



**Universidad Tecnológica de la Habana “José Antonio Echeverría”**

**Facultad de Ingeniería Informática**

**Nueva versión de la Biblioteca de Heurísticas de Asignación para Problemas de  
Rutas de Vehículos con Múltiples Depósitos**

**Informe de la Práctica Profesional II**

**Autor:**

Eric Ramos Aragón ([ceramos@ceis.cujae.edu.cu](mailto:ceramos@ceis.cujae.edu.cu))

**Tutores:**

Dr. C. Isis Torres Pérez ([itorres@ceis.cujae.edu.cu](mailto:itorres@ceis.cujae.edu.cu))

Dr. C. Alejandro Rosete Suárez ([rosete@ceis.cujae.edu.cu](mailto:rosete@ceis.cujae.edu.cu))

**La Habana, Cuba**

17 de enero del 2025

## Resumen

El Problema de Planificación de Rutas de Vehículos con Múltiples Depósitos es un problema de optimización combinatoria que busca minimizar los costos asociados a la distancia o al tiempo de las rutas, cumpliendo con restricciones como la capacidad de los vehículos y los depósitos. Este problema se resuelve típicamente en dos etapas: primero la asignación de clientes a depósitos y a continuación la planificación de las rutas. En este contexto, la Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos es de gran utilidad pues es un componente de software diseñado para brindar diversos algoritmos que abordan la primera fase del problema.

El análisis de la arquitectura actual de esta biblioteca revela varias deficiencias que limitan su rendimiento y capacidad de integración. Estas incluyen un alto acoplamiento entre paquetes que dificulta su flexibilidad, el uso de cálculos de distancia aproximadas que no se adaptan a contextos reales, y una interoperabilidad restringida con sistemas externos debido a su implementación exclusiva en un único lenguaje de programación. Para abordar estas limitaciones, se rediseña la arquitectura de la biblioteca en Java para mejorar la distribución de responsabilidades entre los paquetes y reducir el acoplamiento. Además, se incorpora el uso de distancias reales y se desarrolla una versión del componente en Python, aprovechando las capacidades de este lenguaje para incrementar la interoperabilidad con sistemas externos y ampliar las posibilidades de integración en entornos tecnológicos modernos. Estas mejoras se validan mediante pruebas diseñadas para evaluar el comportamiento de la biblioteca, incluyendo pruebas de funcionalidad y comparaciones entre las nuevas versiones y la original.

**Palabras clave:** Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos, diseño de software, patrones de diseño, heurísticas.

## Abstract

The Multi-Depot Vehicle Routing Problem is a combinatorial optimization problem that aims to minimize costs associated with distance or time while satisfying constraints such as vehicle and depot capacities. This problem is typically solved in two stages: first, the assignment of customers to depots, followed by route planning. In this context, the Assignment Heuristic Library for Multi-Depot Vehicle Routing Problems proves highly useful as a software component providing various algorithms to address the first stage of the problem.

An analysis of the current architecture of this library reveals several shortcomings that limit its performance and integration capabilities. These include high coupling between packages that hinders flexibility, the use of approximate distance calculations that do not align with real-world contexts, and restricted interoperability with external systems due to its exclusive implementation in a single programming language. To address these limitations, the library's architecture is redesigned in Java to improve the distribution of responsibilities among packages and reduce coupling. Additionally, real distance calculations are incorporated, and a Python version of the component is developed, leveraging the capabilities of this language to enhance interoperability with external systems and expand integration possibilities in modern technological environments. These improvements are validated through tests designed to evaluate the library's performance, including functionality tests and comparisons between the new and the original versions.

**Keywords:** Assignment Heuristic Library for Multi-Depot Vehicle Routing Problems, software design, design patterns, heuristics.

**ANEXO A. PLAN INDIVIDUAL DE TAREAS DEL ESTUDIANTE**

<b>Tareas</b>	<b>PP/T</b>	<b>Fecha de entrega</b>	<b>Rol(es) que desarrolla(n) con la tarea</b>
Reunión de inicio de la Práctica Profesional II.	PP	14/10/24	JP
Estudiar el Problema de Planificación con Múltiples Depósitos, sus características, variantes y formas de resolución haciendo énfasis en la fase de asignación de clientes a depósitos.	PP	14/10/24	AN
Análisis de la arquitectura y de las funcionalidades brindadas por BHAVRP para identificar deficiencias que presenta el código de BHAVRP.	PP	16/10/24	AR, AS
Investigar si existen bibliotecas en Java u otros lenguajes de programación que implementen algoritmos de asignación como en BHAVRP y que estén disponibles.	PP	21/10/24	AS
Identificar mejoras para la arquitectura de BHAVRP.	PP	24/10/24	AR, DS
Estudiar patrones y principios de diseño para identificar aquellos que pueden incorporarse para solventar las deficiencias encontradas y mejorar el diseño arquitectónico del componente.	PP	27/10/24	AR, DS
Asimilación de Python como lenguaje de programación.	PP	01/11/24	AN
Redactar la Introducción, el Capítulo 1, las Conclusiones y Recomendaciones derivadas del estudio del estado del arte del Informe de la Práctica Profesional II.	PP	04/11/24	EE
Rediseñar de ser necesario las clases para modelar un problema de planificación de rutas de vehículos con múltiples depósitos.	PP	05/11/24	DS, PG
Implementar en Python las clases necesarias para modelar un problema de planificación de rutas de vehículos con múltiples depósitos.	PP	07/11/24	DS, PG
Rediseñar de ser necesario las clases para modelar el resultado de la asignación de clientes a depósitos en un problema de planificación de rutas de vehículos con múltiples depósitos.	PP	08/11/24	DS, PG
Implementar en Python las clases necesarias para modelar el resultado de la asignación de clientes a depósitos en un problema de planificación de rutas de vehículos con múltiples depósitos.	PP	10/11/24	DS, PG
Rediseñar de ser necesario las clases que implementan el patrón Factory Method para la creación de algoritmos de asignación, cálculo de distancias y métricas de evaluación.	PP	11/11/24	DS, PG

Implementar en Python el patrón <i>Factory Method</i> para la creación de algoritmos de asignación, cálculo de distancias y métricas de evaluación.	PP	13/11/24	DS, PG
Incorporar en ambas bibliotecas BHAVRP el cálculo de distancia real mediante el uso del API OSRM.	PP	18/11/24	PG
Redactar el Capítulo 2 y las Conclusiones y Recomendaciones derivadas de la propuesta de solución del Informe de la Práctica Profesional II.	PP	11/12/24	EE
Diseñar los experimentos necesarios para validar el correcto funcionamiento de las dos versiones de la biblioteca.	PP	20/12/24	PB
Ejecutar los experimentos diseñados para validar el correcto funcionamiento de las nuevas versiones de la biblioteca.	PP	20/12/24	PB
Analizar los resultados obtenidos de los experimentos para identificar fortalezas, áreas de mejora y posibles ajustes en las nuevas versiones de la biblioteca.	PP	22/12/24	PB
Redactar el Capítulo 3 y las Conclusiones y Recomendaciones derivadas de la validación realizada a la solución implementada del Informe de la Práctica Profesional II.	PP	23/12/24	EE
Entregar el informe final de la Práctica Profesionales II.	PP	17/01/24	EE
Elaborar la presentación para la defensa de la Práctica Profesional II.	PP	17/01/24	EE
Defensa de la Práctica Profesionales II.	PP	20 – 22/01/24	EE

Isis Torres Pérez  
Nombre completo y firma del primer tutor

Alejandro Rosete Suárez  
Nombre completo y firma del segundo tutor

Eric Ramos Aragón  
Nombre completo del estudiante

Firma del estudiante

En la columna Rol se deben poner la sigla del rol que contribuye a formar cada tarea. Las siglas de los roles son:

<b>AN</b> - Analista de negocio	<b>AR</b> - Arquitecto	<b>JP</b> - Jefe de proyecto
<b>AS</b> - Analista de sistema	<b>DS</b> - Diseñador de software	<b>ES</b> - Especialista de seguridad
<b>AD</b> - Analista de datos	<b>DI</b> - Diseñador de Interfaz hombre – máquina	<b>EE</b> - Escritor-expositor de trabajos técnicos
<b>PG</b> - Programador	<b>DB</b> - Diseñador de base de datos	<b>II</b> - Habilitador de Infraestructuras Informáticas
<b>PB</b> - Probador	<b>GC</b> - Gestor de Configuración	<b>TD</b> - Facilitador de la Toma de Decisiones

# Índice

Introducción	1
Capítulo 1: Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos	10
1.1. Introducción	10
1.2. Especificaciones de BHAVRP	10
1.2.1. Marco teórico	13
1.2.2. Diseño arquitectónico de BHAVRP	19
1.2.3. Principios de diseño implementados en BHAVRP	25
1.2.4. Patrones de diseño implementados en BHAVRP	30
1.3. Bibliotecas que resuelven la fase de asignación del MDVRP	36
1.4. Bibliotecas especializadas en el cálculo de distancias	42
1.5. Limitaciones y deficiencias que presenta la versión actual de BHAVRP	45
1.6. Conclusiones parciales	48
Capítulo 2: Diseño de la nueva versión de BHAVRP	49
2.1. Introducción	49
2.2. Modificaciones en la arquitectura de BHAVRP	49
2.2.1. Nuevo diseño arquitectónico de BHAVRP	52
2.2.2. Paquetes y funcionalidades generales	53
2.2.3. Particularidades de BHAVRP en Java y Python	71
2.3. Principios de diseño adoptados en la solución	72
2.4. Patrones de diseño adoptados en la solución	75
2.4.1. Patrón <i>Template Method</i>	75
2.5. Conclusiones parciales	79

Capítulo 3: Análisis y validación de BHAVRP	80
3.1.    Introducción	80
3.2.    Experimento para validar del proceso de carga de datos	81
3.2.1.    Descripción de la instancia	81
3.2.2.    Análisis de los resultados del proceso de carga de datos	81
3.3.    Experimento para validar el proceso de creación de la matriz de costos	83
3.3.1.    Descripción de la instancia	84
3.3.2.    Análisis de los resultados del proceso de creación de la matriz de costos	84
3.4.    Experimento para validar el proceso de asignación de los algoritmos	89
3.4.1.    Análisis de los resultados de los algoritmos deterministas	91
3.4.2.    Análisis de los resultados de los algoritmos no deterministas	108
3.5.    Conclusiones parciales	109
Conclusiones	111
Recomendaciones	112

## Índice de figuras

Figura 1: Diagrama de actividades de BHAVRP.....	11
Figura 2: Heurísticas implementadas en BHAVRP.....	13
Figura 3: Variantes del VRP [46] .....	15
Figura 4: Métodos de optimización combinatoria [48]. .....	17
Figura 5: Arquitectura de BHAVRP [24].....	19
Figura 6: Diagrama de clases del paquete problem.....	21
Figura 7: Diagrama de clases del paquete classical.....	22
Figura 8: Diagrama de clases del paquete clustering.....	23
Figura 9: Diagrama de clases del paquete output.....	24
Figura 10: Diagrama de clases del paquete controller. ....	24
Figura 11: Diagrama de clases del paquete factory.....	25
Figura 12: Patrón Singleton implementado en BHAVRP. ....	31
Figura 13: Patrón Factory Method implementado en BHAVRP. ....	33
Figura 14: Arquitectura de la nueva versión de BHAVRP .....	53
Figura 15: Diagrama de clases de los paquetes controller y tools en la nueva versión.	54
Figura 16: Diagrama de clases del paquete problem en la nueva versión.....	56
Figura 17: Diagrama de clases del paquete output en la nueva versión.....	58
Figura 18: Diagrama de clases del paquete classical en la nueva versión. ....	61
Figura 19: Diagrama de clases del paquete clustering en la nueva versión.....	64
Figura 20: Diagrama de clases del paquete others en la nueva versión. ....	67
Figura 21: Diagrama de clases del paquete factory en la nueva versión. ....	69
Figura 22: Diagrama de actividades del algoritmo Farthest First [89]. ....	76
Figura 23: Diagrama genérico común para todos los algoritmos. ....	76
Figura 24: Ejemplo de la implementación del patrón Template Method [90]. ....	77

## Índice de tablas

Tabla 1: Principios de diseño SOLID que cumple BHAVRP.....	27
Tabla 2: Otros principios de diseño implementados en BHAVRP. ....	29
Tabla 3: Patrones de diseño GRASP implementados en BHAVRP. ....	35
Tabla 4: Comparación de otras bibliotecas que resuelven MDVRP.....	40
Tabla 5: Descripción de las clases que integran los paquetes controller y tools. ....	55
Tabla 6: Descripción de las clases que integran el paquete problem.....	57
Tabla 7: Descripción de las clases que integran el paquete output.....	58
Tabla 8: Descripción de las clases que integran el paquete classical (parte I).....	62
Tabla 9: Descripción de las clases que integran el paquete classical (parte II).....	63
Tabla 10: Descripción de las clases que integra el paquete clustering (parte I) .....	65
Tabla 11: Descripción de las clases que integra el paquete clustering (parte II) .....	66
Tabla 12: Descripción de las clases que integran el paquete others. ....	68
Tabla 13: Descripción de las clases que integran el paquete factory. ....	70
Tabla 14: Particularidades de BHAVRP en Java y Python. ....	72
Tabla 15: Descripción de la instancia de la literatura.....	81
Tabla 16: Resultados de la carga de datos en las tres versiones de BHAVRP.....	82
Tabla 17: Descripción de la instancia real. ....	84
Tabla 18: Resultados de la creación de la matriz de costos usando la distancia Euclidean. ....	85
Tabla 19: Resultados de la creación de la matriz de costos usando la distancia Haversine. ....	86
Tabla 20: Resultados de la creación de la matriz de costos usando la distancia Manhattan. ....	87
Tabla 21: Resultados de la creación de la matriz de costos usando la distancia Chebyshev.....	88
Tabla 22: Resultados de la creación de la matriz de costos usando la distancia Real..	89
Tabla 23: Clasificación de algoritmos en BHAVRP por tipo de comportamiento [23]. ...	90
Tabla 24: Resultados de la ejecución del algoritmo Best Cyclic Assignment. ....	92

Tabla 25: Resultados de la ejecución del algoritmo Best Nearest.....	93
Tabla 26: Resultados de la ejecución del algoritmo Clara.....	94
Tabla 27: Resultados de la ejecución del algoritmo Coefficient Propagation. ....	95
Tabla 28: Resultados de la ejecución del algoritmo Cyclic Assignment.....	96
Tabla 29: Resultados de la ejecución del algoritmo Farthest First. ....	97
Tabla 30: Resultados de la ejecución del algoritmo K-means. ....	98
Tabla 31: Resultados de la ejecución del algoritmo Nearest By Customer.....	99
Tabla 32: Resultados de la ejecución del algoritmo Nearest By Depot. ....	100
Tabla 33: Resultados de la ejecución del algoritmo PAM. ....	101
Tabla 34: Resultados de la ejecución del algoritmo Parallel.....	102
Tabla 35: Resultados de la ejecución del algoritmo Sequential Cyclic. ....	103
Tabla 36: Resultados de la ejecución del algoritmo Simplified. ....	104
Tabla 37: Resultados de la ejecución del algoritmo Sweep.....	105
Tabla 38: Resultados de la ejecución del algoritmo Three Criteria Clustering.....	106
Tabla 39: Resultados de la ejecución del algoritmo UPGMC. ....	107
Tabla 40: Resultados de la ejecución del algoritmo Random By Element.....	108
Tabla 41: Resultados de la ejecución del algoritmo Random Sequential Cyclic.....	109

## Introducción

Los Problemas de Planificación de Rutas de Vehículos (*Vehicle Routing Problem, VRP*) son problemas de optimización combinatoria que tiene como objetivo planificar un conjunto de rutas óptimas para una flota de vehículos que debe satisfacer las demandas de un conjunto de clientes dispersos geográficamente. Este problema presenta determinadas particularidades que lo convierten en una problemática de suma importancia siendo investigada por muchos científicos a lo largo de la historia. La primera vez que se publicó un artículo sobre este tema fue en el año 1959 por los investigadores G. Dantzig y J. Ramser [1]. Por la complejidad de este tipo de problemas, generalmente no es recomendable hacer uso de los métodos de optimización exactos o los métodos aproximados, pues no consiguen resolver estos problemas en tiempos de procesamiento de cómputo aceptables. Este tipo de problemas como muchos otros de optimización combinatoria se clasifican como **NP-duros** [2] [3].

Una de las variantes de **VRP** más estudiadas es el Problema de Planificación de Rutas de Vehículos con Múltiples Depósitos (*Multi Depot Vehicle Routing Problem, MDVRP*) [4] que constituye una extensión del **VRP** clásico (*Capacitated Vehicle Routing Problem, CVRP*) [5], donde cada vehículo comienza y termina su ruta en el mismo depósito y limita el tamaño de la flota de vehículos. La diferencia entre estos problemas radica en que la variante **MDVRP** incorpora varios depósitos con ubicaciones diferentes, y los clientes deben ser servidos con los vehículos pertenecientes al depósito que les fue asignado [6].

El **MDVRP** se encuentra presente en diversas actividades cotidianas. Ejemplo de estas aplicaciones prácticas son el transporte público, el transporte obrero y/o escolar, la distribución diaria de alimentos y bebidas hacia puntos de ventas, la transportación de leche y productos agrícolas hacia las plantas de procesamiento, entre otras actividades [7].

Debido a la complejidad de estos problemas, los investigadores han concluido que encontrar una solución óptima es extremadamente costosa en cuanto a tiempo. Por tanto, se recomienda el uso de algoritmos heurísticos (que incluye heurísticas y

metaheurísticas) como alternativa de solución. Dentro de los métodos heurísticos se emplean mayormente las heurísticas que trabajan en dos fases de acuerdo a la estrategia “Agrupar primero, planificar después” (*Cluster First – Route Second, CFRS*). Esta estrategia consiste en realizar la fase de asignación de clientes a depósitos para luego planificar las rutas que van a realizar los vehículos de cada depósito [8] [9].

Cuando se realiza la fase de asignación y los clientes quedan agrupados alrededor de los depósitos, el **MDVRP** puede ser modelado como varios **CVRP** independientes. Aunque usualmente los problemas presentan a los clientes y depósitos mezclados, por lo que se hace necesario resolver el **MDVRP** como un problema de agrupamiento, donde su resultado consiste en un conjunto de clientes asignados a depósitos específicos. Por lo tanto, se propone hallar solución a este problema en dos etapas, similar a como proponen las estrategias **CFRS**. La primera etapa donde los clientes deben ser asignados a los depósitos; y en la segunda etapa los clientes asignados al mismo depósito deben ser ordenados para su posterior visita con los vehículos pertenecientes a dicho depósito [9] [10].

Para cumplir con la primera etapa de esta estrategia existen diferentes algoritmos, también llamados heurísticas de asignación. Según [11] [12] [13], estas heurísticas se clasifican de la siguiente manera:

- Asignación a través de urgencias: los clientes con mayor urgencia o prioridad se asignan en primer lugar. En esta categoría se encuentran los algoritmos de Asignación en Paralelo, Asignación Simplificada y Asignación en Barrido.
- Asignación cíclica: consiste en asignar de forma cíclica un cliente a cada depósito. A esta categoría pertenece el algoritmo de igual nombre, Algoritmo de Asignación Cíclica.
- Asignación por clústeres: un depósito y el conjunto de clientes asignados a él conforman un clúster. A esta categoría pertenecen los algoritmos Coeficiente de Propagación y Agrupamiento por Tres Criterios.

Además de estas heurísticas de asignación, en la literatura existen diversos trabajos donde se utilizan algoritmos de agrupamiento entre los que destacan *K-means* y *K-medoids* [14] [15] [16].

Para facilitar el desarrollo y uso de estas técnicas, han surgido diversas bibliotecas y herramientas que brindan una integración eficaz de estos algoritmos en sistemas complejos. Bibliotecas como **Google OR-Tools** [17] y **VROOM** [18] ofrecen soluciones especializadas para el **MDVRP**, resolviendo la fase de asignación mediante el uso de técnicas de optimización combinatoria y algoritmos de agrupamiento, como *K-means* y el Algoritmo de Barrido (*Sweep algorithm*) [19]. **Google OR-Tools** aprovecha modelos de programación lineal entera y métodos de búsqueda local para encontrar soluciones que respeten restricciones como la capacidad de los vehículos y la distancia máxima de recorrido. Sin embargo, **VROOM** se enfoca en resolver problemas de planificación de rutas de vehículos mediante algoritmos de optimización que se pueden personalizar para ajustarse a diversas restricciones de asignación, como capacidades máximas de carga por vehículo, tiempos de entrega específicos o ventanas de tiempo, prioridades de clientes, límites en la distancia total recorrida por vehículo, así como restricciones de costos asociados al uso de la flota.

Por otra parte, resulta relevante destacar que estas bibliotecas están implementadas en Python, un lenguaje de programación que ha ganado popularidad en el ámbito de la optimización debido a su versatilidad, la amplia disponibilidad de bibliotecas de código abierto y su facilidad de integración con otros sistemas. Python resuelve problemas de interoperabilidad, lo que lo convierte en una opción ideal para desarrollar soluciones personalizadas para los problemas de planificación de rutas de vehículos [17] [20].

En otros lenguajes como Java, también se han desarrollado herramientas orientadas a resolver la fase de asignación en problemas como el **MDVRP**. Herramientas como **JSSprit** [21] y **OptaPlanner** [22], ofrecen soporte para dividir el problema en etapas que incluyen la asignación inicial de clientes a depósitos y la planificación de rutas. Además, **JSSprit** permite configurar soluciones para **MDVRP**, mientras que **OptaPlanner** es una

plataforma de planificación flexible que permite configurar y optimizar la asignación inicial de clientes a depósitos en el **MDVRP**, ajustándose a las restricciones específicas del problema. Otro ejemplo de estos componentes es la Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos (**BHAVRP**) [23] [24].

**BHAVRP** es una propuesta realizada en el proyecto de Optimización y Metaheurísticas de la Facultad de Ingeniería Informática de la Universidad Tecnológica de La Habana “José Antonio Echeverría” (CUJAE), que contiene algoritmos de asignación clásicas y adaptaciones de métodos de agrupamiento para resolver la fase de asignación en la variante **MDVRP**. La variedad de estos algoritmos cubre un amplio rango de estrategias para resolver la tarea de asignación de clientes a depósitos. El comportamiento general de estos algoritmos consiste en determinar el mejor cliente para ser asignado al depósito verificando si las restricciones del problema se cumplen. Adicionalmente, la biblioteca cuenta con otros algoritmos creados por el equipo de desarrollo del proyecto, uno de estos algoritmos consiste en realizar una asignación totalmente al azar, mientras que los demás emplean criterios basados en la distancia para llevar a cabo el proceso de asignación.

Además de la ejecución de estos algoritmos, **BHAVRP** ofrece un conjunto de funcionalidades útiles. Entre ellas se encuentran la carga de ficheros de entrada, que permite utilizar datos en formatos estándar como *.data* o *.txt*, y la configuración de parámetros, que ofrece al usuario la posibilidad de ajustar las heurísticas según las necesidades específicas del problema. Asimismo, **BHAVRP** permite utilizar las siguientes medidas de distancia aproximadas: *Euclidean* [25], *Haversine* [26], *Manhattan* [27] y *Chebyshev* [28], lo cual proporciona flexibilidad adicional para aplicar la métrica más adecuada según el contexto de cada instancia del problema.

Sin embargo, **BHAVRP** actualmente no incorpora cálculos basados en distancias reales, lo que limita su nivel de aplicabilidad en escenarios donde la precisión es fundamental, como la planificación de rutas en áreas urbanas o restricciones geográficas específicas.

Esta carencia puede llevar a soluciones subóptimas, dado que las métricas de distancia aproximadas no consideran factores como la topografía o las restricciones viales, aspectos críticos para optimizar rutas en aplicaciones del mundo real.

Para abordar el uso de distancias reales, se deben considerar diferentes propuestas que permitan calcular distancias y tiempos de viaje con precisión a partir de datos de redes viales reales. Entre estas, destaca *Google Distance Matrix API* [29], que emplea datos en tiempo real, y *GraphHopper* [30], una alternativa de código abierto que utiliza mapas digitales para cálculos eficientes. Su posible incorporación permite ampliar las capacidades de **BHAVRP**, modelando escenarios más realistas y ofreciendo resultados ajustados a las necesidades de casos prácticos que demanden precisión geográfica y eficiencia operativa.

**BHAVRP** presenta otras limitaciones que impactan su diseño y adaptabilidad. Su integración está restringida a aplicaciones desarrolladas en Java, limitando su interoperabilidad con otros sistemas. Desde una perspectiva de diseño, el sistema presenta un alto acoplamiento entre sus clases, lo que dificulta la extensión y el mantenimiento del código. Este componente carece de un manejo robusto de excepciones, lo que afecta su fiabilidad en escenarios complejos. Del mismo modo, carece de soporte para la carga y exportación de datos en formatos estándar como JSON o CSV. Este aspecto dificulta la integración de **BHAVRP** en flujos de trabajo que dependan de datos en otros formatos y limita su capacidad para compartir resultados de manera eficiente. Por último, no incluye herramientas para evaluar la calidad de las asignaciones realizadas. Sin herramientas de evaluación, los usuarios no pueden medir ni comparar la efectividad de las heurísticas, lo que dificulta la mejora continua de los resultados y la toma de decisiones informadas para optimizar el sistema.

Dado este panorama, la implementación de **BHAVRP** en Java plantea desafíos significativos en un entorno cada vez más orientado a la interoperabilidad y a la integración de sistemas diversos. Aunque existen pocas bibliotecas en Python que aborden específicamente problemas como la resolución de la fase de asignación en el

**MDVRP**, el crecimiento de su comunidad y su popularidad en la resolución de problemas de optimización e inteligencia artificial lo posicionan como un lenguaje atractivo. Esto resalta la necesidad de reestructurar su arquitectura, no solo para facilitar su mantenimiento, sino también para maximizar su adaptabilidad y alcance en un ecosistema más amplio y en constante evolución.

El objeto de estudio de este trabajo son los **MDVRP** y el análisis de la eficacia de **BHAVRP** como herramienta para resolverlos. El campo de acción se enfoca en solventar las deficiencias identificadas en **BHAVRP** mediante un rediseño orientado a mejorar su arquitectura y funcionalidad, así como en la implementación de una nueva versión en el lenguaje de programación Python.

Por tanto, se identifica como problema a resolver que las limitaciones de **BHAVRP** restringen su integración y uso. Estas limitaciones incluyen su falta de interoperabilidad con otros lenguajes de programación, su dependencia de bibliotecas privadas, en problemas de diseño como el alto acoplamiento. Estos problemas complican la extensión, el mantenimiento y la reutilización de la biblioteca, afectando su capacidad para adaptarse a nuevos escenarios o integrar tecnologías modernas.

Para dar solución al problema planteado se traza como objetivo general: desarrollar una nueva y mejorada versión del componente de software **BHAVRP** en los lenguajes de programación Java y Python, que solvete las limitaciones presentes en la versión actual.

Para dar cumplimiento al objetivo general se definen los siguientes objetivos específicos con sus tareas asociadas:

1. Identificar deficiencias, limitaciones y mejoras en el diseño arquitectónico de **BHAVRP**.
  - Analizar la arquitectura y las funcionalidades brindadas por **BHAVRP** para identificar deficiencias que presenta en su diseño arquitectónico y en su código.

- Investigar bibliotecas en Java u otros lenguajes que resuelvan la fase de asignación en **MDVRP**, evaluando aquellas que implementen enfoques innovadores o más eficientes que los utilizados actualmente en **BHAVRP**, para incorporar sus técnicas o adaptarlas al sistema.
  - Estudiar patrones y principios de diseño para identificar aquellos que pueden incorporarse para solventar las deficiencias encontradas y mejorar el diseño arquitectónico del componente.
  - Analizar la integración de bibliotecas o herramientas para el cálculo de distancias reales necesarios en la construcción de la matriz de costo empleada en ambas fases en la resolución del **MDVRP**.
  - Asimilar el lenguaje de programación Python.
2. Desarrollar una nueva versión de **BHAVRP** en los lenguajes de programación Java y Python.
- Implementar en Python y refactorizar en Java las clases necesarias para modelar un problema de planificación de rutas de vehículos con múltiples depósitos.
  - Implementar en Python y refactorizar en Java las clases necesarias para modelar el resultado de la asignación de clientes a depósitos en un problema de planificación de rutas de vehículos con múltiples depósitos.
  - Implementar en Python y refactorizar en Java los patrones de diseño necesarios para mejorar el proceso de resolución de la fase de asignación.
  - Incorporar en ambas bibliotecas **BHAVRP** el cálculo de distancia real mediante el uso de bibliotecas o herramientas que brinden esta funcionalidad.
3. Validar el funcionamiento de la nueva versión de **BHAVRP** mediante la ejecución de pruebas.
- Diseñar los experimentos necesarios para validar el correcto funcionamiento de las dos versiones de la biblioteca.
  - Ejecutar los experimentos diseñados para validar el correcto funcionamiento de las nuevas versiones de la biblioteca.

- Analizar los resultados obtenidos de los experimentos para identificar fortalezas, áreas de mejora y posibles ajustes en las nuevas versiones de la biblioteca.

El cumplimiento del objetivo general tiene como valor práctico un aporte significativo, representado en una nueva versión del componente de software **BHAVRP** desarrollado en ambos lenguajes de programación. Este aporte se clasifica como un diseño de software [31] debido que implica una reestructuración de la versión anterior de la biblioteca para optimizar su arquitectura, garantizar su interoperabilidad y facilitar su integración en un entorno más diverso y colaborativo. La evaluación del aporte incluye su valor científico y tecnológico, pues se pone a disposición de la comunidad científica una nueva versión de **BHAVRP** en un lenguaje ampliamente adoptado en el ámbito académico y de la industria, favoreciendo así la evolución continua de herramientas para la resolución de problemas complejos como el **MDVRP**.

Para alcanzar los objetivos mencionados, el trabajo se encuentra estructurado en tres capítulos. **Capítulo 1: Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos**, con epígrafes para abordar su arquitectura y componentes principales y los principios y patrones de diseño que sustentan su desarrollo. Adicionalmente, se realiza un estudio comparativo con otras bibliotecas relevantes, destacando sus fortalezas y limitaciones y se describe la forma en que **BHAVRP** maneja el cálculo y la verificación de distancias. Finalmente, se describen las principales deficiencias de **BHAVRP**, identificando áreas de mejora para optimizar su desempeño en futuros desarrollos.

El **Capítulo 2: Diseño de la nueva versión de BHAVRP**, se dedica a describir la solución mediante artefactos de ingeniería de software, profundizando en el diseño y desarrollo de la nueva versión de **BHAVRP** en Java y Python. Este capítulo abarca el rediseño en la arquitectura de la versión en Java y la implementación de la nueva versión en Python, analizando las decisiones arquitectónicas tomadas y cómo estas contribuyen a superar las limitaciones de la versión anterior.

El **Capítulo 3: Análisis y validación de BHAVRP**, se centra en validar las soluciones propuestas a través de un conjunto de experimentos. En este capítulo, se presentan los experimentos diseñados, las instancias utilizadas y los resultados obtenidos tras la implementación de la nueva versión de **BHAVRP**.

# Capítulo 1: Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos

## 1.1. Introducción

En este capítulo se realiza un análisis detallado de la Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos (**BHAVRP**), una herramienta orientada a resolver la fase de asignación del Problema de Planificación de Rutas de Vehículos con Múltiples Depósitos (**MDVRP**).

En la sección 1.2, se detallan las especificaciones de la biblioteca, describiendo su propósito, componentes principales y la estrategia de solución basada en algoritmos de asignación y agrupamiento. La subsección 1.2.1 presenta el marco teórico que aborda los problemas de planificación de rutas de vehículos, sus variantes y los métodos de resolución documentados en la literatura. En 1.2.2 se aborda su diseño arquitectónico. Posteriormente, en 1.2.3 y 1.2.4, se explican los principios y patrones de diseño implementados para reforzar su estructura.

La sección 1.3 compara **BHAVRP** con otras bibliotecas que, aunque no están específicamente diseñadas para el **MDVRP**, abordan la fase de asignación con herramientas similares. En 1.4, se analizan bibliotecas especializadas en el cálculo de distancias, evaluando su aplicabilidad en la optimización de rutas. Finalmente, en la sección 1.5, se identifican las limitaciones y deficiencias actuales de **BHAVRP**, destacando oportunidades de mejora en su diseño y funcionalidad.

## 1.2. Especificaciones de BHAVRP

En la actualidad existen escasas bibliotecas que implementen métodos heurísticos tanto para la resolución de problemas de planificación de rutas de vehículos como para la resolución de cualquier problema de optimización. El mismo escenario es válido para la variante con múltiples depósitos en los problemas de planificación de rutas de vehículos.

**BHAVRP** es un componente de software que le brinda solución a la fase de asignación de la variante **MDVRP** en las siguientes herramientas: **CaS VRP** [32], **BHC VRP** [33] y **TransO** [34]. El procedimiento es bastante simple, la aplicación que requiere realizar la asignación de clientes a depósitos es responsable de efectuar la carga de los datos del problema. **BHAVRP** efectúa dicha petición y si la carga es exitosa, entonces la aplicación solicita ejecutar alguna de las heurísticas de asignación brindadas por **BHAVRP**. Finalmente se obtiene la solución generada por dicha heurística. El diagrama de actividades de la Figura 1 describe el flujo de funcionamiento de **BHAVRP**, abarcando dos perspectivas principales: el software que consume **BHAVRP** y la biblioteca en sí.

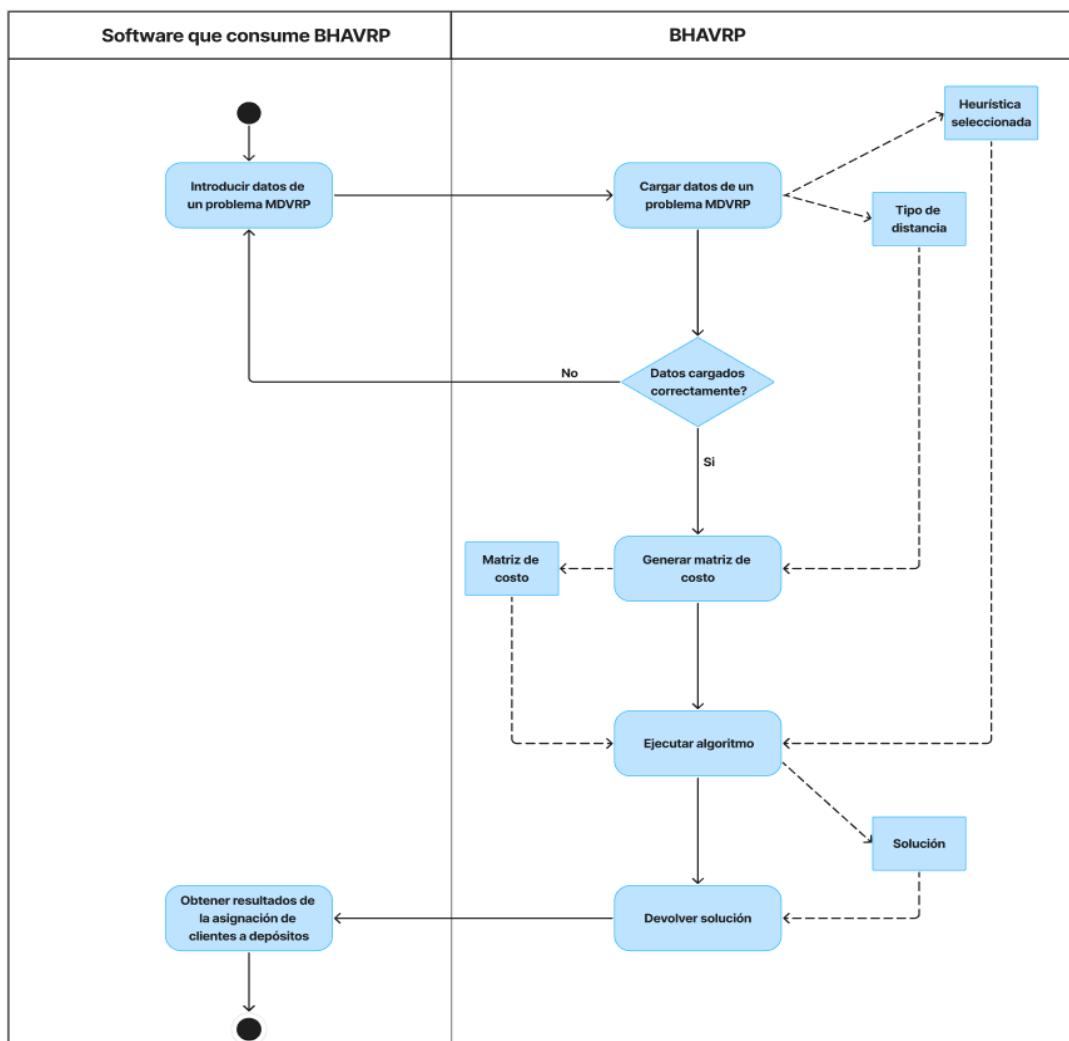


Figura 1: Diagrama de actividades de BHAVRP.

El flujo se inicia desde el software con la introducción de los datos correspondientes a un problema de **MDVRP**, los cuales incluyen datos como clientes, depósitos y el tipo de distancia a usar. Estos datos son procesados por **BHAVRP**, que a través de sus algoritmos asigna eficientemente los clientes a los depósitos. Finalmente, se retorna una asignación al software, de los clientes a los depósitos cerrando el flujo con los resultados del proceso.

La biblioteca está desarrollada en Java, y tiene como objetivo fundamental encapsular el comportamiento de un conjunto de algoritmos que pueden ser utilizados en la fase de asignación de clientes a depósitos en la variante de **MDVRP** [35]. **BHAVRP** cuenta con un diseño robusto que permite la incorporación de nuevos algoritmos de asignación, así como su extensión a otras variantes de múltiples depósitos mediante la incorporación de características propias de estas variantes. Esto sería posible para variantes como Múltiples Depósitos con Flota Heterogénea [36], Múltiples Depósitos con Ventanas de Tiempo [37] y una combinación de las anteriores.

En **BHAVRP** están disponibles 18 algoritmos de asignación, de las cuales 6 se seleccionaron del estado del arte, mientras que 5 son adaptaciones de algoritmos de agrupamiento empleados en la minería de datos. Se puede apreciar en la Figura 2 la totalidad de los algoritmos disponibles en **BHAVRP** organizadas en: algoritmos de agrupamiento (particionales y jerárquicos), métodos clásicos de asignación (basados en urgencias, ciclos y clústeres), además de enfoques aleatorios, estrategias fundamentadas en distancias y otras inspiradas en la asignación cíclica.

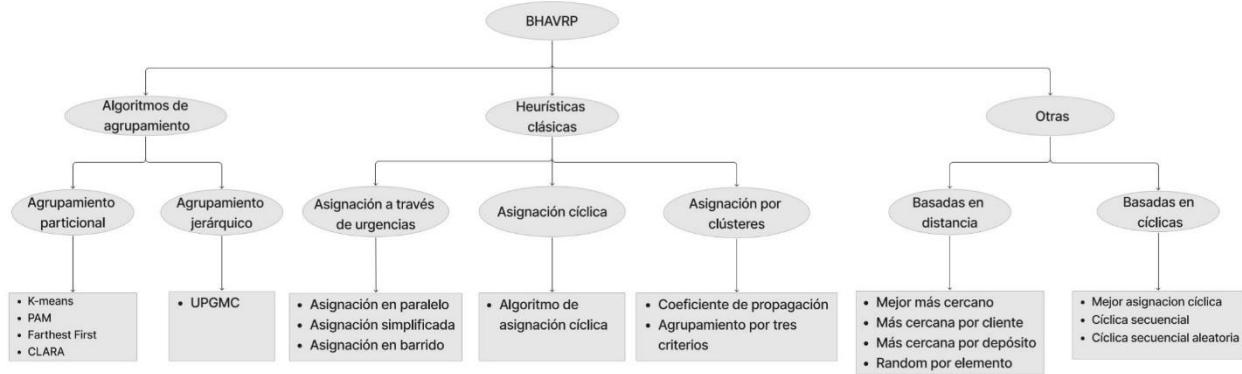


Figura 2: Heurísticas implementadas en BHAVRP.

Además de la ejecución de estas heurísticas, **BHAVRP** ofrece una serie de funcionalidades clave que la hacen robusta y adaptable. Entre ellas, la carga de ficheros de entrada en formatos estándar como *.data* o *.txt*, facilitando la incorporación de datos al sistema. No obstante, sería recomendable ampliar el soporte para otros formatos de entrada basados en estándares ampliamente utilizados, como *JSON* y *XML*. Además, brinda opciones para configurar parámetros, lo que permite al usuario ajustar diferentes aspectos de las heurísticas según las necesidades específicas del problema.

### 1.2.1. Marco teórico

Los **VRP** son una serie de problemas que hacen referencia al transporte de mercancías, bienes o servicios a ciertos puntos con una flota de vehículos. Formalmente, se definen sobre un grafo completo  $G = (V, A)$ , con costes asociados a los arcos ( $A$ ) y donde los clientes dispersos geográficamente representan los vértices ( $V$ ), siendo el vértice 0 ( $v_0$ ) el depósito o almacén donde se encuentran los vehículos de transporte [38].

Existen diversas variantes del Problema de Planificación de Rutas de Vehículos que abordan distintas restricciones y características del entorno de transporte. Estas variantes amplían la complejidad del problema original, y entre las más destacadas se encuentran:

- Problema del Agente Viajero (*Traveling Salesman Problem, TSP*): busca encontrar la ruta más corta para visitar un conjunto de ciudades una sola vez y regresa al punto de origen [39].
- Problema de Planificación de Rutas de Vehículos con Restricciones de Capacidad (*Capacitated Vehicle Routing Problem, CVRP*): involucra la planificación de rutas para vehículos con capacidad limitada, considerando la demanda de los clientes [40].
- Problema de Rutas de Vehículos con Distancia Dinámica (*Dynamic Vehicle Routing Problem, DVRP*): considera cambios en las rutas debido a la variabilidad del tráfico o la aparición de nuevas solicitudes durante la operación [41].
- Problema de Rutas de Vehículos con Restricciones de Tiempo (*Time Windows Vehicle Routing Problem, TVRP*): requiere que los vehículos lleguen a los clientes en un tiempo específico, sin flexibilidad en los horarios de llegada [42].
- Problema de Rutas de Vehículos con Flota Heterogénea (*Heterogeneous Fleet Vehicle Routing Problem, HFVRP*): consiste en planificar rutas para una flota de vehículos que presenten diferentes características [40].

Existen otras variantes del problema de rutas de vehículos que se enfocan en aspectos específicos de la logística y la planificación. Entre ellas destacan:

- Problema de Rutas de Vehículos con Ventanas de Tiempo (*Vehicle Routing Problem with Time Windows, VRPTW*), exige que los vehículos lleguen a los clientes dentro de intervalos de tiempo predefinidos [40].
- Problema de Rutas de Vehículos con Recogidas y/o Entregas (*Vehicle Routing Problem with Pickup and Delivery, VRPPD*), involucra la planificación de rutas que incluyen tanto recogidas como entregas, considerando restricciones específicas de cliente a cliente [43].
- Problema de Rutas de Vehículos con Múltiples Depósitos y Recogidas/Entregas Combinadas (*Multiple Depot Vehicle Routing Problem with Demand, MDVRPPD*):

abarca la asignación de clientes a múltiples depósitos y rutas que combinan recogidas y entregas [44].

- Problema de Recolección y Entrega con Ventanas de Tiempo (*Pickup and Delivery Problem with Time Windows, PDPTW*): abarca la asignación de clientes a múltiples depósitos y rutas que combinan recogidas y entregas [45].

En la Figura 3 se muestran las variantes de problemas de planificación de rutas de vehículos comentadas anteriormente.

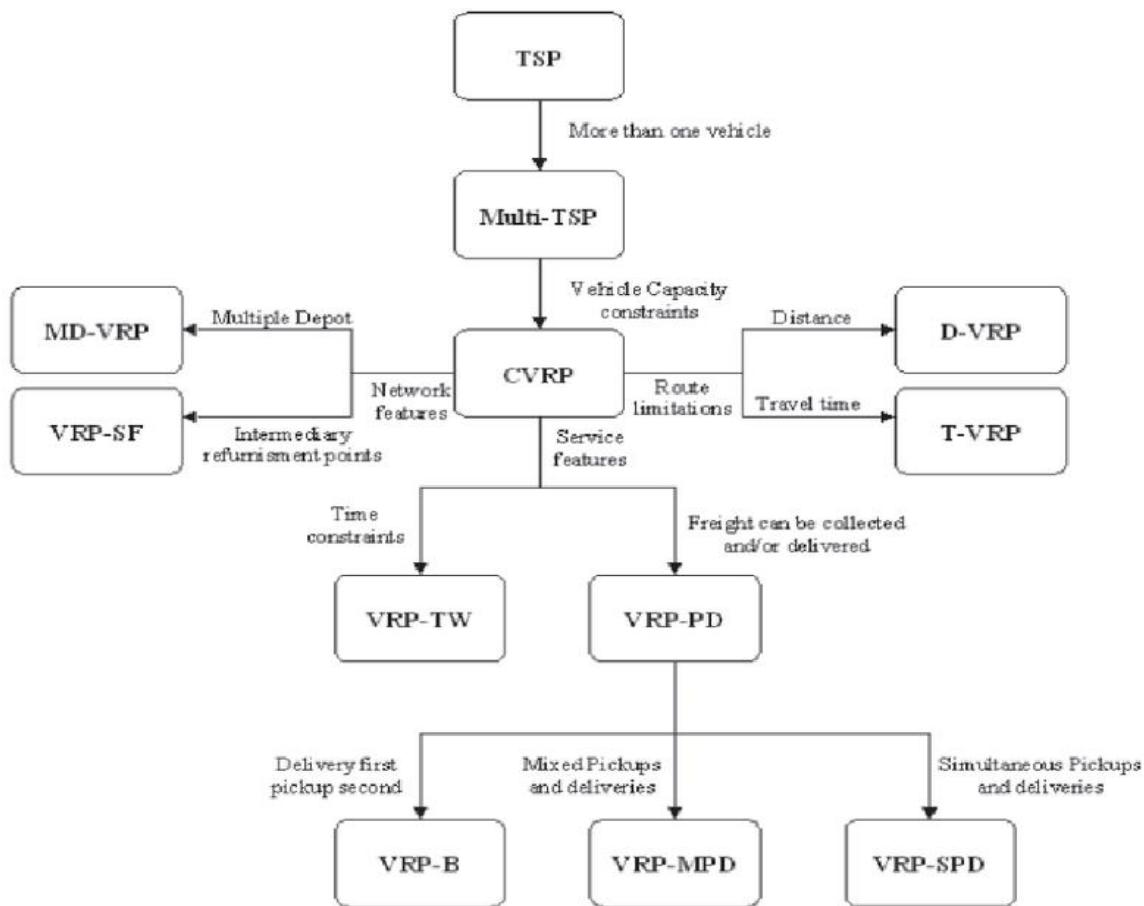


Figura 3: Variantes del VRP [46].

#### 1.2.1.1. Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos

De todas las variantes de Problema de Planificación de Rutas de Vehículos, una de las más importantes y la que es objeto de estudio de esta investigación es el Problema de

Planificación de Rutas de Vehículos con Múltiples Depósitos (*Multiple Depot Vehicle Routing Problem, MDVRP*) [35]. Esta variante representa un problema de optimización combinatoria perteneciente a la categoría denominada **NP**-duro, debido a que es un problema que no cuenta con un algoritmo que encuentre la solución óptima en tiempo polinomial. Para su resolución se emplean mayormente métodos heurísticos que proporcionan soluciones aproximadas en un espacio corto de tiempo.

Esta variante consiste en una flota de vehículos dispuesta desde diferentes depósitos que se desplazan hacia la ubicación de clientes para satisfacer sus demandas. Además, es un problema de optimización que tiene como objetivo minimizar los costos ya sea en distancia o en tiempo, cumpliendo en todo momento las restricciones del problema. Estos problemas están presentes en diversos sectores de la sociedad cubana, como el transporte de mercancías, los servicios de mensajería, entre muchos otros.

El **MDVRP** plantea la necesidad de asignar clientes a depósitos, determinar qué vehículo de la flota será asignado a cada depósito y garantizar que cada vehículo inicie y finalice su ruta en su depósito correspondiente. Estas particularidades del problema propician que su solución generalmente se organice en dos etapas: la primera se enfoca en asignar cada cliente a un depósito adecuado, mientras que la segunda etapa se centra en planificar las rutas de los vehículos a partir de esas asignaciones.

En lo que respecta a los métodos de solución, existen tres categorías a considerar: métodos exactos, heurísticas y metaheurísticas. Entre los métodos exactos se pueden distinguir tres grupos: búsqueda directa de árbol, programación dinámica y programación lineal, entera y mixta [47]. Todos estos métodos son eficientes en problemas de hasta 50 depósitos, debido a restricciones de tiempo computacional [48].

En la Figura 4 se muestra una representación gráfica de la clasificación de los métodos de optimización combinatoria. Esto permite visualizar que los métodos heurísticos engloban tanto las heurísticas tradicionales como las metaheurísticas avanzadas.

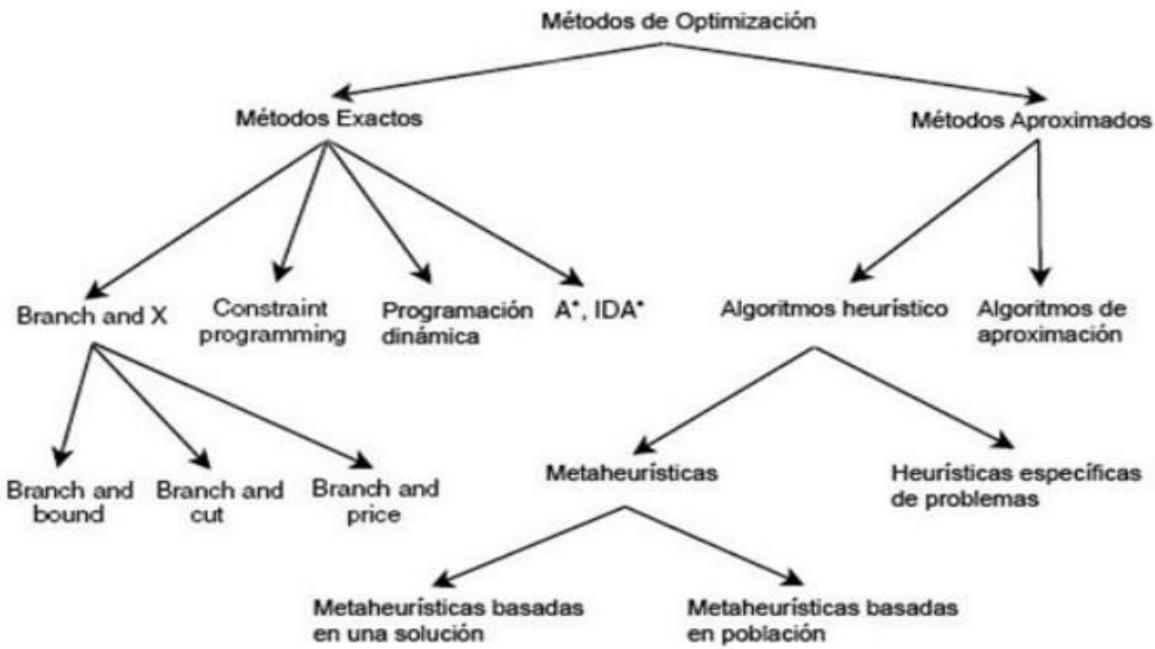


Figura 4: Métodos de optimización combinatoria [48].

El enfoque de esta investigación se centra en las heurísticas, que son procedimientos diseñados para proporcionar soluciones de aceptable calidad mediante una exploración limitada del espacio de búsqueda. La mayoría de las heurísticas clásicas para resolver **VRP** fueron desarrolladas entre los años 1960 y 1990 [5] [49]. Estos métodos por lo general parten de rutas que contienen un nodo único para encontrar el mejor par (nodo, ruta) que representan la mejor intersección. Los métodos heurísticos se pueden clasificar en métodos constructivos, métodos de dos fases y heurísticas de mejora [50] [51].

#### 1.2.1.2. Estrategias de solución empleadas para la resolución del problema

Para resolver el problema que presenta la variante **MDVRP**, se han utilizado principalmente métodos heurísticos, con énfasis en aquellos que aplican la estrategia "Agrupar primero, planificar después" (*Cluster First – Route Second, CFRS*) [52]. Esta estrategia consiste en realizar la asignación de clientes a los depósitos y luego la planificación de rutas de los vehículos. Dentro de las metaheurísticas se destacan las basadas en Búsqueda Tabú (*Tabu Search, TS*) [53] que se mantuvieron como el mejor

enfoque para resolver el **MDVRP** durante mucho tiempo, debido a las características de su implementación [54] [55] [56] [57] [58] [59].

Los algoritmos de Minería de Datos se emplean con frecuencia para abordar tareas de agrupamiento. Entre estos destacan las técnicas de Agrupamiento Particional y Agrupamiento Jerárquico. La Minería de Datos es un campo interdisciplinario de la computación que se centra en el descubrimiento de patrones significativos en grandes volúmenes de datos mediante el uso de métodos avanzados de inteligencia artificial, aprendizaje automático, estadística y bases de datos. Su objetivo principal es extraer conocimiento valioso de los datos y transformarlo en una estructura comprensible, ya sea para su análisis o para su almacenamiento en bases de datos con miras a usos posteriores [16] [60].

Entre las técnicas más representativas, está el Agrupamiento o *Clustering* [61], que es una de las tareas más frecuentes de la Minería de Datos [60] y consiste en generar conjuntos de datos organizados en grupos. La idea fundamental es maximizar la similitud entre los elementos de cada grupo y minimizar la similitud entre los distintos grupos. Esta técnica depende de los datos de partida y de qué medida de semejanza se esté utilizando. Entre los algoritmos de agrupamiento existen dos que son muy importantes, y de los cuales se derivan otros. Estos algoritmos son *K-means* [14] y *K-medoids* [62], que han servido como base para desarrollar otros algoritmos avanzados. Por ejemplo, *Farthest First* [63] es una variante de *K-means* que optimiza la selección inicial de centroides maximizando la distancia entre ellos, mientras que algoritmos como *PAM* [16] y *CLARA* [64] se derivan de *K-medoids*, adaptándose a diferentes tamaños y características de los conjuntos de datos. Además, existen otras técnicas significativas como *DBSCAN* [65], que identifica grupos de densidad en los datos, y los algoritmos jerárquicos, que forman una estructura en niveles para describir las relaciones entre los datos.

### 1.2.2. Diseño arquitectónico de BHAVRP

La arquitectura de **BHAVRP** ha sido diseñada con un enfoque basado en la reutilización y modularidad, lo que garantiza su adaptabilidad. Esta estructura se organiza en dos capas principales: *Application* y *JDK*, cada una con responsabilidades específicas que optimizan la funcionalidad y la interacción entre los componentes de la biblioteca. A continuación, se detalla en la

Figura 5 la organización de estas capas y su rol en el diseño general de **BHAVRP**.

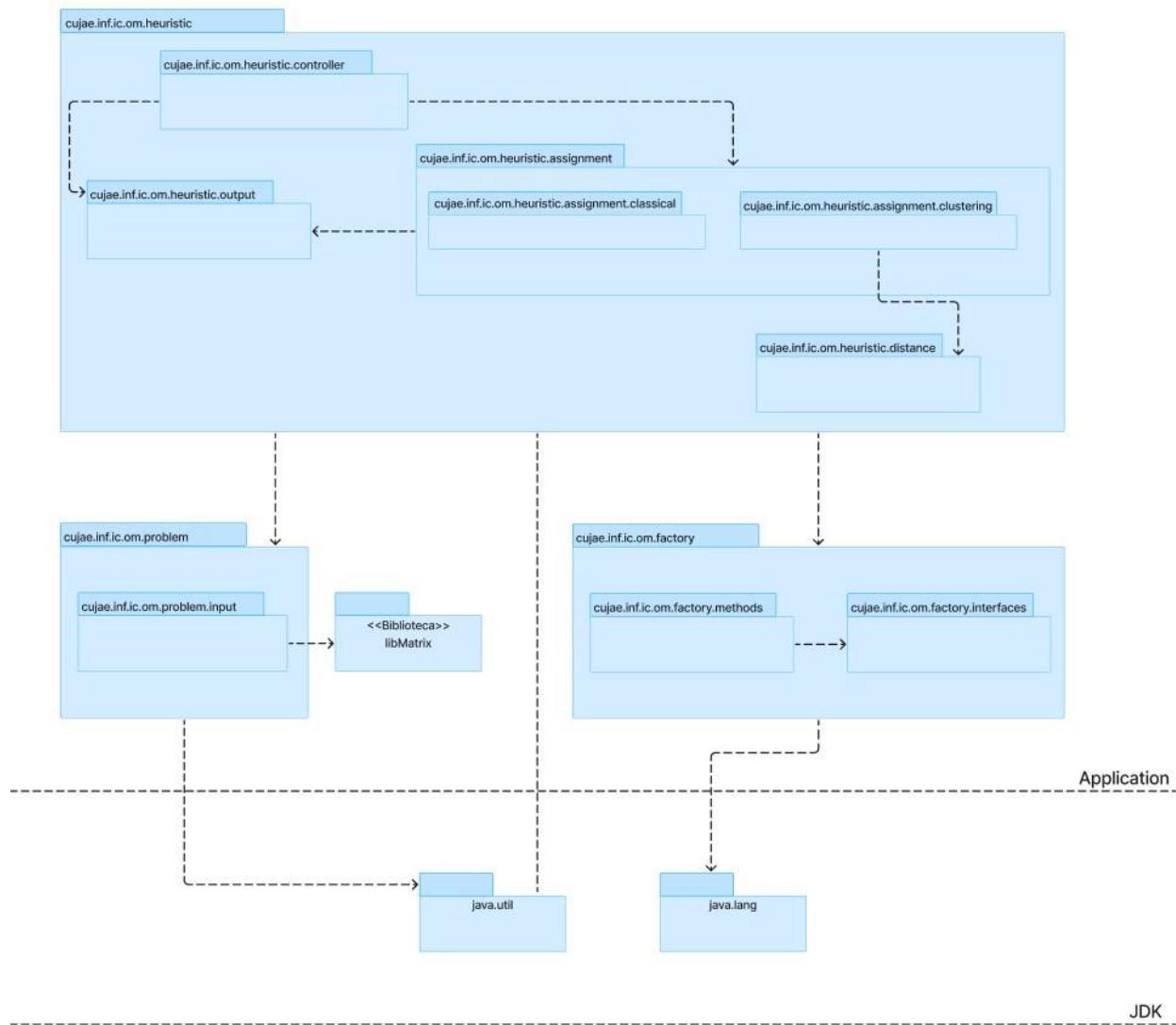


Figura 5: Arquitectura de BHAVRP [24].

La capa *Application* es la parte más importante del sistema, ya que contiene los paquetes y módulos relacionados con la lógica del negocio. Es aquí donde se implementan las funcionalidades específicas del sistema, asegurando que los requisitos del problema se traduzcan en soluciones operativas. Por otro lado, la capa JDK incluye los elementos proporcionados por la plataforma Java, como clases e interfaces estándar, que complementan y soportan la implementación de la lógica del negocio, pero no son específicos del sistema desarrollado.

A continuación, se enuncian los paquetes que contiene la capa *Application*, y se describen las funciones de los mismos.

El paquete *cujae.inf.citi.om.problem* es el encargado de modelar los datos de un problema de planificación de rutas de vehículos con múltiples depósitos, contiene al paquete *cujae.inf.citi.om.problem.input* y la biblioteca *libMatrix*. El primero contiene las clases necesarias para modelar el problema y con la biblioteca se logra modelar la matriz de costo. Esta matriz es fundamental, ya que representa los costos asociados al desplazamiento entre los diferentes nodos, (clientes y depósitos) del problema. La Figura 6 muestra el diagrama de clases del paquete *input*.

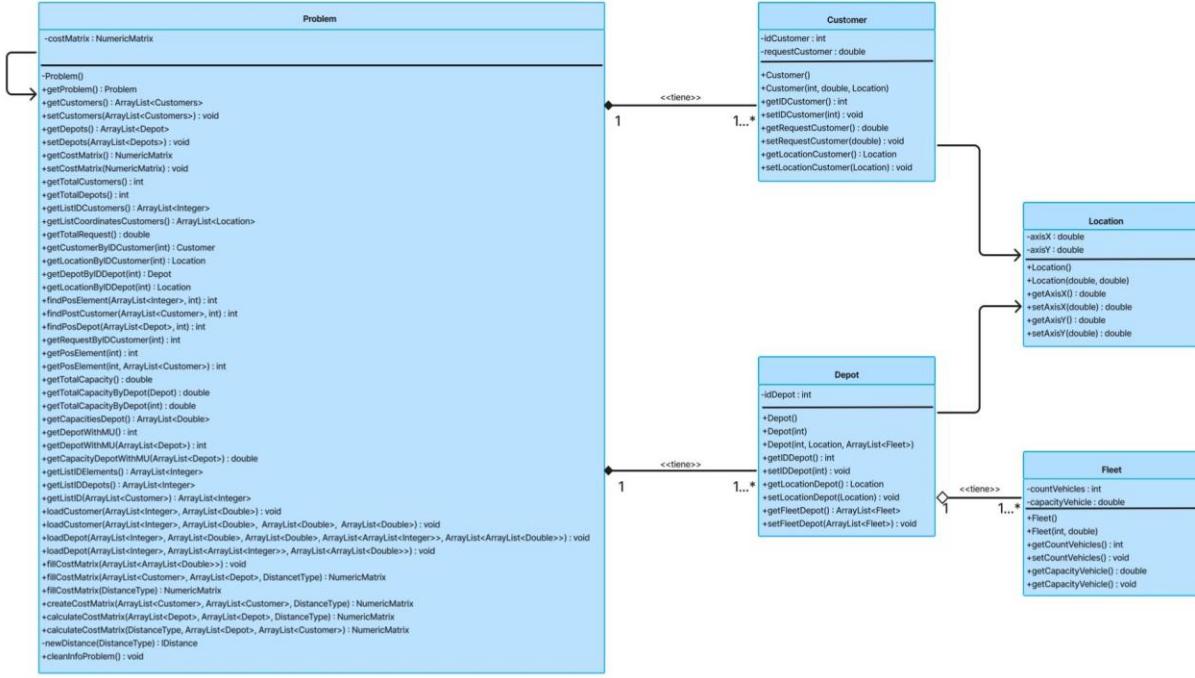


Figura 6: Diagrama de clases del paquete *input*.

En el paquete *cujae.inf.citi.om.heuristic.assignment* se agrupan los paquetes: *classical*, responsable de modelar las clases que implementan las heurísticas clásicas y *clustering*, responsable de modelar las clases que implementan las adaptaciones a los algoritmos de agrupamiento. A continuación, en la Figura 7 y en la Figura 8 muestran los diagramas de clases de estos paquetes.

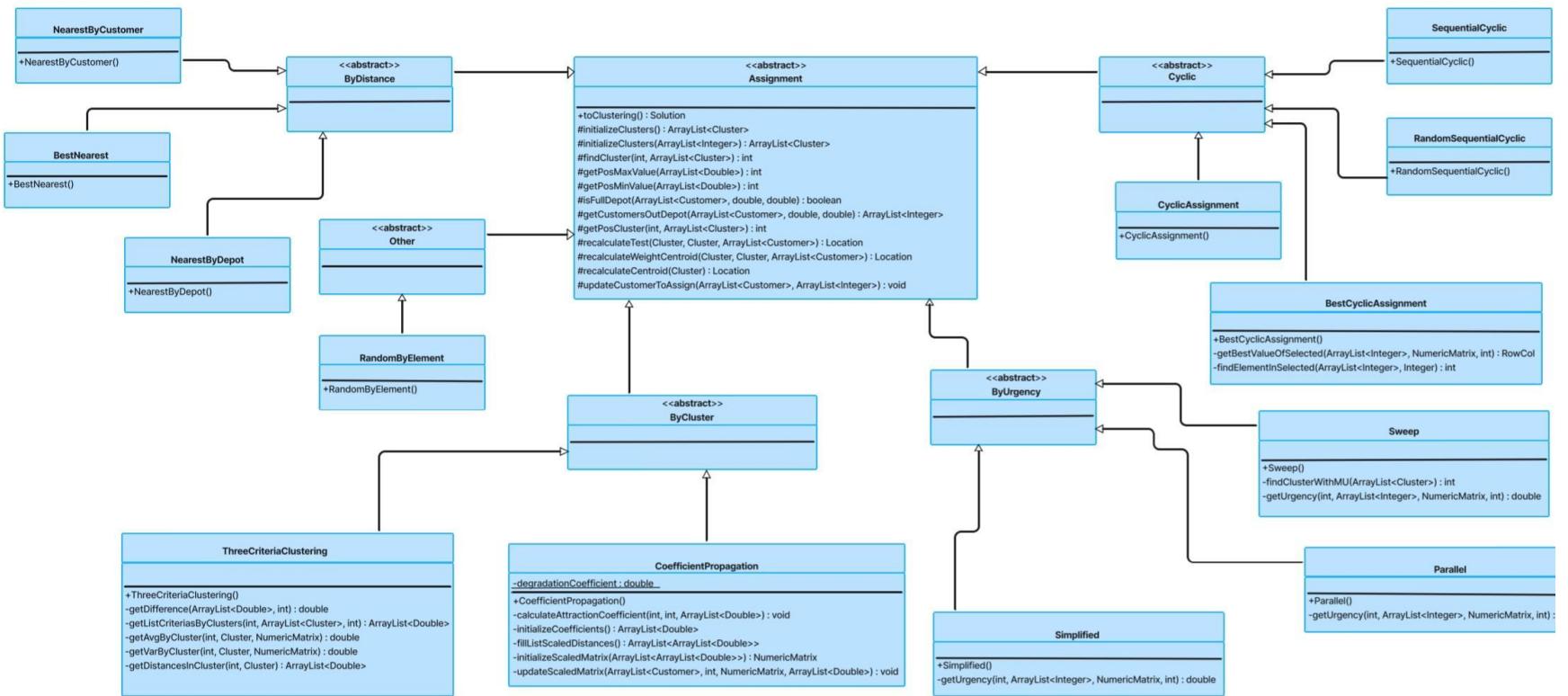


Figura 7: Diagrama de clases del paquete *classical*.

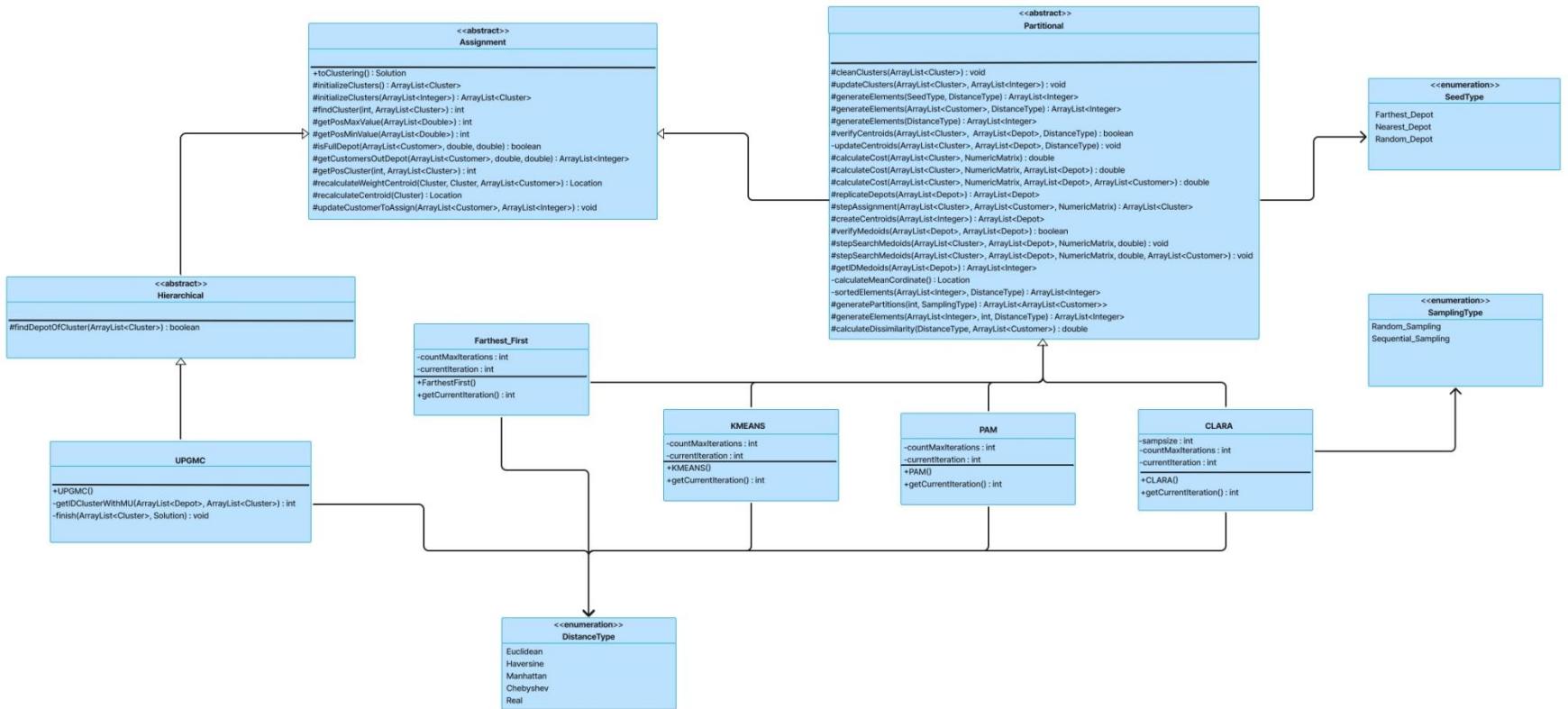


Figura 8: Diagrama de clases del paquete *clustering*.

El paquete *cujae.inf.citi.om.heuristic.output* se utiliza para modelar la solución, es decir, el resultado del proceso de asignación de los clientes a los depósitos. A continuación, en la Figura 9 muestra el diagrama de clases de dicho paquete.

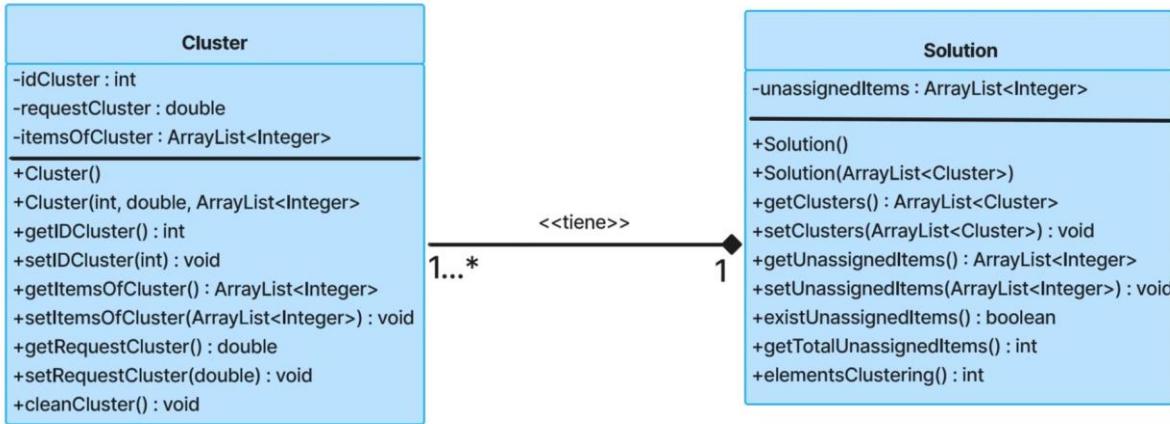


Figura 9: Diagrama de clases del paquete *output*.

El paquete *cujae.inf.citi.om.heuristic.controller* es donde se ubica la clase responsable de todo el proceso de asignación de clientes a depósitos en la biblioteca. En la Figura 10 muestra el diagrama de clases de este último.

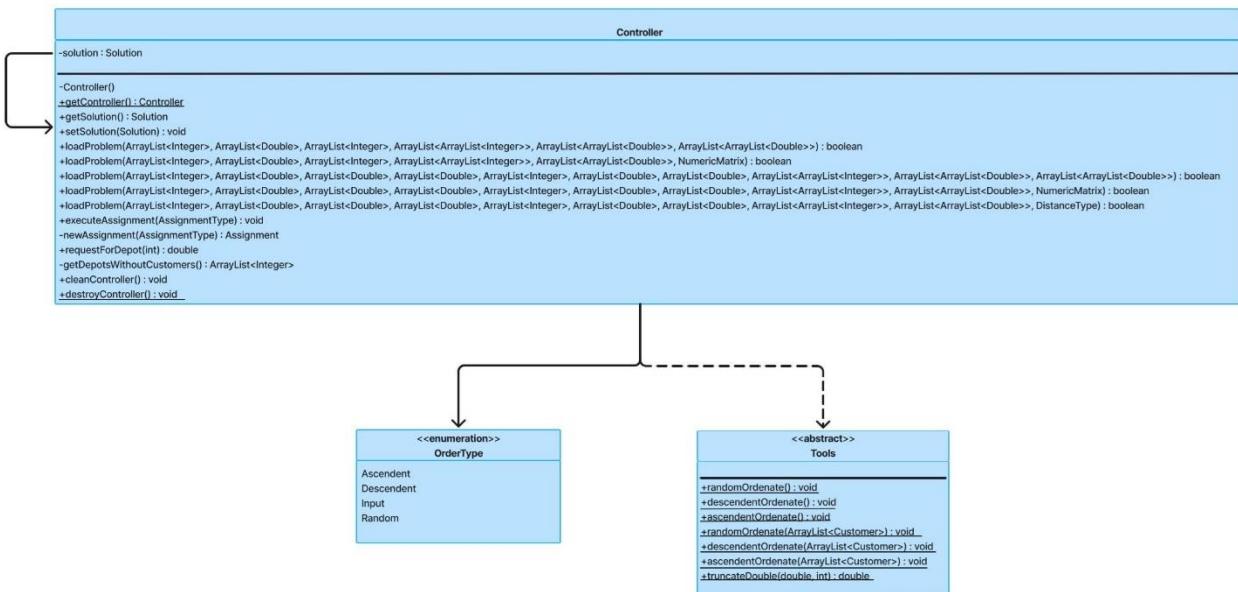


Figura 10: Diagrama de clases del paquete *controller*.

Por último, el paquete *cujae.inf.citi.om.factory* es el responsable de garantizar la carga dinámica de los algoritmos de asignación y las distancias aproximadas. Este paquete implementa el patrón *Factory Method* [66], el cual permiten encapsular la creación de objetos relacionados con las heurísticas y las distancias. Esto facilita la extensión del sistema para añadir nuevas heurísticas y distancias sin modificar el código existente, promoviendo el cumplimiento del Principio Open/Close [67] y mejorando la flexibilidad en la configuración y ejecución del sistema. La Figura 11 muestra el diagrama de clases del paquete descrito.

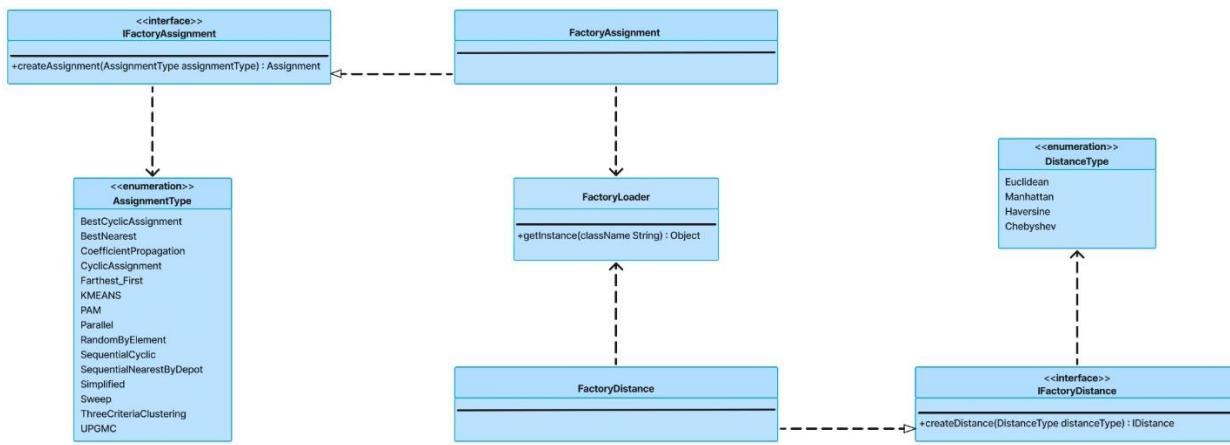


Figura 11: Diagrama de clases del paquete *factory*.

### 1.2.3. Principios de diseño implementados en BHAVRP

En el proceso de desarrollo de software, uno de los objetivos clave es la creación de soluciones robustas, mantenibles y escalables. Para lograr este propósito, se emplean principios y patrones de diseño que guían la toma de decisiones en la arquitectura del sistema. **BHAVRP**, como una biblioteca orientada a la resolución del problema de asignación de clientes a depósitos en la variante **MDVRP**, ha adoptado diversos principios de diseño que no solo mejoran su estructura, sino que también optimizan su mantenimiento y extensión. En esta sección, se presentan los principios y patrones de diseño utilizados en **BHAVRP**.

Los principios de diseño son un conjunto de directrices que ayudan a crear un software robusto, flexible y fácil de mantener. Al aplicarlos, los desarrolladores buscan mejorar la estructura y organización del código, reduciendo la complejidad y facilitando futuras

modificaciones y extensiones del sistema sin que esto afecte su funcionamiento. Estos principios no son reglas estrictas, sino buenas prácticas que orientan el proceso de diseño hacia un código limpio, modular y adaptable [68].

Los principios **SOLID** son un conjunto de directrices de diseño de software que buscan mejorar la calidad, flexibilidad y mantenibilidad del código en la programación orientada a objetos. Estos principios fueron formulados por Robert C. Martin y son ampliamente utilizados en la industria del software [69]. En este análisis, se abordan los principios **SOLID** y como los cumple **BHAVRP**.

En la Tabla 1 se resume como **BHAVRP** cumple con la implementación de estos principios y como se reflejan en su arquitectura y diseño.

Tabla 1: Principios de diseño SOLID que cumple BHAVRP.

Principio	Criterio de evaluación	¿BHAVRP lo cumple?
Principio de Responsabilidad Única ( <i>Single Responsibility Principle, SRP</i> )	Cada clase tiene una única responsabilidad bien definida.	Parcial
	Los métodos de una clase están estrechamente relacionados con su propósito principal.	Parcial
Principio Abierto/Cerrado ( <i>Open/Closed Principle, OCP</i> )	Es posible extender el comportamiento de las clases sin modificar su código existente.	Total
	Se usan interfaces y clases abstractas para permitir la extensión.	Parcial
Principio de Sustitución de Liskov ( <i>Liskov Substitution Principle, LSP</i> )	Las subclases pueden ser utilizadas en lugar de sus clases base sin causar errores.	Total
	Las subclases no lanzan excepciones que no están en la firma de los métodos de la clase base.	Nulo
Principio de Segregación de Interfaces ( <i>Interface Segregation Principle, ISP</i> )	Las interfaces son específicas y cohesivas, en lugar de ser grandes y generales.	Total
	Las clases que implementan interfaces no tienen métodos vacíos o sin implementar.	Total
Principio de Inversión de Dependencias ( <i>Dependency Inversion Principle, DIP</i> )	Se usan abstracciones (interfaces o clases abstractas) en lugar de implementaciones concretas.	Total
	Las dependencias se inyectan en lugar de ser creadas dentro de las clases.	Nulo

A partir de la información presentada en la Tabla 1, se puede observar que la biblioteca **BHAVRP** cumple con la mayoría de los principios **SOLID**, lo cual refleja una estructura

de diseño robusta y bien organizada. En particular, los principios **SRP**, **OCP** e **ISP** se implementan casi en su totalidad, garantizando cohesión, extensibilidad y modularidad en el sistema. Por otra parte, se identifican áreas de mejora en los principios **LSP** y **DIP**. En el caso del **LSP**, se cumple el criterio principal de que las subclases pueden ser utilizadas en lugar de sus clases base sin causar errores, pero existe una deficiencia en el manejo de excepciones, ya que algunas subclases lanzan excepciones que no están especificadas en la firma de los métodos de la clase base.

Abordar estas limitaciones debe ser una prioridad en las futuras versiones de **BHAVRP**, asegurando que todas las dependencias sean inyectadas y que las subclases cumplan completamente con las expectativas de sus clases base. La resolución de estas áreas pendientes debe fortalecer aún más la arquitectura del sistema y su alineación con los principios **SOLID**.

Además de los principios **SOLID**, existen otros principios de diseño que son esenciales para lograr un software robusto y mantenible. Estos principios complementan las directrices ofrecidas por los principios **SOLID**, proporcionando un marco más amplio para la creación de código limpio, modular y adaptable [69] [70] [71].

En el contexto de **BHAVRP**, la aplicación de estos principios no solo mejora la calidad del código, sino que también optimiza su estructura, facilitando su comprensión y mantenimiento. A continuación, se enuncian estos principios, destacando su definición y su implementación en **BHAVRP**, así como su impacto en el desarrollo del software.

A continuación, en la Tabla 2 se presenta como están implementados en **BHAVRP** otros principios de diseño, los cuales complementan a los principios **SOLID**. En la tabla, se explica brevemente cada principio y se ejemplifica su aplicación en la biblioteca, destacando cómo estos enfoques mejoran la estructura y el funcionamiento del sistema.

Tabla 2: Otros principios de diseño implementados en BHAVRP.

Principio	Descripción	Ejemplo en BHAVRP
Ley de Deméter	Limita el conocimiento de un objeto sobre otros, promoviendo un bajo acoplamiento.	En <b>BHAVRP</b> , las clases operan de manera autónoma, llamando solo a métodos de su propia instancia, objetos pasados como argumentos o creados dentro de la clase, minimizando las interacciones innecesarias entre clases.
Principio de Hollywood	Las clases de alto nivel deben llamar a las clases de bajo nivel, no al revés.	En <b>BHAVRP</b> , las clases de alto nivel como <i>Controller</i> o <i>Problem</i> orquestan todo el procesamiento relacionado con las clases de bajo nivel con que interactúan directamente, favoreciendo un diseño limpio y desacoplado.
Diseñar hacia las interfaces	Programar contra interfaces o clases abstractas en lugar de implementaciones concretas.	<b>BHAVRP</b> utiliza interfaces para definir los métodos para la creación de cálculos de distancia aproximada, así como para la creación de los algoritmos de asignación. En cada caso existen clases que implementan estas interfaces y la lógica concreta. De manera absoluta se respeta el contrato que se establece con la interfaz y se evita el atarse a una implementación específica.
Once and Only Once Rule	Cada elemento de conocimiento debe tener una sola representación en el sistema.	La biblioteca <b>BHAVRP</b> asegura la centralización y el control unificado de los datos del problema <b>MDVRP</b> a través de la clase <i>Problem</i> , lo que permite gestionar los cambios de manera eficiente y evita la duplicación de responsabilidades en el sistema.

Los principios implementados en **BHAVRP** trabajan en sinergia para abordar la complejidad del diseño modular y la escalabilidad. La Ley de Deméter y el Principio de Bajo Acoplamiento reducen dependencias entre clases, facilitando la mantenibilidad. El Principio de *Hollywood* complementa "Diseñar hacia las interfaces" al promover la delegación y abstracción, asegurando que los módulos de bajo nivel no interfieran con la lógica central. Además, ejemplos como *IFactoryDistance* y *IFactoryAssignment* ilustran cómo las interfaces refuerzan la flexibilidad, logrando un sistema adaptable, cohesivo y alineado con buenas prácticas de diseño.

#### 1.2.4. Patrones de diseño implementados en BHAVRP

Los patrones de diseño **GoF** son soluciones probadas para problemas comunes que surgen en el diseño de software, y muchos de ellos están directamente relacionados con los principios **SOLID**. Por ejemplo, el patrón *Factory Method*, implementado en **BHAVRP**, ayuda a cumplir el Principio de Responsabilidad Única (**SRP**), al separar la creación de objetos de su uso.

Los patrones **GoF** son un conjunto de 23 patrones de diseño de software que fueron documentados en el libro "*Design Patterns: Elements of Reusable Object-Oriented Software*" por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, conocidos colectivamente como la "Banda de los Cuatro" (*Gang of Four, GoF*). Estos patrones se dividen en tres categorías principales: patrones creacionales, patrones estructurales y patrones de comportamiento [70].

- Los patrones creacionales proporcionan mecanismos de creación de objetos que incrementan la flexibilidad y la reutilización de código existente.
- Los patrones estructurales explican cómo ensamblar objetos y clases en estructuras más grandes a la vez que se mantiene la flexibilidad y eficiencia de la estructura.
- Los patrones de comportamiento se encargan de una comunicación efectiva y la asignación de responsabilidades entre objetos.

De todos los patrones de diseño propuestos por **GoF** este apartado se centra en un grupo. Estos patrones han sido implementados en el componente de software **BHAVRP**

por su relevancia y aplicabilidad a los problemas que aborda el proyecto. A continuación, se detallan los patrones seleccionados, explicando sus características, beneficios y ejemplos de uso.

El patrón *Singleton* garantiza que una clase tenga solo una instancia y proporciona un punto de acceso global a ella. Este patrón es útil en situaciones donde se necesita un control estricto sobre las instancias de una clase, como en la gestión de recursos compartidos. Sin embargo, su uso debe ser cuidadosamente considerado, ya que puede introducir acoplamiento global en la aplicación.

En el contexto de la situación que propone el componente **BHAVRP**, se hace necesario la existencia de una sola instancia de las clases *Controller* y *Problem*. En la Figura 12 se muestra un ejemplo de la implementación del patrón *Singleton* para la clase *Problem*.

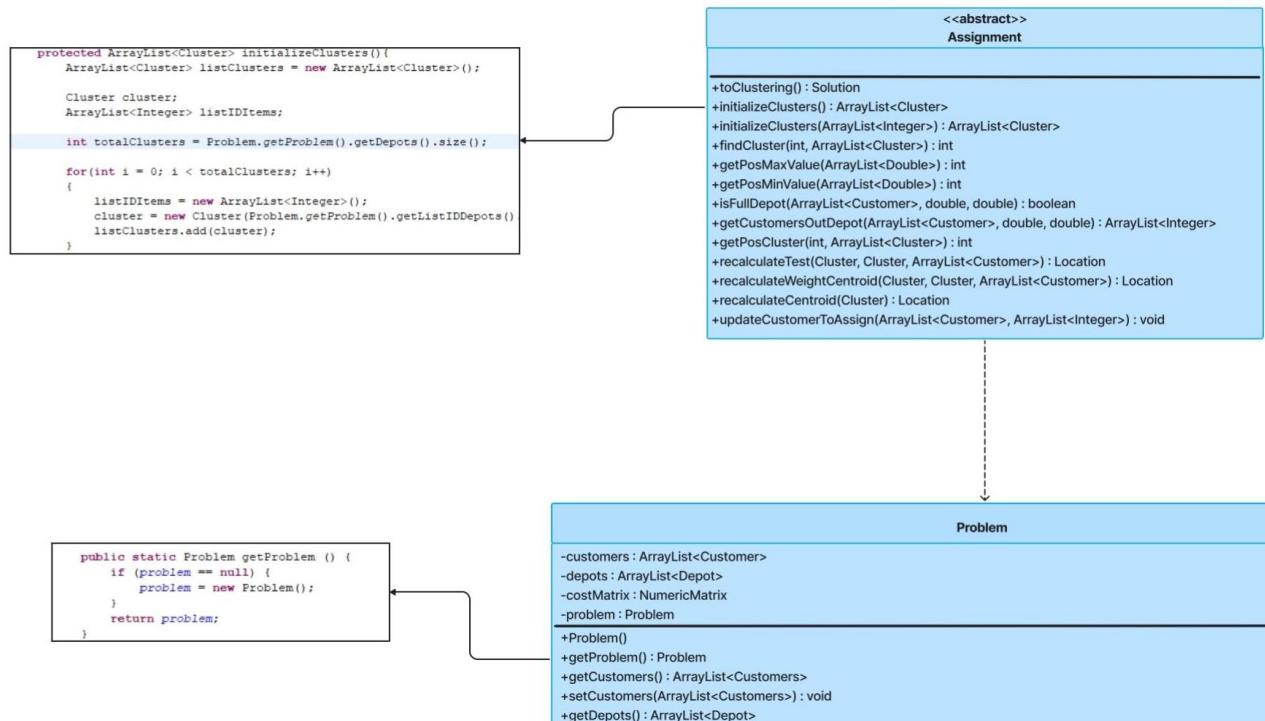


Figura 12: Patrón *Singleton* implementado en BHAVRP.

La clase *Problem* es responsable de manipular los datos que se requieren para la efectuar la asignación, y debido a esto, solamente una instancia de esta clase es la que debe estar ejecutando dicha operación. Por otra parte, la clase *Controller* se encarga de gestionar todo el proceso de asignación de clientes a depósitos, razón por la cual no

puede existir más de una instancia de la clase realizando esta tarea. Con la implementación del patrón *Singleton* en estas clases, cuya función consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella, queda solventado este problema.

Por otra parte, el patrón *Factory Method* define una interfaz para crear un objeto, pero permite a las subclases decidir qué clase instanciar. Este patrón delega la instanciación a las subclases y este patrón es particularmente útil en el desarrollo de software cuando se necesita una forma flexible de crear objetos sin especificar su clase exacta. Esto puede ser especialmente beneficioso en proyectos de desarrollo de software, donde la flexibilidad y la extensibilidad son importantes.

Para realizar la asignación de clientes a depósitos, **BHAVRP** cuenta con varios algoritmos de asignación. De ellos, el usuario debe seleccionar uno para efectuar la asignación de clientes a depósitos. Este proceso se realiza en tiempo de ejecución, y para ello se necesita del patrón *Factory Method*. En la Figura 13 se muestra un ejemplo de la implementación del patrón para los algoritmos de asignación.

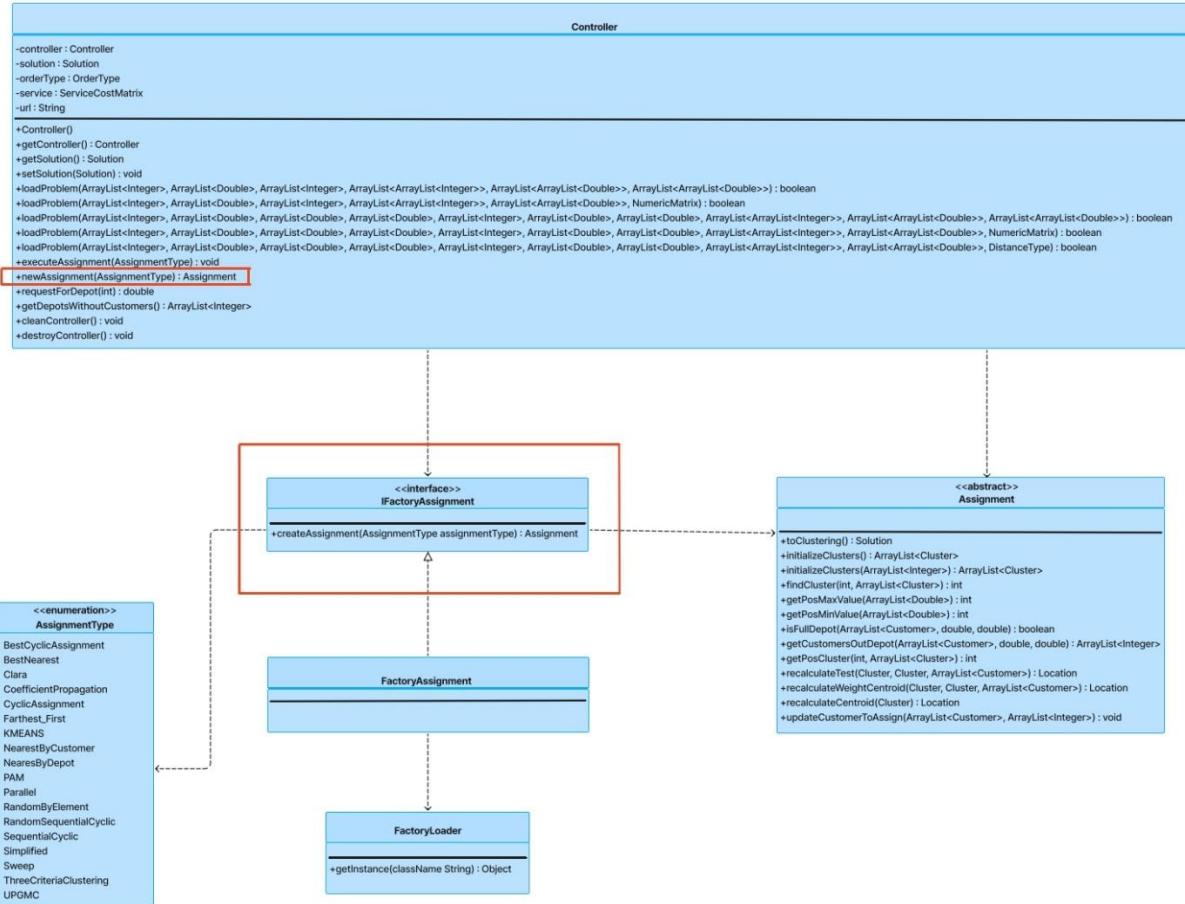


Figura 13: Patrón *Factory Method* implementado en BHAVRP.

Este patrón define una interfaz para la creación de un objeto, pero permite que las subclases decidan cuál de las clases instanciar. En concreto retorna una instancia de una o varias posibles clases, en dependencia de los datos que se le provea, sin conocer previamente las clases de objetos a crear. Para este caso, así como para la creación de la distancia usada en la construcción de la matriz de costo, se implementa este patrón, que permite realizar la carga dinámica de cualquier clase y facilita la incorporación de nuevas heurísticas de asignación.

Para complementar el análisis de patrones de diseño en **BHAVRP**, se podrían considerar otros patrones **GoF** que, aunque aún no están implementados, podrían aportar mejoras significativas. Por ejemplo, el patrón *Strategy* permitiría encapsular diferentes algoritmos de asignación, facilitando su intercambio sin modificar el código que los utiliza, promoviendo una mayor flexibilidad y escalabilidad. Por otro lado, el patrón *Template*

*Method* podría estandarizar los pasos comunes en la ejecución de los algoritmos, garantizando que las subclases implementen detalles específicos mientras mantienen una estructura coherente y reutilizable en los algoritmos. Además, los únicos patrones **GoF** implementados en **BHAVRP** son patrones creacionales, lo que indica una oportunidad para enriquecer su diseño.

Los patrones **GoF** se enfocan más en aspectos estructurales y de comportamiento en la arquitectura del sistema, mientras que los patrones **GRASP** (*General Responsibility Assignment Software Patterns*) se centran en la asignación adecuada de responsabilidades a las clases y objetos dentro del sistema [71]. La transición de los patrones **GoF** a **GRASP** se realiza cuando se busca una mejor asignación de responsabilidades en la implementación concreta del sistema.

A continuación, en la Tabla 3 se resumen los principales patrones **GRASP** implementados en **BHAVRP**, destacando su propósito general y cómo cada uno se aplica específicamente dentro del diseño de la biblioteca.

Tabla 3: Patrones de diseño GRASP implementados en BHAVRP.

Patrón	Criterio de evaluación	Grado de cumplimiento
Controlador	Una clase centralizada gestiona los eventos y actúa como intermediaria.	Total
	El patrón facilita pruebas y promueve modularidad.	Amplio
Experto en información	Las responsabilidades recaen en las clases que poseen la información necesaria.	Bajo
	Promueve cohesión y encapsulamiento.	Amplio
Alta cohesión	Las clases tienen responsabilidades relacionadas y específicas.	Bajo
Bajo acoplamiento	Las dependencias entre clases están minimizadas.	Amplio
	Se garantiza que los cambios en una clase no impacten directamente en otras.	Amplio
Creador	Las clases responsables crean instancias de las clases que necesitan.	Amplio
	La creación está alineada con el ciclo de vida lógico de las instancias.	Amplio

De la Tabla 3 se concluye que en **BHAVRP** se aplican estos patrones de manera efectiva para optimizar su arquitectura. El patrón Controlador se implementa en la clase *Controller*, que centraliza la gestión de eventos del sistema y facilita el desacoplamiento. Sin embargo, es posible mejorar la flexibilidad de la arquitectura evaluando si algunas clases abstractas se pueden convertir en interfaces, permitiendo así una mayor extensibilidad y reutilización. A través del patrón Experto en Información, clases como *Problem*, *Depot* y *Customer* gestionan su propia información, promoviendo cohesión y encapsulación. El patrón Alta Cohesión se observa en clases como *Cluster*, que limitan sus responsabilidades a funciones específicas, facilitando su reutilización. El patrón Bajo Acoplamiento reduce dependencias mediante componentes como la clase *Partitional*,

que centraliza métodos comunes. Finalmente, el patrón Creador asegura que objetos clave como *Customer* y *Depot* sean creados correctamente mediante la clase *Problem*.

La implementación adecuada de principios y patrones de diseño es esencial para crear sistemas bien estructurados y fáciles de mantener. En el caso de **BHAVRP**, la combinación de principios **SOLID**, patrones **GoF** y **GRASP** garantiza una arquitectura que no solo resuelve eficazmente el problema del **MDVRP**, sino que también ofrece un marco flexible para futuras mejoras y adaptaciones. Estos principios proporcionan la base necesaria para integrar nuevas funcionalidades y adaptarse a cambios tecnológicos o requisitos adicionales.

### 1.3. Bibliotecas que resuelven la fase de asignación del MDVRP

El campo de la optimización ha dado lugar al desarrollo de algunas bibliotecas en distintos lenguajes de programación, cada una con su propia gama de algoritmos y funcionalidades. A continuación, se lista una selección de bibliotecas destacadas que están especializadas en la resolución de **VRP** y sus variantes, entre las que se encuentra **MDVRP**. De cada una se presenta una breve descripción de sus características y aplicaciones.

- **JSprit** es una biblioteca de código abierto desarrollada en el lenguaje de programación Java para resolver el Problema del Agente Viajero (*Traveling Salesman Problem, TSP*) y variantes del Problema de Planificación de Rutas de Vehículos (*Vehicle Routing Problem, VRP*). Es un componente ligero, flexible y fácil de usar, basado en una única metaheurística multipropósito que actualmente resuelve una amplia gama de variantes del **VRP**, **TSP** y otros problemas de optimización, entre las variantes de **VRP** que puede resolver se encuentra el **MDVRP** [21]. Al ser compatible con técnicas que optimizan la asignación inicial de clientes, como el Algoritmo de Barrido (*Sweep Algorithm*), **JSprit** puede preasignar clientes a depósitos considerando su proximidad geográfica y las restricciones de capacidad de los depósitos. Posteriormente, esta asignación inicial puede optimizarse con su metaheurística integrada, un algoritmo de optimización basado en la búsqueda local, conocido principalmente como Búsqueda de Vecindad Variable (*Variable Neighborhood Search, VNS*) [72].

- *Google Operation Research Tools (OR-Tools)* es un paquete de software de código abierto desarrollado en C++, con versiones estables en Java, C# y Python, orientado a la optimización, diseñado para abordar problemas complejos en la planificación de rutas de vehículos, flujos de red, programación lineal y entera, y programación de restricciones. Aunque su enfoque principal es resolver problemas completos de **VRP**, permite personalizar soluciones para la fase de asignación mediante programación y heurísticas personalizadas. Entre sus métodos de solución se incluyen heurísticas como Ahorros (*Savings*) e Inserción (*Insertion*), así como metaheurísticas como Búsqueda Local (*Local Search, LS*) y estrategias avanzadas como **VNS**, que le permiten abordar eficientemente tanto la asignación como la planificación [17]. **OR-Tools** permite integrar algoritmos de agrupamiento como *K-means* o *DBSCAN* para preasignar clientes a depósitos. Además, se pueden implementar restricciones personalizadas en el modelo, como límites de capacidad o balance de carga, para ajustar la asignación inicial antes de proceder con la planificación de rutas.
- *Vehicle Routing Open-Source Optimization Machine (VROOM)* es un *solver* para problemas de planificación de rutas de vehículos desarrollado en el lenguaje de programación C++. Esta herramienta utiliza *Open Street Map (OSM)* [73] y/o *Open Source Routing Machine (OSRM)* [74] como *backend* para obtener rutas y devolver soluciones para las diferentes variantes del **VRP**. **VROOM** puede resolver varias de las variantes de los problemas de planificación de rutas de vehículos como: **MDVRP**, **CVRP**, **VRPTW** y **PDPTW** [18]. Además, utiliza algoritmos basados en distancias, evaluando la cercanía geográfica entre clientes y depósitos, para generar asignaciones iniciales que respeten las capacidades de los depósitos. La asignación puede optimizarse iterativamente dentro del mismo sistema antes de planificar las rutas.
- **Scikit-learn** es una biblioteca de aprendizaje automático en Python que, aunque no está diseñada específicamente para resolver problemas de optimización como el **MDVRP**, proporciona una amplia gama de algoritmos de agrupamiento y análisis de datos que pueden ser utilizados para abordar la fase de asignación en este tipo de problemas [75]. Algoritmos como *K-means*, *DBSCAN* y el

agrupamiento jerárquico son útiles para preprocesar los datos y asignar clientes a depósitos de manera eficiente, basándose en proximidad y restricciones de capacidad. Su flexibilidad y compatibilidad con otras bibliotecas de Python la convierten en una herramienta versátil para integrar técnicas de agrupamiento en soluciones personalizadas para el **MDVRP**.

- Capa de servicio para soluciones a Problemas de Planificación de Rutas de Vehículos (**CaSVRP**) [32] es una capa de servicios web dirigidos a la resolución mediante algoritmos metaheurísticos de diferentes variantes de problemas de planificación de rutas de vehículos. Entre las variantes del **VRP** que resuelve se halla **MDVRP**. Esta solución se compone de un módulo de optimización donde se utiliza la Biblioteca de Clases para la Integración de Algoritmos Metaheurísticos (**BiCIAM**) [76]. Además, emplea un módulo para la generación de soluciones iniciales empleando la Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos (**BHCVRP**) [32] y otro para la generación de nuevas soluciones a partir de otras soluciones haciendo uso del Componente para la Ejecución de Operadores de Mutación para Problemas de Planificación de Rutas de Vehículos (**LMOVVRP**) [77].
- Herramienta para la generación de rutas en el Problema del Transporte Obrero (**TransO**) es un software desarrollado en Java para la generación de recorridos en el Problema del Transporte Obrero. La herramienta implementa dos modelos de optimización combinatoria para la resolución de este problema y entre las variantes de **VRP** que resuelve destaca la variante con múltiples depósitos [34].
- Bibliotecas de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos (**BHCVRP**) es un componente que encapsula algoritmos heurísticos para la resolución de problemas de planificación de rutas de vehículos. Esta biblioteca implementa heurísticas de construcción para el **CVRP** y otras variantes de este problema como el **MDVRP**. Actualmente las soluciones obtenidas mediante estas heurísticas pueden ser utilizadas como solución final o como parte de una solución más compleja cuando se emplean las metaheurísticas [33].

La Tabla 4 presenta una comparación entre las bibliotecas antes mencionadas. Las filas de la tabla corresponden a cada biblioteca analizada, mientras que las columnas detallan aspectos como la implementación de la fase de asignación, heurísticas clásicas, algoritmos de agrupamiento, evaluación de resultados, lenguajes soportados, interoperabilidad y si el código es abierto. Es importante resaltar criterios como interoperabilidad, que mide qué tan fácilmente puede integrarse una biblioteca con otros sistemas, y evaluación de resultados, que refleja la capacidad de analizar la calidad de las soluciones obtenidas.

Tabla 4: Comparación de otras bibliotecas que resuelven MDVRP.

Bibliotecas	Características						
	Resuelve la fase de asignación para MDVRP	Implementan heurísticas clásicas	Implementan algoritmos de agrupamiento	Evaluación de resultados	Lenguajes soportados	Interoperabilidad	Código abierto
<b>BHAVRP</b>	Sí	Sí	Sí	No	Java	Baja	Sí
<b>JSprit</b>	Sí	No	No	No	Java	Limitada	Sí
<b>OR-Tools</b>	Sí	Sí	No	No	C++, Python, Java, C#	Alta	Sí
<b>VROOM</b>	Sí	Sí	No	No	C++	Alta	No
<b>Scikit-learn</b>	Sí	No	Sí	Sí	Python	Alta	Sí
<b>CaSVRP</b>	Sí	Sí	No	No	Java	Baja	No
<b>BHCVRP</b>	Sí	Sí	No	No	Java, Python	Baja	Sí

A partir del análisis comparativo de las bibliotecas presentadas en la Tabla 4, se pueden extraer las siguientes conclusiones:

- Todas las bibliotecas analizadas pueden adaptarse para abordar la fase de asignación de clientes a depósitos en problemas de planificación de rutas de vehículos con múltiples depósitos, gracias a las características y funcionalidades que ofrecen, lo que las convierte en herramientas relevantes para esta investigación.
- En cuanto a la implementación de heurísticas clásicas, a excepción de **JSprit** y **Scikit-learn**, todas las bibliotecas implementan heurísticas clásicas, lo que les permite abordar problemas de optimización de manera eficiente. Las herramientas que destacan por su enfoque en estas técnicas son: **BHAVRP**, **BHCVRP** y **CaSVRP**.
- Solo **BHAVRP** y **Scikit-learn** ofrecen algoritmos de agrupamiento, lo que los posiciona como herramientas especialmente útiles cuando las necesidades incluyen esta capacidad para el preprocesamiento o modelado del problema.
- Solo **Scikit-learn** incorpora funcionalidades explícitas para evaluar resultados, utilizando herramientas como métricas integradas, validación cruzada y visualizaciones específicas para analizar la calidad de las soluciones.
- **OR-Tools** se destaca en términos de lenguajes soportados, ya que es compatible con C++, Python, Java y C#, lo que proporciona una mayor flexibilidad para su integración en diversos entornos.
- La mayoría de las bibliotecas, con excepción de **VROOM** son de código abierto, lo que facilita su acceso y adaptación por parte de la comunidad. No obstante, **VROOM** se destaca por su alta interoperabilidad a pesar de no ser abierta.

Por tanto, aunque ninguna de estas bibliotecas está diseñada exclusivamente para resolver la fase de asignación en el **MDVRP**, excepto **BHAVRP**, todas tienen características que pueden aprovecharse para este propósito mediante adaptaciones personalizadas. Las capacidades de agrupamiento, la flexibilidad en restricciones y las herramientas de optimización son las claves para utilizarlas en este contexto, **OR-Tools** se distingue por su interoperabilidad y soporte en múltiples lenguajes, mientras que

**Scikit-learn** se presenta como una herramienta versátil para integrar técnicas de agrupamiento en soluciones personalizadas.

Además, el análisis revela que la mayoría de estas bibliotecas han sido desarrolladas en el lenguaje de programación Java. Sin embargo, la comunidad de Python continúa creciendo rápidamente, especialmente en áreas de inteligencia artificial y optimización, lo cual sugiere un potencial creciente para integrar y ampliar herramientas en Python que resuelvan el **MDVRP** de manera eficiente.

Como observación adicional destaca que **BHAVRP**, aunque tiene características avanzadas como algoritmos de agrupamiento e implementaciones específicas de matrices, podría beneficiarse al incluir funcionalidades para la evaluación de resultados. Además, su baja interoperabilidad y el uso exclusivo de Java limitan su alcance frente a otras bibliotecas más adaptables.

#### 1.4. Bibliotecas especializadas en el cálculo de distancias

El cálculo de distancias es un componente esencial en problemas de optimización de rutas como el **MDVRP**. La precisión y eficiencia en la generación de matrices de costo dependen en gran medida de la estrategia utilizada, ya sea a través de distancias reales, como las calculadas por servicios de mapas, o aproximadas, que ofrecen una solución simplificada pero menos precisa. En esta sección, se exploran bibliotecas especializadas en el cálculo de distancias, evaluando sus capacidades, enfoques y posibles aplicaciones en contextos como el de **BHAVRP** [78]. Esto incluye herramientas que emplean servicios como *Open Street Map (OSM)* [73] para obtener distancias reales y aquellas que recurren a modelos heurísticos o simplificados para cálculos aproximados.

Las distancias aproximadas son una alternativa útil cuando se busca equilibrar la precisión con la eficiencia en problemas de optimización de rutas como el **MDVRP**. A diferencia de las distancias reales, que dependen de datos geoespaciales exactos y cálculos detallados, las distancias aproximadas utilizan modelos simplificados para estimar el costo de las rutas de manera más rápida. Este enfoque es particularmente valioso en contextos donde el tiempo de computación y los recursos son limitados, permitiendo obtener soluciones de calidad sin necesidad de realizar cálculos

exhaustivos. Existen diferentes métodos y técnicas para calcular distancias aproximadas, que varían en términos de complejidad y aplicabilidad. Entre los métodos más comunes se incluyen:

- La distancia *Euclidean* calcula la distancia recta entre dos puntos, ignorando cualquier red vial. Esta distancia es útil cuando se requiere una estimación rápida y no es necesario considerar rutas específicas o la infraestructura vial [25].
- La distancia *Manhattan* se utiliza en sistemas de cuadrícula, calculando la suma de las distancias absolutas entre las coordenadas. La relevancia de esta distancia es evidente en entornos urbanos o cuando se necesita un cálculo simplificado, similar a moverse por una ciudad en forma de cuadrícula [27].
- La distancia *Haversine* calcula la distancia más corta entre dos puntos en una esfera, considerando la curvatura de la Tierra. Las coordenadas de latitud y longitud son utilizadas en *Haversine* y es especialmente útil para cálculos geográficos precisos en sistemas de posicionamiento global (**GPS**). Esta fórmula es muy precisa cuando se requiere determinar distancias entre ubicaciones a nivel global, como en aplicaciones de navegación o planificación de rutas [26].
- La distancia *Chebyshev* mide la mayor diferencia absoluta en cualquiera de las dimensiones entre dos puntos, sin tener en cuenta las diagonales. La aplicación de esta distancia es común en entornos de cuadrícula, donde se permite movimiento en horizontal, vertical o diagonal, como en juegos de ajedrez o sistemas de rutas de red simples. La distancia *Chebyshev* tiene un enfoque más simple y rápido cuando no se necesita precisión geográfica, pero sí eficiencia en movimientos restringidos [28].

Los métodos de cálculo de distancias aproximadas ofrecen soluciones rápidas y eficientes en comparación con los cálculos exactos basados en redes viales. Para implementar estos cálculos, existen varias bibliotecas y herramientas que facilitan su integración en aplicaciones de optimización. Algunas de estas herramientas son:

- **SciPy** es una biblioteca ampliamente utilizada en el ámbito científico y de ingeniería, lo que la hace ideal para cálculos rápidos y eficientes en aplicaciones de optimización de rutas [79].

- **Geopy** es una biblioteca de Python que facilita el cálculo de distancias geográficas utilizando la fórmula *Haversine*, entre otras. Esta biblioteca es especialmente útil para trabajar con distancias geográficas entre coordenadas de latitud y longitud, y soporta tanto distancias reales como aproximadas, ofreciendo soluciones rápidas y sencillas para proyectos que requieren precisión geográfica sin necesidad de datos de redes viales [80].
- **Scikit-learn** también ofrece herramientas para el cálculo de distancias en el contexto de *clustering* y clasificación, como la distancia de *Manhattan* y la *Euclidean*. Esta biblioteca propone una implementación eficiente y se adapta bien a problemas donde se deben considerar múltiples medidas de similitud y distancia entre puntos en espacios multidimensionales [75].

Por consiguiente, las distancias aproximadas ofrecen una solución efectiva para la optimización de rutas, especialmente cuando el tiempo de cálculo es crucial y los recursos son limitados. Los métodos antes mencionados proporcionan diversas alternativas, cada uno adaptándose a necesidades específicas de precisión y eficiencia.

Para el cálculo de distancias reales, *Open Street Map (OSM)* proporciona una plataforma colaborativa que ofrece datos de mapas gratuitos y abiertos. Estos datos incluyen redes viales, información topológica, y etiquetas que describen características de carreteras, como velocidad máxima o tipo de vía. Su enfoque basado en contribuciones comunitarias permite que los datos sean continuamente actualizados y mejorados. Además, **OSM** se utiliza como base en muchas herramientas para calcular rutas y distancias, ya que los datos pueden procesarse para obtener distancias reales basadas en rutas optimizadas por diferentes criterios, como tiempo o costo [81].

Para abordar el cálculo de distancias reales en **BHAVRP**, es fundamental explorar herramientas existentes que ofrezcan precisión geográfica y eficiencia operativa. Actualmente, **BHAVRP** se apoya en cálculos aproximados siendo una limitante en cuanto a la exactitud en la asignación de clientes a depósitos. Entre las herramientas destacadas en este ámbito se encuentran *Google Distance Matrix API*, *GraphHopper* y *Mapbox Directions*.

- *Google Distance Matrix API* calcula distancias y tiempos de viaje reales entre puntos específicos utilizando datos de *Google Maps*. Esta API es compatible con varios modos de transporte e integra información de tráfico en tiempo real para ofrecer estimaciones precisas y contextualmente relevantes. En su funcionamiento, las ubicaciones se geocodifican a coordenadas de latitud y longitud. Las solicitudes incluyen parámetros como el modo de viaje y restricciones de ruta [29].
- *GraphHopper* es una biblioteca de código abierto para la planificación de vehículos sobre redes viales que utiliza datos de **OSM**. *GraphHopper* es conocida por su rapidez y flexibilidad, permitiendo personalizar las rutas generadas mediante perfiles de vehículos y restricciones específicas. Además, se puede ejecutar sin conexión, lo que la hace adecuada para aplicaciones que necesitan control total sobre la planificación de rutas y la capacidad de operar en entornos sin acceso a Internet [30].
- *Mapbox Directions API* proporciona cálculos de rutas y distancias utilizando datos de *Mapbox* siendo una solución robusta para aplicaciones que requieren distancias reales en múltiples modos de transporte, incluyendo caminata, ciclismo y conducción. La API permite optimizar rutas para varias paradas y ofrece opciones para evitar restricciones específicas como carreteras cerradas [82].

Las herramientas y APIs mencionadas ofrecen soluciones para calcular distancias reales basadas en datos geoespaciales precisos. Estas herramientas proporcionan un alto nivel de precisión y flexibilidad, lo que las convierte en opciones valiosas para aplicaciones que requieren una optimización detallada de rutas, como el caso de **BHAVRP**. No obstante, existen enfoques igualmente eficientes, como el uso de distancias aproximadas, que, aunque menos precisas, permiten realizar cálculos rápidos y adecuados para contextos donde la eficiencia es crucial.

### 1.5. Limitaciones y deficiencias que presenta la versión actual de BHAVRP

El componente **BHAVRP**, en su versión actual, presenta ciertas limitaciones en términos de diseño, interoperabilidad y eficiencia que podrían mejorarse para facilitar su

integración en entornos tecnológicos modernos y maximizar su rendimiento. A continuación, se presentan los aspectos clave a considerar.

- Distribución de responsabilidades y acoplamiento entre paquetes: dado que la mayoría del trabajo recae en la capa de aplicación, es posible que haya áreas donde se pueda mejorar la distribución de las responsabilidades. Al mejorar este aspecto se reduce el acoplamiento entre paquetes, lo cual es clave para mejorar la flexibilidad y escalabilidad de la biblioteca en entornos más complejos.
- Interoperabilidad limitada: actualmente, **BHAVRP** se ha utilizado como componente de un software llamado **TransO**, que emplea la biblioteca para resolver la asignación de clientes a depósitos dentro del proceso de solución de un Problema de Planificación de Rutas de Vehículos. Aunque la biblioteca pueda integrarse con otras aplicaciones desarrolladas en Java, su capacidad de comunicación está restringida a sistemas basados en este lenguaje. Esto limita su interoperabilidad con aplicaciones desarrolladas en otros lenguajes, lo cual impide una mayor flexibilidad para integrarse con sistemas más diversos.
- Precisión de los cálculos de distancia: otro aspecto a considerar es la precisión de los cálculos de distancia. Actualmente, **BHAVRP** se basa en cálculos de distancias aproximadas en lugar de distancias reales, lo cual puede afectar la exactitud en la asignación de clientes a depósitos.
- Evaluación de la plataforma de desarrollo: si bien **BHAVRP** está completamente implementado en Java, es importante considerar si su actual implementación satisface plenamente las necesidades a largo plazo en términos de mantenibilidad, integración con otros sistemas y escalabilidad. Aunque Java ha demostrado ser una opción confiable, es fundamental seguir evaluando el entorno tecnológico.
- Uso de componentes personalizados: durante el desarrollo de **BHAVRP** fue necesario crear un componente específico para manejar ciertas funcionalidades que no estaban disponibles en las bibliotecas del entorno original. Tal es el caso de libMatrix, una biblioteca en Java desarrollada para realizar cálculos matriciales, dado que en el momento de su creación no existían bibliotecas disponibles que cubrieran las necesidades del proyecto.

- Manejo de excepciones: **BHAVRP** carece de un manejo robusto de excepciones, lo cual afecta su fiabilidad en escenarios complejos. Implementar un manejo de excepciones más robusto sería un paso importante para aumentar la estabilidad y el comportamiento predecible de la biblioteca.
- Exportación de resultados: Actualmente, **BHAVRP** no cuenta con funcionalidades para exportar los resultados generados en formatos estándar, como CSV. Esto limita la capacidad de integrar los resultados con herramientas externas de análisis o visualización, restringiendo su aplicabilidad en entornos más amplios donde la interoperabilidad de datos es esencial.

Con la migración de **BHAVRP** a Python, se abren nuevas posibilidades para resolver las limitaciones mencionadas. Al aprovechar bibliotecas de código abierto más robustas y eficientes, se puede mejorar tanto el mantenimiento como la ampliación del proyecto. En términos de distribución de responsabilidades y acoplamiento entre paquetes, la reestructuración del código en Python permitiría una mejor organización y una reducción del acoplamiento, lo que favorecería la flexibilidad y escalabilidad del sistema. Para resolver la interoperabilidad limitada, Python facilita la integración con otras plataformas, lo que permitiría la comunicación entre **BHAVRP** y sistemas desarrollados en otros lenguajes. En cuanto a la precisión de los cálculos de distancia, Python cuenta con bibliotecas como *OSRM-py* [83] y *osmnx* [81] que permiten calcular rutas y distancias reales, sustituyendo las distancias aproximadas y mejorando la exactitud en la asignación de clientes a depósitos. Asimismo, la lógica previamente encapsulada en el paquete *distance* podría simplificarse al aprovechar **SciPy**, una biblioteca consolidada que ofrece algoritmos avanzados y optimizados, lo que reduciría la complejidad interna del proyecto. Además, la migración a Python también permitiría evaluar la plataforma de desarrollo de manera más efectiva, dado que Python ofrece un entorno más flexible, con una gran comunidad y múltiples opciones para escalar y mantener el sistema. Finalmente, los componentes personalizados de **BHAVRP**, como la biblioteca *libMatrix*, podrían ser reemplazados por la ampliamente utilizada biblioteca *numpy* [84], que ofrece cálculos matriciales optimizados y de alto rendimiento. Con estas mejoras, **BHAVRP** podría superar sus limitaciones actuales, ser eficiente y fácil de mantener, integrándose mejor en entornos tecnológicos modernos.

Además de estas mejoras inmediatas, la migración a Python y la actualización de la versión en Java abren las puertas a nuevas funcionalidades, como la incorporación de evaluaciones más detalladas para analizar el rendimiento de los algoritmos. Otro aspecto a considerar es implementar opciones para la entrada de datos en formatos estándar como XML y JSON, para facilitar su integración con sistemas modernos. Asimismo, se puede añadir soporte para guardar los resultados en formatos reutilizables, como archivos CSV, que permiten realizar análisis estadísticos o pruebas de validación más completas.

## 1.6. Conclusiones parciales

A partir del análisis realizado en este capítulo sobre **BHAVRP**, su arquitectura y principios de diseño, se concluyen los siguientes aspectos clave:

- **BHAVRP** es una herramienta que facilita la resolución de la fase de asignación en problemas de planificación de rutas de vehículos con múltiples depósitos, al implementar una variedad de algoritmos de asignación y técnicas de agrupamiento. Esta capacidad de integración permite adaptar la biblioteca a diversos escenarios de asignación en el **MDVRP**.
- Los principios y patrones de diseño implementados en **BHAVRP**, como el uso de interfaces y clases bien definidas, optimizan su organización interna y la eficiencia en el proceso de asignación.
- El análisis comparativo con otras bibliotecas demuestra que **BHAVRP** tiene un potencial significativo para fortalecerse mediante la incorporación de enfoques modernos y tecnologías emergentes. En particular, migrar a lenguajes como *Python* podría mejorar su accesibilidad, eficiencia e integración en ecosistemas más amplios de optimización y planificación logística.
- Aunque **BHAVRP** presenta una base sólida para resolver la fase de asignación, se identificaron limitaciones que afectan su eficiencia y aplicabilidad en escenarios más amplios. La identificación de estas áreas de mejora, como la migración a otro lenguaje, una mayor interoperabilidad y el cálculo de distancias reales, proporciona un camino claro para futuros desarrollos.

## Capítulo 2: Diseño de la nueva versión de BHAVRP

### 2.1. Introducción

En este capítulo se aborda el diseño de la nueva versión de **BHAVRP**. Se detallan las modificaciones implementadas en su arquitectura, con el objetivo de optimizar su funcionalidad y adaptabilidad a diferentes entornos tecnológicos.

La sección 2.2 presenta las modificaciones realizadas en la arquitectura de **BHAVRP**, explicando como estas contribuyen a mejorar la modularidad y escalabilidad del sistema. En la subsección 2.2.1 se detallan las deficiencias identificadas en la arquitectura original y las soluciones implementadas para abordarlas, estableciendo una relación directa entre las decisiones de diseño y los problemas resueltos. La subsección 2.2.2 describe los paquetes y funcionalidades generales que abarcan aspectos comunes de la arquitectura. La subsección 2.2.3 profundiza en las características específicas para ambas versiones, destacando los cambios realizados para aprovechar las fortalezas de este lenguaje.

La sección 2.3 detalla los principios de diseño aplicados en la nueva versión, alineados con buenas prácticas de ingeniería de software para garantizar un sistema robusto. Por otra parte, la sección 2.4 explora los patrones de diseño adoptados para estructurar y organizar los componentes de **BHAVRP**, justificando su elección en función de las necesidades del proyecto.

### 2.2. Modificaciones en la arquitectura de BHAVRP

La Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos (**BHAVRP**) es una herramienta desarrollada inicialmente en el lenguaje de programación Java, como parte del proyecto de Optimización y Metaheurísticas de la Facultad de Ingeniería Informática de la CUJAE [23] [24]. Esta herramienta fue concebida con el propósito de encapsular el comportamiento de diversos algoritmos aplicables en la fase de asignación de clientes a depósitos dentro del **MDVRP**.

Sin embargo, como fue analizado previamente en la sección 1.5, la versión actual de **BHAVRP** presenta ciertas deficiencias en su diseño y limitaciones en su funcionalidad que pueden restringir su desempeño en entornos tecnológicos modernos.

- Alto acoplamiento: los paquetes y clases dependen excesivamente entre sí, reduciendo la flexibilidad y dificultando la incorporación de nuevos algoritmos. Sin embargo, esta implementación deriva en un diseño monolítico, donde todos los algoritmos tienen acceso a métodos irrelevantes para su propósito específico.
- Falta de interoperabilidad: restricciones para integrarse con sistemas desarrollados en lenguajes como Python, limitando su adopción en contextos diversos.
- Organización deficiente del código: los métodos comunes y específicos se mezclaban en clases genéricas.
- Manejo inadecuado de excepciones: el sistema no cuenta con un tratamiento robusto para manejar errores, lo que podría ocasionar interrupciones inesperadas.
- Ausencia de persistencia de resultados: **BHAVRP** no ofrece una funcionalidad para guardar los resultados generados, limitando su utilidad en análisis posteriores o comparativas.
- Ausencia de persistencia para resultados: limita la utilidad de **BHAVRP** en análisis posteriores o comparativas al no contar con funcionalidades para guardar resultados generados.
- Cálculo de distancias reales: el sistema no incorpora el uso de la métrica de distancias reales, lo que permite resolver problemas con mayor aplicabilidad en contextos reales.
- Limitaciones en el uso de patrones de diseño: el patrón *Singleton*, aunque efectivo para garantizar una única instancia, puede generar restricciones en escenarios donde se requiere manejar múltiples instancias de manera concurrente. Sería necesario evaluar su impacto y considerar alternativas en el diseño del software.

Con el objetivo de optimizar la estructura del código, se implementa una reestructuración basada en principios de diseño orientado a objetos, distribuyendo las responsabilidades de manera modular. Entre los principales cambios realizados se pueden mencionar:

- Definición de interfaces: se introducen interfaces que establecen contratos claros para el comportamiento de las implementaciones, favoreciendo un diseño orientado a interfaces y asegurando una mayor extensibilidad. En los diagramas

de clases de las Figura 18, Figura 19, Figura 20 y Figura 21 se evidencia este diseño.

- Creación de nuevas clases abstractas: estas clases agrupan métodos comunes que comparten un propósito específico, sirviendo como base para especializaciones posteriores. Esta implementación se evidencia en las Figura 18 y Figura 19.
- Implementación de nuevos patrones de diseño: se reorganiza la estructura de las clases aplicando patrones que mejoran la reutilización, flexibilidad y claridad del código, como se puede apreciar en los epígrafes 2.3 y 2.4.
- Reorganización basada en comportamientos comunes: el código se reestructura para identificar y agrupar características compartidas entre diferentes componentes, lo que facilita la evolución del sistema y reduce la duplicidad.

Además, se incluye la capacidad de trabajar con distancias reales, lo que aumenta la aplicabilidad del sistema en contextos reales. A pesar de estos avances, se reconocen otras deficiencias que serán abordadas en trabajos futuros, como el manejo de excepciones, la persistencia de resultados en diferentes formatos, y la incorporación de métricas para evaluar la calidad de las soluciones generadas.

Adicionalmente, en la actualización de la versión en Java de **BHAVRP**, la lógica del paquete *distance* se separó y encapsuló en una biblioteca externa, lo que permitió reducir la complejidad interna del proyecto y mejorar su modularidad. Sin embargo, en la versión en Python se optará por utilizar **SciPy** [79], una biblioteca ampliamente consolidada por la comunidad, que ofrece algoritmos optimizados y confiables.

La migración de **BHAVRP** a Python permite resolver diversas limitaciones del sistema actual. Al aprovechar las ventajas de Python, como su amplio ecosistema de herramientas de código abierto y su flexibilidad, es posible mejorar la precisión de los cálculos, optimizar la interoperabilidad con otros sistemas y facilitar la integración con plataformas externas. Además, la reestructuración del código permite una distribución más eficiente de responsabilidades y reduce el acoplamiento entre componentes, lo que mejora la escalabilidad y el mantenimiento del sistema a largo plazo. Esta transición no solo aborda las deficiencias identificadas, sino que también incrementa el alcance de

**BHAVRP**, permitiéndole adaptarse a una variedad más amplia de escenarios y necesidades.

### 2.2.1. Nuevo diseño arquitectónico de BHAVRP

En esta subsección, se presenta la nueva arquitectura propuesta para **BHAVRP**, incluye un análisis de los cambios implementados. Además, se explora cómo estos ajustes contribuyen a mejorar tanto la funcionalidad de la biblioteca como su adaptabilidad a entornos tecnológicos modernos. La nueva arquitectura de **BHAVRP** ha sido diseñada para ser compatible con los lenguajes de programación Java y Python, lo que optimiza su flexibilidad y alcance en ambos entornos.

El diseño se basa en un enfoque modular y de múltiples capas, con el objetivo de mejorar la organización, escalabilidad y flexibilidad del sistema. Este rediseño contempla una clara separación de responsabilidades entre sus componentes principales, reduciendo el acoplamiento y facilitando futuras extensiones. Para reflejar los cambios realizados, se emplean colores diferentes: verde para los componentes nuevos, morado para los que han sufrido modificaciones y azul para los que permanecen sin cambios.

Además, se ha introducido un nivel adicional de abstracción en la gestión de datos y cálculos, optimizando el procesamiento de información y habilitando la integración con herramientas externas. La Figura 14 ilustra la estructura y organización de las capas de la nueva arquitectura de **BHAVRP**, destacando el rol de cada una dentro del sistema.

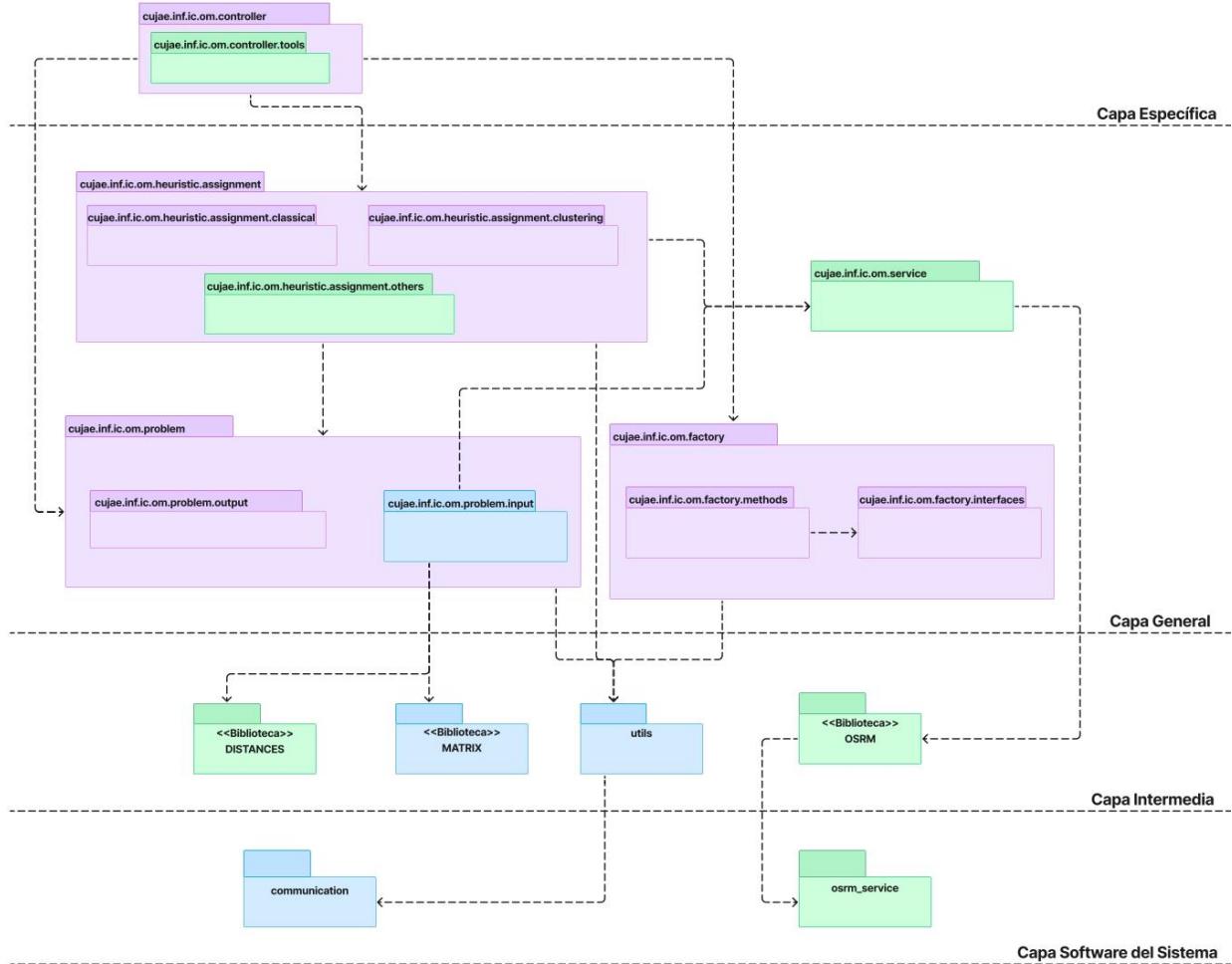


Figura 14: Arquitectura de la nueva versión de BHAVRP.

A continuación, se describen los paquetes y las funcionalidades generales que forman parte de la arquitectura, destacando su propósito y rol dentro del sistema.

### 2.2.2. Paquetes y funcionalidades generales

En la arquitectura de la biblioteca, los aspectos comunes están centralizados en la capa general, que actúa como un puente entre los componentes específicos y las funcionalidades de soporte. Esta organización permite reutilizar elementos clave, estandarizar procesos y facilitar la extensión de la biblioteca sin modificar el núcleo. Los paquetes generales gestionan tareas como el cálculo de distancias, el almacenamiento de datos y la integración con servicios externos.

La capa específica contiene el paquete encargado de gestionar las funcionalidades de la biblioteca. Su objetivo es coordinar la lógica específica de las operaciones necesarias para resolver la fase de asignación de un **MDVRP**. Dentro de esta capa, destaca la clase controladora de la biblioteca.

El diagrama de clases que se muestra en la Figura 15 refleja las clases del paquete *controller*.

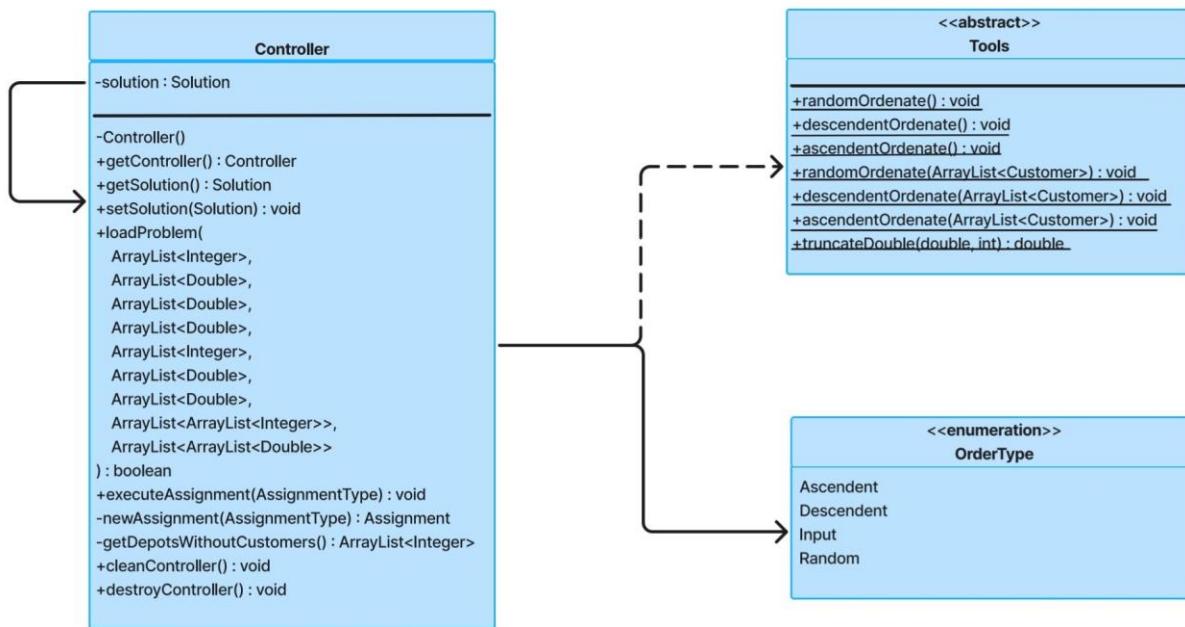


Figura 15: Diagrama de clases de los paquetes *controller* y *tools* en la nueva versión.

En la Tabla 5 se describen las clases que contienen los paquetes *controller* y *tools*.

Tabla 5: Descripción de las clases que integran los paquetes *controller* y *tools*.

Clase	Descripción
<i>OrderType</i>	Enumerado que define los tipos de orden para las demandas de los clientes en el problema. Este orden influye en el resultado de la asignación en algunos algoritmos, ya que determina cómo se organizan las demandas antes de ser asignadas. El tipo de orden afecta la eficiencia y calidad de la solución, y se configura en el <i>Controller</i> para adaptarse a los requisitos de cada algoritmo.
<i>Tools</i>	Clase que proporciona métodos para realizar operaciones relacionadas con el ordenamiento de las capacidades de la flota de vehículo de los depósitos y las demandas de los clientes. Además, brinda otros métodos que permiten la manipulación de las matrices de costos y el truncamiento de valores decimales.
<i>Controller</i>	Clase controladora principal encargada de coordinar las interacciones del sistema.

La capa general está compuesta por paquetes necesarios para el funcionamiento de las heurísticas y la modelación de los datos del **MDVRP** que se desea resolver. A continuación, para cada uno de los paquetes contenidos en esta capa se presenta su diagrama de clases correspondiente y una breve descripción de los elementos participantes. Es relevante mencionar que la estructura general de los paquetes fue reorganizada para mejorar la modularidad y simplificar su integración en las distintas versiones del sistema, mientras que las clases individuales se mantuvieron sin cambios significativos.

El diagrama de clases que se muestra en la Figura 16 refleja las clases del paquete *problem*. Este paquete contiene las clases que modelan un **MDVRP**.

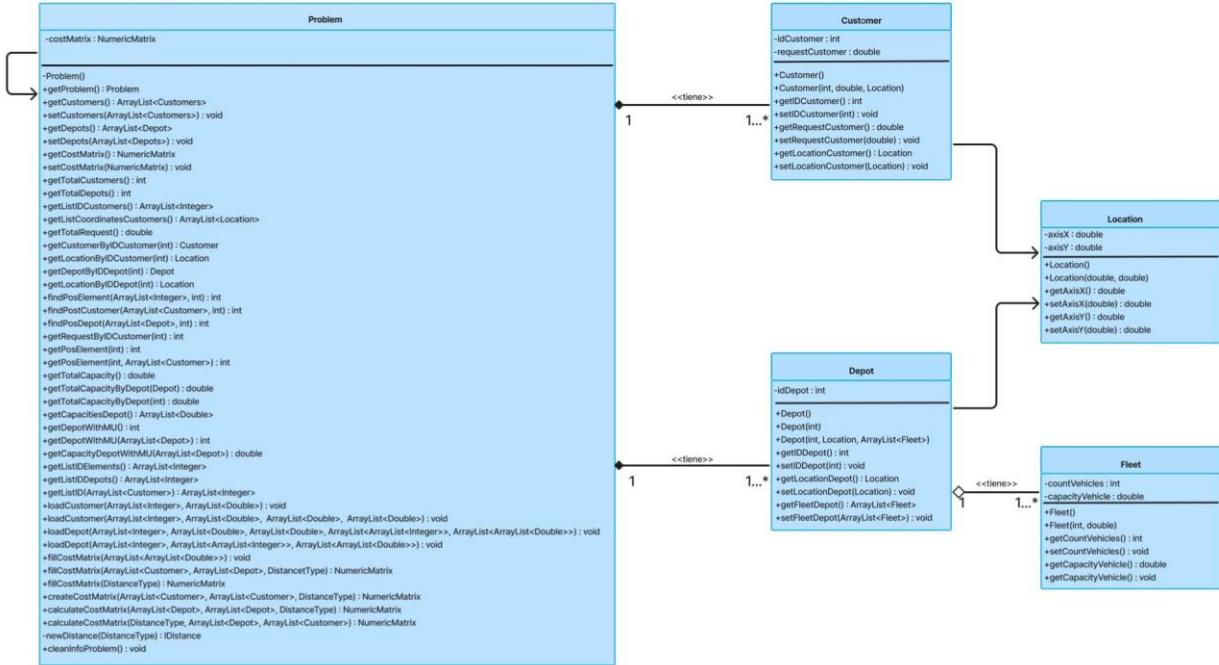


Figura 16: Diagrama de clases del paquete *problem* en la nueva versión.

En la Tabla 6 se describen las clases que integran el paquete *problem*.

Tabla 6: Descripción de las clases que integran el paquete *problem*.

Clases	Descripción
Location	Clase que modela una ubicación geográfica, utilizada para definir las posiciones de los clientes y los depósitos dentro del problema <b>MDVRP</b> .
Customer	Clase que representa un cliente en el <b>MDVRP</b> , con atributos como la ubicación y la demanda, que son necesarios para la asignación.
Fleet	Clase que representa la flota de vehículos que se utilizan para el transporte de clientes en el <b>MDVRP</b> , utilizada para modelar la cantidad y la capacidad de los vehículos.
Depot	Esta clase representa un depósito en el <b>MDVRP</b> , incluyendo su ubicación, capacidad y la flota de vehículos asociada.
Problem	Clase principal que modela el problema <b>MDVRP</b> , gestionando la información global del problema, como los clientes y depósitos.

## Diagrama de clases

El diagrama de clases que se muestra en la Figura 17 refleja las clases del paquete *solution*. Su objetivo es gestionar y representar las soluciones generadas por los algoritmos de asignación. Aunque no se observan cambios sustanciales en las clases de ese paquete, ha sufrido una reubicación dentro del paquete *problem*, siguiendo la ruta *problem.output*.

## Diagrama de clases

Figura 17 refleja las clases del paquete *solution*. Su objetivo es gestionar y representar las soluciones generadas por los algoritmos de asignación. Aunque no se observan cambios sustanciales en las clases de ese paquete, ha sufrido una reubicación dentro del paquete *problem*, siguiendo la ruta *problem.output*.

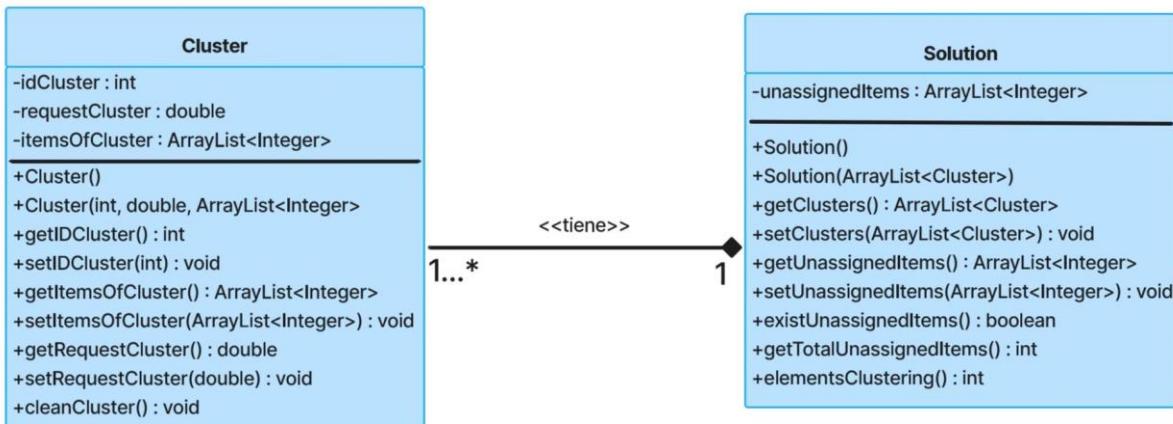


Figura 17: Diagrama de clases del paquete *output* en la nueva versión.

En la Tabla 7 se describen las clases que integran el paquete *output*.

Tabla 7: Descripción de las clases que integran el paquete *output*.

Clase	Descripción
<i>Cluster</i>	Esta clase representa un conjunto de clientes asignados a un depósito. Cada <i>Cluster</i> contiene información sobre el depósito que representa, los clientes asignados y la demanda total cubierta.
<i>Solution</i>	Esta clase modela una lista de objetos <i>Cluster</i> , representando la solución completa generada por los algoritmos de asignación. Se utiliza para almacenar y manipular las soluciones obtenidas en cada ejecución.

El paquete *assignment* se reorganiza con el objetivo de optimizar la estructura del código, se implementa una reestructuración basada en principios de diseño orientado a objetos, distribuyendo las responsabilidades de manera modular. Entre los principales cambios realizados se pueden mencionar:

- Se introduce la interfaz *IAssignment*, que define el contrato único *toClustering*, común a todas las implementaciones del paquete, que asegura un diseño orientado a interfaces y con mayor extensibilidad.
- Se refactoriza la clase abstracta *Assignment* para contener únicamente métodos comunes a todas las implementaciones.
- Se crean las clases abstractas especializadas como *Heuristic* y *Clustering*, que agrupan métodos específicos para heurísticas clásicas y algoritmos de agrupamiento, respectivamente.
- Se subclasifican de heurísticas clásicas en dos ramas: *ByUrgency*, que agrupa las heurísticas basadas en urgencia e incluye interfaces adicionales para variaciones en la implementación de métodos comunes que difieren en detalles específicos, y *ByNotUrgency*, diseñada para heurísticas cíclicas y basadas en clústeres.
- Se crea una nueva clasificación llamada *Others*, que alberga las heurísticas desarrolladas en el proyecto de Optimización y Metaheurísticas de la Facultad de Ingeniería Informática de la Universidad Tecnológica de La Habana “José Antonio Echeverría” (CUJAE) [23] [24]. Esta categoría se divide en dos grupos principales: los algoritmos basados en distancias y los algoritmos inspirados en la cíclica descrita en la literatura.
- Los algoritmos de agrupamiento se jerarquizan en dos subclases: *Hierarchical*, enfocada en algoritmos jerárquicos, y *Partitional*, que aborda algoritmos particionales. Esta última se divide en las subclases *ByMedoids* y *ByCentroids*, especializadas en algoritmos basados en medoides, como *PAM* y *Clara*, y algoritmos basados en centroides, como *K-means* y *Farthest First*, respectivamente.

Este rediseño ofrece varias ventajas significativas:

- Mejora la cohesión y modularidad al centrar cada clase en responsabilidades específicas, evitando la sobrecarga funcional.
- Reduce el acoplamiento al limitar el acceso de las implementaciones a los métodos relevantes para su funcionalidad.

- Mejora la legibilidad y mantenibilidad del código, gracias a una organización más clara y modular.
- Promueve alta cohesión y bajo acoplamiento según los patrones **GRASP**.

El diagrama de clases que se muestra en la Figura 18 refleja las clases del paquete *classical* dentro del paquete *assignment*. Este paquete encapsula las heurísticas clásicas utilizadas en la fase de asignación. Actualmente, las heurísticas que permanecen en *classical* representan métodos clásicos reconocidos en la literatura y se encuentran clasificadas en tres categorías principales: heurísticas basadas en clústeres, por urgencia y cíclicas. Esta clasificación refleja la jerarquía introducida, donde las clases abstractas especializadas, como *ByUrgency* y *ByNotUrgency*, agrupan las características comunes y permiten la extensión en la implementación de métodos particulares.

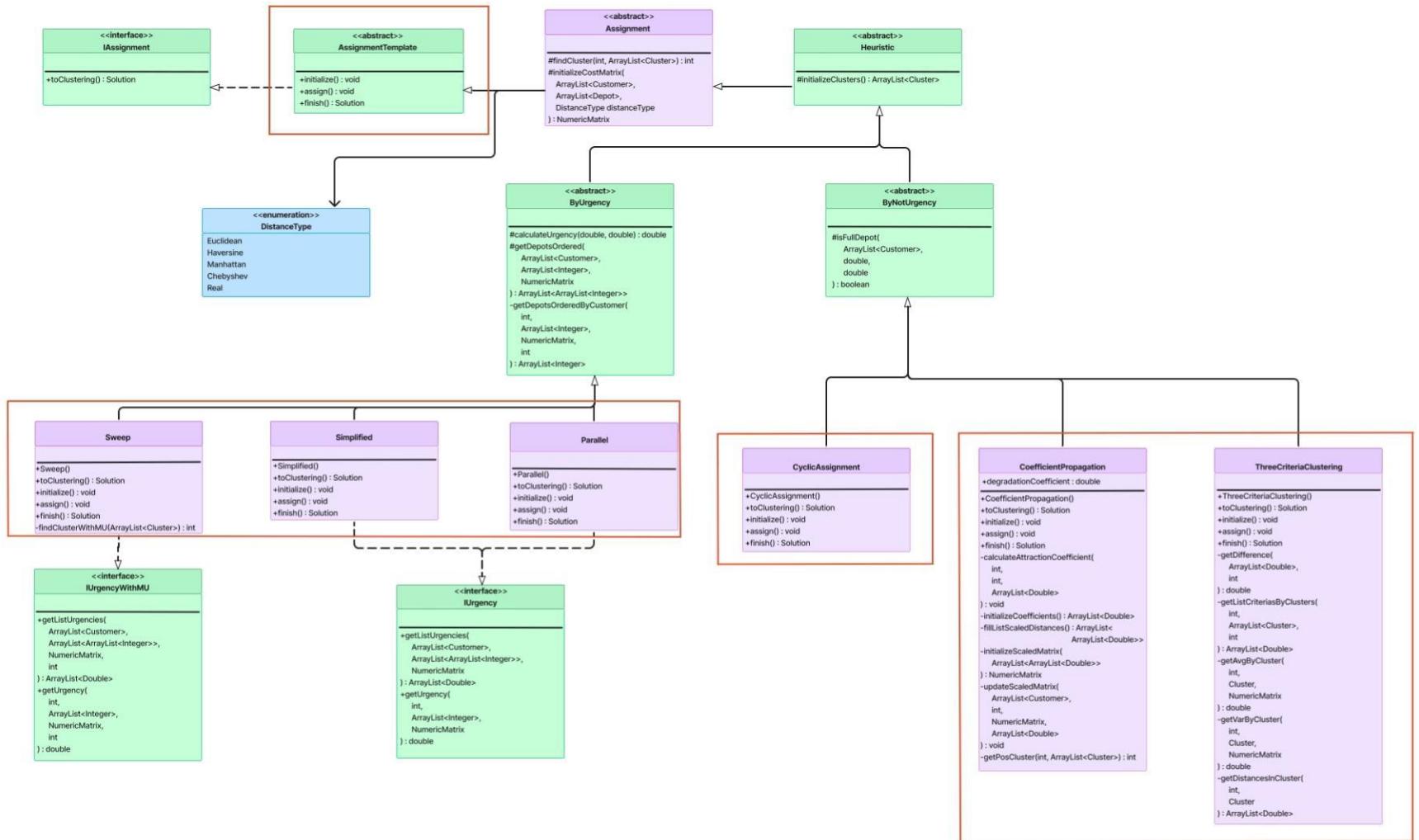


Figura 18: Diagrama de clases del paquete *classical* en la nueva versión.

En la Tabla 8 y en la Tabla 9 se describen las clases que contiene el paquete *classical*.

Tabla 8: Descripción de las clases que integran el paquete classical (parte I).

Clase	Descripción
<i>IAssignment</i>	Interfaz que define el contrato único <i>toClustering</i> para todas las implementaciones del paquete, asegurando un diseño orientado a interfaces y mayor extensibilidad.
<i>AssignmentTemplate</i>	Clase abstracta que proporciona métodos plantilla esenciales para la implementación del patrón de diseño <i>Template Method</i> . Esta clase sirve como base para definir la estructura general de los algoritmos de asignación, delegando los detalles específicos a las subclases que la extienden.
<i>Assignment</i>	Clase abstracta que define la estructura base para los algoritmos de asignación y proporciona métodos generales comunes.
<i>Heuristic</i>	Clase abstracta que extiende <i>Assignment</i> y actúa como base para las heurísticas clásicas.

Tabla 9: Descripción de las clases que integran el paquete classical (parte II).

Clase	Descripción
<i>ByUrgency</i>	Clase abstracta que hereda de <i>Heuristic</i> . Esta clase agrupa los métodos necesarios para las heurísticas basadas en urgencia.
<i>IUrgency</i>	Interfaz que define los métodos comunes para las heurísticas basadas en urgencia <i>Simplified</i> y <i>Parallel</i> .
<i>IUrgencyWithMU</i>	Interfaz que define los métodos comunes para la heurística basada en urgencia <i>Sweep</i> .
<i>Simplified</i>	Esta clase implementa la heurística Asignación Simplificada. Esta clase hereda de <i>ByUrgency</i> e implementa la interfaz <i>IUrgency</i> .
<i>Parallel</i>	Esta clase implementa la heurística Paralela. Esta clase hereda de <i>ByUrgency</i> e implementa la interfaz <i>IUrgency</i> .
<i>Sweep</i>	Esta clase implementa el algoritmo de Barrido. Esta clase hereda de <i>ByUrgency</i> e implementa la interfaz <i>IUrgencyWithMU</i> .
<i>ByNotCluster</i>	Clase abstracta que hereda de <i>Heuristic</i> y agrupa los métodos para las heurísticas clásicas no basadas en urgencia.
<i>ThreeCriteriaClustering</i>	Esta clase implementa la heurística Agrupamiento de Tres Criterios. Esta clase hereda de la clase <i>ByNotCluster</i> .
<i>CoefficientPropagation</i>	Esta clase implementa la heurística Coeficiente de Propagación. Esta clase hereda de la clase <i>ByNotCluster</i> .
<i>CyclicAssignment</i>	Esta clase implementa la heurística Asignación Cíclica. Esta clase hereda de la clase <i>ByNotCluster</i> .

El diagrama de clases que se muestra en la Figura 19 refleja las clases del paquete *clustering*.

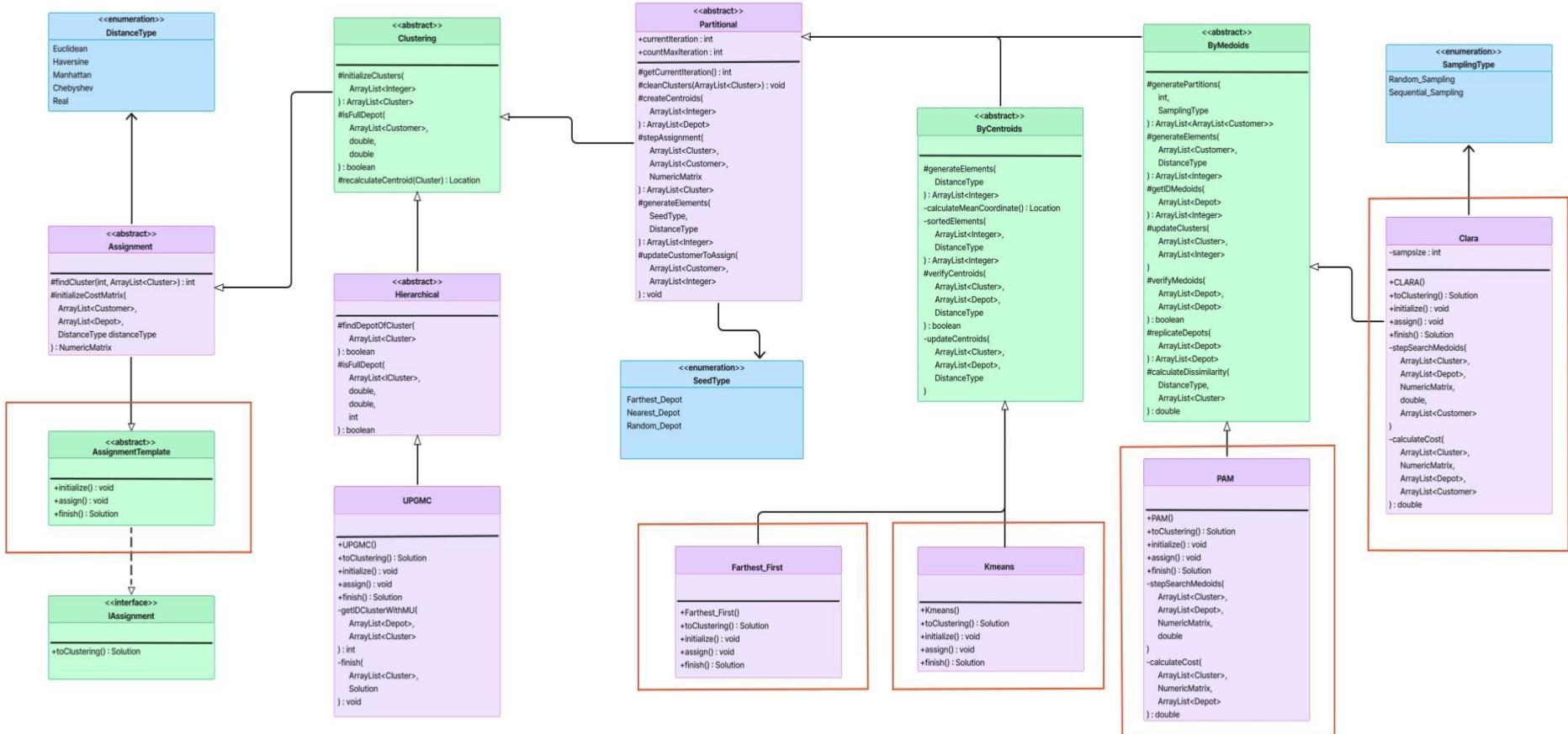


Figura 19: Diagrama de clases del paquete *clustering* en la nueva versión.

En la Tabla 10 y en la Tabla 11 se describen las clases que contiene el paquete *clustering*. La interfaz *IAssignment* y las clases abstractas *AssignmentTemplate* y *Assignment* con sus respectivas descripciones ya se han detallado en la Tabla 8, por lo que esta tabla se centra en las clases específicas relacionadas con los algoritmos de agrupamiento.

Tabla 10: Descripción de las clases que integra el paquete *clustering* (parte I).

Clase	Descripción
<i>Clustering</i>	Clase abstracta que extiende <i>Assignment</i> y actúa como base para las adaptaciones de los algoritmos de agrupamiento.
<i>SeedType</i>	Enumerado que gestiona las configuraciones y estrategias para la selección de semillas iniciales en la mayoría de las adaptaciones de los algoritmos de agrupamiento.
<i>Hierarchical</i>	Clase abstracta que hereda de <i>Clustering</i> y define los métodos específicos para los algoritmos jerárquicos de agrupamiento. Esta estructura facilita añadir nuevos algoritmos jerárquicos sin modificar la base, alineándose con el Principio Abierto/Cerrado.
<i>UPGMC</i>	Esta clase implementa el algoritmo de agrupamiento jerárquico <i>Unweighted Pair Group Method using Centroids</i> , hereda de la clase <i>Hierarchical</i> .
<i>Partitional</i>	Clase abstracta que hereda de <i>Clustering</i> y establece métodos particulares para los algoritmos de agrupamiento particional.

Tabla 11: Descripción de las clases que integra el paquete clustering (parte II).

Clase	Descripción
<i>ByCentroids</i>	Clase abstracta que hereda de <i>Partitional</i> y establece métodos particulares para los algoritmos de agrupamiento particional basados en centroides.
<i>Kmeans</i>	Esta clase implementa el algoritmo de agrupamiento <i>K-Means</i> , optimizando la distancia intra-clúster, heredando de la clase <i>ByCentroids</i> .
<i>Farthest-First</i>	Esta clase implementa el algoritmo particional <i>Farthest First</i> , que selecciona iterativamente el punto más alejado, heredando de la clase <i>ByCentroids</i> .
<i>ByMedoids</i>	Clase abstracta que hereda de <i>Partitional</i> y establece métodos particulares para los algoritmos de agrupamiento particional basados en medoides.
<i>PAM</i>	Esta clase implementa el algoritmo <i>Partitioning Around Medoids</i> para el agrupamiento basado en medianas, heredando de la clase <i>ByMedoids</i> .
<i>SamplingType</i>	Enumerado que define los métodos de muestreo aplicados en algoritmos como <i>Clara</i> para manejar grandes volúmenes de datos.
<i>Clara</i>	Esta clase implementa el algoritmo <i>Clustering Large Applications</i> , una variante de <i>PAM</i> para grandes conjuntos de datos, heredando de la clase <i>ByMedoids</i> .

Como parte de la reorganización del paquete *classical*, las heurísticas desarrolladas en el proyecto de Optimización y Metaheurísticas de la Facultad de Ingeniería Informática de la Universidad Tecnológica de La Habana “José Antonio Echeverría” (CUJAE) [23] [24] fueron trasladadas al paquete *others*. Ese grupo de heurísticas incluye a las basadas en distancias, la de asignación aleatoria y las inspiradas en el algoritmo de asignación cíclica. Esta reorganización se detalla en el diagrama de paquetes que se muestra en la Figura 20, y responde al objetivo de agrupar heurísticas con características similares en módulos más específicos.

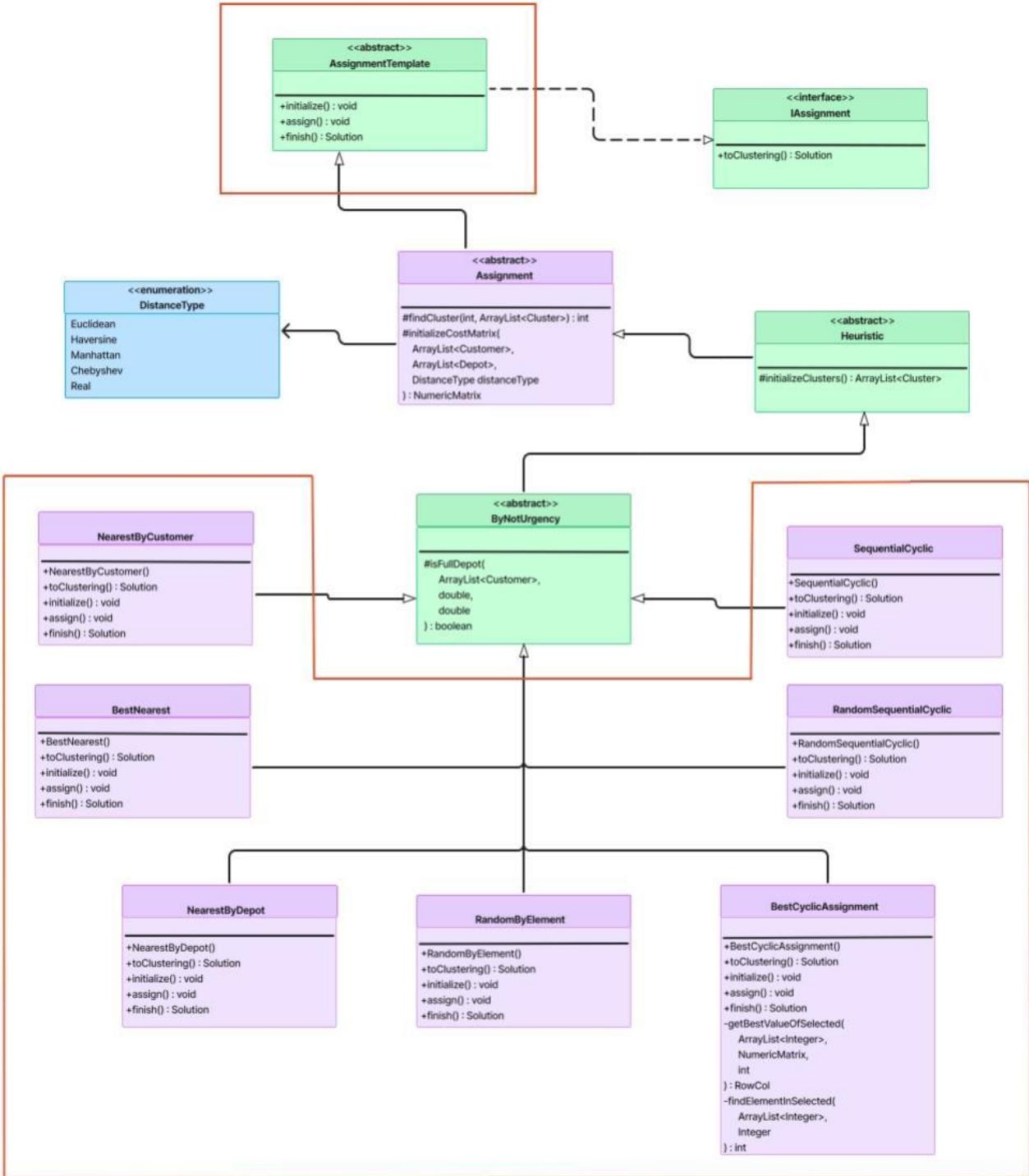


Figura 20: Diagrama de clases del paquete *others* en la nueva versión.

En la Tabla 12 se describen las clases que contiene el paquete *others*. Cabe destacar que la interfaz *IAssignment* y las clases abstractas *Assignment*, *Heuristic* y *ByNotUrgency*, representadas en la Figura 20, ya han sido explicadas previamente en la Tabla 8. Por lo tanto, en esta tabla solo se detallan las clases específicas de este paquete.

Tabla 12: Descripción de las clases que integran el paquete *others*.

Clase	Descripción
<i>NearestByDepot</i>	Esta clase implementa un algoritmo que asigna los clientes a los depósitos en el orden que se presentan los depósitos partiendo del criterio de cercanía a los clientes. Esta clase hereda de la clase <i>ByNotUrgency</i> .
<i>NearestByCustomer</i>	Esta clase implementa un algoritmo que asigna cada cliente a los depósitos considerando el orden de los clientes y se parte del criterio de cercanía a los depósitos. Esta clase hereda de la clase <i>ByNotUrgency</i> .
<i>BestNearest</i>	Esta clase implementa un algoritmo que asigna los clientes a los depósitos partiendo del criterio de ser los mejores candidatos. Esta clase hereda de la clase <i>ByNotUrgency</i> .
<i>RandomByElement</i>	Esta clase implementa un algoritmo que asigna clientes a depósitos de manera aleatoria. Esta clase hereda de la clase <i>ByNotUrgency</i> .
<i>BestCyclicAssignment</i>	Esta clase implementa un algoritmo que asigna el mejor cliente al último cliente o depósito asignando en forma paralela. Esta clase hereda de la clase <i>ByNotUrgency</i> .
<i>Sequential/Cyclic</i>	Esta clase implementa un algoritmo que asigna clientes en forma secuencial por depósitos. Esta clase hereda de la clase <i>ByNotUrgency</i> .
<i>RandomSequential/Cyclic</i>	Esta clase implementa un algoritmo que asigna los clientes a los depósitos en forma secuencial por depósitos escogiendo el depósito al azar. Esta clase hereda de la clase <i>ByNotUrgency</i> .

Por otra parte, el diagrama de clases que se muestra en la Figura 21 refleja las clases del paquete *factory*. Este paquete es responsable de proporcionar los mecanismos para la creación de objetos relacionados con los algoritmos de asignación, ofreciendo una estructura flexible para su configuración. En esta versión, se ha retirado la lógica encargada de la creación del tipo de distancia, debido a que el paquete *distance* se ha separado y encapsulado en una biblioteca independiente.

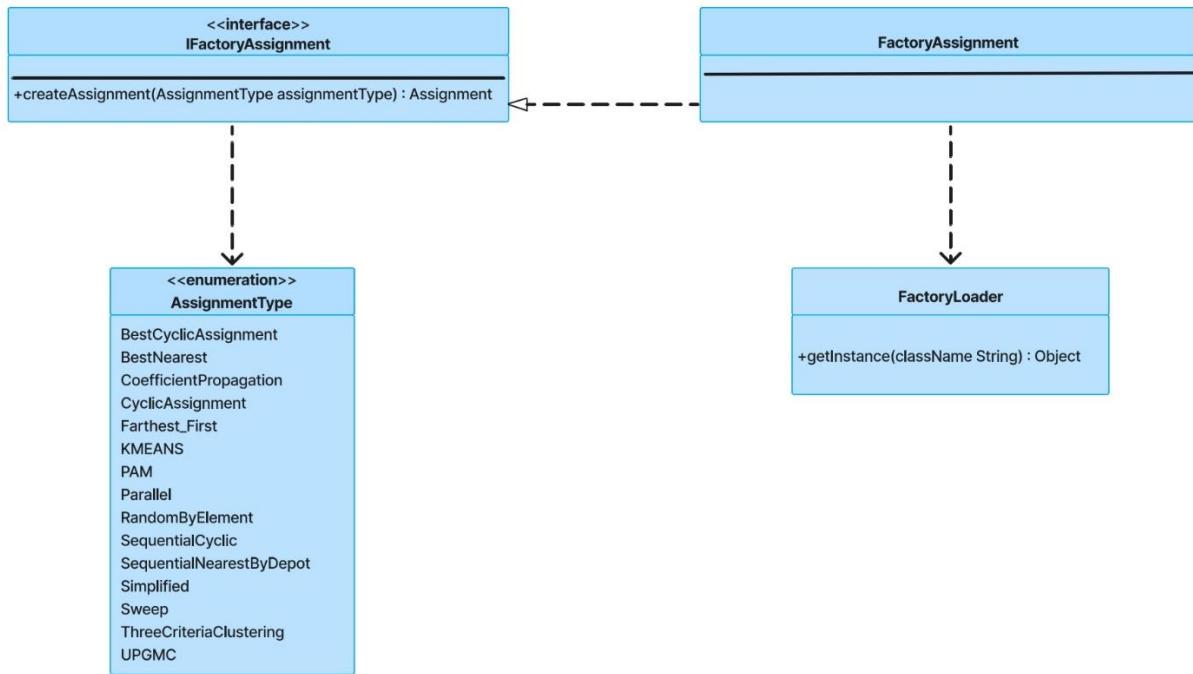


Figura 21: Diagrama de clases del paquete *factory* en la nueva versión.

En la Tabla 13 se describen las clases que contiene el paquete *factory*.

Tabla 13: Descripción de las clases que integran el paquete *factory*.

Clase	Descripción
<i>AssignmentType</i>	Enumerado que contiene los diferentes tipos de algoritmos de asignación que se pueden seleccionar para resolver la fase de asignación en un <b>MDVRP</b> .
<i>IFactoryAssignment</i>	Interfaz que define el método para la creación de algoritmos de asignación, el cual es implementado por la clase <i>FactoryAssignment</i> para crear instancias de los algoritmos de asignación según el tipo especificado.
<i>FactoryAssignment</i>	Clase que implementa el método para la creación de algoritmos de asignación de la interfaz <i>IFactoryAssignment</i> , encargada de crear instancias de los distintos algoritmos de asignación según el tipo proporcionado.
<i>FactoryLoader</i>	Clase que implementa el patrón <i>Factory Method</i> , encargada de gestionar la creación de los objetos necesarios para los algoritmos de asignación en el sistema.

Finalmente, en la capa general, se incorpora un paquete llamado *service*. En este paquete se encuentra únicamente la clase *OSRMService*, que abstrae los detalles técnicos del cálculo de distancias reales mediante **OSRM**. Para ello, ofrece dos opciones para obtener las distancias: a través de un servicio local **OSRM** o mediante el consumo de una API en línea.

- La API en línea de **OSRM** proporciona acceso a servicios **OSRM** basados en la web para los entornos donde no es posible instalar el servicio local. Sin embargo, se identifican problemas asociados al abuso del consumo del servidor en línea, como límites en la cantidad de solicitudes permitidas y bloqueos temporales. Esto

lleva a priorizar la configuración de un servicio local para garantizar disponibilidad constante.

- Para el servicio local de **OSRM** se utiliza una instancia local de *Open Source Routing Machine* instalada y configurada en un servidor local. Este enfoque reduce la dependencia de servicios externos y elimina problemas relacionados con la latencia y los límites de uso impuestos por servidores remotos. Sin embargo, una desventaja es que se requiere la instalación y el inicio previo del servidor local antes de su uso, lo que puede aumentar la complejidad en la configuración y el mantenimiento del entorno.

La clase `OSRMService` permite alternar entre el uso del servicio local o la API en línea mediante configuraciones. Adicionalmente, la clase implementa el uso de la caché interna para almacenar las distancias calculadas entre pares de puntos, evitando cálculos redundantes en iteraciones posteriores. Con esta implementación se ahorran peticiones al servidor, reduciendo significativamente el tiempo total de ejecución del algoritmo. Al final de la construcción de la matriz de costos, la caché es limpiada mediante el método `clearDistanceCache`, lo que garantiza un uso eficiente de la memoria.

Por último, el paquete `osrm_service`, ubicado en la capa de software del sistema, es común para ambas implementaciones, cumpliendo la misma función de gestionar el servicio **OSRM** para los cálculos de distancias reales, ya sea a través de una instancia local o una API remota.

### 2.2.3. Particularidades de BHAVRP en Java y Python

En esta sección se detallan los paquetes de la capa intermedia de **BHAVRP** en sus versiones Java y Python, explicando su función y la manera en que contribuyen al funcionamiento general del sistema. A continuación, en la Tabla 14 se comparan los paquetes de la capa intermedia y la capa software del sistema.

Tabla 14: Particularidades de BHAVRP en Java y Python.

Paquete	Java	Python
<b>Capa Intermedia</b>		
<i>DISTANCES</i>	<i>DISTANCE_MEASURE</i> : proporciona el cálculo de distancias aproximadas.	<b>Scipy</b> : presenta herramientas matemáticas optimizadas para distancias aproximadas.
<i>MATRIX</i>	<i>MATRIX_CLASS</i> : proporciona cálculos matriciales para la confección de la matriz de costos.	<b>NumPy</b> : realiza cálculos matriciales y operaciones con arreglos.
<i>OSRM</i>	No aplica.	<b>osrm-py</b> : gestiona las solicitudes de distancias reales enviadas a <b>OSRM</b> mediante su API.
<i>utils</i>	Bibliotecas Java ( <i>java.util</i> , <i>java.lang</i> ): presenta las clases, implementaciones y excepciones propias del lenguaje.	Bibliotecas Python ( <i>time</i> , <i>typing</i> , <i>abc</i> ): manejo de tiempos, tipos de listas y clases base abstractas.
<b>Capa Software del Sistema</b>		
<i>communication</i>	Facilita la interacción y el flujo de datos entre las bibliotecas Java.	<b>Buffer</b> : protocolo de comunicación en Python para manejar grandes volúmenes de datos.

### 2.3. Principios de diseño adoptados en la solución

En la nueva versión de **BHAVRP**, se mantienen los principios de diseño mencionados en la sección 1.2.3, con el objetivo de garantizar un código más claro, mantenible y escalable. Sin embargo, algunos de estos principios han sido modificados para adaptarse mejor a los nuevos requerimientos y optimizar su implementación. A continuación, se detallan los principios aplicados, junto con ejemplos específicos que evidencian su adopción en las distintas partes de la biblioteca:

- Principio de Alta Cohesión (*High Cohesion*): establece que una clase, módulo o componente debe tener responsabilidades estrechamente relacionadas y centradas en un único propósito. Una alta cohesión asegura que cada elemento del sistema sea claro, eficiente y fácil de mantener, ya que todas sus funcionalidades están enfocadas en resolver un aspecto específico del problema [85]. Este principio se refleja en **BHAVRP** mediante la creación de clases abstractas que agrupan el comportamiento común de los algoritmos, permitiendo una estructura más clara y modular que facilita la implementación y extensión de funcionalidades específicas. Los diagramas que se presentan en la Figura 18, Figura 19 y Figura 20 evidencian este principio.
- Principio de Bajo Acoplamiento (*Low Coupling*): establece que los componentes de un sistema deben tener la menor dependencia posible entre sí. Esto permite que las partes del sistema sean más fáciles de modificar, probar y reutilizar, ya que los cambios en un componente tienen un impacto mínimo en otros [69]. En **BHAVRP**, este principio se refleja en la estructura modular del sistema, donde se utilizan interfaces y clases abstractas para separar las responsabilidades. Un ejemplo de esto es la clase *ByUrgency*, que define una estructura base común para las heurísticas de asignación por urgencia, mientras que las implementaciones específicas como *Simplified*, *Parallel*, o *Sweep* no dependen directamente entre sí, sino que se comunican a través de interfaces y herencia. Esto permite modificar o extender los algoritmos de asignación sin afectar otras partes del sistema, favoreciendo la escalabilidad y la mantenibilidad. En la Figura 18 se presenta un diagrama de clases que ilustra este principio.

Además, se han adoptado algunos nuevos principios para mejorar la organización de la biblioteca. Estos principios permiten una estructuración más apropiada del código, promoviendo la cohesión, la separación de responsabilidades y la reutilización, lo que facilita tanto el mantenimiento como la expansión futura del sistema.

- Principio de Separación de Responsabilidades (*Separation of Concerns*): establece que un sistema debe dividirse en secciones, donde cada una aborde una responsabilidad específica y separada, facilitando la organización,

mantenimiento y escalabilidad del código [86]. En **BHAVRP**, la separación de preocupaciones se refleja claramente en la estructura modular del sistema. El paquete *heuristic* agrupa las clases encargadas de implementar los algoritmos de asignación, así como la separación de la lógica de los distintos tipos de algoritmos. Dentro de este paquete, se distinguen las heurísticas de asignación, las cuales se subdividen en grupos como las de urgencia, cíclicas y basadas en clúster. Además, se establece una separación adicional para las adaptaciones de los algoritmos basados en partición y jerárquicos, permitiendo una mayor claridad y organización en la implementación de cada tipo de algoritmo. Esta organización permite una delimitación clara de responsabilidades, reduciendo la complejidad y facilitando futuras extensiones. Los diagramas que se presentan en la Figura 14, Figura 18 y Figura 19 evidencian este principio.

- Principio de No Repetir Código (*Don't Repeat Yourself, DRY*): enfatiza que cada fragmento de conocimiento o lógica del sistema debe tener una única representación en el código, evitando duplicaciones que dificulten el mantenimiento y aumenten la posibilidad de errores [87]. En **BHAVRP**, este principio se implementa de diversas maneras. Por ejemplo, la clase *Problem* centraliza toda la información relacionada con el problema **MDVRP**, evitando la necesidad de replicar datos o lógica en otras partes del sistema. Además, clases abstractas como *Heuristic*, *ByUrgency*, *ByNotUrgency*, *Clustering* y *Partitional* encapsulan el comportamiento común para cada tipo de algoritmo, asegurando que estos no necesiten redefinir funcionalidades compartidas, como la lógica para evaluar soluciones o manejar estructuras comunes. Este diseño se puede observar en los diagramas presentados en la Figura 16 y Figura 18.
- Principio de Modularidad (*Modularity*): asegura que el sistema esté organizado en módulos independientes, cada uno responsable de una funcionalidad o conjunto de funcionalidades específicas. La modularidad no solo facilita el mantenimiento, sino que también permite reemplazar o extender módulos sin afectar el resto del sistema [88]. En **BHAVRP**, los módulos están definidos como paquetes bien estructurados, como *solution*, que maneja exclusivamente las soluciones generadas en los procesos de asignación y planificación. Este diseño modular

permite extender funcionalidades o ajustar algoritmos sin afectar el resto del sistema, como se puede observar en el diagrama de clases de la Figura 14.

## 2.4. Patrones de diseño adoptados en la solución

En este epígrafe se detalla el patrón de diseño implementado en la nueva versión de **BHAVRP**, el cual aborda necesidades específicas de esta iteración de la biblioteca. Este patrón complementa los patrones de diseño abordados en la subsección 1.2.4, como el patrón *Factory Method* para la creación de objetos relacionados con los algoritmos de asignación y el Patrón Controlador para gestionar la interacción entre las distintas partes del sistema. Ambos patrones de diseño continúan siendo fundamentales en la estructuración y en el funcionamiento de la biblioteca.

### 2.4.1. Patrón *Template Method*

En **BHAVRP**, los algoritmos para la asignación de clientes a depósitos siguen un conjunto de pasos estructurados. El proceso comienza con la preparación de los elementos que se deben asignar y la inicialización de las estructuras necesarias para cada algoritmo. Luego, se procede a distribuir los clientes entre los depósitos, evaluando continuamente la asignación de acuerdo a criterios previamente establecidos, asegurándose de respetar las restricciones definidas en el problema, como las capacidades máximas de los depósitos y las demandas de los clientes. Finalmente, en los métodos basados en agrupamiento, el proceso incluye una fase de ajuste iterativo, donde se evalúan y mejoran las asignaciones hasta alcanzar un criterio de convergencia o llegar al límite de iteraciones permitido.

A continuación, en la **!Error! No se encuentra el origen de la referencia.** se presenta el diagrama de actividades del algoritmo *Farthest First*, donde se pueden observar los tres pasos principales identificados en el proceso de asignación de clientes a depósitos: inicialización, asignación y finalización. Este diagrama sirve como base para establecer la estructura general del proceso en **BHAVRP**.

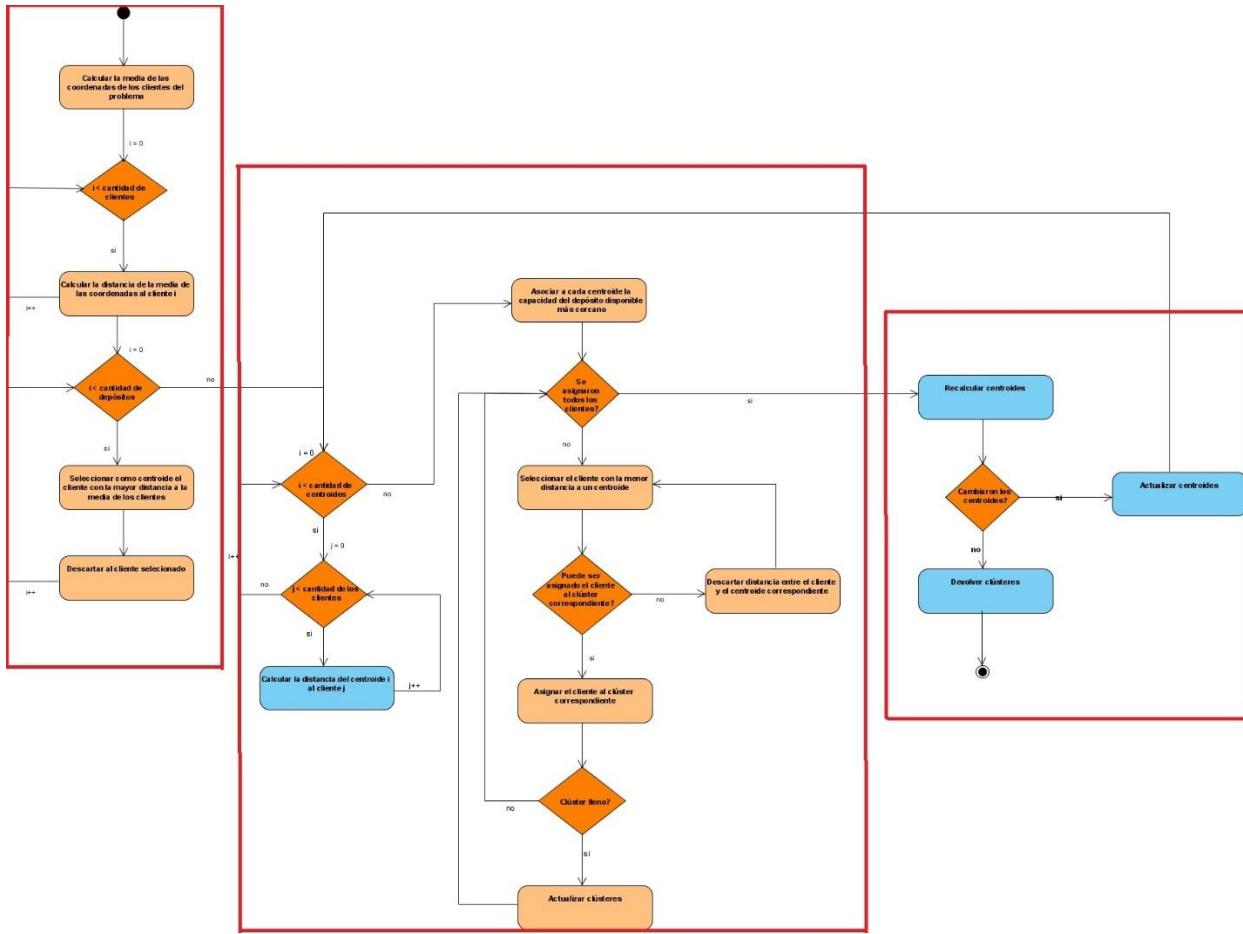


Figura 22: Diagrama de actividades del algoritmo *Farthest First* [89].

Para generalizar el comportamiento común a todos los algoritmos de asignación, se propone en la Figura 23 un diagrama genérico que encapsula los tres pasos previamente establecidos. Este diagrama define una estructura estándar que facilita la comprensión y organización del flujo de trabajo en **BHAVRP**.



Figura 23: Diagrama genérico común para todos los algoritmos.

La necesidad de implementar el patrón *Template Method* surge a partir de la observación del comportamiento de todos los algoritmos de asignación. Estos presentan una estructura común, pero requieren pasos específicos en diferentes puntos del proceso. Este patrón permite encapsular el comportamiento común en una clase base, mientras que las subclases pueden personalizar los pasos específicos [90]. Con esta implementación, se asegura una mayor reutilización del código, mejora la cohesión al mantener una lógica común centralizada y facilita la extensión futura del sistema.

El patrón cuenta con ciertos elementos distintivos. El método plantilla, es el algoritmo común que define el flujo general de los pasos y es invocado en la clase base. Dentro de este patrón, existen métodos abstractos, que son aquellos cuyo comportamiento específico debe ser implementado por las subclases; estos pasos pueden variar dependiendo de la heurística utilizada. Por otro lado, los métodos concretos son aquellos con una implementación predeterminada en la clase base, que las subclases pueden utilizar tal cual sin necesidad de reescribir su comportamiento. En la Figura 24 se muestra un diagrama de clases con un ejemplo de cómo implementar este patrón.

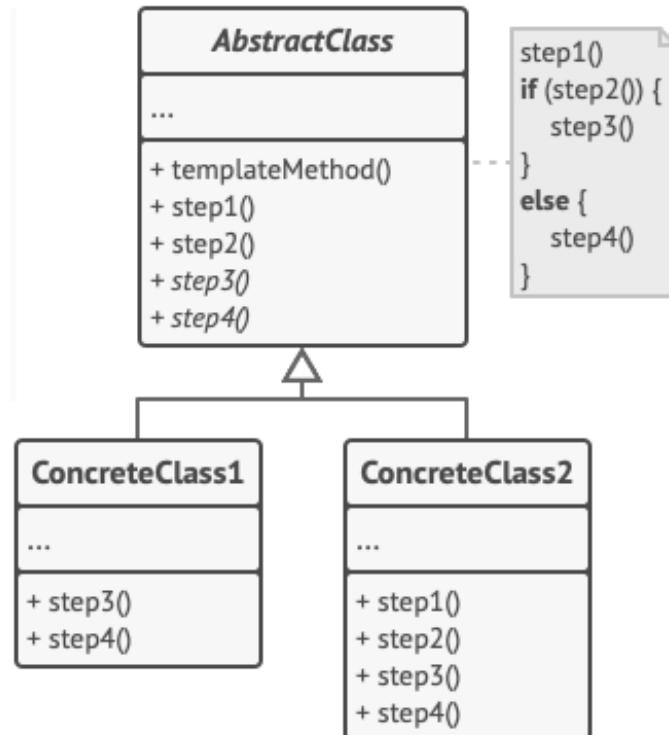


Figura 24: Ejemplo de la implementación del patrón *Template Method* [90].

En el contexto de **BHAVRP**, el proceso de asignación de clientes a depósitos sigue un conjunto de pasos comunes que varían ligeramente según el algoritmo utilizado. Para estructurar esta lógica y facilitar la reutilización del código, se aplicó el patrón *Template Method*. Este patrón define el flujo general del algoritmo en un método base, delegando la implementación de pasos específicos a las subclases.

Para una mejor comprensión, este patrón se puede observar en el contexto general de los algoritmos en los diagramas de clases presentados en las Figura 18, Figura 19 y Figura 20. En dichas figuras, se enmarca cómo el *Template Method* organiza la lógica común, mientras que las subclases especializadas implementan los pasos específicos necesarios para cada algoritmo.

En el caso del algoritmo *Farthest First*, el método *toClustering* definido en la interfaz *IAssignment* es el encargado de encapsular el flujo completo del algoritmo de agrupamiento y los métodos plantilla quedan definidos por la clase abstracta *AssignmentTemplate*:

- Inicialización (*initialize*): genera los elementos iniciales y prepara las estructuras necesarias, como el conjunto de clústeres, los clientes a asignar y los centroides.
- Asignación (*assign*): define el ciclo iterativo en el que los clientes se asignan a los clústeres utilizando una matriz de costos, se actualizan los centroides y se verifica si se requiere una nueva iteración.
- Finalización (*finish*): ensambla la solución final, clasificando los clientes no asignados y los clústeres resultantes.

En la implementación específica del algoritmo *Farthest First*, estos métodos son sobrescritos para proporcionar un comportamiento personalizado mientras siguen el marco general definido por el patrón *Template Method*.

Esta estructura demuestra cómo el patrón *Template Method* permite definir un flujo de trabajo general, dejando a las subclases la responsabilidad de implementar las partes específicas, respetando así principios de diseño como el Principio Abierto/Cerrado [67] y fomentando la reutilización y la extensibilidad.

## 2.5. Conclusiones parciales

A partir del análisis realizado en este capítulo sobre el diseño de la nueva versión de **BHAVRP**, se concluyen los siguientes aspectos clave:

- La actualización de la versión en Java de BHAVRP y su implementación paralela en Python representa una expansión estratégica de la herramienta. Esta doble implementación permite aprovechar las fortalezas de ambos lenguajes, mejorando la accesibilidad de la herramienta, su interoperabilidad en ecosistemas de software más amplios y facilitando la incorporación de nuevas funcionalidades.
- La comparación entre las versiones Java y Python muestra como las deficiencias de la versión original en Java se llegan a resolver en la implementación en Python.
- La reestructuración de la arquitectura de **BHAVRP**, especialmente en el paquete *heuristic*, mejora la flexibilidad mediante la introducción de interfaces y clases abstractas, basándose en el paradigma orientado a objetos.
- La adopción de principios de diseño como alta cohesión, bajo acoplamiento y **DRY**, junto con patrones como *Template Method*, ha sido crucial para garantizar la mantenibilidad y claridad del código.
- El rediseño de **BHAVRP** sienta las bases para una versión eficiente y escalable, preparada para afrontar los desafíos de planificación logística en escenarios más complejos.

## Capítulo 3: Análisis y validación de BHAVRP

### 3.1. Introducción

En este capítulo se presentan los experimentos realizados para analizar y validar el funcionamiento de las distintas versiones de **BHAVRP**. Estos experimentos se enfocan en las principales actividades que realiza **BHAVRP**: carga de datos, generación de la matriz de costo y ejecución de los algoritmos. En la sección 3.2 se describe el primer experimento para validar el proceso de carga de datos de las versiones desarrolladas de la biblioteca. En la sección 3.3 se explica un experimento para mostrar cómo se construye la matriz de costos con los distintos tipos de distancias (incluye el uso de distancias reales) que manejan las versiones de **BHAVRP**. En la sección 3.4 se demuestra el correcto funcionamiento de los algoritmos presentes en las bibliotecas.

Los experimentos que se realizan en este capítulo utilizan las funcionalidades y algoritmos implementados en diferentes versiones de **BHAVRP**, con los lenguajes de programación Java y Python. Las versiones en Java se ejecutan utilizando el IDE Eclipse en su versión 4.2 Juno, mientras que las versiones en Python se ejecutan desde el IDE Visual Studio Code versión 1.95 [91], utilizando Python versión 3.12 [92] al momento de la implementación.

Todos los experimentos se llevan a cabo en un entorno controlado, utilizando un ordenador con las siguientes especificaciones: procesador Intel CORE i5-6400 de 2.7 GHz, 8 GB de memoria RAM y sistema operativo Windows 11 Pro de 64 bits, actualizado a la última versión disponible. Estas características aseguran condiciones homogéneas para comparar el desempeño de las distintas versiones y lenguajes de implementación.

Todos los experimentos diseñados comparan los resultados obtenidos por los algoritmos implementados en las tres versiones de **BHAVRP**:

- Versión original: representa la implementación original de la biblioteca en Java, que sirvió como punto de partida para identificar las oportunidades de mejora abordadas en la sección 1.5.

- Versión actualizada en Java: representa una versión mejorada de la biblioteca original, que incorpora los ajustes presentados en el Capítulo 2: Diseño de la nueva versión de BHAVRP.
- Nueva versión en Python: consiste en una reimplementación paralela de la biblioteca actualizada, en Python.

### 3.2. Experimento para validar del proceso de carga de datos

En esta sección, se compara el proceso de carga de datos entre las tres versiones de la biblioteca: la versión base en Java, la versión actualizada también desarrollada en Java, y la versión migrada a Python. Este análisis tiene como objetivo demostrar que todas las versiones logran procesar y cargar los datos de manera efectiva, cumpliendo con las expectativas del problema **MDVRP**.

#### 3.2.1. Descripción de la instancia

Para la ejecución del experimento se selecciona una instancia de la variante **MDVRP** del autor Cordeau disponibles en [93]. En la Tabla 15 se presenta para la instancia del problema el total de clientes, el total de depósitos, las cantidades de vehículos por depósito y las capacidades de los vehículos.

Tabla 15: Descripción de la instancia de la literatura.

ID	Total de clientes	Total de depósitos	Total de vehículos por depósito	Capacidad de los vehículos
1	50	4	4	80

Esta instancia fue seleccionada por su tamaño moderado, lo que facilita la comparación detallada de los resultados obtenidos en las diferentes versiones de **BHAVRP**. Además, su uso previo en [23] [24] [89] garantiza la validez de sus datos, permitiendo evaluar el desempeño de los algoritmos en el último de los experimentos diseñados.

#### 3.2.2. Análisis de los resultados del proceso de carga de datos

En esta subsección, se analizan los resultados obtenidos al cargar una instancia específica en las tres versiones de **BHAVRP**: la versión original, la versión actualizada y

la versión migrada a Python. El objetivo principal de este análisis es demostrar que el proceso de carga de datos es eficiente y consistente entre las versiones, garantizando que los datos de entrada se interpretan y almacenan correctamente en las estructuras internas de cada implementación.

A continuación, se presenta la Tabla 16 con los resultados obtenidos, donde se detallan la cantidad de clientes y su demanda total, así como la cantidad de depósitos junto con sus capacidades y flotas de vehículos asignadas.

Tabla 16: Resultados de la carga de datos en las tres versiones de BHAVRP.

Proceso de carga de datos		
Versiones		
Original	Actualizada	Python
<terminated> Main [Java Application] C:\Apps\java\jdk-8u281\bin\java	<terminated> MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\java	RESUMEN DE LA CARGA DE DATOS:
RESUMEN DE LA CARGA DE DATOS:	RESUMEN DE LA CARGA DE DATOS:	CANTIDAD DE CLIENTES: 50
CANTIDAD DE CLIENTES: 50	CANTIDAD DE CLIENTES: 50	DEMANDA TOTAL DE LOS CLIENTES: 777.0
DEMANDA TOTAL DE LOS CLIENTES: 777.0	DEMANDA TOTAL DE LOS CLIENTES: 777.0	CANTIDAD DE DEPÓSITOS: 4
CANTIDAD DE DEPÓSITOS: 4	CANTIDAD DE DEPÓSITOS: 4	CAPACIDAD TOTAL DE LOS DEPÓSITOS: 1280.0
CAPACIDAD TOTAL DE LOS DEPÓSITOS: 1280.0	CAPACIDAD TOTAL DE LOS DEPÓSITOS: 1280.0	ID DEPÓSITO: 51
ID DEPÓSITO: 51	ID DEPÓSITO: 51	CANTIDAD DE FLOTAS DEL DEPÓSITO: 1
CANTIDAD DE FLOTAS DEL DEPÓSITO: 1	CANTIDAD DE FLOTAS DEL DEPÓSITO: 1	CANTIDAD DE VEHÍCULOS: 4
CANTIDAD DE VEHÍCULOS: 4	CANTIDAD DE VEHÍCULOS: 4	CAPACIDAD DE LOS VEHÍCULOS: 80.0
CAPACIDAD DE LOS VEHÍCULOS: 80.0	CAPACIDAD DE LOS VEHÍCULOS: 80.0	CAPACIDAD TOTAL DEL DEPÓSITO: 320.0
CAPACIDAD TOTAL DEL DEPÓSITO: 320.0	CAPACIDAD TOTAL DEL DEPÓSITO: 320.0	ID DEPÓSITO: 52
ID DEPÓSITO: 52	ID DEPÓSITO: 52	CANTIDAD DE FLOTAS DEL DEPÓSITO: 1
CANTIDAD DE FLOTAS DEL DEPÓSITO: 1	CANTIDAD DE FLOTAS DEL DEPÓSITO: 1	CANTIDAD DE VEHÍCULOS: 4
CANTIDAD DE VEHÍCULOS: 4	CANTIDAD DE VEHÍCULOS: 4	CAPACIDAD DE LOS VEHÍCULOS: 80.0
CAPACIDAD DE LOS VEHÍCULOS: 80.0	CAPACIDAD DE LOS VEHÍCULOS: 80.0	CAPACIDAD TOTAL DEL DEPÓSITO: 320.0
CAPACIDAD TOTAL DEL DEPÓSITO: 320.0	CAPACIDAD TOTAL DEL DEPÓSITO: 320.0	ID DEPÓSITO: 53
ID DEPÓSITO: 53	ID DEPÓSITO: 53	CANTIDAD DE FLOTAS DEL DEPÓSITO: 1
CANTIDAD DE FLOTAS DEL DEPÓSITO: 1	CANTIDAD DE FLOTAS DEL DEPÓSITO: 1	CANTIDAD DE VEHÍCULOS: 4
CANTIDAD DE VEHÍCULOS: 4	CANTIDAD DE VEHÍCULOS: 4	CAPACIDAD DE LOS VEHÍCULOS: 80.0
CAPACIDAD DE LOS VEHÍCULOS: 80.0	CAPACIDAD DE LOS VEHÍCULOS: 80.0	CAPACIDAD TOTAL DEL DEPÓSITO: 320.0
CAPACIDAD TOTAL DEL DEPÓSITO: 320.0	CAPACIDAD TOTAL DEL DEPÓSITO: 320.0	ID DEPÓSITO: 54
ID DEPÓSITO: 54	ID DEPÓSITO: 54	CANTIDAD DE FLOTAS DEL DEPÓSITO: 1
CANTIDAD DE FLOTAS DEL DEPÓSITO: 1	CANTIDAD DE FLOTAS DEL DEPÓSITO: 1	CANTIDAD DE VEHÍCULOS: 4
CANTIDAD DE VEHÍCULOS: 4	CANTIDAD DE VEHÍCULOS: 4	CAPACIDAD DE LOS VEHÍCULOS: 80.0
CAPACIDAD DE LOS VEHÍCULOS: 80.0	CAPACIDAD DE LOS VEHÍCULOS: 80.0	CAPACIDAD TOTAL DEL DEPÓSITO: 320.0
CAPACIDAD TOTAL DEL DEPÓSITO: 320.0	CAPACIDAD TOTAL DEL DEPÓSITO: 320.0	CARGA EXITOSA: true
CARGA EXITOSA: true	CARGA EXITOSA: true	FIN DE LA CARGA DE DATOS
FIN DE LA CARGA DE DATOS	FIN DE LA CARGA DE DATOS	FIN DE LA CARGA DE DATOS

En la Tabla 15 se presenta la descripción detallada de la instancia utilizada para el problema, mostrando los datos principales que fueron cargados. En el proceso de la carga de datos se puede observar que se cargaron 50 clientes, con una demanda total de 777 unidades, a distribuir entre 4 depósitos, cada uno con 1 flota de 4 vehículos, con una capacidad de 80 unidades por vehículo, para una capacidad total de 320 unidades por depósito y una capacidad total entre todos los depósitos de 1280 unidades.

Este análisis demuestra que los datos han sido correctamente cargados, manteniéndose coherentes tanto en la versión original como en las versiones actualizadas. La información sobre los clientes, depósitos, vehículos y sus capacidades se ha preservado de manera íntegra a lo largo de las actualizaciones, lo que garantiza que el modelo sigue representando de forma precisa los elementos involucrados en el problema.

### 3.3. Experimento para validar el proceso de creación de la matriz de costos

En esta sección se compara el proceso de creación de la matriz de costos entre las tres versiones de **BHAVRP**. La matriz de costos es esencial para la solución del problema **MDVRP**, ya que representa los costos asociados a la distancia entre los clientes y los depósitos. La creación de esta matriz debe ser precisa y eficiente, y es importante que se mantenga consistente en todas las versiones del sistema.

En este caso, la matriz de costos es construida utilizando distancias aproximadas y distancias reales. En el caso de las distancias aproximadas se hace uso de las medidas de distancias soportadas por la biblioteca **BHAVRP**: *Euclidean*, *Manhattan*, *Chebyshev*, *Haversine*. Para mostrar la validez de la propuesta con distancias reales se hace uso del servicio de **OSRM**, que permite obtener las distancias entre puntos considerando datos geográficos reales, restricciones de tránsito y características específicas de las vías. Cada una de estas distancias ofrece una aproximación diferente para calcular el costo de los desplazamientos, lo que permite evaluar la flexibilidad de la biblioteca para adaptarse a distintos escenarios.

### 3.3.1. Descripción de la instancia

Para este experimento se utiliza una instancia basada en coordenadas reales de la ciudad de La Habana. Dicha instancia fue confeccionada a partir de un conjunto de instancias disponibles en [94] y tiene la particularidad de contar con datos geográficos reales, lo que permite validar el proceso de creación de la matriz de costos en un contexto práctico.

En la Tabla 17 se presenta para la instancia del problema el total de clientes, el total de depósitos, las cantidades de vehículos por depósito y las capacidades de los vehículos.

Tabla 17: Descripción de la instancia real.

ID	Total de clientes	Total de depósitos	Total de vehículos por deposito	Capacidad de los vehículos
21	10	2	2	20

### 3.3.2. Análisis de los resultados del proceso de creación de la matriz de costos

En esta subsección, se analizan los resultados obtenidos al cargar la instancia con coordenadas reales en las tres versiones de **BHAVRP**. El objetivo principal de este análisis es demostrar que el proceso de creación de la matriz de costos es eficiente y consistente entre las versiones.

A continuación, en las tablas siguientes se presentan los resultados de la creación de la matriz de costos utilizando los tipos de distancias mencionados en la subsección anterior. Es importante señalar que en la diagonal principal de la matriz se utiliza el valor infinito (*Infinity* o *inf*) para indicar la ausencia de distancia entre un punto y sí mismo, ya que no es posible utilizar el valor 0 debido a que este puede ser considerado como un valor válido dentro del proceso de asignación de los algoritmos.

Tabla 18: Resultados de la creación de la matriz de costos usando la distancia Euclidean.

Versiones	Tipo de distancia: <i>Euclidean</i>
Original	<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 22:58:56) Matriz de costos - Euclidean: C1 Infinity 0.05 0.04 0.05 0.02 0.03 0.06 0.03 0.10 0.04 0.05 0.02 C2 0.05 Infinity 0.02 0.08 0.05 0.08 0.11 0.07 0.12 0.04 0.06 0.05 C3 0.04 0.02 Infinity 0.08 0.03 0.07 0.10 0.06 0.13 0.03 0.06 0.05 C4 0.05 0.08 0.08 Infinity 0.07 0.03 0.06 0.03 0.06 0.09 0.04 0.03 C5 0.02 0.05 0.03 0.07 Infinity 0.05 0.07 0.04 0.12 0.02 0.07 0.04 C6 0.03 0.08 0.07 0.03 0.05 Infinity 0.04 0.00 0.09 0.07 0.05 0.03 C7 0.06 0.11 0.10 0.06 0.07 0.04 Infinity 0.04 0.11 0.09 0.09 0.07 C8 0.03 0.07 0.06 0.03 0.04 0.00 0.04 Infinity 0.09 0.07 0.05 0.03 C9 0.10 0.12 0.13 0.06 0.12 0.09 0.11 0.09 Infinity 0.14 0.06 0.08 C10 0.04 0.04 0.03 0.09 0.02 0.07 0.09 0.07 0.14 Infinity 0.08 0.06 D1 0.05 0.06 0.06 0.04 0.07 0.05 0.09 0.05 0.06 0.08 Infinity 0.03 D2 0.02 0.05 0.05 0.03 0.04 0.03 0.07 0.03 0.08 0.06 0.03 Infinity</pre>
Actualizada	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 22:58:56) Matriz de costos - Euclidean: C1 Infinity 0.05 0.04 0.05 0.02 0.03 0.06 0.03 0.10 0.04 0.05 0.02 C2 0.05 Infinity 0.02 0.08 0.05 0.08 0.11 0.07 0.12 0.04 0.06 0.05 C3 0.04 0.02 Infinity 0.08 0.03 0.07 0.10 0.06 0.13 0.03 0.06 0.05 C4 0.05 0.08 0.08 Infinity 0.07 0.03 0.06 0.03 0.06 0.09 0.04 0.03 C5 0.02 0.05 0.03 0.07 Infinity 0.05 0.07 0.04 0.12 0.02 0.07 0.04 C6 0.03 0.08 0.07 0.03 0.05 Infinity 0.04 0.00 0.09 0.07 0.05 0.03 C7 0.06 0.11 0.10 0.06 0.07 0.04 Infinity 0.04 0.11 0.09 0.09 0.07 C8 0.03 0.07 0.06 0.03 0.04 0.00 0.04 Infinity 0.09 0.07 0.05 0.03 C9 0.10 0.12 0.13 0.06 0.12 0.09 0.11 0.09 Infinity 0.14 0.06 0.08 C10 0.04 0.04 0.03 0.09 0.02 0.07 0.09 0.07 0.14 Infinity 0.08 0.06 D1 0.05 0.06 0.06 0.04 0.07 0.05 0.09 0.05 0.06 0.08 Infinity 0.03 D2 0.02 0.05 0.05 0.03 0.04 0.03 0.07 0.03 0.08 0.06 0.03 Infinity</pre>
Python	<pre>Matriz de costos - Euclidean: C1 inf 0.05 0.04 0.05 0.02 0.03 0.06 0.03 0.10 0.04 0.05 0.02 C2 0.05 inf 0.02 0.08 0.05 0.08 0.11 0.07 0.12 0.04 0.06 0.05 C3 0.04 0.02 inf 0.08 0.03 0.07 0.10 0.06 0.13 0.03 0.06 0.05 C4 0.05 0.08 0.08 inf 0.07 0.03 0.06 0.03 0.06 0.09 0.04 0.03 C5 0.02 0.05 0.03 0.07 inf 0.05 0.07 0.04 0.12 0.02 0.07 0.04 C6 0.03 0.08 0.07 0.03 0.05 inf 0.04 0.00 0.09 0.07 0.05 0.03 C7 0.06 0.11 0.10 0.06 0.07 0.04 inf 0.04 0.11 0.09 0.09 0.07 C8 0.03 0.07 0.06 0.03 0.04 0.00 0.04 inf 0.09 0.07 0.05 0.03 C9 0.10 0.12 0.13 0.06 0.12 0.09 0.11 0.09 inf 0.14 0.06 0.08 C10 0.04 0.04 0.03 0.09 0.02 0.07 0.09 0.07 0.14 inf 0.08 0.06 D1 0.05 0.06 0.06 0.04 0.07 0.05 0.09 0.05 0.06 0.08 inf 0.03 D2 0.02 0.05 0.05 0.03 0.04 0.03 0.07 0.03 0.08 0.06 0.03 inf</pre>

Tabla 19: Resultados de la creación de la matriz de costos usando la distancia *Haversine*.

Versión	Tipo de distancia: <i>Haversine</i>
Original	<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 22:58:56) Matriz de costos - Haversine: C1 Infinity 5.14 3.73 1.88 0.39 2.91 6.97 2.58 1.55 2.00 2.52 0.70 C2 5.14 Infinity 1.43 6.95 4.89 8.05 12.11 7.72 5.32 3.25 2.81 4.56 C3 3.73 1.43 Infinity 5.55 3.46 6.64 10.70 6.31 4.05 1.83 1.57 3.17 C4 1.88 6.95 5.55 Infinity 2.23 1.19 5.23 0.90 2.06 3.87 4.20 2.39 C5 0.39 4.89 3.46 2.23 Infinity 3.21 7.24 2.88 1.85 1.68 2.37 0.73 C6 2.91 8.05 6.64 1.19 3.21 Infinity 4.07 0.33 3.25 4.89 5.35 3.51 C7 6.97 12.11 10.70 5.23 7.24 4.07 Infinity 4.40 7.19 8.92 9.42 7.58 C8 2.58 7.72 6.31 0.90 2.88 0.33 4.40 Infinity 2.96 4.56 5.02 3.18 C9 1.55 5.32 4.05 2.06 1.85 3.25 7.19 2.96 Infinity 2.80 2.51 1.31 C10 2.00 3.25 1.83 3.87 1.68 4.89 8.92 4.56 2.80 Infinity 1.33 1.60 D1 2.52 2.81 1.57 4.20 2.37 5.35 9.42 5.02 2.51 1.33 Infinity 1.86 D2 0.70 4.56 3.17 2.39 0.73 3.51 7.58 3.18 1.31 1.60 1.86 Infinity</pre>
Actualizada	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 22:5 Matriz de costos - Haversine: C1 Infinity 5.14 3.73 1.88 0.39 2.91 6.97 2.58 1.55 2.00 2.52 0.70 C2 5.14 Infinity 1.43 6.95 4.89 8.05 12.11 7.72 5.32 3.25 2.81 4.56 C3 3.73 1.43 Infinity 5.55 3.46 6.64 10.70 6.31 4.05 1.83 1.57 3.17 C4 1.88 6.95 5.55 Infinity 2.23 1.19 5.23 0.90 2.06 3.87 4.20 2.39 C5 0.39 4.89 3.46 2.23 Infinity 3.21 7.24 2.88 1.85 1.68 2.37 0.73 C6 2.91 8.05 6.64 1.19 3.21 Infinity 4.07 0.33 3.25 4.89 5.35 3.51 C7 6.97 12.11 10.70 5.23 7.24 4.07 Infinity 4.40 7.19 8.92 9.42 7.58 C8 2.58 7.72 6.31 0.90 2.88 0.33 4.40 Infinity 2.96 4.56 5.02 3.18 C9 1.55 5.32 4.05 2.06 1.85 3.25 7.19 2.96 Infinity 2.80 2.51 1.31 C10 2.00 3.25 1.83 3.87 1.68 4.89 8.92 4.56 2.80 Infinity 1.33 1.60 D1 2.52 2.81 1.57 4.20 2.37 5.35 9.42 5.02 2.51 1.33 Infinity 1.86 D2 0.70 4.56 3.17 2.39 0.73 3.51 7.58 3.18 1.31 1.60 1.86 Infinity</pre>
Python	<pre>Matriz de costos - Haversine: C1 inf 5.14 3.73 1.88 0.39 2.91 6.97 2.58 1.55 2.00 2.52 0.70 C2 5.14 inf 1.43 6.95 4.89 8.05 12.11 7.72 5.32 3.25 2.81 4.56 C3 3.73 1.43 inf 5.55 3.46 6.64 10.70 6.31 4.05 1.83 1.57 3.17 C4 1.88 6.95 5.55 inf 2.23 1.19 5.23 0.90 2.06 3.87 4.20 2.39 C5 0.39 4.89 3.46 2.23 inf 3.21 7.24 2.88 1.85 1.68 2.37 0.73 C6 2.91 8.05 6.64 1.19 3.21 inf 4.07 0.33 3.25 4.89 5.35 3.50 C7 6.97 12.11 10.70 5.23 7.24 4.07 inf 4.40 7.19 8.92 9.42 7.58 C8 2.58 7.72 6.31 0.90 2.88 0.33 4.40 inf 2.96 4.56 5.02 3.18 C9 1.55 5.32 4.05 2.06 1.85 3.25 7.19 2.96 inf 2.80 2.51 1.31 C10 2.00 3.25 1.83 3.87 1.68 4.89 8.92 4.56 2.80 inf 1.33 1.60 D1 2.52 2.81 1.57 4.20 2.37 5.35 9.42 5.02 2.51 1.33 inf 1.86 D2 0.70 4.56 3.17 2.39 0.73 3.50 7.58 3.18 1.31 1.60 1.86 inf</pre>

Tabla 20: Resultados de la creación de la matriz de costos usando la distancia *Manhattan*.

Versión	Tipo de distancia: <i>Manhattan</i>
Original	<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 22:58:56) Matriz de costos - Manhattan: C1 Infinity 0.05 0.05 0.06 0.02 0.04 0.07 0.04 0.11 0.06 0.07 0.03 C2 0.05 Infinity 0.02 0.11 0.06 0.10 0.13 0.09 0.16 0.06 0.08 0.07 C3 0.05 0.02 Infinity 0.11 0.03 0.09 0.12 0.09 0.15 0.04 0.07 0.07 C4 0.06 0.11 0.11 Infinity 0.08 0.03 0.08 0.03 0.08 0.11 0.04 0.04 C5 0.02 0.06 0.03 0.08 Infinity 0.07 0.10 0.06 0.13 0.03 0.08 0.05 C6 0.04 0.10 0.09 0.03 0.07 Infinity 0.04 0.01 0.11 0.10 0.07 0.04 C7 0.07 0.13 0.12 0.08 0.10 0.04 Infinity 0.04 0.16 0.13 0.12 0.08 C8 0.04 0.09 0.09 0.03 0.06 0.01 0.04 Infinity 0.11 0.09 0.07 0.04 C9 0.11 0.16 0.15 0.08 0.13 0.11 0.16 0.11 Infinity 0.16 0.08 0.09 C10 0.06 0.06 0.04 0.11 0.03 0.10 0.13 0.09 0.16 Infinity 0.09 0.07 D1 0.07 0.08 0.07 0.04 0.08 0.07 0.12 0.07 0.08 0.09 0.04 Infinity 0.04 D2 0.03 0.07 0.07 0.04 0.05 0.04 0.08 0.04 0.09 0.07 0.04 Infinity</pre>
Actualizada	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 22: Matriz de costos - Manhattan: C1 Infinity 0.05 0.05 0.06 0.02 0.04 0.07 0.04 0.11 0.06 0.07 0.03 C2 0.05 Infinity 0.02 0.11 0.06 0.10 0.13 0.09 0.16 0.06 0.08 0.07 C3 0.05 0.02 Infinity 0.11 0.03 0.09 0.12 0.09 0.15 0.04 0.07 0.07 C4 0.06 0.11 0.11 Infinity 0.08 0.03 0.08 0.03 0.08 0.11 0.04 0.04 C5 0.02 0.06 0.03 0.08 Infinity 0.07 0.10 0.06 0.13 0.03 0.08 0.05 C6 0.04 0.10 0.09 0.03 0.07 Infinity 0.04 0.01 0.11 0.10 0.07 0.04 C7 0.07 0.13 0.12 0.08 0.10 0.04 Infinity 0.04 0.16 0.13 0.12 0.08 C8 0.04 0.09 0.09 0.03 0.06 0.01 0.04 Infinity 0.11 0.09 0.07 0.04 C9 0.11 0.16 0.15 0.08 0.13 0.11 0.16 0.11 Infinity 0.16 0.08 0.09 C10 0.06 0.06 0.04 0.11 0.03 0.10 0.13 0.09 0.16 Infinity 0.09 0.07 D1 0.07 0.08 0.07 0.04 0.08 0.07 0.12 0.07 0.08 0.09 0.04 Infinity 0.04 D2 0.03 0.07 0.07 0.04 0.05 0.04 0.08 0.04 0.09 0.07 0.04 Infinity</pre>
Python	<pre>Matriz de costos - Manhattan: C1 inf 0.05 0.05 0.06 0.02 0.04 0.07 0.04 0.11 0.06 0.07 0.03 C2 0.05 inf 0.02 0.11 0.06 0.10 0.13 0.09 0.16 0.06 0.08 0.07 C3 0.05 0.02 inf 0.11 0.03 0.09 0.12 0.09 0.15 0.04 0.07 0.07 C4 0.06 0.11 0.11 inf 0.08 0.03 0.08 0.03 0.08 0.11 0.04 0.04 C5 0.02 0.06 0.03 0.08 inf 0.07 0.10 0.06 0.13 0.03 0.08 0.05 C6 0.04 0.10 0.09 0.03 0.07 inf 0.04 0.01 0.11 0.10 0.07 0.04 C7 0.07 0.13 0.12 0.08 0.10 0.04 inf 0.04 0.16 0.13 0.12 0.08 C8 0.04 0.09 0.09 0.03 0.06 0.01 0.04 inf 0.11 0.09 0.07 0.04 C9 0.11 0.16 0.15 0.08 0.13 0.11 0.16 0.11 inf 0.16 0.08 0.09 C10 0.06 0.06 0.04 0.11 0.03 0.10 0.13 0.09 0.16 inf 0.09 0.07 D1 0.07 0.08 0.07 0.04 0.08 0.07 0.12 0.07 0.08 0.09 inf 0.04 D2 0.03 0.07 0.07 0.04 0.05 0.04 0.08 0.04 0.09 0.07 0.04 inf</pre>

Tabla 21: Resultados de la creación de la matriz de costos usando la distancia *Chebyshev*.

Versiones	Tipo de distancia: <i>Chebyshev</i>
Original	<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 22:58:56) Matriz de costos - Chebyshev: C1 Infinity 0.05 0.03 0.04 0.02 0.03 0.06 0.02 0.10 0.04 0.04 0.02 C2 0.05 Infinity 0.01 0.06 0.04 0.07 0.11 0.07 0.11 0.03 0.05 0.04 C3 0.03 0.01 Infinity 0.06 0.03 0.06 0.10 0.06 0.12 0.02 0.06 0.04 C4 0.04 0.06 0.06 Infinity 0.06 0.02 0.05 0.03 0.06 0.08 0.04 0.02 C5 0.02 0.04 0.03 0.06 Infinity 0.04 0.07 0.04 0.12 0.02 0.06 0.04 C6 0.03 0.07 0.06 0.02 0.04 Infinity 0.04 0.00 0.09 0.06 0.05 0.03 C7 0.06 0.11 0.10 0.05 0.07 0.04 Infinity 0.04 0.09 0.08 0.08 0.07 C8 0.02 0.07 0.06 0.03 0.04 0.00 0.04 Infinity 0.09 0.05 0.05 0.03 C9 0.10 0.11 0.12 0.06 0.12 0.09 0.09 0.09 Infinity 0.14 0.06 0.08 C10 0.04 0.03 0.02 0.08 0.02 0.06 0.08 0.05 0.14 Infinity 0.08 0.06 D1 0.04 0.05 0.06 0.04 0.06 0.05 0.08 0.05 0.06 0.08 Infinity 0.02 D2 0.02 0.04 0.04 0.02 0.04 0.03 0.07 0.03 0.08 0.06 0.02 Infinity</pre>
Actualizada	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 22: Matriz de costos - Chebyshev: C1 Infinity 0.05 0.03 0.04 0.02 0.03 0.06 0.02 0.10 0.04 0.04 0.02 C2 0.05 Infinity 0.01 0.06 0.04 0.07 0.11 0.07 0.11 0.03 0.05 0.04 C3 0.03 0.01 Infinity 0.06 0.03 0.06 0.10 0.06 0.12 0.02 0.06 0.04 C4 0.04 0.06 0.06 Infinity 0.06 0.02 0.05 0.03 0.06 0.08 0.04 0.02 C5 0.02 0.04 0.03 0.06 Infinity 0.04 0.07 0.04 0.12 0.02 0.06 0.04 C6 0.03 0.07 0.06 0.02 0.04 Infinity 0.04 0.00 0.09 0.06 0.05 0.03 C7 0.06 0.11 0.10 0.05 0.07 0.04 Infinity 0.04 0.09 0.08 0.08 0.07 C8 0.02 0.07 0.06 0.03 0.04 0.00 0.04 Infinity 0.09 0.05 0.05 0.03 C9 0.10 0.11 0.12 0.06 0.12 0.09 0.09 0.09 0.09 Infinity 0.14 0.06 0.08 C10 0.04 0.03 0.02 0.08 0.02 0.06 0.08 0.05 0.14 Infinity 0.08 0.06 D1 0.04 0.05 0.06 0.04 0.06 0.05 0.08 0.05 0.06 0.08 Infinity 0.02 D2 0.02 0.04 0.04 0.02 0.04 0.03 0.07 0.03 0.08 0.06 0.02 Infinity</pre>
Python	<pre>Matriz de costos - Chebyshev: C1 inf 0.05 0.03 0.04 0.02 0.03 0.06 0.02 0.10 0.04 0.04 0.02 C2 0.05 inf 0.01 0.06 0.04 0.07 0.11 0.07 0.11 0.03 0.05 0.04 C3 0.03 0.01 inf 0.06 0.03 0.06 0.10 0.06 0.12 0.02 0.06 0.04 C4 0.04 0.06 0.06 inf 0.06 0.02 0.05 0.03 0.06 0.08 0.04 0.02 C5 0.02 0.04 0.03 0.06 inf 0.04 0.07 0.04 0.12 0.02 0.06 0.04 C6 0.03 0.07 0.06 0.02 0.04 inf 0.04 0.00 0.09 0.06 0.05 0.03 C7 0.06 0.11 0.10 0.05 0.07 0.04 inf 0.04 0.09 0.08 0.08 0.07 C8 0.02 0.07 0.06 0.03 0.04 0.00 0.04 inf 0.09 0.05 0.05 0.03 C9 0.10 0.11 0.12 0.06 0.12 0.09 0.09 0.09 inf 0.14 0.06 0.08 C10 0.04 0.03 0.02 0.08 0.02 0.06 0.08 0.05 0.14 inf 0.08 0.06 D1 0.04 0.05 0.06 0.04 0.06 0.05 0.08 0.05 0.06 0.08 inf 0.02 D2 0.02 0.04 0.04 0.02 0.04 0.03 0.07 0.03 0.08 0.06 0.02 inf</pre>

Tabla 22: Resultados de la creación de la matriz de costos usando la distancia Real.

Tipo de distancia: Real	
Versiones	
Actualizada	Python
<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 22:56:48) Matriz de costos - Real: C1 Infinity 7.48 5.48 6.62 2.85 3.67 8.39 3.31 15.68 6.05 6.51 3.67 C2 7.48 Infinity 2.36 11.42 6.50 10.39 15.02 10.04 16.94 5.08 8.15 8.84 C3 5.48 2.36 Infinity 11.29 4.14 8.03 12.66 7.67 16.81 3.82 8.02 6.47 C4 6.62 11.42 11.29 Infinity 8.43 5.08 8.90 4.72 10.16 12.20 4.29 4.15 C5 2.85 6.50 4.14 8.43 Infinity 6.22 9.31 5.87 17.62 3.20 8.83 5.70 C6 3.67 10.39 8.03 5.08 6.22 Infinity 5.98 0.46 16.21 9.36 7.04 3.82 C7 8.39 15.02 12.66 8.90 9.31 5.98 Infinity 5.85 21.65 12.25 12.74 9.29 C8 3.31 10.04 7.67 4.72 5.87 0.46 5.85 Infinity 15.86 9.00 6.68 3.46 C9 15.68 16.94 16.81 10.16 17.62 16.21 21.65 15.86 Infinity 19.36 9.99 12.73 C10 6.05 5.08 3.82 12.20 3.20 9.36 12.25 9.00 19.36 Infinity 11.03 9.07 D1 6.51 8.15 8.02 4.29 8.83 7.04 12.74 6.68 9.99 11.03 Infinity 3.54 D2 3.67 8.84 6.47 4.15 5.70 3.82 9.29 3.46 12.73 9.07 3.54 Infinity</pre>	<pre>Matriz de costos - Real: C1 inf 7.48 5.48 6.62 2.85 3.67 8.39 3.31 15.68 6.05 6.51 3.67 C2 7.48 inf 2.36 11.42 6.50 10.39 15.02 10.04 16.94 5.08 8.15 8.84 C3 5.48 2.36 inf 11.29 4.14 8.03 12.66 7.67 16.81 3.82 8.02 6.47 C4 6.62 11.42 11.29 inf 8.43 5.08 8.90 4.72 10.16 12.20 4.29 4.15 C5 2.85 6.50 4.14 8.43 inf 6.22 9.31 5.87 17.62 3.20 8.83 5.70 C6 3.67 10.39 8.03 5.08 6.22 inf 5.98 0.46 16.21 9.36 7.04 3.82 C7 8.39 15.02 12.66 8.90 9.31 5.98 inf 5.85 21.65 12.25 12.74 9.29 C8 3.31 10.04 7.67 4.72 5.87 0.46 5.85 inf 15.86 9.00 6.68 3.46 C9 15.68 16.94 16.81 10.16 17.62 16.21 21.65 15.86 inf 19.36 9.99 12.73 C10 6.05 5.08 3.82 12.20 3.20 9.36 12.25 9.00 19.36 inf 11.03 9.07 D1 6.51 8.15 8.02 4.29 8.83 7.04 12.74 6.68 9.99 11.03 inf 3.54 D2 3.67 8.84 6.47 4.15 5.70 3.82 9.29 3.46 12.73 9.07 3.54 inf</pre>

De este experimento se puede concluir que ambas versiones de la biblioteca son versátiles y pueden adaptarse para resolver la fase de asignación de clientes a depósitos en diversos contextos, ya que cada tipo de distancia tiene sus propias ventajas dependiendo de la naturaleza del problema y los requisitos específicos de modelado. En particular, el tipo de distancia Real presenta un patrón notablemente diferente, con valores de distancia mucho más altos en comparación con las otras métricas. Esto indica que las distancias reales, probablemente modeladas considerando obstáculos geográficos y otros factores reales, resultan más complejas y largas que las medidas de distancia aproximadas. Por lo tanto, la elección de la métrica dependerá de si el objetivo es un modelo más simplificado o una representación más precisa del entorno real.

### 3.4. Experimento para validar el proceso de asignación de los algoritmos

En esta sección se realiza una comparación entre tres versiones de **BHAVRP**: la versión base de la biblioteca antes de realizar las modificaciones, la versión actualizada en Java, y la nueva versión desarrollada en Python. El experimento tiene como objetivo analizar el desempeño de las tres implementaciones ejecutando los algoritmos de asignación que cada una incluye.

La evaluación se centra en la consistencia de los resultados. Para los algoritmos deterministas, una sola ejecución es suficiente para comprobar que en las tres versiones:

- Cada algoritmo genera el mismo número de clústeres.
- Cada clúster contiene la misma cantidad de clientes asignados.
- Los clientes asignados a cada clúster coinciden.
- Los clientes deben estar ordenados de la misma manera en los clústeres.

En la Tabla 23 se muestran todos los algoritmos que son ejecutados, clasificados como deterministas o aleatorios según su comportamiento.

Tabla 23: Clasificación de algoritmos en BHAVRP por tipo de comportamiento [23].

Algoritmo	Comportamiento	
	Determinista	Aleatorio
<i>BestCyclicAssignment (BCA)</i>	X	
<i>BestNearest (BN)</i>	X	
<i>Clara (CLARA)</i>	X	
<i>CoefficientPropagation (CP)</i>	X	
<i>CyclicAssignment (CA)</i>	X	
<i>FarthestFirst (FF)</i>	X	
<i>Kmeans (KM)</i>	X	
<i>NearestByCustomer (NBC)</i>	X	
<i>NearestByDepot (NBD)</i>	X	
<i>PAM</i>	X	
<i>Parallel (P)</i>	X	
<i>RandomByElement (RBE)</i>		X
<i>RandomSequentialCyclic (RSC)</i>		X
<i>SequentialCyclic (SC)</i>	X	
<i>Simplified (S)</i>	X	
<i>Sweep (SW)</i>	X	
<i>ThreeCriteriaClustering (3CC)</i>	X	
<i>UPGMC</i>	X	

Todos los algoritmos mencionados anteriormente se ejecutan con las siguientes configuraciones:

- Número de iteraciones: 1
- Tipo de distancia aproximada: *Euclidean*

### 3.4.1. Análisis de los resultados de los algoritmos deterministas

El análisis se centra en verificar la consistencia de las soluciones generadas por cada algoritmo, en las tres versiones de **BHAVRP**. Para realizar esta comparación se ejecutan los algoritmos sobre la instancia seleccionada en la subsección 3.2.1, y se registran los resultados obtenidos. Los algoritmos al ser deterministas, no tienen decisiones aleatorias, por lo que deben producir exactamente el mismo resultado. Estos resultados permiten validar si las modificaciones realizadas y la migración a Python preservan la funcionalidad y la precisión de las soluciones entregadas por los algoritmos deterministas de la biblioteca.

A continuación, se presentan los resultados obtenidos de los algoritmos, ejecutados en las tres versiones de la biblioteca. Para cada algoritmo se muestran: la cantidad de clústeres creados, la demanda total de cada clúster, la cantidad de clientes asignados a cada clúster, la lista de los identificadores de los clientes asignados al clúster, en el caso de que queden clústeres sin asignación de clientes, se muestra su identificador, y en el caso que queden clientes sin asignar, se muestran sus identificadores.

Tabla 24: Resultados de la ejecución del algoritmo *Best Cyclic Assignment*.

Algoritmo: BCA		
Versiones		
Original	Actualizada	Python
<pre>VERSION: ORIGINAL HEURÍSTICA: BestCyclicAssignment  ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 214.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14  ID DE LOS ELEMENTOS: [29, 21, 34, 30, 10, 39, 33, 45, 5, 38, 20, 35, 36, 3]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 126.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10  ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 13, 40]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 149.0 TOTAL DE ELEMENTOS DEL CLUSTER : 6  ID DE LOS ELEMENTOS: [46, 12, 47, 18, 14, 25]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 288.0 TOTAL DE ELEMENTOS DEL CLUSTER : 20  ID DE LOS ELEMENTOS: [49, 9, 50, 16, 2, 11, 32, 1, 22, 28, 31, 8, 26, 7, 23, 48, 27, 6, 24, 43]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: ACTUALIZADA HEURÍSTICA: BestCyclicAssignment  ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 214.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14  ID DE LOS ELEMENTOS: [29, 21, 34, 30, 10, 39, 33, 45, 5, 38, 20, 35, 36, 3]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 126.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10  ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 13, 40]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 149.0 TOTAL DE ELEMENTOS DEL CLUSTER : 6  ID DE LOS ELEMENTOS: [46, 12, 47, 18, 14, 25]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 288.0 TOTAL DE ELEMENTOS DEL CLUSTER : 20  ID DE LOS ELEMENTOS: [49, 9, 50, 16, 2, 11, 32, 1, 22, 28, 31, 8, 26, 7, 23, 48, 27, 6, 24, 43]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: BestCyclicAssignment  ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 214.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14  ID DE LOS ELEMENTOS: [29, 21, 34, 30, 10, 39, 33, 45, 5, 38, 20, 35, 36, 3]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 126.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10  ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 13, 40]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 149.0 TOTAL DE ELEMENTOS DEL CLUSTER : 6  ID DE LOS ELEMENTOS: [46, 12, 47, 18, 14, 25]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 288.0 TOTAL DE ELEMENTOS DEL CLUSTER : 20  ID DE LOS ELEMENTOS: [49, 9, 50, 16, 2, 11, 32, 1, 22, 28, 31, 8, 26, 7, 23, 48, 27, 6, 24, 43]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 25: Resultados de la ejecución del algoritmo *Best Nearest*.

<b>Algoritmo: BN</b> <b>Versiones</b>		
<b>Original</b>	<b>Actualizada</b>	<b>Python</b>
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 11:53:46) VERSION: ORIGINAL HEURÍSTICA: BestNearest ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17 ----- ID DE LOS ELEMENTOS: [46, 27, 12, 47, 32, 6, 11, 1, 48, 14, 8, 23, 24, 7, 26, 31, 43] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ----- ID DE LOS ELEMENTOS: [4, 17, 19, 42, 41, 44, 37, 18, 13, 15, 40, 45, 25] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [49, 9, 38, 30, 10, 50, 5, 16, 34, 39, 33] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9 ----- ID DE LOS ELEMENTOS: [29, 21, 20, 2, 35, 3, 36, 22, 28] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 11:51:40) VERSION: ACTUALIZADA HEURÍSTICA: BestNearest ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17 ----- ID DE LOS ELEMENTOS: [46, 27, 12, 47, 32, 6, 11, 1, 48, 14, 8, 23, 24, 7, 26, 31, 43] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ----- ID DE LOS ELEMENTOS: [4, 17, 19, 42, 41, 44, 37, 18, 13, 15, 40, 45, 25] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [49, 9, 38, 30, 10, 50, 5, 16, 34, 39, 33] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9 ----- ID DE LOS ELEMENTOS: [29, 21, 20, 2, 35, 3, 36, 22, 28] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: BestNearest ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17 ----- ID DE LOS ELEMENTOS: [46, 27, 12, 47, 32, 6, 11, 1, 48, 14, 8, 23, 24, 7, 26, 31, 43] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ----- ID DE LOS ELEMENTOS: [4, 17, 19, 42, 41, 44, 37, 18, 13, 15, 40, 45, 25] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [49, 9, 38, 30, 10, 50, 5, 16, 34, 39, 33] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9 ----- ID DE LOS ELEMENTOS: [29, 21, 20, 2, 35, 3, 36, 22, 28] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 26: Resultados de la ejecución del algoritmo Clara.

<b>Algoritmo: CLARA</b>		
<b>Versiones</b>		
<b>Original</b>	<b>Actualizada</b>	<b>Python</b>
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 12:44:58) VERSION: ORIGINAL HEURÍSTICA: CLARA ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 280.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17 ID DE LOS ELEMENTOS: [17, 4, 18, 37, 44, 47, 12, 15, 42, 5, 46, 19, 45, 41, 13, 33, 40] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 222.0 TOTAL DE ELEMENTOS DEL CLUSTER : 15 ID DE LOS ELEMENTOS: [48, 6, 26, 14, 27, 8, 23, 7, 1, 22, 24, 31, 28, 43, 25] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 203.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [16, 11, 50, 9, 2, 29, 38, 49, 32, 20, 3, 35, 36] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 72.0 TOTAL DE ELEMENTOS DEL CLUSTER : 5 ID DE LOS ELEMENTOS: [34, 30, 21, 10, 39] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 12:05:49) VERSION: ACTUALIZADA HEURÍSTICA: CLARA ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 280.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17 ID DE LOS ELEMENTOS: [17, 4, 18, 37, 44, 47, 12, 15, 42, 5, 46, 19, 45, 41, 13, 33, 40] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 222.0 TOTAL DE ELEMENTOS DEL CLUSTER : 15 ID DE LOS ELEMENTOS: [48, 6, 26, 14, 27, 8, 23, 7, 1, 22, 24, 31, 28, 43, 25] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 203.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [16, 11, 50, 9, 2, 29, 38, 49, 32, 20, 3, 35, 36] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 72.0 TOTAL DE ELEMENTOS DEL CLUSTER : 5 ID DE LOS ELEMENTOS: [34, 30, 21, 10, 39] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: CLARA ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 280.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17 ID DE LOS ELEMENTOS: [17, 4, 18, 37, 44, 47, 12, 15, 42, 5, 46, 19, 45, 41, 13, 33, 40] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 222.0 TOTAL DE ELEMENTOS DEL CLUSTER : 15 ID DE LOS ELEMENTOS: [48, 6, 26, 14, 27, 8, 23, 7, 1, 22, 24, 31, 28, 43, 25] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 203.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [16, 11, 50, 9, 2, 29, 38, 49, 32, 20, 3, 35, 36] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 72.0 TOTAL DE ELEMENTOS DEL CLUSTER : 5 ID DE LOS ELEMENTOS: [34, 30, 21, 10, 39] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 27: Resultados de la ejecución del algoritmo *Coefficient Propagation*.

<b>Algoritmo: CP</b> <b>Versiones</b>		
<b>Original</b>	<b>Actualizada</b>	<b>Python</b>
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 12:56:06) VERSION: ORIGINAL HEURÍSTICA: CoefficientPropagation ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 266.0 TOTAL DE ELEMENTOS DEL CLUSTER : 16  ID DE LOS ELEMENTOS: [46, 12, 47, 27, 48, 6, 8, 26, 31, 28, 23, 7, 24, 14, 25, 43]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 200.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 45, 33, 18, 19, 41, 42, 13, 40]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 230.0 TOTAL DE ELEMENTOS DEL CLUSTER : 15  ID DE LOS ELEMENTOS: [49, 9, 50, 16, 34, 30, 38, 11, 32, 1, 22, 5, 10, 2, 39]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 81.0 TOTAL DE ELEMENTOS DEL CLUSTER : 6  ID DE LOS ELEMENTOS: [29, 21, 20, 35, 36, 3]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 12:37:55) ----- VERSION: ACTUALIZADA HEURÍSTICA: CoefficientPropagation ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 266.0 TOTAL DE ELEMENTOS DEL CLUSTER : 16  ID DE LOS ELEMENTOS: [46, 12, 47, 27, 48, 6, 8, 26, 31, 28, 23, 7, 24, 14, 25, 43]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 200.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 45, 33, 18, 19, 41, 42, 13, 40]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 230.0 TOTAL DE ELEMENTOS DEL CLUSTER : 15  ID DE LOS ELEMENTOS: [49, 9, 50, 16, 34, 30, 38, 11, 32, 1, 22, 5, 10, 2, 39]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 81.0 TOTAL DE ELEMENTOS DEL CLUSTER : 6  ID DE LOS ELEMENTOS: [29, 21, 20, 35, 36, 3]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: CoefficientPropagation ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 266.0 TOTAL DE ELEMENTOS DEL CLUSTER : 16  ID DE LOS ELEMENTOS: [46, 12, 47, 27, 48, 6, 8, 26, 31, 28, 23, 7, 24, 14, 25, 43]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 200.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 45, 33, 18, 19, 41, 42, 13, 40]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 230.0 TOTAL DE ELEMENTOS DEL CLUSTER : 15  ID DE LOS ELEMENTOS: [49, 9, 50, 16, 34, 30, 38, 11, 32, 1, 22, 5, 10, 2, 39]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 81.0 TOTAL DE ELEMENTOS DEL CLUSTER : 6  ID DE LOS ELEMENTOS: [29, 21, 20, 35, 36, 3]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 28: Resultados de la ejecución del algoritmo *Cyclic Assignment*.

Algoritmo: CA		
Versiones		
Original	Actualizada	Python
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 13:13:08) VERSION: ORIGINAL HEURÍSTICA: CyclicAssignment ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 260.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [46, 12, 47, 18, 14, 25, 13, 40, 5, 27, 23, 7, 24] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 169.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 6, 48, 8, 26, 43] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 171.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12 ID DE LOS ELEMENTOS: [49, 9, 50, 16, 2, 11, 32, 1, 22, 28, 31, 3] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12 ID DE LOS ELEMENTOS: [29, 21, 34, 30, 10, 39, 33, 45, 38, 20, 35, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 13:08:25) VERSION: ACTUALIZADA HEURÍSTICA: CyclicAssignment ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 260.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [46, 12, 47, 18, 14, 25, 13, 40, 5, 27, 23, 7, 24] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 169.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 6, 48, 8, 26, 43] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 171.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12 ID DE LOS ELEMENTOS: [49, 9, 50, 16, 2, 11, 32, 1, 22, 28, 31, 3] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12 ID DE LOS ELEMENTOS: [29, 21, 34, 30, 10, 39, 33, 45, 38, 20, 35, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: CyclicAssignment ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 260.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [46, 12, 47, 18, 14, 25, 13, 40, 5, 27, 23, 7, 24] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 169.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 6, 48, 8, 26, 43] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 171.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12 ID DE LOS ELEMENTOS: [49, 9, 50, 16, 2, 11, 32, 1, 22, 28, 31, 3] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12 ID DE LOS ELEMENTOS: [29, 21, 34, 30, 10, 39, 33, 45, 38, 20, 35, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 29: Resultados de la ejecución del algoritmo *Farthest First*.

Algoritmo: FF		
Versiones		
Original	Actualizada	Python
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 14:20:06) VERSION: ORIGINAL HEURÍSTICA: Farthest_First  SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 231.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ID DE LOS ELEMENTOS: [17, 4, 44, 37, 42, 19, 47, 41, 18, 15, 12, 13, 45, 40] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 187.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12 ID DE LOS ELEMENTOS: [23, 48, 6, 7, 24, 27, 14, 8, 26, 43, 46, 25] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 204.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [9, 49, 30, 50, 38, 10, 34, 16, 5, 11, 21, 39, 33] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 155.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [3, 22, 20, 2, 28, 1, 29, 35, 31, 32, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 14:18:55) VERSION: ACTUALIZADA HEURÍSTICA: Farthest_First  SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 231.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ID DE LOS ELEMENTOS: [17, 4, 44, 37, 42, 19, 47, 41, 18, 15, 12, 13, 45, 40] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 187.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12 ID DE LOS ELEMENTOS: [23, 48, 6, 7, 24, 27, 14, 8, 26, 43, 46, 25] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 204.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [9, 49, 30, 50, 38, 10, 34, 16, 5, 11, 21, 39, 33] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 155.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [3, 22, 20, 2, 28, 1, 29, 35, 31, 32, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: FarthestFirst ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 231.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ID DE LOS ELEMENTOS: [17, 4, 44, 37, 42, 19, 47, 41, 18, 15, 12, 13, 45, 40] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 187.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12 ID DE LOS ELEMENTOS: [23, 48, 6, 7, 24, 27, 14, 8, 26, 43, 46, 25] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 204.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [9, 49, 30, 50, 38, 10, 34, 16, 5, 11, 21, 39, 33] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 155.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [3, 22, 20, 2, 28, 1, 29, 35, 31, 32, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 30: Resultados de la ejecución del algoritmo *K-means*.

Algoritmo: KM		
Versiones		
Original	Actualizada	Python
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 14:23:27) VERSION: ORIGINAL HEURÍSTICA: KMEANS ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 239.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 17, 41, 18, 19, 47, 42, 37, 44, 13, 12, 25, 40]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 189.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14  ID DE LOS ELEMENTOS: [48, 6, 23, 27, 8, 7, 1, 26, 24, 32, 14, 46, 31, 43]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 182.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12  ID DE LOS ELEMENTOS: [49, 10, 9, 30, 5, 38, 39, 50, 34, 33, 15, 45]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 167.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [20, 2, 29, 3, 22, 35, 16, 21, 28, 11, 36]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 14:22:16) VERSION: ACTUALIZADA HEURÍSTICA: KMEANS ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 239.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 17, 41, 18, 19, 47, 42, 37, 44, 13, 12, 25, 40]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 189.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14  ID DE LOS ELEMENTOS: [48, 6, 23, 27, 8, 7, 1, 26, 24, 32, 14, 46, 31, 43]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 182.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12  ID DE LOS ELEMENTOS: [49, 10, 9, 30, 5, 38, 39, 50, 34, 33, 15, 45]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 167.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [20, 2, 29, 3, 22, 35, 16, 21, 28, 11, 36]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: Kmeans ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 239.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 17, 41, 18, 19, 47, 42, 37, 44, 13, 12, 25, 40]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 189.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14  ID DE LOS ELEMENTOS: [48, 6, 23, 27, 8, 7, 1, 26, 24, 32, 14, 46, 31, 43]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 182.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12  ID DE LOS ELEMENTOS: [49, 10, 9, 30, 5, 38, 39, 50, 34, 33, 15, 45]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 167.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [20, 2, 29, 3, 22, 35, 16, 21, 28, 11, 36]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 31: Resultados de la ejecución del algoritmo *Nearest By Customer*.

<b>Algoritmo: NBC</b> <b>Versiones</b>		
<b>Original</b>	<b>Actualizada</b>	<b>Python</b>
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 14:32:42) VERSION: ORIGINAL HEURÍSTICA: NearestByCustomer  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [5, 9, 10, 16, 30, 33, 34, 38, 39, 49, 50]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 13, 15, 17, 18, 19, 25, 37, 40, 41, 42, 44, 45]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17  ID DE LOS ELEMENTOS: [1, 6, 7, 8, 11, 12, 14, 23, 24, 26, 27, 31, 32, 43, 46, 47, 48]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9  ID DE LOS ELEMENTOS: [2, 3, 20, 21, 22, 28, 29, 35, 36]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 14:29:44) VERSION: ACTUALIZADA HEURÍSTICA: NearestByCustomer  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [5, 9, 10, 16, 30, 33, 34, 38, 39, 49, 50]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 13, 15, 17, 18, 19, 25, 37, 40, 41, 42, 44, 45]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17  ID DE LOS ELEMENTOS: [1, 6, 7, 8, 11, 12, 14, 23, 24, 26, 27, 31, 32, 43, 46, 47, 48]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9  ID DE LOS ELEMENTOS: [2, 3, 20, 21, 22, 28, 29, 35, 36]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSIÓN: PYTHON HEURÍSTICA: NearestByCustomer  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [5, 9, 10, 16, 30, 33, 34, 38, 39, 49, 50]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 13, 15, 17, 18, 19, 25, 37, 40, 41, 42, 44, 45]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17  ID DE LOS ELEMENTOS: [1, 6, 7, 8, 11, 12, 14, 23, 24, 26, 27, 31, 32, 43, 46, 47, 48]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9  ID DE LOS ELEMENTOS: [2, 3, 20, 21, 22, 28, 29, 35, 36]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 32: Resultados de la ejecución del algoritmo *Nearest By Depot*.

<b>Algoritmo: NBD</b> <b>Versiones</b>		
<b>Original</b>	<b>Actualizada</b>	<b>Python</b>
<pre> VERSION: ORIGINAL HEURÍSTICA: NearestByDepot  SOLUCIÓN: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 215.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [46, 27, 12, 47, 32, 6, 11, 1, 48, 14, 8, 23, 43]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 17, 19, 42, 41, 44, 37, 18, 13, 15, 40, 25, 24]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 179.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12  ID DE LOS ELEMENTOS: [49, 9, 38, 30, 10, 50, 5, 34, 39, 33, 45, 26]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 178.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12  ID DE LOS ELEMENTOS: [29, 21, 20, 2, 16, 35, 3, 36, 22, 28, 31, 7]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0 </pre>	<pre> &lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 15:11:03)  VERSION: ACTUALIZADA HEURÍSTICA: NearestByDepot  SOLUCIÓN: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 215.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [46, 27, 12, 47, 32, 6, 11, 1, 48, 14, 8, 23, 43]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 17, 19, 42, 41, 44, 37, 18, 13, 15, 40, 25, 24]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 179.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12  ID DE LOS ELEMENTOS: [49, 9, 38, 30, 10, 50, 5, 34, 39, 33, 45, 26]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 178.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12  ID DE LOS ELEMENTOS: [29, 21, 20, 2, 16, 35, 3, 36, 22, 28, 31, 7]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0 </pre>	<pre> VERSION: PYTHON HEURÍSTICA: NearestByDepot  SOLUCIÓN: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 215.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [46, 27, 12, 47, 32, 6, 11, 1, 48, 14, 8, 23, 43]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 17, 19, 42, 41, 44, 37, 18, 13, 15, 40, 25, 24]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 179.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12  ID DE LOS ELEMENTOS: [49, 9, 38, 30, 10, 50, 5, 34, 39, 33, 45, 26]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 178.0 TOTAL DE ELEMENTOS DEL CLUSTER : 12  ID DE LOS ELEMENTOS: [29, 21, 20, 2, 16, 35, 3, 36, 22, 28, 31, 7]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0 </pre>

Tabla 33: Resultados de la ejecución del algoritmo *PAM*.

Algoritmo: PAM		
Versiones		
Original	Actualizada	Python
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 15:19:21) ----- VERSION: ORIGINAL HEURÍSTICA: PAM ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 145.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10 ----- ID DE LOS ELEMENTOS: [42, 19, 44, 41, 17, 4, 40, 45, 13, 25] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 206.0 TOTAL DE ELEMENTOS DEL CLUSTER : 15 ----- ID DE LOS ELEMENTOS: [1, 32, 22, 27, 8, 48, 28, 6, 31, 26, 23, 7, 14, 24, 43] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 245.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ----- ID DE LOS ELEMENTOS: [5, 38, 49, 12, 37, 46, 9, 10, 15, 47, 30, 33, 18, 39] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 181.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [2, 16, 29, 11, 20, 50, 21, 3, 35, 34, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 15:18:08) ----- VERSION: ACTUALIZADA HEURÍSTICA: PAM ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 145.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10 ----- ID DE LOS ELEMENTOS: [42, 19, 44, 41, 17, 4, 40, 45, 13, 25] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 206.0 TOTAL DE ELEMENTOS DEL CLUSTER : 15 ----- ID DE LOS ELEMENTOS: [1, 32, 22, 27, 8, 48, 28, 6, 31, 26, 23, 7, 14, 24, 43] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 245.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ----- ID DE LOS ELEMENTOS: [5, 38, 49, 12, 37, 46, 9, 10, 15, 47, 30, 33, 18, 39] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 181.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [2, 16, 29, 11, 20, 50, 21, 3, 35, 34, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: PAM ----- SOLUTION: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 145.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10 ----- ID DE LOS ELEMENTOS: [42, 19, 44, 41, 17, 4, 40, 45, 13, 25] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 206.0 TOTAL DE ELEMENTOS DEL CLUSTER : 15 ----- ID DE LOS ELEMENTOS: [1, 32, 22, 27, 8, 48, 28, 6, 31, 26, 23, 7, 14, 24, 43] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 245.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ----- ID DE LOS ELEMENTOS: [5, 38, 49, 12, 37, 46, 9, 10, 15, 47, 30, 33, 18, 39] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 181.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [2, 16, 29, 11, 20, 50, 21, 3, 35, 34, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 34: Resultados de la ejecución del algoritmo *Parallel*.

Algoritmo: P		
Versiones		
Original	Actualizada	Python
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 15:24:18) VERSION: ORIGINAL HEURÍSTICA: Parallel SOLUTION: CANTIDAD DE CLUSTERS: 4 ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [4, 13, 15, 17, 18, 19, 25, 37, 40, 41, 42, 44, 45] ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17 ID DE LOS ELEMENTOS: [1, 6, 7, 8, 11, 12, 14, 23, 24, 26, 27, 31, 32, 43, 46, 47, 48] ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [5, 9, 10, 16, 30, 33, 34, 38, 39, 49, 50] ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9 ID DE LOS ELEMENTOS: [2, 3, 20, 21, 22, 28, 29, 35, 36] TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 15:24:02) VERSION: ACTUALIZADA HEURÍSTICA: Parallel SOLUTION: CANTIDAD DE CLUSTERS: 4 ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [4, 13, 15, 17, 18, 19, 25, 37, 40, 41, 42, 44, 45] ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17 ID DE LOS ELEMENTOS: [1, 6, 7, 8, 11, 12, 14, 23, 24, 26, 27, 31, 32, 43, 46, 47, 48] ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [5, 9, 10, 16, 30, 33, 34, 38, 39, 49, 50] ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9 ID DE LOS ELEMENTOS: [2, 3, 20, 21, 22, 28, 29, 35, 36] TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: Parallel SOLUTION: CANTIDAD DE CLUSTERS: 4 ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ID DE LOS ELEMENTOS: [4, 13, 15, 17, 18, 19, 25, 37, 40, 41, 42, 44, 45] ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17 ID DE LOS ELEMENTOS: [1, 6, 7, 8, 11, 12, 14, 23, 24, 26, 27, 31, 32, 43, 46, 47, 48] ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [5, 9, 10, 16, 30, 33, 34, 38, 39, 49, 50] ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9 ID DE LOS ELEMENTOS: [2, 3, 20, 21, 22, 28, 29, 35, 36] TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 35: Resultados de la ejecución del algoritmo *Sequential Cyclic*.

Algoritmo: SC		
Versiones		
Original	Actualizada	Python
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 15:43:57) VERSION: ORIGINAL HEURÍSTICA: SequentialCyclic ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 3 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 318.0 TOTAL DE ELEMENTOS DEL CLUSTER : 21 ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 13, 25, 14, 6, 48, 27, 1, 32, 11, 38, 5, 49, 9] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 319.0 TOTAL DE ELEMENTOS DEL CLUSTER : 21 ID DE LOS ELEMENTOS: [46, 12, 47, 18, 24, 23, 7, 26, 8, 31, 28, 3, 20, 35, 36, 29, 21, 50, 16, 22, 10] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 140.0 TOTAL DE ELEMENTOS DEL CLUSTER : 8 ID DE LOS ELEMENTOS: [30, 34, 39, 33, 45, 40, 43, 2] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 1 DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: ID DEL DEPÓSITO: 54</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 15:43:14) VERSION: ACTUALIZADA HEURÍSTICA: SequentialCyclic ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 3 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 318.0 TOTAL DE ELEMENTOS DEL CLUSTER : 21 ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 13, 25, 14, 6, 48, 27, 1, 32, 11, 38, 5, 49, 9] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 319.0 TOTAL DE ELEMENTOS DEL CLUSTER : 21 ID DE LOS ELEMENTOS: [46, 12, 47, 18, 24, 23, 7, 26, 8, 31, 28, 3, 20, 35, 36, 29, 21, 50, 16, 22, 10] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 140.0 TOTAL DE ELEMENTOS DEL CLUSTER : 8 ID DE LOS ELEMENTOS: [30, 34, 39, 33, 45, 40, 43, 2] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 1 DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: ID DEL DEPÓSITO: 54</pre>	<pre>VERSION: PYTHON HEURÍSTICA: SequentialCyclic ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 3 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 318.0 TOTAL DE ELEMENTOS DEL CLUSTER : 21 ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 13, 25, 14, 6, 48, 27, 1, 32, 11, 38, 5, 49, 9] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 319.0 TOTAL DE ELEMENTOS DEL CLUSTER : 21 ID DE LOS ELEMENTOS: [46, 12, 47, 18, 24, 23, 7, 26, 8, 31, 28, 3, 20, 35, 36, 29, 21, 50, 16, 22, 10] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 140.0 TOTAL DE ELEMENTOS DEL CLUSTER : 8 ID DE LOS ELEMENTOS: [30, 34, 39, 33, 45, 40, 43, 2] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 1 DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: ID DEL DEPÓSITO: 54</pre>

Tabla 36: Resultados de la ejecución del algoritmo *Simplified*.

<b>Algoritmo: S</b>		
<b>Versiones</b>		
<b>Original</b>	<b>Actualizada</b>	<b>Python</b>
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 23:21:02) VERSION: ORIGINAL HEURÍSTICA: Simplified  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [19, 40, 42, 41, 44, 13, 4, 17, 37, 15, 18, 45, 25]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17  ID DE LOS ELEMENTOS: [48, 27, 46, 7, 6, 23, 43, 32, 26, 8, 1, 24, 12, 14, 47, 11, 31]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [10, 49, 39, 9, 30, 38, 33, 34, 50, 5, 16]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9  ID DE LOS ELEMENTOS: [35, 36, 20, 29, 3, 21, 2, 28, 22]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (15/01/2025 23:15:18) VERSION: ACTUALIZADA HEURÍSTICA: Simplified  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [19, 40, 42, 41, 44, 13, 4, 17, 37, 15, 18, 45, 25]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17  ID DE LOS ELEMENTOS: [48, 27, 46, 7, 6, 23, 43, 32, 26, 8, 1, 24, 12, 14, 47, 11, 31]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [10, 49, 39, 9, 30, 38, 33, 34, 50, 5, 16]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9  ID DE LOS ELEMENTOS: [35, 36, 20, 29, 3, 21, 2, 28, 22]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSIÓN: PYTHON HEURÍSTICA: Simplified  SOLUCIÓN: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [19, 40, 42, 41, 44, 13, 4, 17, 37, 15, 18, 45, 25]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17  ID DE LOS ELEMENTOS: [48, 27, 46, 7, 6, 23, 43, 32, 26, 8, 1, 24, 12, 14, 47, 11, 31]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [10, 49, 39, 9, 30, 38, 33, 34, 50, 5, 16]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9  ID DE LOS ELEMENTOS: [35, 36, 20, 29, 3, 21, 2, 28, 22]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 37: Resultados de la ejecución del algoritmo Sweep.

<b>Algoritmo: SW</b>		
<b>Versiones</b>		
<b>Original</b>	<b>Actualizada</b>	<b>Python</b>
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 17:04:53)  VERSION: ORIGINAL HEURÍSTICA: Sweep  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 13, 15, 17, 18, 19, 25, 37, 40, 41, 42, 44, 45]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17  ID DE LOS ELEMENTOS: [1, 31, 27, 32, 8, 26, 46, 48, 11, 7, 6, 23, 43, 24, 12, 14, 47]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [34, 9, 50, 30, 16, 49, 38, 10, 39, 5, 33]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9  ID DE LOS ELEMENTOS: [35, 36, 20, 29, 21, 3, 2, 28, 22]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 17:00:52)  VERSION: ACTUALIZADA HEURÍSTICA: Sweep  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13  ID DE LOS ELEMENTOS: [4, 13, 15, 17, 18, 19, 25, 37, 40, 41, 42, 44, 45]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER : 17  ID DE LOS ELEMENTOS: [1, 31, 27, 32, 8, 26, 46, 48, 11, 7, 6, 23, 43, 24, 12, 14, 47]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11  ID DE LOS ELEMENTOS: [34, 9, 50, 30, 16, 49, 38, 10, 39, 5, 33]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER : 9  ID DE LOS ELEMENTOS: [35, 36, 20, 29, 21, 3, 2, 28, 22]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: Sweep  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 205.0 TOTAL DE ELEMENTOS DEL CLUSTER: 13  ID DE LOS ELEMENTOS: [4, 13, 15, 17, 18, 19, 25, 37, 40, 41, 42, 44, 45]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 262.0 TOTAL DE ELEMENTOS DEL CLUSTER: 17  ID DE LOS ELEMENTOS: [1, 31, 27, 32, 8, 26, 46, 48, 11, 7, 6, 23, 43, 24, 12, 14, 47]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 177.0 TOTAL DE ELEMENTOS DEL CLUSTER: 11  ID DE LOS ELEMENTOS: [34, 9, 50, 30, 16, 49, 38, 10, 39, 5, 33]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 133.0 TOTAL DE ELEMENTOS DEL CLUSTER: 9  ID DE LOS ELEMENTOS: [35, 36, 20, 29, 21, 3, 2, 28, 22]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 38: Resultados de la ejecución del algoritmo *Three Criteria Clustering*.

<b>Algoritmo: 3CC</b> <b>Versões</b>		
<b>Original</b>	<b>Actualizada</b>	<b>Python</b>
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 17:09:21) VERSION: ORIGINAL HEURÍSTICA: ThreeCriteriaClustering  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 182.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [6, 23, 7, 43, 24, 48, 27, 26, 8, 14, 25]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 231.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ID DE LOS ELEMENTOS: [19, 40, 41, 42, 13, 4, 17, 44, 47, 18, 37, 12, 15, 45]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 209.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ID DE LOS ELEMENTOS: [49, 10, 39, 30, 9, 5, 38, 33, 34, 50, 11, 16, 21, 46]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 155.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [35, 36, 3, 20, 29, 2, 28, 22, 31, 32, 1]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 17:07:41) VERSION: ACTUALIZADA HEURÍSTICA: ThreeCriteriaClustering  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 182.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [6, 23, 7, 43, 24, 48, 27, 26, 8, 14, 25]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 231.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ID DE LOS ELEMENTOS: [19, 40, 41, 42, 13, 4, 17, 44, 47, 18, 37, 12, 15, 45]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 209.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ID DE LOS ELEMENTOS: [49, 10, 39, 30, 9, 5, 38, 33, 34, 50, 11, 16, 21, 46]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 155.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [35, 36, 3, 20, 29, 2, 28, 22, 31, 32, 1]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: ThreeCriteriaClustering  SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 182.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [6, 23, 7, 43, 24, 48, 27, 26, 8, 14, 25]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 231.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ID DE LOS ELEMENTOS: [19, 40, 41, 42, 13, 4, 17, 44, 47, 18, 37, 12, 15, 45]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 209.0 TOTAL DE ELEMENTOS DEL CLUSTER : 14 ID DE LOS ELEMENTOS: [49, 10, 39, 30, 9, 5, 38, 33, 34, 50, 11, 16, 21, 46]  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 155.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ID DE LOS ELEMENTOS: [35, 36, 3, 20, 29, 2, 28, 22, 31, 32, 1]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 39: Resultados de la ejecución del algoritmo UPGMC.

Algoritmo: UPGMC		
Versiones		
Original	Actualizada	Python
<pre>&lt;terminated&gt; Main [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 17:23:20) VERSION: ORIGINAL HEURÍSTICA: UPGMC ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 266.0 TOTAL DE ELEMENTOS DEL CLUSTER : 16 ----- ID DE LOS ELEMENTOS: [46, 27, 47, 12, 48, 6, 31, 28, 26, 8, 43, 23, 7, 25, 14, 24] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 167.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [4, 42, 44, 15, 37, 17, 41, 19, 18, 40, 13] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 206.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ----- ID DE LOS ELEMENTOS: [49, 9, 34, 30, 5, 10, 50, 16, 38, 11, 45, 33, 39] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 138.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10 ----- ID DE LOS ELEMENTOS: [29, 21, 36, 35, 20, 3, 32, 1, 2, 22] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (14/01/2025 17:15:57) VERSION: ACTUALIZADA HEURÍSTICA: UPGMC ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 266.0 TOTAL DE ELEMENTOS DEL CLUSTER : 16 ----- ID DE LOS ELEMENTOS: [46, 27, 47, 12, 48, 6, 31, 28, 26, 8, 43, 23, 7, 25, 14, 24] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 167.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [4, 42, 44, 15, 37, 17, 41, 19, 18, 40, 13] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 206.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ----- ID DE LOS ELEMENTOS: [49, 9, 34, 30, 5, 10, 50, 16, 38, 11, 45, 33, 39] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 138.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10 ----- ID DE LOS ELEMENTOS: [29, 21, 36, 35, 20, 3, 32, 1, 2, 22] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: UPGMC ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 4 ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 266.0 TOTAL DE ELEMENTOS DEL CLUSTER : 16 ----- ID DE LOS ELEMENTOS: [46, 27, 47, 12, 48, 6, 31, 28, 26, 8, 43, 23, 7, 25, 14, 24] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 167.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [4, 42, 44, 15, 37, 17, 41, 19, 18, 40, 13] ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 206.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ----- ID DE LOS ELEMENTOS: [49, 9, 34, 30, 5, 10, 50, 16, 38, 11, 45, 33, 39] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 138.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10 ----- ID DE LOS ELEMENTOS: [29, 21, 36, 35, 20, 3, 32, 1, 2, 22] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

De los resultados obtenidos, se puede concluir que las tres versiones de **BHAVRP** mantienen una alta consistencia en los resultados de asignación de clientes a los depósitos, reflejando que las modificaciones y la migración a Python no han comprometido la calidad de las soluciones entregadas por los algoritmos deterministas. De esta manera, cada versión se adapta bien a los requerimientos de asignación de clientes a depósitos, asegurando que los algoritmos deterministas sigan generando soluciones eficientes y consistentes a través de las diferentes implementaciones de **BHAVRP**.

### 3.4.2. Análisis de los resultados de los algoritmos no deterministas

El análisis se centra en verificar que los algoritmos no deterministas presentes en las versiones actualizada y nueva de **BHAVRP** funcionen. Para realizar esta comparación se ejecutan los algoritmos sobre la instancia seleccionada en la subsección 3.2.1, y se registran los resultados obtenidos. A continuación, se presentan los resultados obtenidos de los algoritmos, ejecutados en las tres versiones de la biblioteca.

Tabla 40: Resultados de la ejecución del algoritmo *Random By Element*.

Algoritmo: RBE	
Versiones	
Actualizada	Python
<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (16/01/2025 00:31:20) VERSION: ACTUALIZADA HEURÍSTICA: RandomByElement ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 141.0 TOTAL DE ELEMENTOS DEL CLUSTER : 11 ----- ID DE LOS ELEMENTOS: [16, 25, 26, 15, 50, 32, 13, 35, 17, 40, 4]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 291.0 TOTAL DE ELEMENTOS DEL CLUSTER : 16 ----- ID DE LOS ELEMENTOS: [49, 11, 3, 47, 34, 29, 19, 27, 33, 28, 14, 2, 8, 44, 7, 31]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 158.0 TOTAL DE ELEMENTOS DEL CLUSTER : 10 ----- ID DE LOS ELEMENTOS: [45, 43, 6, 18, 23, 46, 9, 38, 36, 20]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 187.0 TOTAL DE ELEMENTOS DEL CLUSTER : 13 ----- ID DE LOS ELEMENTOS: [5, 24, 22, 37, 39, 12, 21, 42, 48, 41, 1, 30, 10]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>	<pre>VERSION: PYTHON HEURÍSTICA: RandomByElement ----- SOLUTION: CANTIDAD DE CLUSTERS: 4  ID CLUSTER: 54 DEMANDA DEL CLUSTER: 250.0 TOTAL DE ELEMENTOS DEL CLUSTER: 18 ----- ID DE LOS ELEMENTOS: [39, 38, 46, 3, 24, 34, 44, 5, 4, 45, 28, 37, 14, 23, 42, 31, 40, 35]  ID CLUSTER: 51 DEMANDA DEL CLUSTER: 245.0 TOTAL DE ELEMENTOS DEL CLUSTER: 13 ----- ID DE LOS ELEMENTOS: [22, 16, 48, 18, 29, 2, 49, 41, 19, 6, 30, 47, 27]  ID CLUSTER: 52 DEMANDA DEL CLUSTER: 94.0 TOTAL DE ELEMENTOS DEL CLUSTER: 6 ----- ID DE LOS ELEMENTOS: [9, 33, 11, 13, 1, 43]  ID CLUSTER: 53 DEMANDA DEL CLUSTER: 188.0 TOTAL DE ELEMENTOS DEL CLUSTER: 13 ----- ID DE LOS ELEMENTOS: [10, 36, 26, 50, 12, 17, 20, 32, 25, 8, 21, 15, 7]  TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 0</pre>

Tabla 41: Resultados de la ejecución del algoritmo *Random Sequential Cyclic*.

Algoritmo: RSC	
Versiones	
Actualizada	Python
<pre>&lt;terminated&gt; MainDePrueba [Java Application] C:\Apps\java\jdk-8u281\bin\javaw.exe (16/01/2025 00:51:17) VERSIÓN: ACTUALIZADA HEURÍSTICA: RandomSequentialCyclic ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 3 ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 318.0 TOTAL DE ELEMENTOS DEL CLUSTER : 21 ----- ID DE LOS ELEMENTOS: [4, 17, 37, 15, 44, 42, 19, 41, 13, 25, 14, 6, 48, 27, 1, 32, 11, 38, 5, 49, 9] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 318.0 TOTAL DE ELEMENTOS DEL CLUSTER : 21 ----- ID DE LOS ELEMENTOS: [29, 21, 50, 16, 2, 22, 28, 31, 8, 26, 7, 23, 24, 43, 18, 47, 12, 46, 10, 30, 36] ----- ID CLUSTER: 52 DEMANDA DEL CLUSTER: 141.0 TOTAL DE ELEMENTOS DEL CLUSTER : 8 ----- ID DE LOS ELEMENTOS: [45, 33, 39, 34, 20, 35, 3, 40] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 1 DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: ----- ID DEL DEPÓSITO: 53</pre>	<pre>VERSIÓN: PYTHON HEURÍSTICA: RandomSequentialCyclic ----- SOLUCIÓN: CANTIDAD DE CLUSTERS: 3 ----- ID CLUSTER: 53 DEMANDA DEL CLUSTER: 318.0 TOTAL DE ELEMENTOS DEL CLUSTER: 22 ----- ID DE LOS ELEMENTOS: [49, 9, 38, 30, 18, 50, 5, 16, 34, 11, 21, 39, 2, 12, 29, 37, 15, 32, 46, 33, 45, 17] ----- ID CLUSTER: 51 DEMANDA DEL CLUSTER: 320.0 TOTAL DE ELEMENTOS DEL CLUSTER: 19 ----- ID DE LOS ELEMENTOS: [27, 47, 6, 1, 18, 48, 4, 14, 22, 8, 23, 25, 44, 24, 7, 26, 13, 31, 19] ----- ID CLUSTER: 54 DEMANDA DEL CLUSTER: 139.0 TOTAL DE ELEMENTOS DEL CLUSTER: 9 ----- ID DE LOS ELEMENTOS: [42, 41, 40, 43, 28, 20, 3, 35, 36] ----- TOTAL DE CLIENTES NO ASIGNADOS: 0 TOTAL DE DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: 1 DEPÓSITOS SIN ASIGNACIÓN DE CLIENTES: ----- ID DEL DEPÓSITO: 54</pre>

De los resultados obtenidos, se puede concluir que los algoritmos no deterministas funcionan en ambas versiones de **BHAVRP**, teniendo en cuenta que asignan los clientes a los depósitos de manera aleatoria, siempre que el depósito tenga disponibilidad.

### 3.5. Conclusiones parciales

En este capítulo, se ha llevado a cabo una comparación exhaustiva entre las diferentes versiones de **BHAVRP**, enfocándose en su capacidad para ejecutar algoritmos de asignación y en cómo las modificaciones y migraciones impactan el desempeño y la consistencia de los resultados. A continuación, se presentan las conclusiones parciales derivadas de los análisis realizados:

- Las tres versiones de **BHAVRP** demuestran una alta consistencia en los resultados de asignación de clientes a depósitos cuando se utilizan algoritmos deterministas. Esto valida que las modificaciones implementadas, tanto en Java

como en Python, no afectaron la capacidad de estos algoritmos para resolver el problema de asignación.

- Los resultados obtenidos en la sección 3.4 confirman que las implementaciones de los algoritmos deterministas son equivalentes en las tres versiones, dado que generan soluciones idénticas.
- El análisis también ha revelado que, independientemente de las mejoras tecnológicas y la migración entre lenguajes, las tres versiones de **BHAVRP** se adaptan eficientemente para resolver la fase de asignación de clientes a depósitos en diversos contextos. Cada implementación puede ser utilizada dependiendo de los requisitos específicos de tiempo y complejidad de cada problema.
- La generación de la matriz de costos utilizando distancias reales a través del servicio **OSRM** ha sido un avance significativo. Este enfoque permite incorporar datos geográficos precisos y considerar factores como restricciones de tránsito y características específicas de las vías, lo que incrementa la aplicabilidad de BHAVRP en escenarios reales y mejora la calidad de las soluciones obtenidas.

## Conclusiones

Después de dar cumplimiento a los objetivos trazados en este trabajo y a partir de los resultados alcanzados se puede concluir que **BHAVRP** es una herramienta que facilita la resolución de la fase de asignación en problemas de planificación de rutas de vehículos con múltiples depósitos, al implementar una variedad de algoritmos de asignación y técnicas de agrupamiento. Los principios y patrones de diseño implementados en **BHAVRP**, como el uso de interfaces y clases bien definidas, optimizan su organización interna y la eficiencia en el proceso de asignación. El análisis comparativo con otras bibliotecas demuestra que **BHAVRP** tiene un potencial significativo para fortalecerse mediante la incorporación de enfoques modernos y tecnologías emergentes.

El análisis del diseño de la nueva versión de **BHAVRP** permite concluir que se realizaron mejoras significativas tanto en la versión en Java como en la implementación en Python, cada una enfocada en resolver problemas específicos identificados previamente. La actualización en Java corrigió deficiencias de la versión original, mientras que la implementación en Python representó un cambio orientado a ampliar la accesibilidad e integración con ecosistemas de tecnologías modernas. La reestructuración del paquete *heuristic*, basada en el uso de interfaces y clases abstractas, incrementó la flexibilidad de la herramienta en ambas versiones. Además, la adopción de principios de diseño como alta cohesión, bajo acoplamiento y el patrón *Template Method* ha sido esencial para garantizar la mantenibilidad y claridad del código.

El análisis comparativo entre las versiones de **BHAVRP** ha demostrado que las modificaciones implementadas en Java y la migración paralela a Python no comprometen la consistencia de los resultados al ejecutar algoritmos de asignación deterministas. Esto confirma que ambas implementaciones mantienen la capacidad de resolver la fase de asignación de clientes a depósitos. Un avance destacado es la incorporación del servicio **OSRM** para generar matrices de costos basadas en distancias reales, lo que permite integrar datos geográficos precisos y factores como restricciones de tránsito. Esto incrementa significativamente la aplicabilidad de **BHAVRP** en escenarios reales, optimizando la calidad de las soluciones obtenidas y consolidando su utilidad en la planificación logística.

## Recomendaciones

A partir del desarrollo de este trabajo se identifican un conjunto de aspectos a ser considerados en futuras iteraciones de este proceso de investigación. A continuación, se presentan las principales recomendaciones que se derivan de este trabajo:

- Fortalecer el manejo de excepciones y errores en **BHAVRP**.
- Facilitar la carga de instancias de problemas en formatos más diversos como CSV, JSON o XML, adaptados a distintos entornos.
- Permitir la carga de datos en otros formatos y fuentes externas.
- Mejorar la capacidad de exportación de resultados.
- Integrar métricas avanzadas de evaluación de asignaciones.
- Garantizar la implementación correcta del patrón Singleton en **BHAVRP**.
- Incorporar la capacidad de devolver múltiples soluciones en los algoritmos de asignación.
- Realizar pruebas estadísticas sobre los algoritmos con componentes aleatorios, para mejorar la validez de los resultados experimentales.
- Comparar los tiempos de ejecución de las versiones actuales de **BHAVRP**, destacando las diferencias entre las implementaciones en Java y Python.

## Referencias bibliográficas

- [1] G. & R. J. Dantzig, «The Truck Dispatching Problem.,» *Management Science*, vol. 6, nº 1, pp. 80-91, 1959.
- [2] M. R. & J. D. S. Garey, Computers and Intractability: A Guide to the Theory of NP-Completeness., W.H. Freeman., 1979.
- [3] G. Laporte, «The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms.,» *European Journal of Operational Research*, vol. 59, nº 3, pp. 345-358, 2009.
- [4] A. G. G. D. D. A. M. J. P. G. N. G. F. D. Iván Gallego Mateos, «Desarrollo de un método híbrido para la resolución de mdvrp,» *Revista de la Escuela Jacobea de Postgrado*, pp. 45-64, 2013.
- [5] P. & V. D. Toth, The Vehicle Routing Problem., SIAM, 2002.
- [6] M. & P. J. Y. Gendreau, Handbook of Metaheuristics., Springer, 2010.
- [7] J. F. & L. G. Cordeau, The Vehicle Routing Problem., Les Cahiers du GERAD., 2003.
- [8] E. O.-B. Maria Jose Dillarza Ramos, «Heurística de dos fases para resolver el problema de ruteo de vehículos periódico.,» Medellín, 2016.
- [9] M. M. Solomon, «Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints.,» *Operations Research*, vol. 35, nº 2, pp. 254-265, 1987.
- [10] T. C. T. G. & G. M. Vidal, «Heuristics for multi-depot vehicle routing problems.,» *European Journal of Operational Research*, vol. 238, nº 2, pp. 293-303, 2014.
- [11] M. L. Fisher, «The Vehicle Routing Problem: A Taxonomy and Annotated Bibliography.,» *Operations Research*, vol. 42, nº 4, pp. 478-481, 1994.

- [12] J. J. & M.-V. J. M. Salazar-González, «Heuristics for the Multi-Depot Vehicle Routing Problem.,» *Computers & Operations Research*, vol. 37, nº 4, pp. 722-733, 2010.
- [13] N. Christofides, «Worst-case analysis of a new heuristic for the travelling salesman problem.,» *Technical Report, Graduate School of Industrial Administration*, 1976.
- [14] R. S. S. M. S. I. Mohiuddin Ahmed, «The k-means Algoritm: A Comprehensive Survey and Perfomance Evaluation,» Edit Cowan University, 2020.
- [15] D. S. V. Preeti Arora, «Analysis of K-means and K-medoids Algorithm for Big Data,» *Procedia Journal Center*, pp. 507-512, 2016.
- [16] L. & R. P. J. Kaufman, Finding Groups in Data: An Introduction to Cluster Analysis., Wiley-Interscience., 1990.
- [17] «Google OR-Tools Documentation,» [En línea]. Available: <https://developers.google.com/optimization>. [Último acceso: 20 11 2024].
- [18] «VROOM GitHub Repository,» [En línea]. Available: <https://github.com/VROOM-Project/vroom>. [Último acceso: 20 11 2024].
- [19] M. Sukaridhoto, «Sweep Nearest Algorithm for Capacitated Vehicle Routing Problem,» *Proceedings of the IEEE International Conference on Artificial Intelligence and Computational Intelligence (AICI)*, pp. 163-170, 2023.
- [20] G. O. Suite, «Gekko Documentation,» [En línea]. Available: <https://gekko.readthedocs.io/en/latest/>. [Último acceso: 22 11 2024].
- [21] «JSprit: A Java-based library for solving vehicle routing problems,» [En línea]. Available: <https://jsprit.readthedocs.io/en/latest/>. [Último acceso: 22 11 2024].
- [22] T. B. P. O. S. OptaPlanner. [En línea]. Available: <https://www.optaplanner.org/>. [Último acceso: 22 11 2024].
- [23] M. M. Martel, «Biblioteca de Heuristicas de Asignación para el Problema de Planificación de Rutas de Vehículos con Múltiples Depósitos,» Universidad

Tecnológica de la Habana “José Antonio Echeverría” Facultad de Ingeniería Informática, La Habana, 2019.

- [24] N. P. Fonseca, «Incorporación de algoritmos de agrupamiento en la Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos,» Facultad de Ingenieria Informatica Universidad Tecnologica de La Habana "Jose Antonio Echeverria", La Habana, 2022.
- [25] S. C. Gupta, «Euclidean Distance in Optimization Problems,» *Optimization Techniques*, vol. 3, nº 2, pp. 100-110, 2017.
- [26] W. Haversine, «The Haversine Formula: A Mathematical Approach to Calculate the Distance Between Two Points on the Earth,» *Journal of Geodesy*, vol. 11, pp. 107-120, 2002.
- [27] M. K. a. J. P. J. Han, Data Mining: Concepts and Techniques, 3rd ed., Elsevier, 2012.
- [28] B. K. Hachem, «Applications of Chebyshev Distance in Route Optimization Problems,» *Computational Geometry*, vol. 25, pp. 85-92, 2016.
- [29] G. Developers, «Distance Matrix API,» Google for Developers., [En línea]. Available: <https://developers.google.com/maps/documentation/distance-matrix>. [Último acceso: 26 Noviembre 2024].
- [30] G. GmbH, «GraphHopper Directions API,» GraphHopper, [En línea]. Available: <https://www.graphhopper.com/products/>. [Último acceso: 26 Noviembre 2024].
- [31] S. B. M. S. a. U. B. P. Offermann, «Artifact types in information systems design science- a literature review,» *Global Perspectives on Design Science Research: 5th International Conference*, pp. 77-92, 2010.
- [32] E. H. Fernández, «Nueva versión de la Capa de servicio para soluciones a Problemas de Planificación de rutas de Vehículos,» 2016.

- [33] L. D. Subí, «Nueva versión de la biblioteca de heurísticas de construcción para Problemas de Planificación de Rutas de Vehículos,» 2016.
- [34] R. J. N. Reyes, «Herramienta para la Generación de Rutas en el Problema de Transporte Obrero: TransO,» Facultad de Ingeniería Informática (CUJAE), La Habana, 2017.
- [35] W. M. G. J. U. G. a. P. K. S. G. P. R. Jayarathna, «A survey on multi-depot vehicle routing problem,» *International Journal of Engineering Research & Technology*, vol. 10, nº 1, pp. 49-55, 2021.
- [36] L. P. Y. S. y. X. Z. J. Li, «Multi-Depot Heterogeneous Vehicle Routing Optimization for Hazardous Materials Transportation,» *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, nº 1, pp. 120-130, 2024.
- [37] S. S. R. y. A. M. A. S. K. Ghosh, «Multi-Objective Ant Colony Optimization for Multi-Depot Heterogeneous Vehicle Routing Problem,» *Proceedings of the 2019 IEEE International Conference on Soft Computing and Network Security (ICSNS)*, p. 89–95, 2019.
- [38] A. F. J. R. y. P. M. P. F. L. S. Oliveira, «Vehicle routing problems: Methods and applications,» *Computers & Operations Research*, vol. 38, nº 6, pp. 1059-1064, 2011.
- [39] A. H. y. G. L. M. Gendreau, «A tabu search heuristic for the traveling salesman problem,» *Management Science*, vol. 40, nº 10, pp. 1276-1289, 1994.
- [40] P. T. a. D. Vigo, *The Vehicle Routing Problem*, SIAM, 2002.
- [41] B. A. M. D. L. d. K. y. W. G. G. M. V. W. M. P. van den Bergh, «The Dynamic Vehicle Routing Problem: A Case Study,» *Computers & Operations Research*, vol. 41, nº 1, pp. 1-9, 2014.

- [42] A. H. y. G. L. M. Gendreau, «A Tabu Search Heuristic for the Time Windows Vehicle Routing Problem,» *Journal of the Operational Research Society*, vol. 44, nº 6, pp. 583-595, 1993.
- [43] D. Psaraftis, «The Pickup and Delivery Problem with Time Windows,» *Transportation Science*, vol. 22, nº 1, pp. 1-12, 1988.
- [44] A. H. y. G. L. M. Gendreau, «A Tabu Search Heuristic for the Multiple Depot Vehicle Routing Problem with Pickup and Delivery,» *Transportation Science*, vol. 35, nº 2, pp. 250-261, 2001.
- [45] P. T. y. D. Vigo, «The Pickup and Delivery Problem with Time Windows,» *European Journal of Operational Research*, vol. 140, nº 3, pp. 349-360, 2002.
- [46] J. Gonzalez-Feliu, «ResearchGate,» [En línea]. Available: [https://www.researchgate.net/figure/Les-principales-variantes-du-PTV-VRP-en-anglais\\_fig3\\_272481061](https://www.researchgate.net/figure/Les-principales-variantes-du-PTV-VRP-en-anglais_fig3_272481061). [Último acceso: 11 12 2024].
- [47] S. R. a. E. W. B. L. Golden, *The Vehicle Routing Problem: Latest Advances and New Challenges.*, Springer, 2008.
- [48] E.-G. Talbi, *Metaheuristics: From Design to Implementation.*, Hoboken, NJ, USA: Wiley, 2009.
- [49] G. C. a. J. W. Wright, «Scheduling of Vehicles from a Central Depot to a Number of Delivery Points,» *Operations Research*, vol. 12, nº 4, pp. 568-581, 1964.
- [50] E. C. G. L. R. y. J. A. O. C. L. B. Rocha Medina, «Una revisión al estado del arte del problema de ruteo de vehículos: Evolución histórica y métodos de solución,» *Ingeniería*, vol. 16, nº 2, pp. 35-54, 2011.
- [51] J. E. y. J. E. L. Daza, «Algoritmo Meta-heurístico para Resolver el CVRP-HF,» *Proceedings of the Latin American and Caribbean Conference for Engineering and Technology (LACCEI)*, 2011.

- [52] M. W. T. T. y. D. Vigo, *Vehicle Routing: Problems, Methods, and Applications*, SIAM, 2014.
- [53] G. G. y. M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997.
- [54] J. F. G. M. & L. G. Cordeau, «A tabu Search heuristic for periodic and multi-depot vehicle routig problems,» *Networks*, vol. 30, nº 2, pp. 105-119, 1997.
- [55] B. L. T. D., «A two-phase tabu search approach to the location routing problem,» *European Journal of Operational Research*, 1999.
- [56] J. L. G. & B. F. Renaud, «A Tabu Search heuristic for the multi-depot vehicle routing problem,» *Computers & Operations Research*, vol. 23, nº 3, pp. 228-235, 1996.
- [57] P. & L. R. Parthanadee, «Periodic Product Distribution from multi-depot under limited supplies,» *IEE Transactions*, 2006.
- [58] I. G. Mateos, «Desarrollo de un método híbrido para la resolución del MDVRP,» *Revista de la escuela Jacobea de Posgrado*, 2013.
- [59] Y. A. H. Ortiz, «Diseño de un sistema de ruteo de vehículos con múltiples depósitos en empresas de transporte de carga por carretera,» 2016.
- [60] W. W. Cohen, *Data Mining Techniques*, Wiley, 1997.
- [61] M. N. M. y. P. J. F. A. K. Jain, «Data Clustering: A Review,» *ACM Computing Surveys*, vol. 31, nº 3, pp. 264-323, 1999.
- [62] G. L. M. G. X. L. Donghua Yu, «An improved k-medoids algorithm based on step increasing and optimizing medoids,» *Expert Systems with Applications*, pp. 464-473, 2018.
- [63] R. F. S. y. P. S. V. C. A. B. Almeida, «Farthest-First Traversal: An Efficient Sampling Strategy for Data Clustering,» *Proceedings of the SIAM International Conference on Data Mining*, pp. 387-395, 2007.

- [64] L. K. y. P. Rousseeuw, «Clustering Large Applications (CLARA),» de *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.
- [65] H. K. J. S. y. X. X. M. Ester, «A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,» *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226-231, 1996.
- [66] E. H. R. J. R. & V. J. Gamma, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.
- [67] B. Meyer, Object-Oriented Software Construction, 1988.
- [68] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship., Upper Saddle River, NJ, USA: Prentice Hall, 2008.
- [69] R. C. Martin, Agile Software Development: Principles, Patterns, and Practices., Pearson Education, 2003.
- [70] E. H. R. J. R. & V. J. Gamma, Design Patterns: Elements of Reusable Object-Oriented Software., Addison-Wesley, 1994.
- [71] C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development., Pearson Education, 2002.
- [72] N. & H. P. Mladenović, «Variable Neighborhood Search,» *Computers & Operations Research*, vol. 24, nº 11, pp. 1097-1100, 1997.
- [73] L. J. G. Y. y. P. M. M. Minghini, «OpenStreetMap as a multidisciplinary nexus: perspectives, practices and procedures,» *ISPRS International Journal of Geo-Information*, vol. 11, nº 4, p. 230, 2022.
- [74] D. L. y. C. Vetter, «Real-Time Routing with OpenStreetMap Data,» *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS)*, pp. 513-516, 2011.

- [75] V. V. P. Pedregosa, «Scikit-learn: Machine Learning in Python,» *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.
- [76] R. D. Hernández, «Nueva versión de la biblioteca de clases BiCIAM para solucionar problemas multiobjetivo,» 2014.
- [77] L. d. C. M. Lores, «Operadores de mutación basados en heurísticas de construcción para Problemas de Planificación de Rutas de Vehículos,» 2017.
- [78] Project-OSRM, «OSRM: Open Source Routing Machine,» [En línea]. Available: <https://project-osrm.org/>. [Último acceso: 31 12 2024].
- [79] S. S. Distance, «scipy.spatial.distance\_matrix, SciPy v1.14.1 Manual».
- [80] «Geopy: Geocoding, Reverse Geocoding, Distance Calculations,» Python Software Foundation, [En línea]. Available: <https://pypi.org/project/geopy/>. [Último acceso: 05 dic 2024].
- [81] G. Boeing, «Using OSMnx to Investigate OpenStreetMap Data,» Data Science Prophet.
- [82] Mapbox, «Navigation APIs,» [En línea]. Available: <https://docs.mapbox.com/api/navigation/>. [Último acceso: 05 dic 2024].
- [83] OSRM-py, «OSRM-py Documentation, GitHub,» [En línea]. Available: <https://github.com/Project-OSRM/osrm-py>. [Último acceso: 27 11 2024].
- [84] NumPy, «NumPy Documentation,» NumPy, [En línea]. Available: <https://numpy.org/doc/stable/>. [Último acceso: 27 11 2024].
- [85] R. C. Martin, Design Principles and Design Patterns, 2000.
- [86] P. Clements, Documenting Software Architectures: Views and Beyond, 2010.
- [87] A. H. a. D. Thomas, The Pragmatic Programmer: Your Journey to Mastery, 2019.

- [88] D. L. Parnas, «On the Criteria to Be Used in Decomposing Systems into Modules,» *Communications of the ACM*, vol. 15, nº 12, pp. 1053-1058, 1972.
- [89] I. T. P. Eric Ramos Aragon, «Adaptacion de algoritmos de agrupamiento para la asignacion de clientes a depositos en BHAVRP,» Facultad de Ingenieria Informatica, Universidad Tecnologica de La Habana "Jose Antonio Echeverria", Cujae, La Habana, 2022.
- [90] Refactoring.Guru, «Refactoring.Guru,» [En línea]. Available: <https://refactoring.guru/es/>. [Último acceso: 27 12 2024].
- [91] SoftZone, «SoftZone,» SoftZone, [En línea]. Available: <https://www.softzone.es/noticias/open-source/visual-studio-code-1-95/>. [Último acceso: 13 01 2025].
- [92] Python.org, «Python.org,» [En línea]. Available: <https://www.python.org/downloads/>. [Último acceso: 13 01 2025].
- [93] E. Alba, «Networking and emerging optimiztion.,» Universisdad de Malaga, Enero 2014. [En línea]. Available: <https://neo.lcc.uma.es/publications/public2011.html>. [Último acceso: 13 01 2025].
- [94] L. Suarez, «GitHub,» 2022. [En línea]. Available: <https://github.com/lsuarez96/ErgoMoveInstances>. [Último acceso: 14 01 2025].