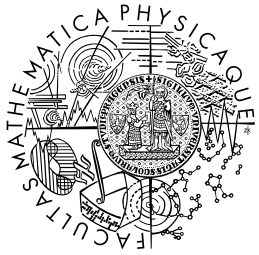

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Kareem Hamed Osman Abdelaal Elsewirky

Nature-Inspired Algorithmic Approach to Capacitated Vehicle-Routing Problem with Preferred Times

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis: RNDr. Jiří Fink, Ph.D.

Study programme: Computer Science

Prague 2025

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague date: 5th May, 2025 Kareem Elsewirky

Author's signature

To my parents for their support, to my friends for their company, and to my supervisor for his guidance.

Title: Nature-Inspired Algorithmic Approach to Capacitated Vehicle-Routing Problem with Preferred Times

Author: Kareem Hamed Osman Abdelaal Elsewirky

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Jiří Fink, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This thesis presents a nature-inspired algorithmic approach to the capacitated vehicle routing problem with preferred time windows. Using nature-inspired algorithms, the thesis optimizes route planning based on real geographical data, delivery constraints, and preferred delivery times. The results are then compared and analyzed to determine the feasibility and quality of the solutions produced.

Keywords: Vehicle Routing Problem, Genetic Algorithms, Ant-Colony Optimization, Hill-Climbing

Contents

1 Introduction

1.1 Background

The Vehicle Routing Problem (VRP) [1] is a fundamental challenge in logistics and transportation, focusing on optimizing routes for a fleet of vehicles to deliver goods or services to a set of locations, often referred to as customers. For example, a delivery company with multiple trucks must determine efficient routes to deliver packages across a city, starting and ending at a central depot. The objective is to minimize the overall distance traveled or the total cost of the routes while ensuring that each customer is visited exactly once.

Theoretical research of the VRP started with the truck dispatching problem brought forward by Dantzig and Ramser in 1959 [2], which aimed to find the optimum routing of a fleet of gasoline delivery trucks between a bulk terminal and a large number of service stations supplied by the terminal. There are many other variants of the vehicle-routing problem. One of the variants is the variable capacity vehicle-routing problem, which, as the name suggests, is the classical VRP but with vehicles of variable capacity. Ever since then, the problem has been increasingly studied in depth, with different variants such as the capacitated vehicle-routing problem (CVRP), where the delivery vehicles have a limited (constant) capacity, and the VRP with time windows (VRPTW) where each delivery must be made at a specified time or time range called a time-window (CRPTW). There has also been increasing focus on what are called "rich" VRPs [3] that differ from classical VRPs by incorporating more realistic constraints such as multiple depots, variable vehicle capacities, multiple trips by the same delivery vehicle, and loading capacities. We combine both constraints and analyze and compare the performance of three different nature-inspired metaheuristic algorithms.

As an NP-hard problem [4], VRP has been extensively studied in operations research and logistics. Optimization algorithms, particularly heuristics and nature-inspired methods, have been widely used to tackle the problem (see survey [5]). Although classical VRP assumes straightforward objectives, such as minimizing distance or cost, real-world scenarios often introduce additional constraints, including vehicle capacities, delivery time windows, and revenue variability based on delivery timing. Other methods for finding solutions to variants of the VRP have also varied from the direct optimization approach via linear programming [6], which can be useful for smaller problem instances, for example, to metaheuristic algorithms [5] and reinforcement learning [7].

This study explores a variant of the VRP that incorporates time-window and capacity constraints (VRP-TWC). Geographical data from OpenStreetMaps is used to simulate realistic distances, and package demand is modeled based on population densities. The problem is defined in JSON format to allow flexible testing of configurations, including vehicle capacities, package distributions, and time-window partitions.

The scope of the VRP-TWC in this study is limited to single-source deliveries from a central depot using homogeneous vehicles (constant capacity). Deliveries must occur immediately upon reaching their destinations, with no waiting time allowed at delivery points.

To solve the VRP-TWC, this thesis investigates three nature-inspired algorithms:

- Genetic Algorithm [8]
- Ant-Colony Optimization [9]
- Hill-Climbing [10]

The performance of these algorithms is analyzed and compared to assess their effectiveness in solving the VRP-TWC under realistic constraints.

1.2 Motivation

In a competitive global environment, businesses increasingly rely on efficient routing solutions that optimize operational costs while maximizing revenue. For e-commerce and courier companies, timely deliveries within predefined time windows often determine customer satisfaction and profitability. This study [11] highlights the challenges local delivery platforms face in balancing cost-efficiency with reliable delivery services. It emphasizes that meeting delivery time windows is crucial for maintaining high service guarantees and customer satisfaction. Revenues may vary according to delivery time, adding complexity to the optimization problem. While the classical VRP is an interesting combinatorial and optimization problem, it is also not very reflective of real-world scenarios, as there are minimal constraints that mainly correspond to the graphical nature of the problem.

This thesis addresses the VRP-TWC, where vehicles must comply with capacity restrictions and deliver packages within time windows that influence profitability. By incorporating variable revenues and real-world geographical data, the problem mirrors practical challenges faced by modern logistics systems, and this classifies this as at least partly a “rich” variant of the VRP as described previously.

The goal is to provide information on the application of nature-inspired algorithms to solve the VRP-TWC and compare their overall performance on a variety of problems derived from real-world data. It seeks to identify efficient strategies for optimizing delivery routes, balancing cost minimization and revenue maximization while adhering to capacity and time-window constraints.

1.3 Contribution

The contributions of this thesis are as follows.

- We create a problem and solution model to describe a so-called "rich" VRP variant, the VRP-TWC, which is described further in Chapter 2.
- Adapting the three nature-inspired algorithms, namely, genetic algorithm, ant-colony optimization, and hill-climbing, to solve the VRP-TWC variant of the vehicle-routing problem. The algorithms are described in more detail in Sections 3.2, 3.3, and 3.4, respectively.
- Generate problem instances from real-life data from OpenStreetMap in the form of GeoJSON described in more detail in Section 4.1.

- Run and compare the performance of algorithms in three different experimental setups. We compare algorithm performance in general when it comes to the fitness of the solutions generated and the runtimes of the algorithms in Section 4.3. The relation between the number of packages in a problem and its impact on runtime for each algorithm is then investigated and compared in Section 4.4. We then investigate the impact of the relationship between the package weights and the vehicle capacity and its effect on profitability, which is further described in Section 4.5.

- We create a program in the Julia programming language to solve any instance of VRP-TWC, provided that it is in the valid JSON form described by the JSON schema provided in the code repository[1], using any of the 3 nature-inspired algorithms. The user documentation for the program is located in the repository. Those interested can also delve deeper into the source code and utilize the developer documentation as a reference provided in the same repository under VRPTWC/docs/build/api/index.html.

[1] `https://gitlab.mff.cuni.cz/teaching/nprg045/fink/kareem_elsewirky`

2 Problem Statement

In the context of VRP-TWC, the objective is to determine the optimal delivery paths for a fleet of vehicles to maximize revenue and minimize costs. Key features of the problem include

- Capacity Constraints: Each vehicle has a limited capacity, restricting the total weight of the packages it can carry.
- Time Horizon: The problem operates within a defined time horizon (e.g., 1 to 100), within which all deliveries must be completed.
- Time Windows and Revenue: Each package has an associated time interval within the overall time horizon, divided into specific partitions. Each partition has an associated revenue, indicating the potential earnings from delivering the package within that specific time window. The goal is to deliver packages in a way that maximizes total revenue.
- Single-Source Delivery: All packages originate from a single depot or factory. Each package has an associated weight, which must be accounted for when loading vehicles.
- Vehicle Routing: Each vehicle must start and end its route at the depot. Vehicles cannot visit the same node twice, ensuring an efficient path. There is no waiting time allowed before delivering a package; each delivery must occur as soon as the vehicle reaches the destination.

2.1 The Model

2.1.1 Mathematical Formulation

1. **Packages and Depot**

   The problem involves a set of packages P, each represented by a unique vertex corresponding to the delivery location. All deliveries originate from and return to a single depot (or factory), represented by the vertex with ID 0.

2. **Vehicle Fleet**

   The fleet consists of homogeneous vehicles, each with a carrying capacity Q. Vehicles must start and end their routes at the depot.

3. **Distances and Travel Costs**

   The distances between delivery points and the depot are given by a 2-dimensional distance matrix that specifies the travel cost between any two packages or between a package's delivery location and the depot. The travel cost between any two vertices is calculated as the product of the distance between those vertices and a specified multiplicative constant C.

4. **Time Range**

   A time range (T_{start}, T_{end}) defines the period within which all packages need to be delivered. T_{start} is fixed at 0, while T_{end} is defined as the end time of the last partition for the furthest package from the depot.

5. **Time Partitions and Revenue**

   Each package $p \in P$ has a partitioned time range, dividing the overall time horizon into discrete segments. Each partition has an associated revenue, indicating the potential earnings from delivering the package within that specific time window.

The main constraints are as follows:

- Each vehicle has a fixed capacity, restricting the number of packages it can carry.
- Deliveries must be completed within the predefined time windows.
- Routes must start and end at the central depot.
- Every package is delivered by exactly one vehicle.

2.1.2 The Solution

The solution will be represented as a list of vehicles. Each list of vehicles contains a list of packages that have been delivered by that vehicle. The list of packages is ordered, meaning that the vehicle's path will be identical to the list of package IDs. The revenue gained is then accumulated over those packages based on which time-partition the delivery time of the package falls in. The time-partition can be retroactively retrieved because vehicles can't wait, which means that we don't have to guess when the vehicle arrived or left a delivery location. There is also the added caveat that deliveries happen instantaneously upon arrival at a delivery location for the sake of simplicity, since this would not add any layers of combinatorial complexity to the problem.

2.1.3 Constraints for feasibility of a solution

- The route must start at vertex 0 and must end at vertex 0.
- Every package is delivered by exactly one vehicle exactly once.

Further obvious constraints can be added such as the weight of the packages must not exceed the capacity of the delivery vehicle Q. So that for delivery vehicle t, loaded with packages $b_1, \ldots, b_m$ and corresponding weights $h_1, \ldots, h_m$ we have

$$\sum_{i=1}^{m} h_i \leq Q$$

2.1.4 Objective Function

Assume that delivery vehicle t delivers packages $b_1, b_2, \ldots, b_{m_t}$ at time windows $w_1, w_2, \ldots, w_{m_t}$. Then the total revenue gained is given by

$$r = \sum_{t \in T} \sum_{i=1}^{m_t} p_{b_{i_t}, w_{i_t}}$$

where $p_{b_{i_t}, w_{i_t}}$ is the revenue gained by delivering package $b_{i,t}$ in time window w_{i_t}.

The cost of delivering the packages for a given truck t taking a route $V = (v_1, \ldots, v_m)$ will be given by

$$c = \sum_{t \in T} \sum_{\substack{i=1 \\ v_i, v_{i+1} \in V}}^{m-1} C \cdot d_{v_i, v_{i+1}}$$

where $d_{v_i, v_{i+1}}$ is the distance of traveling from v_i to v_{i+1}, and C is the distance multiplier. The goal of the program is to maximize the profit g given by

$$g = r - c \tag{2.1}$$

where we are assuming that the route taken by each delivery vehicle is a list containing each delivered package. Then we sum over all delivery vehicles the sum of the cost of traveling between each 2 vertices and finally the cost of traveling back to the depot, which we then multiply by the multiplicative constant C provided in the input to get the actual cost.

3 Algorithms

This chapter outlines the methodologies used to solve VRP-TWC. Three nature-inspired algorithms, Genetic Algorithm, Ant-Colony Optimization, and Hill-Climbing, were selected based on their successful application to similar combinatorial optimization problems. Each algorithm is presented with an overview of its core principles, followed by specific modifications to address time window constraints, vehicle capacities, and variable revenue considerations.

3.1 Overview of Algorithmic Approaches

The complex nature of the VRP-TWC requires algorithms that can explore a large solution space and adapt effectively to the changing demands of the problem. Nature-inspired algorithms are known for their adaptability and effectiveness in finding near-optimal solutions to NP-hard problems such as VRP. The following sections detail each selected algorithm.

3.1.1 Common Functions

The algorithms tested share some common functions. These functions specifically are the fitness function, **fitness(**s**,** P**)**, and the repairing function, **RepairSolution(**s**,** P**)**, both of which take as parameters a solution s and a problem instance P. The fitness function is based on (2.1) and is written to be logically identical for the 3 algorithms. The function will compute the costs incurred by each vehicle and the revenues earned by delivering the packages and return the profit, which is the difference between revenue and cost. The repair function is intended to limit the search space to strictly viable solutions and to ensure that any solution modified or generated by an algorithm remains valid with respect to the problem constraints. First, it cleans up unnecessary vehicles and depot references by removing empty vehicles and deleting depot placeholders from package lists. Subsequently, it rechecks the feasibility of the route of each vehicle, adjusting or reassigning packages to prevent late deliveries. If some packages cannot be delivered on time within existing routes, the function may create new vehicles to accommodate them without exceeding capacity limits. Finally, it reintroduces the depot nodes at the start and end of each route to restore a proper routing structure. This process maintains the integrity of candidate solutions and contributes to a more robust and effective search for high-quality solutions.

Moreover, two of the algorithms, namely the genetic algorithm and the hill-climbing algorithm, share a function to generate the random initial population. The function, **GenerateRandomPopulation(**P**,** N**)** takes as parameters the problem instance P and the number N of solutions in the initial population. The initial population is constructed by randomly permuting all available packages and then distributing them into vehicles. Starting from the shuffled list, the algorithm sequentially adds packages to a vehicle until the addition of another package exceeds the capacity of that vehicle. Then it starts assigning packages to a new vehicle and continues this process until all packages have been placed. By handling packages in random order and stopping before exceeding capacity,

this approach creates a diverse range of starting solutions. After the routes are formed, a repair function is applied to ensure feasibility, for instance, by verifying the correct location of the depot and compliance with capacity restrictions. The result is a valid, though not necessarily optimal, initial solution from which Hill Climbing begins its iterative improvement process.

3.2 Genetic Algorithm (GA)

The Genetic Algorithm (GA) is an evolutionary algorithm inspired by the process of natural selection. It is particularly effective in solving complex optimization problems through the iterative refinement of a population of potential solutions.

3.2.1 Principles of Genetic Algorithm

GAs operate on a population of potential solutions, or chromosomes, by applying genetic operations such as selection, crossover, and mutation to produce new offspring. This process enables exploration and exploitation of the solution space. The algorithm proceeds as follows:

- Initialization: A population of random chromosomes is generated, each representing a possible solution to the VRP. Each chromosome encodes a set of delivery routes.
- Selection: Chromosomes are selected for reproduction based on their fitness, which is evaluated according to the total cost of the route, the adhesion to time window constraints, and the revenue generated from timely deliveries.
- Crossover: Selected chromosomes undergo cross-over to combine the routes of the parent solutions, resulting in offspring with potentially better routes.
- Mutation: Small random changes are applied to some offspring, altering delivery sequences or reassigning packages to maintain diversity and avoid premature convergence.
- Replacement: The new offspring population replaces the old population and the process iterates until convergence or a stopping criterion is reached.

3.2.2 Pseudocode for Genetic Algorithm

The pseudocode for the general Genetic Algorithm (GA) is presented below:

Algorithm 1: Genetic Algorithm for the VRP-TWC

Require: Problem instance P, population size N, number of generations G, tournament size τ, crossover probability p_c, mutation probability p_m, elitism count E.

Ensure: Best solution s^* and its fitness $f(s^*)$.

Population $\leftarrow$ GenerateRandomPopulation(P, N) ▷ Feasible initial solutions respecting capacity.
for all $s \in$ Population **do**
    $s_{fitness} \leftarrow$ Fitness(s, P) ▷ Evaluate each solution.
end for
$s^* \leftarrow \arg\max_{s \in \text{Population}} f(s)$ ▷ Track the best solution found.
for $g = 1$ to G **do** ▷ Iterate over generations.
    Sort Population by fitness, descending
    while |Offspring| $< N$ **do**
        $p_1 \leftarrow$ TournamentSelection(Population, τ)
        $p_2 \leftarrow$ TournamentSelection(Population, τ)
        if rand() $< p_c$ **then** ▷ Crossover with capacity constraints.
            $c \leftarrow$ UniformCrossoverWithCapacity(p_1, p_2, P)
        else
            $c \leftarrow \arg\max_{x \in \{p_1, p_2\}} f(x)$ ▷ If no crossover, take the fitter parent.
        end if
        if rand() $< p_m$ **then** ▷ Mutate offspring by moving a package.
            MoveMutation(c, P)
        end if
        RepairSolution(c, P) ▷ Ensure feasibility (e.g., depot at start/end, capacity constraints).
        $f(c) \leftarrow$ Fitness(c, P)
        Offspring $\leftarrow$ Offspring $\cup \{c\}$
    end while
    Population $\leftarrow$ Offspring
    $s_{best}^{(g)} \leftarrow \arg\max_{s \in \text{Population}} f(s)$
    if $f(s_{best}^{(g)}) > f(s^*)$ **then**
        $s^* \leftarrow s_{best}^{(g)}$ ▷ Update the global best solution.
    end if
end for
return $s^*, f(s^*)$

3.2.3 Modifications for VRP-TWC

To adapt GA for VRP-TWC:

- Encoding: Each solution is represented by a structure that contains information about the path taken by each vehicle in the solution and the fitness of the solution.
- Initialization: The initial population is created randomly using the algorithm described in the common functions section by utilizing the

  GenerateRandomPopulation(P, N) function.

- Fitness function: The fitness of each solution is determined by equation (2.1).
- **UniformCrossoverWithCapacity(**p_1**,** p_2**,** P**)** takes as parameters two solutions, representing parents p_1 and p_2 as well as the problem instance P. This function utilizes a variant of uniform crossover. Uniform crossover is a genetic algorithm operator that combines two parent solutions by selecting each feature (or gene) independently of either parent with equal probability. The variant used here must also be bound by several constraints to ensure the validity of a solution. A package is only added to the child vehicle if it has not already been assigned to another vehicle in the child solution, packages are included only if their addition does not exceed the maximum weight capacity of the child vehicle; furthermore, a set of package IDs is maintained to ensure that each package is assigned exactly once in all vehicles in the child solution.
- **TournamentSelection(***Population***,** τ**)** is a popular form of selection that is being used in this instance, in which a subset of τ individuals (a "tournament") is randomly chosen from the population, and the best individual in this subset is selected for reproduction. The "best" individual is determined based on fitness, with a higher fitness value being preferred.
- **MoveMutation(c, P)** takes as parameters a solution c referring to a child solution created from existing solutions by crossover and the problem instance P. Through this process, a package is chosen at random and randomly switched to either a new vehicle (while satisfying capacity constraints) or moved to an entirely new vehicle. This adds diversity to the solution space and encourages exploration.
- **RepairSolution(s, P)** is the repair function described in the section of common functions that is used to ensure the feasibility of solutions.

3.3 Ant-Colony Optimization (ACO)

Ant-Colony Optimization (ACO) is inspired by the foraging behavior of ants, which deposit pheromones to mark favorable paths. Subsequent ants will then interpret the increased pheromones on favorable paths as an indication of high reward, which encourages even more ants to explore that path. ACO is known for its ability to effectively solve pathfinding and routing problems.

3.3.1 Principles of Ant-Colony Optimization

ACO simulates a colony of ants constructing solutions by exploring paths based on pheromone levels. There are 2 major steps in most ACO algorithms. The first step is solution generation, where ants typically start at a random vertex of a graph and decide which will be the next vertex to be visited. For each pair of vertices x, y, a transition probability is calculated which is proportional to

$$(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta}) \tag{3.1}$$

where τ_{xy} is the amount of pheromone between the 2 vertices, and η_{xy} is the attractiveness, typically a heuristic value, between the vertices, where α and β are constants determining the influence of both variables. The second step is pheromone update, in which there are two steps: pheromone evaporation and pheromone placement. The evaporation and placement of pheromones take place according to the relationship $\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_k \tau_{xy}^k$ where ρ is a constant that determines the evaporation rate. If k ants passed along the edge (x, y) and L_k is the quality of the solution found by ant k then $\tau_{xy}^k = \frac{S}{L_k}$ (or 0 if ant k did not cross the edge (x, y)) where S is a constant.

3.3.2 Pseudocode for Ant Colony Optimization Algorithm

The pseudocode for the general Ant Colony Optimization (ACO) algorithm is presented below:

Algorithm 2: Update Pheromones

Require: Pheromone matrix τ, set of solutions $\{R_k\}$ found by ants, evaporation rate ρ, deposit parameter q, problem P.
Ensure: Updated pheromone matrix τ.

$\tau \leftarrow (1-\rho) \times \tau$ ▷ Evaporate pheromone on all edges.
for all R_k in local solutions **do**
    for $i = 1$ to $|R_k| - 1$ **do**
        $u \leftarrow R_k[i], v \leftarrow R_k[i+1]$
        $\tau[u, v] \leftarrow \tau[u, v] + q$
        $\tau[v, u] \leftarrow \tau[v, u] + q$
    end for
end for
return τ

Algorithm 3: Ant Colony Optimization (ACO) for the VRP-TWC

Require: Problem instance P, number of ants m, parameters α, β, ρ, q, maximum iterations G, tolerance tol.
Ensure: Best solution s^* and its fitness $f(s^*)$.

$\tau \leftarrow$ InitializePheromoneMatrix(P) ▷ Initialize all pheromone levels to a constant.
$\eta \leftarrow$ PrecomputeHeuristics(P) ▷ Compute attractiveness based on distances and potential profits.
$s^* \leftarrow$ null
$f(s^*) \leftarrow -\infty$
no_improvement $\leftarrow 0$
for $g = 1$ to G **do**
    LocalSolutions $\leftarrow \emptyset$
    LocalFitnesses $\leftarrow \emptyset$
    for $k = 1$ to m **do** ▷ Each ant constructs a solution.
        $s_k \leftarrow$ ConstructRoute($P, \tau, \eta, \alpha, \beta$) ▷ Build a feasible route respecting vehicle capacity, starting/ending at depot, and selecting next packages probabilistically.
        $f(s_k) \leftarrow$ Fitness(s_k, P)
        Append s_k to LocalSolutions and $f(s_k)$ to LocalFitnesses
    end for
    $s_{\text{best\_local}} \leftarrow \arg\max_{s \in \text{LocalSolutions}} f(s)$
    if $f(s_{\text{best\_local}}) > f(s^*)$ **then**
        $s^* \leftarrow s_{\text{best\_local}}$ ▷ Update global best solution.
        $f(s^*) \leftarrow f(s_{\text{best\_local}})$
        no_improvement $\leftarrow 0$
    else
        no_improvement $\leftarrow$ no_improvement $+ 1$
        if no_improvement $> tol$ **then**
            break ▷ Stop early if no improvement beyond tolerance.
        end if
    end if
    UpdatePheromones!(τ, LocalSolutions, P, q, ρ) ▷ Evaporate pheromones and reinforce those on edges used by the found solutions.
end for
return $s^*, f(s^*)$

3.3.3 Modifications for VRP-TWC

Adapting ACO for the VRP-TWC involves representing each solution as a two-dimensional vector, where each row corresponds to the route taken by a particular vehicle. This variant generally follows the previously described process of constructing solutions and updating pheromone levels, but incorporates additional constraints and parameters.

Initially, the algorithm establishes a $(n+1) \times (n+1)$ pheromone matrix τ, where all values are set to 1 provided by the **InitializePheromoneMatrix(P)** function. In addition to this, an attractiveness matrix ν of the same dimensions is prepared

through the **PrecomputeHeuristics**(P) function, defined as $\nu_{i,j} = \frac{1}{(d_{i,j}+10^{-6})^{\beta}}$, where $d_{i,j}$ represents the distance between the vertices i and j, and β is a constant. The small constant 10^{-6} is included to avoid division by zero errors.

During the solution generation phase, **ConstructRoute**(P, τ, ν, α, β) is called for each of the k ants. This function works by repeatedly selecting the next package to be delivered from an unvisited set. This selection relies on the level of pheromones and the attractiveness between the current package and the candidates. Capacity constraints are strictly enforced: If assigning a package p to the current vehicle of the ant i does not exceed the allowed capacity Q, that package is chosen with high probability; if not, it may be skipped to encourage exploration. More formally, if the last visited package for ant i is q, then for each unvisited package p, the selection probability is proportional to the pheromone and attractiveness values $\tau_{p,q}$ and $\nu_{p,q}$. The ant then samples from these probabilities to choose the next package. This process is repeated until each ant constructs a complete route.

After all ants have built their routes, the **Update Pheromones**(τ, R_k, ρ, q, P) where R_k is the set of routes found by the k ants and q is the pheromone deposit parameter. The algorithm applies pheromone evaporation, $\tau \leftarrow (1-\rho)\tau$, where ρ is the evaporation constant. Subsequently, the reinforcement of the pheromones is carried out by adding a constant amount q to the entries $\tau_{i,j}$ corresponding to the edges used in the constructed solutions.

The entire process, which comprises the construction of routes, updating pheromones and ensuring constraints, repeats for a specified number of iterations or until a tolerance condition is met. This stopping criterion activates when the algorithm detects no significant improvement in the fitness function (2.1) after a certain number of iterations.

3.4 Hill-Climbing (HC)

Hill-Climbing (HC) is a local search algorithm that iteratively improves a single solution by exploring its neighborhood. Although HC is simpler than GA and ACO, it can effectively refine solutions by exploiting local improvements.

3.4.1 Principles of Hill-Climbing

The HC algorithm works by iteratively making small adjustments to the current solution: The general Hill Climbing algorithm can be described mathematically as follows:

- **Problem Definition:** Let $\mathcal{S}$ denote the solution space of the optimization problem, and let $f : \mathcal{S} \to \mathbb{R}$ represent the objective function to be maximized.
- **Initialization:** Start with an initial solution $s_0 \in \mathcal{S}$. This solution may be generated randomly or based on problem-specific heuristics.
- **Neighbor Generation:** For the current solution s_t, define a neighborhood $N(s_t) \subseteq \mathcal{S}$, which consists of solutions reachable from s_t through a single modification (e.g., swapping elements, changing assignments, etc.).

- **Selection:** Evaluate the fitness of a randomly generated subset of neighbors $s' \in N(s_t)$ using the objective function $f(s')$. Select the neighbor s_{t+1} such that:
$$s_{t+1} = \arg \max_{s' \in N(s_t)} f(s'),$$
provided $f(s_{t+1}) > f(s_t)$. If no such s_{t+1} exists, the algorithm is terminated.

- **Termination:** The algorithm stops if a predefined maximum number of iterations T is reached or no improvement is observed in the fitness function (local optimum).

3.4.2 Modifications for VRP-TWC

The hill-climbing algorithm is implemented in Julia and uses the following key function:

GenerateNeighbors(s, P, N_{count}): To explore the vicinity of the current solution s for the problem P, the algorithm produces a set of neighboring solutions of size N_{count} by making controlled, small modifications in the form of 3 different operators.

1. If multiple vehicles exist, one approach is to randomly select two different vehicles and attempt to move a package from the first to the second, provided that the capacity constraint is satisfied.

2. With small probability, the algorithm creates an entirely new vehicle and moves a package into it, thereby exploring a slightly more diverse range of solutions.

3. If there is only a single vehicle, the neighbor is generated by randomly swapping the order of two packages within that vehicle's route.

After performing these changes, a repair function is called to ensure the modified solution remains feasible, for example by reintroducing the depot in the correct positions or adjusting any routes that might have become invalid. This step guarantees that even after introducing variability, the resulting neighbors all represent valid solutions to the problem. By repeating this process multiple times, the algorithm constructs a diverse neighborhood of candidate solutions that can be evaluated to guide the Hill Climbing search toward better results.

3.5 Summary

This chapter outlined the methodology adopted to solve the Capacitated Vehicle Routing Problem with Time Windows and Variable Revenue (VRP-TWC). By employing three nature-inspired algorithms—Genetic Algorithm (GA), Ant-Colony Optimization (ACO), and Hill-Climbing (HC)—the study leverages their strengths in exploring large solution spaces and adapting to complex constraints. The Genetic Algorithm utilizes evolutionary principles with selection, crossover, and mutation operations, tailored to ensure feasible solutions under capacity and time-window restrictions. The Ant-Colony Optimization algorithm mimics

pheromone-based foraging behavior to identify efficient routes while accounting for revenue variability. Finally, the Hill-Climbing algorithm iteratively improves solutions through controlled local adjustments. Shared components, such as the fitness and repair functions, ensure consistency across methods. Together, these algorithms provide a robust framework for optimizing delivery routes, balancing cost minimization and revenue maximization. The subsequent chapters will analyze and compare their performance to evaluate their effectiveness in addressing the VRP-TWC.

Algorithm 4: Hill Climbing Algorithm for the VRP-TWC

Require: Problem instance P, maximum iterations T, tolerance tol, threshold ϵ.
Ensure: Best solution s^* and its fitness $f(s^*)$.

$s \leftarrow \text{RandomInitialSolution}(P)$ ▷ Generate an initial feasible solution.
RepairSolution(s, P) ▷ Ensure s is fully feasible (e.g., respects capacity, time-windows).
$s_{fitness} \leftarrow f(s)$ ▷ Evaluate fitness of the initial solution.
$s^* \leftarrow s$
$s^*_{fitness} \leftarrow f(s)$
no_improvement $\leftarrow 0$
$t \leftarrow 0$
while $t < T$ **do** ▷ Repeat until maximum iterations or no improvement for tol steps.
    $N(s) \leftarrow \text{GenerateNeighbors}(s, P, N_{\text{count}})$ ▷ Generate a set of neighbor solutions.
    for all $s' \in N(s)$ **do**
        RepairSolution(s', P) ▷ Ensure each neighbor is feasible.
        $s'_{fitness} \leftarrow f(s')$
    end for
    $s_{\text{best}} \leftarrow \arg\max_{s' \in N(s)} f(s')$ ▷ Select the best neighbor.
    if $f(s_{\text{best}}) > f(s)$ **then** ▷ Check for improvement over the current solution.
        $s \leftarrow s_{\text{best}}$
        $s_{fitness} \leftarrow f(s_{\text{best}})$
        $s^* \leftarrow s$
        $s^*_{fitness} \leftarrow f(s)$
        no_improvement $\leftarrow 0$
    else
        no_improvement $\leftarrow$ no_improvement $+ 1$
        if no_improvement $\geq tol$ **then**
            break ▷ Terminate if no improvement for tol consecutive iterations.
        end if
    end if
    $t \leftarrow t + 1$
end while
RepairSolution(s^*, P) ▷ Final repair step if needed.
return $s^*, f(s^*)$ ▷ Return the best solution found and its fitness.

4 Experiments

In this chapter, we present a detailed experimental analysis to evaluate the performance of the three nature-inspired algorithms in solving the Capacitated Vehicle Routing Problem with Time Windows and Variable Revenue (VRP-TWC). The goal of this chapter is twofold: first, to analyze the behavior and efficiency of each algorithm individually, and second, to compare their overall performance across various problem configurations.

Further experiments are designed to explore the influence of critical factors such as the ratio of vehicle capacity to package weight, the number of packages to be delivered, and the resulting runtimes of the algorithms. By systematically varying these parameters, we aim to identify patterns, trends, and relationships that affect the quality of the solutions obtained and the computational effort required. Specifically, we investigate:

- How the ratio of vehicle capacity to package weight influences the feasibility and profitability of the solutions.
- The impact of the number of packages on algorithm performance, both in terms of solution quality and computational runtime.
- Comparison of GA, ACO, and HC runtimes to assess their scalability and efficiency under different problem sizes.

Each experiment uses multiple test instances, with results analyzed using key performance indicators such as total profit (fitness) and execution time. The chapter concludes with a comparative analysis to highlight each algorithm's strengths, weaknesses, and practical applicability in solving the VRP-TWC under realistic constraints. Each algorithm is also run on the same problem instance multiple times to add resistance to random variations in results, which is built into heuristic-based optimization algorithms.

4.1 Obtaining Test Data

The test data for customer locations was generated based on a GeoJSON file exported from OpenStreetMap of an area of Prague, Czech Republic. The test data generation process for vehicle routing problems is implemented through a structured pipeline. The workflow consists of the following key components:

4.1.1 GeoJSON Data Processing

1. **File Validation**: The system first verifies the existence of the GeoJSON file used to generate customer locations.
2. **Residential Building Extraction**: The algorithm filters features using the following criteria:
    - Presence of a "building" property in the feature metadata
    - Specific classification as "residential" building type

- Valid geometry types (Polygon or MultiPolygon)

3. **Coordinate Extraction**: Spatial coordinates are collected from valid features.

4.1.2 Customer Location Generation

A set of N customer locations is randomly sampled from the extracted residential coordinates using a uniform distribution.

4.1.3 Distance Matrix Calculation

1. Pairwise distances are computed using the Haversine formula [12]:

$$d = 2R \cdot \left(\frac{\sqrt{a}}{\sqrt{1-a}} \right) \tag{4.1}$$

where $R = 6,371,000$m, (Earth's radius) and

$$a = \sin^2(\Delta\phi/2) + \cos\phi_1 \cos\phi_2 \sin^2(\Delta\lambda/2) \tag{4.2}$$

where

- ϕ_1, ϕ_2 are the latitudes of two points *in radians*
- λ_1, λ_2 are the longitudes of two points *in radians*
- $\Delta\phi = \phi_2 - \phi_1$ (difference in latitudes)
- $\Delta\lambda = \lambda_2 - \lambda_1$ (difference in longitudes)

2. Results are stored in a symmetric dictionary structure for efficient lookup.

4.1.4 Vehicle Configuration

The configuration of all vehicles used in the experiments was kept constant to maintain fairness between experiments and problem sets with different package sizes. The chosen configuration is that every vehicle has a capacity of 100 units.

4.1.5 Package Configuration

1. Each package is assigned:

   - Random weight between 1-100 units (100 being the vehicle's capacity).
   - Time-dependent revenue function with 1-10 breakpoints randomly selected.
   - Piecewise linear revenue profile with random values within $[d_{max}, 2d_{max}]$.

   where d_{max} represents the maximum pairwise distance in the data set to maintain positive profits.

This automated generation enables the reproducible (through a random seed) instances of varied VRP-TWC instances with configurable parameters and geographic constraints. The implementation ensures topological validity by deriving customer locations from actual residential areas while maintaining computational efficiency through pre-computed distance matrices.

The problem sets used to analyze the performance of the algorithms are divided into two categories, the identifying factor being the number of packages described in each problem instance. Easier problems are defined as problems with 10 to 100 packages; more complex problems have 100 to 1000 packages. The problem generation script can also generate easier debugging problems with fewer than 10 packages. The problem instances were generated using a script that used data obtained from OpenStreetMap[13] by retrieving residential buildings, assigning a package to each of these buildings, and then randomly generating package data such as weight, revenue and time windows.

Parameter	**Value**
Vehicle capacity	2
Time horizon	[0, 20]
Packages (excl. depot)	4
Distance multiplier	1
Time multiplier	1

Table 4.1: Problem Instance Summary

Described in 4.1 is a summary of a problem instance. This is a telling problem to test any heuristic algorithm on, as a greedy algorithm that prioritizes immediate profits is usually unable to find the optimal solution, which, due to the problem's small size, can be found and verified by hand.

Before running the genetic and ant-colony optimization algorithms on every problem instance, a random search is carried out for hyperparameters. During random search, a combination of hyperparameters is sampled randomly from sets of predefined values for each hyperparameter to determine which combination will likely yield better results. Random search for the GA hyperparameters is performed on crossover rate, mutation rate, population size, tournament size, and number of generations. Similarly, for ACO, the random search is performed for the number of ants and the values α, β, ρ, and q.

4.1.6 Problem Configuration

In terms of the problem configuration, a constant value of 1 was chosen for both time and distance multipliers. This was done to simplify the values and keep the experiments fair. Adding a non-unit multiplier depends entirely on the specific problem, and its effect is predictable. The impact on profitability is already known since a higher time multiplier or lower time multiplier will produce results based on the specific time-partitions described in the problem. It is also obvious that a higher distance multiplier results in an increase in the travel cost of vehicles and, therefore, a decrease in profitability.

4.2 Obtaining Hyperparameters

Hyperparameters are configuration parameters that control the learning process of optimization algorithms, distinct from model parameters that are learned during training. Their appropriate selection is crucial for algorithm performance, as they directly influence convergence speed, solution quality, and computational efficiency.

The iteration count for all algorithms was fixed at 500, a common benchmark in metaheuristic studies to balance solution quality and computational efficiency.

4.2.1 Genetic Algorithm Hyperparameters

The implemented genetic algorithm requires optimization of four key hyperparameters:

- **Population Size**: Number of candidate solutions in each generation (50, 100, 200)
- **Tournament Size**: Selection pressure in parent selection (2, 5, 10)
- **Crossover Rate**: Probability of genetic recombination (0.5, 0.6, 0.7, 0.8)
- **Mutation Rate**: Probability of random genetic changes (0.05, 0.1, 0.2)

The parameter space is explored using random search with 50 samples, evaluating each configuration's average fitness across multiple problem instances.

4.2.2 Ant Colony Optimization Hyperparameters

The ant colony optimization implementation requires tuning of five principal parameters:

- **Number of Ants (n_ants)**: (5, 10, 20, 50, 100, 200)
- **Alpha (α)**: Pheromone influence exponent (0.1, 0.5, 1, 2, 5, 10)
- **Beta (β)**: Heuristic information exponent (0.1, 0.5, 1, 2, 5, 10)
- **Rho (ρ)**: Pheromone evaporation rate (0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1.0)
- **Q**: Pheromone deposit factor (0.05, 0.1, 0.2, 0.3)

The optimization process employs an extended random search with 200 samples, using the following pheromone update rule:

4.2.3 Random Grid-Search

Grid search is a systematic hyperparameter optimization technique that evaluates all possible combinations of parameters from predefined discrete sets. For n hyperparameters with k_i possible values each, this requires evaluating $\Pi_{i=1}^{n} k_i$ configurations - a computationally expensive process that grows exponentially with parameter count. Due to the cost of running a comprehensive grid-search on a large problem instance, the random grid-search approach was utilized.

A random grid search is a hybrid approach that combines elements of grid search and random sampling. Instead of exhaustively evaluating all possible combinations of hyperparameters (as in traditional grid search), it:

- Defines a discrete "grid" of candidate values for each hyperparameter (predefined by the researcher)
- Randomly samples combinations from this grid
- Evaluates only the subset of these sampled configurations

This approach balances thoroughness with computational feasibility, which is particularly critical for optimization algorithms, where each evaluation is expensive. Random-Search is a technique that is used in practice and often produces similar results to a comprehensive grid-search [14].

The hyperparameters returned by the random grid-search for ACO and GA are as follows:

ACO Parameter	**Value**
Number of Ants (n_{ants})	200
Alpha (α)	2.0
Beta (β)	0.1
Rho (ρ)	0.1
Q	0.1

Table 4.2: Ant Colony Optimization (ACO) Hyperparameters

GA Parameter	**Value**
Population Size	200
Mutation Rate	0.2
Crossover Rate	0.8
Tournament Size	5

Table 4.3: Genetic Algorithm (GA) Hyperparameters

4.3 Relative Algorithm Performances

In this section, the relative performance of the three algorithms is analyzed and compared on identical problem instances in multiple runs of the algorithms. The results shown in this section have been obtained using the algorithm described in algorithm 5, which runs each algorithm $\mathbf{R = 10}$ times on each problem instance to reduce the effect of random deviations in the solutions, despite random errors are built into the nature of the algorithms being studied. The results are then aggregated into a data frame to plot the fitness and runtime graphs shown in Figure 4.1 and Figure 4.2, respectively.

Algorithm 5: Run Experiments with Normalization

Require: Set of problem instances $\mathcal{P}$, list of algorithms $\mathcal{A}$, number of runs R
Ensure: Normalized fitness and runtimes for each algorithm

```
results ← Dict()                                   ▷ Initialize results dictionary
problem_counter ← 1
for all p ∈ 𝒫 do
    fitnesses_by_algorithm ← {}
    runtimes_by_algorithm ← {}
    for all a ∈ 𝒜 do
        algorithm_fitnesses ← [ ]
        algorithm_runtimes ← [ ]
        for r ← 1 to R do
            start_time ← time()
            (solution, fitness) ← solve(p, a)
            runtime ← time() − start_time
            Append fitness to algorithm_fitnesses
            Append runtime to algorithm_runtimes
        end for
        fitnesses_by_algorithm[a] ← mean(algorithm_fitnesses)
        runtimes_by_algorithm[a] ← mean(algorithm_runtimes)
    end for
    best_avg_fitness ← max({f | f ∈ fitnesses_by_algorithm})
    for all a ∈ 𝒜 do
        normalized_fitnesses[a] ←
            [ f / best_avg_fitness | f ∈ fitnesses_by_algorithm[a] ]
        results[p][a] ←
            {
              NormalizedFitness: normalized_fitnesses[a],
              Runtimes: runtimes_by_algorithm[a]
            }
    end for
end for
return results
```

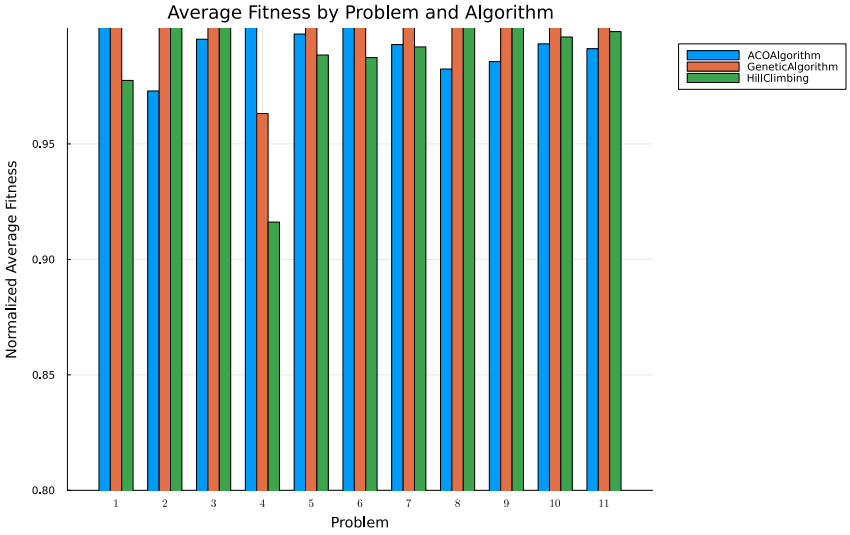

Figure 4.1: Normalized fitnesses of solutions to the problem set produced by each algorithm (higher is better).

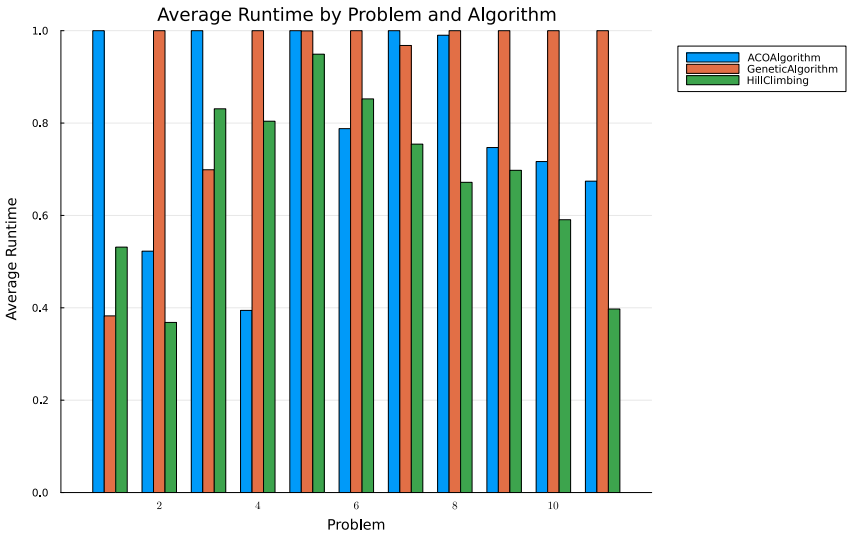

Figure 4.2: Normalized runtimes of algorithms to the problem set produced by each algorithm (higher is better).

4.4 Relation Between Number of Packages and Runtimes

In this section, the runtimes of the algorithms are analyzed with respect to the number of packages contained in a problem instance. This experiment was carried out with the same number of iterations for each algorithm and the same number of solutions generated for each algorithm to maintain consistency.

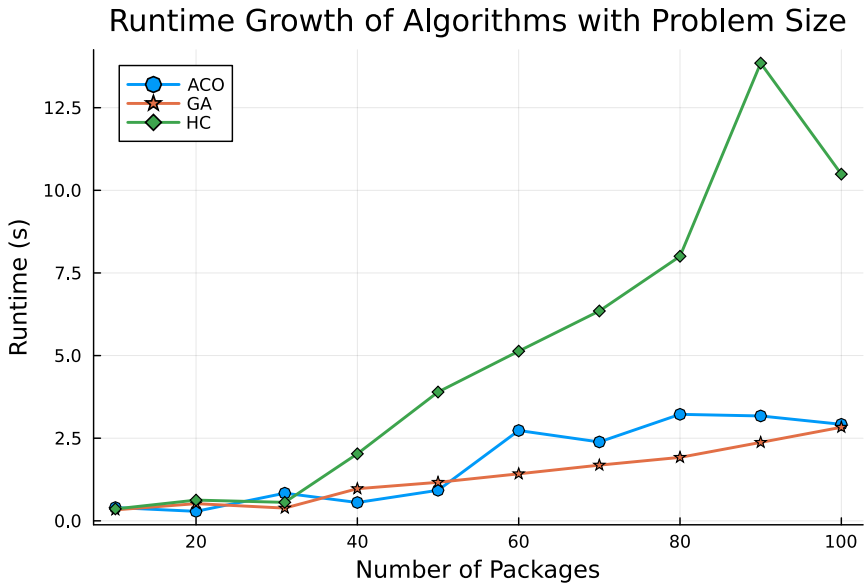

Figure 4.3: Algorithm Runtime relation to number of packages in medium problem set.

Across the medium problem set containing instances with packages ranging in number from 10 to 100, the hill-climbing algorithm was affected the most by the increase in the number of packages, resulting in a longer run-time. This can be due to the nature of the neighbor generation approach implemented in the algorithm, which can cause it to be slower than the other two algorithms. The GA and ACO, however, tend to perform similarly for the most part up until higher package counts, where GA seems to be more consistent.

The algorithms were also tested on a hard set of problems with the number of packages ranging from 100 to 1000. The results are shown in Figure 4.4.

4.5 Impact of Package Weight to Vehicle Capacity Ratio on Profitability

To better understand how the ratio of package weight to vehicle capacity impacts profitability, this problem instance was conducted using problem instances specifically designed to test this relationship.

Experiment Setup

Five different sets consisting of 10 randomly generated problems were created, each characterized by packages with maximum total weights corresponding to 20%,

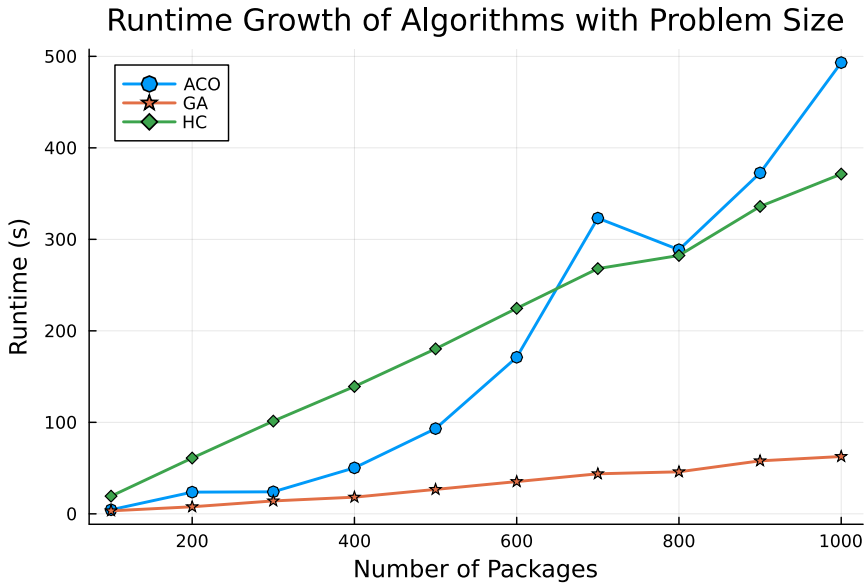

Figure 4.4: Algorithm Runtime relation to number of packages in hard problem set.

40%, 60%, 80%, and 100% of the vehicle's maximum capacity, respectively. The number of customers in each of the categories was set to 200 to maintain consistency and simulate a decently sized problem. These problem sets allow us to analyze the influence of this ratio on the overall profitability of solutions generated by the three algorithms: Genetic Algorithm (GA), Ant-Colony Optimization (ACO), and Hill-Climbing (HC). Each algorithm was run on each problem instance 10 times, and the fitness results were averaged to reduce random uncertainty. Each problem instance in the five sets followed the standard VRP-TWC schema and included:

- A predefined set of packages with weights that adhere to the respective percentage thresholds.
- A fixed vehicle capacity, ensuring consistent comparability across problem instances.
- Geographical data simulated using OpenStreetMaps.

The algorithms were executed on these problem sets with optimized hyperparameters obtained via random search described in section 4.2.

Results and Analysis

The experimental results are summarized across the five capacity intervals, highlighting how the package weight to vehicle capacity ratio impacts the algorithmic performance. The analysis focuses on the average profitability achieved by each algorithm for each ratio range. The accompanying plot 4.5 provides a representation of the results, showing a quartile box-plot and illustrating trends and variations across the different intervals.

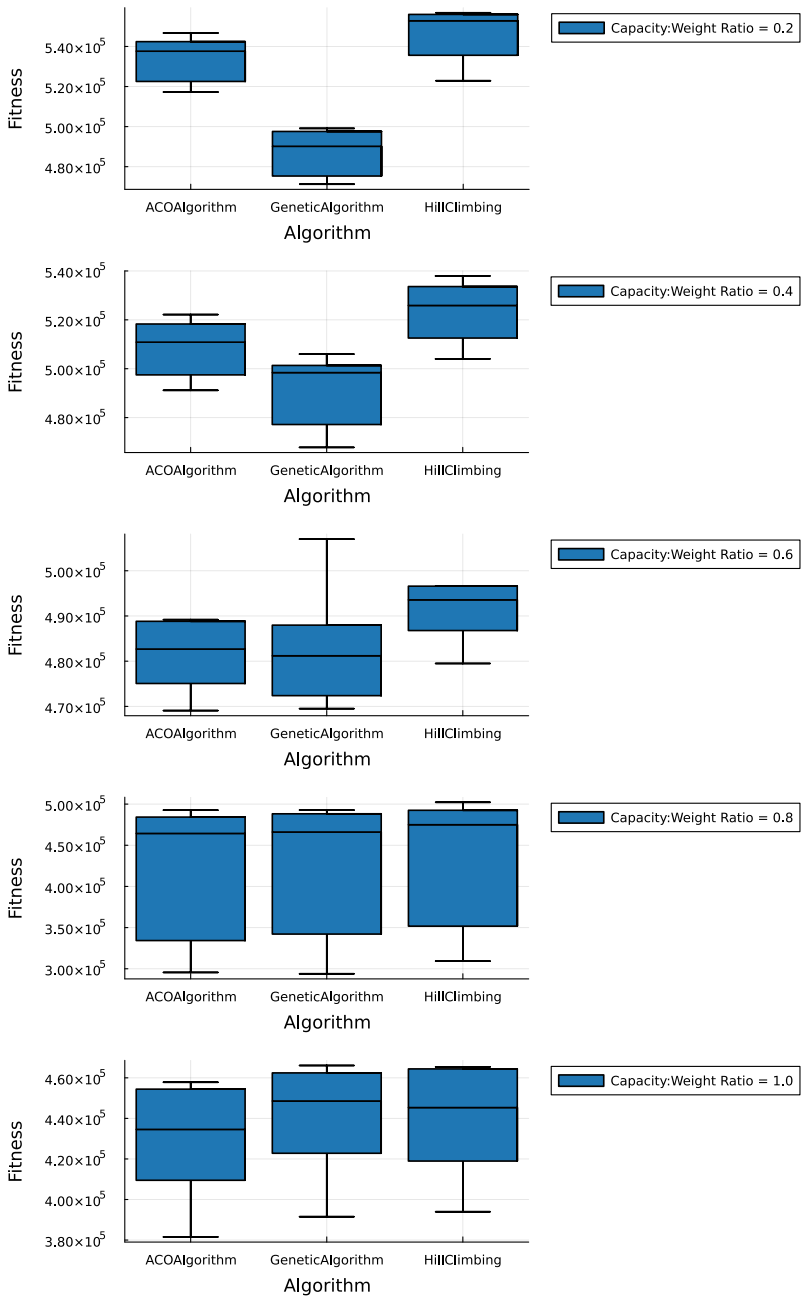

Figure 4.5: Impact of package weight to vehicle capacity ratio on solution fitness.

Conclusion

This thesis presented a comparative study of three nature-inspired algorithms—Genetic Algorithm (GA), Ant-Colony Optimization (ACO), and Hill-Climbing (HC) to solve the Capacitated Vehicle Routing Problem with Time Windows with Variable Revenue and Capacity constraints (VRP-TWC). The objective was to determine efficient delivery routes that maximize revenue while adhering to vehicle capacity constraints and time-window restrictions.

Summary of Findings

The experiments were designed to analyze the algorithms' performance across different problem complexities (easy, medium, and hard) and under varying conditions such as package count and vehicle capacities. The key findings can be summarized as follows:

- **Relative Algorithm Performance:** GA provided strong solutions while achieving the best runtimes overall. HC scaled efficiently with increasing problem size, offering competitive solutions at the cost of only a slight runtime increase over GA. ACO, while performing reasonably well for small problems, exhibited the worst runtimes as the problem size increased beyond 100 packages.
- **Scalability with Package Count:** The runtimes of all algorithms increased with the number of packages, but the extent of this increase varied significantly. ACO's and HC's performance degraded for large problem instances, while HC scaled more effectively. However, both algorithms were outperformed by GA, which was multiple times faster than both HC and ACO for larger problem instances.
- **Relation between vehicle capacity to package weight ratio and the quality of solutions:** The experiments revealed a clear relationship between vehicle capacity to package weight ratio and solution quality between algorithms. As the ratio decreased, indicating tighter capacity constraints, the quality of solutions tended to degrade, especially for GA. HC maintained robust performance in varying ratios, suggesting resilience to limited capacity. GA showed a notable sensitivity at lower ratios, but recovered quickly as the ratio increased. ACO also had a good showing for lower ratios but struggled more as the ratio increased, possibly indicating being stuck in local optima. Overall, the behavior at higher ratios is expected since the bigger the packages relative to the vehicle's capacity, the fewer choices each algorithm has to make in terms of organizing the packages in the vehicles.
- **Algorithm-Specific Behavior:**
  - **Genetic Algorithm (GA):** GA delivered strong, consistent solutions across all problem sizes and achieved the best runtime performance. Its evolutionary operators (crossover and mutation) ensured diversity and efficient exploration of the solution space.

- **Hill-Climbing (HC):** HC scaled efficiently with increasing problem size, offering solutions that were often competitive with GA while incurring only a minor increase in runtime. It is also worth noting that in the experiments, Hill-Climbing was especially susceptible to getting stuck in local optima.
- **Ant-Colony Optimization (ACO):** ACO struggled with runtime performance for large problem sizes, particularly as the number of packages exceeded 100. Although capable of finding reasonable solutions, its computational cost increased significantly and might benefit from a parallelized implementation.

Practical Implications

The results of this study provide valuable insights for solving real-world vehicle routing problems:

- For **small to medium-sized instances**, GA is the most effective choice, providing strong solutions with minimal runtime overhead. HC can also be considered when slightly longer runtimes are acceptable.
- For **large-scale problems**, GA emerges as the most scalable approach, offering competitive solutions with manageable runtime growth.
- ACO is less suitable for large problem instances due to its poor scalability and increasing computational costs.

Future Work

This research can be extended in several directions:

- Exploring hybrid algorithms that combine the strengths of GA, HC, and ACO to enhance solution quality and scalability further.
- Investigating parallelized implementations to reduce runtime for large problem instances. It is obvious that the experiments conducted in this thesis suffered from a lack of parallelization, and therefore, especially the figures in the runtime experiments can be improved on.
- Utilizing a different crossover or neighbor generation approach for the genetic algorithm and hill-climbing algorithms respectively could also be considered to improve the performance.
- Incorporating additional real-world constraints, such as traffic conditions or dynamic time windows, to make the VRP-TWC model more applicable to real logistics systems.

In conclusion, this thesis demonstrates the effectiveness of nature-inspired algorithms in solving the VRP-TWC, with GA achieving the best overall performance in terms of runtime and solution quality. HC provides a scalable alternative for

large problem instances, while ACO highlights areas for improvement in computational efficiency for more extensive scenarios. These findings contribute to the development of efficient and scalable optimization techniques for modern logistics challenges.

Bibliography

1. Toth, Paolo; Vigo, Daniele. *The vehicle routing problem*. SIAM, 2002.
2. Dantzig, George B; Ramser, John H. The truck dispatching problem. *Management science*. 1959, vol. 6, no. 1, pp. 80–91.
3. Caceres-Cruz, Jose; Arias, Pol; Guimarans, Daniel; Riera, Daniel; Juan, Angel A. Rich Vehicle Routing Problem: Survey. *ACM Comput. Surv.* 2014, vol. 47, no. 2. ISSN 0360-0300. Available from DOI: 10.1145/2666003.
4. Lenstra, Jan Karel; Kan, AHG Rinnooy. Complexity of vehicle routing and scheduling problems. *Networks*. 1981, vol. 11, no. 2, pp. 221–227.
5. Dixit, Aditya; Mishra, Apoorva; Shukla, Anupam. Vehicle routing problem with time windows using meta-heuristic algorithms: a survey. In: *Harmony Search and Nature Inspired Optimization Algorithms: Theory and Applications, ICHSA 2018*. Springer, 2018, pp. 539–546.
6. Kara, Imdat; Bektas, Tolga. Integer linear programming formulation of the generalized vehicle routing problem. In: *Proc. of the 5-th EURO/INFORMS Joint International Meeting*. Citeseer Istanbul, 2003, pp. 06–10.
7. Nazari, MohammadReza; Oroojlooy, Afshin; Snyder, Lawrence; Takac, Martin. Reinforcement Learning for Solving the Vehicle Routing Problem. In: Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; Garnett, R. (eds.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018, vol. 31. Available also from: https://proceedings.neurips.cc/paper_files/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf.
8. Holland, John H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975. ISBN 9780262581110.
9. Dorigo, Marco; Stützle, Thomas. *Ant Colony Optimization*. MIT Press, 2004. ISBN 9780262042192.
10. Russell, Stuart; Norvig, Peter. Artificial Intelligence: A Modern Approach. *Pearson Education Limited*. 3rd. 2016. ISBN 9781292153964.
11. Schrotenboer, Albert H.; Uit het Broek, Michiel A. J.; Buijs, Paul; Ulmer, Marlin W. Fighting the E-commerce Giants: Efficient Routing and Effective Consolidation for Local Delivery Platforms. *arXiv preprint arXiv:2108.12608*. 2021. Available also from: https://arxiv.org/abs/2108.12608.
12. Wikipedia contributors. *Haversine formula — Wikipedia, The Free Encyclopedia*. 2025. Available also from: https://en.wikipedia.org/w/index.php?title=Haversine_formula&oldid=1284363134. [Online; accessed 25-April-2025].
13. OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org* [https://www.openstreetmap.org]. 2017.
14. Bergstra, James; Bengio, Yoshua. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 2012, vol. 13, no. null, pp. 281–305. ISSN 1532-4435.

List of Figures

List of Tables