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

Autor: Eric Ramos Aragón

Tutores: Dr. C. Isis Torres Pérez

Dr. C. Alejandro Rosete Suárez

La Habana, Cuba

30 de noviembre del 2024

Resumen

Abstract

Índice

Introducción 1

Capítulo 1: Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos 9

1.1. Introducción 9

1.2. Especificaciones de BHAVRP 9

1.2.1. Marco teórico 11

1.2.2. Diseño arquitectónico de BHAVRP 16

1.2.3. Principios de diseño implementados en BHAVRP 21

1.2.4. Patrones de diseño implementados en BHAVRP 25

1.3. Bibliotecas para resolver la fase de asignación del MDVRP 31

1.4. Limitaciones y deficiencias que presenta la versión actual de BHAVRP 37

1.5. Conclusiones parciales 40

Capítulo 2: Concepción y materialización 42

2.1. Introducción 42

Capítulo 3: Validación… 43

3.1. Introducción 43

Conclusiones 44

Recomendaciones 45

Índice de figuras

Figura 1: Heurísticas implementadas en BHAVRP. 11
Figura 2: Variantes del VRP. 13
Figura 3: Arquitectura de BHAVRP [23]. 17
Figura 4: Diagrama de clases del paquete Problem. 18
Figura 5: Diagrama de clases del paquete Assignment. 19
Figura 6: Diagrama de clases del paquete Output. 19
Figura 7: Diagrama de clases del paquete Controller. 20
Figura 8: Diagrama de clases del paquete Factory. 20
Figura 9: Patrón Singleton implementado en BHAVRP. 27
Figura 10: Patrón Factory Method implementado en BHAVRP. 28

Índice de tablas

Tabla 1: Principios de diseño SOLID que cumple BHAVRP. 22
Tabla 2: Otros principios de diseño implementados en BHAVRP. 24
Tabla 3: Patrones de diseño GRASP implementados en BHAVRP. 29
Tabla 4: Comparación de otras bibliotecas que resuelven MDVRP. 35

Introducción

Los Problemas de Planificación de Rutas de Vehículos (*Vehicle Routing Problem*, **VRP**) son problemas de optimización combinatoria que consisten en planificar un conjunto de rutas óptimas para una flota de vehículos. 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 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**-duro [2] [3].

Una de las variantes de **VRP** más estudiadas es el Planificación de Rutas de Vehículos con Múltiples Depósitos (*Multi Depot Vehicle Routing Problem*, **MDVRP**) [4] es una extensión del **VRP** clásico (*Capacitated Vehicle Routing Problem*, **CVRP**) [5] y como tal exige que cada vehículo comience y termine 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 no un depósito, sino varios depósitos con ubicaciones diferentes, y los clientes deben ser servidos con los vehículos destinados a uno de los depósitos [6].

EL **MDVRP** se encuentra presente en diversas actividades cotidianas. Ejemplo de estas aplicaciones prácticas son el transporte público, obrero y escolar, la distribución diaria de alimentos y bebidas hacia los puntos de ventas en la ciudad, 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. Es por esta razón que para resolver el problema de manera eficiente han preferido usar métodos

heurísticos y metaheurísticas antes que métodos exactos. Dentro de los métodos heurísticos se emplean mayormente las metaheurísticas 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 [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 y ordenados por depósitos. Por lo tanto, se propone hallar solución a este problema en dos etapas. La primera etapa donde los clientes deben ser asignados a los depósitos; y en la segunda los clientes asignados al mismo depósito deben ser ordenados para su posterior visita con los vehículos pertenecientes al mismo depósito [9] [10].

Para cumplir con la primera etapa de esta estrategia existen diferentes algoritmos, también llamados heurísticas de asignación. Estas heurísticas están clasificadas en [11] [12] [13]:

- Asignación a través de urgencias, donde 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.
- La Asignación cíclica consiste en asignar de forma cíclica un cliente a cada depósito. A esta categoría pertenece el algoritmo de igual nombre Algoritmo Asignación Cíclica.
- En la 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, diversas bibliotecas y herramientas han surgido en el campo, permitiendo una integración eficaz de estos algoritmos en sistemas complejos. Bibliotecas como **Google OR-Tools** [17] y **VROOM** [18] ofrecen soluciones especializadas para el **MDVRP**, centrándose en la primera fase de asignación mediante el uso de técnicas de optimización combinatoria y algoritmos de agrupamiento, como K-means y heurísticas clásicas como el algoritmo *Sweep*. **Google OR-Tools**, por ejemplo, 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. **VROOM**, por otro lado, se enfoca en resolver problemas de planificación de rutas de vehículos mediante algoritmos de optimización rápidos que se pueden personalizar para ajustarse a diversas restricciones de asignación, como capacidades máximas de carga por vehículo, tiempos de entrega específicos (*time windows*), prioridades de clientes, límites en la distancia total recorrida por vehículo, y restricciones de costos asociados al uso de la flota.

Es relevante destacar que muchas de 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. En el caso de **BHAVRP**, se está planeando una nueva versión en Python para aprovechar estas ventajas y ampliar su alcance como un componente de código abierto, adaptándose al contexto actual del desarrollo de este tipo de herramientas [17] [19].

En otros lenguajes como Java, también se han desarrollado herramientas orientadas a resolver la fase de asignación en problemas como el **MDVRP**. Herramientas como **JSprit**

[20] y **OptaPlanner** [21], ofrecen soporte para dividir el problema en etapas que incluyen la asignación inicial de clientes a depósitos y la planificación de rutas. **JSprit**, por ejemplo, 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**) [22] [23].

Este componente 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”, que contiene heurísticas de asignación clásicas y adaptaciones de métodos de agrupamiento para resolver la fase de asignación en la variante **MDVRP**. La variedad de estas heurísticas 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.

Además de la ejecución de estas heurísticas, **BHAVRP** ofrece una serie de funcionalidades clave que la hacen robusta y adaptable, como la carga de ficheros de entrada, permitiendo la carga de datos en formatos estándar como *.data* o *.txt*, y la configuración de parámetros, ofreciendo al usuario la posibilidad de modificar distintos parámetros en las heurísticas para ajustarse a las necesidades específicas del problema. Asimismo, **BHAVRP** permite utilizar las siguientes medidas de distancia aproximadas: *Euclidian* [24], *Haversine* [25]*, Manhattan* [26] y *Chebyshev* [27], lo cual proporciona flexibilidad adicional para aplicar la métrica más adecuada según el contexto de cada instancia del problema.

Para abordar el uso de distancias reales, se planea integrar **OSRMLib**, una biblioteca desarrollada en la Facultad de Ingeniería Informática de la Universidad Tecnológica de La Habana "José Antonio Echeverría". Esta herramienta, basada en **OSRM** (*Open Source Routing Machine*), permite calcular distancias y tiempos de viaje exactos utilizando datos reales de redes viales. Su incorporación ampliará las capacidades de BHAVRP, permitiendo modelar escenarios más realistas y ofreciendo resultados ajustados a las necesidades de casos prácticos que demanden precisión geográfica y eficiencia operativa. Además, existen otras soluciones destacadas, como *Google Distance Matrix API* [28], que proporciona distancias y tiempos de viaje a través de datos en tiempo real, y **GraphHopper** [29], una alternativa de código abierto basada en mapas digitales para cálculos eficientes de rutas y distancias. Aunque estas herramientas no serán implementadas directamente, su funcionalidad será explorada más a fondo en el capítulo 1 para contrastar su aplicabilidad en diferentes contextos.

BHAVRP brinda solución a la fase de asignación de clientes a depósitos en la variante con múltiples depósitos en herramientas desarrolladas en el contexto nacional, como lo es **TransO** [30]. Esta herramienta de software permite la generación de recorridos en el Problema del Transporte Obrero, donde la variante **MDVRP** está presente a la hora de modelar el problema. Aunque la biblioteca pueda integrarse con otras aplicaciones desarrolladas en el lenguaje de programación *Java*, su capacidad de comunicación está restringida a sistemas basados en este lenguaje. Además, la biblioteca presenta ciertas deficiencias técnicas que impactan en su eficacia y flexibilidad, tales como la falta de un manejo consistente de excepciones, la ausencia de un módulo dedicado a la evaluación de calidad en las asignaciones realizadas, la dependencia de cálculos de distancias aproximadas en lugar de reales, lo cual afecta la precisión en la asignación de clientes a depósitos y la falta de soporte para la carga de archivos en un formato estándar. Esto limita su interoperabilidad con aplicaciones desarrolladas en otros lenguajes, lo que representa una deficiencia al impedir una mayor flexibilidad para integrarse con sistemas más diversos.

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. Estos aspectos evidencian la necesidad de reestructurar su arquitectura, no solo para facilitar su mantenimiento a largo plazo, sino también para maximizar su adaptabilidad en un ecosistema de software más amplio y en constante evolución. Se identifica como problema a resolver que las limitaciones de **BHAVRP** restringen su integración y uso, debido a la falta de interoperabilidad con otros lenguajes de programación y a su dependencia de bibliotecas privadas.

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 solvente las limitaciones presentes en la versión actual de la biblioteca en Java.

Para dar cumplimiento al objetivo general se definieron los siguientes objetivos específicos con sus respectivas tareas:

1. Identificar deficiencias, limitaciones y mejoras en el diseño arquitectónico de **BHAVRP**.
    - Analizar la arquitectura y de las funcionalidades brindadas por **BHAVRP** para identificar deficiencias que presenta 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 métodos para calcular distancias reales mediante el uso del API **OSRM**.
    - 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 el patrón *Factory Method* para la creación de aquellos objetos que se requieran en el proceso de resolución de la fase de asignación y para el cálculo de la evaluación de la asignación.
    - Implementar en Python y en *Java* las clases necesarias para modelar métricas de calidad que evalúen el resultado en la asignación de clientes a depósitos en un problema de planificación de rutas de vehículos con múltiples depósitos.
    - Implementar en Python y en *Java* las clases necesarias para modelar las heurísticas de asignación disponibles en **BHAVRP** teniendo en cuenta su clasificación. Incorporar el tratamiento de excepciones en los casos que lo requiera.
    - Incorporar en ambas bibliotecas **BHAVRP** el cálculo de distancia real mediante el uso del API **OSRM**.
3. Validar el funcionamiento de la nueva versión de **BHAVRP** mediante la ejecución de pruebas.
    - Diseñar los casos de prueba necesarios para validar el correcto funcionamiento de la nueva versión de la biblioteca.
    - Ejecutar en una herramienta los casos de prueba diseñados para validar el correcto funcionamiento de la nueva versión 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

sistema 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. **El Capítulo 1: Estado del arte** presenta un análisis detallado de **BHAVRP**, enfocado en su propósito, componentes principales, estrategia de solución mediante algoritmos de asignación y agrupamiento, y los principios de diseño que sustentan su desarrollo. También se realiza un estudio comparativo con otras bibliotecas relevantes, destacando sus fortalezas y limitaciones. Finalmente, se describen las principales deficiencias de **BHAVRP**, identificando áreas de mejora para optimizar su desempeño en futuros desarrollos.

El **Capítulo 2: Concepción y materialización** 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 Python. Este capítulo abarca desde la identificación de requisitos hasta la implementación de las clases y algoritmos necesarios, analizando las decisiones arquitectónicas tomadas y cómo estas contribuyen a superar las limitaciones de la versión anterior.

El **Capítulo 3: Validación** se centra en la evaluación exhaustiva de la solución a través de un estudio experimental. En este capítulo, se presentan 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 presenta un análisis detallado de la Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos (**BHAVRP**), una herramienta diseñada específicamente para realizar la fase de asignación en el Problema de Planificación de Rutas de Vehículos con Múltiples Depósitos (**MDVRP**). El capítulo inicia describiendo las generalidades de la biblioteca, enfatizando en su propósito, sus componentes principales y la estrategia de solución empleada basada en algoritmos de asignación y de agrupamiento. Además, se presenta la arquitectura de **BHAVRP**, así como los principios de diseño y patrones de software que sustentan su desarrollo.

También se incluyen otras bibliotecas relevantes que, aunque no están específicamente diseñadas para resolver el **MDVRP**, ofrecen capacidades para abordar la fase de asignación mediante heurísticas y algoritmos de agrupamiento. Por último, se discuten las principales deficiencias que presenta la versión actual de **BHAVRP**, identificando áreas clave de mejora y oportunidades para optimizar su desempeño en futuros desarrollos.

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.

La Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos (**BHAVRP**) le da solución a la fase de asignación de la variante **MDVRP** en las siguientes herramientas: **CaSVRP** [31], **BHCVRP** [32] y **TransO** [30]. 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.

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** [33]. **BHAVRP** consta con un diseño robusto que permite la incorporación de nuevos algoritmos de asignación, así como su extensión a otras variantes de múltiples depósitos, como Múltiples Depósitos con Flota Heterogénea [34] y Múltiples Depósitos con Ventanas de Tiempo [35] y una combinación de las anteriores.

En **BHAVRP** están disponibles 15 heurísticas de asignación, de las cuales 6 fueron seleccionadas del estado del arte, mientras que las restantes son adaptaciones de algoritmos de agrupamiento utilizados en minería de datos. Se puede apreciar en la Figura 1 la totalidad de las heurísticas disponibles en **BHAVRP** organizadas en categorías: algoritmos de agrupamiento (particionales y jerárquicos), métodos clásicos de asignación basados en urgencias, ciclos y clústeres, además de enfoques aleatorios y estrategias fundamentadas en distancias.

Figura 1: 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, permite la carga de ficheros de entrada en formatos estándar como *.data* o *.txt*, facilitando la incorporación de datos al sistema. 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 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 representan los vértices (V), siendo el vértice 0 (v_0) el depósito o almacén donde se encuentran los vehículos de transporte y la mercancía [36].

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 Vendedor Viajero (*Traveling Salesman Problem*, **TSP**)**:** busca encontrar la ruta más corta que visita un conjunto de ciudades una sola vez y regresa al punto de origen [37].

- Problema 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 [38].
- 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 [39].
- Problema de Rutas de Vehículos con Restricciones de Tiempo (*Time Windows Vehicle Routing Problem*, **TVRP**): requiere que los vehículos lleguen a los clientes dentro de un intervalo de tiempo específico [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**) [38].
- Problema de Rutas de Vehículos con Recogidas y/o Entregas (*Vehicle Routing Problem with Pickup and Delivery*, **VRPPD**) [41].
- Problema de Rutas de Vehículos con Múltiples Depósitos y Recogidas/Entregas Combinadas (*Multiple Depot Vehicle Routing Problem with Demand*, **VRPMPD**) [42].
- Problema de Recolección y Entrega con Ventanas de Tiempo (*Pickup and Delivery* Problem with Time Windows, PDPTW) [43].

En la Figura 2 se muestra una imagen sobre las variantes que se derivan del problema de planificación de rutas de vehículos original.

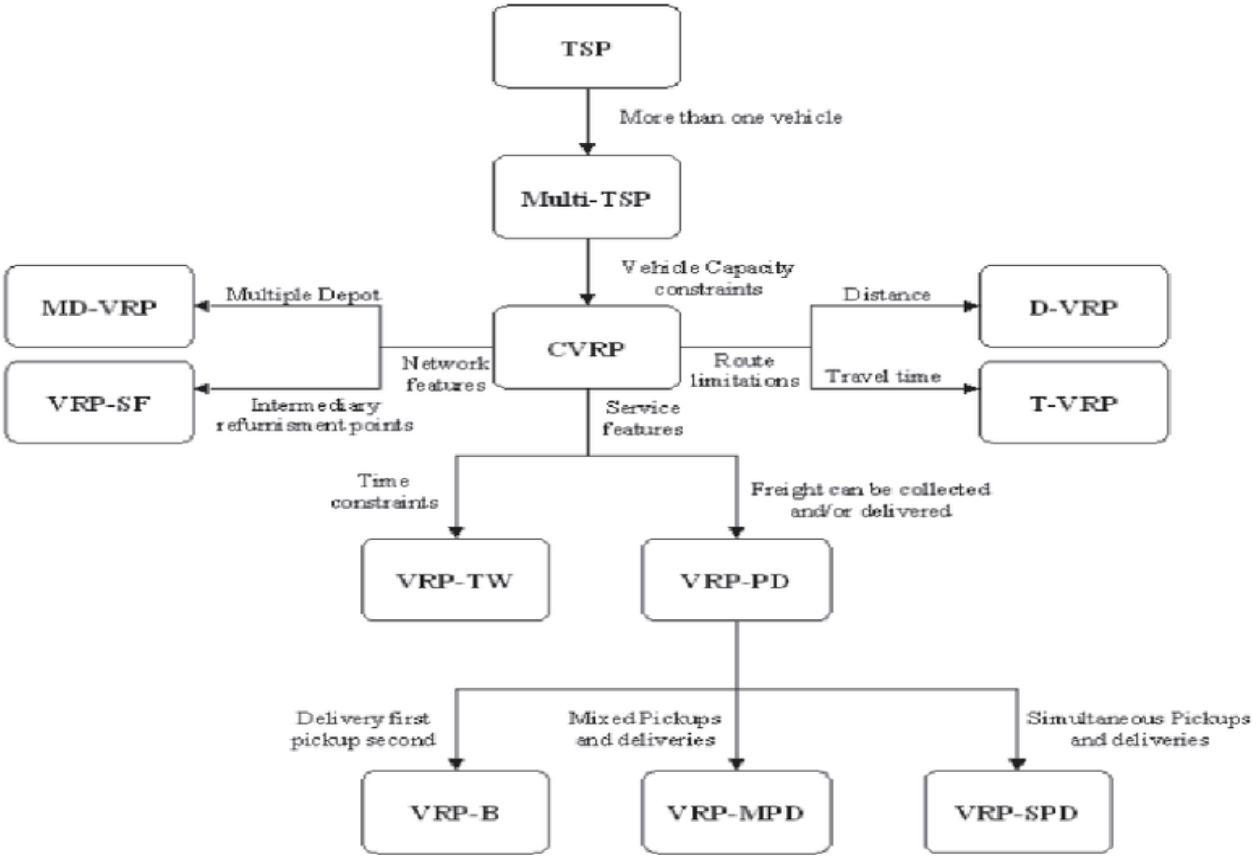

Figura 2: Variantes del VRP.

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**) [33]. Esta variante representa un problema de optimización combinatoria perteneciente a la categoría denominada **NP**-duro, debido a que es un problema que no puede resolverse en un tiempo polinomial y crece exponencialmente a medida que se introducen nuevas variables. 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. Es un problema de optimización que se puede expresar como minimizar los costos ya sea en distancia o en tiempo, satisfaciendo en todo momento las restricciones del problema. Estos problemas están presentes en diversos sectores de nuestra sociedad, como el transporte de mercancías y 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. Por estas condiciones, la solución del problema generalmente se organiza 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, hay tres categorías a tomar en cuenta: métodos exactos, heurísticas y metaheurísticas. Los métodos exactos son eficientes en problemas de hasta 50 depósitos, debido a restricciones de tiempo computacional. Los métodos exactos se pueden clasificar en tres grupos: búsqueda directa de árbol, programación dinámica y programación lineal y entera [44].

Las heurísticas son procedimientos que proporcionan 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. 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 [5].

El método de dos fases se encuentran los métodos de asignación elemental, el algoritmo de ramificación y acotamiento truncados, el algoritmo de los pétalos, el método de rutear primero y asignar después y los procedimientos de búsqueda total. Entre los Métodos de Agrupamiento Elemental (*Elementary Clustering Methods*) [16] se encuentra el algoritmo

de barrido (*Sweep Algorithm*) [1], el algoritmo basado en asignación generalizada (*Generalizad-Assignment-Based Algorithm*) [45] y la heurística basada en localización (*Location Based Heurisitc*) [46].

Las metaheurísticas fueron desarrolladas hacia finales de la década de los 90. Se caracterizan por realizar procedimientos de búsqueda para encontrar soluciones de aceptable calidad, mediante la aplicación de operadores independientes del dominio que modifican soluciones intermedias guiadas por la idoneidad de su función objetivo. Dentro de estas, las metaheurísticas se pueden clasificar en categorías generales como las basadas en trayectorias, las basadas en poblaciones, las híbridas y las basadas en modelos, dependiendo de sus principios de operación y objetivos, entre otros enfoques importantes [47].

1.2.1.2. Estrategias de solución empleadas para la resolución del problema

Para resolver el problema que presenta la variante **MDVRP**, los investigadores han preferido utilizar métodos heurísticos antes que algoritmos exactos. Dentro de los métodos heurísticos se emplean mayormente las metaheurísticas que trabajan en dos fases de acuerdo a la estrategia “Agrupar primero, planificar después” (*Cluster First – Route Second*, **CFRS**) [48]. Esta estrategia consiste en realizar la asignación de clientes a los depósitos y luego la planificación de rutas de los vehículos. De las metaheurísticas destacan también las basadas en Búsqueda Tabú [49] y que se mantuvieron como el mejor enfoque para resolver el **MDVRP** durante mucho tiempo, debido a las características de su implementación.

Además de las heurísticas de asignación, 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. Este campo integra diversas técnicas derivadas de la inteligencia artificial y la estadística, que consisten en algoritmos diseñados para procesar conjuntos de datos de manera eficiente y obtener resultados significativos [16] [50].

Entre las técnicas más representativas, está el Agrupamiento o *Clustering* [51], que es una de las tareas más frecuentes de la Minería de Datos [50] 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 derivan otros. Estos algoritmos son *K-means* [14] y *K-medoids* [52], que han servido como base para desarrollar otros algoritmos avanzados. Por ejemplo, *Farthest First* [53] 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* [54] se derivan de *K-medoids*, adaptándose a diferentes tamaños y características de los conjuntos de datos.

Además, existen otras técnicas significativas como *DBSCAN* [55], que identifica grupos de densidad en los datos, y algoritmos jerárquicos, que forman una estructura en niveles para describir las relaciones entre los datos. Estas herramientas cubren un amplio espectro de aplicaciones dependiendo de la naturaleza y complejidad del problema.

1.2.2. Diseño arquitectónico de BHAVRP

La arquitectura de **BHAVRP** ha sido diseñada con un enfoque en la reutilización y modularidad, lo que garantiza su adaptabilidad y escalabilidad. 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 3 la organización de estas capas y su rol en el diseño general de **BHAVRP**.

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.

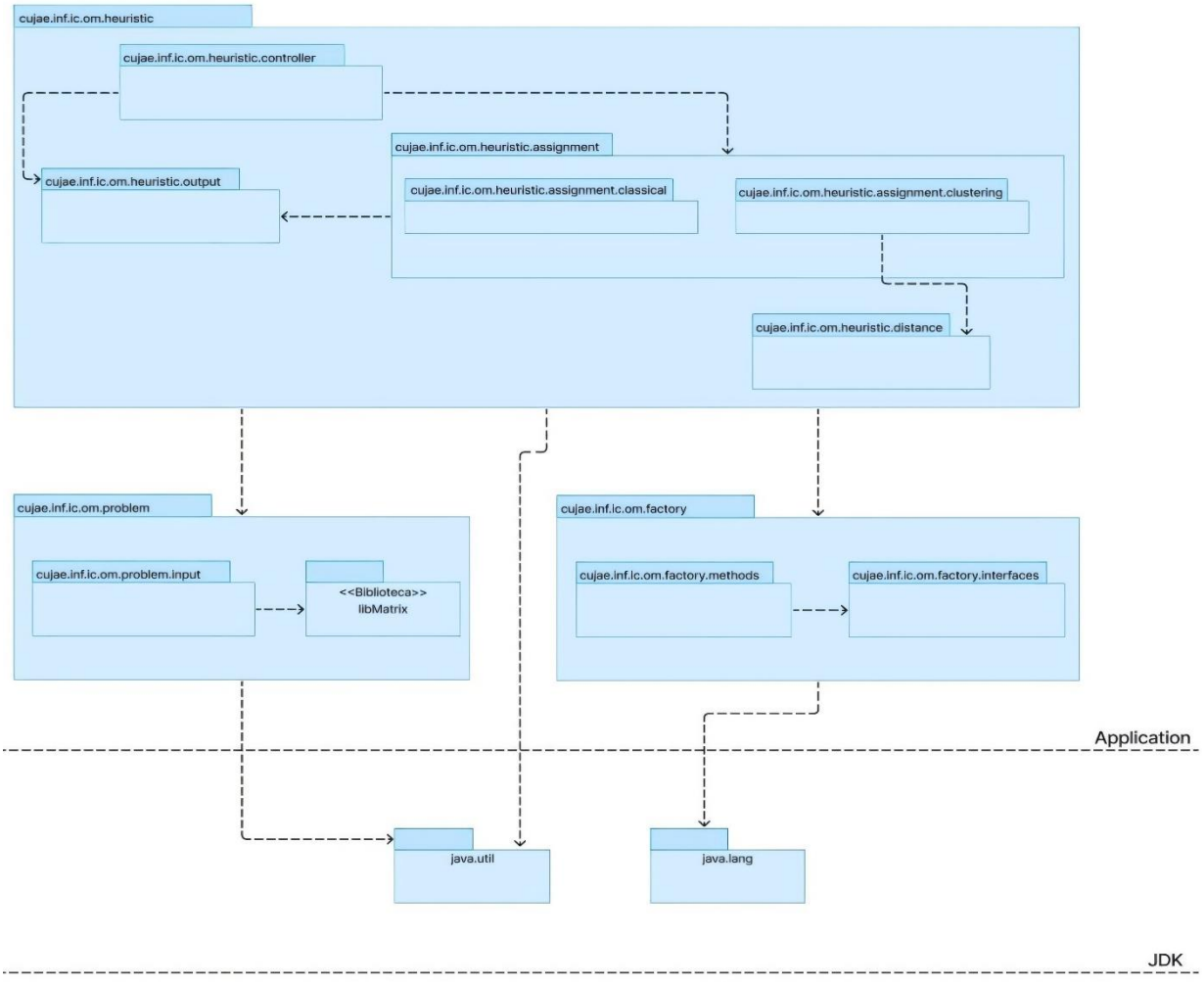

Figura 3: Arquitectura de BHAVRP *[23]*.

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 4 muestra el diagrama de clases del paquete *Problem*.

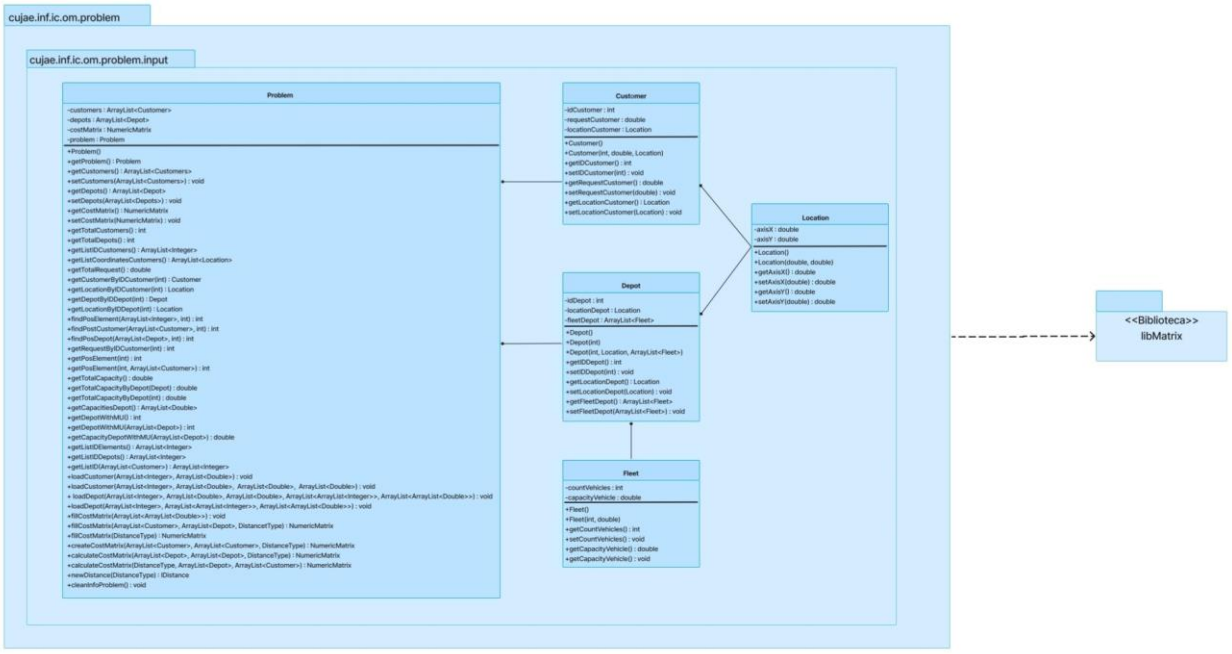

Figura 4: Diagrama de clases del paquete *Problem*.

En el paquete cujae.inf.citi.om.heuristic se agrupan los paquetes: cujae.inf.citi.om.heuristic.assignment responsable de modelar las clases que implementan las heurísticas de asignación, la Figura 5 muestra el diagrama de clases del paquete *Assignment*, 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, la Figura 6 muestra el diagrama de clases del paquete *Output*, y cujae.inf.citi.om.heuristic.controller donde se ubica la clase responsable de todo el

proceso de asignación de clientes a depósitos en la biblioteca, la Figura 7 muestra el diagrama de clases del paquete *Controller*.

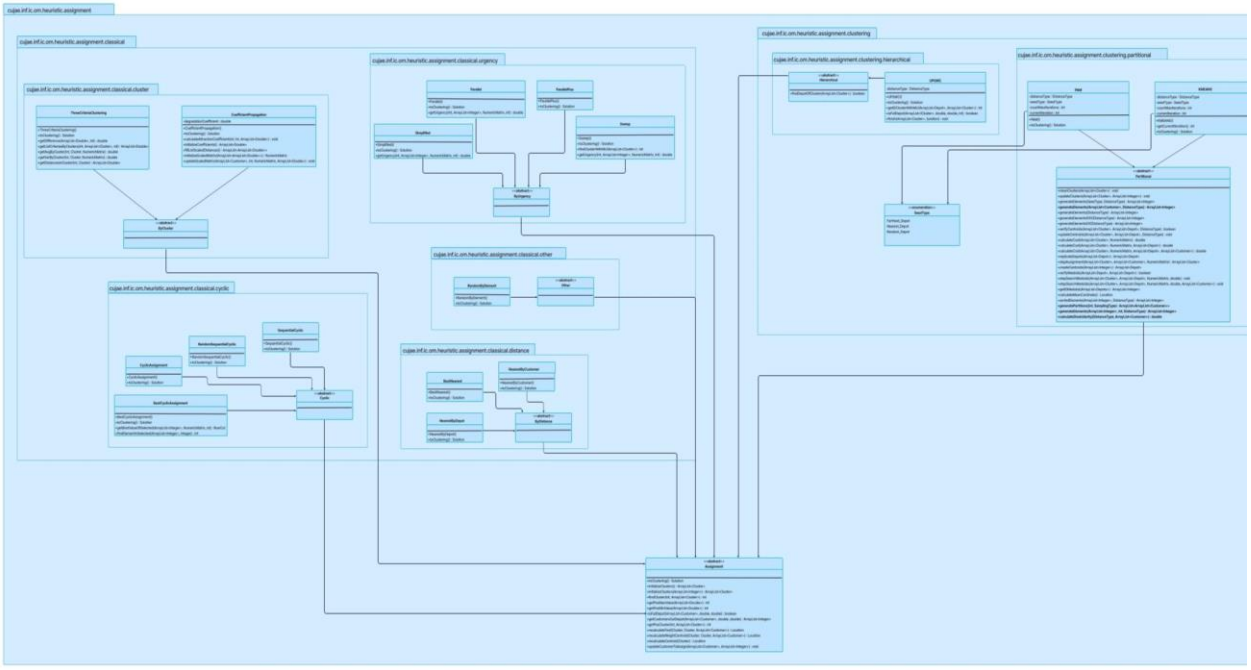

Figura 5: Diagrama de clases del paquete *Assignment*.

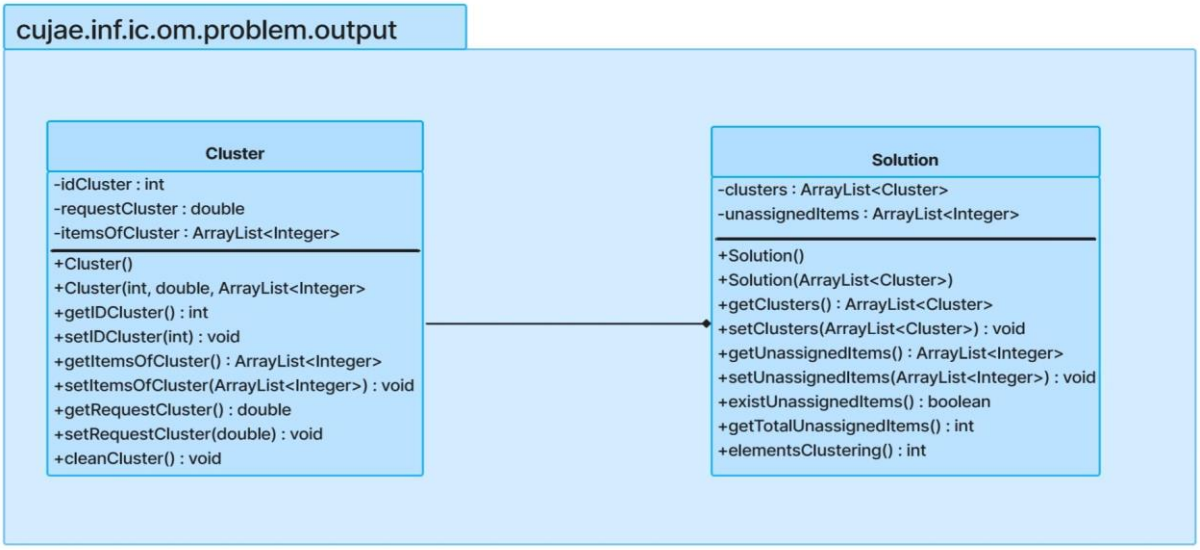

Figura 6: Diagrama de clases del paquete *Output*.

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

Por último, el paquete cujae.inf.citi.om.factory es el responsable de garantizar la carga dinámica de las heurísticas de asignación. Este paquete implementa el patrón *Factory Method*, los cuales permiten encapsular la creación de objetos relacionados con las heurísticas y las distancias. Esto facilita la extensión del sistema para añadir nuevas heurísticas sin modificar el código existente, promoviendo el cumplimiento del *Open/Closed Principle* y mejorando la flexibilidad en la configuración y ejecución del sistema. La Figura 8 muestra el diagrama de clases del paquete *Factory*.

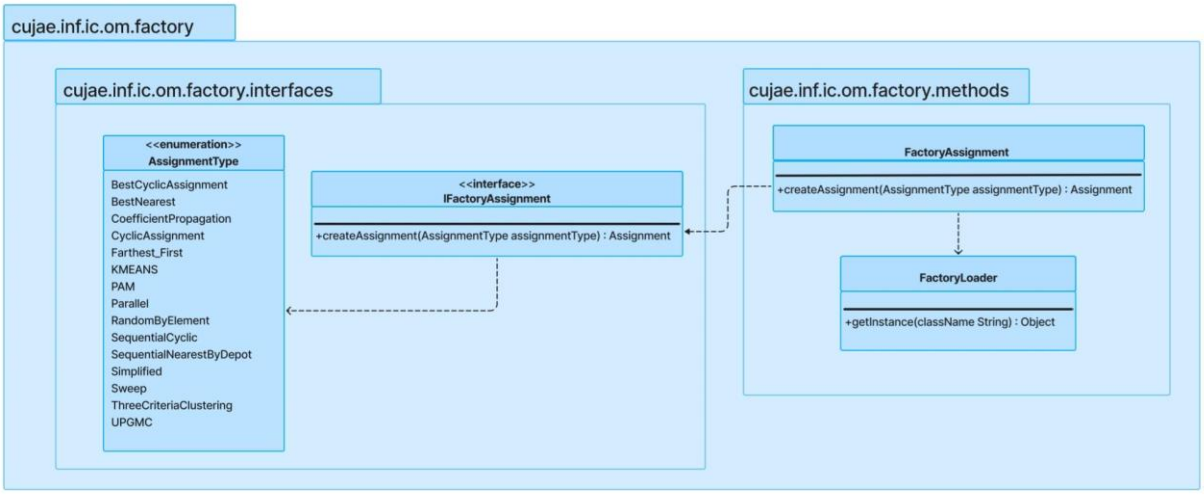

Figura 8: Diagrama de clases del paquete *Factory*.

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 con múltiples depósitos de **VRP**, ha adoptado diversos principios y patrones de diseño que no solo mejoran su estructura, sino que también optimizan su mantenimiento y extensión. En esta sección, se presentan los principios de diseño y los patrones utilizados en **BHAVRP**.

Los principios de diseño son un conjunto de directrices que ayudan a crear 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 [56].

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. En este análisis, se abordan únicamente los principios **SOLID** que están presentes en **BHAVRP**.

La aplicación de los principios **SOLID** no es binaria (cumple o no cumple), sino que es un espectro, un código puede adherirse en mayor o menor medida a estos principios, y la evaluación debe considerar el contexto y las necesidades específicas del proyecto. 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?
Single Responsibility Principle	Cada clase tiene una única responsabilidad bien definida.	Sí
	Los métodos de una clase están estrechamente relacionados con su propósito principal.	Sí
Open/Closed Principle	Es posible extender el comportamiento de las clases sin modificar su código existente.	Sí
	Se usan interfaces y clases abstractas para permitir la extensión.	Sí
Liskov Substitution Principle	Las subclases pueden ser utilizadas en lugar de sus clases base sin causar errores.	Sí
	Las subclases no lanzan excepciones que no están en la firma de los métodos de la clase base.	No
Interface Segregation Principle	Las interfaces son específicas y cohesivas, en lugar de ser grandes y generales.	Sí
	Las clases que implementan interfaces no tienen métodos vacíos o sin implementar.	Sí
Dependency Inversion Principle	Se usan abstracciones (interfaces o clases abstractas) en lugar de implementaciones concretas.	Sí
	Las dependencias se inyectan en lugar de ser creadas dentro de las clases.	No

Partiendo de la tabla anterior, 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** (*Single Responsibility Principle*), OCP (*Open/Closed Principle*), **ISP** (*Interface Segregation Principle*) y **DIP** (*Dependency Inversion Principle*) se implementan casi en su totalidad, garantizando cohesión, extensibilidad y modularidad en el sistema.

Sin embargo, se identifican áreas de mejora en los principios **LSP** (*Liskov Substitution Principle)* y **DIP**. En el caso del **LSP**, si bien las subclases pueden reemplazar a las clases base sin problemas en la mayoría de los casos, hay situaciones donde las subclases lanzan excepciones no especificadas en la firma de los métodos de la clase base. Esto podría causar problemas de robustez y consistencia en la biblioteca. Respecto al **DIP**, aunque se hace uso de abstracciones, aún se identifican dependencias creadas dentro de las clases, lo cual limita la flexibilidad y dificulta pruebas unitarias más independientes.

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. Resolver estas áreas pendientes fortalecerá 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 [57] [58] [59].

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 **BHAVRP**, 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 **BHAVRP**, la clase *Controller* orquesta el proceso de asignación sin que las clases de bajo nivel interactúen directamente con ella, favoreciendo un diseño limpio y desacoplado.
Diseñar hacia las interfaces	Programar contra interfaces o clases abstractas en lugar de implementaciones concretas.	**BHAVRP** usa la interfaz *IFactoryDistance* para definir los métodos de cálculo de distancias, mientras que la clase *FactoryDistance* implementa la lógica concreta, respetando la interfaz sin atarse a una implementación específica.

Once and Only Once Rule	Cada elemento de conocimiento debe tener una sola representación en el sistema.	La clase *Problem* en **BHAVRP** centraliza la modelización de los datos del problema **MDVRP**, asegurando que todos los cambios en los datos se gestionen de manera controlada y unificada, evitando la duplicación de responsabilidades.

En resumen, **BHAVRP** implementa varios principios de diseño clave que mejoran su estructura y mantenibilidad. La Ley de Deméter reduce el acoplamiento entre clases al limitar el conocimiento de una clase sobre otras. El Principio de *Hollywood* asegura que las clases de alto nivel gestionen el flujo de ejecución, sin que las clases de bajo nivel interfieran en la lógica de alto nivel. El principio de Diseñar hacia las interfaces se ejemplifica en la interfaz *IFactoryDistance*, que permite la flexibilidad de implementación sin atarse a detalles concretos. Por último, la *Once and Only Once Rule* se aplica al centralizar la gestión de los datos en la clase *Problem*, evitando duplicaciones y asegurando la consistencia del sistema. Estos principios contribuyen a un código modular, flexible y fácil de mantener.

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 [57].

- 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 la "Banda de los Cuatro", este apartado se centra en algunos patrones específicos. 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 9 se muestra un ejemplo de la implementación del patrón *Singleton* para la clase *Problem*.

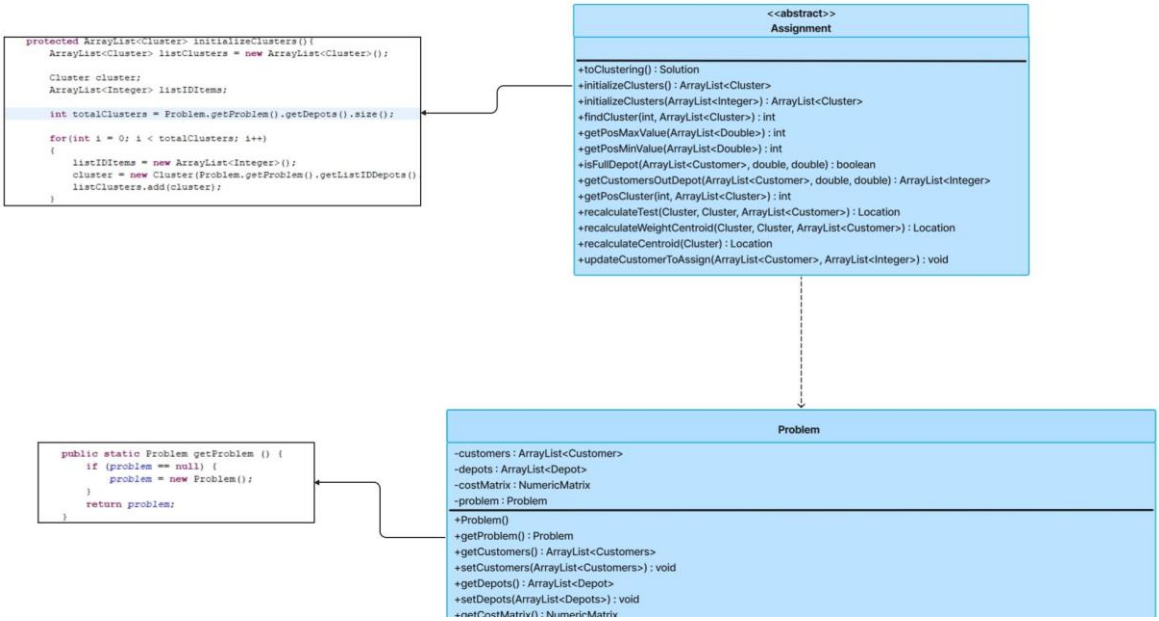

Figura 9: Patrón *Singleton* implementado en BHAVRP.

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.

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. 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 como los mencionados en los resultados de búsqueda, donde la flexibilidad y la extensibilidad son importantes.

Para realizar la asignación de clientes a depósitos, **BHAVRP** cuenta con varias heurísticas de asignación. De ellas, el usuario debe seleccionar una 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 10**¡Error! No se encuentra el origen de la referencia.** se muestra un ejemplo de la implementación del patrón para las heurísticas de asignación.

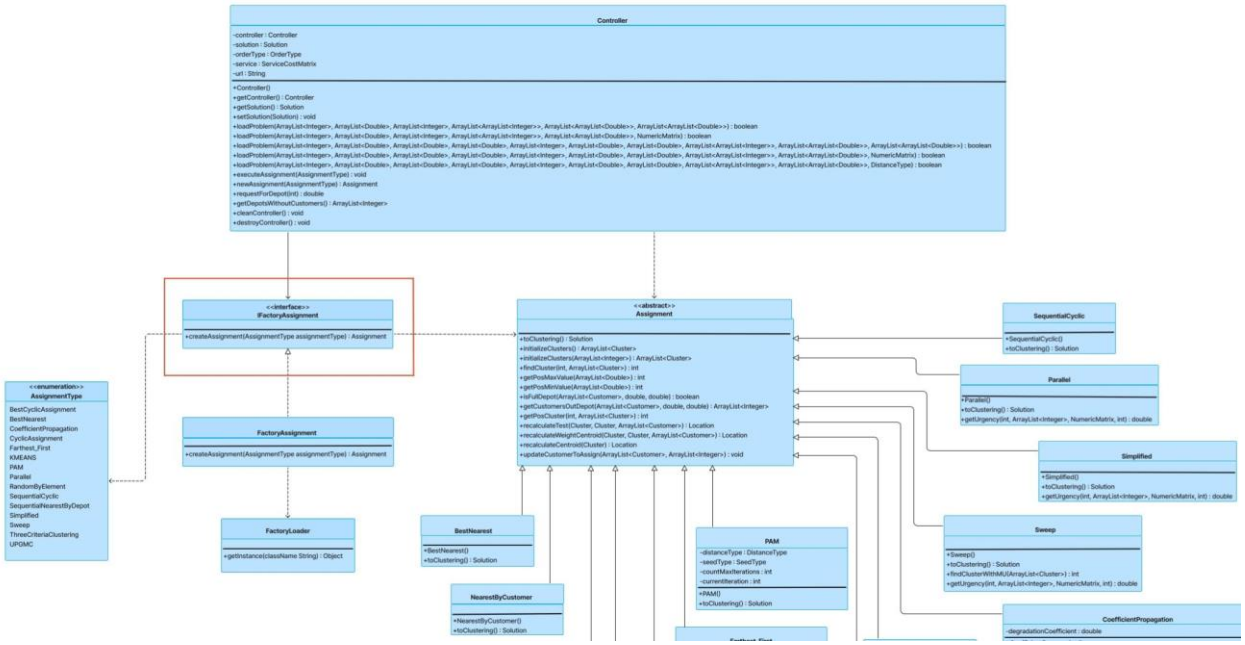

Figura 10: Patrón *Factory Method* implementado en BHAVRP.

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 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 heurísticas, garantizando que las subclases implementen detalles específicos mientras mantienen una estructura coherente y reutilizable en los algoritmos.

Los patrones **GoF**, aunque fundamentales, 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. La transición de los patrones **GoF** a **GRASP** se realiza cuando se busca una mejor asignación de responsabilidades en la implementación concreta del sistema.

La Tabla 3 a continuación resume 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	¿**BHAVRP** lo cumple?
Controlador	Una clase centralizada gestiona los eventos y actúa como intermediaria.	Sí
	El patrón facilita pruebas y promueve modularidad.	Sí
Experto en información	Las responsabilidades recaen en las clases que poseen la información necesaria.	Sí
	Promueve cohesión y encapsulamiento.	Sí
Alta cohesión	Las clases tienen responsabilidades relacionadas y específicas.	Sí
	Las dependencias entre clases están minimizadas.	Sí

Bajo acoplamiento	Se garantiza que los cambios en una clase no impacten directamente en otras.	Sí
Creador	Las clases responsables crean instancias de las clases que necesitan.	Sí
	La creación está alineada con el ciclo de vida lógico de las instancias.	Sí

De la tabla anterior 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. 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. Al aplicar estos principios y patrones, se logra una solución sólida, escalable y lista para integrarse con otros sistemas, cumpliendo con los objetivos de optimización y eficiencia en la asignación de clientes a depósitos.

1.3. Bibliotecas para resolver 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 presenta una selección de bibliotecas destacadas que están especializadas en la resolución de **VRP** y sus variantes, entre las que destaca **MDVRP**, junto con 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 Vendedor Viajero (*Traveling Salesman Problem*, **TSP**) y el 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** [20].

  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 *Variable Neighborhood Search* (**VNS**). Este enfoque ajusta iterativamente las soluciones mediante cambios en la vecindad para mejorar los resultados.
- *Google Operation Research Tools* (**OR-Tools**) es un paquete de software de código abierto 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 [17]z.

  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* del problema de planificación de rutas de vehículos desarrollado en el lenguaje de programación C++. Esta herramienta utiliza *Open Street Map* (**OSM**) y/o *Open Source Routing Machine* (**OSRM**) 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].

  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 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 [60].

  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**.

Además de estas bibliotecas, existen herramientas más generales que abordan el problema del **MDVRP** de forma integral, resolviendo tanto la asignación de clientes a depósitos como la planificación de rutas de los vehículos.

- 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 [30].
- Capa de servicio para soluciones a Problemas de Planificación de Rutas de Vehículos (**CaSVRP**) 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**). 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**) 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 (**LMOVRP**) [61] [31].
- 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 [32].
- Biblioteca de Clases para la Integración de Algoritmos Metaheurísticos (**BiCIAM**) es una herramienta desarrollada en Java que implementa un modelo unificado para metaheurísticas. Inicialmente orientada a algoritmos basados en un punto, como Búsqueda Aleatoria [62] y Recocido Simulado [63], ha evolucionado para

incluir algoritmos poblacionales como Algoritmos Genéticos [64] y Estrategias Evolutivas [65]. Además, **BiCIAM** integra un Multigenerador, permitiendo la colaboración entre generadores metaheurísticos, y soporta variantes avanzadas como NSGA-II para problemas multiobjetivo [66] [67].

A continuación, se presenta una comparación de las bibliotecas previamente mencionadas considerando un conjunto de características relevantes. En la Tabla 4 se detallan los criterios establecidos para llevar a cabo dicho análisis.

Tabla 4: Comparación de otras bibliotecas que resuelven MDVRP.

Bibliotecas	Características							
	Resuelve la fase de asignación para **MDVRP**	Implementan h**eurísticas clásicas**	Implementan **algoritmos de agrupamiento**	Cálculo de distancias	Evaluación de resultados	Lenguajes soportados	Interoperabilidad	Código abierto
BHAVRP	Sí	Sí	Sí	Matrices personalizadas	No	Java	Baja	No
JSprit	Sí	No	No	Basado en matrices	No	Java	Limitada	Sí
OR-Tools	Sí	Sí	No	Matrices o funciones (opcional)	No	C++, Python, Java, C#	Alta	Sí
VROOM	Sí	Sí	No	Matrices proporcionadas	No	C++	Alta	No
Scikit-learn	Sí	No	Sí	No aplica	Sí	Python	Alta	Sí
TransO	Sí	Sí	No	Basado en matrices	No	Java	Baja	No
CaSVRP	Sí	Sí	No	Basado en matrices	No	Java	Baja	No
BHCVRP	Sí	Sí	No	Basado en matrices	No	Java	Baja	No
BiCIAM	Sí	Sí	No	No aplica	No	Java	Baja	No

A partir del análisis comparativo de las bibliotecas presentadas en la tabla anterior, 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.
- 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. Destacan **BHAVRP**, **CaSVRP** y **BiCIAM** por su enfoque en estas técnicas.
- 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.
- La mayoría de las bibliotecas, como **JSprit**, **CaSVRP**, **TransO** y **BHCVRP**, utilizan matrices predefinidas para el cálculo de distancias. **BHAVRP** se distingue al permitir matrices personalizadas, mientras que **OR-Tools** ofrece flexibilidad con funciones definidas por el usuario.
- Solo **Scikit-learn** incorpora funcionalidades explícitas para evaluar resultados, lo que podría ser un área de mejora para las demás bibliotecas, especialmente para proyectos que requieran análisis detallado de calidad de 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** y las desarrolladas en Java, 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.

En resumen, aunque ninguna de estas bibliotecas está diseñada exclusivamente para resolver la fase de asignación en el **MDVRP**, 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 para resolver el **MDVRP** de manera eficiente.

Como observación adicional destaca que **BHAVRP**, aunque tiene características avanzadas como algoritmos de agrupamiento y matrices personalizadas, 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. 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.

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, lo cual también ayudaría a reducir el acoplamiento entre paquetes. Este aspecto es clave, ya que un alto

acoplamiento puede limitar la flexibilidad y escalabilidad de la biblioteca en entornos más complejos.

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. Una solución podría incluir la adopción de estándares de intercambio de datos ampliamente utilizados, como *XML*, que facilitarían la integración con aplicaciones en otros lenguajes.

Además, la biblioteca no cuenta con un manejo de excepciones consistente, lo que puede dificultar la detección y corrección de errores durante la ejecución. Este aspecto es especialmente crítico en aplicaciones complejas como **BHAVRP**, donde los algoritmos trabajan con grandes volúmenes de datos y requieren una alta precisión. Asimismo, **BHAVRP** carece de un módulo dedicado que encapsule métricas de evaluación para medir la calidad de las asignaciones generadas por los algoritmos, lo cual limita las posibilidades de analizar y mejorar la eficiencia de las soluciones obtenidas.

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.

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 que limitan la exactitud en la asignación de clientes a depósitos. Entre las herramientas destacadas en este ámbito se encuentran *Google Distance Matrix API* y **GraphHopper**.

- *Google Distance Matrix API* calcula distancias y tiempos de viaje reales entre puntos específicos utilizando datos de *Google Maps*. 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 [28].
- **GraphHopper** es una biblioteca de código abierto para enrutamiento sobre redes viales que utiliza datos de *OpenStreetMap*. 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 el enrutamiento y la capacidad de operar en entornos sin acceso a Internet [29].

Si bien **BHAVRP** está completamente implementado en Java y este es un lenguaje de programación robusto y utilizado en muchos entornos, 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 para asegurar que **BHAVRP** continúe siendo eficiente y relevante en un entorno en constante evolución.

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. *libMatrix* es una biblioteca Java desarrollada para realizar cálculos matriciales, dado que en el momento de su creación no existían bibliotecas disponibles que cubrieran las necesidades del proyecto de manera eficiente.

En Python, existen bibliotecas de código abierto que podrían reemplazar las funciones de estos componentes personalizados en **BHAVRP**, simplificando así el mantenimiento y la eficiencia del proyecto. La biblioteca *numpy* [68] puede sustituir a *libMatrix* al ofrecer cálculos matriciales optimizados y ampliamente utilizados. Para los cálculos de distancia

basados en coordenadas, *OSRM-py* [69] y *osmnx* [70] proporcionan herramientas robustas para calcular rutas y distancias reales, similar a lo que realiza *OSRMLib*. Finalmente, *scipy.spatial.distance* [71] ofrece una variedad de métricas de distancia, incluidas las métricas *Euclidian*, *Haversine, Manhattan* y *Chebyshev*, permitiendo un manejo flexible y completo de las medidas de distancia en Python.

1.5. 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**.
- La arquitectura de **BHAVRP** está diseñada para que sea extensible, permitiendo la incorporación de nuevas heurísticas y técnicas de resolución, lo cual amplía su potencial de aplicación y su adaptabilidad a variantes complejas del **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. Estos aspectos contribuyen a que sea una herramienta flexible y modular.
- 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 lenguajes más modernos, una mayor interoperabilidad y el cálculo de distancias reales, proporciona un camino claro para futuros desarrollos.

Capítulo 2: Concepción y materialización

2.1. Introducción

Capítulo 3: Validación…

3.1. Introducción

Conclusiones

Recomendaciones

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 Algortihm: 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.

[18] «VROOM GitHub Repository,» [En línea]. Available: https://github.com/VROOM-Project/vroom.

[19] G. O. Suite, «Gekko Documentation,» [En línea]. Available: https://gekko.readthedocs.io/en/latest/.

[20] «JSprit: A Java-based library for solving vehicle routing problems,» [En línea]. Available: https://jsprit.readthedocs.io/en/latest/.

[21] T. B. P. O. S. OptaPlanner. [En línea]. Available: https://www.optaplanner.org/.

[22] 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.

[23] 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.

[24] S. C. Gupta, «Euclidean Distance in Optimization Problems,» *Optimization Techniques,* vol. 3, nº 2, pp. 100-110, 2017.

[25] 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.

[26] M. K. a. J. P. J. Han, Data Mining: Concepts and Techniques, 3rd ed., Elsevier, 2012.

[27] B. K. Hachem, «Applications of Chebyshev Distance in Route Optimization Problems,» *Computational Geometry,* vol. 25, pp. 85-92, 2016.

[28] G. Developers, «Distance Matrix API,» Google for Developers., [En línea]. Available: https://developers.google.com/maps/documentation/distance-matrix. [Último acceso: 26 Noviembre 2024].

[29] G. GmbH, «GraphHopper Directions API,» GraphHopper, [En línea]. Available: https://www.graphhopper.com/products/. [Último acceso: 26 Noviembre 2024].

[30] 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.

[31] E. H. Fernández, «Nueva versión de la Capa de servicio para soluciones a Problemas de Planificación de rutas de Vehículos,» 2016.

[32] L. D. Subí, «Nueva versión de la biblioteca de heurísticas de construcción para Problemas de Planificación de Rutas de Vehículos,» 2016.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] P. T. a. D. Vigo, The Vehicle Routing Problem, SIAM, 2002.

[39] 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.

[40] 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.

[41] D. Psaraftis, «The Pickup and Delivery Problem with Time Windows,» *Transportation Science,* vol. 22, nº 1, pp. 1-12, 1988.

[42] 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.

[43] 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.

[44] S. R. a. E. W. B. L. Golden, The Vehicle Routing Problem: Latest Advances and New Challenges., Springer, 2008.

[45] A. H. y. G. L. M. Gendreau, «A Generalized Assignment Heuristic for the Vehicle Routing Problem with Time Windows,» *European Journal of Operational Research,* vol. 116, nº 3, pp. 500-510, 1999.

[46] F. A. S. N. y. A. M. R. F. L. S. R. P. Silva, «A Location-Based Heuristic for the Location-Routing Problem,» *Computers & Operations Research,* vol. 34, nº 10, pp. 2963-2985, 2007.

[47] E.-G. Talbi, Metaheuristics: From Design to Implementation., Hoboken, NJ, USA: Wiley, 2009.

[48] M. W. T. T. y. D. Vigo, Vehicle Routing: Problems, Methods, and Applications, SIAM, 2014.

[49] G. G. y. M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.

[50] W. W. Cohen, Data Mining Techniques, Wiley, 1997.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship., Upper Saddle River, NJ, USA: Prentice Hall, 2008.

[57] E. H. R. J. R. &. V. J. Gamma, Design Patterns: Elements of Reusable Object-Oriented Software., Addison-Wesley, 1994.

[58] C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development., Pearson Education, 2002.

[59] R. C. Martin, Agile Software Development: Principles, Patterns, and Practices., Pearson Education, 2003.

[60] V. V. P. Pedregosa, «Scikit-learn: Machine Learning in Python,» *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[61] L. d. C. M. Lores, «Operadores de mutación basados en heurísticas de construcción para Problemas de Planificación de Rutas de Vehículos,» 2017.

[62] J. Matyas, «Random Optimization,» *Automation and Remote Control,* vol. 26, nº 2, pp. 246-253, 1965.

[63] C. D. G. a. M. P. V. S. Kirkpatrick, «Optimization by Simulated Annealing,» *Science,* vol. 220, nº 4598, pp. 671-680, 1983.

[64] J. H. Holland, «Adaptation in Natural and Artificial Systems,» University of Michigan Press, 1975.

[65] H.-P. Schwefel, Numerical Optimization of Computer Models, Wiley, 1981.

[66] R. D. Hernández, «Nueva versión de la biblioteca de clases BiCIAM para solucionar problemas multiobjetivo,» 2014.

[67] A. P. S. A. a. T. M. K. Deb, «A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II,» *IEEE Transactions on Evolutionary Computation,* vol. 6, nº 2, pp. 182-197, 2002.

[68] NumPy, «NumPy Documentation,» NumPy, [En línea]. Available: https://numpy.org/doc/stable/.

[69] OSRM-py, «OSRM-py Documentation, GitHub,» [En línea]. Available: https://github.com/Project-OSRM/osrm-py.

[70] G. Boeing, «Using OSMnx to Investigate OpenStreetMap Data,» Data Science Prophet.

[71] S. S. Distance, «scipy.spatial.distance_matrix, SciPy v1.14.1 Manual».

[72] H. R. Y. S. K. a. F. Y. Serap Ercan Comert, «A cluster first-route second approach for a capacitated vehicle routing problem: a case study,» InderScience Online, 2018.

[73] O. V. L. Tansini, «Adapted Clustering Algorithms for the Assignment Problem in MDVRPTW,» 2004.

[74] R. M. Cuquero, «Algoritmos Heurísticos en Optimización Combinatoria,» Universidad de Valencia, Facultad de Ciencias Matemáticas, Valencia.

[75] A. V. Breeman, «Improvement heuristic for the Vehicle Routing Problem based on simulated annealing,» *European Journal of Operational Research,* pp. 480-490, 1995.

[76] A. M. a. P. T. N. Christofides, «The vehicle routing problem,» de *Combinatorial Optimization*, Wiley, 1979, pp. 315-338.

[77] E. H. R. J. R. &. V. J. Gamma, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.