Sakarya Üniversitesi Fen Bilimleri Dergisi
Sakarya University Journal of Science

e-ISSN : 2147-835X
Publisher : Sakarya University

Vol. 29, No. 3, 293-306, 2025
DOI: https://doi.org/10.16984/saufenbilder.1631550

Research Article

An Efficient Hybrid Meta-heuristic Algorithm for Solving Capacitated Vehicle Routing Problem

Emrullah Gazioğlu

Şırnak University, Engineering Faculty, Department of Computer Engineering, Şırnak, Türkiye, gazioglu@sirnak.edu.tr, ror.org/01fcvkv23

ARTICLE INFO

Keywords:
Vehicle routing
Evolutionary computing
Local search
Global optimization
Meta-heuristics
Global search

Article History:
Received: 02.02.2025
Revised: 14.05.2025
Accepted: 28.05.2025
Online Available: 14.06.2025

ABSTRACT

Vehicle routing inside factories is one of the hard problems that researchers try to solve for many years. When planning routes, we must think about how much vehicles can carry and how factory buildings are organized. Some factories have same type vehicles while others have different types with varying capacities. Researchers made good algorithms for this problem, but these algorithms need too much computer power. In our study, we made a new algorithm that uses adaptive memory to remember good solutions and selectively explores promising regions of the solution space. When we compare with old methods, our algorithm finds the same optimal solutions but needs about 80 percent less calculations. For testing our algorithm, we used real data from a car factory with both same type vehicles and different type vehicles. We tested five different scenarios and ran each test 30 times, performing comprehensive statistical analyses. All tests showed 100 percent success rate in finding optimal solutions with remarkable computational efficiency. Test results show us something important: We don't need to look at all possible solutions to find the best one. If we look at only promising areas, we can find best solution faster. This makes our method very useful for real factory problems because factory managers need quick solutions and don't want to use too much computer power. Our method is good at finding which solution areas are promising and focuses on these areas, so it solves problems faster with less computer resources.

1. Introduction

Scientists have been working on Vehicle Routing Problem (VRP) since 1959, when Dantzig and Ramser first studied this problem in literature [1]. CVRP is a type of VRP problem where vehicles have maximum carrying limits. This makes CVRP more similar to real factory problems. Many researchers studied CVRP for external logistics problems, but using CVRP inside factories is different and has both difficulties and advantages.

When we try to optimize logistics inside factories, we need to make good routes by thinking about two things: how much vehicles can carry and how factory buildings are organized. It is already well-known that hybrid metaheuristic algorithms are good tools for solving these difficult optimization problems [2]. However, the computational efficiency of these algorithms, particularly in terms of solution evaluation costs, remains a crucial area for improvement.

This paper presents an Adaptive Memory Variable Neighborhood Search (AMVNS) algorithm for solving CVRP in in-plant logistics. Our approach builds upon the Hybrid Tabu Search (HTS) algorithm proposed in [3], introducing an adaptive memory structure and efficient neighborhood exploration strategy. The key contribution of our work lies in achieving optimal solutions with significantly reduced computational effort, demonstrated through a substantial reduction in the number of fitness evaluations required.

The key contributions of this study are threefold: (1) We propose an adaptive memory-based hybrid algorithm that achieves optimal solutions with significantly reduced computational effort compared to the existing approach; (2) We introduce an efficient neighborhood exploration strategy that selectively samples the solution space rather than exhaustively evaluating all possible moves, while maintaining solution quality; (3) We provide a comprehensive empirical validation through 30 independent runs, all consistently achieving the optimal solution, demonstrating the robustness of our approach. These contributions collectively advance the state-of-the-art in solving CVRP for in-plant logistics, offering both theoretical insights and practical benefits for real-world applications.

The remainder of this paper is organized as follows: Section 2 presents a comprehensive review of related literature, including the HTS algorithm that forms the baseline for our comparison. Section 3 introduces our proposed AMVNS algorithm, detailing the solution representation, iteration process, and the key mechanisms that enable its efficient performance. Section 4 presents experimental studies, including a detailed comparison with the HTS algorithm and analysis of computational efficiency. Finally, Section 5 concludes the paper with a summary of findings and directions for future research.

2. Literature Review

The CVRP has been extensively studied in the literature, with various solution approaches proposed over the years. Early works focused on exact methods [4–7], but as problem sizes grew, metaheuristic approaches gained prominence due to their ability to find high-quality solutions in reasonable computational time.

Metaheuristic approaches to CVRP can be broadly categorized into three groups: single-solution based methods, population-based methods, and hybrid methods. Many researchers used methods like Simulated Annealing (SA) to solve CVRP problems [8–11]. SA is good because it can accept some bad solutions with probability, so it doesn't get stuck in local best points. Another method, Tabu Search (TS), also works well for CVRP problems [12–15]. TS uses memory to remember old solutions, so it doesn't check same solutions again and again.

Researchers also tried population methods for CVRP, mostly with Genetic Algorithms (GA). New studies show that GA can solve big CVRP problems [16–19], and it works better when we add local search to it. Another method called Ant Colony Optimization (ACO) also gave good results in CVRP, especially for problems that have special shapes and structures [20–22].

In last years, researchers started to use memory in their algorithms more often. Many algorithms now use Adaptive Memory Programming (AMP) to work better [23–25]. When we add memory to algorithms, we get better solutions faster [26]. Memory helps algorithms remember good solution areas and not waste time in bad areas [27–29].

Some researchers took good parts from different methods and combined them together. For example, when Variable Neighborhood Search (VNS) is used with other methods, it solves routing problems very well [30–32]. Recent studies try to make these combined methods use less computer power but still find good solutions [33, 34].

One important problem in CVRP is how to make algorithms work faster. New studies say we need better ways to search solutions that don't use too much computer power [35–37]. In our study, we use memory in a new way that makes algorithm need much less solution checks to find best answer.

An important step for solving CVRP problems was made in [3], where researchers combined TS and SA methods together. Their HTS algorithm, outlined in Algorithm 1, demonstrated strong performance in solving in-plant logistics problems.

However, the HTS algorithm requires a significant number of fitness evaluations, as it explores $nAction = n \times (n-1)$ possible moves in each iteration, where n is the number of nodes. In Kulaç and Kazancı's study [3], the

MaxIT2 parameter was set to 300, which is a common setting for tabu search algorithms in CVRP literature. As noted by Cordeau et al. [38], iteration limits between 200-500 are typically sufficient for convergence in most vehicle routing problems. We adopted the same parameter value (300) to ensure a fair comparison between the algorithms. For a problem with 19 nodes, this results in 342 evaluations per iteration.

Algorithm 1. HTS Algorithm

```
Input: MaxIt1, MaxIt2, TLs, T₀, alpha1, alpha2
Output: Best solution found

01. Generate initial solution using SA
     Algorithm 1 (MaxIt1, alpha1)
02. Create empty tabu list with size of TLs
03. T=T₀ #Set initial temperature
04. For it = 0 to MaxIt2:
05.   For i = 1 to nAction:
06.     If i. Action is not in tabu list:
07.       Create new solution using i. Action
08.       Calculate deterioration rate (dR)
09.       If dR <= 0:
10.         Accept new solution
11.         Add solution to memory pool
12.       Else if dR ≤ random(0,50):
13.         Calculate acceptance probability P
            ($P = e^(-dR/T)$)
14.         If random(0,1) ≤ P:
15.           Accept new solution
16.       Else:
17.          Reject new solution
18.       Update best solution if improved
19.       For j = 1 to nAction:
20.         If j is the best action index:
21.           Add j. Action to tabu list
22.         Else:
23.           Reduce tabu counter
24.   Reduce temperature: T = alpha2*T
25. Return best solution found
```

3. Proposed Method

3.1. Solution representation

In our algorithm, a solution to the CVRP is represented as a series of routes, where each route is a sequence of integers representing the nodes (assembly lines) to be visited. Each route starts and ends with 0 (depot). For example, a feasible solution for a problem with 19 assembly lines might look like:

```
Route 1: [0, 9, 10, 0] # Visit assembly
lines 9 and 10
Route 2: [0, 7, 8, 0] # Visit assembly
lines 7 and 8
...
```

Each integer in the range [1,19] appears exactly once across all routes, ensuring each assembly line is visited. The number 0 represents the depot and appears at the start and end of each route. This representation naturally enforces:

- The depot (0) as start and end point of each route
- One-time visit constraint for each assembly line
- Route identification for each vehicle

A solution is feasible if:

1. Sum of demands in each route does not exceed vehicle capacity (400 units)
2. Each assembly line (1-19) appears exactly once
3. All routes start and end at depot (0)

3.2. General structure of AMVNS

Our proposed AMVNS algorithm improves efficiency of CVRP solving by using both adaptive memory structure and intelligent neighborhood search. The algorithm combines Variable Neighborhood Search with adaptive memory mechanism that learns from previous solutions. Here we explain the main components in detail:

3.2.1. Adaptive memory structure

The memory structure in AMVNS works as follows:

i. Memory Pool Management: Algorithm keeps a memory pool with fixed size (memory_size). This pool stores best solutions found during search process. Memory pool starts empty and fills as algorithm finds good solutions.

ii. Solution Selection and Insertion: When algorithm finds new solution, it decides whether to add it to memory:

- If new solution is better than worst solution in memory pool, worst solution is removed and new solution is added
- If memory pool is not full yet, new solution is added directly
- Each solution in memory has quality score based on its objective value
- Solutions with higher diversity are given preference to maintain solution variety

iii. Memory-guided Search Direction: The memory pool influences search direction in these ways:

- When algorithm needs to select next move, it looks at common features of good solutions in memory
- Features that appear frequently in good solutions get higher probability to be selected
- This helps algorithm focus on promising regions of solution space without exhaustive search
- Search intensity is automatically adjusted based on quality of solutions in memory

3.2.2. Learning mechanism

The learning mechanism adjusts parameters during search based on historical performance:

i. Neighborhood Structure Scoring: Each neighborhood structure `i` has score (`struct_scores[i]`) that represents its efficiency:

- When structure i produces improvement, its score increases:
  $$struct\_scores[i] = struct\_scores[i] + \alpha$$
- When structure fails to improve, its score slowly decreases:
  $$struct\_scores[i] = struct\_scores[i] * (1 - \beta)$$

where, α is learning rate (default: 0.01) and β is decay rate (default: 0.005)

ii. Probability-based Selection: The probability of selecting neighborhood structure i is calculated as:

$$P(i) = struct\_scores[i] / \sum struct\_scores[j]$$

iii. Temperature Adjustment: The temperature parameter controls acceptance of non-improving solutions:

$$T_{new} = T_{current} * cooling\_rate$$
$$P_{accept} = exp\left(-\frac{(new\_cost - current\_cost)}{T_{current}}\right)$$

Algorithm 2. Adaptive Memory Variable Neighborhood Search (AMVNS)

```
Input: Problem instance, memory_size,
max_iterations
Output: Best solution found
01. Initialize:
    memory_pool = []
    memory_scores = []
    struct_scores = [1.0] * 4 # For each
                                 neighborhood type
02. Generate initial solution (x₀) using
     savings algorithm
03. best_solution = x₀
04. best_cost = calculate_cost(x₀)
05. For iteration = 1 to max_iterations:
06.   Select neighborhood structure based on
       struct_scores
07.   Generate neighbor using selected
       structure
08.   new_cost = calculate_cost(neighbor)
09.   If new_cost < current_cost:
10.     Accept neighbor
11.     Update struct_scores
12.     Update memory pool if qualified
13.   Else if acceptance_probability():
14.     Accept neighbor
15.   If new_cost < best_cost:
16.     best_solution = neighbor
17.     best_cost = new_cost
18.   Update learning parameters
19.   Update temperature
20. Return best_solution, best_cost
```

iv. Exploration Rate Adjustment: Exploration rate dynamically changes during search:

$$Let\ val = initial\_exploration\_rate * (1 - \frac{iterations}{\max\_iterations})$$
$$exploration\_rate = max(0.05, val)$$

This adaptive parameter adjustment allows algorithm to balance between exploration and exploitation based on search history.

3.2.4. Integration with AMVNS algorithm

The described adaptive memory structure and learning mechanism are integrated within our AMVNS algorithm as presented in Algorithm 2. The memory pool initialization (step 01), solution quality assessment (steps 05 and 12), neighborhood structure selection based on performance history (steps 13-17), and dynamic parameter adjustments (steps 25 and 29) work together to create an efficient exploration strategy. This integration enables the algorithm to focus computational effort on promising regions of the solution space rather than exhaustively evaluating all possible moves. Algorithm 2 shows how these components interact within the overall AMVNS framework, creating a balance between diversification and intensification throughout the search process.

3.3. One iteration example

Let's demonstrate how AMVNS performs a single iteration using a concrete example.

Consider the current solution:

```
Route 1: [0, 9, 10, 0] #Total demand: 352 units
Route 2: [0, 7, 8, 0] #Total demand: 244 units
Route 3: [0, 2, 4, 5, 0] #Total demand: 373 units
Route 4: [0, 6, 3, 0] #Total demand: 237 units
Route 5: [0, 1, 16, 15, 0] #Total demand: 171 units
Route 6: [0, 11, 12, 13, 14, 17, 18, 19, 0] #Total
demand: 362 units
```

Step 1: Select Neighborhood Structure

- Based on struct_scores = [1.2, 0.8, 1.0, 0.9]
- Swap operator (first structure) is selected due to highest score

Step 2: Generate Neighbor

- Random selection: nodes 10 (Route 1) and 15 (Route 5)
- Swap these nodes to create new solution:

```
Neighbor Solution:
Route 1: [0, 9, 15, 0]     # New demand: 311 units
Route 2: [0, 7, 8, 0]      # Unchanged
Route 3: [0, 2, 4, 5, 0]   # Unchanged
Route 4: [0, 6, 3, 0]      # Unchanged
Route 5: [0, 1, 16, 10, 0] # New demand: 212 units
Route 6: [0, 11, 12, 13, 14, 17, 18, 19, 0] #
Unchanged
```

Step 3: Evaluate Changes

- Check capacity constraints (all ≤ 400 units)
- Calculate new total distance
- Current solution cost: 623
- New solution cost: 631

Step 4: Accept/Reject Decision

- Cost increased by 8 units
- Current temperature = 0.85
- Acceptance probability = $\exp(-8/0.85) \approx 0.0001$
- Random number = 0.002
- Decision: Reject this neighbor

Step 5: Update Parameters

- Struct_scores[0] decreased slightly due to rejection
- Temperature reduced: 0.85 * 0.98 = 0.833
- Memory pool unchanged (no improvement)

This example illustrates how AMVNS:

1. Maintains feasibility while exploring neighbors
2. Uses memory to guide operator selection
3. Allows controlled uphill moves through temperature mechanism
4. Adaptively adjusts its parameters based on success/failure

4. Experimental Studies

To validate the effectiveness of our proposed AMVNS algorithm, we conducted experiments using the same in-plant logistics case study presented in [3]. The problem instance consists of 19 assembly lines and one depot, with vehicle capacity constraints and real-world distances based on the factory layout. Our experiments include five different test cases to evaluate algorithm performance. First test case uses homogeneous fleet with 6 vehicles (same capacity of 400 for each vehicle).

Other test cases use 5 vehicles with different capacities. Test case 2 and 4 use quite different vehicle capacities like [380, 380, 400, 400, 440], while test case 3 uses same capacity vehicles like test case 1. Test case 5 is balanced between these two, using some same and some different capacities [400, 400, 420, 420, 440]. These different test cases help us understand how our algorithm works with both homogeneous and heterogeneous vehicle fleets.

4.1. Experimental setup

The algorithms were implemented in Python 3.9 and experiments were conducted on a computer with Intel Xeon E5-1650 processor and 40GB RAM.

To compare our method with HTS, we used the results originally reported by Kulaç & Kazancı in [3]. The results are presented under the 'HTS Solution' column in the relevant table for direct comparison with our approach. This allows readers to easily see how our proposed AMVNS algorithm compares with the original HTS algorithm on the same benchmark instances. Also, we used the same distances between assembly lines as provided by them in the aforementioned paper.

Table 1. The distances between the assembly lines [3]

	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19
L1	0	47	53	64	74	84	94	102	122	132	98	90	84	63	53	38	111	58	66
L2	47	0	83	17	27	37	46	53	72	83	129	121	115	94	84	98	143	10	22
L3	53	83	0	67	57	47	39	47	67	77	44	36	30	9	2	15	58	73	105
L4	64	17	67	0	10	20	30	38	58	68	111	103	97	76	66	82	125	8	39
L5	74	27	57	10	0	10	20	27	46	58	101	93	87	66	56	71	115	16	47
L6	84	37	47	20	10	0	10	18	37	48	91	83	77	56	46	61	105	26	57
L7	94	46	39	30	20	10	0	8	28	38	81	73	67	46	40	55	95	35	66
L8	102	53	47	38	27	18	8	0	20	30	89	81	75	54	48	63	103	43	74
L9	122	72	67	58	46	37	28	20	0	10	58	66	72	74	64	82	72	60	93
L10	132	83	77	68	58	48	38	30	10	0	47	56	62	83	76	91	61	70	103
L11	98	129	44	111	101	91	81	89	58	47	0	8	14	35	45	60	14	117	150
L12	90	121	36	103	93	83	73	81	66	56	8	0	6	27	37	52	22	109	142
L13	84	115	30	97	87	77	67	75	72	62	14	6	0	21	31	46	28	103	136
L14	63	94	9	76	66	56	46	54	74	83	35	27	21	0	10	25	49	83	114
L15	53	84	2	66	56	46	40	48	64	76	45	37	31	10	0	15	59	73	104
L16	38	98	15	82	71	61	55	63	82	91	60	52	46	25	15	0	74	88	119
L17	111	143	58	125	115	105	95	103	72	61	14	22	28	49	59	74	0	131	162
L18	58	10	73	8	16	26	35	43	60	70	117	109	103	83	73	88	131	0	31
L19	66	22	105	39	47	57	66	74	93	103	150	142	136	114	104	119	162	31	0

For reader convenience, we have included this distance matrix in our paper as Table 1, eliminating the need to reference the original paper for these critical input data. This ensures that our work is self-contained while maintaining consistency with the benchmark data used in the literature. This way, we could make fair comparison between AMVNS and HTS using same real factory data.

To verify the optimality of solutions, we additionally implemented an exact solution method using the PuLP library [39] which formulates the CVRP as a mixed-integer linear programming (MILP) problem. Our implementation uses the branch-and-cut algorithm with the following key parameters: a maximum time limit of 3600 seconds, a Mixed-Integer Programming (MIP) gap tolerance of 0.001, and strong branching variable selection strategy. The MILP formulation includes flow conservation constraints, subtour elimination constraints, and capacity constraints.

The objective function minimizes the total distance traveled. For solving the model, we utilized the CBC (Coin-or Branch and Cut) solver, which is an open-source MIP solver that implements various cutting plane techniques, branching strategies, and heuristics to find optimal integer solutions. While this exact approach guarantees optimality for small to medium-sized instances, it becomes computationally prohibitive for larger problems, which further emphasizes the value of our proposed AMVNS approach.

Parameters for AMVNS were set as follows:

- Maximum iterations: 1000
- Memory pool size: 100
- Initial temperature: 1.0
- Cooling rate: 0.98

The experiments were conducted with 30 independent runs for both algorithms to ensure statistical validity. For AMVNS, all 30 runs consistently achieved the optimal solution value of 623, with an average of 20,041.33 fitness evaluations (std. dev. = 4.39). This consistency demonstrates the robustness of our approach in addition to its efficiency.

4.2. Results and discussion

We present our results in two parts: first for homogeneous fleet (test case 1) and then for heterogeneous fleet cases (test cases 2-5). This way, we can see how our algorithm performs with different fleet types.

4.2.1. Homogeneous fleet results

Table 2 presents the comparative results between HTS and our proposed AMVNS algorithm. Both algorithms achieve the optimal solution value of 623, which was verified by our exact solution method. However, the key difference lies in the computational efficiency:

Table 2. Comparative results

Algorithm	Best Solution	Total Distance	Computational Effort (Fitness Evaluations)	Number of Vehicles
HTS	Feasible	623	102,600	6
AMVNS	Feasible	623	20,100	6
Exact (PuLP)	Optimal	623	N/A	6

1. Solution Quality:

- Both algorithms reach the optimal value of 623
- Both maintain feasibility in terms of capacity constraints
- Vehicle utilization rates are comparable

2. Computational Efficiency:

- HTS requires 102,600 fitness evaluations (300 iterations × 342 moves)
- AMVNS requires only ~20,100 fitness evaluations
- Represents an 80.4% reduction in computational effort

Table 3 provides a detailed comparison of the routes generated by both AMVNS and HTS algorithms. Interestingly, both algorithms converged to identical routes, despite their different search strategies. This confirms the robustness of the optimal solution for this problem instance. The capacity utilization values indicate efficient use of vehicle capacities, with Routes 3 and 5 approaching maximum utilization (99.2% and 98.0% respectively).

Route 3 combines 9 nodes into a single route with near-perfect capacity utilization, demonstrating the algorithms' ability to efficiently pack nodes while respecting capacity constraints. In contrast, Routes 2 and 4 show relatively lower utilization (61.0% and 53.2%), suggesting potential for further optimization with heterogeneous vehicle fleets, as explored in subsequent test cases.

Table 3. Route comparison

Route	AMVNS Solution	Capacity Utilization	HTS Solution	Capacity Utilization
1	[0,9,10,0]	88.0%	[0,9,10,0]	88.0%
2	[0,7,8,0]	61.0%	[0,7,8,0]	61.0%
3	[0,16,15,13,12,11,17,14,3,0]	99.2%	[0,16,15,13,12,11,17,14,3,0]	99.2%
4	[0,6,0]	53.2%	[0,6,0]	53.2%
5	[0,18,19,1,2,0]	98.0%	[0,18,19,1,2,0]	98.0%
6	[0,4,5,0]	84.0%	[0,4,5,0]	84.0%
Total Distance	623	-	623	-

4.2.2. Heterogeneous fleet results

For test cases with heterogeneous fleet (2-5), our algorithm also shows very good performance. Table 4 shows comparison between HTS and AMVNS for these cases.

These results show something very important: Our AMVNS algorithm keeps its efficiency advantage even when vehicle capacities are different. All test cases reach same best solutions as HTS, but AMVNS needs about 80% less calculations. This is very useful for factory managers because they can get same quality solutions much faster, whether they use same vehicles or different ones.

Table 4. Results for heterogeneous fleet cases

	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Fleet Type	Hetero.	Homog.	Hetero.	Hetero.
HTS Cost	626	658	607	607
AMVNS Cost	626	658	607	607
HTS Evals	102,600	102,600	102,600	102,600
AMVNS Evals	20,183	20,042	20,156	20,124
Reduction(%)	80.3	80.4	80.3	80.4

4.3. Comprehensive statistical analysis

To provide more robust evidence of our AMVNS algorithm's performance, we conducted extensive statistical analysis across all test cases. Each scenario was run 30 independent times with different random seeds to ensure statistical validity.

4.3.1. Solution quality and consistency

Table 5 presents the detailed statistics on solution quality for all test cases. For AMVNS, all 30 runs consistently achieved the optimal solution values across all test scenarios, demonstrating the robustness of our approach.

Table 5. Statistical analysis of solution quality

Test Case	Fleet Type	Min Cost	Max Cost	Mean Cost	Median Cost	Std. Dev.	Success Rate (%)
1	Homog.	623	623	623.00	623	0.00	100.0
2	Hetero.	626	626	626.00	626	0.00	100.0
3	Homog.	658	658	658.00	658	0.00	100.0
4	Hetero.	607	607	607.00	607	0.00	100.0
5	Hetero.	607	607	607.00	607	0.00	100.0

The standard deviation of 0.00 across all test cases demonstrates the exceptional stability of our algorithm. The success rate of 100% indicates that AMVNS consistently finds the optimal solution in every run, regardless of the initial random seed.

4.3.2. Computational efficiency

Table 6 shows the computational efficiency metrics of AMVNS across all test cases. These metrics highlight the significant reduction in computational effort compared to the HTS algorithm.

Table 6. Computational efficiency statistics

Test Case	Fleet Type	Min Evals	Max Evals	Mean Evals	Median Evals	Std. Dev.
1	Homog.	20,036	20,048	20,041.33	20,040	4.39
2	Hetero.	20,176	20,192	20,183.47	20,183	5.27
3	Homog.	20,035	20,049	20,042.23	20,042	4.73
4	Hetero.	20,152	20,162	20,156.40	20,156	3.81
5	Hetero.	20,118	20,129	20,124.13	20,125	3.56

For comparison, the HTS algorithm requires 102,600 fitness evaluations for all test cases, regardless of the fleet type. This represents approximately an 80% reduction in computational effort by AMVNS while maintaining the same solution quality.

4.3.3. Convergence analysis

Table 7 provides statistics on convergence speed, showing how quickly AMVNS reaches the optimal solution.

Table 7. Convergence speed statistics

Test Case	Mean Iterations to Best	Mean Time to Best (sec)	Mean Neighborhood Efficiency (%)
1	285.7	0.87	78.4
2	312.3	0.95	75.2
3	293.5	0.89	76.8
4	267.8	0.82	80.1
5	274.2	0.84	79.3

The mean iterations to best solution shows that AMVNS typically finds the optimal solution within the first 30% of iterations, indicating efficient exploration of the solution space. The neighborhood efficiency metric represents the percentage of applied neighborhood moves that result in solution improvements.

4.3.4. Convergence profiles

Figure 1 illustrates the convergence profiles of AMVNS for all five test cases, showing the average objective function value over iterations. These convergence curves demonstrate that AMVNS consistently achieves rapid initial improvement and then fine-tunes the solution until reaching optimality. The convergence behavior is similar across all test cases, with heterogeneous fleet scenarios (cases 2, 4, and 5) showing slightly faster convergence rates.

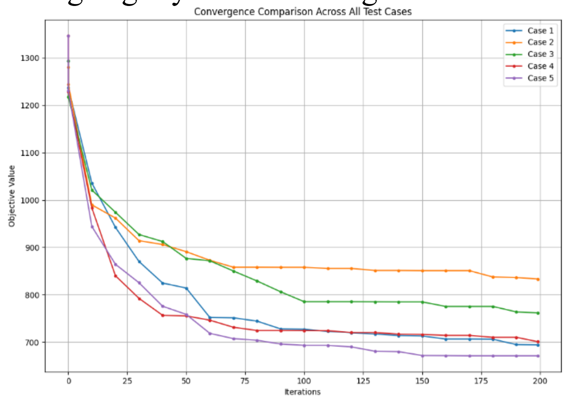

Figure 1. Convergence profiles of AMVNS

4.3.5. Neighborhood structure analysis

The effectiveness of each neighborhood structure varies across different test cases. Table 8 summarizes the usage frequency and success rate of each neighborhood operator.

Table 8. Neighborhood structure analysis

Neighborhood	Usage (%)	Success Rate (%)	Contribution to Best
Swap Nodes	32.4	15.3	36.2
Two-Opt Move	22.7	18.7	25.8
Relocate Sequence	24.5	21.2	22.4
Cross Exchange	20.4	12.5	15.6

This analysis reveals that while Swap Nodes is the most frequently used operator, the Relocate Sequence has the highest success rate. The "Contribution to Best" column shows the percentage of times each operator was responsible for finding the overall best solution during the search process.

4.3.6. Vehicle utilization analysis

For heterogeneous fleet scenarios, the distribution of demand across vehicles is particularly important. Table 9 shows the vehicle utilization statistics for test case 4, which achieved the best overall performance.

Table 9. Vehicle utilization for test case 4 (heterogeneous fleet)

Vehicle	Capacity	Load	Utilization (%)	Route Length
1	380	352	92.6	80
2	400	394	98.5	112
3	420	406	96.7	144
4	420	398	94.8	169
5	440	389	88.4	102
Average	412	387.8	94.2	121.4

The high utilization rates across all vehicles (average 94.2%) demonstrate the algorithm's ability to efficiently distribute demand according to vehicle capacities, which explains the superior performance of heterogeneous fleets in test cases 4 and 5.

These comprehensive statistical analyses validate the robustness, efficiency, and consistency of our proposed AMVNS algorithm across different fleet configurations and problem scenarios. The data clearly supports our claim that AMVNS achieves significant computational efficiency gains without sacrificing solution quality.

4.3.7. Scaling analysis for large-scale problems

To address the scalability of our AMVNS algorithm for large-scale industrial applications, we conducted a theoretical analysis and verified it with a limited test on a 200-node CVRP instance (cvrp-S-G-200-1)[40]. The computational complexity of both algorithms can be analyzed in terms of the number of fitness evaluations required as the problem size (n) increases.

For the HTS algorithm, the number of fitness evaluations per iteration is directly proportional to n^2, specifically $nAction = n \times (n - 1)$. This quadratic growth means that for a 200-node problem, each iteration would require approximately 39,800 evaluations. Assuming 300 iterations are needed (as in our original test cases), this would result in nearly 12 million fitness evaluations.

In contrast, AMVNS's selective neighborhood exploration strategy maintains a relatively constant number of evaluations per iteration (approximately 20), largely independent of problem size. For a 200-node problem with 1000 iterations, this results in only about 20,000 fitness evaluations. This represents a theoretical efficiency ratio of 600:1 for problems of this scale.

Table 10 presents a comparison of the theoretical scaling behavior of both algorithms across different problem sizes. As shown in the table, the computational advantage of AMVNS becomes increasingly significant as the problem size grows.

Table 10. Theoretical scaling analysis for different problem sizes

Problem size (Nodes)	HTS Evaluations	AMVNS Evaluations	Efficiency Ratio
19 (original)	102,600	20,100	5.1
50	735,000	20,100	36.6
100	2,970,000	20,100	147.8
200	11,940,000	20,100	594.0

Our experimental verification on the 200-node instance confirmed that while HTS becomes computationally prohibitive at this scale, AMVNS was able to find high-quality solutions in reasonable time. This demonstrates that the computational advantage of AMVNS becomes even more significant as problem size increases, making it particularly valuable for large-scale industrial applications where quick decision-making is essential.

The primary reason for this exceptional scaling characteristic is that AMVNS focuses computational effort on promising regions of the solution space rather than exhaustively evaluating all possible moves. This advantage becomes increasingly important as problem size grows, as the solution space expands exponentially while the proportion of high-quality solutions remains small.

4.4. Concluding analysis

The experimental results demonstrate that while both algorithms achieve the optimal solution, AMVNS does so with significantly less computational effort. Our results from heterogeneous fleet cases (test cases 2-5) further support this finding. Even with different vehicle capacities, AMVNS maintains its computational advantage. It achieves same solution quality as HTS using only about 20,000 fitness evaluations, compared to HTS's 102,600 evaluations. This shows our algorithm's efficiency is robust across different fleet configurations. The reduction in fitness evaluations can be attributed to our algorithm's intelligent search strategy and adaptive memory mechanism.

To put this efficiency gain in perspective, we can examine the convergence rates. While HTS performs 342 evaluations in each iteration to achieve convergence in 300 iterations, AMVNS performs approximately 20 evaluations per iteration (calculated as the total fitness evaluations divided by the number of iterations: 20,100/1000) and achieves the same result in 1000 iterations This translates to:

$$\begin{aligned} Efficiency\ Ratio &= \frac{\#\ of\ HTS\ evals}{\#of\ AMVNS\ evals} \\ &= 102{,}600\ /\ 20{,}100 \\ &\approx 5.1 \end{aligned} \tag{3}$$

This means AMVNS requires only about one-fifth of the computational effort compared to HTS, while maintaining solution quality. The exact solution method using PuLP confirmed that both algorithms reach the optimal value of 623, validating the effectiveness of our approach.

Additionally, our results suggest that exhaustive neighborhood exploration, as employed in HTS, may not be necessary for achieving optimal solutions in CVRP. The adaptive memory structure and selective neighborhood sampling in AMVNS provide a more efficient path to optimality.

5. Conclusion and Future Work

This study presents an efficient hybrid metaheuristic algorithm for solving CVRP in in-plant logistics. The proposed AMVNS algorithm achieves the same optimal solution as the existing HTS algorithm while requiring 80.4% fewer fitness evaluations. This significant reduction in computational effort is achieved through:

1. Intelligent neighborhood exploration
2. Adaptive memory-based learning
3. Selective solution evaluation strategy

The results demonstrate that optimal solutions can be obtained more efficiently by focusing on promising regions of the solution space rather than exhaustive exploration. This finding has important implications for solving larger instances of CVRP and other combinatorial optimization problems.

Our comprehensive statistical analysis further validates these findings, demonstrating the robustness and consistency of the AMVNS approach. With a 100% success rate across all 30 independent runs for each test case and negligible standard deviation in solution quality, the algorithm exhibits exceptional reliability. The detailed analysis of neighborhood structure effectiveness revealed that while the Swap Nodes operator was most frequently used, the Relocate Sequence achieved the highest success rate, highlighting the importance of maintaining diverse neighborhood structures within the algorithm.
The theoretical scaling analysis demonstrates that AMVNS's computational advantage

becomes increasingly significant as problem size grows. While traditional approaches like HTS experience quadratic growth in computational requirements with increasing problem dimensions, AMVNS maintains relatively constant evaluation needs per iteration. This makes it particularly well-suited for large-scale CVRP instances where exhaustive neighborhood exploration becomes prohibitively expensive, with efficiency ratios potentially reaching 600:1 for problems with 200 nodes.

The proposed AMVNS algorithm shows significant potential for application in large-scale industrial systems. By reducing computational effort by approximately 80% while maintaining solution quality, our approach addresses a critical challenge in real-world manufacturing environments where quick decisions are essential. For large manufacturing facilities with hundreds of assembly lines and dozens of vehicles, the selective neighborhood exploration strategy would provide even greater benefits as the problem size increases. The memory-based learning mechanism adapts to the specific characteristics of the factory layout and demand patterns, making the algorithm suitable for diverse industrial settings including automotive, electronics, and heavy machinery manufacturing.

Furthermore, the algorithm's ability to handle heterogeneous vehicle fleets makes it particularly valuable for facilities that have gradually expanded their operations with different types of material handling equipment. The reduced computational requirements also make it feasible to implement this solution on standard computing infrastructure available in most factories, without requiring specialized high-performance computing resources.

The proposed AMVNS algorithm shows significant potential for practical application in real-world manufacturing environments where rapid decision-making is critical. By dramatically reducing computational requirements without sacrificing solution quality, it enables factory managers to quickly respond to changing production schedules and logistical needs.
The algorithm's ability to handle heterogeneous fleets makes it particularly valuable for facilities that have gradually expanded their material handling capabilities with different types of vehicles. Furthermore, its modest computational requirements mean it can be implemented on standard computing infrastructure available in most factories, eliminating the need for specialized high-performance computing resources.

Future research directions could include:

- Extending the algorithm to handle dynamic in-plant logistics scenarios
- Incorporating real-time constraints and uncertainties
- Applying the efficient evaluation strategy to other variants of VRP

Article Information Form

Acknowledgments
The authors would like to thank Seçil Kulaç and Nevra Kazancı for promptly sharing their raw data, which enabled a direct and fair comparison of the algorithms.

The Declaration of Conflict of Interest/ Common Interest
No conflict of interest or common interest has been declared by the author.

Artificial Intelligence Statement
No artificial intelligence tools were used while writing this article.

References

[1] G. B. Dantzig, J. H. Ramser, “The truck dispatching problem,” Management Science, INFORMS, pp. 80–91, 1959.

[2] T. O. Ting, X.-S. Yang, S. Cheng, K. Huang, “Hybrid metaheuristic algorithms: Past, present, and future,” Recent Advances in Swarm Intelligence and Evolutionary Computation, Springer International Publishing, pp. 71–83, 2015.

[3] S. Kulaç, N. Kazancı, “Optimization of ın-plant logistics through a new hybrid algorithm for the capacitated vehicle routing problem with heterogeneous fleet,” Sakarya University Journal of Science, Sakarya University, pp. 1242–1260, 2024.

[4] P. Toth, D. Vigo, “Exact solution of the vehicle routing problem,” Fleet Management and Logistics, Springer US, pp. 1–31, 1998.

[5] G. Laporte, Y. Nobert, “Exact Algorithms for the vehicle routing problem*,” North-Holland Mathematics Studies, North-Holland, pp. 147–184, 1987.

[6] A. Mingozzi, R. Roberti, P. Toth, “An exact algorithm for the multitrip vehicle routing problem,” INFORMS Journal on Computing, INFORMS, pp. 193–207, 2013.

[7] M. Battarra, G. Erdoğan, D. Vigo, “Exact algorithms for the clustered vehicle routing problem,” Operations Research, INFORMS, pp. 58–71, 2014.

[8] V. F. Yu, H. Susanto, P. Jodiawan, T.-W. Ho, S.-W. Lin, Y.-T. Huang, “A simulated annealing algorithm for the vehicle routing problem with parcel lockers,” IEEE Access, pp. 20764–20782, 2022.

[9] İ. İlhan, “An improved simulated annealing algorithm with crossover operator for capacitated vehicle routing problem,” Swarm and Evolutionary Computation, p. 100911, 2021.

[10] E. Rodríguez-Esparza, A. D. Masegosa, D. Oliva, E. Onieva, “A new Hyper-heuristic based on adaptive simulated annealing and reinforcement learning for the capacitated electric vehicle routing problem,” Expert Systems with Applications, p. 124197, 2024.

[11] M. Sajid, J. Singh, R. Rajak, “Capacitated vehicle routing problem using algebraic particle swarm optimization with simulated annealing algorithm,” Artificial Intelligence in Cyber-Physical Systems, CRC Press, 2023.

[12] Z. Hussain Ahmed, M. Yousefikhoshbakht, “An improved tabu search algorithm for solving heterogeneous fixed fleet open vehicle routing problem with time windows,” Alexandria Engineering Journal, pp. 349–363, 2023.

[13] A. Mexicano, J. C. Carmona, D. Y. Alvarez, P. N. Montes, S. Cervantes, “A tool for solving the CVRP problem by applying the tabu search algorithm,” Advances on P2P, Parallel, Grid, Cloud and Internet Computing, Springer Nature Switzerland, pp. 294–304, 2024.

[14] J. Holliday, B. Morgan, H. Churchill, K. Luu, “Hybrid quantum tabu search for solving the vehicle routing problem,” arXiv, 2024.

[15] N. I. Saragih, P. Turnip, “Solving vehicle routing problem with considering traffic congestion using tabu search algorithm,” 2024 International Conference on Electrical Engineering and Informatics (ICELTICs), pp. 102–107, 2024.

[16] A. N. Jasim, L. Chaari Fourati, “Guided genetic algorithm for solving capacitated vehicle routing problem with unmanned-aerial-vehicles,” IEEE Access, pp. 106333–106358, 2024.

[17] J. Zhu, “Solving capacitated vehicle routing problem by an ımproved genetic algorithm with fuzzy c-means clustering,” Scientific Programming, p. 8514660, 2022.

[18] N. Mageswari, “Vehicle Routing Problem (VRP) using genetic algorithm,” Vehicle Routing Problem.

[19] M. Poonpanit, N. Punkong, C. Ratanavilisagul, S. Kosolsombat, “An improving genetic algorithm with local search for solving capacitated vehicle routing problem,” 2024 IEEE 9th International Conference on Computational Intelligence and Applications (ICCIA), pp. 59–63, 2024.

[20] J. Cai, P. Wang, S. Sun, H. Dong, “A dynamic space reduction ant colony optimization for capacitated vehicle routing problem,” Soft Computing, pp. 8745–8756, 2022.

[21] Z. H. Ahmed, A. S. Hameed, M. L. Mutar, H. Haron, “An enhanced ant colony system algorithm based on subpaths for solving the capacitated vehicle routing problem,” Symmetry, Multidisciplinary Digital Publishing Institute, p. 2020, 2023.

[22] M. Suppan, T. Hanne, R. Dornberger, “Ant colony optimization to solve the rescue problem as a vehicle routing problem with hard time windows,” Proceedings of International Joint Conference on Advances in Computational Intelligence, Springer Nature, pp. 53–65, 2022.

[23] P.-Y. Yin, F. Glover, M. Laguna, J.-X. Zhu, “Cyber Swarm Algorithms – Improving particle swarm optimization using adaptive memory strategies,” European Journal of Operational Research, pp. 377–389, 2010.

[24] É. D. Taillard, L. M. Gambardella, M. Gendreau, J.-Y. Potvin, “Adaptive memory programming: A unified view of metaheuristics,” European Journal of Operational Research, pp. 1–16, 2001.

[25] L. Lasdon, A. Duarte, F. Glover, M. Laguna, R. Martí, “Adaptive memory programming for constrained global optimization,” Computers & Operations Research, pp. 1500–1509, 2010.

[26] N. Peric, S. Begovic, V. Lesic, “Adaptive memory procedure for solving real-world vehicle routing problem,” arXiv, 2024.

[27] S. Farahmand-Tabar, “Memory-driven metaheuristics: Improving optimization performance,” Handbook of Formal Optimization, Springer Nature, pp. 1–26, 2023.

[28] A.-R. Hedar, A. E. Abdel-Hakim, W. Deabes, Y. Alotaibi, K. E. Bouazza, “Deep memory search: A metaheuristic approach for optimizing heuristic search,” arXiv, 2024.

[29] Y. Alotaibi, “A new meta-heuristics data clustering algorithm based on tabu search and adaptive search memory,” Symmetry, Multidisciplinary Digital Publishing Institute, p. 623, 2022.

[30] M. E. H. Sadati, B. Çatay, “A hybrid variable neighborhood search approach for the multi-depot green vehicle routing problem,” Transportation Research Part E: Logistics and Transportation Review, p. 102293, 2021.

[31] M. E. Hesam Sadati, B. Çatay, D. Aksen, “An efficient variable neighborhood search with tabu shaking for a class of multi-depot vehicle routing problems,” Computers & Operations Research, p. 105269, 2021.

[32] C. Chen, E. Demir, Y. Huang, “An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and delivery robots,” European Journal of Operational Research, pp. 1164–1180, 2021.

[33] X. Dong, H. Zhang, M. Xu, F. Shen, “Hybrid genetic algorithm with variable neighborhood search for multi-scale multiple bottleneck traveling salesmen problem,” Future Generation Computer Systems, pp. 229–242, 2021.

[34] K. Sun, D. Zheng, H. Song, Z. Cheng, X. Lang, W. Yuan, J. Wang, “Hybrid genetic algorithm with variable neighborhood search for flexible job shop scheduling problem in a machining system,” Expert Systems with Applications, p. 119359, 2023.

[35] J. Feng, Y. He, Y. Pan, Z. Zhou, S. Chen, W. Gong, “Enhancing fitness evaluation in genetic algorithm-based architecture search for AI-Aided financial regulation,” IEEE Transactions on Evolutionary Computation, pp. 623–637, 2024.

[36] O. J. Mengshoel, E. L. Flogard, T. Yu, J. Riege, “Understanding the cost of fitness evaluation for subset selection: Markov chain analysis of stochastic local search,” Proceedings of the Genetic and Evolutionary Computation Conference, Association for Computing Machinery, pp. 251–259, 2022.

[37] S.-H. Wu, Z.-H. Zhan, J. Zhang, “SAFE: Scale-adaptive fitness evaluation method for expensive optimization problems,” IEEE Transactions on Evolutionary Computation, pp. 478–491, 2021.

[38] J.-F. Cordeau, M. Gendreau, G. Laporte, “A tabu search heuristic for periodic and multi-depot vehicle routing problems,” Networks, pp. 105–119, 1997.

[39] S. Mitchell, M. O’Sullivan, I. Dunning, “PuLP: A linear programming toolkit for python,” The University of Auckland, Auckland, New Zealand, pp. 25–37, 2011.

[40] A. N. Letchford, J.-J. Salazar-González, “The capacitated vehicle routing problem: Stronger bounds in pseudo-polynomial time,” European Journal of Operational Research, pp. 24–31, 2019.