Research practice I
Final report
Vehicle Routing Problem With Stochastic Demands

Juan Jose Castrillon and Juan Carlos Rivera
jjcastrilc@eafit.edu.co, jrivera6@eafit.edu.co

School of Applied Sciences and Engineering, Universidad EAFIT

November 12, 2024

Abstract

This paper addresses the Vehicle Routing Problem with Stochastic Demands (VRPSD), focusing on optimizing vehicle routes under demand uncertainty. The study involves the development and implementation of a comprehensive algorithm in Python that integrates a GRASP heuristic, a Split-S procedure, and a set partitioning model with Gurobi. The main findings demonstrate that a parallel simulation for all vehicles, including dynamic vehicle coordination, significantly improves solutions by reducing the total distance traveled. Additionally, this implementation offers a more realistic approach by considering finite fleet sizes and dynamic recourse actions. The results highlight the effectiveness of the proposed methods in managing stochastic demands, providing valuable insights for logistics and supply chain management.

Keywords: VRPSD, Vehicle Routing, Stochastic Demands, GRASP, Split-S, Set Partitioning.

1 Introduction

The Vehicle Routing Problem with Stochastic Demands (VRPSD) is a significant area of study within logistics and supply chain management. This problem extends the classical Vehicle Routing Problem (VRP) by incorporating the uncertainty of customer demands, which adds complexity to the optimization process. In VRPSD, the objective is to design optimal routes for a fleet of vehicles to service a set of customers with uncertain demands, aiming to minimize the expected total cost, including travel and recourse costs.

This problem can be formally defined on a complete, undirected graph $G = (V, E)$, where $V = \{0, 1, \ldots, n\}$ represents the vertices, with vertex 0 being the depot and vertices 1, ..., n representing the customers. Each customer $i \in V \setminus \{0\}$ has a stochastic demand ξ_i, which is a random variable with a known probability distribution. The total demand for a route must not exceed the vehicle capacity Q.

The VRPSD can be expressed as a two-stage stochastic programming problem. In the first stage, the routes are planned based on expected demands. In the second stage, the actual demands are realized, and the routes are adjusted accordingly. If a vehicle's capacity is exceeded during its route, a recourse action, typically returning to the depot to reload, is implemented. However, this paper explores the possibility of another vehicle assisting in completing the route if it has the capacity to do so.

This problem has numerous practical applications, including logistics, delivery services, and supply chain management, where demand uncertainty is a common challenge. Solving the VRPSD effectively can lead to significant cost savings and improved service levels. The motivation for studying VRPSD is driven by the need to develop robust and adaptive routing strategies that can handle demand uncertainties effectively.

The structure of this paper is organized as follows: Section 2 presents a detailed problem definition of the VRPSD. Section 3 reviews the state of the art in VRPSD research. Section 4 describes the solution methodologies and algorithms used in this study, including the GRASP algorithm, the Split-S procedure, and the set partitioning approach with Gurobi. Section 5 provides computational results and performance analysis. Finally, Section 6 concludes the paper with a summary of findings and potential directions for future research.

2 Problem Definition

The Vehicle Routing Problem with Stochastic Demands (VRPSD) involves designing optimal routes for a fleet of vehicles to service a set of customers with uncertain demands. This problem can be formally defined on a complete, undirected graph $G = (V, E)$, where $V = \{0, 1, \ldots, n\}$ represents the vertices, with vertex 0 being the depot and vertices $1, \ldots, n$ representing the customers. The edges $E = \{(i, j) : i, j \in V, i \neq j\}$ are associated with a distance or cost d_{ij} for traveling between vertices i and j.

Each customer $i \in V \setminus \{0\}$ has a stochastic demand ξ_i, which is a random variable with a known probability distribution. The total demand for a route must not exceed the vehicle capacity Q. The objective is to minimize the expected total cost of the routes, including the cost of traveling and any additional costs incurred due to demand variability, such as returning to the depot to reload.

Mathematically, the VRPSD can be expressed as a two-stage stochastic programming problem. In the first stage, the routes are planned based on expected demands. In the second stage, the actual demands are realized, and the routes are adjusted accordingly. If a vehicle's capacity is exceeded during its route, a recourse action, typically returning to the depot to reload, is implemented, but other options can be implemented as is the case of this paper.

Key elements of the problem include:

- **Graph Representation:** $G = (V, E)$ with vertices V and edges E.
- **Stochastic Demands:** Customer demands ξ_i are random variables.

- **Vehicle Capacity:** Each vehicle has a capacity Q.
- **Objective:** Minimize the expected total cost, including travel and recourse costs.
- **Recourse Action:** Returning to the depot to reload if capacity is exceeded or explore the possibility that another vehicle visits the following nodes on this route in which capacity is exceeded.

This problem has numerous practical applications, including logistics, delivery services, and supply chain management, where demand uncertainty is a common challenge. Solving the VRPSD effectively can lead to significant cost savings and improved service levels.

3 State of the Art

The vehicle routing problem with stochastic demand has been one of the most studied combinatorial optimization problems in the industry due to its wide implementation in different areas such as waste collection. Because of this, a wide variety of algorithms have been created, whether heuristic or not.

For example, Marinaki *et al.* (2023) discuss a novel approach by integrating a hybrid Dragonfly Algorithm with Combinatorial Expanding Neighborhood Topology to address VRPSD, showcasing its effectiveness through benchmarking against existing swarm intelligence algorithms. This method emphasizes the adaptability and efficiency of route optimization in dynamic environments like biological waste management.

Another article regarding this problem is Salavati-Khoshghalb *et al.* (2019) where authors present an innovative hybrid recourse policy for VRPSD, focusing on a rule-based approach that incorporates risk and distance considerations. This policy is designed to manage the uncertainty of customer demands more effectively, with computational experiments demonstrating its capability to solve complex routing problems efficiently.

The paper Mendoza & Villegas (2013) introduces a multi-space sampling heuristic for VRPSD, blending randomized heuristics, route partitioning procedures, and a set partitioning model to efficiently navigate the solution space. This innovative approach, validated through computational experiments, demonstrates competitive accuracy and efficiency, identifying new optimal solutions and proving effective across various instances. This addition further enriches the VRPSD discourse, showcasing the continuous development of sophisticated methodologies to tackle the complexities of stochastic demand in vehicle routing.

This papers contribute significantly to the VRPSD literature by introducing advanced algorithms and policies that enhance the decision-making process under uncertainty, offering valuable insights for applications in logistics, waste management, and beyond. These studies highlight the ongoing evolution in solving VRPSD, underscoring the importance of adaptability and comprehensive risk management in route optimization strategies.

4 Solution Method / Methodology

In this section, we describe the methods, techniques, and algorithms developed during the research practice to address the Vehicle Routing Problem with Stochastic Demands (VRPSD).

The methodology employed in this study involves the development and implementation of a comprehensive algorithm in Python, which integrates several advanced techniques to optimize vehicle routing under demand uncertainty. The approach is structured as follows:

4.1 Heuristic GRASP Algorithm

For the implementation of the algorithm used to solve the VRSPD,,the initial step involves the implementation of a Greedy Randomized Adaptive Search Procedure (GRASP) heuristic algorithm. The primary objective of this algorithm is to minimize the total distance traveled by all vehicles while ensuring that all nodes are visited, starting and ending at the depot. The GRASP algorithm generates an initial TSP that will be implemented in the s-split.

The Greedy Randomized Adaptive Search Procedure (GRASP) is a multi-start or iterative process metaheuristic for solving combinatorial and optimization problems. Each iteration of GRASP consists of two phases: a construction phase and a local search phase. The algorithm's objective is to find high-quality solutions by exploring various regions of the solution space.

In the construction phase, a feasible solution is incrementally built, one element at a time, using a greedy randomized approach. The process begins with an empty solution and iteratively adds elements based on a greedy function, which evaluates the potential benefit of including each candidate element. The inclusion of elements is done probabilistically, where a restricted candidate list (RCL) of the best candidates is generated, and one element is randomly selected from this list to be added to the solution. This randomness helps in diversifying the search and avoiding local optima.

Once a complete solution is constructed, the local search phase aims to improve it by exploring its neighborhood. This phase involves iteratively applying local modifications to the solution, such as swapping, adding, or removing elements, to find a better solution within the local neighborhood. The local search continues until no further improvement can be found, indicating that a local optimum has been reached.

The GRASP algorithm repeats the construction and local search phases multiple times, each time starting from a different randomly generated initial solution. By doing so, it explores different regions of the solution space, increasing the chances of finding a global optimum or a high-quality solution. The best solution found over all iterations is returned as the final solution.

GRASP is particularly effective for large and complex problems due to its simplicity, flexibility, and ability to escape local optima through randomized construction. It can be easily parallelized, making it suitable for high-performance computing environments. Additionally, GRASP's iterative nature allows it to balance exploration and exploitation, providing a robust approach to solving combinatorial optimization problems.

4.2 TSP with Split-S Procedure

Following the GRASP algorithm, the next step involves implementing a Split-S procedure on the generated Traveling Salesman Problem (TSP) solutions. This procedure splits the TSP route into sub-routes based on the fixed number of available vehicles in the fleet. The Split-S procedure ensures that each sub-route does not exceed the vehicle capacity and that all nodes are visited.

The Split-S algorithm for a fixed fleet is designed to partition a Traveling Salesman Problem (TSP) solution into feasible routes for multiple vehicles, ensuring that no vehicle exceeds its capacity. Here, we provide a detailed explanation of the algorithm's steps based on the provided code:

The algorithm begins by initializing several matrices:

- V is a matrix of dimensions $p \times (n+1)$, initialized to infinity, where p is the number of sublists (vehicles), and n is the number of nodes.
- W is a matrix of dimensions $p \times (n+2)$, also initialized to infinity, which will store intermediate route costs.
- P is a three-dimensional matrix of dimensions $p \times m \times (n+1)$, initialized to zero, where m is the number of main lists (routes).

The matrices V and W are used to keep track of the minimum cost of routes, and P stores the predecessors for constructing the routes.

For each route k in the main list:

1. For each node i from 1 to $n+1$:
    - Initialize the current load to zero.
    - For each node j from i to n, add the demand $q[j]$ to the current load.
    - If the current load exceeds the vehicle capacity Q, break the loop.
    - Calculate the costs of three potential routes:
        - $Ruta1$ starting at node i.
        - $Ruta2$ starting at node $i+1$.
        - $Ruta3$ starting at node $i+2$.
    - If the current load does not exceed Q and the new route cost is less than the current cost in V, update V and P.
2. If no feasible solution is found for a given i, break the loop.
3. Update W with the values of V for the next iteration.

After building the cost matrix V, the algorithm reconstructs the routes:

1. Initialize an empty list of routes.
2. For each vehicle t:

- Determine the endpoint of the route by checking the predecessor matrix P.
- Trace back the route from the endpoint to the start using P, appending each node to the route list.

The algorithm returns the cost matrix V, the predecessor matrix P, and the list of routes. The pseudocode of this algorithm is presented at the end of the document.

4.3 Set Partitioning with Gurobi

After simulating the Split-S procedure multiple times and storing all feasible solutions, we employ a set partitioning model to optimize the combination of routes. The set partitioning problem is solved using the Gurobi optimizer, which aims to minimize the total distance traveled by the fleet. The model selects the best combination of sub-routes that collectively cover all nodes while minimizing the overall travel distance.

4.4 Simulation of Vehicle Capacity Constraints

In the final step for the simulation of the vehicles' route, a normal distribution was implemented to simulate the demand of each node, the vehicles know the average of the demands of each node before reaching it, but the true demand is only known when The vehicle arrives at the node and in addition two algorithms were implemented to evaluate the robustness of the proposed routing solutions under vehicle capacity constraints:

- **Return-to-Depot Simulation:** In this simulation, if a vehicle exceeds its capacity at any point during its route, it returns to the depot to reload before continuing with the route. This scenario models a conservative approach where vehicles strictly adhere to their capacity limits.
- **Nearby Vehicle Assistance Simulation:** In this simulation, if a vehicle exceeds its capacity, a nearby vehicle with available capacity is dispatched to complete the remaining deliveries. This approach evaluates the potential benefits of dynamic vehicle coordination and cooperation in real-time to handle capacity overflows efficiently.

These simulations are critical for comparing the efficiency of allowing a vehicle to return to the depot versus utilizing nearby vehicles to assist in completing the route. By analyzing the results from both simulations, we can determine the most effective strategy for managing vehicle capacities under stochastic demand conditions.

5 Results

In this section, we present the results obtained from the implementation of the VRPSD algorithm. Specifically, we focus on the routes of the first two vehicles from instances 1 and 3. Additionally, we provide the total distance traveled by each vehicle and summarize the results of both simulation approaches.

5.1 Graphical Representation of Routes

The following figures illustrate the routes of the first two vehicles from instances 1 and 2. Each figure includes the graph of the routes and the total distance traveled by the vehicles.

5.1.1 Instance 1

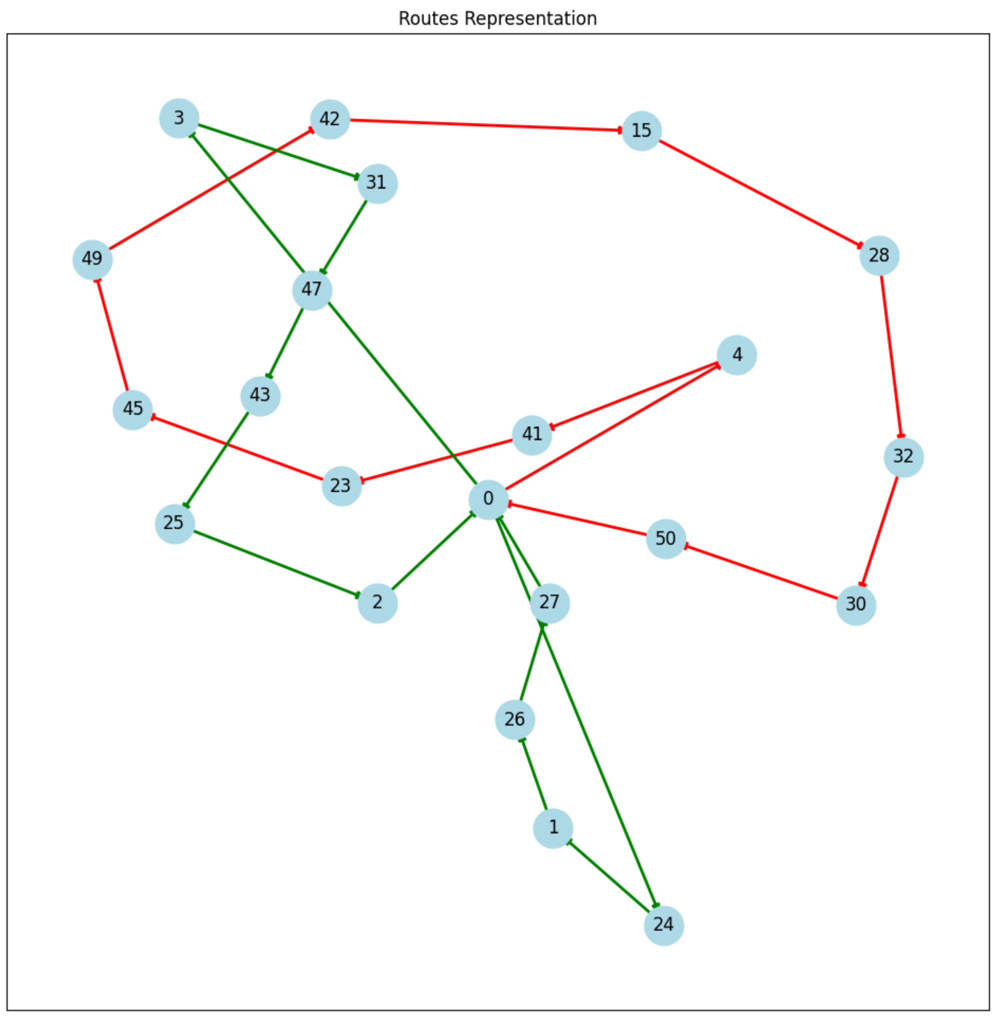
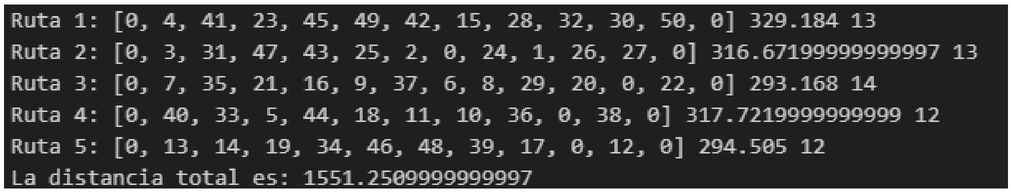


Figure 1: Graphic representation of the route of the first two vehicles of instance 1 with return to depot simulation.



```
Ruta 1: [0, 4, 41, 23, 45, 49, 42, 15, 28, 32, 30, 50, 0] 329.184 13
Ruta 2: [0, 3, 31, 47, 43, 25, 2, 0, 24, 1, 26, 27, 0] 316.67199999999997 13
Ruta 3: [0, 7, 35, 21, 16, 9, 37, 6, 8, 29, 20, 0, 22, 0] 293.168 14
Ruta 4: [0, 40, 33, 5, 44, 18, 11, 10, 36, 0, 38, 0] 317.7219999999999 12
Ruta 5: [0, 13, 14, 19, 34, 46, 48, 39, 17, 0, 12, 0] 294.505 12
La distancia total es: 1551.2509999999997
```

Figure 2: Routes and total route of instance 1 with return to depot simulation.

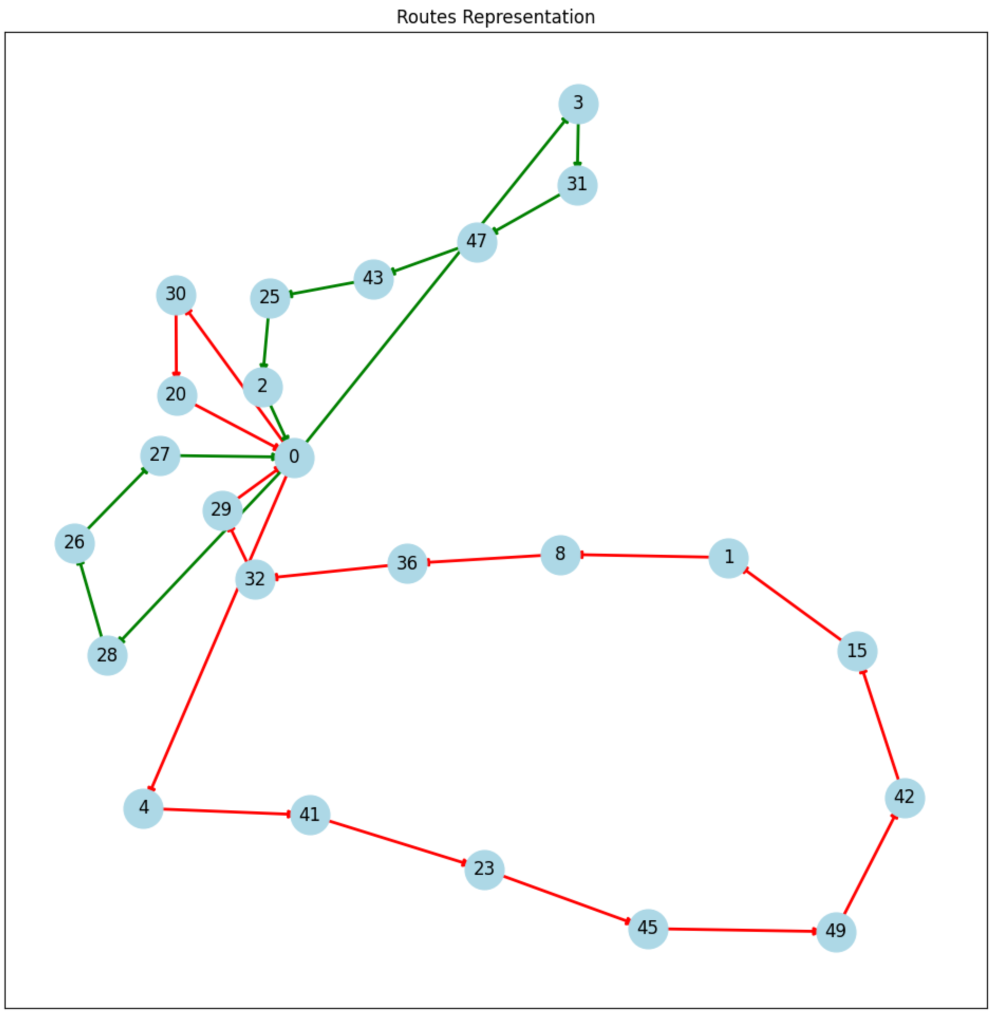
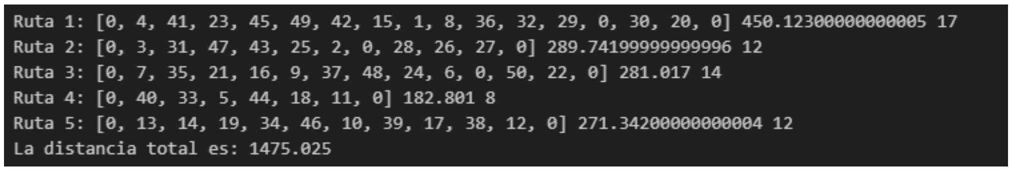


Figure 3: Graphic representation of the route of the first two vehicles of instance 1 with nearby vehicle assitance.



```
Ruta 1: [0, 4, 41, 23, 45, 49, 42, 15, 1, 8, 36, 32, 29, 0, 30, 20, 0] 450.12300000000005 17
Ruta 2: [0, 3, 31, 47, 43, 25, 2, 0, 28, 26, 27, 0] 289.74199999999996 12
Ruta 3: [0, 7, 35, 21, 16, 9, 37, 48, 24, 6, 0, 50, 22, 0] 281.017 14
Ruta 4: [0, 40, 33, 5, 44, 18, 11, 0] 182.801 8
Ruta 5: [0, 13, 14, 19, 34, 46, 10, 39, 17, 38, 12, 0] 271.34200000000004 12
La distancia total es: 1475.025
```

Figure 4: Routes and total route of instance 1 with nearby vehicle assitance.

5.1.2 Instance 2

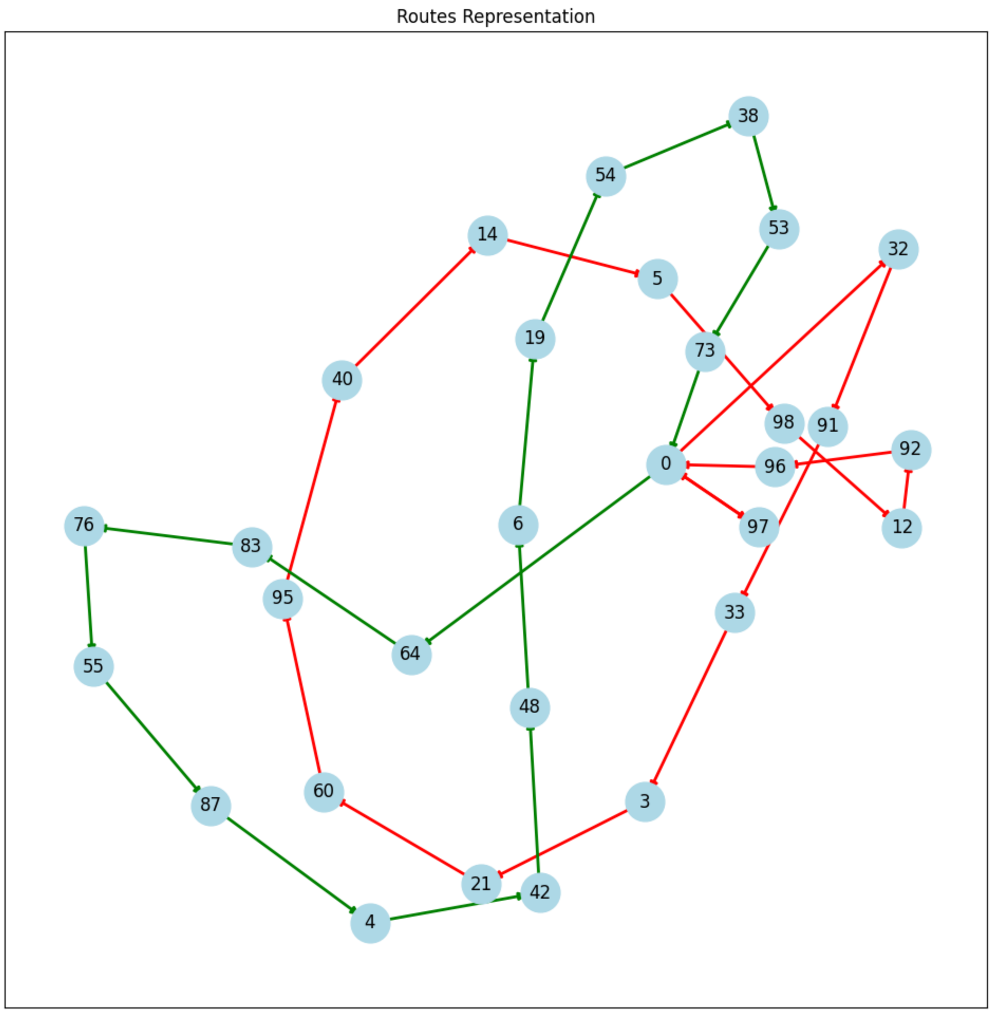
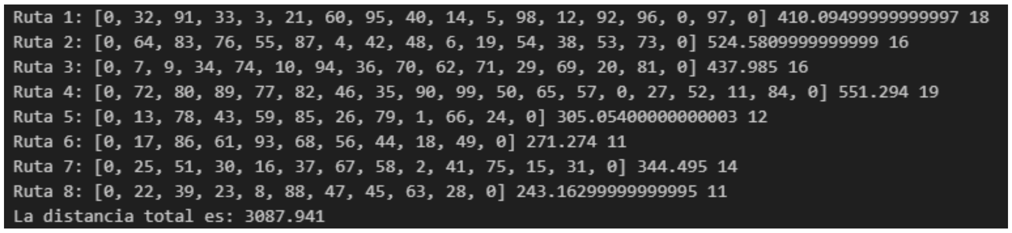


Figure 5: Graphic representation of the route of the first two vehicles of instance 2 with return to depot simulation.



```
Ruta 1: [0, 32, 91, 33, 3, 21, 60, 95, 40, 14, 5, 98, 12, 92, 96, 0, 97, 0] 410.09499999999997 18
Ruta 2: [0, 64, 83, 76, 55, 87, 4, 42, 48, 6, 19, 54, 38, 53, 73, 0] 524.5809999999999 16
Ruta 3: [0, 7, 9, 34, 74, 10, 94, 36, 70, 62, 71, 29, 69, 20, 81, 0] 437.985 16
Ruta 4: [0, 72, 80, 89, 77, 82, 46, 35, 90, 99, 50, 65, 57, 0, 27, 52, 11, 84, 0] 551.294 19
Ruta 5: [0, 13, 78, 43, 59, 85, 26, 79, 1, 66, 24, 0] 305.05400000000003 12
Ruta 6: [0, 17, 86, 61, 93, 68, 56, 44, 18, 49, 0] 271.274 11
Ruta 7: [0, 25, 51, 30, 16, 37, 67, 58, 2, 41, 75, 15, 31, 0] 344.495 14
Ruta 8: [0, 22, 39, 23, 8, 88, 47, 45, 63, 28, 0] 243.16299999999995 11
La distancia total es: 3087.941
```

Figure 6: Routes and total route of instance 2 with return to depot simulation.

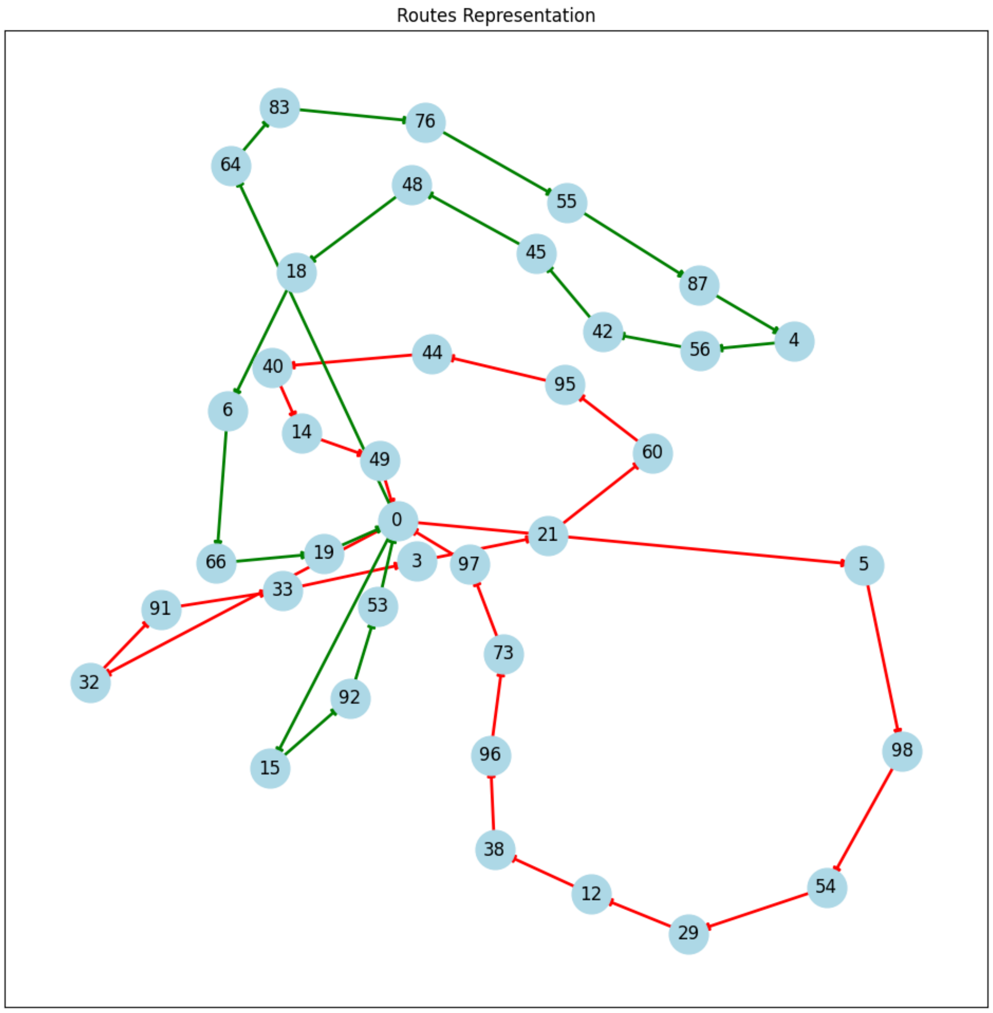
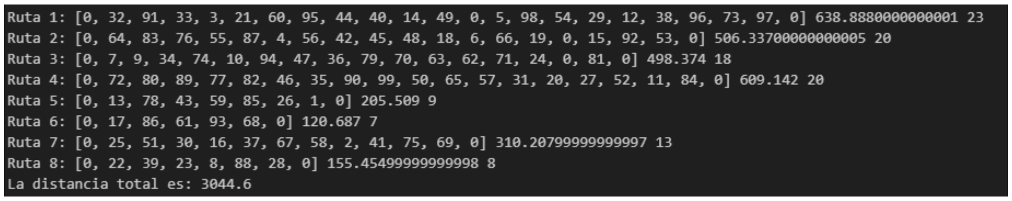


Figure 7: Graphic representation of the route of the first two vehicles of instance 2 with nearby vehicle assitance.



```
Ruta 1: [0, 32, 91, 33, 3, 21, 60, 95, 44, 40, 14, 49, 0, 5, 98, 54, 29, 12, 38, 96, 73, 97, 0] 638.8880000000001 23
Ruta 2: [0, 64, 83, 76, 55, 87, 4, 56, 42, 45, 48, 18, 6, 66, 19, 0, 15, 92, 53, 0] 506.33700000000005 20
Ruta 3: [0, 7, 9, 34, 74, 10, 94, 47, 36, 79, 70, 63, 62, 71, 24, 0, 81, 0] 498.374 18
Ruta 4: [0, 72, 80, 89, 77, 82, 46, 35, 90, 99, 50, 65, 57, 31, 20, 27, 52, 11, 84, 0] 609.142 20
Ruta 5: [0, 13, 78, 43, 59, 85, 26, 1, 0] 205.509 9
Ruta 6: [0, 17, 86, 61, 93, 68, 0] 120.687 7
Ruta 7: [0, 25, 51, 30, 16, 37, 67, 58, 2, 41, 75, 69, 0] 310.20799999999997 13
Ruta 8: [0, 22, 39, 23, 8, 88, 28, 0] 155.45499999999998 8
La distancia total es: 3044.6
```

Figure 8: Routes and total route of instance 2 with nearby vehicle assitance.

5.2 Summary of Simulation Results

The table below summarizes the total distance traveled for three instances using both simulation approaches.

Table 1: Summary of Total Distance Traveled in Three Instances

Instance	**Return to Depot (km)**	**Nearby Vehicle Assistance (km)**
Instance 1	1551.25	1475.02
Instance 2	3087.94	3044.6
Instance 3	3910	3777

6 Conclusions and Future Research

The main conclusions of this project highlight that the implementation of a parallel simulation for all vehicles significantly improves the solutions. Specifically, the option for nearby vehicles to complete a route that another vehicle cannot finish due to capacity constraints leads to a reduction in the total distance traveled by all vehicles. This approach demonstrates the effectiveness of dynamic vehicle coordination in optimizing routes under stochastic demands.

As mentioned in the introduction, the goal was to compare the implementation of the VRPSD algorithm with the solutions proposed by Villegas and Mendoza. Although the exact same instances were not used, it is considered that a more comprehensive solution to this problem was implemented. Unlike Villegas and Mendoza, who employed a set partitioning and a split-S approach, this implementation also included a finite fleet solution, which is more reflective of real-world scenarios. Additionally, instead of only implementing a return to the depot when a vehicle exceeds its capacity, as done by Villegas and Mendoza, this project also integrated route interleaving for nearby routes in cases where this option results in a shorter total distance traveled compared to directly returning to the depot.

The contributions of this project are substantial, demonstrating that incorporating more realistic constraints and dynamic solutions can lead to better optimization outcomes. The parallel simulation approach and the consideration of route interleaving are particularly noteworthy, offering practical insights for real-world applications.

For future research, it is recommended to explore the following directions:

- Investigating the impact of different demand distributions on the performance of the algorithm.

- Extending the approach to consider additional real-world constraints, such as time windows and varying vehicle speeds.
- Comparing the performance of the algorithm with other state-of-the-art methods using a standardized set of instances.
- Exploring the potential of machine learning techniques to predict demand patterns and further optimize the routing decisions.

Overall, this project has successfully demonstrated a robust and adaptable solution to the VRPSD, providing a strong foundation for further advancements in the field of vehicle routing under uncertainty.

Pseudocodes

Split-S

```
Algorithm: Split-S Algorithm for Fixed Fleet

1.  Initialize V and W with dimensions p x (n+1) and p x (n+2) respectively, filled with infini
2.  Initialize P with dimensions p x m x (n+1), filled with zeros
3.  Set V[i][0] and W[i][0] to 0 for i from 0 to p-1
4.  Set W[i][-1] to 0 for i from 0 to p-1
5.  for k from 0 to m-1 do
6.      for i from 1 to n+1 do
7.          j = i
8.          load = 0
9.          while j <= n and load <= Q do
10.             load += q[j]
11.             if i == j then
12.                 Ruta1 = [0, i, 0]
13.                 Ruta2 = [0, i+1, 0]
14.                 Ruta3 = [0, i+2, 0]
15.                 for i <- 1 to m do calculate cost{i} end for
18.             else if j == i+1 then
19.                 Ruta1.insert(2, j)
20.                 Ruta2.insert(2, j-1)
21.                 Ruta3.insert(2, j-1)
22.                 for i <- 1 to m do calculate cost{i} end for
25.             else if j == i+2 then
26.                 Ruta1.insert(3, j)
27.                 Ruta2.insert(2, j)
28.                 Ruta3.insert(3, j-1)
29.                 for i <- 1 to m do calculate cost{i} end for
32.             else
33.                 if cost2 <= cost3 then
34.                     pos = Ruta2.index(max(Ruta2))
```

```
35.                     Ruta2.insert(pos+1, j)
36.                 else
37.                     pos = Ruta2.index(max(Ruta2))
38.                     Ruta2.insert(pos+1, j)
39.                 Ruta1.insert(-1, j)
40.                 Ruta3.insert(1, j)
41.                 for i <- 1 to m do calculate cost{i} end for
46.             if load <= Q and (W[0][i-1] + cost1 < V[0][j]) then
47.                 V[0][j] = W[0][i-1] + cost1
48.                 P[0][k][j] = i - 1
49.             if load <= Q and (W[1][i-1] + cost2 < V[1][j]) then
50.                 V[1][j] = W[1][i-1] + cost2
51.                 P[1][k][j] = i - 1
52.             if load <= Q and (W[2][i-1] + cost3 < V[2][j]) then
53.                 V[2][j] = W[2][i-1] + cost3
54.                 P[2][k][j] = i - 1
55.             j += 1
56.         if W[0][i] == float('inf') then break
57.     w <- v
58. Initialize routes as an empty list of lists with p sublists
59. for t from 0 to p-1 do
60.     j = n-1
61.     k = m-1
62.     while k >= 0 do
63.         i = P[t][k][j]
64.         routes[t].append(i)
65.         j = i
66.         k -= 1
67. return V, P, routes
```

References

Marinaki, Magdalene, Taxidou, Andromachi, & Marinakis, Yannis. 2023. A hybrid Dragonfly algorithm for the vehicle routing problem with stochastic demands. *Intelligent Systems with Applications*, **18**, 200225.

Mendoza, Jorge E., & Villegas, Juan G. 2013. Amulti-space sampling heuristic for the vehicle routing problem with stochastic demands. *Optimization Letters*, **7**.

Salavati-Khoshghalb, Majid, Gendreau, Michel, Jabali, Ola, & Rei, Walter. 2019. A hybrid recourse policy for the vehicle routing problem with stochastic demands. *EURO Journal on Transportation and Logistics*, **8**(3), 269–298.