Analysis of the Efficiency of Selected Optimization Algorithms

May 2025

Contents

1 Theoretical Introduction

The aim of the project was to compare the efficiency of selected optimization algorithms in solving the Capacitated Vehicle Routing Problem (CVRP).

The Capacitated Vehicle Routing Problem (CVRP) is an extension of the classical Traveling Salesman Problem (TSP), in which one must plan routes for a fleet of vehicles so that:

- each customer is visited exactly once,
- the demand of each customer does not exceed any vehicle's capacity,
- each route starts and ends at the same depot,
- the total cost (usually measured as total route length) is minimized.

CVRP belongs to the class of NP-hard problems—the number of possible solutions grows exponentially with the number of customers. Metaheuristics do not guarantee finding the optimal solution but allow obtaining high-quality solutions in a reasonable time, which is crucial for real-world logistics applications.

1.1 Objective Function

Comparison of results was conducted as follows. A solution is represented as a vector of consecutive points to visit, where the value 0 denotes a return to the depot. The objective function value is calculated as the total distance needed to visit all points—the smaller, the better. During evaluation, consecutive sequences of zeros are treated as a single event, ensuring a more organized search of the solution space and eliminating multiple "visits" to the same solution.

The number of zeros inserted corresponds to the number of cities minus one, increasing the combinatorial solution space from $N!$ to $(2N-1)!$. Although this significantly enlarges the theoretical search space, it allows capturing routes that might be missed by a greedy generation without zeros. However, the introduction of extra zeros also introduces some redundancy. Since zero-sequences can be treated as a single element, the effective solution space is significantly reduced, allowing faster discovery of near-optimal routes.

1.2 Neighborhood Operators

In the algorithms used in this experiment, two neighborhood sampling operators (mutations) were implemented: *swap* and *reversal*.

1.2.1 Swap

Two indices of the solution vector are chosen at random. The procedure repeats until at least one of the chosen points is non-zero (i.e., it allows one endpoint to be a depot return, provided the other is an actual customer). Once the condition is met, the values at these two positions are swapped.

1.2.2 Reversal

The reversal operator also begins by randomly selecting two indices, subject to the same criterion as the swap operator, and the subsequence must not consist solely of zeros. The chosen indices mark the beginning and end of a subsequence in the vector. That subsequence is then reversed and written back in place, generating a new neighboring solution.

1.3 Tabu Search Algorithm

The initial individual is generated completely at random—the solution vector of visits and depot returns is created by random order. This randomly generated solution (the current individual), evaluated by the objective function, serves as the starting point and the basis for generating the first neighborhood set.

Next, the stopping condition is checked. In our approach, the criterion is a predetermined number of iterations.

Then the best solution from the current neighborhood is identified. The new solution is added to the tabu list if it is not already present. If the new solution is better than the best found so far and not in the tabu list, it replaces the current individual. After this update, the stopping condition is evaluated again, closing the algorithm's cycle.

Algorithm 1 Tabu Search Algorithm

```
current ← randomIndividual
best ← current
add current to tabu
for i = 1 to iterations do
    neighborhood ← generate neighborhood(current, neighbourhood_size)
    candidate ← best individual from neighborhood not in tabu
    current ← candidate
    if current better than best then
        best ← current
        add current to tabu
    end if
end for
return best
```

1.4 Simulated Annealing Algorithm

Similarly to Tabu Search, the initial solution is chosen at random and evaluated by the objective function.

In the main loop, the stopping condition is checked. If it is not met, the neighborhood of the current best solution is generated. The best candidate is selected from the neighborhood.

- If the candidate is better than the current solution, it is accepted unconditionally.
- If it is worse, acceptance depends on the difference in objective values and the current temperature: the higher the temperature, the greater the probability of accepting a worse solution.

After potentially updating the solution, the temperature is decreased according to the chosen cooling schedule. The probability of accepting a worse solution is given by

$$P = \exp\left(\frac{f(V_n) - f(V_c)}{T}\right),$$

where:

* $f(V_n)$ — is the objective value of the new candidate, * $f(V_c)$ — is the objective value of the current solution, * T — is the current temperature.

The algorithm's cycle ends after this operation.

Algorithm 2 Simulated Annealing Algorithm

$current \leftarrow randomIndividual$
$best \leftarrow current$
$T \leftarrow T_{initial}$
for $i = 1$ to $iterations$ **do**
    $neighborhood \leftarrow$ generate neighborhood($current$, $neighbourhood\_size$)
    for all ind in $neighborhood$ **do**
        if ind better than $current$ **or** $P_{acceptance}(ind, current, T) > random(0, 1)$ **then**
            $current \leftarrow ind$
        end if
    end for
    if $current$ better than $best$ **then**
        $best \leftarrow current$
    end if
    $T \leftarrow \alpha \times T$ ▷ example cooling schedule with coefficient $\alpha = 0.999$
end for
return $best$

1.5 Hybrid Annealing–Tabu (Hardening)

The hybrid combines simulated annealing's mechanisms with a tabu memory: we allow multiple "reheating" phases while preventing returns to recently visited solutions via a tabu list. Increasing temperature helps escape local optima, and the tabu list supports exploration of new areas. The hybrid aims to maximize post-optimum exploration.

After reaching the minimum temperature, a countdown of additional iterations (e.g., 100) begins. Once that count elapses, the system is "reheated" again—temperature is set to a defined level (which may differ from the initial). The cooling–heating cycle, supported by tabu memory, repeats until the total iteration limit is reached.

Algorithm 3 Hybrid Tabu–Annealing Algorithm

$current \leftarrow randomIndividual$
$best \leftarrow current$
add *current* to tabu list
$T \leftarrow T_{initial}$
$isCooling \leftarrow true$
$iterToHeat \leftarrow iterations\_to\_start\_heating$
for $i = 1$ to *iterations* **do**
    $neighborhood \leftarrow$ generate neighborhood($current$, $neighbourhood\_size$)
    for all *ind* in *neighborhood* **do**
        if *ind* not in tabu **then**
            if *ind* better than *current* **or** $P_{acceptance}(ind, current, T) > random(0, 1)$ **then**
                $current \leftarrow ind$
            end if
        end if
    end for
    if *current* better than *best* **then**
        $best \leftarrow current$
        add *current* to tabu
    end if
    if $isCooling$ **then**
        if $iterToHeat = 0$ **then**
            $iterToHeat \leftarrow iteration\_to\_start\_heating$
        end if
        $T \leftarrow \alpha \times T$ ▷ example cooling schedule with coefficient $\alpha = 0.995$
        if $T = T_{final}$ **then**
            $isCooling \leftarrow$ false
        end if
    else
        if $iterToHeat > 0$ **then**
            $iterToHeat \leftarrow iterToHeat - 1$
        else
            $T \leftarrow \beta \times T$ ▷ example heating schedule with coefficient $\beta = 1.15$
            if $T = T_{max}$ **then**
                $isCooling \leftarrow$ true
            end if
        end if
    end if
end for
return *best*

2 Test Data

Experiments employed standard benchmark data from the VRP repository available at:

`http://vrp.atd-lab.inf.puc-rio.br/index.php/en/`

We used instances from **Augerat's Set A**, commonly applied in literature for evaluating CVRP heuristics and metaheuristics.

2.1 Data File Structure

Each instance file contains:

- **Header**:
  - `NAME` – instance name,
  - `COMMENT` – comments (e.g., number of vehicles, optimal value),
  - `TYPE` – problem type (`CVRP`),
  - `DIMENSION` – number of nodes (depot + customers),
  - `EDGE_WEIGHT_TYPE` – distance metric (e.g., `EUC_2D`),
  - `CAPACITY` – vehicle capacity.
- **NODE_COORD_SECTION** – coordinates (X, Y) of each node.
- **DEMAND_SECTION** – demand of each node.
- **DEPOT_SECTION** – depot node number (usually 1), ended by `-1`.
- **EOF** – end-of-file marker.

Example:

```
NAME : A-n32-k5
TYPE : CVRP
DIMENSION : 32
CAPACITY : 100
NODE_COORD_SECTION
1 82 76
2 96 44
...
DEMAND_SECTION
1 0
2 19
...
DEPOT_SECTION
1
-1
EOF
```

2.2 Used Instances

We tested the following CVRP instances from Set A:

- `A-n32-k5` – 32 nodes, 5 vehicles
- `A-n37-k6` – 37 nodes, 6 vehicles
- `A-n39-k5` – 39 nodes, 5 vehicles
- `A-n45-k6` – 45 nodes, 6 vehicles
- `A-n48-k7` – 48 nodes, 7 vehicles
- `A-n54-k7` – 54 nodes, 7 vehicles
- `A-n60-k9` – 60 nodes, 9 vehicles
- `A-n61-k9` – 61 nodes, 9 vehicles

Each instance varies in size and complexity, enabling comprehensive comparison of algorithm quality and scalability.

3 Parameter Tuning

Parameter tuning involved analyzing algorithm behavior under various settings. Focus was on results after 50,000 iterations with a population of 40 individuals per iteration. This provided reliable comparison and optimization of parameters for each metaheuristic.

4 Comparison of Algorithms

Each algorithm was run 100 times to minimize randomness effects. Below are initial parameters and summary results.

4.1 Initial Parameters for Tabu Search

Parameter	Value
Number of iterations	15 000
Neighborhood operator	reversal
Tabu list size	200
Neighborhood size	40

4.2 Initial Parameters for Simulated Annealing

Parameter	Value
Number of iterations	15 000
Neighborhood operator	reversal
Neighborhood size	40
Initial temperature	700.0
Final temperature	0.0
Cooling schedule	geometric
Cooling rate	0.9987

4.3 Initial Parameters for Hybrid Tabu–Annealing

Parameter	Value
Number of iterations	15 000
Iterations to reheat	50
Neighborhood operator	reversal
Tabu list size	250
Neighborhood size	40
Initial temperature	700.0
Final temperature	0.001
Maximum reheating temperature	20.0
Cooling schedule	geometric
Cooling rate	0.995
Heating schedule	geometric
Heating rate	1.5

Instance	Opt. value	Tabu (100 runs)				SA (100 runs)				Tabu+SA (100 runs)			
		Best	Worst	Avg	Std	Best	Worst	Avg	Std	Best	Worst	Avg	Std
N32-k5	**784**	784	827	**786.94**	4.54	784	837	802.78	20.85	784	831	793.78	16.34
N37-k6	**949**	949	978	**956.75**	8.26	949	1035	978.47	18.11	949	1004	963.98	13.13
N39-k5	**822**	825	849	837.08	4.84	822	905	839.71	15.81	822	857	**830.27**	5.93
N45-k6	**944**	965	1018	987.26	11.89	944	1019	977.51	13.79	947	994	**965.9**	9.41
N48-k7	**1073**	1106	1185	1160.44	13.5	1091	1178	1128.24	16.21	1074	1150	**1114.03**	13.65
N54-k7	**1167**	1216	1288	1255.59	15.91	1177	1307	1227.45	26.43	1172	1244	**1201.7**	16.26
N60-k9	**1354**	1394	1503	1449.34	22.73	1376	1572	1440.74	33.38	1364	1464	**1414.46**	22.68
N61-k9	**1034**	1069	1133	1099.9	14.2	1040	1136	1079.94	21.25	1036	1102	**1067.41**	14.35

Table 1: Algorithm results for various instances

4.4 Algorithm Behaviors

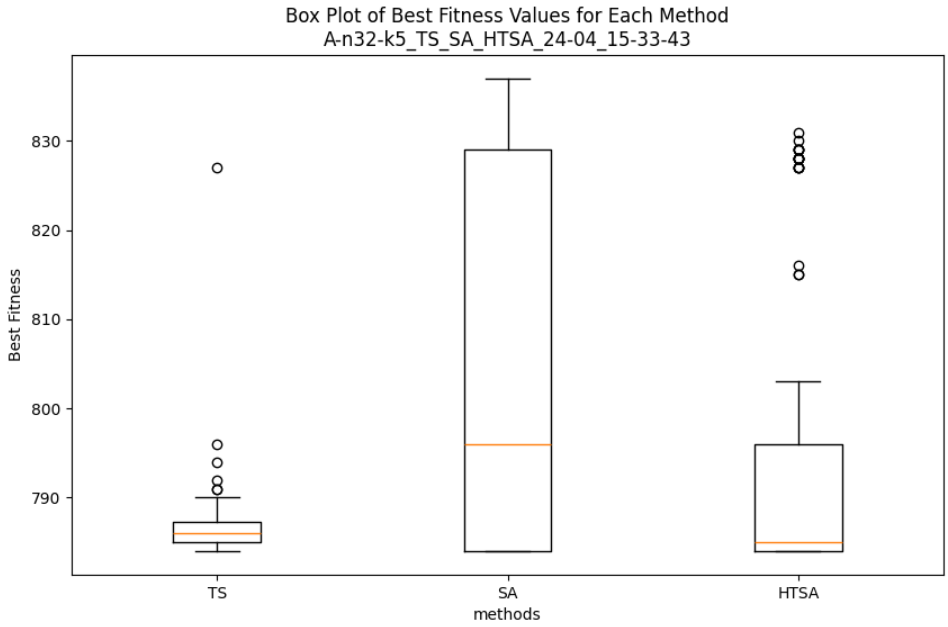

Figure 1: Box plot for A-n32-k5 data

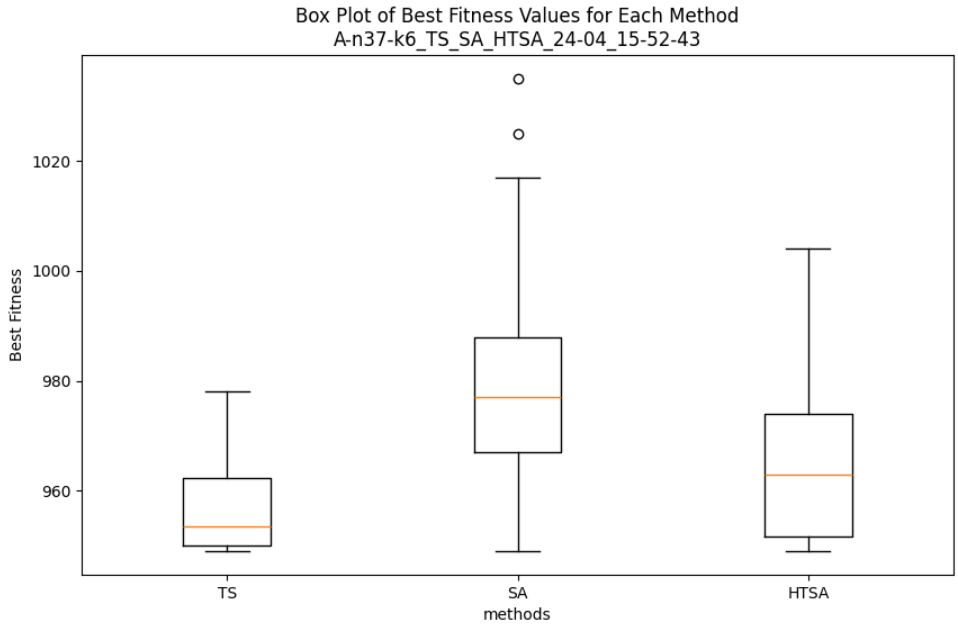

Figure 2: Box plot for A-n37-k6 data

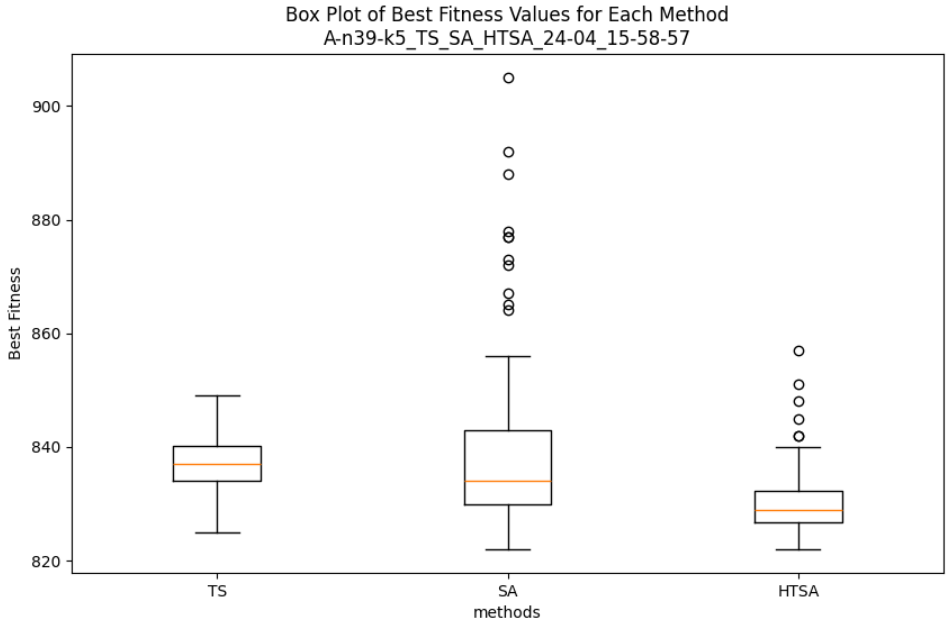

Figure 3: Box plot for A-n39-k5 data

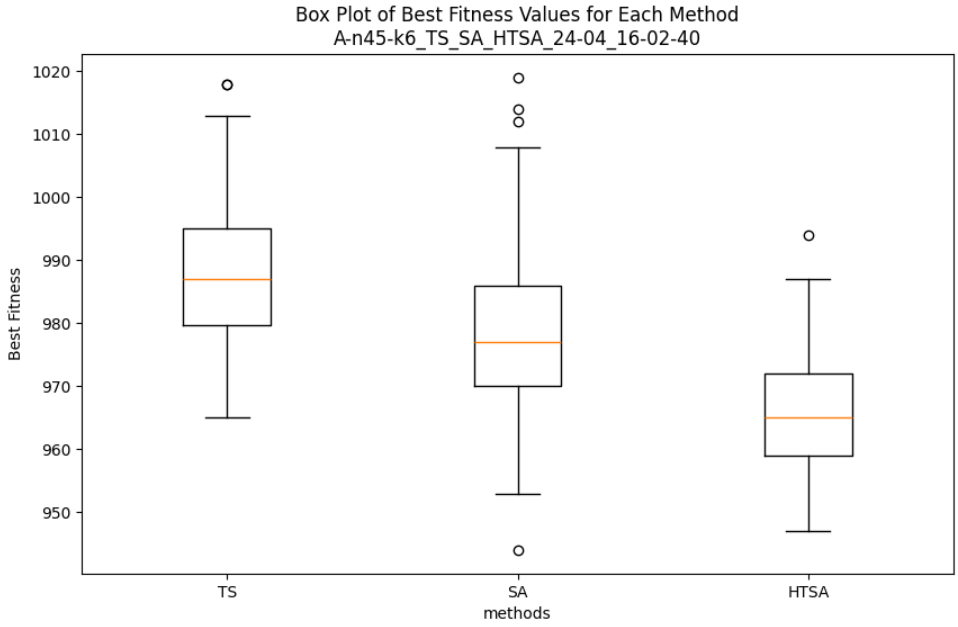

Figure 4: Box plot for A-n45-k6 data

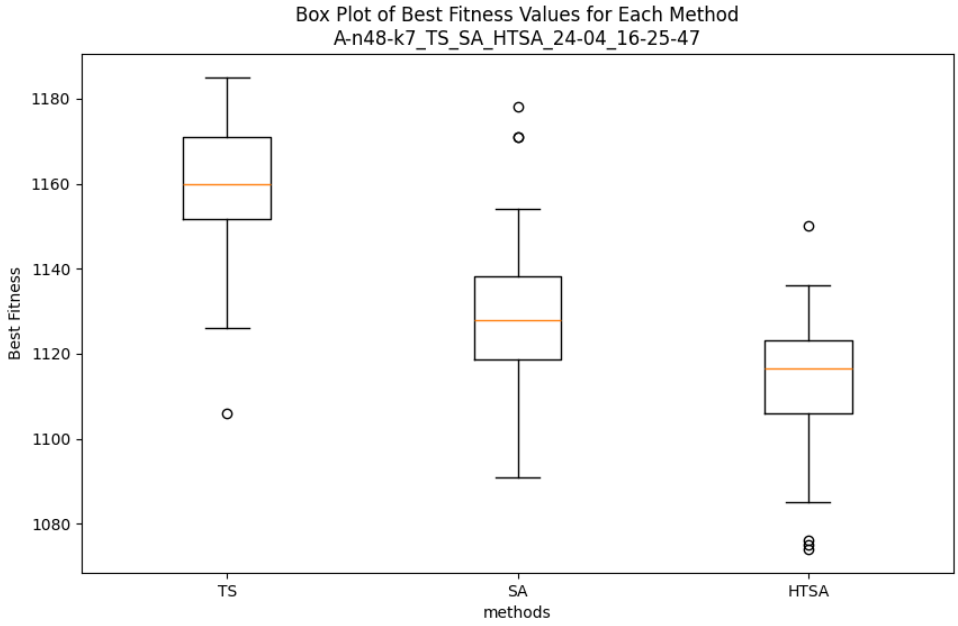

Figure 5: Box plot for A-n48-k7 data

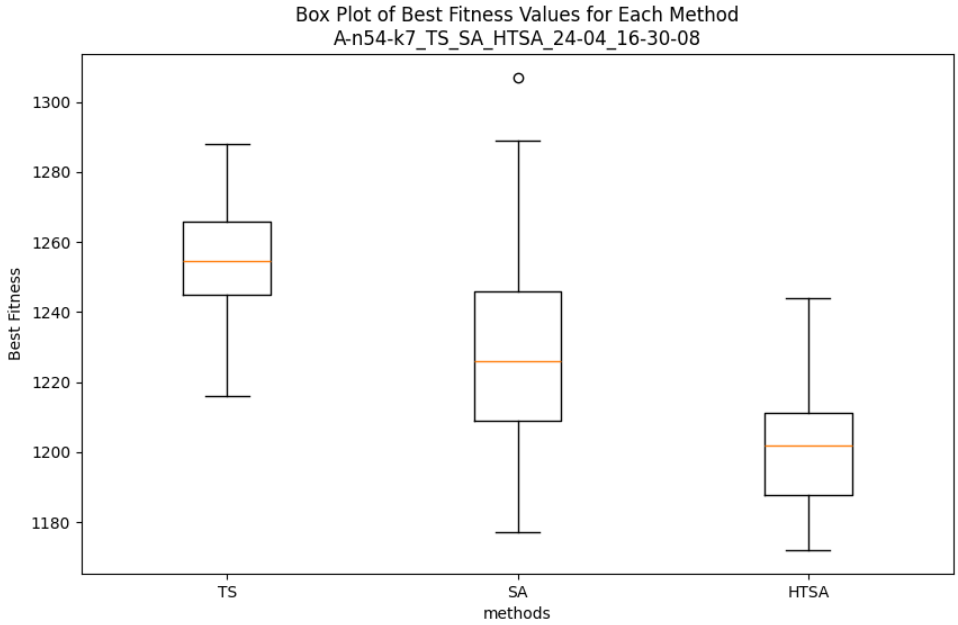

Figure 6: Box plot for A-n54-k7 data

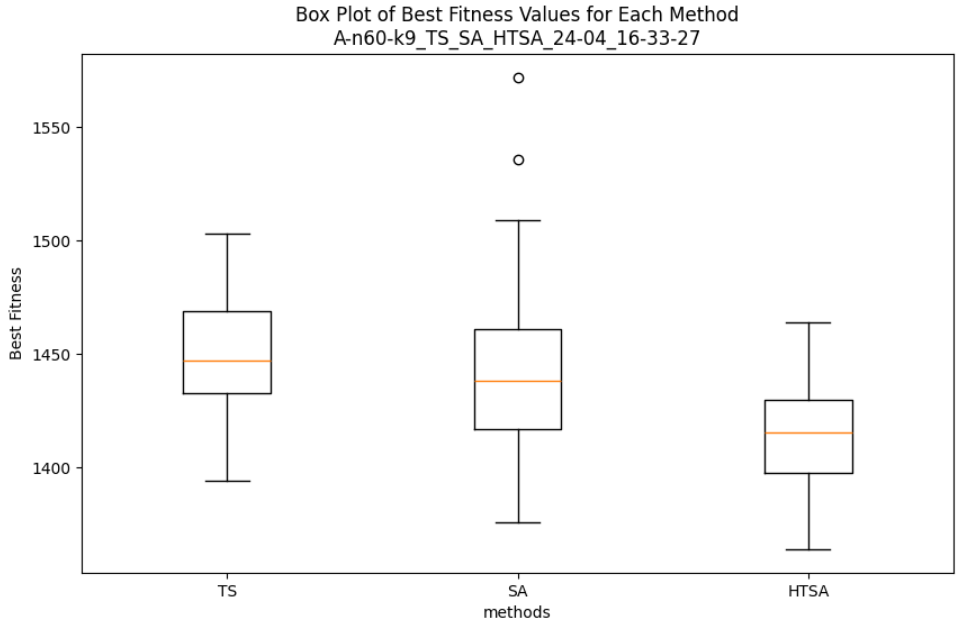

Figure 7: Box plot for A-n60-k9 data

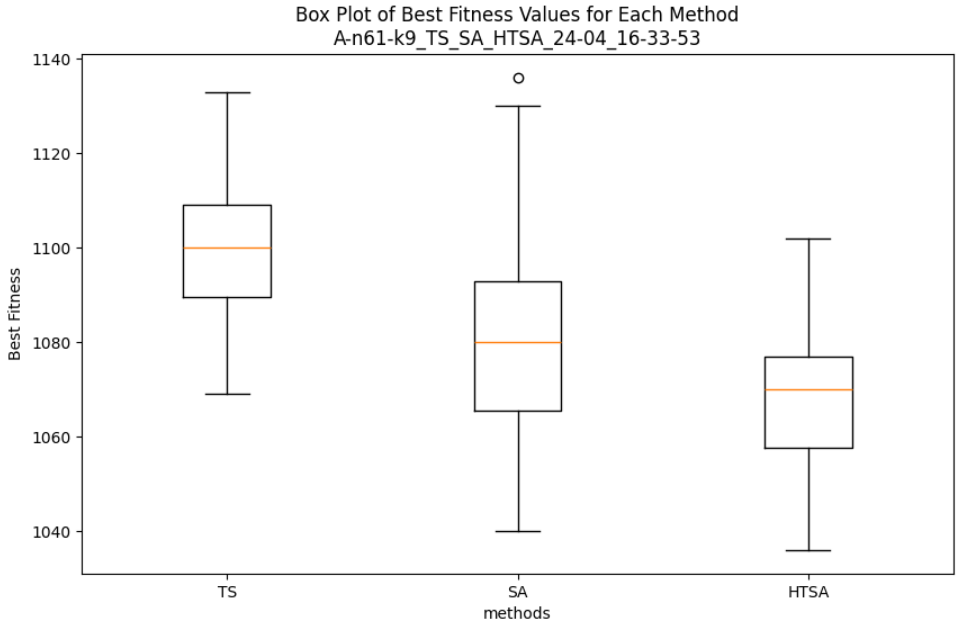

Figure 8: Box plot for A-n61-k9 data

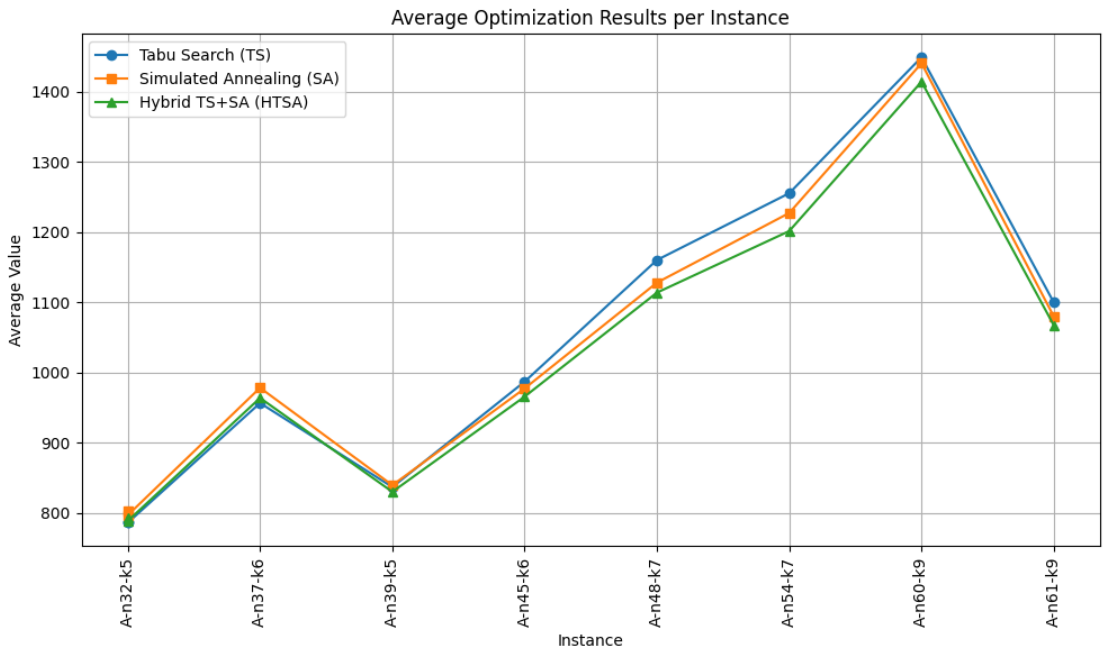

Figure 9: Trend chart of averaged method results for different problem instances

4.5 Tabu Search

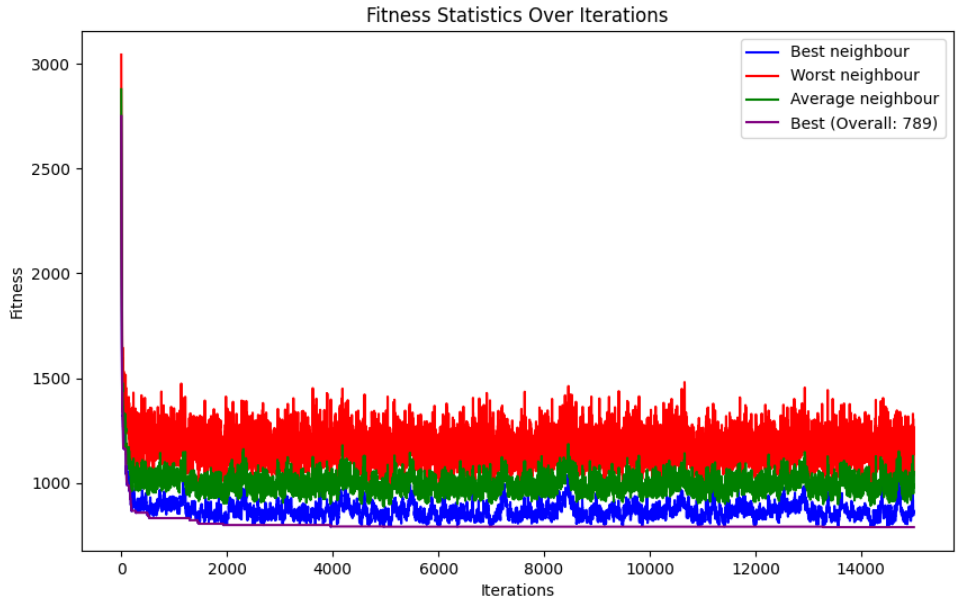

Figure 10: Visualization of Tabu Search behavior on the small instance

Figure 11: Visualization of Tabu Search behavior on the medium instance

Figure 12: Visualization of Tabu Search behavior on the large instance

On the displayed charts, one can observe characteristic "jumps" in the search trajectory of Tabu Search. This indicates transitions between successive local optima: the current solution's value does not stray far from the best solution found so far, yet it does not coincide with it entirely. This demonstrates that the algorithm effectively explores the solution space, avoiding both premature convergence and unnecessary drifting into

worse solutions. Such behavior confirms the proper choice of tabu list size and solution-selection strategy.

4.6 Simulated Annealing

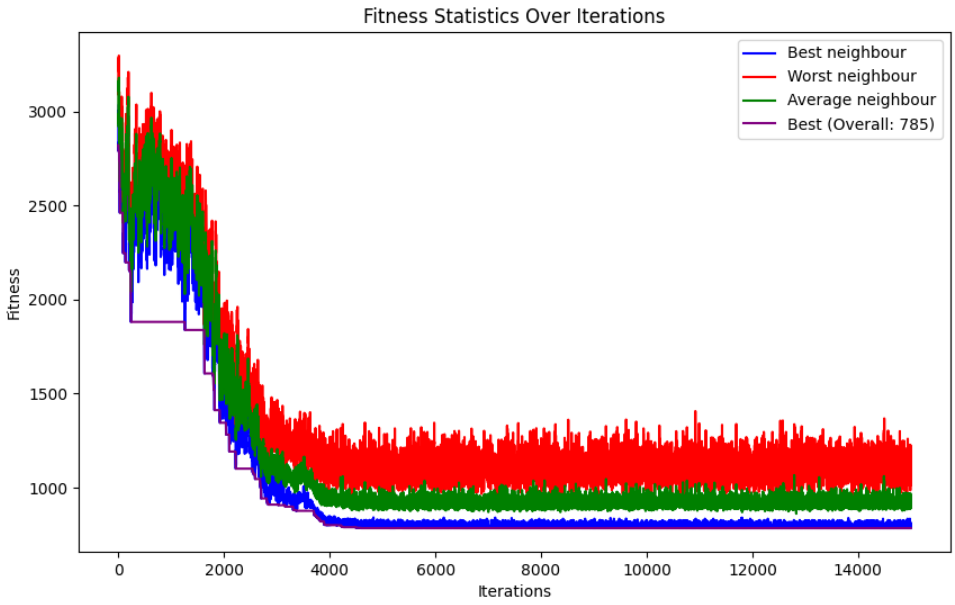

Figure 13: Visualization of Simulated Annealing behavior on the small instance

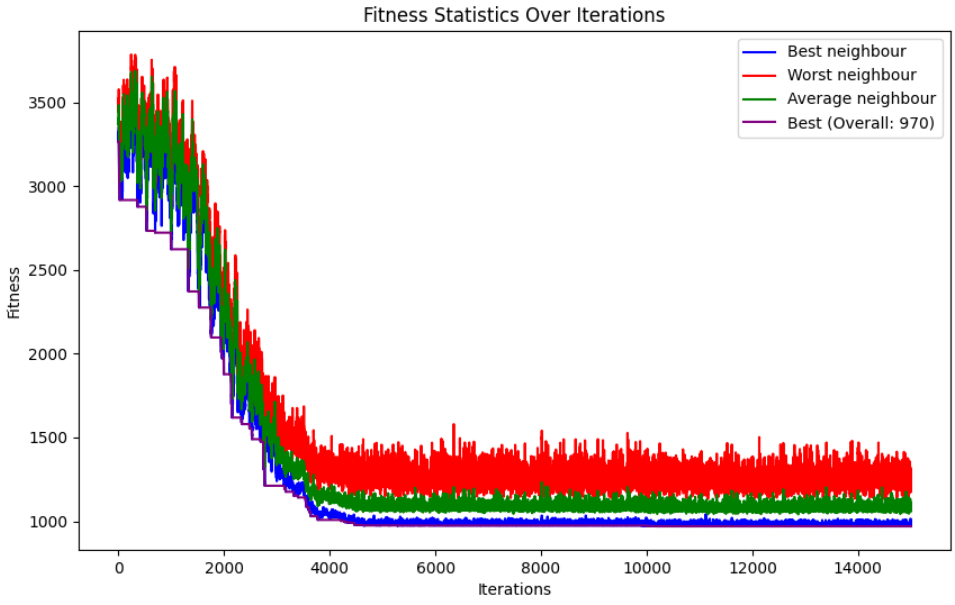

Figure 14: Visualization of Simulated Annealing behavior on the medium instance

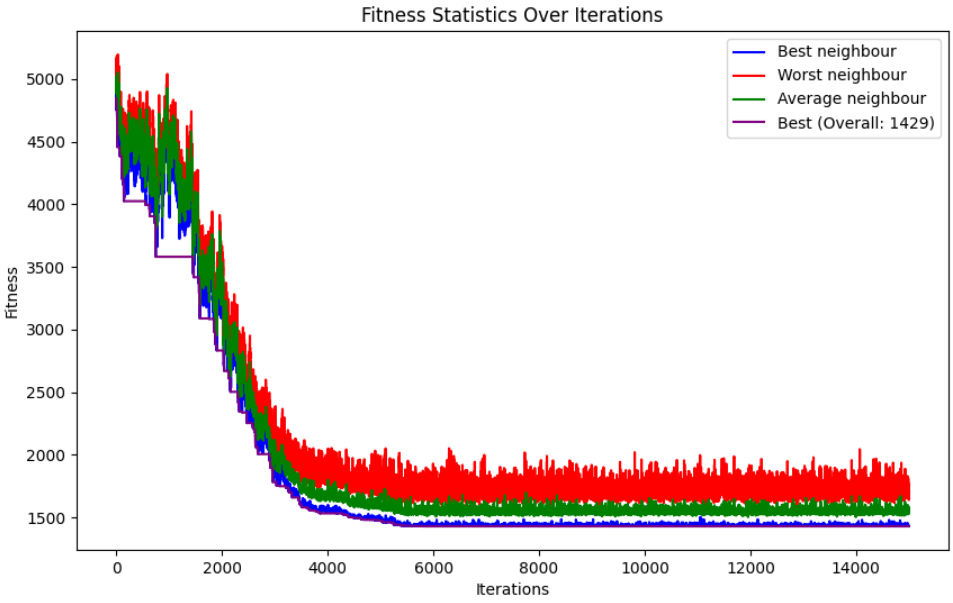

Figure 15: Visualization of Simulated Annealing behavior on the large instance

The plotted charts show the typical course of Simulated Annealing. In the initial phase, at high temperature, the search behaves stochastically—the algorithm frequently accepts worse solutions, preventing premature trapping in local minima.

As the temperature decreases, the search becomes progressively more focused. Worse solutions are accepted less often, and the quality of generated solutions gradually improves. In the final phase, local search predominates: the trajectory of the current solution almost coincides with that of the best neighbor, indicating stabilization and fine-tuning of the final solution.

4.7 Hybrid Tabu–Annealing

Figure 16: Visualization of Hybrid Tabu+SA with reheating behavior on the small instance

Figure 17: Visualization of Hybrid Tabu+SA with reheating behavior on the medium instance

Figure 18: Visualization of Hybrid Tabu+SA with reheating behavior on the large instance

The hybrid algorithm combining Simulated Annealing with Tabu Search exhibits a pattern similar to classical annealing but introduces iterative "reheating," giving it a cyclic character.

In the initial phase of each cycle, rapid cooling promotes intensive exploration. After a set number of iterations (user-defined), the system is "reheated"—the temperature rises, though typically to a lower level than the starting temperature. The plot clearly shows that the peak temperatures of successive cycles gradually decrease. Upon reaching each threshold, the cooling cycle repeats, and the process continues until the total iteration limit is reached.

In this hybrid, the tabu list functions as in classic Tabu Search—storing recently visited solutions to prevent their repetition. This forces the algorithm to explore new regions of the solution space.

Such an approach balances global exploration (via cyclic reheating) with local refinement (during cooling). As a result, the algorithm can effectively jump between different solution-space regions while thoroughly examining local neighborhoods.

5 Program Profiling

To analyze algorithm performance, we used profiling to precisely measure execution times of code segments and identify bottlenecks.

5.1 What is a Profiler?

A profiler is a tool for monitoring program execution in real time or post-completion. It provides information on:

- function execution times,
- number of calls to code segments,
- memory usage,
- call sequences.

This allows optimization by:

- eliminating inefficient operations,
- reducing computational complexity of critical sections,
- better resource management.

5.2 Profiling in This Project

We used the profiler integrated in CLion (via WSL) to analyze functions responsible for:

- generating initial solutions,
- computing the objective function (route length),
- mutation operators,
- selecting best individuals each iteration.

Profiling on instances like `A-n32-k5` and `A-n61-k9` identified the most time-consuming code. Optimizations included:

- simplifying data I/O routines,
- optimizing mutation index selection,
- improving tabu list management using `unordered_set` and `deque` for $O(1)$ oldest-entry removal.

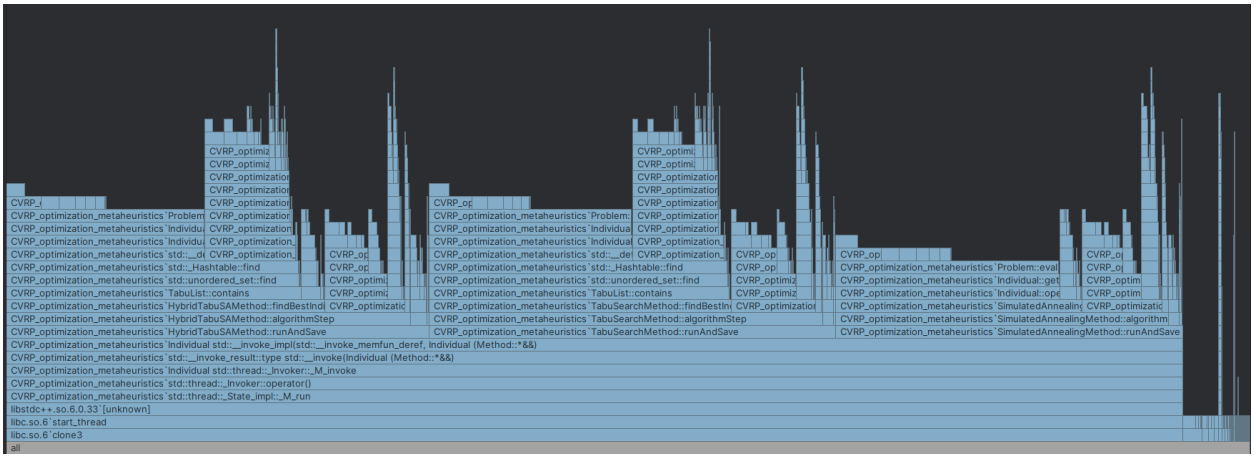

Figure 19: Profiler output view in CLion (WSL)

6 Conclusions

We compared three approaches for CVRP: Simulated Annealing, Tabu Search, and a hybrid combining both with reheating and a tabu list.

For the two smallest instances (A-n32-k5, A-n37-k6), Tabu Search yielded the best and most stable results. For larger, more complex instances, the hybrid method excelled by combining global exploration (annealing) with tabu memory, better escaping local optima and achieving superior objective values.

Overall, the hybrid approach proved most effective for the hardest instances, confirming that combining optimization strategies may significantly improve solution quality. All methods were carefully tuned; only classic Simulated Annealing showed higher variability on the smallest instance.