Solving MDHFVRP and Developing a User Interface for Agricultural Logistics Planning

KHERMANE Zakaria[1] and ELOUNEG Amine[1]

Supervised by: ANNAD Oussama[1]

Abstract

This study addresses the resolution of a complex Vehicle Routing Problem (VRP) for CPH Agriculture, specifically a Multi-Depot Heterogeneous Capacitated Vehicle Routing Problem (MDHFVRP). The primary objective is to optimize agricultural inputs distribution by minimizing logistics costs while satisfying operational constraints.

Two distinct approaches were developed and compared: an exact method using the IBM ILOG OPL CPLEX solver, based on the branch-and-cut algorithm, and an approximate method employing a genetic metaheuristic. The mathematical modeling of the problem was formulated, taking into account the company's specificities such as multiple depots, heterogeneous fleet, and capacity constraints. A comparative analysis of both methods performance was conducted, evaluating their effectiveness in terms of solution quality and computation time for various problem sizes.

Results demonstrate that the exact method is more suitable for small-scale problems, offering optimal solutions, while the genetic algorithm proves more efficient for large instances, providing good quality solutions within reasonable computation times.

Additionally, a graphical user interface named VRS (Vehicle Routing Solver) was developed to facilitate the solution's use by CPH Agriculture's logisticians.

Keywords: Vehicle Routing Problem, multi-depot, heterogeneous fleet, exact method, genetic algorithm, logistics optimization

Introduction :

The Vehicle Routing Problem (VRP) has been a cornerstone of logistics optimization since its introduction by Dantzig & Ramser [1] in 1959. As supply chains grow increasingly complex, variants such as the Multi-Depot Heterogeneous Capacitated Vehicle Routing Problem MDHFVRP have emerged to address real-world challenges (Bettinelli et al., 2011) [2]. This study focuses on solving the MDHFVRP for CPH Agriculture, a company facing the intricate task of distributing agricultural inputs from multiple depots using a diverse fleet of vehicles.

The importance of efficient routing in agriculture cannot be overstated. As Toth & Vigo [3] point out, optimizing distribution networks can lead to significant cost savings and improved service quality. In the agricultural sector, where margins are often tight and product freshness is crucial, these optimizations can make the difference between profit and loss.

This paper presents a comprehensive approach to solving CPH Agriculture's distribution challenges. We detail the mathematical formulation of the problem, describe the implementation of both an exact method using IBM ILOG OPL CPLEX and a genetic algorithm, we also conduct a comparative analysis of their computational performances in terms of solution quality and execution time. Additionally, we discuss the development of a user-friendly interface for practical application.

Related Works

The Multi-Depot Heterogeneous Capacitated Vehicle Routing Problem (MDHFVRP) represents a complex extension of the classical Vehicle Routing Problem (VRP), incorporating multiple depots and a heterogeneous fleet with varying capacities. Several key studies have addressed this specific problem or closely related variants:

Salhi et al. (2014) [4] provided a comprehensive formulation of the (MDHFVRP) and proposed a variable neighborhood search implementation. Their work is fundamental in understanding the complexities involved in solving this variant of the VRP. They presented a mathematical model and developed a metaheuristic approach specifically tailored to the multi-depot and heterogeneous fleet aspects of the problem.

Bolaños et al. (2018) [5] proposed a metaheuristic algorithm for solving the Multi-Depot Vehicle Routing Problem with a Heterogeneous Fleet (MDHFVRP). Their approach uses a modified genetic algorithm combined with local search strategies, featuring a hybrid initialization procedure and inter-route and intra-route neighborhood structures for solution improvement. This flexible framework can adapt to similar routing problems, demonstrating the effectiveness of combining population-based methods with local search techniques for complex VRP variants.

Nucamendi-Guillén et al. (2020) [6] introduced the multi-depot open location routing problem with a heterogeneous fixed fleet (MD-OLRP). They developed a Mixed Integer Linear Programming (MILP) model to minimize total cost, select carriers to be contracted, vehicles to be used, and collection routes. To solve larger instances, they proposed an intelligent metaheuristic incorporating problem-specific knowledge. Their metaheuristic is a multi-start algorithm with two phases: construction and improvement. The construction phase selects a subset of vehicles and builds initial routes, while the improvement phase applies various local search techniques. Their main contributions include the formulation of a new problem statement, its application to a real-world case, and the development of an effective metaheuristic to solve it.

Our work builds upon these foundational studies, aiming to contribute to the field by comparing exact and metaheuristic approaches specifically tailored to the MDHVVRP, with a focus on practical applications in agricultural logistics.

2 Problem Description and Mathematical Formulation

2.1 The Multi-Depot Heterogeneous Capacitated Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a fundamental optimization challenge in logistics, involving the design of optimal routes for a fleet of vehicles to serve a set of geographically dispersed customers while minimizing total transportation costs (Dantzig, G. B., & Ramser, J. H., 1959) [1]. Our study focuses on a complex variant known

as the Multi-Depot Heterogeneous Capacitated Vehicle Routing Problem (MDHFVRP) which extends the classical VRP by incorporating multiple depots and a heterogeneous fleet of vehicles with varying capacities and operational costs [7]. This variant precisely matches the problem faced by CPH Agriculture company.

2.2 Mathematical Model

To address this complex problem, we adapt the mathematical model proposed by Salhi et al. [4] for the MDHFVRP. The model is formulated as follows :

Let :

N : the set of all nodes, respectively depots and clients, indexed from 1 to (m + n), where:

- depots are indexed from 1 to m, where m is the number of depots.
- clients are indexed from (m + 1) to (m + n), where n is the number of clients.

V : the set of vehicle types, indexed from 1 to v.

PO_i : weight of the order for client i

VO_i : volume of the order for client i

CP^k : maximum weight capacity for vehicle type k

CV^k : maximum volume capacity for vehicle type k

Cf^k : fixed cost of using vehicle type k

Cva^k : variable (per kilometer) cost for vehicle type k

W^k : fleet size for vehicle type k

D_{ij} : distance between nodes i and j (i, j = 1.., n+m).

Decision Variables:

$$X_{ij}^{kd} = \begin{cases} 1 \; if \; vehicle \; type \; k \; originating \; from \; depot \; d \; travels \; arc \; (i,j) \\ 0 \; otherwise \end{cases} \quad \begin{matrix} \forall i,j = 0 \dots n \; ; \\ \forall k = 1 \dots v \end{matrix} \tag{1.1}$$

Y_{ij} : Non-negative continuous variable that gives the total remaining weight in the vehicle before reaching node j by traveling arc (i,j) (1.2)

Z_{ij} : Non-negative continuous variable that gives the total remaining volume in the vehicle before reaching node j by traveling arc (i,j) (1.3)

Objective Function :

$$min \sum_{d=1}^{m} \sum_{k=1}^{K} \sum_{i=1}^{m} \sum_{j=m+1}^{m+n} Cf^k \times X_{ij}^{kd} + \sum_{d=1}^{m} \sum_{k=1}^{K} \sum_{i=1}^{m+n} \sum_{j=1}^{m+n} Cva^k \times D_{ij} \times X_{ij}^{kd} \tag{1.4}$$

Subject to :

$$\sum_{d=1}^{m}\sum_{k=1}^{v}\sum_{i=1}^{m+1} X_{ij}^{kd} = 1 \qquad \forall j = m+1 \dots m+n \qquad (1.5)$$

$$\sum_{d=1}^{m}\sum_{k=1}^{v}\sum_{j=1}^{m+1} X_{ij}^{kd} = 1 \qquad \forall i = m+1 \dots m+n \qquad (1.6)$$

$$\sum_{i=1}^{m+1} X_{ij}^{kd} = \sum_{i=1}^{m+1} X_{ji}^{kd} \qquad \forall k = 1 \dots v; \forall j = 1 \dots m+n; \ \forall d = 1 \dots m \qquad (1.7)$$

$$\sum_{i=1}^{m}\sum_{j=m+1}^{m+n} Y_{ij} = \sum_{j=m+1}^{m+n} PO_j \qquad (1.8)$$

$$\sum_{i=1}^{m}\sum_{j=m+1}^{m+n} Z_{ij} = \sum_{j=m+1}^{m+n} VO_j \qquad (1.9)$$

$$\sum_{i=1}^{m+n} Y_{ij} - \sum_{i=1}^{m+n} Y_{ji} = PO_j \qquad \forall j = m+1 \dots m+n \qquad (1.10)$$

$$\sum_{i=1}^{m+n} Z_{ij} - \sum_{i=1}^{m+n} Z_{ji} = VO_j \qquad \forall j = m+1 \dots m+n \qquad (1.11)$$

$$Y_{ij} \le \sum_{d=1}^{m}\sum_{k=1}^{v} CPo^k \times X_{ij}^{kd} \qquad \forall i = 1 \dots m+n; \ \forall j = m+1 \dots m+n \qquad (1.12)$$

$$Z_{ij} \le \sum_{d=1}^{m}\sum_{k=1}^{v} CV^k \times X_{ij}^{kd} \qquad \forall i = 1 \dots m+n; \ \forall j = m+1 \dots m+n \qquad (1.13)$$

$$X_{di}^{kb} = 0 \qquad \forall i = m+1 \dots m+n; \ \forall k = 1 \dots v; \ d \ne b = 1 \dots m \qquad (1.14)$$

$$X_{id}^{kb} = 0 \qquad \forall i = m+1 \dots m+n; \ \forall k = 1 \dots v; \ d \ne b = 1 \dots m \qquad (1.15)$$

$$Y_{ij} \le \sum_{d=1}^{m}\sum_{k=1}^{v} (CPo^k - PO_i) \times X_{ij}^{kd} \qquad \forall i = 1 \dots m+n; \ \forall j = m+1 \dots m+n \qquad (1.16)$$

$$Z_{ij} \le \sum_{d=1}^{m}\sum_{k=1}^{v} (CV^k - VO_i) \times X_{ij}^{kd} \qquad \forall i = 1 \dots m+n; \ \forall j = m+1 \dots m+n \qquad (1.17)$$

$$Y_{ij} \ge \sum_{d=1}^{m}\sum_{k=1}^{v} PO_j \times X_{ij}^{kd} \qquad \forall i = m+1 \dots m+n; \ \forall j = m+1 \dots m+n \qquad (1.18)$$

$$Z_{ij} \ge \sum_{d=1}^{m}\sum_{k=1}^{v} VO_j \times X_{ij}^{kd} \qquad \forall i = m+1 \dots m+n; \ \forall j = m+1 \dots m+n \qquad (1.19)$$

$$Y_{ij} = 0 \qquad \forall i = m+1 \dots m+n; \ \forall j = 1 \dots m \qquad (1.20)$$

$$Z_{ij} = 0 \qquad \forall i = m+1 \dots m+n; \ \forall j = 1 \dots m \qquad (1.21)$$

$$Y_{ij} = 0 \qquad \forall i = 1 \dots m; \ \forall j = 1 \dots m \qquad (1.22)$$

$$Z_{ij} = 0 \qquad \forall i = 1 \dots m;\ \forall j = 1 \dots m \qquad (1.23)$$

$$Y_{ii} = 0 \qquad \forall i = m+1 \dots m+n \qquad (1.24)$$

$$Z_{ii} = 0 \qquad \forall i = m+1 \dots m+n \qquad (1.25)$$

$$X_{ij}^{kd} = 0 \qquad \forall i = 1 \dots m;\ \forall j = 1 \dots m;\ \forall k = 1 \dots v;\ \forall d = 1 \dots m \qquad (1.26)$$

$$X_{ii}^{kd} = 0 \qquad \forall i = 1 \dots m+n;\ \forall k = 1 \dots v;\ \forall d = 1 \dots m \qquad (1.27)$$

$$\sum_{d=1}^{m} \sum_{i=1}^{m} \sum_{j=m+1}^{m+n} X_{ij}^{kd} \leq W^k \qquad \forall k = 1 \dots v \qquad (1.28)$$

The objective function (1.4) seeks to minimize the total cost of the tours. Constraint (1.5) ensures that each client is visited only once. Constraint (1.6) guarantees that at most one type of vehicle from a given depot will cover an arc (i,j). Constraint (1.7) ensures the conservation of flows (a vehicle entering a client site must leave it). Constraints (1.8) and (1.9) ensure that the total weight and volume of goods leaving all depots are exactly equal to the total weight and volume of goods requested by all clients, respectively. Constraints (1.10) and (1.11) ensure that the remaining weight and volume after visiting client j are exactly the weight and volume before visiting this client minus the weight and volume of their order, respectively. Constraints (1.12) and (1.13) ensure that the weight and volume capacities of the vehicle of any type are not exceeded, respectively. Constraints (1.14) and (1.15) ensure that a vehicle leaving a depot (or returning to a depot) cannot be connected to a different depot, respectively. Constraints (1.16) and (1.17) impose that the weight and volume on an arc cannot be greater than the weight and volume on the vehicle after delivery to client iii on the arc (i,j) and originating from any depot, respectively. Constraints (1.18) and (1.19) Constraints (1.18) and (1.19) ensure that the remaining weight and volume after traveling arc (i,j) are non-negative, respectively. These constraints together form a comprehensive model to optimize vehicle routing considering various practical constraints related to vehicle capacity, order fulfillment, and cost minimization.

This comprehensive mathematical formulation captures the complexities of CPH Agriculture's distribution system, providing a solid foundation for developing solution approaches.

3. Solution Approaches

To address the complex MDHFVRP for CPH Agriculture, we employed two distinct approaches: an exact method using IBM ILOG CPLEX and an approximate method using a Genetic Algorithm. This dual approach allows us to compare the performance and applicability of both methods for various problem sizes.

3.1 Exact Method: IBM ILOG OPL CPLEX

We utilized IBM ILOG CPLEX Optimization Studio for the exact solution method. For problems involving integer variables, such as our MDHFVRP, CPLEX employs a Branch-and-Cut approach [8].

```
// Declarations of sets and indices
int n = ...;
int m = ...;
int K = ...;
// Definition of sets
range N = 1..m+n;
range depots = 1..m;
range clients = m+1..m+n;
range V = 1..K;
// Parameters
float PO[N] = ...;
float VO[N] = ...;
float CPo[V] = ...;
float CV[V] = ...;
int W[V] =...;
float Cf[V] =...;
float Cva[V] =...;
float D[N][N] = ...;
// Decision variables
dvar boolean X[N][N][V][depots];
dvar float+ Y[N][N];
dvar float+ Z[N][N];
// Objective function
minimize
     sum(d in 1..m,k in 1..K,i in 1..m,j in m+1..m+n)Cf[k]*X[i][j][k][d]
     +sum(d in 1..m, k in 1..K, i in 1..m+n, j in 1..m+n )Cva[k]*D[i][j]*X[i][j][k][d];
// Constraints
subject to {
      // Constraint (1.5)
      forall(j in m+1..m+n)
        sum(d in 1..m, k in 1..K, i in 1..m+n)X[i][j][k][d]==1;
     // Constraint (1.6)
     forall(i in m+1..m+n)
         sum(d in 1..m, k in 1..K, j in 1..m+n)X[i][j][k][d]==1;
     // Constraint (1.7)
     forall(k in 1..K, j in 1..m+n, d in 1..m)
         sum(i in 1..m+n)X[i][j][k][d]==sum(i in 1..m+n)X[j][i][k][d];
     // Constraint (1.8)
     sum(i in 1..m, j in m+1..m+n)Y[i][j]==sum(j in m+1..m+n)PO[j];
     // Constraint (1.9)
     sum(i in 1..m, j in m+1..m+n)Z[i][j]==sum(j in m+1..m+n)VO[j];
      // Constraint (1.10)
      forall(j in m+1..m+n)
         sum(i in 1..m+n)Y[i][j]-sum(i in 1..m+n)Y[j][i]==PO[j];
      // Constraint (1.11)
      forall(j in m+1..m+n)
         sum(i in 1..m+n)Z[i][j]-sum(i in 1..m+n)Z[j][i]==VO[j];
     // Constraint (1.12)
     forall(i in 1..n+m, j in m+1..m+n)
         Y[i][j]<=sum(d in 1..m, k in 1..K)CPo[k]*X[i][j][k][d];
     // Constraint (1.13)
     forall(i in 1..n+m, j in m+1..m+n)
         Z[i][j]<=sum(d in 1..m, k in 1..K)CV[k]*X[i][j][k][d];
    // Constraint (1.14)
    forall(i in m+1..m+n, k in 1..K, d in 1..m, b in 1..m: b!=d) X[d][i][k][b]==0;
     // Constraint (1.15)
     forall(i in m+1..m+n, k in 1..K, d in 1..m, b in 1..m: b!=d) X[i][d][k][b]==0;
     // Constraint (1.16)
     forall(i in 1..m+n, j in m+1..m+n)
         Y[i][j]<=sum(d in 1..m, k in 1..K)(CPo[k]-PO[i])*X[i][j][k][d];
     // Constraint (1.17)
     forall(i in 1..m+n, j in m+1..m+n)
         Z[i][j]<=sum(d in 1..m, k in 1..K)(CV[k]-VO[i])*X[i][j][k][d];
    // Constraint (1.18)
    forall(i in m+1..m+n, j in m+1..m+n)
         Y[i][j]>=sum(d in 1..m, k in 1..K)PO[j]*X[i][j][k][d];
    // Constraint (1.19)
    forall(i in m+1..m+n, j in m+1..m+n)
         Z[i][j]>=sum(d in 1..m, k in 1..K)VO[j]*X[i][j][k][d];
     // Constraint (1.20)
     forall(i in m+1..m+n, j in 1..m) Y[i][j]==0;
     // Constraint (1.21)
     forall(i in m+1..m+n, j in 1..m) Z[i][j]==0;
     // Constraint (1.22)
     forall(i in 1..m, j in 1..m) Y[i][j]==0;
     // Constraint (1.23)
     forall(i in 1..m, j in 1..m) Z[i][j]==0;
     // Constraint (1.24)
     forall(i in m+1..m+n) Y[i][i]==0;
     // Constraint (1.25)
     forall(i in m+1..m+n) Z[i][i]==0;
    // Constraint (1.26)
    forall(i in 1..m, j in 1..m, k in 1..K, d in 1..m)X[i][j][k][d]==0;
    // Constraint (1.27)
    forall(i in 1..m+n, k in 1..K, d in 1..m)X[i][i][k][d]==0;
    // Constraint (1.28)
    forall(k in 1..K)
      sum(d in 1..m, i in 1..m, j in m+1..m+n) X[i][j][k][d]<=W[k];
}
```

Figure 1: Implementation of the MDHFVRP Mathematical Model in CPLEX

3.2 Approximate Method: Genetic Algorithm

3.2.1 Algorithm Design

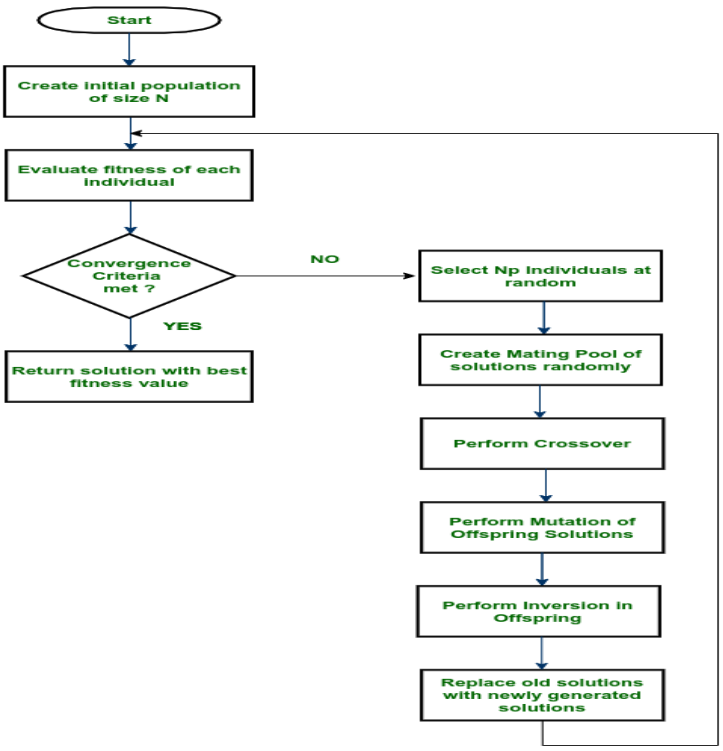

Figure 2: Steps of the Genetic Algorithm [9]

The Algorithm follows a standard structure (Mitchell [10]), as illustrated in Figure 2 :

1. Initial Phase :
    - Chromosome Creation: Randomly generate an initial population of solutions.
    - Fitness Evaluation: Calculate the fitness of each chromosome based on total cost.
2. Evolutionary Loop :
    - Parent Selection: Use methods like roulette wheel or tournament selection.
    - Crossover: Recombine parent chromosomes to produce offspring.
    - Mutation: Introduce random variations in offspring to explore new solution spaces.
    - Evaluation: Calculate fitness of new chromosomes.
    - Population Update: Form new generation by selecting best individuals from parents and offspring.
3. Termination :
    - Stop when maximum generations are reached or a satisfactory solution is found.

3.2.2 Implementation in Python

The following figure shows the Genetic Algorithm's implementation in Python with specific adaptations for the MDHFVRP [11] :

```python
# GA-VRP Function
def genetic_algorithm_vrp(coordinates, distance_matrix_local, parameters, fixed_cost, variable_cost, capacity_volume,
                          capacity_weight, population_size=5, vehicle_types=1, n_depots=1, route='closed',fleet_size=[],
                          mutation_rate=0.1, crossover_rate=0.1, elite=0, generations=50, penalty_value=1000,
                          graph=True, map=True, selection='rw'):
    start = tm.time()
    count = 0
    solution_report = ['None']
    max_capacity_volume, max_capacity_weight = copy.deepcopy(capacity_volume), copy.deepcopy(capacity_weight)
    for i in range(0, n_depots):
        parameters[i, 0] = 0
        parameters[i, 1] = 0
        parameters[i, 2] = 0
    population = initial_population(coordinates, distance_matrix_local, population_size=population_size,
                                    vehicle_types=vehicle_types, n_depots=n_depots)
    cost, population = target_function(population, distance_matrix_local, parameters, fixed_cost, variable_cost,
                                       max_capacity_volume, max_capacity_weight, penalty_value,route=route,
                                       fleet_size=fleet_size)
    cost, population = (list(t) for t in zip(*sorted(zip(cost, population))))
    if (selection == 'rw'):
        fitness = fitness_function(cost, population_size)
    elif (selection == 'rb'):
        rank = [[i] for i in range(1, len(cost) + 1)]
        fitness = fitness_function(rank, population_size)
    elite_ind = elite_distance(population[0], distance_matrix_local, route=route)
    cost = copy.deepcopy(cost)
    elite_cst = copy.deepcopy(cost[0][0])
    solution = copy.deepcopy(population[0])
    print('Generation = ', count, ' Distance = ', elite_ind, ' f(x) = ', round(elite_cst, 2))

    # Définir le graphique initial
    plt.ion()  # Activer le mode interactif
    fig, ax = plt.subplots()
    line, = ax.plot([], [], 'r-', label='Coût Total')  # Initialiser une ligne vide
    ax.set_xlim(0, generations)
    ax.set_ylim(0, elite_cst*1.25)  # Estimation initiale de max_cost; ajustez selon vos données
    plt.title('Evolution of Total Cost over Generations')
    plt.xlabel('Generation')
    plt.ylabel('Total Cost')
    plt.legend()
    plt.show()

    elite_costs = []  # Stocker les coûts élites pour chaque génération
    generations_x = []  # Stocker les indices de génération pour l'axe x

    while (count <= generations - 1):
        offspring = breeding(cost, population, fitness, distance_matrix_local, n_depots, elite, max_capacity_volume,
                             max_capacity_weight, fixed_cost, variable_cost, penalty_value, parameters, route,
                             vehicle_types, fleet_size, crossover_rate=crossover_rate)
        offspring = mutation(offspring, mutation_rate=mutation_rate, elite=elite)
        cost, population = target_function(offspring, distance_matrix_local, parameters, fixed_cost, variable_cost,
                                           max_capacity_volume, max_capacity_weight, penalty_value, route=route,
                                           fleet_size=fleet_size)
        cost, population = (list(t) for t in zip(*sorted(zip(cost, population))))
        elite_child = elite_distance(population[0], distance_matrix_local, route=route)
        if (selection == 'rw'):
            fitness = fitness_function(cost, population_size)
        elif (selection == 'rb'):
            rank = [[i] for i in range(1, len(cost) + 1)]
            fitness = fitness_function(rank, population_size)
        if (elite_ind > elite_child):
            elite_ind = elite_child
            solution = copy.deepcopy(population[0])
            elite_cst = copy.deepcopy(cost[0][0])

        # Mise à jour des listes pour les générations et les coûts
        elite_costs.append(elite_cst)
        generations_x.append(count)
        ax.set_xlim(0, count*1.5)
        line.set_xdata(generations_x)
        line.set_ydata(elite_costs)
        ax.relim()
        ax.autoscale_view()
        fig.canvas.draw()
        fig.canvas.flush_events()

        count = count + 1
        print('Generation = ', count, ' Distance = ', elite_ind, ' f(x) = ', round(elite_cst, 2))

    plt.ioff()  # Désactiver le mode interactif
    plt.show()  # Afficher le graphique final


    if (graph == True):
        plot_tour_coordinates(coordinates, solution, n_depots=n_depots, route=route)

    if map == True:
        depot_indices = [i for i in range(n_depots)]
        draw_routes_on_map(coordinates, solution, depot_indices, output_file='final_solution_map.html')

        # Ouvrir la carte dans un navigateur web
        webbrowser.open_new_tab('final_solution_map.html')

    solution_report = show_report(solution, distance_matrix_local, parameters, fixed_cost, variable_cost,
                                  route=route)
    end = tm.time()
    print('Algorithm Time: ', round((end - start), 2), ' seconds')
    return solution_report, solution
```

Figure 3: Implementation of the Genetic Algorithm in Python

The genetic algorithm shown in the figure is designed to solve our HFVRP problem using the predefined functions shown in the table below :

Predifined function	Role
initial_population	Creates the initial set of solutions
target_function	Evaluates the fitness of each solution
fitness_function	Calculates fitness scores
breeding	Generates new offspring solutions
mutation	Introduces random changes to solutions
elite_distance	Identifies the best solution in a population
plot_tour_coordinates	Visualizes the route
draw_routes_on_map	Generates a map of the final solution
show_report	Produces a summary of the solution

Table 1: Predefined Functions and Their Roles

3.3 Comparison between the two methods :

To evaluate the performance of both the exact method (CPLEX) and the approximate method (Genetic Algorithm), we conducted extensive tests on various problem sizes. The instances were categorized into three classes: Small: $m \leq 5$, $n \leq 13$, $v \leq 6$, Medium: $3 \leq m \leq 5$, $25 \leq n \leq 35$, $10 \leq v \leq 13$ and Large: $3 \leq m \leq 5$, $50 \leq n \leq 60$, $15 \leq v \leq 20$, Where m is the number of depots, n is the number of clients, and v is the number of vehicle types.

Tests were conducted on a Windows system with an Intel Core i9-10900K CPU (3.7 GHz) and 32GB RAM, using only one CPU core for fair comparison, the following table shows one example test of each category of instants.

	Method	m	n	v	CPU-T (s)	Solution's quality
Small instances	Exact	2	8	3	80.10	Optimal
	Approximate	2	8	3	63.14	Optimal
Medium size instances	Exact	3	25	10	1521.59	Optimal
	Approximate	3	25	10	314.18	Feasible
Large size instances	Exact	3	50	15	+1600.00	/
	Approximate	3	50	15	1546.07	Feasible

Table 2: Computational Results for Exact and Approximate Methods

For small instances, both methods consistently found optimal solutions. The exact method generally outperformed the genetic algorithm in terms of computation time. For example, in a base case (m=2, n=8, v=3), CPLEX took 80.10 seconds while the genetic algorithm took 63.14 seconds. However, for medium-sized problems, the exact method began to show limitations. For a base case (m=3, n=25, v=10), CPLEX found the optimal solution in 1521.59 seconds, while the genetic algorithm provided a near-optimal solution in just 314.18 seconds. Finally, for large instances, the exact method consistently hit the time limit without providing solutions. The genetic algorithm, however, continued to produce near-optimal solutions. For instance, in a case with m=3, n=50, v=15, the genetic algorithm provided a near-optimal solution in 1546.07 seconds.

In conclusion, while the exact method provides guaranteed optimal solutions for small instances, the genetic algorithm demonstrates superior scalability and practical applicability for larger, more complex routing problems typical in real-world scenarios.

4. Vehicle Routing Solver (VRS) Interface

4.1 Software Architecture

Figure 4: Vehicle Routing Solver (VRS) Main Interface

The VRS is a comprehensive, multi-tabbed application designed for efficient transportation planning. It consists of seven main tabs, each with interactive controls tailored to specific functions. This structure allows users to manage all phases of the routing process from a single, user-friendly interface, making it a powerful tool for transport planning operations.

4.2 User Interaction and Functionalities

The VRS offers a range of functionalities across its various tabs:

Tab	Functionality
Clients Tab	• Data Import: Users can load client data from Excel files. • Data Management: Ability to add, modify, and delete client information directly through the graphical interface.
Warehouses Tab	• Data Import: Similar to the Clients tab, allows importing warehouse data. • Data Management: Direct addition, deletion, or modification of warehouse details.
Orders Tab	• Data Import and Management: Importing order data and manipulation (adding, deleting, modifying) of delivery orders.
Vehicles Tab	• Configuration: Definition and adjustment of vehicle properties such as type, capacities, and costs.
Solver Tab	• Route Configuration: - Open/Closed: Choice between an open route where the vehicle doesn't return to the starting point, or closed where it does.

	• Resolution Mode: - Auto: The system automatically determines the best algorithm parameters based on the data. - Manual: Users can adjust genetic algorithm parameters such as population size, crossover rate, etc. • Algorithm Execution: - "Start" button: Launches the algorithm execution with provided parameters and shows real-time progress. - Log Display: Real-time visualization of operation logs, helping diagnose problems or track progress.

Table 3: Tab's Functionalities

4.3 Visualization of results

The VRS provides comprehensive tools for visualizing and interpreting the optimization results :

Report : The Report tab displays the optimization results in a tabular format, showing detailed information about the routes for each vehicle.

Geographic Map :

Figure 5 : VRS Displayed Geographic map.

The Geographic Map tab offers a graphical display of the optimized routes on a map, which can be viewed in a web browser

Conclusion :

The study addressed the MDHFVRP for CPH Agriculture by comparing exact and approximate methods. The exact method using CPLEX proved optimal for small problems but unsuitable for large instances. The genetic algorithm demonstrated better scalability, providing near-optimal solutions for large problems within reasonable time frames.

A user interface, the Vehicle Routing Solver (VRS), was developed to integrate the genetic algorithm, allowing CPH Agriculture's logisticians to efficiently optimize their daily operations.

Future work will focus on enhancing the VRS with additional features such as map visualization, automatic method selection, and incorporation of more complex constraints, thus broadening its applicability to various real-world logistics scenarios.

Bibliography :

[1] Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. Management Science, 6(1), 80-91.

[2] Bettinelli, A., Ceselli, A., & Righini, G. (2011). A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, *19*(5), 723-740.

[3] Toth, P., & Vigo, D. (Eds.). (2014). Vehicle routing: Problems, methods, and applications. Society for Industrial and Applied Mathematics.

[4] Salhi, S., Imran, A., & Wassan, N. A. (2014). The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. Computers & Operations Research, 52, 315-325.

[5] Bolanos, R., Escobar, J., & Echeverri, M. (2018). A metaheuristic algorithm for the multi-depot vehicle routing problem with heterogeneous fleet. *International Journal of Industrial Engineering Computations*, *9*(4), 461-478.

[6] Nucamendi-Guillén, S., Gómez Padilla, A., Olivares-Benitez, E., & Moreno-Vega, J. M. (2020). The multi-depot open location routing problem with a heterogeneous fixed fleet. *Expert Systems with Applications, 165*, 113846.

[7] https://www.upperinc.com/glossary/route-optimization/multi-depot-heterogeneous-fleet-vehicle-routing-problem-mdhfvrp/

[8] ILOG, Inc. (2008). AMPL CPLEX System Version 11.0 User's Guide: Standard (Command-line) Version Including CPLEX Dire

[9] https://www.geeksforgeeks.org/simple-genetic-algorithm-sga/

[10] Mitchell, M. (1996). An introduction to genetic algorithms mit press. *Cambridge, Massachusetts. London, England*, *1996*.

[11] Valdecy. (n.d.). *pyVRP* [Software]. GitHub. https://github.com/Valdecy/pyVRP/blob/master/src/pyVRP.py