# Capacitated Vehicle Routing Problem Report

# Table of Contents

**1. Introduction**

1.1 Problem Statement

1.2 Background & Importance

1.3 Problem Representation

1.4 Fitness Function

**2. Methodology**

2.1 Genetic Algorithm (Evolutionary Approach)

2.2 Greedy Search (Non-Evolutionary)

2.3 Random Search (Baseline)

2.4 Simulated Annealing (Metaheuristic)

**3. Experimental Setup**

3.1 Instances Used

3.2 Parameters Tested

3.3 Evaluation Metrics

3.4 Run Configuration

**4. Results & Analysis**

4.1 Summary of Genetic Algorithm results

4.2 Convergence Behavior

4.3 Comparison with Baseline Methods

4.4 Parameter Influence

4.5 Observations

**5. Conclusion**

# 1. Introduction

## 1.1 Problem Statement

The Capacitated Vehicle Routing Problem (CVRP) is a classical combinatorial optimization problem in the field of logistics and transportation. It involves determining the optimal set of routes for a fleet of vehicles that must deliver goods to a set of customers from a central depot. Each customer has a specific demand, and each vehicle has a limited carrying capacity.  
The goal is to minimize the total distance traveled (or total cost), while ensuring that:

* Each customer is visited exactly once by a single vehicle
* The total demand on any route does not exceed the vehicle’s capacity
* All routes start and end at the central depot.

This problem is critical in real-world logistics and supply chain operations, such as delivery services, waste collection, and distribution of goods, where efficient route planning can significantly reduce operational costs and environmental impact.  
In this project, we implement and analyze solution strategies for the Capacitated Vehicle Routing Problem (CVRP), with a particular focus on comparing the performance of evolutionary algorithms—such as Genetic Algorithms—with non-evolutionary and alternative metaheuristic approaches, including Greedy Search, Random Search, and Simulated Annealing.

## 1.2 Background & Importance

The CVRP extends the classical Traveling Salesman Problem by introducing multiple vehicles and capacity constraints, making it significantly more complex and computationally challenging. Since finding exact solutions becomes infeasible for large instances, metaheuristic algorithms such as Genetic Algorithms (GAs) are widely used to obtain near-optimal solutions efficiently.  
  
Solving CVRP effectively has practical importance across various industries, enabling cost reduction, improved delivery performance, and better resource utilization.

## 1.3 Problem Representation

In the context of evolutionary algorithms, representing a solution to the CVRP is a critical design choice that impacts the performance and correctness of the algorithm.  
  
A typical representation is a permutation of customer nodes, where the depot is implied and used to segment routes based on vehicle capacity constraints.

For example, a chromosome might look like:  
*[3, 5, 1, 2, 4, 6]*  
  
This sequence represents the order in which customers are visited. To convert this into a valid CVRP solution:

* Start from the depot.
* Traverse the customer list in order.
* Keep track of the vehicle's remaining capacity.
* When the next customer's demand cannot be fulfilled, insert a depot return and start a new route with a fresh vehicle.

Example route segmentation with capacity checks:  
*[Depot → 3 → 5 → Depot → 1 → 2 → 4 → Depot → 6 → Depot]*  
  
This structure ensures:

* Each customer is visited exactly once.
* Vehicle capacity is respected.
* All routes start and end at the depot.

## 1.4 Fitness Function

The fitness function is a critical component of the evolutionary algorithm, as it evaluates the quality of each solution (individual) in the population. In the Capacitated Vehicle Routing Problem (CVRP), the fitness of a solution is typically measured by the total distance traveled across all routes.  
  
Objective: Minimize the total distance traveled by all vehicles while fulfilling customer demands and respecting vehicle capacity constraints.  
  
Fitness Calculation Steps:

1. Segment the customer sequence into feasible routes based on vehicle capacity.
2. For each route:  
    Start and end at the depot.  
    Sum the distances between consecutive locations (using the distance matrix).
3. Accumulate the total distance across all routes.

Then the total fitness (cost) is:  
f(x) = ∑₍ᵣ ∈ ᴿ₎ ∑ᵢ₌₀^{|r|-1} d(rᵢ, rᵢ₊₁)  
Where r₀ and r\_{|r|+1} are both the depot.

* Lower fitness values indicate better solutions.
* Infeasible solutions (e.g., exceeding capacity) may be penalized or repaired.
* This function guides the selection and evolution process.

# 2. Methodology

## 2.1 Genetic Algorithm (Evolutionary Approach)

The Genetic Algorithm (GA) is inspired by natural evolution. It starts with a population of randomly generated solutions and iteratively evolves them using genetic operators. Each individual represents a customer visit sequence, and its fitness is calculated based on the total route distance with depot returns and capacity limits.  
  
Key features:

* Representation: Permutation of customer IDs (no depot included)
* Fitness Function: Total distance traveled, segmented by vehicle capacity
* Selection: Tournament-based (configurable size)
* Crossover: Ordered Crossover (OX)
* Mutation: Swap mutation
* Elitism: Top E individuals preserved each generation (E ∈ {1, 3, 5})
* Fitness Curve Logging: Best, Average, Worst fitness recorded per generation
* Evaluation Budget: Fixed as populationSize × generations × runs

***Pseudocode:***

## *GeneticAlgorithm(CVRP problem, GAConfig config):*

## *Initialize population with random permutations*

## *Evaluate fitness of all individuals*

## *Set globalBest as best individual*

## *For generation in 1 to config.generations:*

## *newPopulation = []*

## *// Elitism: preserve top E individuals*

## *elites = selectTopIndividuals(population, config.elitismCount)*

## *While newPopulation.size < (populationSize - elitismCount):*

## *parent1 = tournamentSelect(population, config.tournamentSize)*

## *parent2 = tournamentSelect(population, config.tournamentSize)*

## *If rand() < config.Px:*

## *childRoute = OX\_Crossover(parent1.route, parent2.route)*

## *Else:*

## *childRoute = copy(parent1.route)*

## *If rand() < config.Pm:*

## *applySwapMutation(childRoute)*

## *child = new Individual(childRoute)*

## *child.evaluateFitness()*

## *newPopulation.add(child)*

## *Add elites to newPopulation*

## *population = newPopulation*

## *Update globalBest if improved*

## *Return globalBest, and fitness curve (for logging)*

## 2.2 Greedy Search (Non-Evolutionary)

The Greedy algorithm constructs a solution incrementally by always choosing the closest unvisited customer that can be served without exceeding the vehicle’s capacity. When no such customer is available, the vehicle returns to the depot and starts a new route.

***Pseudocode:***

*GreedySearch(CVRP problem):  
 Initialize route as empty  
 Set current = depot, load = 0  
 While unvisited customers remain:  
 Find nearest unvisited customer within capacity  
 If found:  
 Add to route, update current and load  
 Else:  
 Return to depot, reset load  
 Return route as Individual and evaluate fitness*

## 2.3 Random Search (Baseline)

Random Search generates a specified number of completely random valid solutions. Each candidate is evaluated, and the best one is selected. It serves as a simple baseline for performance comparison.

***Pseudocode:***

RandomSearch(CVRP problem, evaluations):  
 best = null  
 For i from 1 to evaluations:  
 Generate random permutation of customers  
 Build route from permutation  
 Evaluate fitness  
 If better than best:  
 best = current  
 Return best

## 2.4 Simulated Annealing (Metaheuristic)

Simulated Annealing (SA) is a metaheuristic inspired by the annealing process in metallurgy. It starts with a random solution and explores neighboring solutions while gradually reducing the probability of accepting worse solutions, allowing it to escape local optima.

***Pseudocode:***

*SimulatedAnnealing(CVRP problem, T\_init, T\_final, coolingRate):  
 current = random valid solution  
 best = current  
  
 While T > T\_final:  
 Repeat iterationsPerTemp times:  
 neighbor = swap two cities in current  
 Δ = neighbor.fitness - current.fitness  
 If Δ < 0 or exp(-Δ / T) > random(0,1):  
 current = neighbor  
 If current is better than best:  
 best = current  
 T = T \* coolingRate  
 Return best*

# 3. Experimental Setup

This section outlines the dataset instances, parameter configurations, and evaluation metrics used in the experiments. The goal is to compare the performance of Genetic Algorithm (GA), Greedy Search, Random Search, and Simulated Annealing (SA) on a variety of problem sizes and configurations.

## 3.1 Instances Used

The experiments use benchmark instances from the CVRP Set A, specifically:  
A-n32-k5, A-n37-k6, A-n39-k5, A-n45-k6, A-n48-k7, A-n54-k7, A-n60-k9  
Each `.vrp` file specifies the depot, customer locations, demands, and vehicle capacity. Instances vary in size and vehicle constraints to assess scalability.

## 3.2 Parameters Tested

A Taguchi L16 orthogonal array was used to systematically test different combinations of GA parameters. Each configuration was run 10 independent times per instance and per elitism level. The total number of configurations per instance was:  
16 Taguchi configs × 3 elitism levels = 48 configurations per instance

The parameter levels included:

* Population Sizes: 100, 400, 800, 1000
* Generations: 50, 200, 400, 1000
* Crossover Rates (Px): 0.7, 0.8, 0.9, 1.0
* Mutation Rates (Pm): 0.01, 0.03, 0.05, 0.10
* Tournament Sizes: 3, 5, 7, 9
* Elitism Count: 1, 3, 5

Each configuration used deterministic evaluation budgets defined as:  
Evaluations = populationSize × generations × 10 runs

## 3.3 Evaluation Metrics

Each algorithm's performance was assessed using the following metrics:

* Best Fitness: Lowest total distance achieved
* Worst Fitness: Highest total distance observed
* Average Fitness: Mean distance over 10 runs
* Standard Deviation: Variation in results
* Execution Time: Total time taken across all 10 runs
* Evaluations: Total number of fitness evaluations performed (equalized across configurations)

## 3.4 Run Configuration

All algorithms were implemented in Java. To ensure fair comparison of execution times, the Genetic Algorithm (GA) was run in sequential (single-threaded) mode with no parallelism.

The TaguchiTuner class automated the execution of all 48 GA configurations per instance, derived from a Taguchi L16 orthogonal array and 3 elitism levels. Each configuration was evaluated using 10 independent runs, initialized with consistent random seeds for reproducibility.

For each configuration, the following were recorded:

* Final performance metrics: Best, Worst, Average, Standard Deviation
* Execution time for all 10 runs
* Deterministic evaluation count, computed as:

Evaluations = populationSize × generations × 10 runs

* Generation-by-generation fitness logs: Best, Average, and Worst fitness per generation, saved for convergence analysis

Baseline algorithms (Greedy Search, Random Search, Simulated Annealing) were executed once per instance using default configurations. All algorithms shared the same solution representation, decoding logic, and fitness evaluation to ensure consistency and fairness.

**4. Results & Analysis**

This section presents the results obtained from running the Genetic Algorithm across various parameter configurations, compared with non-evolutionary approaches. For each instance, the configuration that yielded the lowest best fitness value is reported. Additionally, graphs showing the relationship between the number of generations and best fitness are included.

**4.1 Summary of results**

The best-performing Genetic Algorithm (GA) configurations for each CVRP instance are summarized in the table below. These were selected from the Taguchi experimental runs based on lowest average fitness across 10 trials.

**Genetic Algorithm**



Across all instances, GA delivered robust and competitive performance. The best results were typically obtained using:

* **High population sizes (800–1000)**
* **Longer evolution (400–1000 generations)**
* **Moderate to low mutation rates (0.01–0.05)**
* **Elitism count of 1**, which appeared to preserve diversity and allow continued exploration.

Larger instances like **A-n60-k9** and **A-n54-k7** showed greater variance, as expected due to their complexity. Even so, GA achieved strong best-case results and reasonable average performance, suggesting solid scalability. Notably, many of the top-performing configurations shared similar parameter settings, reinforcing the value of systematic tuning via the Taguchi method.

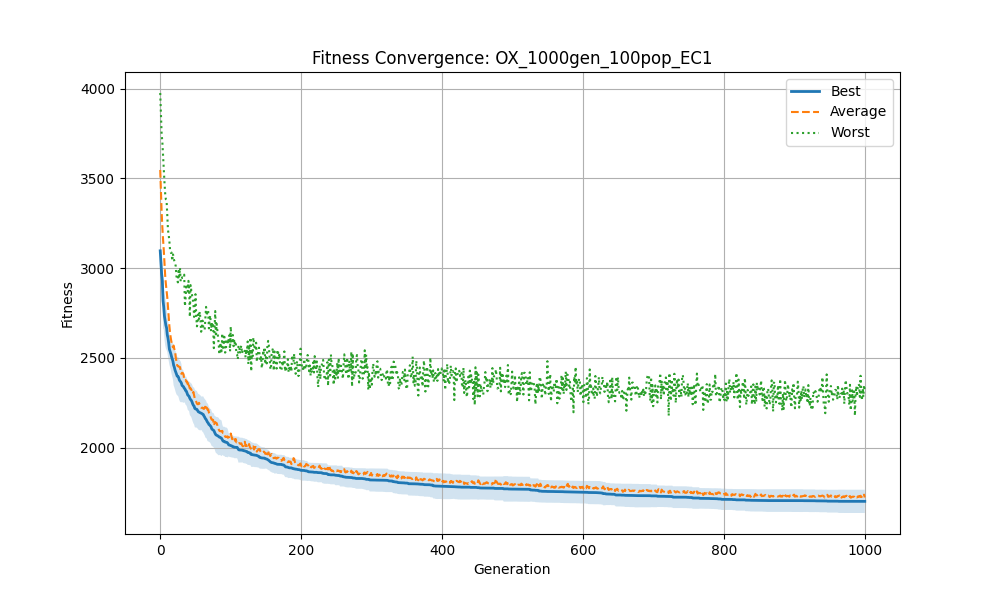
**4.2 Convergence Behavior**

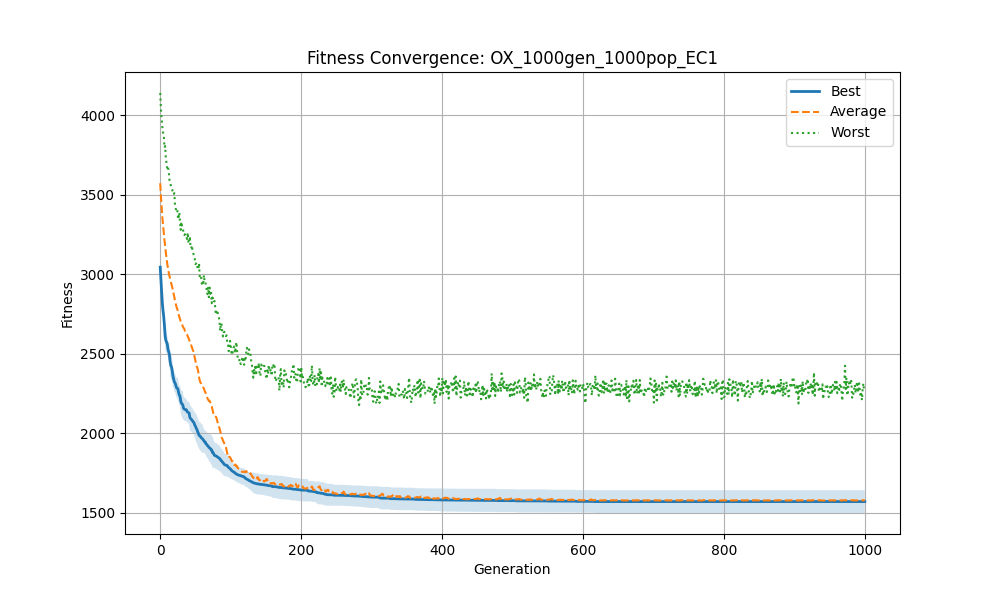
**A graph of a graph

AI-generated content may be incorrect.**

**A graph of a fitness convergence

AI-generated content may be incorrect.**





To better understand the dynamic behavior of the Genetic Algorithm (GA) across different configurations, convergence plots were analyzed for several runs using Elitism Count = 1 (EC=1). These plots depict how fitness (cost) evolves across generations, showcasing the algorithm’s ability to balance exploration and exploitation.

Small Configuration: pop:100, gen:50

In this low-resource configuration, the GA demonstrates rapid early improvement in both best and average fitness. However, convergence slows significantly before generation 50, and the gap between best and worst solutions remains noticeable. This suggests limited exploration and early stagnation, likely due to insufficient population diversity and evolution time.

Medium Configuration: pop:400, gen:400

Here, convergence is significantly more gradual and stable. The best and average curves steadily decline until around generation 300, after which the algorithm approaches a plateau. The narrow shaded area indicates low variance, showing consistent performance across runs. This configuration appears to strike a good balance between exploration and convergence speed.

High Generation, Small Population: pop:100, gen:1000

With a small population and long evolution, the GA quickly improves in the first 100–200 generations. However, convergence flattens as early as generation 300. The small population likely leads to early loss of diversity, causing premature convergence despite the high number of generations.

High Configuration: pop:1000, gen:1000

This configuration demonstrates the strongest convergence behavior. The best and average fitness both improve rapidly and continue refining until near the final generations. The close overlap between best and average fitness, along with a very small standard deviation, indicates stable convergence and effective exploitation. This confirms the advantage of combining large population size with deep evolution.

Insights:

* Short runs (e.g., 50gen) can improve quickly but often plateau early.
* Large populations and longer generations offer more stability and deeper convergence.
* EC=1 effectively maintains diversity in early phases, allowing meaningful exploration.
* The shape of the best vs. average fitness gap reveals how consistently the population is improving.

These observations reinforce the importance of careful parameter tuning, and further validate why configurations like pop:1000, gen:1000 consistently produced strong performance in the final results.

**4.3 Comparison with Baseline Methods**

**4.3.1 Random Algorithm**

Random Search performed the worst overall. It had the highest average fitness and the widest spread between best and worst solutions. This reflects its unguided nature — while it can occasionally find a decent solution, its lack of learning or feedback means it wastes evaluations on unpromising areas of the search space. Even with fast runtime, Random Search is clearly not a viable alternative for meaningful optimization in CVRP.



**4.3.2 Greedy Algorithm**

The Greedy Search algorithm produced solutions almost instantly (under 0.2 seconds), making it computationally efficient. However, as expected from a deterministic and myopic method, its solution quality was poor relative to GA and other heuristics. It consistently produced the same result across runs, resulting in a standard deviation of zero. While it sometimes approached GA’s results in small instances (e.g., A-n32-k5), it failed to scale effectively on larger problems, with fitness values diverging significantly from the GA's best.



**4.3.3 Metaheuristics – Simulated Annealing Algorithm**

Surprisingly, **Simulated Annealing outperformed the Genetic Algorithm** on multiple CVRP instances in terms of **best** and **average fitness**. For example:

* On **A-n32-k5**, SA achieved a best of **810.69**, compared to GA's **820.39**
* On **A-n39-k5**, SA outperformed GA in both best and average metrics
* SA also maintained **lower standard deviations**, suggesting more stable convergence

This result suggests that **SA’s focused local search and probabilistic acceptance of worse solutions** allowed it to escape local minima more effectively than GA in some settings. In contrast, the GA may have **suffered from premature convergence** or required more tuning of its parameters (e.g., population size, elitism).



**4.3.4 Genetic Algorithm with Local search**

To improve the performance of the Genetic Algorithm, a local search component was added to refine offspring solutions during evolution. This hybrid approach, referred to as GA+LS, consistently outperformed the standard GA and even outmatched Simulated Annealing in most CVRP instances. For example, in A-n39-k5, GA+LS achieved a best fitness of 835.23, compared to 851.09 for SA and 888.93 for standard GA. The inclusion of local search led to significant improvements in both solution quality and stability, with reduced standard deviations across runs.

However, these gains came at a computational cost. Due to the intensive nature of applying local search in each generation, runtimes increased substantially—especially on larger instances like A-n60-k9, where execution time exceeded one full day. Despite this, the trade-off appears worthwhile when solution accuracy is prioritized.

The integration of local search highlights the strength of memetic algorithms in solving combinatorial problems like CVRP. It suggests that while evolutionary strategies provide exploration, combining them with exploitation-focused techniques like local search can yield highly competitive results.



**4.4 Parameter Influence**

To understand the impact of various Genetic Algorithm parameters on performance, the Taguchi L16 orthogonal array was used to systematically test combinations of population size, number of generations, crossover and mutation rates, tournament size, and elitism count.

Key observations from parameter influence:

* Population Size and Generations (pop × gen):  
  Larger population sizes (800–1000) combined with longer runs (400–1000 generations) consistently produced better results. This suggests that solution diversity and sufficient evolutionary time are critical for convergence in CVRP.
* Crossover Rate (Px):  
  Higher crossover rates (0.9–1.0) generally led to stronger performance, emphasizing the role of recombination in exploring new areas of the solution space.
* Mutation Rate (Pm):  
  Mutation rates around 0.03–0.05 offered the best balance. Too low (e.g., 0.01) resulted in premature convergence, while too high (e.g., 0.10) introduced excessive randomness, destabilizing convergence.
* Tournament Size:  
  Moderate tournament sizes (5–7) produced more reliable results than extreme values. Smaller tournaments lacked selection pressure, while larger ones risked convergence to local optima.
* Elitism Count:  
  An elitism value of 3 frequently appeared in top-performing configurations, striking a balance between preserving strong individuals and allowing diversity. Too little elitism led to regression; too much reduced exploration.

Overall, the analysis supports the idea that parameter synergy—not just individual values—determines performance. Configurations with high exploration (mutation, diversity) and stable exploitation (elitism, selection pressure) delivered the most reliable results.

**4.5 Observations**

The experiments reveal several important insights about solving the CVRP with evolutionary and heuristic methods:

* GA+Local Search consistently outperformed all other methods, but at the cost of increased runtime. This highlights the power of hybrid or memetic approaches in combinatorial optimization.
* Simulated Annealing performed better than standard GA in some cases, especially in small- to mid-sized instances. This was an unexpected but informative result, showing that local search can outperform population-based methods when the latter are not well-tuned.
* Greedy Search was extremely fast but not scalable, often producing poor results on larger instances. Its deterministic nature and lack of global perspective limited its performance.
* Random Search consistently underperformed, confirming the necessity of guided search strategies in VRP problems.
* The convergence behavior of GA varied significantly with configuration. Some runs showed rapid early convergence followed by stagnation, especially with low mutation or high elitism.
* Parameter tuning and logging (e.g., fitness curves) were essential in identifying weaknesses like early stagnation or poor exploration.

**5. Conclusions**

This study applied and evaluated several optimization algorithms for solving the Capacitated Vehicle Routing Problem (CVRP), with a focus on Genetic Algorithms (GA), Random and Greedy Search, and Simulated Annealing (SA). The performance of the GA was extensively analyzed through a systematic Taguchi parameter tuning process, while hybridization with local search was explored to further enhance solution quality.

The experiments demonstrated that:

* Simulated Annealing initially outperformed the standard GA, particularly on small-to-medium-sized instances. Its ability to perform deep local exploration allowed it to escape local optima and produce stable results.
* Genetic Algorithm with Local Search (GA+LS) surpassed all baseline methods in both best and average fitness across nearly all instances. However, it incurred a significant increase in runtime, particularly for larger instances.
* Greedy Search proved computationally efficient but myopic, producing poor-quality solutions that did not scale well.
* Random Search, while fast, consistently failed to find competitive solutions due to its unguided nature.
* The influence of parameters such as population size, mutation rate, and elitism was substantial. Larger populations, moderate mutation, and balanced elitism (e.g., EC=3) led to more robust and stable performance.

Overall, the study confirms the value of hybrid metaheuristics and systematic parameter tuning for solving complex combinatorial problems like CVRP. While Simulated Annealing offers a strong baseline, memetic approaches (like GA+LS) represent a powerful strategy when runtime is acceptable. Future work could explore adaptive parameter control, parallel implementations, and hybrid combinations of GA and SA for even greater performance.