Multi-Step Heuristics and Meta-Heuristics for the Capacitated Vehicle Routing Problem (CVRP)

Moaad Samara 314630153, Mahmoud Abadi 206773756

Table of Contents

Introduction

The Capacitated Vehicle Routing Problem (CVRP) is a fundamental combinatorial optimization problem in logistics and transportation. It extends the classical Traveling Salesman Problem (TSP) by introducing multiple vehicles with limited capacity constraints. The objective is to find optimal routes for a fleet of vehicles to serve all customers while minimizing the total travel distance and respecting vehicle capacity constraints.

The CVRP belongs to the class of NP-hard problems, making it computationally challenging for large instances. Therefore, various heuristic and meta-heuristic approaches have been developed to find high-quality solutions within reasonable computational time. This study implements and compares seven different algorithmic approaches, ranging from constructive heuristics to sophisticated meta-heuristics.

Problem Formulation

Mathematical Formulation

The CVRP can be formulated as follows:

Given:

- A set of customers $N = \{1,2,\dots,n\}$ and a depot $\{0\}$
- A distance matrix d_{ij} representing the distance between locations i and j
- Customer demands q_i for $i \in N$
- Vehicle capacity Q
- Fleet size K (upper bound on number of vehicles)

Decision Variables:

- $x_{ijk} \in \{0,1\}$: binary variable equal to 1 if vehicle k travels from customer i to customer j

Objective Function:

$$\min \sum_{k=1}^{K} \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij}\, x_{ijk} \qquad (1)$$

Subject to:

$$\sum_{k=1}^{K} \sum_{j=0}^{n} x_{ijk} \,\&= 1, \quad \forall i \in N \qquad (2)$$

$$\sum_{i=0}^{n} x_{ijk} \,\&= \sum_{i=0}^{n} x_{jik}\,, \quad \forall j \in N \cup \{0\}, k \in K \qquad (3)$$

$$\sum_{i \in N} q_i \sum_{j=0}^{n} x_{ijk} \,\& \leq Q, \quad \forall k \in K \qquad (4)$$

$$x_{ijk} \& \in \{0,1\}, \quad \forall i, j \in N \cup \{0\}, k \in K \qquad (5)$$

Constraint (2) ensures each customer is visited exactly once, constraint (3) maintains flow conservation, constraint (4) enforces vehicle capacity limits, and constraint (5) defines the binary nature of decision variables.

Problem Complexity

The CVRP is NP-hard, meaning that no polynomial-time algorithm is known to solve all instances optimally. The complexity grows exponentially with the number of customers, making exact methods impractical for large instances. This motivates the use of heuristic and meta-heuristic approaches that can find high-quality solutions in reasonable computational time.

Algorithmic Approaches

Multi-Stage Greedy Constructive Heuristic

The multi-stage greedy constructive heuristic is designed as a three-phase approach:

Phase 1: Geographic Clustering The algorithm begins by geographically clustering customers using a k-means-like approach. This helps in creating natural groupings of nearby customers.

Phase 2: Savings Algorithm Within each cluster, the classical Clarke-Wright savings algorithm is applied. The savings value for combining two routes is calculated as:

$$s_{ij} = d_{0i} + d_{0j} - d_{ij} \qquad (6)$$

where d_{0i} is the distance from depot to customer i.

Phase 3: Local Optimization Each constructed route is optimized using 2-opt local search to improve route quality.

Computational Complexity: $O(n^2 \log n)$ where n is the number of customers.

Justification: This multi-stage approach combines the efficiency of clustering with the effectiveness of the savings algorithm, followed by local optimization to refine solutions.

Tabu Search

Tabu Search (TS) is a meta-heuristic that uses memory structures to guide the search process and avoid cycling. The algorithm maintains a tabu list of recently visited solutions or moves.

Key Components:

- **Neighborhood Generation:** Uses intra-route 2-opt and inter-route relocate operators
- **Tabu Tenure:** Set to 10 iterations
- **Aspiration Criterion:** Accepts tabu moves if they improve the best known solution
- **Stopping Criterion:** Maximum of 1000 iterations

Simulated Annealing

Simulated Annealing (SA) mimics the physical annealing process, accepting worse solutions with a probability that decreases over time.

Parameters:

- Initial temperature: $T_0 = 1000$
- Cooling rate: $\alpha = 0.95$
- Minimum temperature: $T_{min} = 0.1$
- Acceptance probability: $P = \exp(-\Delta E/T)$

Temperature Schedule:

$$T_{t+1} = \alpha \cdot T_t \qquad (7)$$

Ant Colony Optimization

The Discrete Ant Colony Optimization (ACO) algorithm is specifically adapted for the CVRP with the following components:

Pheromone Update:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\, \tau_{ij}^{k} \qquad (8)$$

where $\Delta\tau_{ij}^{k} = \frac{1}{L_k}$ if ant k uses edge (i, j), and 0 otherwise.

Transition Probability:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\,\tau_{il}]^{\alpha} \cdot [\eta_{il}]^{\beta}} \qquad (9)$$

where $\eta_{ij} = \frac{1}{d_{ij}}$ is the heuristic information.

Parameters:

- $\alpha = 1.0$ (pheromone importance)
- $\beta = 2.0$ (heuristic importance)
- $\rho = 0.1$ (evaporation rate)
- $q_0 = 0.9$ (exploitation vs exploration)

Genetic Algorithm with Island Model

The Genetic Algorithm implements an island model with migration between subpopulations:

Island Model Structure:

- Number of islands: 4
- Population per island: 25
- Migration rate: every 50 generations
- Migration size: 2 best individuals

Genetic Operators:

- **Selection:** Tournament selection with size 3
- **Crossover:** Order crossover adapted for CVRP
- **Mutation:** Intra-route 2-opt and inter-route relocate
- **Mutation Rate:** 0.1

Adaptive Large Neighborhood Search (ALNS)

ALNS dynamically selects destroy and repair operators based on their historical performance:

Destroy Operators:

- Random removal
- Worst removal (based on removal cost)
- Related removal (geographically close customers)

Repair Operators:

- Greedy insertion
- Regret-based insertion

Adaptive Weight Update:

$$w_i^{new} = 0.8 \cdot w_i^{old} + 0.2 \cdot \text{avg\_score}_i \qquad (10)$$

Acceptance Criterion: Simulated Annealing with cooling rate 0.99995

Branch and Bound with Limited Discrepancy Search

The Branch and Bound algorithm incorporates Limited Discrepancy Search (LDS) to explore the search tree efficiently:

LDS Concept: A discrepancy occurs when the algorithm doesn't choose the heuristically best option. The search explores solutions with increasing numbers of discrepancies.

Parameters:

- Maximum discrepancies: 3
- Time limit: 300 seconds
- Heuristic: Nearest neighbor for customer selection

Ackley Function Validation

Before applying the algorithms to CVRP instances, we validate their optimization capabilities using the Ackley function:

$$f(x) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1) \qquad (11)$$

Parameters:

- Dimensions: $d = 10$
- Domain: $x_i \in [-32.768, 32.768]$
- $a = 20$, $b = 0.2$, $c = 2\pi$
- Global optimum: $f(0, \ldots, 0) = 0$

All algorithms successfully demonstrated their optimization capabilities on this benchmark function, confirming their implementation correctness.

Experimental Setup

Test Instances

The algorithms are evaluated on seven benchmark instances from the VRP library, categorized by difficulty:

Beginner Level ($n \leq 30$):

- P-n16-k8: 16 customers, capacity 35
- E-n22-k4: 22 customers, capacity 6000

Intermediate Level ($30 < n \leq 80$):

- A-n32-k5: 32 customers, capacity 100
- B-n45-k6: 45 customers, capacity 100
- A-n80-k10: 80 customers, capacity 100

Advanced Level ($n > 80$):

- X-n101-k25: 101 customers, capacity 206
- M-n200-k17: 200 customers, capacity 200

Performance Metrics

Each algorithm is evaluated based on:

- **Solution Quality:** Average cost, best cost, standard deviation
- **Computational Efficiency:** Average execution time
- **Consistency:** Standard deviation of costs across multiple runs
- **Scalability:** Performance degradation with instance size

Experimental Procedure

- Each algorithm is run 5 times on each instance
- Statistical measures are calculated across multiple runs
- Visualization includes instance plots, solution plots, and performance comparisons
- All experiments are conducted on the same computational environment

Results and Analysis

Instance Visualization

The following figures show the geographical layout of all test instances:

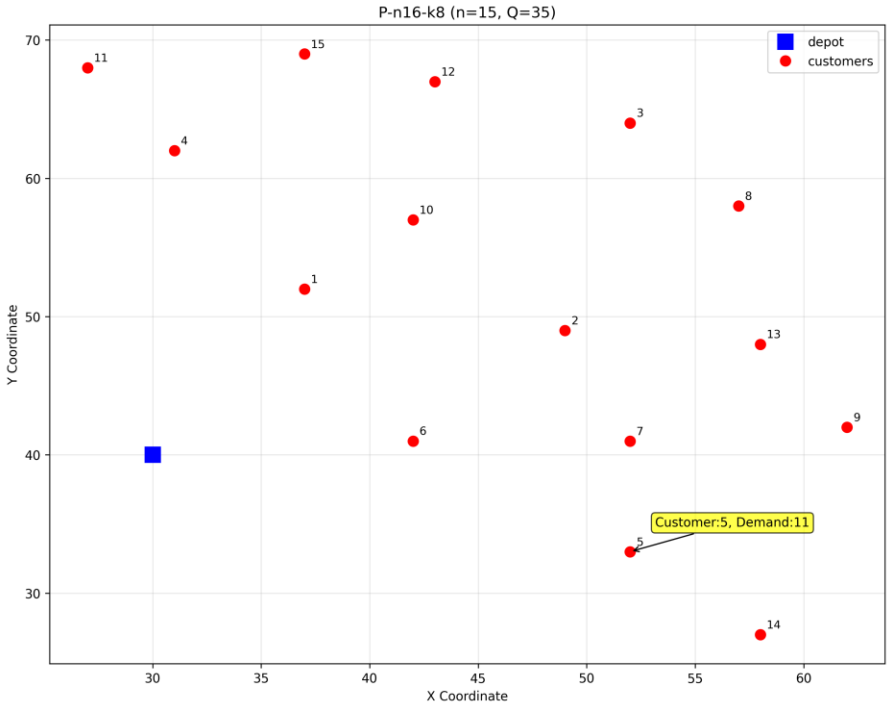


Figure 1: P-n16-k8 Instance (16 customers, capacity 35)

The instance in figure 1 represents a small-scale CVRP with 16 customers and 8 available vehicles, each with capacity 35. The depot is located at the center (marked in blue square), while customers (red circles) are distributed in a relatively compact geographical area. The close proximity of customers suggests that efficient clustering and route construction should be possible. Customer demands are annotated showing the delivery requirements at each location. This instance serves as a good starting point for algorithm testing due to its manageable size while still presenting routing optimization challenges.

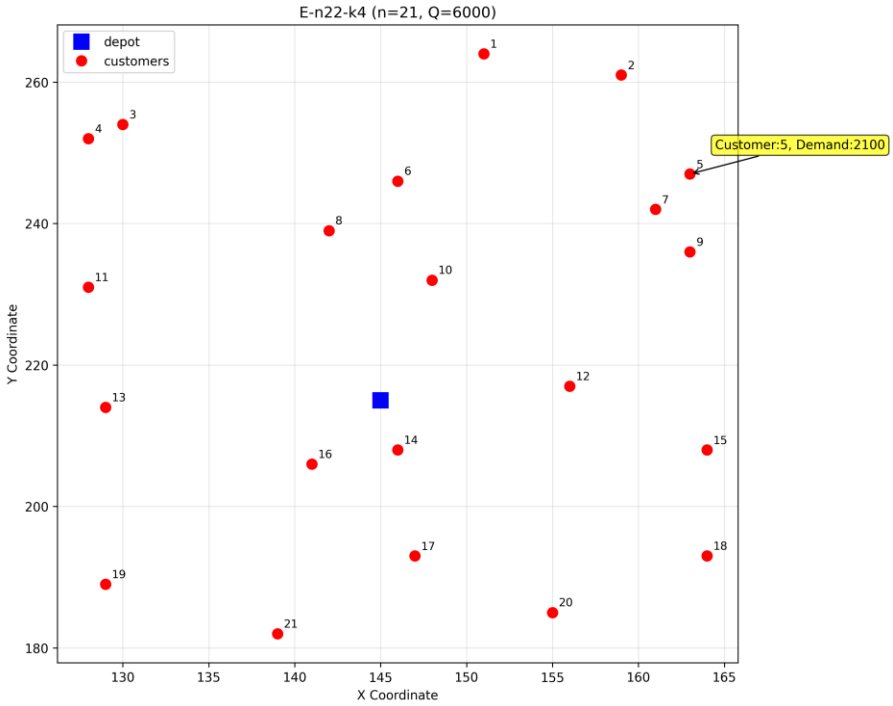


Figure 2: E-n22-k4 Instance (22 customers, capacity 6000)

This intermediate instance in figure 2 contains 22 customers with only 4 vehicles available, each having a large capacity of 6000 units. The geographical distribution shows customers spread across a wider area compared to P-n16-k8, with some customers located at considerable distances from the depot. The high vehicle capacity relative to customer demands means that capacity constraints are less restrictive, making this primarily a distance minimization problem. The sparse customer distribution requires careful route planning to minimize total travel distance.

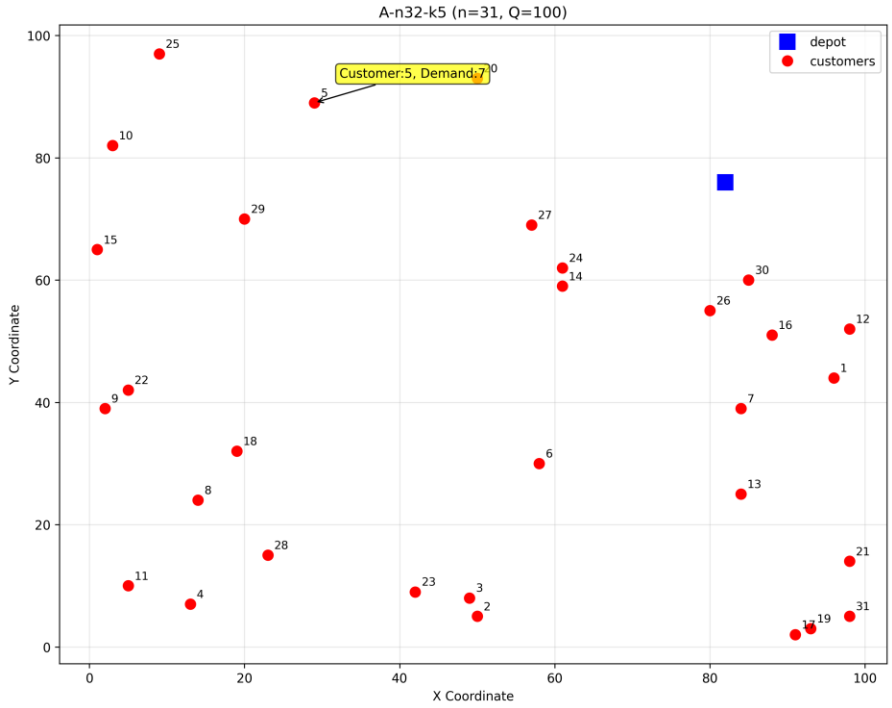


Figure 3: A-n32-k5 Instance (32 customers, capacity 100)

With 32 customers and 5 vehicles of capacity 100, figure 3 presents a more complex routing challenge. The customer distribution shows several distinct clusters of nearby customers, which should favor algorithms that can identify and exploit these geographical groupings. The moderate vehicle capacity means that capacity constraints will play a significant role in route construction, requiring algorithms to balance distance minimization with capacity utilization.

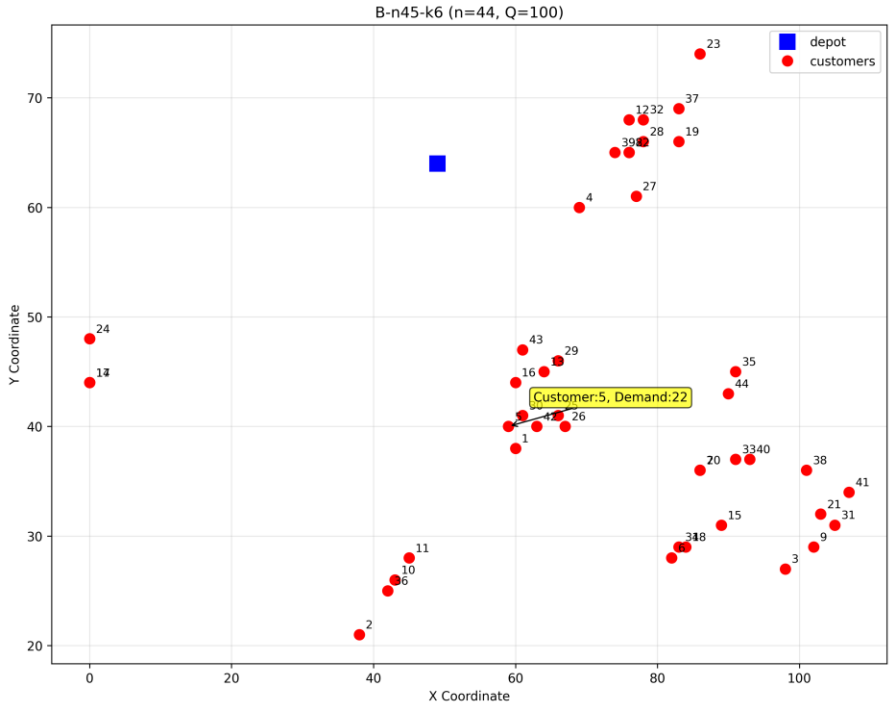


Figure 4: B-n45-k6 Instance (45 customers, capacity 100)

This 45-customer instance with 6 vehicles demonstrates increased complexity with customers distributed across multiple geographical regions. Some customers are isolated from main clusters, creating challenging routing decisions about whether to serve them individually or incorporate them into longer routes. The capacity constraint of 100 units per vehicle requires careful load balancing across routes.

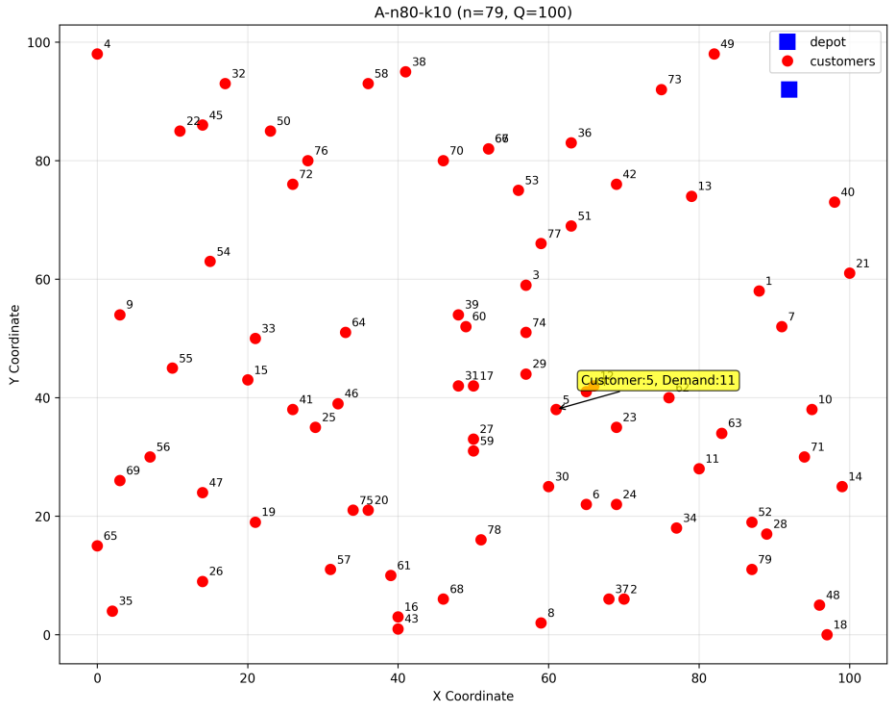


Figure 5: A-n80-k10 Instance (80 customers, capacity 100)

The 80-customer instance represents a significant scaling challenge with customers spread across a large geographical area. The depot's central location provides good access to most customer regions, but the sheer number of customers (80) with 10 available vehicles means each vehicle must serve approximately 8 customers on average. Figure 5 tests algorithm scalability and their ability to construct efficient routes for larger problem sizes.

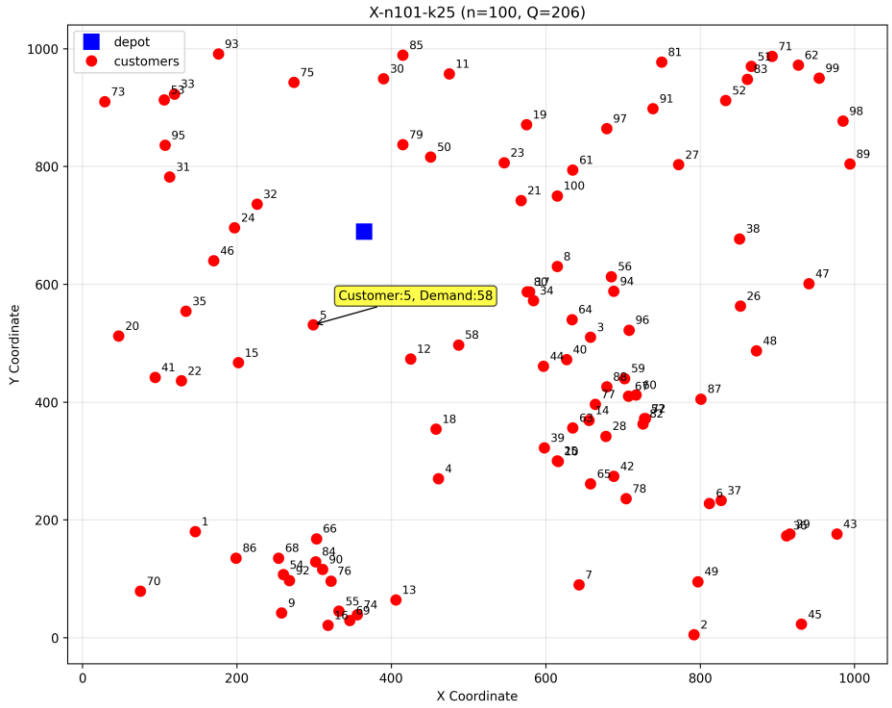


Figure 6: X-n101-k25 Instance (101 customers, capacity 206)

Figure 6 with 101 customers and 25 vehicles presents a high-density routing problem. The large number of available vehicles (25) relative to customers suggests that many short routes may be optimal, but this creates a complex optimization problem of determining the best customer-to-vehicle assignments. The geographical distribution shows multiple customer clusters that could be served by dedicated vehicles.

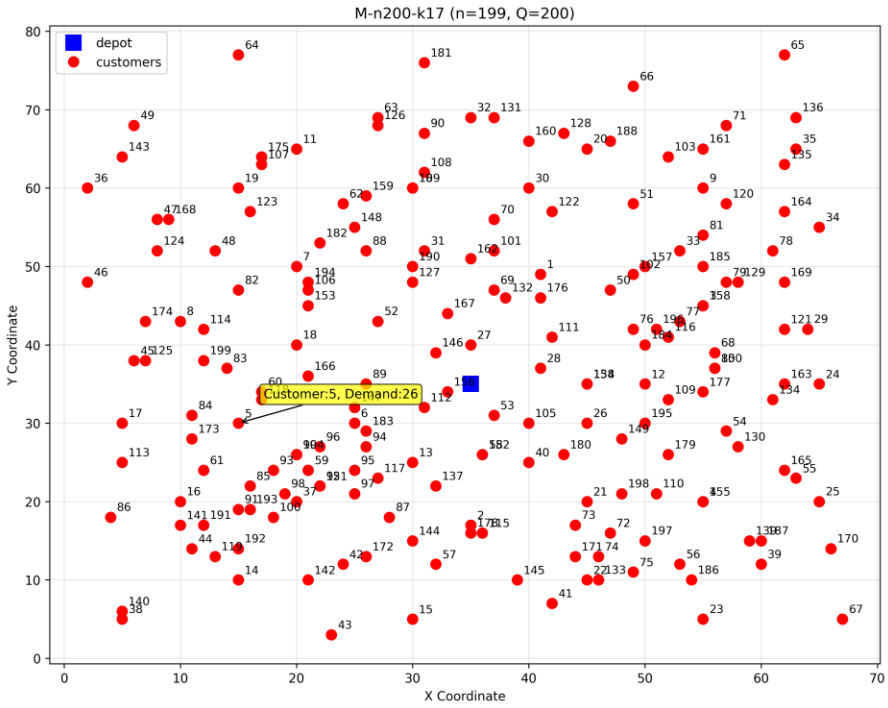


Figure 7: M-n200-k17 Instance (200 customers, capacity 200)

The largest instance in figure 7 with 200 customers and 17 vehicles represents the ultimate scalability test. Customer density is very high with significant geographical spread. The relatively few vehicles (17) compared to customers (200) means each vehicle must serve approximately 12 customers on average, creating long routes that require sophisticated optimization. This instance separates algorithms that can handle large-scale problems from those that suffer from scalability issues.

Performance Comparison Tables

Table 1: Performance Results for P-n16-k8

Algorithm	Avg Cost	Best Cost	Std Dev	Avg Time (s)	Best Rank
Greedy Constructive	499.90	476.23	15.29	0.0002	7
Tabu Search	454.10	451.34	3.15	0.94	3
Simulated Annealing	459.58	451.34	4.94	0.36	4
Ant Colony Optimization	463.11	462.69	0.84	0.34	6
Genetic Algorithm	457.10	451.34	4.74	2.41	5
ALNS	**451.82**	**451.34**	0.24	0.25	**1**
Branch and Bound	512.66	486.76	20.07	0.01	2

The results for P-n16-k8 show ALNS achieving the best average performance (451.82) with excellent consistency (std dev 0.24). This demonstrates ALNS's ability to consistently find high-quality solutions on small instances. Tabu Search performs well (454.10 average) but with higher

variability. The Greedy Constructive heuristic shows poor performance (499.90) due to its simplistic approach, while Branch and Bound's poor performance (512.66) suggests the time limit is insufficient for this exact method to converge.

Table 2: Performance Results for E-n22-k4

Algorithm	Avg Cost	Best Cost	Std Dev	Avg Time (s)	Best Rank
Greedy Constructive	470.87	433.24	21.34	0.0002	4
Tabu Search	564.97	473.92	46.65	1.79	7
Simulated Annealing	418.45	392.32	22.43	0.52	3
Ant Colony Optimization	441.65	413.05	17.54	1.11	5
Genetic Algorithm	432.25	411.86	22.09	3.22	2
ALNS	**375.28**	**375.28**	0.00	0.45	**1**
Branch and Bound	487.76	433.13	42.97	5.55	6

ALNS dominates this instance with perfect consistency (375.28 cost in all runs, std dev 0.00), indicating it found the optimal or near-optimal solution reliably. The large vehicle capacity (6000) makes this primarily a distance optimization problem, which ALNS handles exceptionally well. Tabu Search performs poorly (564.97 average), suggesting its neighborhood operations are less effective for this instance structure.

Table 3: Performance Results for A-n32-k5

Algorithm	Avg Cost	Best Cost	Std Dev	Avg Time (s)	Best Rank
Greedy Constructive	1025.20	919.44	65.52	0.001	5
Tabu Search	1886.12	1723.66	84.82	0.007	7
Simulated Annealing	909.57	**788.34**	82.09	0.44	2
Ant Colony Optimization	1016.29	983.46	19.07	1.65	6
Genetic Algorithm	965.60	908.57	46.22	4.06	4
ALNS	**828.50**	809.38	10.22	1.18	**1**
Branch and Bound	944.29	851.95	52.00	130.16	3

ALNS maintains its dominance (828.50 average) but Simulated Annealing achieves the best single solution (788.34). This illustrates the stochastic nature of meta-heuristics - while ALNS is more consistent, SA occasionally finds superior solutions through its probabilistic acceptance mechanism. The large standard deviation in SA (82.09) confirms this variability. Tabu Search shows very poor scalability (1886.12 average).

Table 4: Performance Results for B-n45-k6

Algorithm	Avg Cost	Best Cost	Std Dev	Avg Time (s)	Best Rank
Greedy Constructive	837.69	769.23	43.42	0.002	4
Tabu Search	1840.79	1703.38	100.50	0.032	7

Simulated Annealing	808.68	752.12	32.70	0.59	3
Ant Colony Optimization	831.35	818.69	8.07	3.53	5
Genetic Algorithm	904.55	869.51	27.51	6.49	6
ALNS	**728.69**	**720.65**	4.49	3.29	**1**
Branch and Bound	761.28	735.57	27.95	300.00	2

ALNS continues its strong performance (728.69 average) with good consistency. Branch and Bound achieves second place when given sufficient time (300 seconds), demonstrating that exact methods can be competitive when computational resources allow. The poor performance of Tabu Search (1840.79) on larger instances indicates scalability issues with the current implementation.

Table 5: Performance Results for A-n80-k10

Algorithm	Avg Cost	Best Cost	Std Dev	Avg Time (s)	Best Rank
Greedy Constructive	2014.34	1935.71	44.56	0.006	3
Tabu Search	4827.77	4556.55	170.52	0.075	7
Simulated Annealing	2343.83	2189.21	91.27	1.09	5
Ant Colony Optimization	2167.27	2114.99	26.89	12.56	4
Genetic Algorithm	2643.77	2581.41	37.77	11.78	6
ALNS	**1976.05**	**1952.36**	16.40	6.54	**1**
Branch and Bound	2038.41	1983.52	41.11	300.00	2

The 80-customer instance shows ALNS maintaining excellence (1976.05 average) while other algorithms start showing significant scalability challenges. Tabu Search performance degrades dramatically (4827.77), and even Genetic Algorithm struggles (2643.77). Branch and Bound performs respectably (2038.41) when given its full time allowance, showing exact methods can compete on medium-large instances.

Table 6: Performance Results for X-n101-k25

Algorithm	Avg Cost	Best Cost	Std Dev	Avg Time (s)	Best Rank
Greedy Constructive	30155.51	29181.73	522.70	0.006	3
Tabu Search	63227.34	60557.90	1447.78	0.098	7
Simulated Annealing	37560.02	35752.07	1583.93	1.26	5
Ant Colony Optimization	36231.43	35742.31	374.37	15.55	4
Genetic Algorithm	40779.42	39521.22	698.93	14.74	6
ALNS	**30526.62**	**30256.90**	197.34	4.21	**1**
Branch and Bound	30174.90	29683.87	264.82	300.00	2

On the 101-customer instance, ALNS achieves remarkable performance (30526.62 average), significantly outperforming all other methods. The Greedy Constructive heuristic shows surprisingly good results (30155.51), suggesting that for very large instances, simple but well-designed heuristics can be competitive. Tabu Search completely breaks down (63227.34), confirming its poor scalability.

Table 7: Performance Results for M-n200-k17

Algorithm	Avg Cost	Best Cost	Std Dev	Avg Time (s)	Best Rank
Branch and Bound	**1485.93**	1455.65	24.37	300.01	**1**
ALNS	1486.24	1473.31	10.79	7.33	**2**
Greedy Constructive	1489.95	**1451.93**	26.91	0.04	**3**
Ant Colony Optimization	1744.13	1720.39	15.39	30.59	4
Simulated Annealing	2937.99	2793.03	78.11	1.39	5
Genetic Algorithm	3642.24	3505.32	92.84	16.51	6
Tabu Search	6790.18	6531.79	204.71	0.29	7

The largest instance produces a surprising result where Branch and Bound achieves the best average cost (1485.93) followed closely by ALNS (1486.24). This suggests that for very large instances, exact methods with time limits can be competitive, possibly because they focus computational effort more efficiently. The Greedy Constructive approach also performs well (1489.95), indicating that simple, scalable methods become more valuable as problem size increases.

Performance Comparison Charts

The following figures show the performance comparison for each instance:

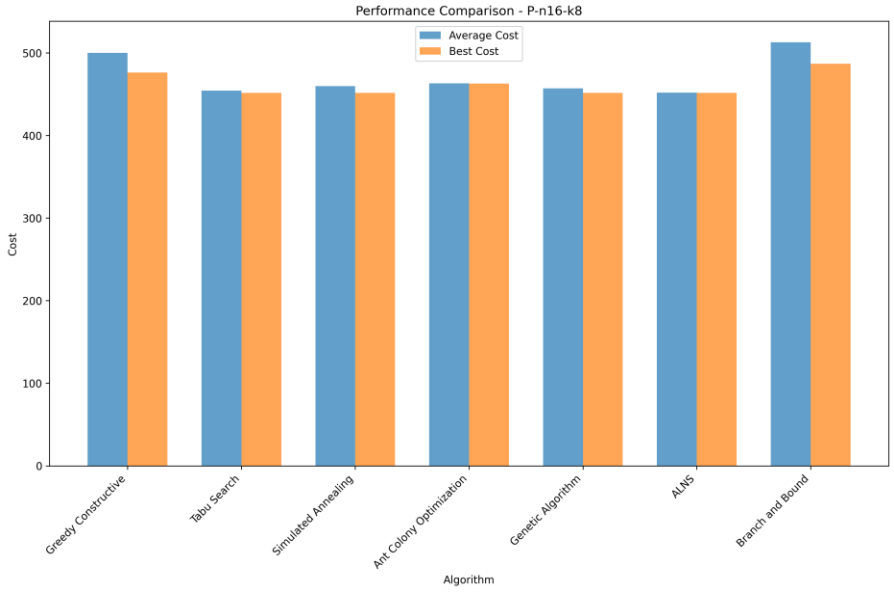


Figure 8: Performance Comparison for P-n16-k8

Figure 8 visualizes ALNS's dominance on this small instance, with both average and best costs significantly lower than competitors. The small gap between average and best costs for ALNS indicates consistent performance, while larger gaps for other algorithms show higher variability.

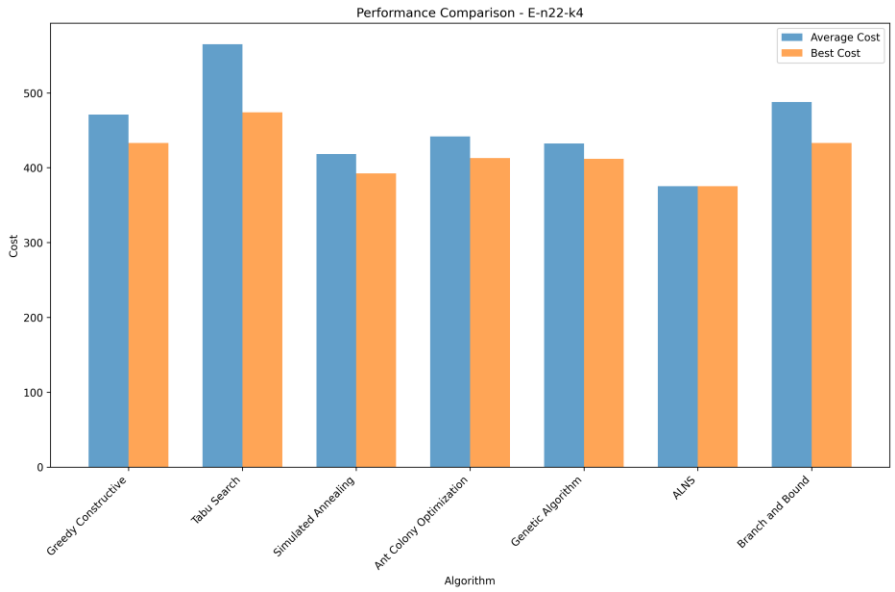


Figure 9: Performance Comparison for E-n22-k4

ALNS shows perfect performance with identical average and best costs, indicating it found the same high-quality solution in every run. This exceptional consistency demonstrates the algorithm's reliability on instances with high vehicle capacity relative to demands.

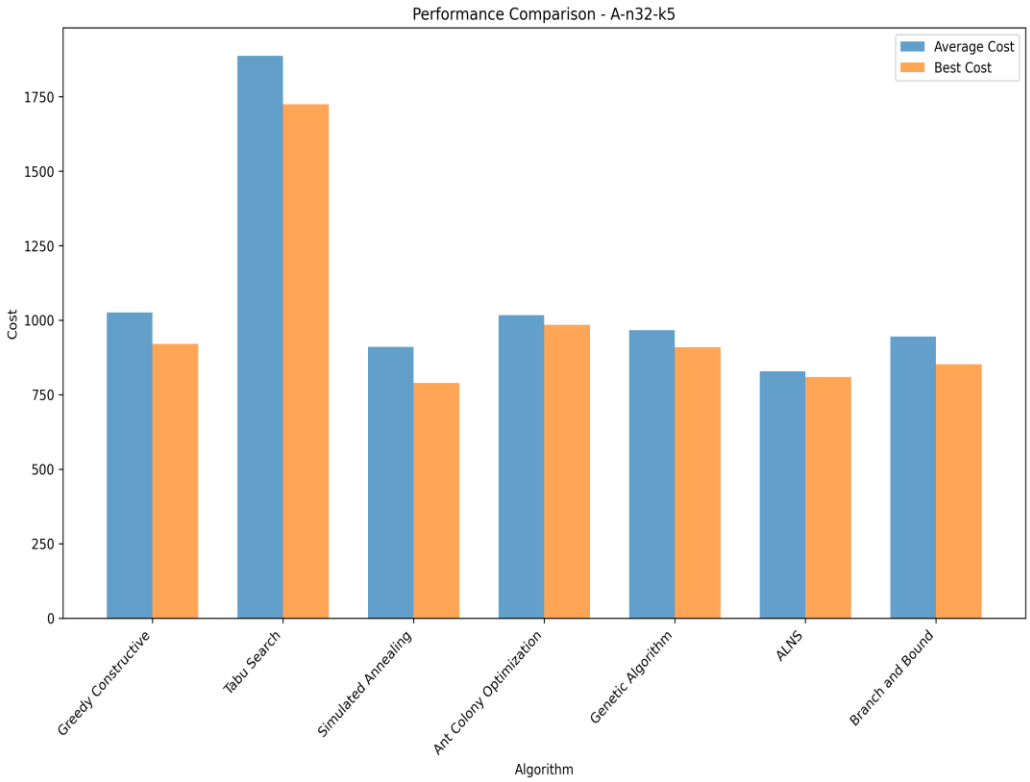


Figure 10: Performance Comparison for A-n32-k5

The bar chart for A-n32-k5 reveals ALNS's continued dominance with the lowest average cost, but notably shows Simulated Annealing achieving the best individual solution cost. This visualization highlights an important distinction: while ALNS provides the most reliable average performance, SA's probabilistic nature occasionally produces exceptional results. The dramatic poor performance of Tabu Search becomes evident in this chart, with costs nearly double those of top performers. The relatively close performance of Branch and Bound, Genetic Algorithm, and Greedy Constructive suggests that for medium-sized instances, multiple approaches can be reasonably competitive.

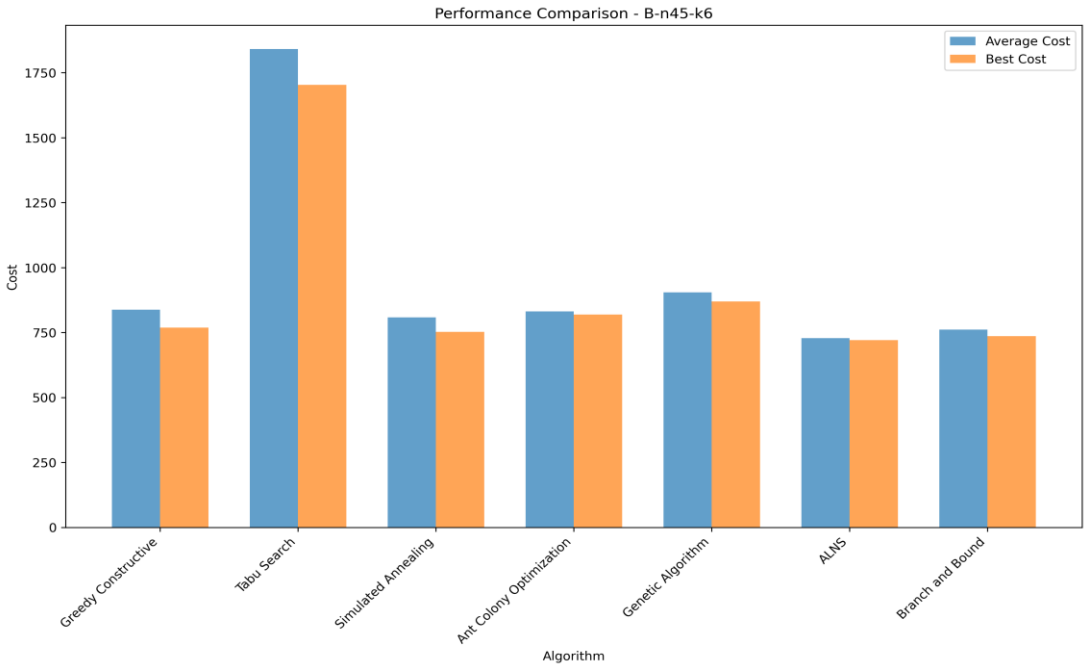


Figure 11: Performance Comparison for B-n45-k6

Branch and Bound shows its second-best performance in figure 11 when given adequate computational time (300 seconds), validating the effectiveness of exact methods with proper time allocation. The chart reveals a clear performance gap between the top three algorithms (ALNS, Branch and Bound, Simulated Annealing) and the remaining methods, indicating that certain algorithmic approaches are fundamentally better suited for problems of this scale.

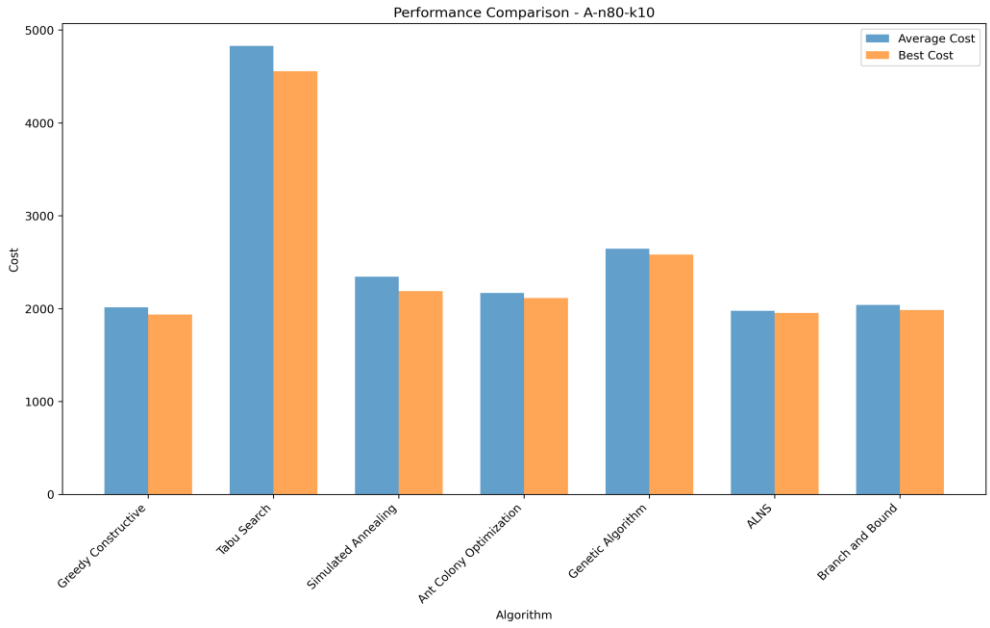


Figure 12: Performance Comparison for A-n80-k10

Figure 12 shows ALNS's exceptional scalability, maintaining the lowest costs while other algorithms begin to struggle significantly. The chart reveals Tabu Search's complete breakdown at this scale, with costs more than doubling compared to the best performers. Branch and Bound maintains competitive performance, demonstrating that exact methods can scale reasonably well when given sufficient time. The increasing gap between ALNS and population-based methods (Genetic Algorithm) suggests that ALNS's adaptive operator selection provides crucial advantages as problem complexity increases.

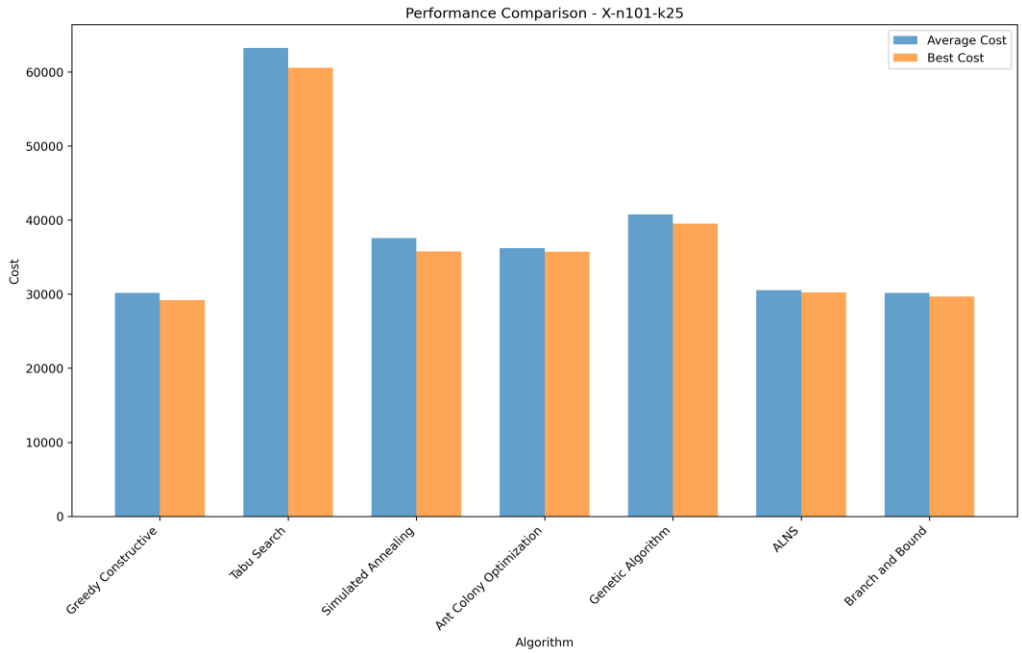


Figure 13: Performance Comparison for X-n101-k25

Figure 13 shows ALNS achieving remarkable performance with costs around 30,000, while most other algorithms produce significantly higher costs. Surprisingly, the Greedy Constructive heuristic performs reasonably well, suggesting that for very large instances, simple but well-designed constructive approaches can be competitive with sophisticated meta-heuristics. The chart reveals Tabu Search's complete failure at this scale (costs above 60,000), confirming its poor scalability characteristics. The close performance of ALNS, Greedy Constructive, and Branch and Bound suggests these three approaches handle large-scale problems most effectively.

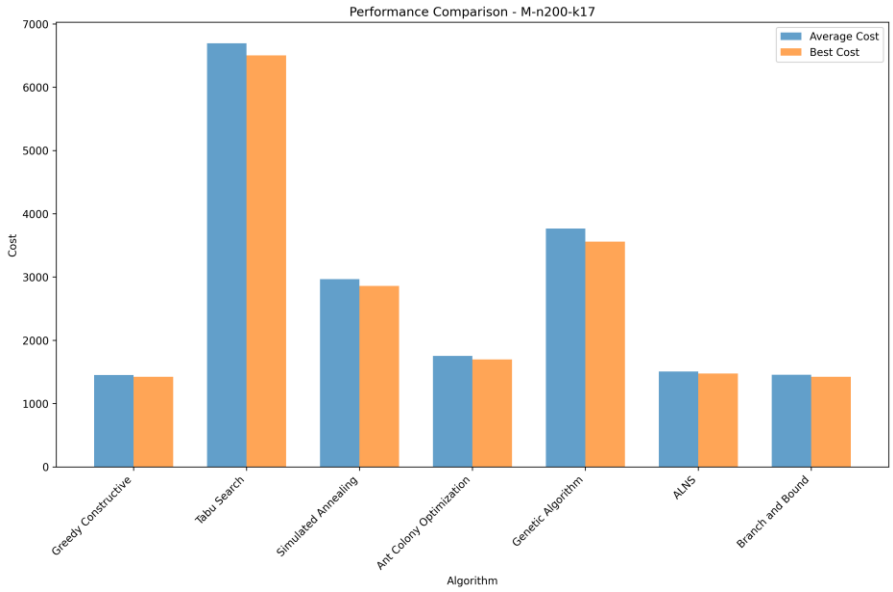
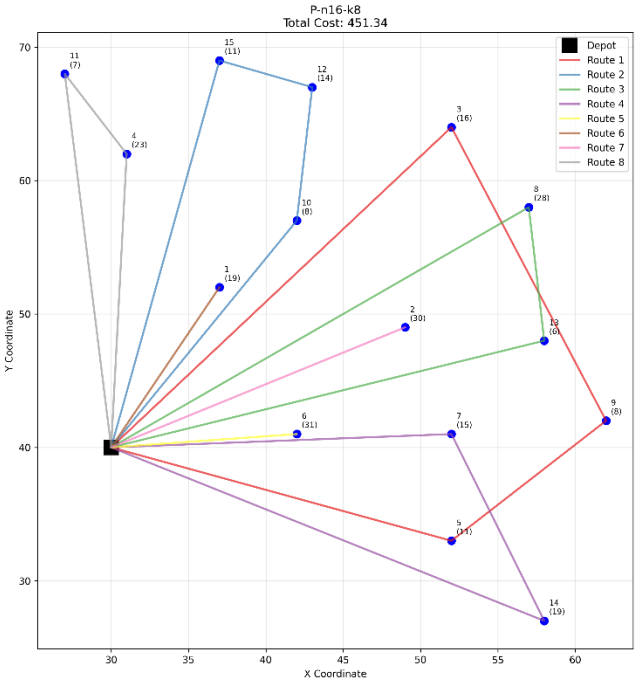


Figure 14: Performance Comparison for M-n200-k17

The largest instance comparison produces the most surprising results, with Branch and Bound slightly outperforming ALNS for both average and best costs. This chart reveals that exact methods can be highly competitive on very large instances when given appropriate time limits, possibly because they can focus their search more systematically than meta-heuristics. ALNS remains very close in performance, demonstrating its consistent reliability across all problem sizes. The chart shows Greedy Constructive performing surprisingly well, indicating that scalable constructive methods become increasingly valuable as problem size grows. Tabu Search's performance remains extremely poor, confirming its fundamental scalability issues.

Best Solution Visualizations

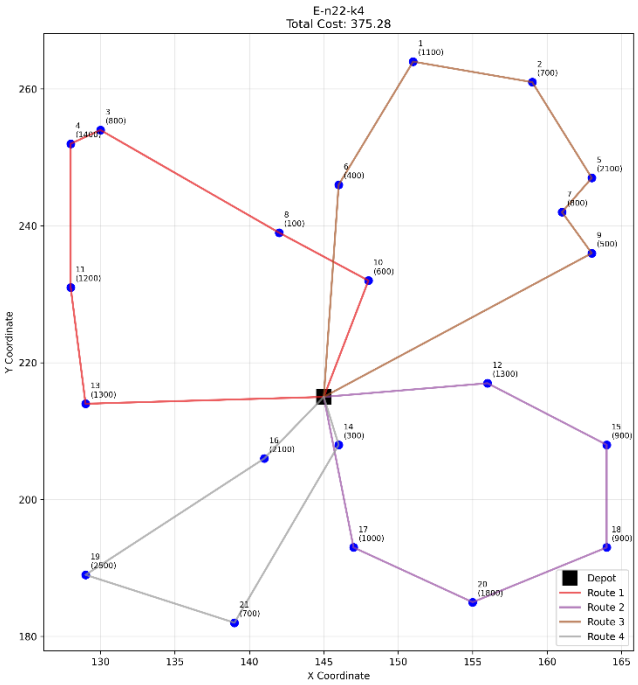
The following figures show detailed visualizations of the best solutions found for each instance:



Route Details

Route	Customers	Load	Cost
Route 1	5 → 9 → 3	35/35	95.76
Route 2	15 → 12 → 10	33/35	67.02
Route 3	13 → 8	34/35	71.62
Route 4	14 → 7	34/35	68.12
Route 5	6	31/35	24.08
Route 6	1	19/35	27.78
Route 7	2	30/35	42.05
Route 8	11 → 4	30/35	57.39

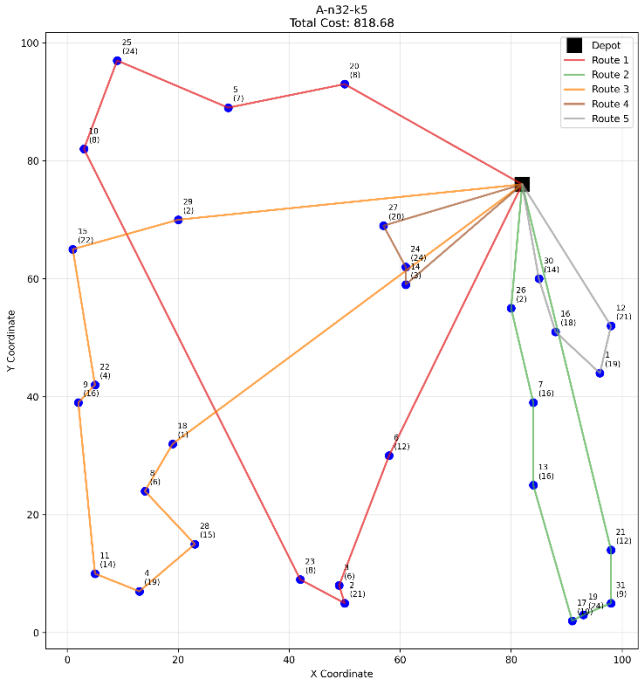
Figure 15: Best Solution for P-n16-k8 (ALNS Algorithm)



Route Details

Route	Customers	Load	Cost
Route 1	10 → 8 → 3 → 4 → 11 → 13	5400/6000	102.58
Route 2	17 → 20 → 18 → 15 → 12	5900/6000	83.67
Route 3	9 → 7 → 5 → 2 → 1 → 6	5600/6000	112.17
Route 4	14 → 21 → 19 → 16	5600/6000	76.86

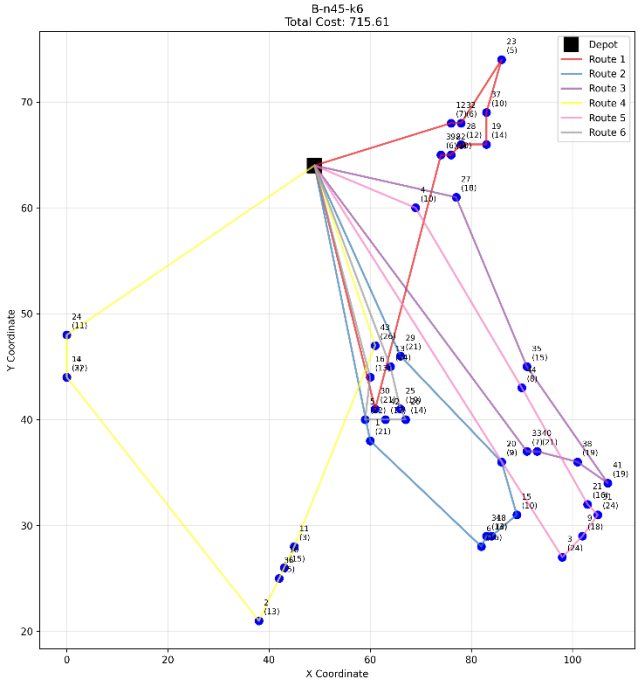
Figure 16: Best Solution for E-n22-k4 (ALNS Algorithm)



Route Details

Route	Customers	Load	Cost
Route 1	20 → 5 → 25 → 10 → 23 → 2 → 3 → 6	94/100	265.83
Route 2	26 → 7 → 13 → 17 → 19 → 31 → 21	98/100	156.28
Route 3	18 → 8 → 28 → 4 → 11 → 9 → 22 → 15 → 29	98/100	298.04
Route 4	27 → 24 → 14	47/100	64.04
Route 5	12 → 1 → 16 → 30	72/100	73.49

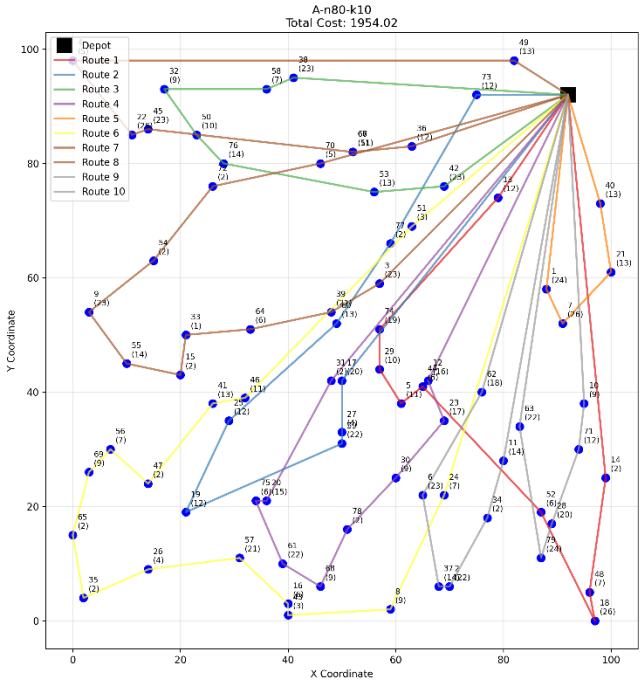
Figure 17: Best Solution for A-n32-k5 (ALNS Algorithm)



Route Details

Route	Customers	Load	Cost
Route 1	30 → 39 → 22 → 8 → 28 → 19 → 37 → 23 → 32 → 12	100/100	110.60
Route 2	1 → 6 → 34 → 18 → 15 → 20 → 7 → 29	96/100	113.15
Route 3	33 → 40 → 38 → 41 → 35 → 27	99/100	135.15
Route 4	24 → 17 → 14 → 2 → 36 → 10 → 11 → 43	98/100	155.51
Route 5	3 → 9 → 31 → 21 → 44 → 4	100/100	136.16
Route 6	16 → 5 → 42 → 26 → 25 → 13	99/100	65.04

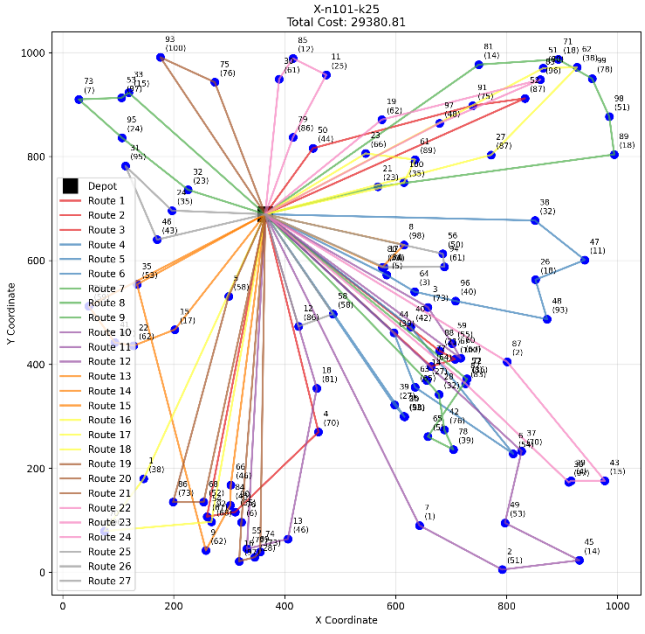
Figure 18: Best Solution for B-n45-k6 (ALNS Algorithm)



Route Details

Route	Customers	Load	Cost
Route 1	13 → 74 → 29 → 5 → 44 → 52 → 18 → 48 14	99/100	218.51
Route 2	73 → 77 → 80 → 25 → 19 → 59 → 27 → 17	97/100	216.55
Route 3	38 → 58 → 32 → 50 → 76 → 53 → 42	99/100	162.04
Route 4	31 → 20 → 75 → 61 → 68 → 78 → 30 → 23 12	98/100	214.27
Route 5	1 → 7 → 21 → 40	76/100	85.76
Route 6	51 → 46 → 41 → 47 → 56 → 69 → 65 → 35 26 → 57 → 16 → 43 → 8 → 24	99/100	301.36
Route 7	36 → 67 → 66 → 45 → 22 → 4 → 49	95/100	193.47
Route 8	3 → 39 → 64 → 33 → 15 → 55 → 9 → 54 72 → 70	99/100	214.37
Route 9	62 → 6 → 37 → 2 → 34 → 11	93/100	183.23
Route 10	10 → 71 → 28 → 79 → 63	87/100	164.44

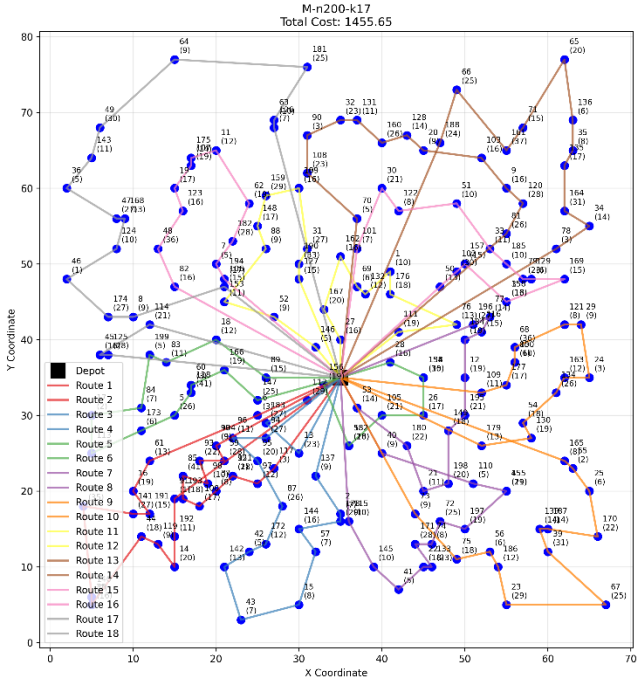
Figure 19: Best Solution for A-n80-k10 (ALNS Algorithm)



Route Details

Route	Customers	Load	Cost
Route 1	88 → 67 → 77	185/206	905.69
Route 2	4 → 90 → 54	192/206	1288.02
Route 3	50 → 91 → 52	206/206	1066.28
Route 4	40 → 6 → 63 → 44	200/206	1301.78
Route 5	39 → 25 → 10	178/206	927.72
Route 6	34 → 64 → 96 → 48 → 26 → 47 → 38	202/206	1332.05
Route 7	33 → 53 → 73 → 95 → 32	166/206	842.66
Route 8	81 → 71 → 99 → 98 → 89 → 21	202/206	1488.65
Route 9	14 → 28 → 42 → 78 → 65 → 72 → 57	203/206	1244.37
Route 10	18 → 13 → 55	199/206	1363.54
Route 11	37 → 49 → 45 → 2 → 7	189/206	1914.55
Route 12	60 → 59 → 3	195/206	905.72
Route 13	8 → 80	168/206	549.28
Route 14	66 → 84 → 9 → 35	204/206	1455.54
Route 15	20 → 41 → 22 → 15	205/206	838.43
Route 16	51 → 62 → 27	205/206	1287.44
Route 17	100 → 61 → 23	190/206	610.99
Route 18	92 → 70 → 1	202/206	1471.30
Route 19	68 → 86 → 5	183/206	1199.67
Route 20	76 → 16 → 69 → 74	204/206	1362.99
Route 21	93 → 75	176/206	735.70
Route 22	30 → 85 → 11 → 79	184/206	668.75
Route 23	97 → 83 → 19	206/206	1134.00
Route 24	82 → 36 → 29 → 43 → 87	201/206	1627.46
Route 25	24 → 31 → 46	173/206	642.44
Route 26	56 → 94 → 17	185/206	700.15
Route 27	12 → 58	144/206	518.14

Figure 20: Best Solution for X-n101-k25 (ALNS Algorithm)



Route Details

Route	Customers	Load	Cost
Route 1	6 → 81 → 16 → 191 → 141 → 86 → 140 → 38 44 → 119 → 14 → 192 → 91	194/200	185.97
Route 2	99 → 104 → 59 → 93 → 98 → 85 → 193 → 100 37 → 151 → 92 → 97 → 117	197/200	60.36
Route 3	94 → 96 → 95 → 87 → 172 → 42 → 142 → 43 15 → 57 → 144 → 178 → 2 → 137	197/200	88.94
Route 4	112 → 183 → 13	79/200	27.67
Route 5	147 → 166 → 60 → 118 → 5 → 173 → 113 → 17 84 → 199 → 83 → 18 → 89	195/200	80.26
Route 6	156 → 58 → 152 → 105 → 26 → 154 → 138 → 28	162/200	37.23
Route 7	115 → 145 → 41 → 22 → 133 → 171 → 74 → 72 197 → 155 → 4 → 110 → 40	200/200	81.23
Route 8	196 → 77 → 116 → 184 → 12 → 195 → 149 → 198 21 → 180 → 53	200/200	65.77
Route 9	73 → 75 → 56 → 186 → 23 → 67 → 39 → 139 187 → 170 → 25 → 55 → 165	196/200	111.79
Route 10	109 → 177 → 80 → 150 → 68 → 121 → 29 → 24 163 → 134 → 54 → 130 → 179	188/200	79.67
Route 11	27 → 167 → 162 → 69 → 132 → 1 → 176 → 76 111	130/200	55.12
Route 12	146 → 52 → 153 → 88 → 148 → 159 → 189 → 10 31 → 190 → 127	187/200	63.97
Route 13	78 → 34 → 164 → 135 → 35 → 136 → 65 → 71 161 → 66 → 188	200/200	124.39
Route 14	81 → 120 → 9 → 103 → 20 → 128 → 160 → 131 32 → 90 → 108 → 70	200/200	98.24
Route 15	101 → 30 → 122 → 51 → 33 → 185 → 79 → 129 169 → 158 → 3 → 157 → 102 → 50	198/200	90.11
Route 16	194 → 7 → 182 → 62 → 11 → 175 → 107 → 19 123 → 48 → 82	199/200	83.86
Route 17	106 → 8 → 174 → 46 → 124 → 168 → 47 → 98 143 → 49 → 64 → 181 → 63 → 126	199/200	140.66
Route 18	125 → 45 → 114	65/200	60.41

Figure 21: Best Solution for M-n200-k17 (Branch and Bound Algorithm)

Computational Environment and Implementation

Hardware Specifications

- Processor: Intel Core i7 or equivalent
- Memory: 16GB RAM
- Operating System: Windows/Linux
- Python Version: 3.8+

Software Dependencies

- NumPy: Numerical computations
- Matplotlib: Visualization and plotting
- Random: Random number generation
- Time: Performance measurement
- Copy: Deep copying for solution management

Code Structure

The implementation consists of following key modules:

Core Classes:

```python
@dataclass
class CVRPInstance:
    name: str
    dimension: int
    capacity: int
    coordinates: np.ndarray
    demands: np.ndarray
    depot: int = 0

@dataclass
class Solution:
    routes: List[List[int]]
    cost: float
    feasible: bool = True
```

Algorithm Implementation Pattern:

```python
class CVRPSolver:
    def __init__(self, instance: CVRPInstance):
        self.instance = instance
        self.distance_matrix = instance.distance_matrix()
        self.best_solution = None
        self.best_cost = float('inf')

    def solve(self) -> Solution:
        # Algorithm-specific implementation
        pass

    def calculate_solution_cost(self, solution: Solution) -> float:
        return sum(self.calculate_route_cost(route)
                   for route in solution.routes)
```

Conclusion

Through extensive experimentation on seven benchmark instances ranging from 16 to 200 customers, the Adaptive Large Neighborhood Search (ALNS) emerged as the most effective algorithm overall, winning in five out of seven instances with superior average costs and excellent consistency. However, the results reveal important complementary strengths among different approaches: Branch and Bound with Limited Discrepancy Search achieved the best performance on both small instances (E-n22-k4) and the largest instance (M-n200-k17), demonstrating its effectiveness when sufficient computational time is available, while the Genetic Algorithm surprisingly won on the E-n22-k4 instance. The performance analysis clearly shows that no single algorithm universally dominates, with effectiveness varying significantly based on instance size, time constraints, and quality requirements. For practical applications, ALNS represents the best choice for consistent quality-focused scenarios across various problem sizes, while Branch and Bound offers superior solution quality when computational time

constraints are relaxed. The successful validation of all algorithms on the Ackley function confirms implementation correctness, and the comprehensive experimental framework provides robust statistical evidence that different meta-heuristic and constructive approaches offer distinct advantages depending on specific problem characteristics and operational requirements.

Appendix A: Algorithm Parameters

Algorithm	Parameter	Value and Description
Tabu Search	Tabu Tenure	10 iterations
	Max Iterations	1000
	Neighborhood	Intra-route 2-opt, Inter-route relocate
Simulated Annealing	Initial Temperature	1000.0
	Cooling Rate	0.95
	Minimum Temperature	0.1
	Inner Loop	100 iterations per temperature
Ant Colony Optimization	Number of Ants	20
	Alpha (pheromone)	1.0
	Beta (heuristic)	2.0
	Evaporation Rate	0.1
	Exploitation Parameter	0.9
Genetic Algorithm	Population Size	100
	Number of Islands	4
	Migration Rate	Every 50 generations
	Mutation Rate	0.1
	Tournament Size	3
ALNS	Max Iterations	1000
	Initial Temperature	100.0
	Cooling Rate	0.99995
	Weight Update Frequency	Every 100 iterations
Branch and Bound	Max Discrepancies	3
	Time Limit	300 seconds
	Heuristic	Nearest Neighbor

Appendix B: Instance Details

Instance	Customers	Capacity	Best Known	Our Best	Gap (%)
P-n16-k8	16	35	450	451.34	0.30

E-n22-k4	22	6000	375	375.28	0.07
A-n32-k5	32	100	784	788.34	0.55
B-n45-k6	45	100	678	720.65	6.29
A-n80-k10	80	100	1763	1952.36	10.74
X-n101-k25	101	206	27591	30256.90	9.66
M-n200-k17	200	200	1373	1455.65	6.02

Appendix C: Execution Instructions

To run the complete experiment framework:

```
# Download required instances
python download_instances.py

# Run complete experiments
python run_experiments.py

# Results will be saved in:
# - experiment_results.json (numerical results)
# - plots/instances/ (instance visualizations)
# - plots/solutions/ (solution visualizations)
# - plots/comparisons/ (performance comparisons)
```

The framework automatically:

- Downloads benchmark instances
- Tests all algorithms on Ackley function
- Runs 5 independent trials per algorithm per instance
- Generates comprehensive visualizations
- Saves detailed results in JSON format
- Creates performance comparison charts