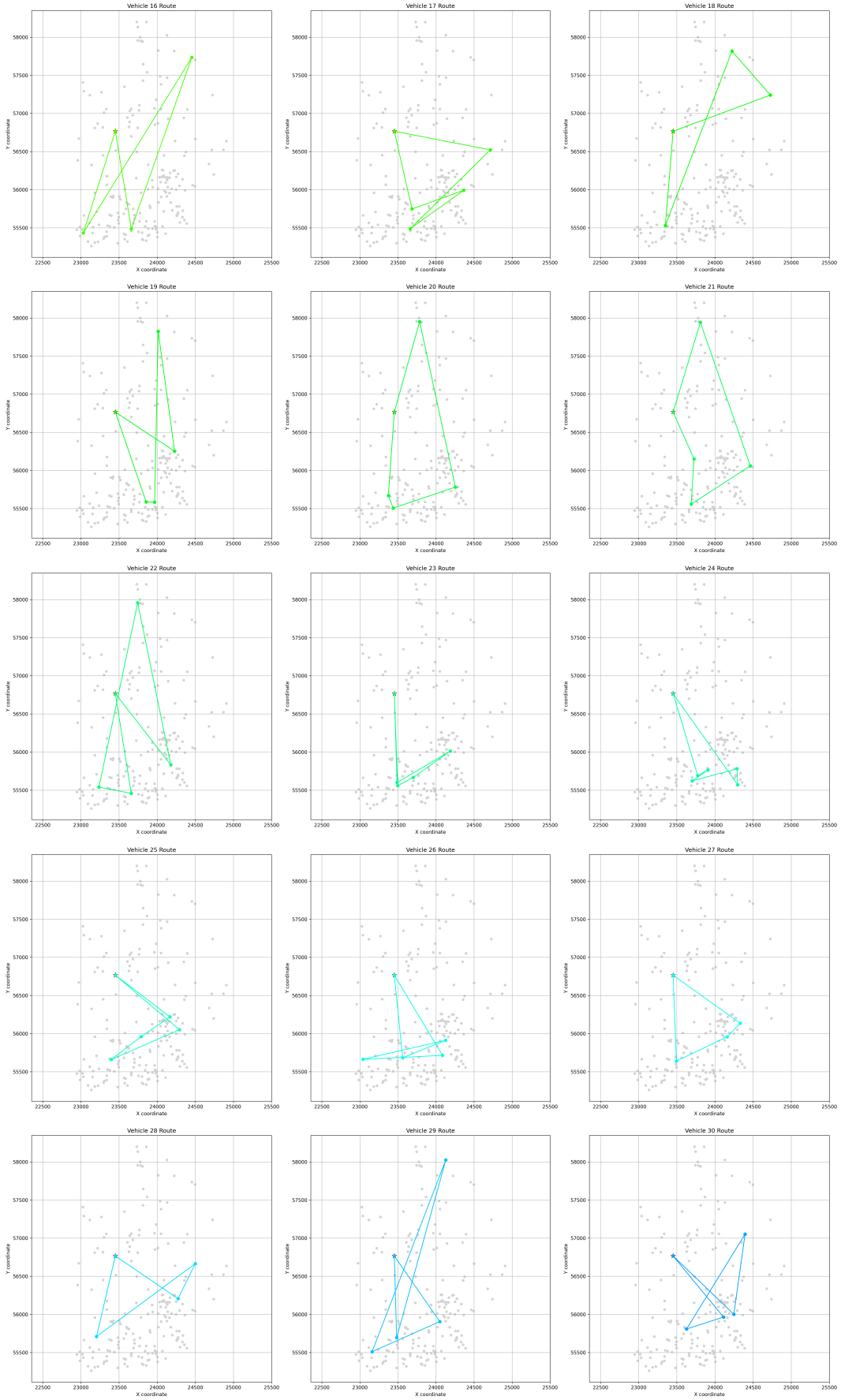

Figure 3: Individual routes for each vehicle (16–30)

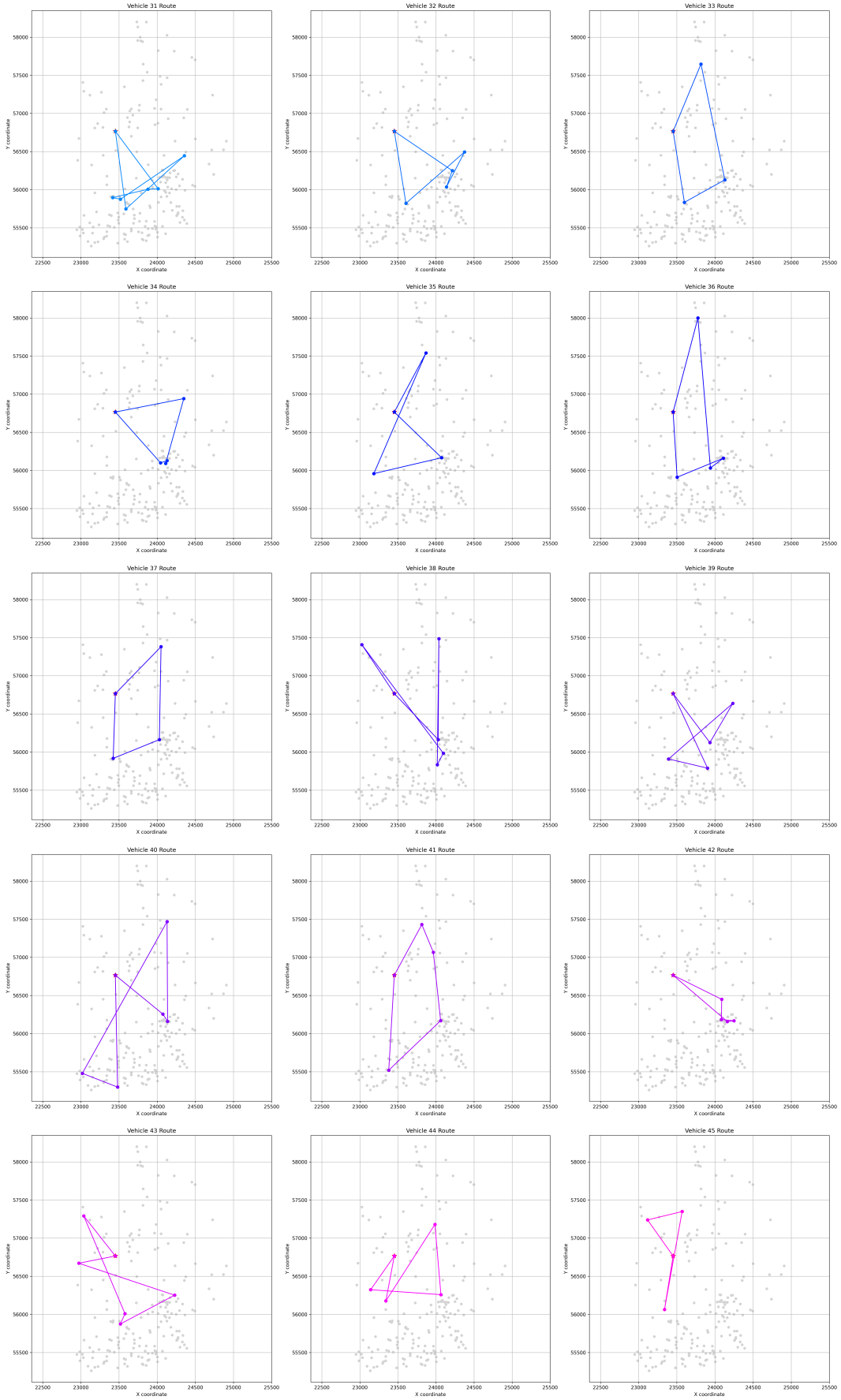

Figure 4: Individual routes for each vehicle (31–45)

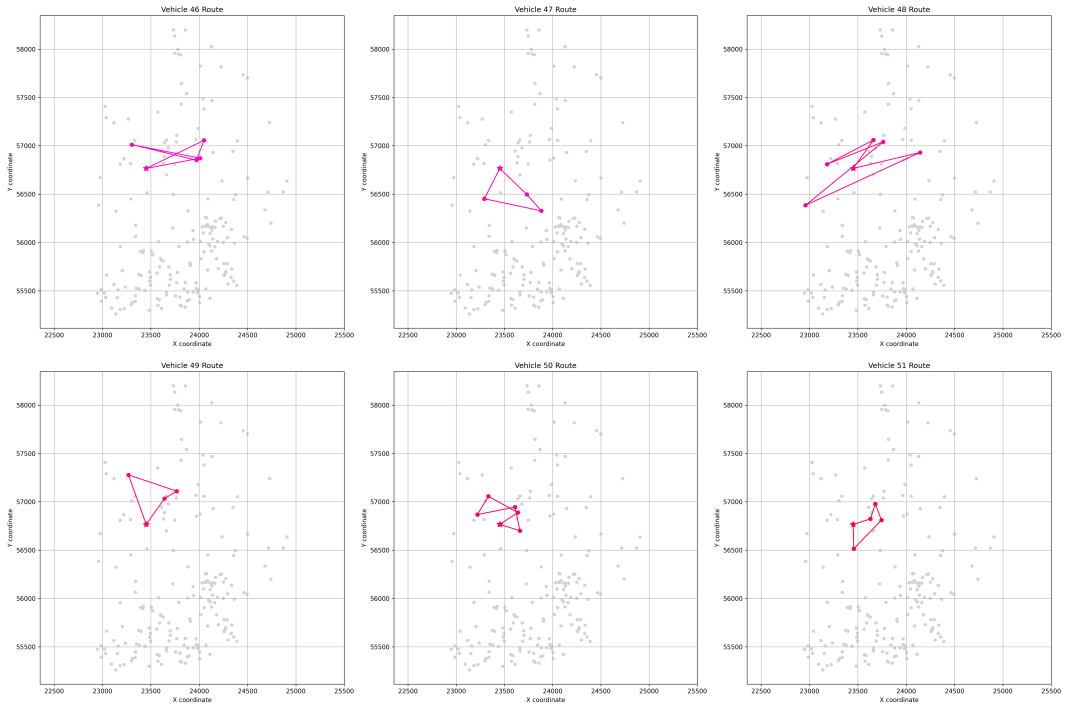

Figure 5: Individual routes for each vehicle (46–51)