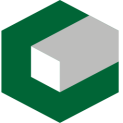
Universidad Tecnológica de La Habana

“José Antonio Echeverría”



Facultad de Ingeniería Informática

**Nueva versión de la Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos**

*Informe de la Práctica Profesional 2*

**Autor:** Ananda de la Caridad Morales Morales

**Tutores:**

Dr. C. Isis Torres Pérez

Dr. C. Alejandro Rosete Suárez

**La Habana, Julio 2024**

**Resumen**

Dentro de los tipos de problemas de optimización se encuentran los problemas de optimización combinatoria, que incluye a los Problemas de Planificación de Rutas de Vehículos. Estos problemas cuentan con una flota de vehículos, buscan satisfacer las demandas de clientes dispersos geográficamente y tienen como objetivo minimizar costos siempre que se cumplan con las restricciones existentes. A diario se producen nuevas situaciones prácticas en el mundo, lo que conlleva al surgimiento de nuevas variantes. Algunos de los métodos de solución para los problemas de planificación de rutas de vehículos son las heurísticas de construcción, que persiguen encontrar de manera rápida y sencilla una primera solución cercana al óptimo. Sin embargo, surge el inconveniente de que la gran mayoría de heurísticas sólo puedan ser empleadas en la variante básica. Además, en la actualidad existen pocas bibliotecas que implementen métodos heurísticos para solucionar estos problemas.

En este contexto surge la Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos, conocida como BHCVRP, con el propósito de dar solución a cuatro tipos de problemas de planificación de rutas de vehículos mediante el empleo de siete heurísticas de construcción. En este trabajo se presenta una nueva versión de dicha biblioteca implementada en los lenguajes de programación Java y Python, que incorpora dos nuevas heurísticas de construcción: Inserción de Kilby y el Algoritmo de Ahorro basado en *Matching*. Incluye tratamiento de excepciones propias y el empleo del patrón de diseño *Template*. Por último, se comparan los resultados alcanzados con la versión anterior de BHCVRP para verificar el correcto funcionamiento de las nuevas versiones; así como con la herramienta OR-Tools para comparar la calidad de las soluciones obtenidas.

**Palabras claves:** BHCVRP, diseño de software, heurísticas de construcción, patrones de diseño, Python, VRP.

**Abstract**

Among the types of optimization problems is the so-called combinatorial optimization problem, which includes Vehicle Routing Problems. These problems involve a fleet of vehicles, seek to satisfy the demands of geographically dispersed customers and aim to minimize costs as long as existing constraints are met. New practical situations occur daily in the world, which leads to the increase of these problems and the emergence of new variants. Some of the solution methods for vehicle route planning problems are construction heuristics, which aim to quickly and easily find a first solution close to the optimum. However, there is the drawback that the vast majority of heuristics can only be used in the basic variant. In addition, there are currently few libraries that implement heuristic methods to solve these problems.

In this context, BHCVRP arises with the purpose of providing solutions to four types of vehicle route planning problems by employing seven construction heuristics. This paper presents a new version of this library implemented in the Python programming language, which incorporates two new construction heuristics: Kilby's Insertion and the Matching-based Saving Algorithm, as well as its own exception handling and the Template design pattern. Finally, the results achieved are compared with the previous version of BHCVRP to verify that they correspond to each test instance and with the OR-Tools tool to compare the quality of the solutions obtained.

**Keywords:** BHCVRP, software design, construction heuristics, design patterns, Python, VRP.

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

|  |  |  |
| --- | --- | --- |
| **Tareas** | **Fecha de entrega** | **Rol(es) que desarrolla(n) con la tarea** |
| Reunión de inicio de la práctica. | 29/4/2024 | JP |
| Asimilación de Python como lenguaje de programación. | 4/5/2024 | AN |
| Identificar las deficiencias que presenta el código de BHCVRP. | 8/5/2024 | AS, AR |
| Investigar si existen bibliotecas en Java que tengan implementadas algunas de las heurísticas existentes en BHCVRP y que estén disponibles. | 10/5/2024 | AS, AR |
| Estudiar patrones y principios de diseño para saber cuáles nuevos se pueden incorporar en el componente. | 12/5/2024 | AR |
| Diseñar los casos de prueba necesarios para validar el funcionamiento de las heurísticas de construcción existentes en BHCVRP para el Problema de Planificación de Rutas de Vehículos con Capacidades (Capacitated VRP, CVRP). | 15/5/2024 | PB |
| Entrega de la primera versión del informe de PP (incluye carátula, resumen, Introducción, estructura del documento, primera versión de la bibliografía, conclusiones parciales y finales) | 15/5/2024 | EE |
| Diseñar los casos de prueba necesarios para validar el funcionamiento de las heurísticas de construcción existentes en BHCVRP para el Problema de Planificación de Rutas de Vehículos con Múltiples Depósitos (Multi Depot VRP, MDVRP). | 18/5/2024 | PB |
| Diseñar los casos de prueba necesarios para validar el funcionamiento de las heurísticas de construcción existentes en BHCVRP para el Problema de Planificación de Rutas de Camiones y Remolques (Truck and Trailer Routing Problem, TTRP). | 21/5/2024 | PB |
| Diseñar los casos de prueba necesarios para validar el funcionamiento de las heurísticas de construcción existentes en BHCVRP para el Problema de Planificación de Rutas de Vehículos con Flota Heterogénea (Heterogenous Fleet VRP, HFVRP). | 24/5/2024 | PB |
| Ejecutar en una herramienta los casos de prueba diseñados para validar el funcionamiento de las heurísticas de construcción existentes en BHCVRP para los distintos VRP: CVRP, MDVRP, TTRP y HFVRP. | 27/5/2024 | PB |
| Entrega de versión del informe de PP (se agrega primera versión del Cap 1, con citas desde el texto, y versión mejorada de la bibliografía) | 29/5/2024 | EE |
| Implementar en Python las clases necesarias para modelar CVRP, MDVRP, TTRP y HFVRP. | 3/6/2024 | AS, PG |
| Implementar en Python el patrón *Factory Method* y las clases necesarias para la creación de distancias y heurísticas de construcción existentes en la versión Java, utilizando tratamiento de excepciones. | 10/6/2024 | AS, PG |
| Entrega de versión del informe de PP (se completa Cap 1, primera versión Cap 2, refinar) | 12/6/2024 | EE |
| Implementar en Python las clases necesarias para modelar las rutas, soluciones y operadores de post-optimización, utilizando tratamiento de excepciones.. | 17/6/2024 | AS, PG |
| Implementar en Python la nueva heurística de construcción Inserción de Kilby, utilizando tratamiento de excepciones. | 24/6/2024 | PG |
| Entrega de versión del informe de PP (se completa Cap 1 y 2, primera versión Cap 3, refinar) | 26/6/2024 | EE |
| Implementar en Python la nueva heurística de construcción Algoritmo de Ahorro basado en *Matching*, utilizando tratamiento de excepciones. | 1/7/2024 | PG |
| Diseñar los casos de prueba necesarios para validar el funcionamiento de las nuevas heurísticas de construcción incorporadas para diferentes instancias de VRP. | 3/7/2024 | PB |
| Ejecutar en una herramienta los casos de prueba diseñados para validar el funcionamiento de las nuevas heurísticas de construcción incorporadas para diferentes instancias de VRP. | 8/7/2024 | PB |
| Comparar los resultados obtenidos con OR-Tools, aplicando pruebas estadísticas. | 10/7/2024 | PB |
| Entrega de versión del informe de PP (COMPLETA) | 10/7/2024 | EE |
| Entrega del informe final de tesis al tutor (lista para vistazo final) | 18/7/2024 | JP, EE |
| Rectificar señalamientos del informe de PP. | 20/7/2024 | EE |
| Entrega del informe final de PP y preparación para la defensa. | 21/7/2024 | EE |

|  |  |
| --- | --- |
| \_\_\_\_\_\_Isis Torres Pérez\_\_\_\_\_\_\_\_\_  Nombre completo y firma del primer tutor | Alejandro Rosete Suárez\_  Nombre completo y firma del segundo tutor |
| Ananda de la Caridad Morales Morales  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:  **AN**- Analista de negocio **AR**- Arquitecto **JP**-Jefe de proyecto  **AS**- Analista de sistema **DS**- Diseñador de software **ES**- Especialista de seguridad  **AD**- Analista de datos **DI**- Diseñador de Interfaz hombre – máquina **EE**- Escritor-expositor de trabajos técnicos  **PG**- Programador **DB**- Diseñador de base de datos **II**- Habilitador de Infraestructuras Informáticas  **PB**- Probador **GC**- Gestor de Configuración **TD**- Facilitador de la Toma de Decisiones |

**Índice**

[Introducción 1](#_Toc176801203)

[Capítulo 1: Fundamentos teóricos 8](#_Toc176801204)

[1.1 Introducción 8](#_Toc176801205)

[1.2 Problema de optimización combinatoria 8](#_Toc176801206)

[1.2.1 Problema de planificación de rutas de vehículos 11](#_Toc176801207)

[1.2.2 Variantes de problemas de planificación de rutas de vehículos 13](#_Toc176801208)

[1.3 Algoritmos heurísticos 16](#_Toc176801209)

[1.3.1 Heurísticas de construcción 18](#_Toc176801210)

[1.4 Bibliotecas que implementan métodos heurísticos 21](#_Toc176801211)

[1.5 Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos 27](#_Toc176801212)

[1.5.1 Vista de la arquitectura de BHCVRP 27](#_Toc176801213)

[1.5.2 Deficiencias en BHCVRP 30](#_Toc176801214)

[1.6 Conclusiones parciales 31](#_Toc176801215)

[Capítulo 2: Diseño de la solución 33](#_Toc176801216)

[2.1 Introducción 33](#_Toc176801217)

[2.2 Modificaciones en la arquitectura de BHCVRP 33](#_Toc176801218)

[2.2.1 Nuevas heurísticas de construcción en BHCVRP 34](#_Toc176801219)

[2.3 Nueva versión de BHCVRP 37](#_Toc176801220)

[2.3.1 Particularidades de BHCVRP versión Java 39](#_Toc176801221)

[2.3.2 Particularidades de BHCVRP versión Python 39](#_Toc176801222)

[2.3.3 Paquetes y funcionalidades en común 40](#_Toc176801223)

[2.4 Principios de diseño 53](#_Toc176801224)

[2.5 Patrones de diseño 54](#_Toc176801225)

[2.5.1 Patrón Singleton 55](#_Toc176801226)

[2.5.2 Patrón *Factory Method* 57](#_Toc176801227)

[2.5.3 Patrón *Template* 59](#_Toc176801228)

[2.5.4 Otros patrones de diseño 60](#_Toc176801229)

[2.6 Conclusiones parciales 62](#_Toc176801230)

[Capítulo 3: Análisis experimental 63](#_Toc176801231)

[3.1 Introducción 63](#_Toc176801232)

[3.2 Descripción de los experimentos 63](#_Toc176801233)

[3.2.1 Descripción de las instancias 63](#_Toc176801234)

[3.2.2 Configuración de los experimentos 67](#_Toc176801235)

[3.3 Análisis de los resultados 71](#_Toc176801236)

[3.3.1 Resultados obtenidos con BHCVRP en Python 72](#_Toc176801237)

[3.3.2 Comparación de los resultados con la versión anterior de BHCVRP 86](#_Toc176801238)

[3.3.3 Comparación de los resultados con OR-Tools 105](#_Toc176801239)

[3.4 Conclusiones parciales 110](#_Toc176801240)

[Conclusiones 112](#_Toc176801241)

[Recomendaciones 114](#_Toc176801242)

[Referencias bibliográficas 115](#_Toc176801243)

**Índice de tablas**

[Tabla 1: Comparación de bibliotecas de clases que implementan métodos heurísticos. 25](#_Toc176801244)

[Tabla 2: Descripción de las clases que integran el paquete *data*. 42](#_Toc176801245)

[Tabla 3: Descripción de las clases que integran el paquete *solution*. 43](#_Toc176801246)

[Tabla 4: Descripción de las clases que integran el paquete *postoptimization*. 44](#_Toc176801247)

[Tabla 5: Descripción de las clases que integran el paquete *tools*. 45](#_Toc176801248)

[Tabla 6: Descripción de las clases que integran el paquete *distance*. 46](#_Toc176801249)

[Tabla 7: Descripción de las clases que integran el paquete *interfaces.* 48](#_Toc176801250)

[Tabla 8: Descripción de las clases que integran el paquete *methods*. 49](#_Toc176801251)

[Tabla 9: Descripción de las clases que integran el paquete *heuristic*. 51](#_Toc176801252)

[Tabla 10: Descripción de las clases que integran el paquete exceptions. 52](#_Toc176801253)

[Tabla 11: Descripción de las clases que integran el paquete *controller*. 53](#_Toc176801254)

[Tabla 12: Descripción de las instancias CVRP. 64](#_Toc176801255)

[Tabla 13: Descripción de las instancias HFVRP. 65](#_Toc176801256)

[Tabla 14: Descripción de las instancias MDVRP. 66](#_Toc176801257)

[Tabla 15: Descripción de las instancias TTRP. 67](#_Toc176801258)

[Tabla 16: Resumen de los resultados en las ocho instancias CVRP (I). 73](#_Toc176801259)

[Tabla 17: Resumen de los resultados en las ocho instancias CVRP (II). 73](#_Toc176801260)

[Tabla 18: Resumen de los resultados en las ocho instancias HFVRP (I). 76](#_Toc176801261)

[Tabla 19: Resumen de los resultados en las ocho instancias HFVRP (II). 76](#_Toc176801262)

[Tabla 20: Resumen de los resultados en las ocho instancias MDVRP (I). 80](#_Toc176801263)

[Tabla 21: Resumen de los resultados en las ocho instancias MDVRP (II). 80](#_Toc176801264)

[Tabla 22: Resumen de los resultados en las ocho instancias TTRP (I). 83](#_Toc176801265)

[Tabla 23: Resumen de los resultados en las ocho instancias TTRP (II). 83](#_Toc176801266)

[Tabla 24: Rankings de las heurísticas para las cuatro variantes VRP. 86](#_Toc176801267)

[Tabla 25: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias CVRP (I). 87](#_Toc176801268)

[Tabla 26: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias CVRP (II). 87](#_Toc176801269)

[Tabla 27: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias HFVRP (I). 88](#_Toc176801270)

[Tabla 28: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias HFVRP (II). 88](#_Toc176801271)

[Tabla 29: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias MDVRP (I). 89](#_Toc176801272)

[Tabla 30: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias MDVRP (II). 89](#_Toc176801273)

[Tabla 31: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias TTRP (I). 90](#_Toc176801274)

[Tabla 32: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias TTRP (II). 90](#_Toc176801275)

[Tabla 33: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a costo para la variante CVRP. 96](#_Toc176801276)

[Tabla 34: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a costo para la variante HFVRP. 97](#_Toc176801277)

[Tabla 35: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a costo para la variante MDVRP. 98](#_Toc176801278)

[Tabla 36: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a costo para la variante TTRP. 99](#_Toc176801279)

[Tabla 37: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a tiempo para la variante CVRP. 101](#_Toc176801280)

[Tabla 38: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a tiempo para la variante HFVRP. 102](#_Toc176801281)

[Tabla 39: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a tiempo para la variante MDVRP. 103](#_Toc176801282)

[Tabla 40: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a tiempo para la variante TTRP. 104](#_Toc176801283)

[Tabla 41: Resumen de los resultados obtenidos por BHCVRP y OR-Tools en las ocho instancias CVRP. 105](#_Toc176801284)

[Tabla 42: Resumen de los resultados obtenidos por BHCVRP y OR-Tools en las ocho instancias HFVRP. 106](#_Toc176801285)

[Tabla 43: Resultados de la prueba de Wilcoxon en la comparación con los resultados de BHCVRP y OR-Tools en cuanto a costo para las variantes CVRP y HFVRP. 107](#_Toc176801286)

[Tabla 44: Resultados de la prueba de Wilcoxon en la comparación con los resultados de BHCVRP y OR-Tools en cuanto a tiempo de ejecución para las variantes CVRP y HFVRP. 109](#_Toc176801287)

**Índice de figuras**

[Figura 1: Clases de complejidad en problemas de optimización [39]. 9](#_Toc176801288)

[Figura 2: Métodos de optimización combinatoria [23]. 10](#_Toc176801289)

[Figura 3: Representación gráfica de un VRP [39]. 11](#_Toc176801290)

[Figura 4: Clasificación de los algoritmos aproximados [23]. 16](#_Toc176801291)

[Figura 5: Patrón n-capas con enfoque basado en reutilización de BHCVRP [39]. 28](#_Toc176801292)

[Figura 6: Patrón n-capas con enfoque basado en reutilización de la nueva versión de BHCVRP. 38](#_Toc176801293)

[Figura 7: Diagrama de clases del paquete *data*. 41](#_Toc176801294)

[Figura 8: Diagrama de clases del paquete *solution*. 43](#_Toc176801295)

[Figura 9: Diagrama de clases del paquete *postoptimization*. 44](#_Toc176801296)

[Figura 10: Diagrama de clases del paquete *tools*. 45](#_Toc176801297)

[Figura 11: Diagrama de clases del paquete *distance*. 46](#_Toc176801298)

[Figura 12: Diagrama de clases del paquete *factory*. 47](#_Toc176801299)

[Figura 13: Diagrama de clases del paquete *heuristic*. 50](#_Toc176801300)

[Figura 14: Diagrama de clases del paquete *exceptions*. 52](#_Toc176801301)

[Figura 15: Implementación del patrón Singleton en la clase *StrategyHeuristic*. 56](#_Toc176801302)

[Figura 16: Implementación del patrón Factory Method. 58](#_Toc176801303)

[Figura 17: Implementación del patrón *Template Method*. 60](#_Toc176801304)

[Figura 18: Gráfica de barras para el costo de las heurísticas en la variante CVRP. 74](#_Toc176801305)

[Figura 19: Gráfica de barras para el tiempo de ejecución de las heurísticas en la variante CVRP. 75](#_Toc176801306)

[Figura 20: Gráfica de barras para el costo de las heurísticas en la variante HFVRP. 78](#_Toc176801307)

[Figura 21: Gráfica de barras para el tiempo de ejecución de las heurísticas en la variante HFVRP. 79](#_Toc176801308)

[Figura 22: Gráfica de barras para el costo de las heurísticas en la variante MDVRP. 81](#_Toc176801309)

[Figura 23: Gráfica de barras para el tiempo de ejecución de las heurísticas en la variante MDVRP. 82](#_Toc176801310)

[Figura 24: Gráfica de barras para el costo de las heurísticas en la variante TTRP. 84](#_Toc176801311)

[Figura 25: Gráfica de barras para el tiempo de ejecución de las heurísticas en la variante TTRP. 85](#_Toc176801312)

[Figura 26: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en distancia de las soluciones para la variante CVRP. 92](#_Toc176801313)

[Figura 27: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en distancia de las soluciones para la variante HFVRP. 93](#_Toc176801314)

[Figura 28: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en distancia de las soluciones para la variante MDVRP. 94](#_Toc176801315)

[Figura 29: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en distancia de las soluciones para la variante TTRP. 95](#_Toc176801316)

[Figura 30: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en tiempo de las soluciones para la variante CVRP. 100](#_Toc176801317)

[Figura 31: Comportamiento de las soluciones obtenidas por BHCVRP y OR-Tools en cuanto a tiempo de ejecución para la variante CVRP. 108](#_Toc176801318)

[Figura 32: Comportamiento de las soluciones obtenidas por BHCVRP y OR-Tools en cuanto a tiempo de ejecución para la variante HFVRP. 108](#_Toc176801319)

# **Introducción**

Un problema de optimización combinatoria tiene como objetivo encontrar la mejor solución entre un conjunto finito de posibilidades. Cada solución está formada por una combinación de elementos de un conjunto dado. Estos problemas suelen involucrar la selección de un subconjunto de elementos que maximice o minimice una función objetivo, sujeto a ciertas restricciones [1]. En esta categoría se encuentra un problema bastante frecuente en los sistemas logísticos, que es la distribución de bienes y servicios desde ciertos depósitos a usuarios dispersos geográficamente. A través de estudios realizados en el campo de la Investigación de Operaciones, se determina que un correcto manejo de estos problemas puede producir ahorros considerables en los costos totales de transportación. En la mayoría de las ocasiones, se desea mejorar la planificación eficiente de las rutas de los vehículos, determinando cómo visitar a los clientes para satisfacer sus demandas [2]. En este contexto, surge el denominado Problema de Planificación de Rutas de Vehículos (VRP, por las siglas en inglés de *Vehicle Routing Problem*) [3].

De manera general, un VRP busca diseñar un conjunto de rutas para que una flota de vehículos preste servicio a un conjunto de clientes, sujeto a una serie de restricciones. Este problema es empleado en la gestión de la cadena de suministro en la entrega física de bienes y servicios. Existen diversas variantes de VRP y se formulan en función de la naturaleza de los bienes transportados, la calidad del servicio requerido, las características de los clientes y los vehículos, entre otros [4]. Según [5], los problemas de planificación de rutas de vehículos se introducen en 1959 y se describe como una variante generalizada del Problema del Viajante de Comercio (TSP, por las siglas de *Traveling Salesman Problem*) [6]. En 1964, se propone el primer algoritmo efectivo para resolver VRP, denominado Algoritmo de Ahorros [7]. A partir de estos sucesos, incrementan las investigaciones y trabajos en el área de la planificación de rutas de vehículos. Por consiguiente, surgen dos enfoques: definición de modelos que incorporen cada vez más características de la vida real y búsqueda de algoritmos para resolver eficientemente estos problemas [8].

Para resolver situaciones prácticas se proponen modelos básicos que pueden ser extendidos a diferentes campos de la logística y el transporte [9, 10]. Algunos ejemplos de aplicaciones prácticas de VRP se encuentran en el transporte obrero, las rutas de ómnibus, la recogida de desechos sólidos, la distribución del periódico y correo, entre otros. Estas situaciones cotidianas pueden ser modeladas con sus características, a través de las diferentes variantes de VRP existentes. Dentro de las variantes más conocidas se encuentran: el Problema de Planificación de Rutas de Vehículos con Capacidades (CVRP) [11], el Problema de Planificación de Rutas de Vehículos con Múltiples Depósitos (MDVRP) [12], el Problema de Planificación de Rutas de Vehículos con Flota Heterogénea (HFVRP) [13], el Problema de Planificación de Rutas de Camiones y Remolques (TTRP) [14, 15], el Problema de Planificación de Rutas de Autobuses Escolares (SBRP) [16], el Problema de Planificación de Rutas de Vehículos con Ventanas de Tiempo (VRPTW) [17], entre otros [18-21].

Los problemas de planificación de rutas de vehículos son mayormente complejos de resolver para instancias que contengan un número de clientes elevado. Este fenómeno se debe al esfuerzo computacional requerido y a que los VRP pertenecen a la clase de problemas NP-duro [9]. Los problemas en esta categoría son considerados muy difíciles de resolver [22]. Para desempeñar esta tarea, en ocasiones se necesita comprometer requisitos como la optimalidad y construir una buena solución, a pesar de no garantizar el óptimo. Para eso, se aplican los algoritmos heurísticos, específicamente las heurísticas y metaheurísticas, como alternativa de solución [23].

Particularmente, el propósito de las heurísticas es guiar el proceso de búsqueda en la mejor dirección y sugerir qué camino tomar cuando existen varias opciones disponibles [24, 25]. El término heurístico se refiere a un procedimiento que trata de aportar soluciones a un problema con un buen rendimiento, en cuanto a calidad de soluciones y recursos empleados. Esto significa que constituyen métodos más sencillos que brindan soluciones satisfactorias a un problema dado mediante algoritmos específicos. Además, pueden ofrecer la solución final del problema o ser aplicadas dentro del contexto de las metaheurísticas [8, 25].

Por otra parte, las heurísticas se clasifican en dos grupos: heurísticas de construcción y heurísticas de mejora [26, 27]. Las heurísticas de construcción se encargan de construir literalmente una solución del problema, paso por paso [26]. Sin embargo, las heurísticas de mejora parten de una solución y reparan las rutas previamente construidas, con el objetivo de conseguir una determinada mejoría [25]. Dentro de las heurísticas de construcción clásicas de la literatura, utilizadas mayormente para la resolución de los VRP, se encuentran: el Algoritmo de Ahorros [5], el Algoritmo de Barrido [28], la Heurística de Inserción en Paralelo de Christofides, Mingozzi y Toth [29], Heurística de Inserción Secuencial de Mole & Jameson [30], entre otras [19, 24, 31, 32].

Debido a la necesidad de resolver problemas de optimización combinatoria utilizando dichos algoritmos y favorecer la reutilización de sus códigos, es apropiado agruparlos dentro de componentes de software. Actualmente, existen pocas bibliotecas que implementan métodos heurísticos para resolver, ya sea VRP u otros problemas de optimización. Algunos de los componentes de software que cumplen con dichas características son:

* BiCIAM: biblioteca de clases en Java, que implementa un modelo unificado de metaheurísticas mono-objetivo y multi-objetivo [33, 34].
* METSlib: biblioteca que implementa metaheurísticas básicas de la literatura [35].
* MALLBA: biblioteca desarrollada en C++, que contiene modelos para la optimización combinatoria [36].
* VRPH: biblioteca de heurísticas de búsqueda local para VRP en C++ [37].
* MOMHLib++: biblioteca diseñada en C++, para la implementación de metaheurísticas multi-objetivos [38].
* BHCVRP: biblioteca de heurísticas de construcción para VRP en Java [39, 40].
* OR-Tools: software de código abierto desarrollado principalmente en Python, para resolver problemas de optimización combinatoria, incluyendo VRP en sus distintas variantes como CVRP y HFVRP [41].
* JSprit: herramienta de código abierto desarrollada en Java, que implementa el principio *Ruin and Recreate* [42]y resuelve varias variantes VRP [43].
* VROOM: software de código abierto desarrollado en C++, para resolver VRP utilizando heurísticas para obtener la solución inicial, en dependencia de la variante del problema [44].

De los componentes de software antes mencionados, los que permiten resolver VRP son las bibliotecas BiCIAM [33, 34], VRPH [37] y BHCVRP [39, 40], y las herramientas OR-Tools [41], JSprit [43] y VROOM [44]. De este conjunto, BHCVRP es la única que permite hacerlo a través de heurísticas de construcción exclusivamente. Generalmente, en la literatura se evidencia la carencia de bibliotecas de heurísticas para la resolución de problemas de optimización combinatoria, debido a que la mayor parte del desarrollo se centra en los algoritmos metaheurísticos. Además, existen pocas adaptaciones de las heurísticas a las distintas variantes VRP, ya que no se consideran las características propias de cada uno de estos problemas. Por último, la mayoría de los componentes de software existentes para solucionar estos problemas están desarrollados para C++ y Java. Debido al crecimiento de la comunidad en Python para trabajos con Inteligencia Artificial y problemas de optimización, sería interesante profundizar en cómo solucionar VRP con heurísticas desde este lenguaje [45, 46].

La Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos (BHCVRP) es una propuesta realizada por el proyecto de Optimización y Metaheurísticas de la Facultad de Ingeniería Informática de la Universidad Tecnológica de La Habana “José Antonio Echeverría” (CUJAE) [39, 40]. Esta biblioteca está desarrollada en Java y contiene la adaptación de siete heurísticas de construcción de la literatura para las variantes CVRP, MDVRP, HFVRP y TTRP. A pesar de constituir un aporte, presenta deficiencias en su diseño y código fuente relacionado con las heurísticas implementadas, así como dificultades en la interoperabilidad con otros componentes.

A partir de la situación descrita anteriormente se plantea como problema a resolver la existencia de deficiencias en la arquitectura de BHCVRP, teniendo en cuenta la manera en que se encuentran implementadas las heurísticas para resolver cada variante VRP.

En este trabajo, el objeto de estudio se enmarca en los problemas de optimización combinatoria, los algoritmos heurísticos y el diseño de software. Específicamente, el campo de acción se enfoca en los Problemas de Planificación de Rutas de Vehículos, las heurísticas de construcción y los patrones de diseño. Para dar solución al problema planteado se identifica como objetivo general: Desarrollar una nueva versión de BHCVRP en Java y Python, que permita incorporar otras heurísticas de construcción de la literatura para diferentes variantes de problemas de planificación de rutas de vehículos.

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

1. Caracterizar el marco teórico investigativo sobre los problemas de planificación de rutas de vehículos, el uso de los algoritmos heurísticos para resolverlos y los componentes de software que agrupan estos algoritmos para su reutilización.
   1. Realizar búsquedas bibliográficas para identificar las características que presentan los problemas de planificación de rutas de vehículos.
   2. Realizar búsquedas bibliográficas para realizar un estudio de la literatura referente a los resultados obtenidos por algoritmos heurísticos en distintas variantes VRP.
   3. Realizar un análisis de los componentes de software existentes y las tecnologías, para la resolución de variantes VRP haciendo uso de algoritmos heurísticos.
2. Rediseñar la arquitectura de la biblioteca de clases para la utilización de otras heurísticas de construcción en diferentes variantes VRP.
   1. Analizar la versión de BHCVRP propuesta anteriormente para determinar los cambios a realizar.
   2. Diseñar las modificaciones necesarias en la arquitectura para permitir la incorporación de otras variantes VRP y heurísticas de construcción.
3. Implementar adaptaciones de otras heurísticas de construcción para los problemas de planificación de rutas de vehículos.
   1. Analizar nuevas heurísticas de construcción y variantes VRP para su incorporación en la nueva versión de la biblioteca.
   2. Seleccionar las heurísticas de construcción y variantes VRP a incorporar a la biblioteca.
   3. Implementar en Python y Java, las clases y métodos necesarios para adaptar las heurísticas seleccionadas teniendo en cuenta las nuevas características que presenten las variantes VRP a incorporar.
4. Validar el funcionamiento de las heurísticas adaptadas a través de un conjunto de experimentos con instancias de la literatura.
   1. Definir los escenarios para cada uno de los experimentos con las heurísticas adaptadas.
   2. Analizar los resultados obtenidos con cada experimento utilizando pruebas estadísticas no paramétricas.
   3. Comparar los resultados obtenidos para las variantes CVRP, MDVRP, HFVRP y TTRP con los resultados obtenidos en la versión anterior de BHCVRP, verificando que funcionen igual.
   4. Comparar los resultados actuales obtenidos para las variantes CVRP y HFVRP con los resultados de la herramienta OR-Tools, identificando cuál es mejor para cada caso.

Este trabajo solo se enfoca en los objetivos específicos 1 y 2. Del objetivo específico 3, se estarán realizando las tareas desde el enfoque de las nuevas heurísticas de construcción en Python. El objetivo específico 4 será cumplido parcialmente, ya que los experimentos se harán con un alcance limitado. Con vistas a la tesis se incorporarán nuevas variantes VRP, otros métodos de post-optimización, entre otras tareas.

El cumplimiento del objetivo general tiene como valor práctico un aporte significativo en la resolución de diferentes VRP a través de la biblioteca desarrollada en Python. Se propone como artefacto de salida un diseño de software, según la clasificación presentada en [47]. Es importante destacar que los resultados obtenidos ponen a disposición de la comunidad científica un componente de software que brinda nuevos algoritmos para la resolución de distintas variantes de VRP. Además, la biblioteca permite la colaboración con otros componentes que emplean metaheurísticas o sistemas de información geográficos con servicios de planificación de rutas de vehículos.

Para alcanzar los objetivos mencionados el trabajo se estructura en tres capítulos, con secciones importantes, tales como conclusiones, recomendaciones y referencias bibliográficas. El Capítulo 1: Fundamentos teóricos, presenta una revisión detallada del estado del arte sobre VRP, sus características y variantes. Por otra parte, abarca los algoritmos heurísticos, describe algunas de las heurísticas de construcción más utilizadas en estos problemas y presenta los componentes de software relacionados con estas temáticas. El Capítulo 2: Diseño de la solución, describe la solución mediante la vista de la arquitectura de la biblioteca de clases y los patrones implementados. Además, incluye elementos como la adaptación realizada de los nuevos algoritmos incorporados. El Capítulo 3: Análisis experimental, se encarga de demostrar la validación de la solución mediante un conjunto de experimentos. Además, se presentan los resultados alcanzados y un análisis del comportamiento de dichas heurísticas aplicando pruebas estadísticas no paramétricas. Por último, se realizan dos comparaciones: una para garantizar igualdad en las implementaciones tanto en Java como en Python, y otra entre OR-Tools y la versión de BHCVRP en Python, para identificar cuál funciona mejor en diferentes variantes VRP.

# **Capítulo 1: Fundamentos teóricos**

## **1.1 Introducción**

En este capítulo se describen los aspectos teóricos de la investigación relacionados con los problemas de optimización combinatoria, específicamente los problemas de planificación de rutas de vehículos y sus variantes. Por otra parte, se presentan los métodos heurísticos como principal alternativa de solución para estos problemas. Además, se enfatiza en las heurísticas de construcción y en las bibliotecas que implementan dichos métodos. Por último, se describe la arquitectura de la Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos.

## **1.2 Problema de optimización combinatoria**

En el mundo actual, con el aumento de su complejidad y competitividad, se requiere cada vez más del proceso de toma de decisiones en disímiles actividades. Por ende, la optimización adquiere mayor importancia en las aplicaciones de la vida real donde se necesita seleccionar la mejor opción. La optimización consiste en hallar el valor máximo o mínimo de una o varias funciones objetivo, donde se cumplen determinadas restricciones que condicionan la elección de la solución adecuada y el propósito es alcanzar costos mínimos o máximos beneficios [17].

Dentro del campo de la optimización se encuentran los problemas lineales, considerados los más sencillos de resolver debido a la linealidad de su función objetivo y restricciones. La mayoría de los problemas lineales se pueden solucionar a partir de métodos exactos como el método Simplex [48]. Existen algunos casos específicos donde se emplean variantes de dicho método, por ejemplo: Simplex Dual [48], Programación Lineal Paramétrica [48], Técnica de Ramificación y Acotamiento [48], entre otros. Aunque, la mayor parte de los problemas de optimización están catalogados como difíciles de resolver [26].

A continuación, en la Figura 1, se presenta la relación existente entre las diferentes clases utilizadas para definir distintos problemas de optimización según su complejidad [23]:

* P: problemas que pueden ser resueltos por algoritmos que demoran un tiempo polinomial en resolverlos.
* NP: problemas que pudieran ser resueltos por un algoritmo no determinístico en un tiempo polinomial.
* NP-completo: problemas que actualmente no cuentan con un algoritmo capaz de obtener el estado óptimo en un tiempo polinomial, pero que tampoco se ha probado que no pueda existir dicho algoritmo, para al menos un problema.
* NP-duro: problemas NP-completo que no han podido ser reducidos a problemas NP. Son considerados intratables o muy difíciles de resolver.



Figura 1: Clases de complejidad en problemas de optimización [39].

Teniendo en cuenta lo mencionado anteriormente, aparece la optimización combinatoria enmarcada dentro de la rama denominada optimización en matemática aplicada y en ciencias de la computación. Los problemas de optimización combinatoria están presentes en múltiples problemas de la vida cotidiana, desde los problemas de planificación de rutas de vehículos, asignación de recursos hasta problemas de ingeniería de software; de forma tal, que están presentes en el campo industrial y científico [26].

Para resolver los problemas de optimización combinatoria surgen diversas técnicas, representadas en la Figura 2, que se clasifican en dos categorías fundamentales: exactas y aproximadas. Los métodos exactos garantizan encontrar la solución óptima, aunque con el aumento del tamaño de las instancias se afecta de manera significativa el tiempo de ejecución. Esta vía de solución es aplicable solamente a instancias de tamaño pequeño y mediano, por tanto, es necesaria la aplicación de métodos aproximados donde intervienen los algoritmos heurísticos. Estos métodos heurísticos sacrifican la posibilidad de alcanzar el óptimo, pero solucionan grandes instancias en un tiempo razonable [49].

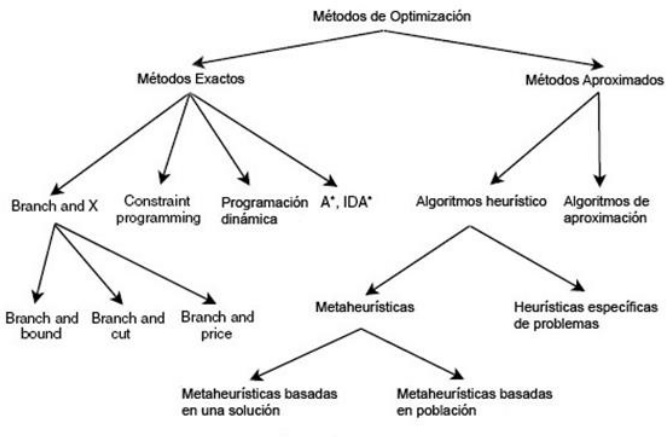


Figura 2: Métodos de optimización combinatoria [23].

Un problema de optimización combinatoria puede representarse mediante un modelo matemático con ecuaciones y expresiones que describen el problema en cuestión. Dichos modelos están formados por:

* Restricciones: imponen ciertos límites a los valores posibles, es decir, los coeficientes son entradas del modelo, donde su lado derecho es un valor numérico que representa una cota inferior o superior [50].
* Variables: constituyen las salidas del modelo y generalmente poseen un valor numérico, aunque existen otras maneras de codificar algunos tipos de información según el contexto [50].
* Función objetivo: determina la efectividad para cada solución. Según la cantidad de objetivos a tratar de resolver, los problemas de optimización se clasifican en mono-objetivos o multi-objetivos. El propósito de los mono-objetivos es obtener una solución óptima para un objetivo determinado; puede ser máximo o mínimo. Sin embargo, los multi-objetivos obtienen un conjunto de soluciones conocidas como Frente de Pareto [51].

### **1.2.1 Problema de planificación de rutas de vehículos**

El Problema del Viajante de Comercio (TSP, por las siglas en inglés de *Travelling Salesman Problem*) [52] constituye el punto de partida para la formulación de otros problemas de optimización combinatoria más complejos y utilizados en situaciones prácticas, como es el caso de los problemas de planificación de rutas de vehículos. El TSP se basa en un conjunto de ciudades y el costo que se obtiene al viajar de un punto a otro. Posee como objetivo fundamental encontrar la ruta de costo mínimo para visitar todas las ciudades pasando solo una vez por cada una de ellas, y regresando al punto de partida [53]. Por tanto, el TSP es considerado el problema de planificación de rutas de vehículos más simple, a pesar de pertenecer a la clase de problemas NP-duro y ser uno de los problemas de optimización combinatoria más difundidos [31, 54].

Los Problemas de Planificación de Rutas de Vehículos (VRP, por las siglas en inglés de *Vehicle Routing Problems*) modelan el fenómeno de la distribución de bienes y servicios. De manera general, un VRP consiste determinar el conjunto de rutas de costo mínimo que cumplan con las restricciones del problema, dado una cantidad de clientes dispersos geográficamente y una flota de vehículos perteneciente a uno o varios depósitos [31]. A continuación, en la Figura 3 se muestra una visualización de este problema:

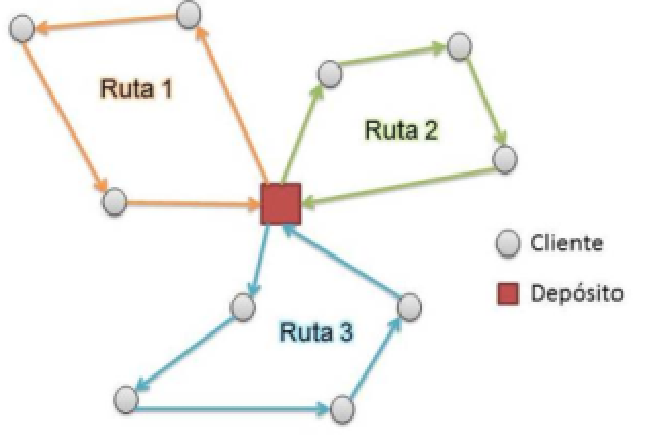


Figura 3: Representación gráfica de un VRP [39].

Los vehículos deben ser capaces de satisfacer cierta demanda por parte de los clientes. En la mayoría de los casos, la demanda constituye un bien que ocupa lugar en los vehículos y lo común es que no satisfaga la demanda de todos los clientes en una misma ruta. Por otra parte, existen otras situaciones, donde la demanda es un servicio, es decir, el cliente espera ser visitado por el vehículo. En ocasiones, los clientes presentan restricciones relacionadas con un horario de servicio, por lo que surgen los intervalos o ventanas de tiempo en los que el vehículo puede arribar al cliente [9]. Además, en problemas donde los vehículos poseen diferentes características pueden surgir restricciones de compatibilidad con los clientes. Como alternativa de solución para esta variante se decide que cada cliente sólo pueda ser visitado por algunos de los vehículos [55].

En la vida real, los vehículos y mercancías a distribuir son ubicadas en depósitos. Generalmente, se exige que la ruta comience y finalice en el mismo depósito, aunque el recorrido puede terminar en un punto diferente en algunos casos. Por otra parte, existen problemas con múltiples depósitos, donde cada uno de estos presenta características diferentes como: ubicación, cantidad y capacidad de sus vehículos [56].

Es importante destacar, que cada vehículo posee un costo variable proporcional asociado a la distancia recorrida. Los problemas que contienen vehículos con capacidades iguales se denominan de flota homogénea, y en caso contrario, de flota heterogénea. Usualmente, la cantidad de vehículos disponibles se trata como un parámetro de entrada o una variable de decisión. Además, se asume que cada vehículo recorre solo una ruta en el período de planificación, aunque existen variantes donde un vehículo pudiera realizar más de un recorrido [57].

En la actualidad se realiza un constante esfuerzo para resolver los problemas de planificación de rutas de vehículos. Según [5], en 1959, se realiza por primera vez una formulación de dicho problema para una aplicación de distribución de combustible. En 1964, [7] propone el primer algoritmo efectivo para su resolución: el conocido Algoritmo de Ahorros. Por tanto, la esfera de la planificación de rutas de vehículos ha crecido de manera exponencial, tomando dos perspectivas fundamentales. Se trabaja hacia la definición de modelos que incorporen características cada vez más cercanas a la realidad y hacia la búsqueda de algoritmos que permitan resolver estos problemas de manera eficiente [31].

Por consiguiente, surgen nuevas variantes de planificación de rutas de vehículos a partir de las necesidades existentes en la vida real. A continuación, el siguiente epígrafe describe algunas de las variantes más tratadas en la literatura.

### **1.2.2 Variantes de problemas de planificación de rutas de vehículos**

Los modelos básicos creados para los VRP son utilizados para resolver situaciones prácticas en el campo de la logística y el transporte [9, 58]. Dentro de las aplicaciones prácticas del VRP se encuentran: el transporte obrero, las rutas de ómnibus, la recolección de desechos sólidos, la distribución del periódico y correo, entre otros. Dichas situaciones cotidianas, pueden ser modeladas a través de diferentes variantes VRP existentes o la fusión de estas, respetando sus características. A continuación, se muestran algunas de estas variantes:

* **Problema de Planificación de Rutas de Vehículos con Capacidades** (CVRP, por las siglas en inglés de *Capacitated VRP*): constituye la variante clásica del problema de planificación de rutas de vehículos, incluyendo restricciones relacionadas con la capacidad de los vehículos. Además, se establece que cada cliente posee una demanda específica para ser atendida y la demanda total de los clientes en cada ruta no puede exceder la capacidad del vehículo empleado en dicha ruta. Por tanto, es un problema que presenta un conjunto de vehículos idénticos, ubicados en un depósito central y que satisface la demanda de los clientes, sujeto a restricciones de capacidad de sus vehículos [24, 59].
* **Problema de Planificación de Rutas de Vehículos con Múltiples Depósitos** (MDVRP, por las siglas en inglés de *Multi-Depot VRP*): es una extensión del clásico CVRP que incorpora varios depósitos con ubicaciones diferentes. En este problema cada vehículo debe comenzar y finalizar su recorrido en el mismo depósito. Además, cada depósito cuenta con una flota limitada de vehículos con capacidad restringida y homogénea, utilizada para satisfacer las demandas de los clientes, cuyos datos son conocidos de antemano. Según [60], si los clientes se agrupan cerca de los depósitos, el problema se puede modelar como CVRP independientes. Por otra parte, si los clientes y depósitos se encuentran mezclados, el problema debe ser resuelto en dos fases: asignación de clientes a depósitos y determinación de recorridos a realizar por cada depósito para visitar a los clientes asignados [61].
* **Problema de Planificación de Rutas de Vehículos con Flota Heterogénea** (HFVRP, por las siglas en inglés de *Heterogenous Fleet VRP*): esta variante presenta características y/o capacidades diferentes en los vehículos de la flota. La capacidad heterogénea es considerada para determinar las rutas a construir, debido a que un vehículo más grande puede realizar una ruta más larga o de mayor concentración de demanda. La cantidad de vehículos de cada tipo es limitada y existe un costo fijo o costo por unidad de distancia recorrida asociada a cada tipo de vehículo [56].
* **Problema de Planificación de Rutas de Camiones y Remolques** (TTRP, por las siglas en inglés de *Truck and Trailer Routing Problem*): este problema considera el empleo de remolques como parte de la flota de vehículos. Además, presenta ciertas particularidades como la definición de tipos de cliente debido a las restricciones de accesos existentes en las ubicaciones de los mismos. Existen clientes accedidos por camión y remolque, conocidos como clientes de vehículo completo (VC) y otros accedidos solamente por camión sin el remolque denominados clientes de camión puro (TC). Para este tipo de problema surgen diferentes soluciones a partir del diseño de tres tipos de rutas según los clientes a visitar y el vehículo utilizado (PTR, PVR o CVR) [55].
* **Problema de Planificación de Rutas de Autobuses Escolares** (SBRP, por las siglas en inglés de *School Bus Routing Problem*): se centra en el uso eficiente de una flota de vehículos para transportar a los estudiantes desde sus casas hasta la escuela, utilizando una cantidad determinada de autobuses. De manera general, trata de encontrar el programa más eficiente para una flota de autobuses escolares. Cada vehículo debe recoger a los estudiantes asignados en las paradas, cumpliendo con determinadas restricciones; por ejemplo, tiempo máximo de recorrido, capacidad máxima del autobús, ventanas de tiempo para la llegada a la escuela, entre otros. Este problema se puede clasificar según los siguientes criterios: cantidad de escuelas (una o múltiples), entorno del servicio (urbano o rural) y flotas mixtas (homogénea o heterogénea) [16].
* **Problema de Planificación de Rutas de Vehículos con Ventanas de Tiempo** (VRPTW, por las siglas en inglés de *VRP with Time Windows*): es considerada una extensión del CVRP, ya que considera las capacidades e incluye como restricción que cada cliente tiene asociada una ventana de tiempo. Esta restricción representa un horario de servicio en que los vehículos pueden arribar a los clientes y se define un tiempo de servicio. El objetivo de este problema radica en minimizar la distancia total de los repartos de manera que se cumpla con la demanda de todos los clientes posibles respetando sus respectivas ventanas de tiempo [58].

A partir del surgimiento de nuevas características en los problemas de la vida real, aparecen nuevas variantes, por ejemplo: el Problema de Planificación de Rutas de Vehículos Periódico (PVRP, *Periodic Vehicle Routing Problem*) [57], el Problema de Planificación de Rutas de Vehículos Dinámico (DVRP, *Dynamic Vehicle Routing Problem*) [62], el Problema de Planificación de Rutas de Vehículos Abierto (OVRP, *Open Vehicle Routing Problem*) [63, 64], el Problema de Planificación de Rutas de Vehículos con Entregas Divididas (SDVRP, *Split Delivery Vehicle Routing Problem*) [65], el Problema de Planificación de Rutas de Vehículos Estocástico (SVRP, *Stochastic Vehicle Routing Problem*) [66], el Problema de Planificación de Rutas de Vehículos con Devoluciones (VRPB, *Vehicle Routing Problem with Backhauls*) [64], el Problema de Planificación de Rutas de Vehículos Ecológicos (GVRP, *Green Vehicle Routing Problem*) [18], el Problema de Planificación de Rutas de Vehículos con Instalaciones de Satélites (VRPSF, *Vehicle Routing Problem with Satellites Fleet*) [67], el Problema de Planificación de Rutas de Vehículos Eléctricos (EVRP, *Electric Vehicle Routing Problem*) [21], el Problema de Planificación de Rutas de Vehículos con Dependencia del Tiempo (TDVRP, *Time-dependent Vehicle Routing Problem*) [64], entre otros. Además, en ocasiones, se requiere la fusión de las variantes mencionadas anteriormente para lograr solucionar una situación del mundo real.

Los problemas de planificación de rutas de vehículos, generalmente, son complejos de resolver para instancias con un número de clientes elevado, debido al costo computacional requerido y a que pertenecen a la categoría de los problemas NP-duro. Con el objetivo de solucionar estos problemas, a veces se comprometen requisitos como la optimalidad y se construye una estructura de control para encontrar una buena solución, aunque no garantice la respuesta óptima. Por consiguiente, es común emplear algoritmos heurísticos o aproximados, como las heurísticas y metaheurísticas, en este tipo de problemas [23]. En el siguiente epígrafe se describen algunos de los algoritmos más abarcados en la literatura.

## **1.3 Algoritmos heurísticos**

Los métodos aproximados surgen a partir de la inminente necesidad de resolver problemas de mayor complejidad. Estos obtienen buenas soluciones, aunque no necesariamente la óptima. Además, la rapidez del proceso adquiere igual relevancia que la calidad de la solución obtenida [23]. A continuación, en la Figura 4, se muestran sus clasificaciones según [23]:

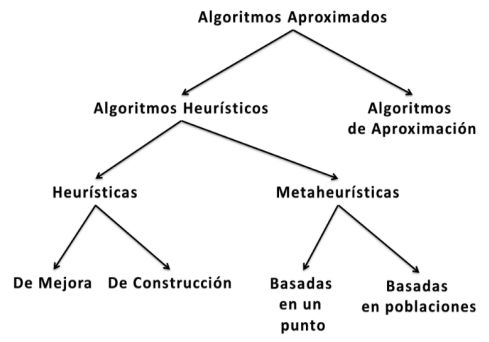


Figura 4: Clasificación de los algoritmos aproximados [23].

Los métodos heurísticos no siempre poseen una base matemática formal, pues pueden ser desarrollados por intuición. Las heurísticas están diseñadas para resolver un problema y/o instancia específica, mientras que las metaheurísticas se consideran algoritmos de propósito general utilizados para solucionar cualquier problema de optimización [23].

La palabra heurística es un término derivado de *heuriskein*, palabra griega que significa encontrar o descubrir. Este vocablo es empleado en el campo de la optimización para describir una clase de algoritmos de resolución de problemas [26]. El objetivo fundamental de una heurística constituye la guía del proceso de búsqueda en la dirección más apropiada, proporcionando el camino a tomar cuando hay varias opciones disponibles [24]. Específicamente, el término heurística se refiere al procedimiento que busca aportar soluciones a un problema con un buen rendimiento, en cuanto a calidad de soluciones y cantidad de recursos utilizados. Por tanto, se consideran métodos sencillos para obtener soluciones satisfactorias a un problema determinado con algoritmos específicos [8]. De manera general, este término se relaciona con la tarea de resolver con cierta inteligencia problemas reales a partir de conocimiento disponible [68]. Según [69], una heurística es un método que busca soluciones cercanas al óptimo, aunque no garantice encontrarlo, con un costo computacional razonable.

Es importante destacar, que las heurísticas son diseñadas para un problema específico [26]. Por consiguiente, según su naturaleza son enmarcadas en dos categorías generales:

* Heurísticas de construcción: consisten en construir paso a paso una solución del problema. En algunos casos son deterministas y suelen basarse en la mejor elección por cada iteración [26].
* Heurísticas de mejora: parten de una solución y realizan intercambios de clientes en las rutas para mejorar las soluciones. Esto implica reparar las rutas previamente construidas para obtener una determinada mejoría [62].

Por otra parte, existen métodos denominados metaheurísticas con el objetivo de obtener mejores resultados que los alcanzados por heurísticas tradicionales. El término metaheurística surge en 1986 según [70]. Las metaheurísticas constituyen una clase de algoritmos aproximados diseñados para resolver problemas difíciles de optimización combinatoria. Dichos métodos son procedimientos iterativos que combinan de manera inteligente diferentes conceptos para explorar el espacio de búsqueda y guían una heurística subordinada [26]. Además, las metaheurísticas son aplicadas cuando no existe un método exacto de resolución a un problema de optimización, requiere demasiado costo computacional o no se conoce la solución óptima. Por último, las metaheurísticas permiten enfrentar problemas de gran tamaño mediante la búsqueda de buenas soluciones en un tiempo razonable [70].

A continuación, se muestran algunas de las diferentes clasificaciones que toman los algoritmos metaheurísticos a partir de criterios basados en su filosofía y procedimientos [23]:

* Inspiradas o no en la naturaleza: se enfoca en los orígenes y no es muy relevante, ya que se dificulta clasificar algunas metaheurísticas recientes, generalmente híbridas. Dentro de los ejemplos de inspiración más comunes se encuentran los algoritmos evolutivos o genéticos, las colonias de hormigas y abejas, el recocido simulado de la física, así como la optimización de enjambres de partículas.
* Basadas en poblaciones o en una única solución: las metaheurísticas poblacionales proponen la evolución de toda una población de soluciones. Algunos ejemplos son el Enjambre de Partículas [23] y los Algoritmos Evolutivos [23]. Además, están orientadas a la exploración, permitiendo una mayor diversidad en el espacio de soluciones. Sin embargo, los algoritmos basados en una sola solución, manipulan y transforman la solución describiendo una trayectoria en el espacio de soluciones durante el proceso de búsqueda. Dentro de los ejemplos más conocidos se encuentran la Búsqueda Local [13], Recocido Simulado [55] y Búsqueda Tabú [71]. Por último, se enfocan en la explotación, pero no tienen el poder de intensificar la búsqueda en determinadas regiones.

Para dar cumplimiento a uno de los objetivos trazados, se explican las heurísticas de construcción en el próximo epígrafe.

### **1.3.1 Heurísticas de construcción**

La mayoría de las heurísticas poseen diversas naturalezas y son diseñadas para problemas específicos, dificultando su generalización o aplicación a problemas similares. Sin embargo, dentro de los casos especiales se encuentran los métodos de construcción, con el objetivo de construir literalmente paso a paso una solución del problema [26].

Las heurísticas de construcción (HC) para problemas de planificación de rutas de vehículos se dividen en cuatro grupos según [72]:

* Basadas en ahorros (*Saving Based*): mediante ahorros producidos por la sustitución de nodos en la ruta por otros que aún no están, se mantiene un proceso de constante intercambio mientras se cumplan las restricciones. Dentro de los algoritmos más conocidos se encuentran el Algoritmo de Ahorros (*Save Algorithm*) [7] con su versión secuencial y paralela, así como la variante basada en *Matching* [31].
* De inserción o basadas en coste (*Cost Based*): son empleadas para generar un conjunto de rutas iniciales. Se construye una solución mediante constantes inserciones de nodos [30]. Dentro de las heurísticas más empleadas se encuentran Vecino más cercano , Inserción Secuencial de Mole & Jameson [30] e Inserción de Kilby [62]*.*
* Ruta primero – Asignar después (*Route first – Cluster second*): se determina el recorrido para visitar a todos los clientes y luego se particiona en varias rutas para que cada una sea factible, garantizando de esta manera el cumplimiento de las restricciones [31]. Dentro de los algoritmos que poseen este comportamiento se encuentran los Algoritmos de Pétalos [32].
* Asignar primero – Ruta después (*Cluster first – Route second*): se busca generar un grupo de clientes que pertenezcan a una misma ruta en la solución final y luego se crea una ruta para cada grupo que visite a todos sus clientes determinando el orden en que serán visitados [31]. Dentro de los algoritmos más conocidos se encuentra el Algoritmo de Barrido [28], el Algoritmo de Gillet-Miller [73] y la Heurística de Asignación Generalizada de Fisher y Jaikumar [24].

La mayoría de las heurísticas de construcción están diseñadas para el VRP más sencillo, es decir, el problema de planificación de rutas de vehículos con capacidades (CVRP). Por tanto, deben realizarse adaptaciones a estos métodos constructivos para aplicarlos en otras variantes. A continuación, se describen algunos de los algoritmos más significativos en la literatura de los problemas de planificación de rutas de vehículos:

* Heurística del Vecino más cercano con lista de candidatos restringidos (*Nearest Neighbor with RLC*): método determinista, pues se basa en la mejor elección en cada iteración. Esta variante incluye una lista de *k* vecinos más cercanos al cliente en cuestión. Consta de tres procesos: inicialización, inserción y finalización [31, 39].
* Heurística de Inserción Secuencial de Mole & Jameson: heurística de inserción que crea una solución mediante sucesivas inserciones de clientes en las rutas. En cada iteración se obtiene una solución parcial, donde las rutas visitan solamente un subconjunto de clientes y selecciona un cliente no visitado para insertar en dicha solución. Para ello, se siguen los pasos de creación de una ruta, inserción y optimización [30].
* Algoritmo de Ahorros (*Save Algorithm*): constituye el primer algoritmo efectivo para la resolución de un VRP. Este algoritmo plantea que en una solución dos rutas diferentes pueden ser combinadas formando una nueva ruta. Se parte de una solución inicial y se realizan las uniones que den mayores ahorros siempre que se cumplan las restricciones del problema. Existen dos versiones, la paralela que trabaja sobre todas las rutas de manera simultánea y la secuencial que construye las rutas una a una [7].
* Algoritmo de Barrido (*Sweep*): heurística basada en *Cluster first – Route second*, donde primero se forman grupos de clientes y luego se construyen las rutas para cada grupo. En este algoritmo los grupos se forman girando una semirrecta con origen en el depósito e incorporando clientes barridos por dicha semirrecta hasta que se viole la restricción de capacidad. Luego, los clientes se ordenan resolviendo un TSP en cada grupo. Esta heurística generalmente emplea coordenadas polares y distancia euclideana [28].

Además de las heurísticas de construcción antes mencionadas existen otras relevantes como: Algoritmo de Ahorros basado en *Matching* [74], la Heurística de Inserción en Paralelo de Christofides, Mingozzi y Toth (CMT) [75], la Heurística de Inserción de Kilby [62], la Heurística de Localización de Bramel y Simchi-Levi [76], la Heurística de Asignación Generalizada de Fisher y Jaikumar [24], entre otras [31]. En muchos casos también se utiliza regularmente un algoritmo de construcción aleatoria. Este algoritmo consiste en seleccionar en cada iteración un elemento al azar dependiendo de las características del problema; por ejemplo, lista de clientes, depósitos, entre otros [31].

## **1.4 Bibliotecas que implementan métodos heurísticos**

Para solucionar problemas de optimización resulta importante el uso de una biblioteca que implemente métodos heurísticos que garanticen la reutilización de los mismos. La mayor parte de las investigaciones se han centrado en el desarrollo de bibliotecas con algoritmos metaheurísticos, ya que son aplicables de manera general a cualquier tipo de problema de optimización. A continuación, se presentan las principales bibliotecas encontradas en la actualidad que implementan algoritmos metaheurísticos:

* BiCIAM: biblioteca de clases que implementa un modelo unificado de algoritmos metaheurísticos y desarrollada en el lenguaje de programación Java. Inicialmente se enfocaba en los algoritmos metaheurísticos basados en un punto, por ejemplo, Búsqueda Aleatoria [23], Escalador de Colinas [23], Búsqueda Tabú [71] y Recocido Simulado [55]. En la actualidad, se han incorporado nuevos algoritmos poblacionales como las Estrategias Evolutivas [23], Algoritmo Genético [23] y Algoritmo de Estimación de Distribuciones [23]. Además, se incorpora un portafolio de algoritmos con el fin de que los diferentes generadores metaheurísticos implementados colaboren y compitan entre sí, denominado Multigenerador. Por otra parte, se crea una versión con cambios en la arquitectura y estructuras de datos empleadas, e incorporando nuevas variantes de algoritmos como Escalador de Colinas con Reinicio [23] y Enjambre de Partículas [23]. Por último, la versión más reciente, une la versión mono-objetivo y multi-objetivo, las cuales se crearon en paralelo, y se añade el algoritmo multi-objetivo NSGAII [34].
* METSlib: biblioteca que implementa algoritmos metaheurísticos básicos de la literatura, tales como Búsqueda Local Aleatoria [23], Búsqueda con Vecindad Variable [23], Búsqueda Local Iterada [23], Recocido Simulado [55] y Búsqueda Tabú [71]. Esta biblioteca presenta como objetivo principal facilitar la implementación y adaptación de modelos, ya que una vez creados son aplicables a cualquier algoritmo implementado [35].
* MALLBA: biblioteca de clases desarrollada en el lenguaje de programación C++. Posee como objetivo fundamental proporcionar una biblioteca de estructuras algorítmicas para problemas de optimización combinatoria, que incluye métodos de resolución genéricos, exactos, heurísticos e híbridos. Para su diseño se tiene en cuenta la implementación secuencial y paralela de cada método de resolución [36].
* MOMHLib++: biblioteca de clases orientada a objetos, desarrollada en el lenguaje de programación C++. Su diseño busca la implementación de metaheurísticas multi-objetivos y permite la incorporación de nuevas metaheurísticas [39].
* *Open Metaheuristics*: biblioteca de clases desarrollada en los lenguajes de programación C/C++. Uno de sus propósitos es seguir una búsqueda de aprendizaje adaptada en el diseño de metaheurísticas. Por otra parte, permite realizar pruebas experimentales a los algoritmos metaheurísticos a través de aproximación estadística. El diseño del código está separado en tres componentes: algoritmos, problemas y comunicación. Por último, se enfoca en la simplicidad para asimilar el diseño e implementación de metaheurísticas creando una interfaz propia [39].
* *Operation Research Tools* (OR-Tools): software de código abierto, rápido y portable desarrollada por Google para solucionar problemas de optimización combinatoria y con soporte en varios lenguajes de programación como C++, C#, Java y Python. Resuelve diferentes variantes de VRP tales como CVRP, HFVRP, MDVRP, VRPTW y problemas de entregas y recogidas. Presenta algunas heurísticas de inserción dentro de las estrategias de solución, pero incluye mayormente metaheurísticas como Búsqueda Local [13], Recocido Simulado [55] y Búsqueda Tabú [71] [41].
* Jsprit: herramienta de código abierto desarrollada en el lenguaje de programación Java, que implementa el principio *Ruin and Recreate* [42],de manera modular y extensible. Su propósito general es resolver una amplia gama de variantes de VPR, incluyendo CVRP, MDVRP, VRPTW, VRPB, VRP con entregas y recogidas, HFVRP, TSP y *Dial a Ride Problem* (DARP). Además, emplea metaheurísticas como *Large Neighborhood Search* (LNS) [23] y variantes del Recocido Simulado [23] [43].
* OptaPlanner: marco de trabajo desarrollado en el lenguaje de programación Java, con el objetivo de configurar diferentes aspectos como las restricciones, condición de parada, heurísticas de construcción y algoritmos de búsqueda local, para obtener la solución. Además, brinda una configuración de *benchmark* para encontrar la mejor estrategia de búsqueda local. Dentro de las metaheurísticas que implementa se encuentran la Búsqueda Tabú [71] y el Recocido Simulado [55] [77].

Sin embargo, para la resolución de problemas de planificación de rutas de vehículos e incluso otros problemas de optimización, existen escasas bibliotecas que implementen algoritmos heurísticos. A continuación, se describen las bibliotecas encontradas en la literatura para este fin:

* Biblioteca de heurísticas de búsqueda local para el problema de ruteo de vehículos (VRPH): desarrollada en los lenguajes de programación C/C++ con un enfoque orientado a objetos. Dicha biblioteca emplea estructuras de datos eficientes para aplicar métodos que construyan soluciones iniciales factibles. También incorpora heurísticas de búsqueda local que pueden ser utilizadas para mejorar soluciones y aplica específicamente siete para generar soluciones a la variante CVRP [59].
* Biblioteca de código abierto para búsqueda heurística (Hipster): el software se desarrolla en código abierto aprobado por *Apache 2.0 License*. Surge a partir de la escasez de bibliotecas de búsqueda estándar en el lenguaje Java con un modelo extensible, flexible y sencillo de utilizar. Dentro de sus características fundamentales se encuentran la amplia gama de algoritmos de búsqueda clásicos implementados de manera iterativa como Dijsktra, A\*, IDA\*, AD\*, entre otros [39].
* *Vehicle Routing Open-source Optimization Machine* (VROOM): software de código abierto desarrollado en lenguaje de programación C++. Posee como objetivo fundamental resolver VRPs y otros problemas relacionados con logística y tareas distribuidas geográficamente. Además, emplea varias heurísticas para encontrar las soluciones iniciales en dependencia de la variante del problema, por ejemplo, una versión ajustada de la heurística de inserción de Christofides para TSP, heurísticas de agrupamiento usando árboles de expansión para CVRP, versiones ajustadas de la heurística de inserción de Solomon para CVRP y VRPTW, entre otras. Por último, emplea catorce operadores diferentes para la búsqueda local [44].
* Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos (BHCVRP): está desarrollada en el lenguaje de programación Java y presenta la adaptación de siete heurísticas clásicas de la literatura, como son el Algoritmo de Barrido, el Algoritmo de Ahorros en su versión paralela y secuencial, la Heurística del Vecino más Cercano con Lista de Candidatos Restringidos, la Heurística de Mole & Jameson, la Heurística de Inserción de Christofides, Mingozzi y Toth (CMT) y el método aleatorio, para cuatro variantes de VRP: CVRP, MDVRP, HFVRP y TTRP [39, 40].

A continuación, se realiza una comparación de las bibliotecas mencionadas anteriormente, teniendo en cuenta un conjunto de características. En la Tabla 1, se muestran los criterios establecidos para dicho estudio.

Tabla 1: Comparación de bibliotecas de clases que implementan métodos heurísticos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Características** | | | | | | |
| **Bibliotecas** | **Resuelve VRPs** | **Implementa HC** | **Implementa MH** | **Lenguaje de programación** | **Interoperable** | **Extensible** | **Libre y de código abierto** |
| BiCIAM | ✓ |  | ✓ | Java |  | ✓ | ✓ |
| METSlib | ✓ |  | ✓ | C++ |  | ✓ |  |
| MALLBA |  | ✓ |  | C++ |  |  |  |
| MOMHLib++ | ✓ |  | ✓ | C++ |  | ✓ |  |
| Open Metaheuristics | ✓ |  | ✓ | C/C++ |  | ✓ |  |
| OR-Tools | ✓ |  | ✓ | C++/C#/Java/Python | ✓ | ✓ | ✓ |
| Jsprit | ✓ |  | ✓ | Java |  | ✓ |  |
| OptaPlanner | ✓ |  | ✓ | Java |  | ✓ |  |
| VRPH | ✓ | ✓ |  | C/C++ |  |  |  |
| Hipster |  |  |  | Java |  | ✓ | ✓ |
| VROOM | ✓ | ✓ | ✓ | C++ |  | ✓ |  |
| BHCVRP | ✓ | ✓ |  | Java | ✓ | ✓ | ✓ |

Teniendo en cuenta el análisis comparativo sobre las bibliotecas presentadas en la Tabla 1, se puede arribar a las siguientes conclusiones:

* Las bibliotecas Hipster y MALLBA resuelven problemas de optimización diferentes a los VRP que son los de interés en esta investigación.
* Los componentes de software VRPH, BHCVRP, VROOM, OR-Tools y Jsprit poseen en común la característica de solucionar la variante CVRP. De manera general, entre estas se solucionan los casos de MDVRP, HFVRP y VRPTW.
* Las bibliotecas de metaheurísticas por la naturaleza de sus algoritmos pueden ser utilizadas en cualquier problema de optimización, ya que son de propósito general.
* La mayoría de las bibliotecas de clases implementan metaheurísticas.
* La única biblioteca interoperable es OR-Tools, aunque existe un servicio web de BHCVRP que se encuentra en desuso [39].
* Todas las bibliotecas de metaheurísticas son extensibles, mientras que las bibliotecas de heurísticas solo cuentan con Hipster y BHCVRP que extienden de manera parcial. Por tanto, se pueden agregar nuevos algoritmos de manera sencilla.
* De todas las bibliotecas solo BiCIAM, OR-Tools y BHCVRP son libres.
* Solo las bibliotecas de clases BHCVRP y VROOM implementan heurísticas de construcción para la resolución de variantes VRP.

Por último, resulta interesante analizar las tecnologías empleadas en el desarrollo de estos componentes de softwares que solucionan problemas de planificación de rutas de vehículos. A partir del estudio plasmado anteriormente, se evidencia que la mayoría de estos componentes se han desarrollado en los lenguajes de programación C++ y Java. Sin embargo, en los últimos tiempos la comunidad de Python ha crecido exponencialmente, sobre todo en trabajos relacionados con Inteligencia Artificial y problemas de optimización [45, 46]. A pesar de la existencia de OR-Tools en este lenguaje, se identifican deficiencias; por ejemplo, se aplican pocas heurísticas de construcción para solucionar VRP y las heurísticas existentes son deterministas, lo cual limita el espacio de búsqueda de nuevas soluciones.

## **1.5 Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos**

BHCVRP es una biblioteca de clases que implementa heurísticas de construcción para resolver problemas de planificación de rutas de vehículos. Esta biblioteca fue desarrollada en [39], en el lenguaje de programación Java y contiene la adaptación de siete heurísticas clásicas para las variantes CVRP, MDVRP, HFVRP y TTRP. Las heurísticas implementadas son: el Algoritmo de Barrido, las dos versiones del Algoritmo de Ahorros (Secuencial y Paralela), la Heurística del Vecino más Cercano con Lista de Candidatos Restringidos, la Heurística de Inserción de Mole & Jameson, la Heurística de Christofides, Mingozzi y Toth, y un método aleatorio [40].

### **1.5.1 Vista de la arquitectura de BHCVRP**

El enfoque utilizado en la estructuración en capas de BHCVRP está basado en reutilización [39], y se ilustra a continuación, en la Figura 5:

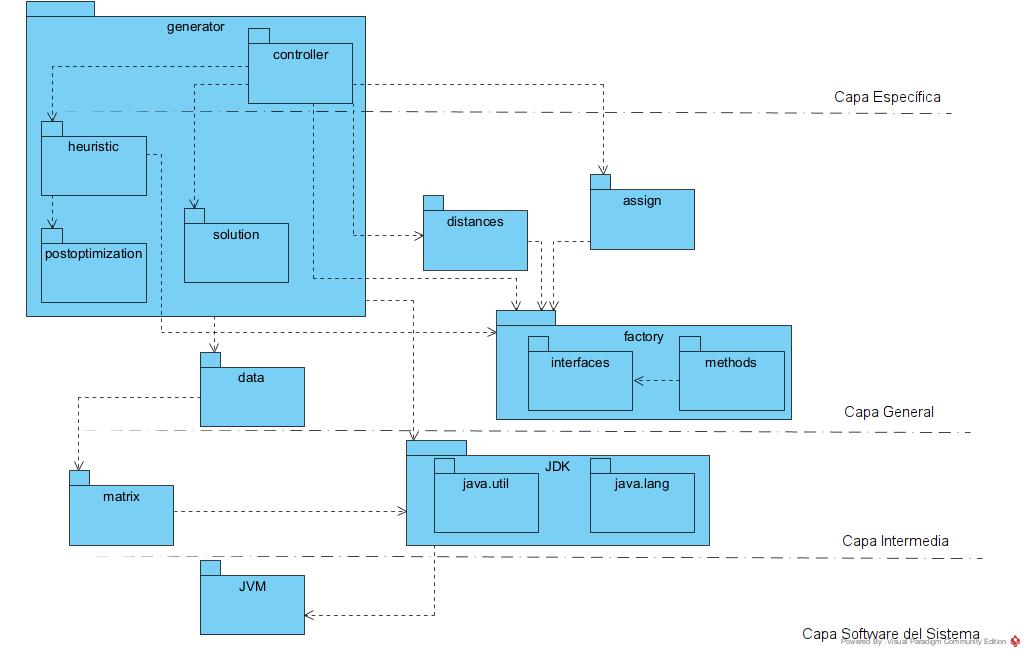


Figura 5: Patrón n-capas con enfoque basado en reutilización de BHCVRP [39].

La vista de la arquitectura se encuentra proyectada en cuatro capas: específica, general, intermedia y software del sistema. Los paquetes que conforman cada capa son:

1. Capa específica:

* *controller*: paquete responsable del control centralizado del sistema, donde se obtienen las posibles soluciones al problema y se aplica el patrón Singleton.

1. Capa general:

* *heuristic: paquete encargado de presentar todas las heurísticas disponibles para la construcción de rutas.*
* *assign: paquete encargado de la asignación de clientes a depósitos en la variante MDVRP.*
* *solution:* paquete encargado de modelar una solución y el conjunto de rutas en dependencia del tipo de VRP.
* *post\_optimization:* paquete que contiene métodos de post-optimización, necesarios para la obtención de soluciones en determinadas heurísticas.
* *distances*: paquete que contiene los distintos tipos de distancias con sus requerimientos.
* *data*: paquete responsable de modelar los datos del VRP en cuestión.
* *factory*: paquete que contiene las interfaces y métodos necesarios para la implementación del patrón de diseño *Factory Method* [78].

1. Capa intermedia:

* JDK:paquete que contienen clases, implementaciones y excepciones necesarias para el desarrollo del sistema a través de las librerías *java.util* y *java.lang* [79].
* *matrix*: paquete que contiene los métodos necesarios para el trabajo con matrices de costo.

1. Capa software del sistema:

* JVM: *Java Virtual Machine* constituye la comunicación entre las diferentes librerías utilizadas en la tecnología Java. Esta se realiza principalmente a través de llamadas de métodos y paso de objetos dentro del mismo proceso JVM.

### **1.5.2 Deficiencias en BHCVRP**

A partir del estudio y el análisis de la arquitectura y del código fuente de la biblioteca de clases BHCVRP, se han identificado un conjunto de deficiencias que no permiten realizar un uso eficiente de la misma. A continuación, se describen estas deficiencias y posibles mejoras:

* No presenta un paquete para tratar excepciones propias relacionadas con parámetros como la distancia, tiempo, capacidad, existencia o no de un depósito, cliente, vehículo, entre otros. Solamente se utilizan las que brinda el paquete *java.lang* propio del lenguaje de programación.
* A nivel de código se presentan ciertas deficiencias. Se debe mejorar la implementación de las heurísticas para propiciar la flexibilidad y reutilización del código.
* Por otra parte, los métodos *loadProblem* de la clase *StrategyHeuristic*, obligan a utilizar parámetros como el tipo de problema o tipo de cliente, y para que sea reutilizable debe estar acorde para todas las variantes VRP.
* El paquete *assign*, debe actualizarse por el nuevo componente BHAVRP [80] donde se identifica en la variante MDVRP posibles mejoras en la incorporación de nuevos métodos de asignación, así como admitir por parte del usuario una asignación predeterminada.
* Con el objetivo de proporcionar mayor flexibilidad a la biblioteca, se debe permitir al usuario el uso de los pasos de post-optimización a su conveniencia. Además, se debe incorporar nuevos métodos de post-optimización, ya que solo existe *ThreeOpt* aplicado en la Heurística de Inserción de Mole & Jameson y el Algoritmo de Ahorros en su versión paralela.
* Se deben incorporar nuevas variantes de VRP debido a la importancia que tienen en la vida práctica y en la literatura. Por ejemplo, *Open VRP,* VRP con ventanas de tiempo y Problemas de Planificación de Rutas de Autobuses Escolares.
* Además, la biblioteca debe ser capaz de adaptar nuevas heurísticas de construcción de la literatura como el Algoritmo de Pétalos, la Heurística de Asignación Generalizada de Fisher y Jaikumar, el Algoritmo de Gilett-Miller, entre otras.
* Como parte de las buenas prácticas que se siguen en el diseño de software, se necesitan aplicar patrones y principios de diseño para favorecer la flexibilidad y reutilización, ya que solo se aplican *Singleton* [78]y *Factory Method* [78].
* Existe una deficiencia arquitectónica, ya que la biblioteca no brinda servicio diferenciado para dos tipos de usuarios: uno experto con conocimientos de optimización con necesidades de configurar la mayor cantidad de parámetros posibles y otro usuario con conocimientos de su problema VRP real con una configuración predeterminada a partir de los mejores resultados alcanzados en experimentos. Actualmente, no se proporciona una configuración por defecto.
* A la biblioteca le falta interoperabilidad, lo cual implica que no se comunique e integre con otros componentes, plataformas o aplicaciones, intercambie datos, se utilice en diferentes entornos, siga con estándares de codificación, de la manera más efectiva posible. En primera instancia, sólo se pensó ser capaz de funcionar con la herramienta TransO [81] y el componente BHAVRP [80, 82]. Aunque existe una versión web en desarrollo [39].

## **1.6 Conclusiones parciales**

A partir del estudio realizado sobre el marco teórico investigativo de los problemas de optimización combinatoria, concretamente los Problemas de Planificación de Rutas de Vehículos, sus variantes y los algoritmos heurísticos que existen para resolver dicho problema, se obtienen las siguientes conclusiones:

* Los Problemas de Planificación de Rutas de Vehículos constituyen un problema de optimización combinatoria perteneciente a la categoría de problemas NP-duro y una alternativa de solución apropiada son los algoritmos heurísticos y metaheurísticos.
* A partir de la incorporación de nuevas características en la vida real, surgen numerosas variantes de Problemas de Planificación de Rutas de Vehículos.
* Para los Problemas de Planificación de Rutas de Vehículos existen heurísticas de construcción que obtienen buenas soluciones y pueden ser aplicadas como punto de partida en el contexto de las metaheurísticas.
* Los componentes de software han centrado mayormente su desarrollo en los algoritmos metaheurísticos.
* De los componentes de software estudiados, solo BiCIAM, VRPH, BHCVRP, JSprit, VROOM y OR-Tools solucionan Problemas de Planificación de Rutas de Vehículos. Sin embargo, solo Hipster y BHCVRP implementan heurísticas de construcción.
* Debido a la relevancia del lenguaje de programación Python para los trabajos relacionados con Inteligencia Artificial, es interesante explorar sus potencialidades. Además, de los componentes estudiados previamente, solo OR-Tools se encuentra desarrollado en Python y resuelve VRPs.
* La Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos, solo es capaz de resolver las variantes CVRP, MDVRP, HFVRP y TTRP, con siete heurísticas de construcción.
* La Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos, presenta deficiencias en cuanto a la arquitectura y código fuente. Principalmente sus debilidades se encuentran en la implementación de las heurísticas de construcción y la ausencia de excepciones propias. Por último, las mejoras deben enfocarse en la aplicación de métodos de post-optimización y la incorporación de nuevas variantes VRP y heurísticas de construcción.

# **Capítulo 2: Diseño de la solución**

## **2.1 Introducción**

La solución que se presenta en este capítulo consiste en una actualización de la biblioteca de clases BHCVRP. La nueva versión parte de un conjunto de cambios introducidos en la arquitectura para garantizar la extensibilidad del componente. Como parte de las transformaciones relevantes mencionar que se propone la primera versión de la biblioteca que cubre dos lenguajes de programación: Python y Java. Además, se incorporan dos nuevas heurísticas de construcción: Heurística de Inserción de Kilby y el Algoritmo de Ahorros basado en *Matching*. Para describir la solución se muestra una vista de la arquitectura y los diagramas de clases más significativos. Por último, se presentan los patrones y principios de diseño utilizados.

## **2.2 Modificaciones en la arquitectura de BHCVRP**

La Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos (BHCVRP) es una propuesta del proyecto de Optimización y Metaheurísticas, de la Facultad de Ingeniería Informática de la CUJAE en el año 2016 [39, 40]. Esta biblioteca está desarrollada en el lenguaje de programación Java y su objetivo es utilizar heurísticas de construcción para solucionar diferentes variantes VRP. Con este fin la biblioteca cuenta con siete heurísticas de construcción para las variantes CVRP, MDVRP, HFVRP y TTRP: la Heurística del Vecino más Cercano con Lista de Candidatos Restringidos, el Algoritmo de Barrido, el Algoritmo de Ahorros en sus versiones secuencial y paralela, la Heurística de Inserción de Mole & Jameson, la Heurística de Inserción en Paralelo de Christofides, Mingozzi y Toth (CMT) y un método aleatorio.

A partir del análisis realizado en la sección 1.5, no solo se encuentran un conjunto de deficiencias, sino también oportunidades para mejorar su funcionamiento. Estas modificaciones se listan a continuación. Es importante destacar que algunas quedan para el trabajo de diploma.

* Rediseñar las clases para la incorporación de nuevas heurísticas de construcción y diferentes variantes VRP.
* Incorporar la versión restante del Algoritmo de Ahorros, es decir, su variante basada en *Matching*.
* Incorporar la heurística de inserción de Kilby.
* Incorporar otros métodos de post-optimización a través de los operadores 2-opt, *Relocate, Exchange*, entre otros.
* Ofrecer un paquete de excepciones propias para el tratamiento de restricciones relacionadas con la distancia, capacidad, entre otras.
* Adaptar el patrón de diseño *Template* para mejorar la reutilización del código y la flexibilidad de las heurísticas implementadas.
* Convertir el paquete *distance* en un .jar para que pueda ser empleado por cualquier usuario que necesite trabajar con cálculo de distancias.
* La incorporación de BHAVRP para tratar los métodos de asignación de clientes a depósitos en la variante MDVRP.
* Utilizar un formato XML para la carga de datos.

### **2.2.1 Nuevas heurísticas de construcción en BHCVRP**

Como parte de la extensión de BHCVRP se incorporan dos nuevas heurísticas de construcción: la Heurística de Inserción de Kilby y el Algoritmo de Ahorros basado en *Matching*. A continuación, se describe el funcionamiento de cada una de estas heurísticas constructivas, así como las modificaciones necesarias para su incorporación.

***2.2.1.1 Heurística de Inserción de Kilby (Kilby Algorithm)***

El algoritmo de Kilby, similar a la propuesta por Mole & Jameson [30], constituye una heurística de inserción donde se crea una solución a través de inserciones sucesivas de clientes en las rutas. Este método se centra en minimizar los costos al asignar clientes a los itinerarios de los vehículos. Comienza con un enfoque voraz, donde se agrega el cliente más cercano en cada momento. Luego, se aplican métodos de optimización para mejorar las rutas de cada vehículo [83, 84].

El costo de inserción de un cliente en , siendo el conjunto de rutas actuales, se calcula a partir de la siguiente fórmula: . Además, en el mecanismo de inserción, en cada paso se inserta el cliente no visitado que minimice el costo total . Por último, la heurística de inserción de Kilby utiliza como método de post-optimización el algoritmo 3-opt [23]. En un futuro se pueden aplicar otros métodos como son: el algoritmo 2-opt [26], el operador *Relocate* [23] y el operador *Exchange* [23]. El objetivo de estos métodos es mejorar el costo de la solución obtenida [84].

Según [84], el algoritmo consta de tres pasos:

1. Inicialización: se crea la cantidad de rutas correspondientes con la cantidad de vehículos de la flota. Se calcula a cada cliente el costo dado por la fórmula planteada anteriormente, colocando los primeros en cada ruta.
2. Inserción: calcular . Insertar al cliente en la asociada a él. Si quedan rutas y clientes por analizar ir al paso 2.
3. Optimización: si ya está completa o no quedan clientes por analizar, aplicar los pasos de post-optimización.

Para la incorporación de esta heurística se realizan algunas modificaciones que se mencionan a continuación:

* Se implementa una clase que modela el comportamiento de la heurística para las cuatro variantes de VRP existentes (ver Figura 13).
* Se adiciona la heurística de inserción de Kilbycomo tipo de heurística que puede ser empleada para resolver VRP en la biblioteca (ver Figura 12).
* Se incorporan tres clases para implementar el algoritmo 2-opt, el operador *Relocate* y el operador *Exchange*, respectivamente, utilizados como paso de post-optimización en esta heurística.(ver Figura 9)*.*

***2.2.1.2 Algoritmo de Ahorros basado en Matching (Matching-based Saving Algorithm)***

A partir del Algoritmo de Ahorros [7], surge la variante basada en *Matching* [74]*.* Esto se debe a la decisión de unir dos rutas y , donde se descartan otras posibles uniones, ya que y dejan de ser extremos en la nueva ruta. Elegir siempre el máximo ahorro es una estrategia demasiado voraz e implica, en algunos casos, que las uniones que permanecen factibles no sean buenas. En el Algoritmo de Ahorros basado en *Matching*, se considera la afectación que provoca la unión a realizar, a las posibles uniones en próximas iteraciones. Para esto, se considera un conjunto de ahorros potenciales entre las rutas, donde cada ahorro representa la reducción en costos al combinar dos rutas en una sola. Luego, se utiliza un *matching* de peso máximo para seleccionar las combinaciones de rutas que maximizan los ahorros totales, garantizando una solución globalmente buena [31].

Hallar un *matching* de peso máximo puede resolverse en tiempo polinomial, en dependencia de la cantidad de nodos. Sin embargo, para problemas grandes se recomienda hallar el *matching* mediante alguna heurística. Según [85] y [86], al combinar dos rutas se considera la posibilidad de insertar una al final de la otra como en el Algoritmo de Ahorros original [7] y todas las posibilidades de armar una nueva ruta con los clientes de ambas. En las uniones dadas por el *matching* de peso máximo, una misma ruta no participa en más de una unión. Sin embargo, podría ser ventajoso que la ruta resultante de una unión sea inmediatamente unida con otra. Por tanto, surgen dos alternativas: combinar solo las rutas que correspondan al arco de mayor peso del *matching* o conectar los nodos donde se calcula el *matching* con nodos ficticios de pesos tales que prioricen la inclusión de algunos de los ficticios en el *matching*. De esta última manera, se garantiza que no todos los arcos del *matching* correspondan con uniones de rutas [31]. Por último, en [87] se propone modificar el ahorro para privilegiar las uniones que están lejos de ser no factibles. Cuando no hay más combinaciones de rutas que sean factibles, algunas rutas son divididas a través de probabilidades, permitiendo continuar con la búsqueda de soluciones [31].

Según [31], este algoritmo consta de cuatro pasos:

1. Inicialización: para cada cliente construir la ruta .
2. Cálculo de ahorros: actualizar para cada par de rutas y que pueda ser combinado manteniendo la factibilidad. Si ningún par de rutas puede ser combinado, terminar.
3. *Matching*: resolver un problema de *matching* de peso máximo donde se tienen las rutas de la solución actual como nodos y existe un arco entre las rutas y con peso si su combinación es factible.
4. Uniones: dado el *matching* de peso máximo, combinar todo par de rutas y tal que pertenezca al *matching*. Ir al paso 2.

Para la incorporación de esta heurística se realizan algunas modificaciones que se mencionan a continuación:

* Se implementa una clase que modela el comportamiento de la heurística para las cuatro variantes de VRP existentes (ver Figura 13).
* Se adiciona el Algoritmo de Ahorros basado en *Matching* como tipo de heurística que puede ser empleada para resolver VRP en la biblioteca (ver Figura 12).
* Se incorporan métodos auxiliares que implementa el problema de *matching* de peso máximo en el proceso de construcción de rutas del Algoritmo de Ahorros basado en *Matching* (ver Figura 13)*.*

## **2.3 Nueva versión de BHCVRP**

Como resultado de las modificaciones realizadas a la biblioteca, se muestran en las Figura 6, el diseño arquitectónico de la nueva versión de BHCVRP para los lenguajes de programación Java y Python. Se utilizan colores diferentes para reflejar los cambios introducidos. En el caso de los paquetes que son nuevos en la arquitectura se emplea el color verde, el color morado para aquellos paquetes que sufrieron algún tipo de cambio y el color azul para los que se mantienen.

En primera instancia, la arquitectura de BHCVRP se basa en el patrón n- capas con un enfoque basado en reutilización [88]. A continuación, en la Figura 6, se muestra el diagrama de estructuración en capas para la nueva versión.

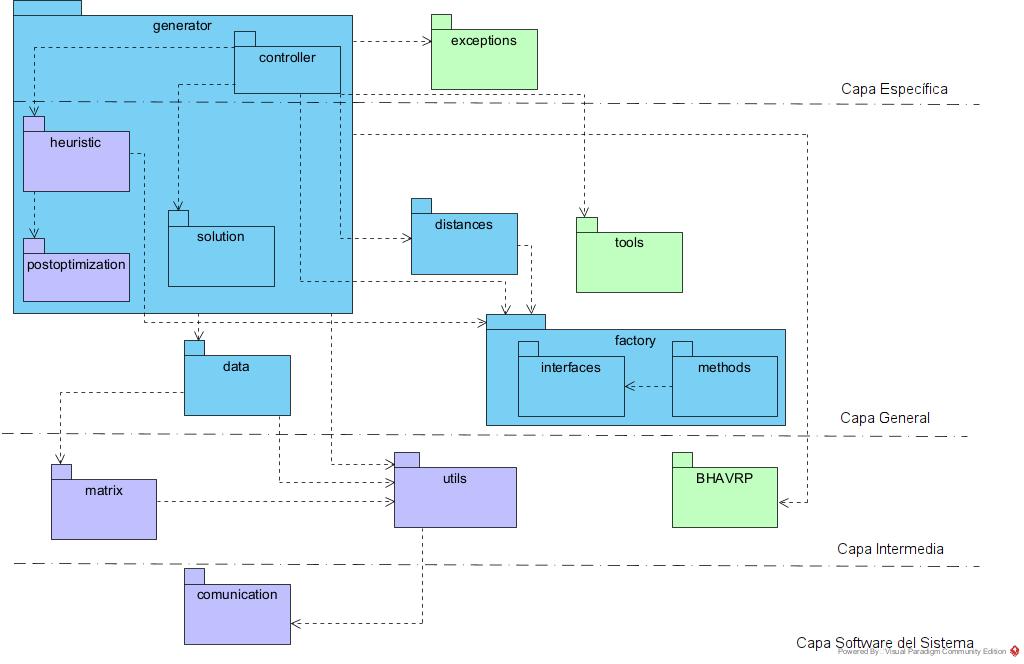


Figura 6: Patrón n-capas con enfoque basado en reutilización de la nueva versión de BHCVRP.

A continuación, se explican las funciones de los paquetes que contiene cada capa para las versiones implementadas en Java y Python.

### **2.3.1 Particularidades de BHCVRP versión Java**

En la capa intermedia se presentan los siguientes paquetes:

* *matrix*: se emplea la librería *libmatrix* que contiene las clases necesarias para modelar la matriz de costo.
* *utils*: paquete relacionado con la tecnología Java, que contienen clases, implementaciones y excepciones necesarias para el desarrollo del sistema a través de las librerías *java.util* y *java.lang.*
* *BHAVRP*: se corresponde con la Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos (BHAVRP) [80]. Su objetivo es manejar los métodos de asignación de clientes a depósitos.

En la capa software del sistema se presenta el siguiente paquete:

* *comunication*: *Java Virtual Machine* constituye la comunicación entre las diferentes librerías utilizadas en la tecnología Java. Esta se realiza principalmente a través de llamadas de métodos y paso de objetos dentro del mismo proceso JVM.

### **2.3.2 Particularidades de BHCVRP versión Python**

En la capa intermedia se presentan los siguientes paquetes:

* *matrix*: se emplea la librería *numpy* que contiene los métodos necesarios para el trabajo con matrices. En este caso, se aplicaría para la construcción de matrices de costo.
* *utils*: paquete relacionado con la tecnología Python, que contienen clases, implementaciones y excepciones necesarias para el desarrollo del sistema a través de las librerías *typing, time, tqdm* y *sys.*
* *BHAVRP*: se utiliza la Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos (BHAVRP) [80] desarrollada en el lenguaje Java, a través de la librería *jpype* existente en Python.

En la capa software del sistema se presenta el siguiente paquete:

* *comunication*: en Python se utiliza el protocolo de comunicación *buffer.* Este consiste en tener capacidad para manejar grandes volúmenes de datos y su flexibilidad de uso.

### **2.3.3 Paquetes y funcionalidades en común**

La capa general está compuesta por paquetes necesarios para el funcionamiento de las heurísticas y la modelación de los datos del problema de planificación de rutas de vehículos que se quiere 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.

El diagrama de clases que se muestra en la Figura 7 refleja las clases del paquete *data.* Su objetivo es contener las clases que modelan un problema de planificación de rutas de vehículos.

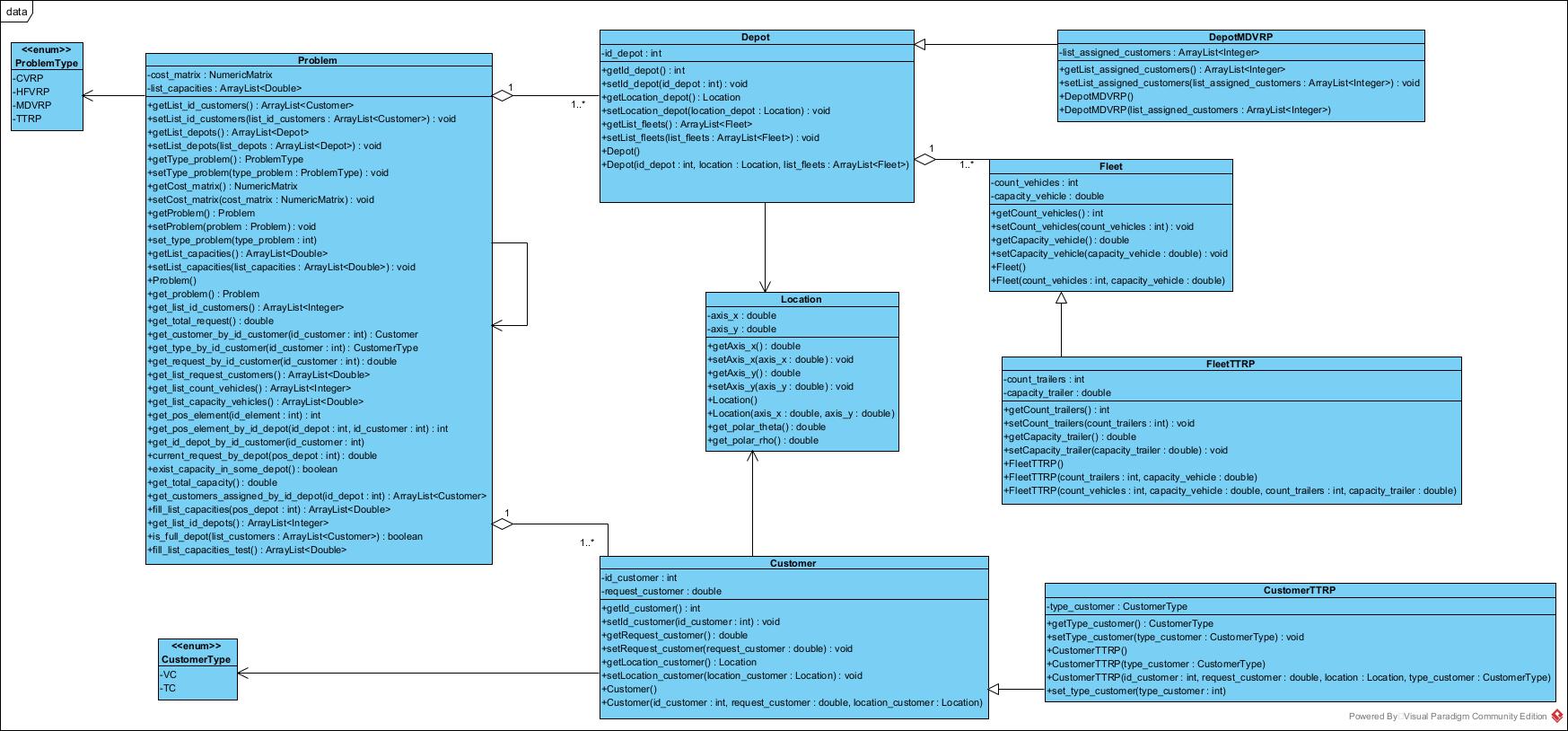


Figura 7: Diagrama de clases del paquete *data*.

En la Tabla 2 se describen las clases que contienen el paquete *data.*

Tabla 2: Descripción de las clases que integran el paquete *data*.

|  |  |
| --- | --- |
| Clases | Descripción |
| *ProblemType* | Enumerado que indica las variantes de problema de planificación de rutas de vehículos que pueden ser resueltas con BHCVRP. |
| *Customer* | Clase que modela los datos de un cliente. |
| *CustomerTTRP* | Clase que modela los datos de un cliente para la variante TTRP. |
| *CustomerType* | Enumerado que indica los tipos de clientes cuando la variante de planificación de rutas de vehículos es TTRP. |
| *Depot* | Clase que modela los datos de un depósito. |
| *DepotMDVRP* | Clase que modela los datos de un depósito para la variante MDVRP. |
| *Fleet* | Clase que modela los datos de una flota de vehículos. |
| *FleetTTRP* | Clase que modela los datos de una flota de vehículos para la variante TTRP. |
| *Location* | Clase que modela la ubicación geográfica de un cliente o de un depósito. |
| *Problem* | Clase que controla el problema a solucionar. Contiene los clientes, los depósitos, la matriz de costo y el tipo de problema. |

El diagrama de clases que se muestra en la Figura 8 refleja las clases del paquete *solution.* Su objetivo es contener las clases que modelan una solución del problema de planificación de rutas de vehículos.

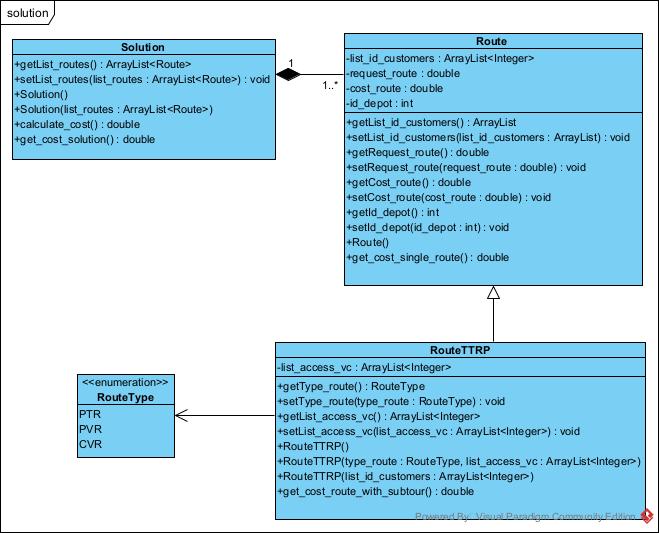


Figura 8: Diagrama de clases del paquete *solution*.

En la Tabla 3 se describen las clases que contienen el paquete *solution.*

Tabla 3: Descripción de las clases que integran el paquete *solution*.

|  |  |
| --- | --- |
| Clases | Descripción |
| *Route* | Clase que modela los datos de una ruta. |
| *RouteTTRP* | Clase que modela los datos de una ruta para la variante TTRP. |
| *RouteType* | Enumerado que indica el tipo de ruta para el caso de la variante TTRP. |
| *Solution* | Clase que modela la solución que se obtiene con una heurística de construcción, es decir, el orden en que serán visitados los clientes y el costo en distancia de los recorridos. |

El diagrama de clases que se muestra en la Figura 9 refleja las clases del paquete *postoptimization.* Su objetivo es contener los diferentes métodos para el uso de post-optimización. De color azul se encuentra la clase existente, de color morado se encuentra *StepOptimization* que se convierte en interfaz y de color verde las nuevas clases que se corresponden con los métodos de post-optimización a incorporar.

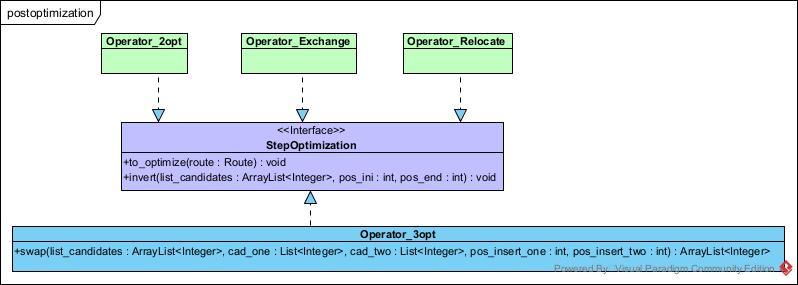


Figura 9: Diagrama de clases del paquete *postoptimization*.

En la Tabla 4 se describen las clases que contienen el paquete *postoptimization.*

Tabla 4: Descripción de las clases que integran el paquete *postoptimization*.

|  |  |
| --- | --- |
| Clases | Descripción |
| *StepOptimization* | Interfaz que contiene los métodos de uso común para los métodos de postoptimización. |
| *Operator\_3opt* | Clase que modela el operador 3opt para intercambiar tres nodos no adyacentes de una ruta. |
| *Operator\_2opt* | Clase que modela el operador 2opt para intercambiar dos nodos no adyacentes de una ruta. |
| *Operator\_Exchange* | Clase que modela el operador *exchange* para intercambiar dos nodos de la ruta. |
| *Operator\_Relocate* | Clase que modela el operador *relocate* para mover un nodo a una nueva posición en la ruta. |

El diagrama de clases que se muestra en la Figura 10 refleja las clases del paquete *tools.* Su objetivo es contener diferentes clases auxiliares, por ejemplo, para el ordenamiento, la carga de ficheros, entre otros. En color azul se representa una clase que no presenta cambios y en color verde las incorporadas recientemente.

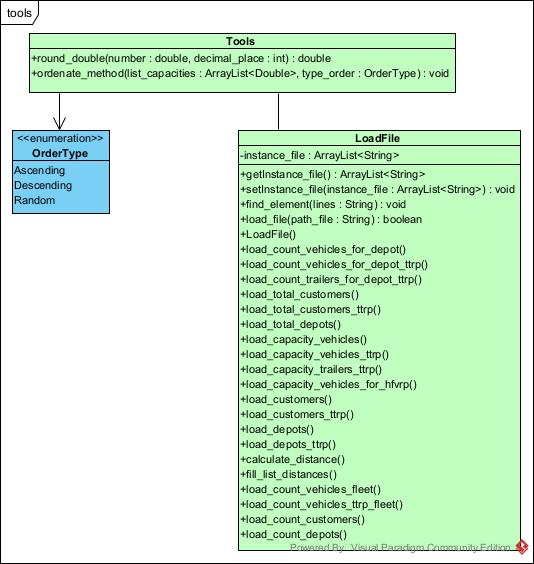


Figura 10: Diagrama de clases del paquete *tools*.

En la Tabla 5 se describen las clases que contienen el paquete *tools.*

Tabla 5: Descripción de las clases que integran el paquete *tools*.

|  |  |
| --- | --- |
| Clases | Descripción |
| *OrderType* | Enumerado que contiene los distintos tipos de ordenamiento de la lista de capacidades para la variante HFVRP. |
| *LoadFile* | Clase que permite la carga de ficheros de la literatura. |
| *Tools* | Clase auxiliar para realizar operaciones como redondeo y ordenamiento. |

El diagrama de clases que se muestra en la Figura 11 refleja las clases del paquete *distance.* En color morado se presenta la interfaz *Distance* y en azul las clases que la implementan. Su objetivo es contener las clases que modelan los cálculos de los distintos tipos de distancia.

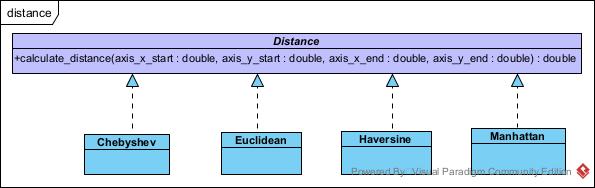


Figura 11: Diagrama de clases del paquete *distance*.

En la Tabla 6 se describen las clases que contienen el paquete *distance.*

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

|  |  |
| --- | --- |
| Clases | Descripción |
| *Distance* | Interfaz que define como calcular la distancia entre dos puntos. Además, posee cuatro clases que usan dicha interfaz, para implementar sus respectivos tipos de distancias: *Euclidean, Manhattan, Chebyshev* y *Haversine*. |

El paquete *factory* está compuesto por los siguientes paquetes:

* *interfaces:* paquete que contiene las interfaces y los enumerados para la creación de heurísticas de construcción, métodos para el cálculo de distancias entre dos puntos y métodos de post-optimización.
* *methods:* paquete que contiene las clases que implementan los métodos declarados en las interfaces del paquete *interfaces*.

La Figura 12 muestra el diagrama de clases con la relación que se establece entre los elementos de ambos paquetes. En color azul se encuentran las clases que permanecen igual, en color morado las que sufren cambios y en verde las nuevas que se incorporan.

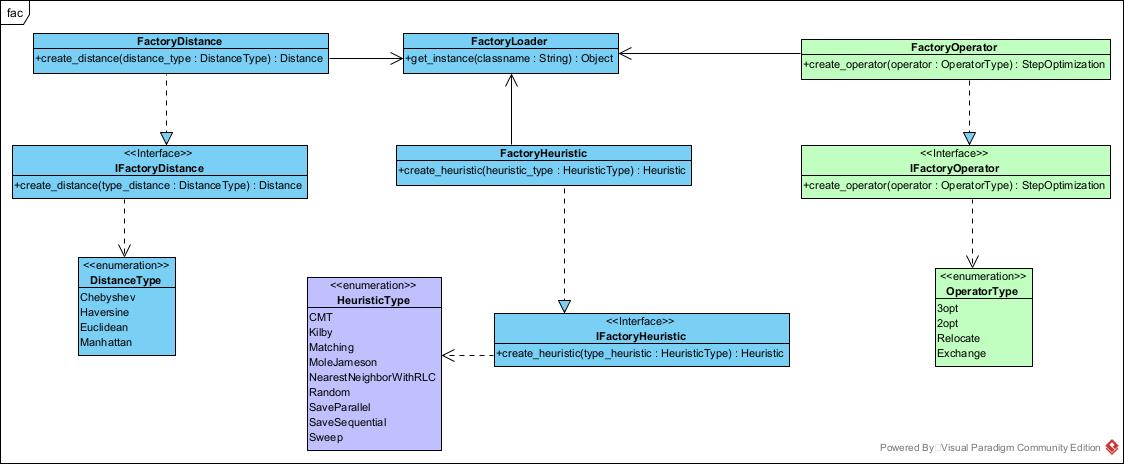


Figura 12: Diagrama de clases del paquete *factory*.

En las Tabla 7 y Tabla 8 se describen los elementos participantes en el artefacto antes mencionado.

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

|  |  |
| --- | --- |
| Clases | Descripción |
| *IFactoryDistance* | Interfaz que declara los métodos para la creación de métodos para el cálculo de distancias entre dos puntos necesarios para la conformación de la matriz de costo. |
| *IFactoryHeuristic* | Interfaz que declara los métodos para la creación de heurísticas de construcción. |
| *IFactoryOperator* | Interfaz que declara los métodos para la creación de métodos de post-optimización. |
| *DistanceType* | Enumerado que indican los tipos de métodos para calcular distancia entre dos puntos en la biblioteca para el cálculo de la matriz de costo. |
| *HeuristicType* | Enumerado que indican los tipos de heurísticas de construcción para resolver problemas de planificación de rutas de vehículos en la biblioteca. |
| *OperatorType* | Enumerado que indican los tipos de métodos de post-optimización a utilizar en determinadas heurísticas de construcción en la biblioteca. |

Tabla 8: Descripción de las clases que integran el paquete *methods*.

|  |  |
| --- | --- |
| Clases | Descripción |
| *FactoryDistance* | Clase que implementa la interfaz *IFactoryDistance* y construye un método para el cálculo de la distancia entre dos puntos para conformar la matriz de costo. |
| *FactoryHeuristic* | Clase que implementa la interfaz *IFactoryHeuristic* y construye una heurística de construcción para resolver problemas planificación de rutas de vehículos. |
| *FactoryOperator* | Clase que implementa la interfaz *IFactoryOperator* y construye un método para usar pasos de psot-optimización en problemas planificación de rutas de vehículos. |
| *FactoryLoader* | Clase que implementa un método genérico para la carga dinámica de cualquier objeto. |

El paquete *heuristic* se encarga de presentar todas las heurísticas disponibles para la construcción de rutas. A continuación, en la Figura 13, se muestra su diagrama de clases correspondiente. En color azul se muestran las clases que permanecen intactas, en color morado las que sufren modificaciones producto del patrón de diseño *Template* y en color verde las heurísticas nuevas incorporadas.

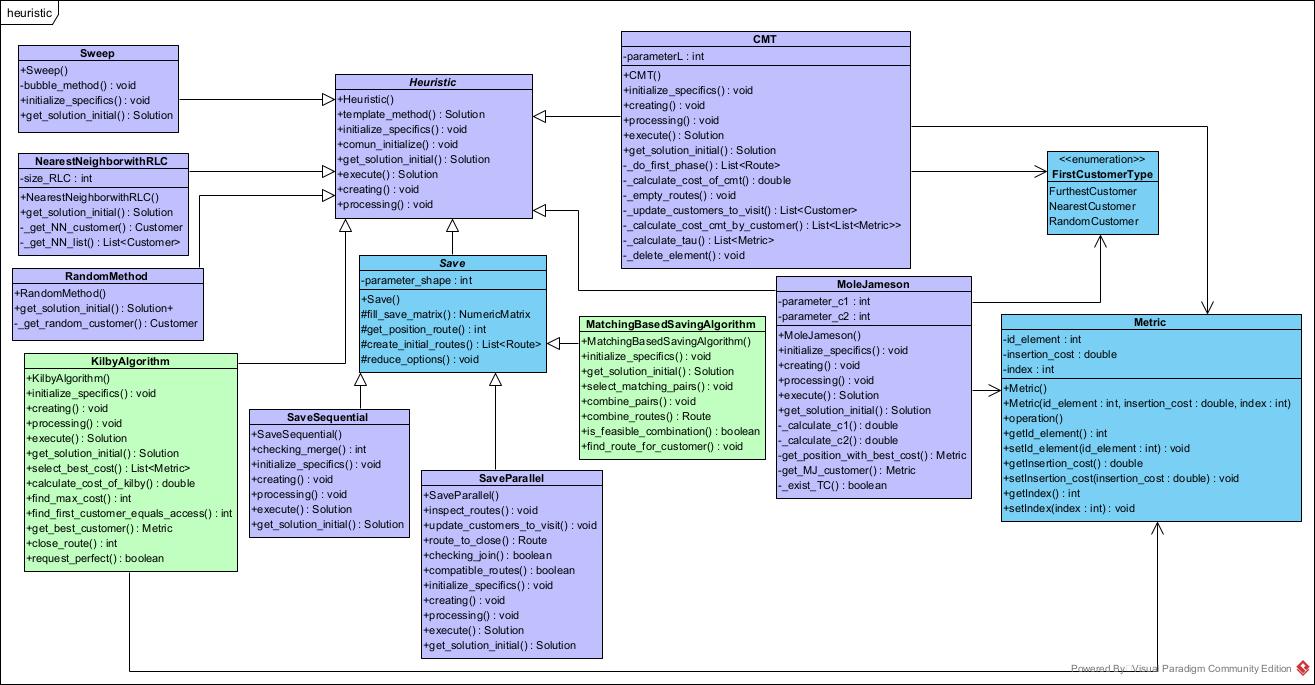


Figura 13: Diagrama de clases del paquete *heuristic*.

En la Tabla 9 se describen las clases que contienen el paquete *heuristic.*

Tabla 9: Descripción de las clases que integran el paquete *heuristic*.

|  |  |
| --- | --- |
| Clases | Descripción |
| *Heuristic* | Clase abstracta que define el comportamiento de una heurística de construcción. De aquí heredan las seis de las heurísticas implementadas: *CMT, KilbyAlgorithm, MoleJameson, NearestNeighborWithRLC, RandomMethod* y *Sweep.* |
| *Save* | Clase abstracta que hereda de *Heuristic,* que modela el Algoritmo de Ahorros. De aquí herendan las heurísticas *SaveParallel, SaveSequential* y *MatchingBasedSavingAlgorithm.* |
| *FirstCustomerType* | Enumerado que indica la forma de seleccionar el primer cliente en las heurísticas *MoleJameson* y *CMT*. |
| *Metric* | Clase auxiliar utilizada para almacenar la información de los costos de inserción en las heurísticas de inserción. |

Por último, en la capa específica se encuentran los siguientes paquetes:

* *exceptions*: paquete encargado de manejar excepciones propias del contexto VRP, tales como capacidad, distancia, tiempo, costo, entre otros.
* *controller*: paquete responsable del control centralizado del sistema, donde se obtienen las posibles soluciones al problema y se aplica el patrón *Singleton* (ver Figura 15).

El diagrama de clases que se muestra en la Figura 14 refleja las clases del paquete *exceptions* y se utiliza el color verde para resaltar que es nuevo en su totalidad*.*

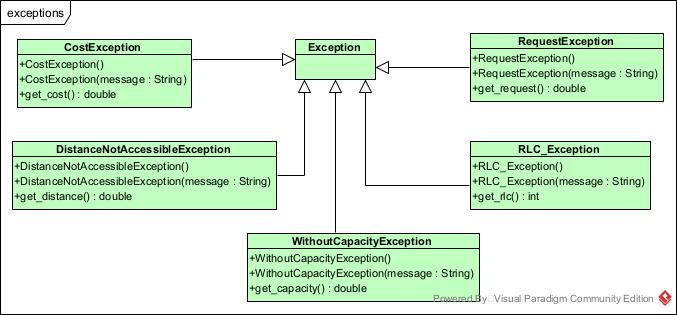


Figura 14: Diagrama de clases del paquete *exceptions*.

En la Tabla 10 se describen las clases que contienen el paquete *exceptions.*

Tabla 10: Descripción de las clases que integran el paquete exceptions.

|  |  |
| --- | --- |
| Clases | Descripción |
| *CostException* | Clase que maneja la excepción relacionada con el costo, que debe ser mayor que cero. |
| *DistanceNotAccessibleException* | Clase que maneja la excepción relacionada con un límite de distancia a recorrer por determinado vehículo. |
| *RequestException* | Clase que maneja la excepción relacionada con la demanda, que debe ser mayor que cero. |
| *RLC\_Exception* | Clase que maneja la excepción relacionada con la heurística del Vecino más Cercano con Lista de Candidatos Restringidos, donde la lista no puede ser mayor que la mitad de la cantidad de clientes. |
| *WithoutCapacityException* | Clase que maneja la excepción relacionada con la capacidad de un vehículo. |

En la Tabla 11 se describe la clase que contiene el paquete *controller.*

Tabla 11: Descripción de las clases que integran el paquete *controller*.

|  |  |
| --- | --- |
| Clases | Descripción |
| *StrategyHeuristic* | Clase que controla el proceso de construcción de soluciones a los problemas de planificación de rutas de vehículos en la biblioteca. |

## **2.4 Principios de diseño**

Los principios de diseño de software son fundamentales para construir sistemas que sean robustos, escalables y fáciles de mantener. Estos principios actúan como cimientos en la arquitectura de software, guiando a los desarrolladores en la toma de decisiones de diseño que resultan en un software eficiente y confiable [89].

A continuación, se describen los principios de diseño aplicados en BHCVRP:

* Encapsulación de la variabilidad: implica separar las partes del sistema que pueden cambiar de las que deben permanecer inalteradas. Por tanto, se protegen las partes estables y se facilitan futuras modificaciones en las partes variables sin afectar al resto del sistema [90]. Esto se puede evidenciar en las heurísticas, distancias y pasos de post-optimización, donde el código que maneja estos paquetes está separado en clases específicas.
* Hollywood: sugiere que, al diseñar un programa, las llamadas para interactuar deben iniciarse desde las partes de alto nivel hacia las partes de bajo nivel. En otras palabras, las piezas principales actúan como coordinadoras, y no al revés. Esto tiene un efecto beneficioso, ya que evita la creación de dependencias complicadas entre las diferentes partes del sistema [91]. En el contexto de las heurísticas y las distancias, cuando se va a crear una de estas, se permite que ellas mismas decidan cuando crear las instancias que necesiten, en vez de crear todos los tipos de heurísticas y distancias.
* Cohesión y acoplamiento: la cohesión se refiere al grado de relación existente entre los elementos de una clase y el acoplamiento mide la dependencia de una clase respecto a otros elementos externos. Para mantener un buen diseño de software se busca obtener alta cohesión y bajo acoplamiento. Esto garantiza que los métodos estén estrechamente relacionados mediante llamadas entre ellos o utilizando variables con ámbito a nivel de clase, por tanto, dicha clase realiza una única tarea. Además, minimiza el empleo de componentes externos, mejorando la encapsulación [78, 92]. En BHCVRP, este principio se evidencia, por ejemplo, en los paquetes *data* y *heuristic* (ver Figura 7 y Figura 13)*.*
* *Don´t Repeat Yourself*: se centra en evitar la duplicación innecesaria de información o código, promoviendo la eficiencia y la mantenibilidad del sistema. La idea principal detrás de DRY es que toda pieza de conocimiento debe tener una única, precisa y completa representación dentro de un sistema. En BHCVRP, se evidencia en la clase *Heuristic* que brinda una implementación reutilizable para las clases hijas *RandomMethod, Sweep* y *NearestNeighborWithRLC* (ver Figura 13)*.*
* *Open/Close*:plantea que los módulos del software deben ser abiertos para su extensión, pero cerrados para su modificación [89]. En BHCVRP se sigue este principio con el objetivo de que las clases que la conforman puedan extenderse cuando sea necesario, pero sin modificar el código fuente de las mismas. Para conseguir esto se utiliza el polimorfismo tal y como se muestra en el diagrama de clases del paquete *heuristic* (ver Figura 13)*,* donde *Heuristic* constituye una clase abstracta encargada de modelar la heurística deseada. A partir del método *getSolutionInitial()*, se obtiene una solución que representa el conjunto de rutas factibles.

## **2.5 Patrones de diseño**

Durante el proceso de diseño orientado a objetos, es común encontrarse con ciertos tipos de problemas de manera recurrente. Para analizar, compartir y documentar el conocimiento sobre estos problemas, se han desarrollado los patrones de diseño. Estos patrones son modelos formales aplicables a diferentes dominios y nos ayudan a seguir pautas comunes en la solución de problemas similares en su estructura [78, 93]. En esencia, los patrones de diseño nos permiten identificar problemas concretos junto con sus soluciones. Son una forma de documentar la experiencia acumulada en las estrategias empleadas para resolver problemas específicos dentro de un contexto determinado [94].

En el ámbito del diseño de software, los patrones de diseño *Gang of Four* (GoF, por sus siglas en inglés) son esenciales. Estos patrones, descritos en [93], se han convertido en herramientas indispensables para los programadores. Los patrones GoF se dividen en tres categorías según su propósito:

* Patrones estructurales: permiten crear grupos de objetos para abordar tareas complejas.
* Patrones de comportamiento: definen la comunicación entre los objetos de un sistema y el flujo de información entre ellos.
* Patrones creacionales: facilitan la creación de objetos sin necesidad de instanciarlos directamente, lo que brinda a los programas mayor flexibilidad para decidir qué objetos utilizar.

En la versión anterior de la biblioteca de clases se garantizó la flexibilidad y reusabilidad del diseño con la utilización de patrones. Por lo que en esta nueva versión se decidió reutilizar la implementación de los patrones creacionales *Factory Method* y *Singleton* [93-95]. Es importante destacar que los patrones de diseño que se proponen emplear en los siguientes sub-epígrafes son comunes para BHCVRP en ambos lenguajes de programación, Java y Python.

### **2.5.1 Patrón Singleton**

El patrón Singleton está diseñado para restringir la creación de objetos pertenecientes a una clase a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. En otras palabras, provee un mecanismo para limitar el número de instancias de una clase, por lo que el mismo objeto es siempre compartido por distintas partes del código [93-95].

Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase controla el acceso a un recurso físico único o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación.

En la nueva versión de la biblioteca se hizo necesario el uso de este patrón en la clase *StrategyHeuristic*, que es la clase controladora, para garantizar los aspectos antes mencionados. En la Figura 7 y Figura 15, se puede apreciar la implementación de este patrón en BHCVRP en las clases *Problem* y *StrategyHeuristic* respectivamente.

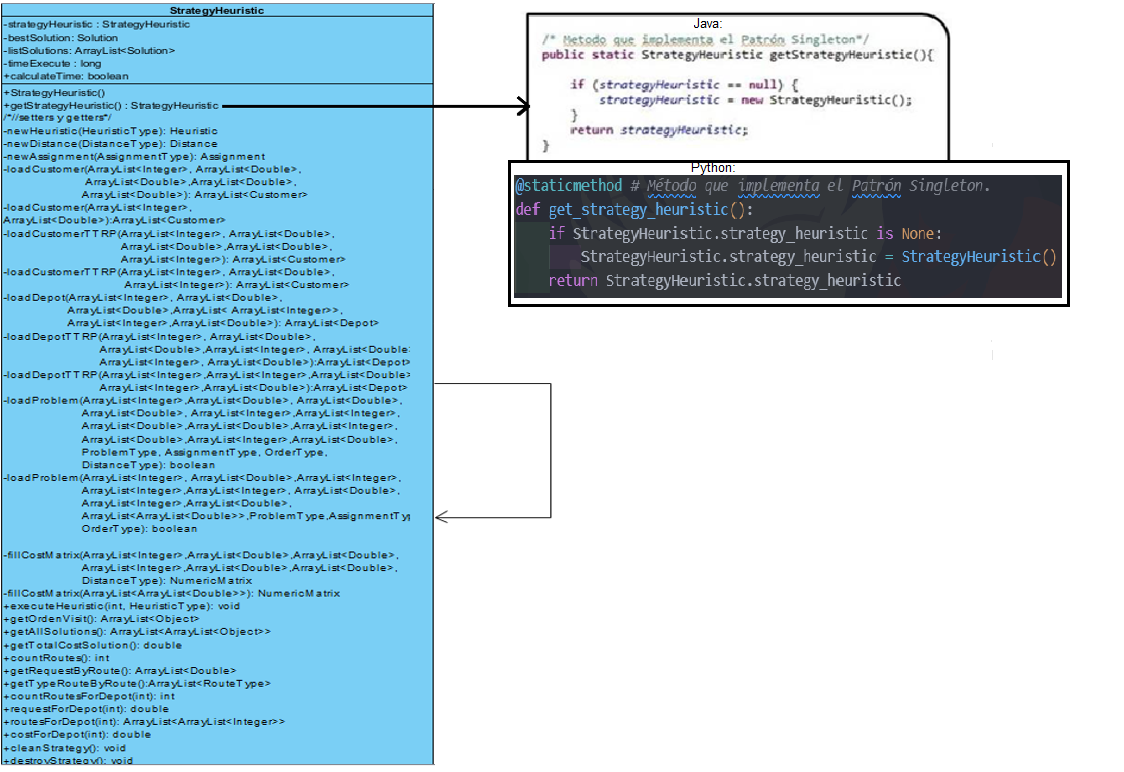


Figura 15: Implementación del patrón Singleton en la clase *StrategyHeuristic*.

### **2.5.2 Patrón *Factory Method***

El patrón *Factory Method* define una interfaz para la creación de un objeto dejando a sus sub-clases la responsabilidad de decidir qué 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 [93-95].

La nueva versión del componente está diseñada de forma robusta para permitir la incorporación de nuevas heurísticas de construcción, métodos para el cálculo de distancias y métodos para los pasos de postoptimización. Para estos casos se implementó este patrón que permite realizar la carga dinámica de cualquier clase.

Para la implementación del mecanismo de diseño se creó la clase *FactoryLoader*, que contiene el método estático *getInstance(*), que recibe como parámetro el nombre de la clase que se desea instanciar. El comportamiento de este método se basa en verificar que exista una clase con el mismo nombre que el parámetro que se le pasa y devolver una instancia de la misma. En la Figura 16, se puede apreciar el diseño de clases que implementa este patrón para el caso de las heurísticas de construcción, las distancias y los operadores. En color azul se representan las clases que se mantienen igual, en color morado las que han sido modificadas y en color verde las nuevas que se incorporan.

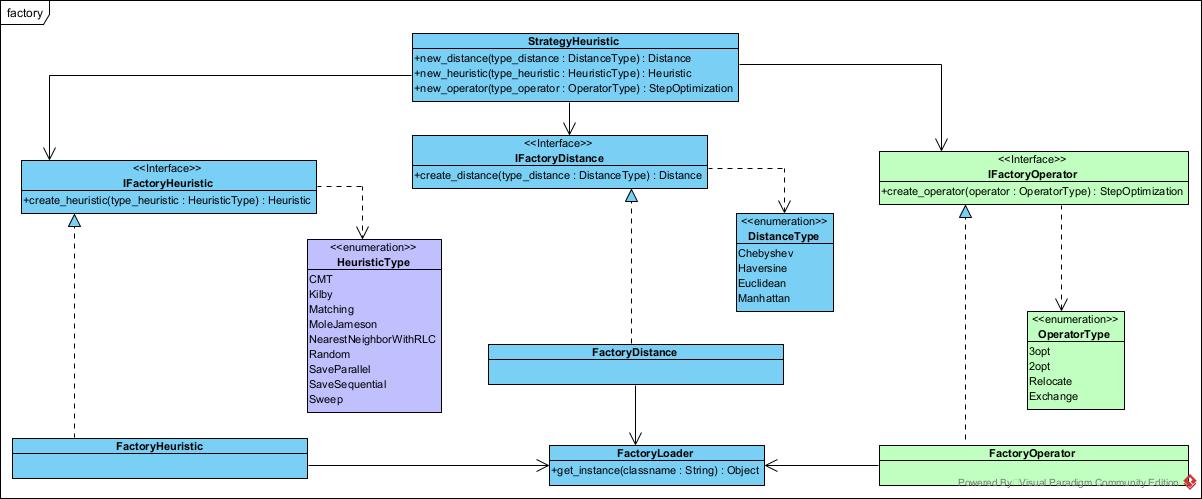


Figura 16: Implementación del patrón Factory Method.

### **2.5.3 Patrón *Template***

El patrón de diseño *Template Method* es un patrón estructural que proporciona una plantilla para la operación en lugar de implementarla completamente. Este patrón define el esqueleto de un algoritmo en una operación, delegando algunos pasos a las subclases. La intención consiste en permitir que las subclases redefinan ciertos pasos de un algoritmo sin alterar la estructura del mismo. El patrón *Template Method* facilita la reutilización de código y mantiene la flexibilidad en la implementación de algoritmos [93].

Una de las principales ventajas del patrón *Template Method* es que permite cambiar la secuencia de pasos de un algoritmo sin modificar su estructura. Esto es especialmente útil cuando diferentes variantes de un algoritmo comparten la misma estructura, pero difieren en sus pasos. Al definir estos pasos en métodos individuales, las subclases pueden sobrescribir esos métodos para personalizar el comportamiento del algoritmo según sea necesario. Esto mejora la cohesión y reduce la duplicación de código, ya que la lógica común se encapsula en la clase base [93].

En BHCVRP este patrón se aplica para manejar los comportamientos de las heurísticas y solucionar la deficiencia relacionada con flexibilidad y reutilización. Se tiene una clase abstracta *Heuristic,* que contiene la configuración de los métodos necesarios para sus respectivas clases hijas. La función *comun\_initialize()* agrupa todos los parámetros que se repiten, mientras que *initialize\_specifics()* permite a cada clase definir que más necesita. Los métodos *creating(), processing()* y *execute()* permiten crear las rutas, procesarlas de manera distinta según los intereses de cada heurística en particular y ejecutarlas para obtener una solución respectivamente. Por último, el método *get\_solution\_initial()* obtiene la solución esperada y *template\_method()* define el esquema general de trabajo de todas las heurísticas. A continuación, en la Figura 18, se muestra el diagrama de clases relacionado con lo descrito anteriormente. Este diagrama constituye una simplificación de la Figura 13, para resaltar solamente los aspectos relacionados con el patrón de diseño *Template.*



Figura 17: Implementación del patrón *Template Method*.

### **2.5.4 Otros patrones de diseño**

Dentro de los patrones más usados además de los GoF también se encuentra el grupo de los patrones GRASP, este nombre es un acrónimo de *General Responsibility Assignment Software Patterns*. El nombre se eligió para indicar la importancia de captar estos principios cuando se desea diseñar eficazmente el software orientado a objetos. Estos patrones son los que describen los principios fundamentales de la asignación de responsabilidades en objetos [94, 95]. Existen diferencias significativas entre estos grupos debido a que los GoF dan una solución implementable con su propio diagrama de clases que muestra la forma en que deben ser usados; mientras los GRASP no implementan las soluciones, más bien llevan a pensar en el diseño, a nivel de principios generales. Los patrones pertenecientes a este grupo son: Experto, Creador, Bajo Acoplamiento, Alta Cohesión, Controlador, Polimorfismo, Fabricación Pura, Indirección y No Hables con Extraños. A continuación, se describen los que se ponen en práctica en la implementación de BHCVRP [39]:

* **Patrón Controlador:** este patrón permite resolver el tema de quién es el responsable de atender un evento del sistema. La solución consiste en asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase. Por ejemplo, en BHCVRP se pone en práctica en la clase *StrategyHeuristic* que es la clase controladora de todo el proceso de la biblioteca (ver Figura 15).
* **Patrón Experto:** con este patrón se pretende resolver el tema de delegar responsabilidades a cada objeto. La solución es asignar una responsabilidad al experto en información, es decir, la clase que tiene la información necesaria para llevar a cabo la responsabilidad. En BHCVRP se aplica con la clase *Problem* que es la responsable de todos datos de un problema VRP y de brindar información necesaria respecto a ellos (ver Figura 7).
* **Patrón Creador:** el problema de saber a quién le corresponde crear una nueva instancia de una clase es abordado con este patrón. El mismo propone como solución crear una nueva instancia por la clase que: tiene la información necesaria para realizar la creación del objeto, usa directamente las instancias creadas del objeto, almacena o maneja varias instancias de la clase y contiene o agrega la clase. La clase controladora *StrategyHeuristic* de BHCVRP contiene una instancia de *Problem* que demuestra el uso de este patrón (ver Figura 15).
* **Patrón de Bajo Acoplamiento:** este patrón permite resolver el problema de dar soporte a las bajas dependencias y al incremento de la reutilización. La solución es realizar un diseño donde las clases estén menos ligadas entre sí. De tal forma que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. Un ejemplo en BHCVRP se evidencia en la clase *Save,* una clase base creada para agrupar los métodos comunes de las sub-clases *Save\_Sequential* y *Save\_Parallel* (ver Figura 13).
* **Patrón de Alta Cohesión:** este patrón es usado si se presentan problemas para manejar la complejidad. La solución es asignar responsabilidades de manera que la información que almacena una clase sea coherente y esté relacionada con la clase. En BHCVRP se emplea en la clase *Customer*, la responsable de los datos de un cliente.

## **2.6 Conclusiones parciales**

Tras el proceso de rediseño de la arquitectura de la biblioteca de clases y con la adaptación de nuevas heurísticas de construcción en diferentes variantes de VRP se obtienen las siguientes conclusiones:

* Se realizan las modificaciones en la arquitectura de BHCVRP para la incorporación de nuevas heurísticas de construcción, cubriendo las deficiencias planteadas en la versión anterior. Además, se tienen en cuenta el patrón repositorio y las arquitecturas basada en componente y n-capas.
* Se incorporan mejoras para el funcionamiento de la biblioteca como son el tratamiento de excepciones propias relacionadas con la distancia, capacidad de los vehículos, velocidad y tiempo de ejecución, así como la aplicación de principios de diseño como encapsulación de la variabilidad, *Hollywood*, cohesión y acoplamiento, *Open/Close* y *Don´t Repeat Yourself*.
* Se propone la primera versión de la biblioteca de clases que cubre los lenguajes de programación Python y Java.
* Se incorporan dos heurísticas de construcción: el Algoritmo de Ahorro basado en *Matching* y la Heurística de Inserción de Kilby.
* En el diseño de la aplicación se incorpora el uso del patrón *Template*.

# **Capítulo 3: Análisis experimental**

## **3.1 Introducción**

En este capítulo se presentan los resultados obtenidos con las nueve heurísticas implementadas en la nueva versión de BHCVRP para las cuatro variantes de problemas de planificación de rutas de vehículos. Inicialmente, se explican los diferentes tipos de experimentos a realizar, las características de las instancias y las configuraciones necesarias para las heurísticas y variantes VRP. Por último, se analizan los resultados alcanzados en los diferentes experimentos.

## **3.2 Descripción de los experimentos**

Para validar el funcionamiento de la nueva versión de BHCVRP, se decide realizar los siguientes experimentos:

1. Experimento 1: consiste en ejecutar las nueve heurísticas implementadas en Python, para analizar su funcionamiento en las cuatro variantes VRP.
2. Experimento 2: se realiza una comparación de los resultados actuales obtenidos en Python con los resultados de la versión anterior de la biblioteca en Java, teniendo en cuenta siete heurísticas.
3. Experimento 3: se realiza una comparación de los mejores resultados obtenidos en BHCVRP con los resultados de OR-Tools para las variantes CVRP y HFVRP.

### **3.2.1 Descripción de las instancias**

Para la ejecución de los experimentos se seleccionan para las cuatro variantes de planificación de rutas de vehículos instancias de la literatura. A continuación, las Tabla 12, Tabla 13, Tabla 14 y Tabla 15, muestran las instancias empleadas en las variantes CVRP, HFVRP, MDVRP y TTRP respectivamente.

Para la variante CVRP se seleccionan instancias del autor Cordeau disponibles en [96], con ligeras transformaciones. Para cada instancia se mantienen los clientes, cantidad de vehículos y sus capacidades con sus datos originales, sólo se adaptan para trabajar con un único depósito. Para la experimentación se utilizan un total de ocho instancias de prueba. En la siguiente Tabla 12, se presenta para cada instancia del problema: el total de clientes, el total de vehículos y la capacidad de estos vehículos.

Tabla 12: Descripción de las instancias CVRP.

|  |  |  |  |
| --- | --- | --- | --- |
| Problema | Total de clientes | Total de vehículos | Capacidad de los vehículos |
| 1 | 50 | 5 | 200 |
| 2 | 75 | 9 | 250 |
| 3 | 75 | 3 | 480 |
| 4 | 100 | 5 | 400 |
| 5 | 360 | 5 | 500 |
| 6 | 200 | 50 | 200 |
| 7 | 400 | 100 | 200 |
| 8 | 600 | 150 | 200 |

Las cantidades de clientes de estas instancias oscilan entre 50 y 600. La cantidad de vehículos varía en la mayoría de las instancias y sus capacidades oscilan entre 200 y 500.

En el caso particular de la variante HFVRP, no están disponibles instancias de pruebas en la literatura y se decide utilizar las instancias seleccionadas para la variante CVRP. Para cada instancia se mantienen los clientes y cantidad de vehículos con sus datos originales, mientras que sus capacidades se distribuyen de manera aleatoria entre los vehículos, siempre respetando la capacidad total original. Con este procedimiento se obtiene una flota de vehículos con varios tipos de vehículos diferentes según su capacidad. A continuación, la Tabla 13 muestra en la segunda y tercera columna el total de clientes y vehículos respectivamente. Luego, en las restantes columnas se presentan los datos para cada tipo de vehículo definido (total y capacidad).

Tabla 13: Descripción de las instancias HFVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datos originales | | | Flota de Vehículos | |
| Problema | Total de clientes | Total de vehículos | Total | Capacidad |
| 1 | 50 | 5 | 1 | 200 |
| 1 | 400 |
| 1 | 100 |
| 1 | 250 |
| 1 | 50 |
| 2 | 75 | 9 | 2 | 250 |
| 2 | 100 |
| 2 | 455 |
| 1 | 90 |
| 1 | 50 |
| 1 | 500 |
| 3 | 75 | 3 | 2 | 470 |
| 1 | 500 |
| 4 | 100 | 5 | 1 | 400 |
| 1 | 100 |
| 1 | 200 |
| 2 | 650 |
| 5 | 360 | 5 | 3 | 500 |
| 1 | 400 |
| 1 | 600 |
| 6 | 200 | 50 | 25 | 240 |
| 25 | 160 |
| 7 | 400 | 100 | 50 | 320 |
| 50 | 80 |
| 8 | 600 | 150 | 75 | 220 |
| 75 | 180 |

Al igual que en la variante CVRP, se dispone de ocho instancias para realizar los experimentos. Estas instancias tienen las mismas características en cuanto a la cantidad de clientes, vehículos y la capacidad total.

Para la variante MDVRP se seleccionan instancias del autor Cordeau disponibles en [96]. Para los experimentos se utilizan un total de ocho instancias de prueba. En la Tabla 14 se presenta para cada instancia del problema: el total de clientes, el total de depósitos, las cantidades de vehículos por depósito y las capacidades de los vehículos.

Tabla 14: Descripción de las instancias MDVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problema | Total de clientes | Total de depósitos | Total de vehículos por depósito | Capacidad de los vehículos |
| 1 | 50 | 4 | 4 | 80 |
| 2 | 100 | 2 | 5 | 200 |
| 3 | 160 | 4 | 5 | 60 |
| 4 | 360 | 9 | 5 | 60 |
| 5 | 288 | 4 | 6 | 175 |
| 6 | 249 | 4 | 8 | 500 |
| 7 | 50 | 4 | 2 | 160 |
| 8 | 144 | 4 | 3 | 190 |

Las cantidades de clientes de estas instancias oscilan entre 50 y 360. Para el caso de los depósitos, la máxima cantidad es nueve y la mínima dos. Además, la cantidad de vehículos y sus capacidades varía en la mayoría de las instancias.

Por último, para la variante TTRP se emplean ocho instancias de referencias de la literatura reportadas en [71] como se muestra en la Tabla 15. Estas instancias TTRP provienen de siete problemas VRP de prueba [75].

Tabla 15: Descripción de las instancias TTRP.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | Clientes | | | Camiones | | Remolques | |
| Total | VC | TC | Total | Capacidad | Total | Capacidad |
| 1 | 50 | 38 | 12 | 5 | 100 | 3 | 100 |
| 2 | 75 | 38 | 37 | 9 | 100 | 5 | 100 |
| 3 | 100 | 50 | 50 | 8 | 150 | 4 | 100 |
| 4 | 120 | 60 | 60 | 7 | 150 | 4 | 100 |
| 5 | 100 | 50 | 50 | 10 | 150 | 5 | 100 |
| 6 | 199 | 100 | 99 | 17 | 150 | 9 | 100 |
| 7 | 50 | 13 | 37 | 5 | 100 | 3 | 100 |
| 8 | 150 | 38 | 112 | 12 | 150 | 6 | 100 |

Las cantidades de clientes de estas instancias oscilan entre 50 y 199. Las capacidades de los camiones toman valores entre 100 y 150. Para el caso de los remolques la capacidad siempre es 100.

### **3.2.2 Configuración de los experimentos**

Todas las heurísticas adaptadas a las distintas variantes fueron implementadas en el lenguaje de programación Python versión 3.12.0, utilizando el IDE de programación PyCharm 2024.1.3 (*Community Edition*) [97]. Para el caso de la versión anterior en Java [39], se utilizó el IDE Apache NetBeans 16 [98] y JDK 11. Los experimentos se ejecutaron en un ordenador con un procesador Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz, 20 GB de RAM y sistema operativo de 64 bits, versión Windows 10. En todos los casos se realizan 20 ejecuciones de las heurísticas seleccionadas debido a que no son determinísticas y sus resultados varían, por lo que es necesario realizar pruebas estadísticas.

#### **3.2.2.1 Heurísticas de construcción y parámetros**

Se utilizan las nueve heurísticas de construcción implementadas en BHCVRP. Algunas de estas heurísticas requieren de parámetros en su ejecución. A continuación, se presenta la configuración de estos parámetros para las heurísticas que lo requieren:

1. Algoritmo Aleatorio (*Random*): este algoritmo construye una solución realizando sucesivas inserciones de los clientes de forma aleatoria.
2. Heurística del Vecino más Cercano con Lista de Candidatos Restringidos (*NN*): propone la construcción de una solución insertando al cliente más cercano dentro de la lista de candidatos restringida al último cliente ya insertado.
   * Tamaño de la lista de candidatos restringidos (*size\_RLC*): 3
3. Algoritmo de Ahorros en su versión secuencial (*SaveSeq*): este algoritmo plantea que en una solución dos rutas diferentes pueden ser combinadas formando una nueva ruta. Esta versión del algoritmo propone la construcción de las rutas una a una.
   * *(parameter\_shape):* 1
4. Algoritmo de Ahorros en su versión paralela (*SaveParall*): esta otra versión del algoritmo tiene la misma filosofía, pero trabaja sobre todas las rutas de forma simultánea.
   * *(parameter\_shape):* 1
5. Algoritmo de Ahorros basado en *Matching* (*SaveMatch*): esta heurística constituye otra alternativa del algoritmo, pero resuelve un problema de *matching* de peso máximo teniendo en cuenta los pesos de las aristas de cada nodo perteneciente a la lista de clientes en las rutas. Por último, verifica si la unión de dichas rutas es factible, teniendo en cuenta la posible afectación en el futuro.
   * *(parameter\_shape):* 1
6. Heurística de Inserción Secuencial de Mole & Jameson (*MJ*): esta heurística de inserción utiliza dos medidas para decidir el próximo cliente a insertar en la solución.
   * (c1): 1
   * (c2): 1
   * Forma de seleccionar el primer cliente a insertar en la ruta (*select\_customer\_MJ\_type*): 1 (el más cercano al depósito).
7. Heurística de Barrido (*Sweep*): esta heurística trabaja en dos fases, primero forma los grupos de clientes y luego construyen las rutas para cada grupo.
8. Heurística de Inserción en Paralelo de Christofides, Mingozzi y Toth (*CMT*): esta heurística de inserción opera en dos fases. En la primera fase se determina la cantidad de rutas a utilizar, junto con un cliente para inicializar cada ruta. En la segunda fase se crean dichas rutas y se inserta el resto de los clientes en ellas según el costo de inserción.
   * (*parameter\_L*): 1
9. Algoritmo de Kilby (*Kilby*): esta heurística de inserción construye tantas rutas como vehículos y se añade el cliente que más cercano esté en cada momento según su costo de inserción.

#### **3.2.2.2 Variantes VRP**

En los casos de las variantes existentes, se requiere de distintos tipos de configuración para realizar su resolución. En el caso de MDVRP es necesario realizar la asignación de los clientes a cada depósito a priori de la ejecución de la heurística. Para la variante HFVRP se debe organizar la lista de las capacidades para lograr obtener una solución que respete las capacidades de los vehículos de la flota y que sea comprensible para el usuario. BHCVRP contempla varias opciones para las configuraciones de cada una de estas variantes VRP.

A continuación, se muestran los métodos de asignación para MDVRP que utiliza la biblioteca BHAVRP [80]: *BestCyclicAssignment*, *BestNearest*, *CoefficientPropagation*, *K\_Means*, *NearestByCustomer*, *NearestByDepot*, *PAM*, *Parallel*, *RandomByElement*, *RandomNearestByCustomer*, *RandomNearestByDepot*, *RandomSequentialCyclic*, *RandomSequentialNearestByDepot*, *SequentialCyclic*, *SequentialNearestByDepot*, *Simplified*, *Sweep*, *ThreeCriteriaClustering* y *UPGMC.*

Por último, los métodos de ordenamiento para HFVRP son:

* *Ascending:* método que ordena la lista de capacidades de menor a mayor.
* *Descending:* método que ordena la lista de capacidades de mayor a menor.
* *Random:* método que ordena la lista de capacidades de forma aleatoria.

#### **3.2.2.3 Pruebas estadísticas no paramétricas**

Las técnicas estadísticas que involucran la estimación de parámetros, la construcción de intervalos de confianza y la realización de pruebas de hipótesis se agrupan bajo el término de estadística paramétrica. Estas metodologías requieren la definición previa de una forma específica para la distribución de la variable aleatoria y para los estadísticos derivados de los datos. Dentro de la estadística paramétrica, se presupone que la población de la que se extraen muestras es normal o presenta una similitud notable a la normalidad. Esta suposición es crucial para garantizar la validez de las pruebas de hipótesis. Sin embargo, en numerosos escenarios, no es posible identificar la distribución primigenia ni la distribución de los estadísticos, lo que significa que, en lugar de estimar parámetros, se buscan comparar distribuciones. Este enfoque se denomina estadística no paramétrica [99].

Para evaluar los resultados obtenidos, se diseña una prueba de hipótesis con el fin de determinar si existen diferencias en el comportamiento de los algoritmos en general:

: No existen diferencias significativas entre los algoritmos.

: Existen diferencias significativas entre los algoritmos.

Todas las pruebas de hipótesis se realizan con un nivel de significancia de .

Para examinar el comportamiento general de los algoritmos, se emplea la prueba estadística no paramétrica de Friedman [100]. Sin embargo, para analizar el rendimiento de los algoritmos en ambas versiones de la biblioteca y con OR-Tools, se aplica la prueba de Wilcoxon [101]. A continuación, se detallan las características de estas pruebas:

* Friedman: se utiliza para el análisis de varianza de dos clasificaciones por rango, en situaciones donde se cuentan con muestras correlacionadas y problemas distintos. Esta prueba implica calcular un *ranking* para los resultados generados por cada uno de los algoritmos para cada problema, asignando el *ranking* 1 al mejor resultado y el *ranking* al peor. La hipótesis nula de esta prueba asume que los resultados producidos por los algoritmos son equivalentes, lo que implicaría que sus *rankings* son comparables [100].
* Wilcoxon: es una técnica estadística diseñada para comparar dos muestras independientes en el contexto de varios problemas. Es particularmente útil para datos medidos a nivel ordinal, permitiendo no solo identificar la dirección de las diferencias entre pares de datos, sino también evaluar la magnitud relativa de esas diferencias. La prueba opera mediante la identificación de diferencias entre los resultados de dos algoritmos, seguido de un proceso de *ranking* basado en estas diferencias, sin tener en cuenta el signo, a través de todos los problemas. Importante es que el *ranking* se asigna de 1 a N, en lugar de hasta . Posteriormente, se suman los *rankings*, distinguindo entre aquellos que son negativos y positivos, resultando en dos valores, R+ y R−. Si el menor de estos valores es igual o inferior al valor esperado de la distribución T de Wilcoxon para N grados de libertad, se concluye que no hay suficiente evidencia para aceptar la hipótesis nula, lo que indica que uno de los algoritmos es superior [101].

## **3.3 Análisis de los resultados**

Esta sección está dividida en tres partes, una por cada tipo de experimento realizado. En la primera sección se analiza el comportamiento de cada una de las heurísticas de construcción para cada variante del problema de planificación de rutas de vehículos. Se muestran gráficas para un mejor análisis de los resultados obtenidos respecto al costo de las soluciones y el tiempo de ejecución en las cuatro variantes. Además, se demuestra con la prueba estadística no paramétrica de Friedman [100], la validez de los resultados obtenidos.

En la segunda sección se analizan los resultados obtenidos de comparar las siete heurísticas de construcción de la versión anterior en Java de la biblioteca con estas mismas heurísticas en la versión actual en Python para las cuatro variantes VRP. Se compara de forma detallada los resultados en cuanto a costo en distancia y tiempo en segundos. La prueba de Wilcoxon [101] es utilizada para detectar diferencias significativas entre los algoritmos de cada versión de la biblioteca.

Por último, en la tercera parte se analizan los mejores resultados obtenidos en la versión actual desarrollada en Python con OR-Tools, para las variantes CVRP y HFVRP. Se tiene en cuenta el costo en distancia, el tiempo y se aplica la prueba de Wilcoxon [101] para detectar diferencias.

### **3.3.1 Resultados obtenidos con BHCVRP en Python**

Después de realizados los experimentos se analizan los resultados y se muestra un resumen por cada variante VRP. En las tablas que muestran el resumen de los resultados por cada problema se resalta en color azul, la mejor solución obtenida y el menor tiempo de ejecución. Para los valores de costo, se consideran dos cifras significativas y para los tiempos de ejecución se admiten cuatro cifras significativas. Se coloca entre paréntesis la cantidad de rutas obtenidas para las mejores soluciones alcanzadas. En el caso de la variante TTRP, el primer valor corresponde con las rutas de tipo CVR, el segundo con las de tipo PVR y por último las de tipo PTR.

En todos los casos se grafican los resultados alcanzados en cuanto a costo de las soluciones y tiempo de ejecución utilizando la herramienta Excel.

#### **3.3.1.1 Análisis de los resultados para la variante CVRP**

En la Tabla 16 y la Tabla 17 se resumen los resultados obtenidos sobre los ocho problemas de la variante CVRP seleccionados previamente.

Tabla 16: Resumen de los resultados en las ocho instancias CVRP (I).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *Random* | | | *NN* | | | *SaveSeq* | | | *SaveParall* | | | *SaveMatch* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 1537.38 | **0.0019** |  | 1474.08 | 0.1186 |  | 576.60 | 0.5448 |  | 568.86 | 1.2490 |  | 1005.35 | 0.6707 |
| 2 |  | 2392.85 | **0.0535** |  | 2249.47 | 0.3844 |  | 815.77 | 1.2557 |  | 750.04 | 3.1785 |  | 1260.79 | 2.1261 |
| 3 |  | 2344.06 | **0.0456** |  | 2214.83 | 0.4135 |  | 702.43 | 1.2211 |  | 673.47 | 2.9397 |  | 1203.82 | 2.2798 |
| 4 |  | 3051.01 | **0.0584** |  | 2729.04 | 0.8748 |  | 878.09 | 2.7445 |  | 786.31 | 6.4984 |  | 1748.28 | 4.2555 |
| 5 |  | 45456.2 | **0.1443** |  | 22842.9 | 16.024 |  | 6215.2 | 94.805 |  | 5386.7 | 309.6680 |  | 35587.11 | 180.0369 |
| 6 |  | 14776.5 | **0.1187** |  | 13947.04 | 5.0068 |  | 2981.08 | 16.2967 |  | 2689.54 | 47.8148 |  | 6442.67 | 33.7310 |
| 7 |  | 43345.77 | **0.0544** |  | 41755.41 | 3.4151 |  | 8426.49 | 12.167 |  | 7261.48 | 496.4777 |  | 14597.07 | 236.7240 |
| 8 |  | 89997.35 | **0.188** |  | 86530.78 | 6.434 |  | 17402.21 | 451.87 |  | 13678.65 | 951.639 |  | 29516.93 | 889.3156 |

Tabla 17: Resumen de los resultados en las ocho instancias CVRP (II).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *MJ* | | | *CMT* | | | *Kilby* | | | *Sweep* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 612.15 | 2.4681 |  | 640.49 | 0.5086 |  | 1060.82 | 1.1911 |  | 1514.74 | 0.2259 |
| 2 |  | 816.92 | 8.7697 |  | 854.40 | 1.3093 |  | 1255.48 | 5.9780 |  | 2386.57 | 0.7546 |
| 3 |  | 676.99 | 15.7553 |  | 881.26 | 1.7931 |  | 1856.99 | 2.6769 |  | 2354.26 | 0.8910 |
| 4 |  | 835.15 | 36.1031 |  | 1100.78 | 3.3230 |  | 1570.03 | 7.3232 |  | 3252.77 | 1.9394 |
| 5 |  | 6534.7 | 337.7738 |  | 17693.7 | 31.3677 |  | 32017.5 | 311.9699 |  | 44109.3 | 87.2194 |
| 6 |  | 2875.5 | 125.1227 |  | 2454.23 | 9.7152 |  | 6316.00 | 445.3573 |  | 14925.9 | 13.2280 |
| 7 |  | 7495.7 | 63.1086 |  | 6467.23 | 136.5664 |  | 18904.3 | 304.4157 |  | 43031.5 | 10.8614 |
| 8 |  | 15604.8 | 214.7608 |  | 13421.6 | 62.2868 |  | 42390.56 | 500.9123 |  | 85832.8 | 449.4740 |

A partir de los resultados alcanzados se puede concluir que:

* Las mejores soluciones se encontraron en la heurística *SaveParall.* Sin embargo, también se obtienen buenos resultados para las heurísticas *SaveSeq* y *MJ.*
* Las soluciones más costosas obtenidas con las heurísticas *Sweep* y *NN* son superadas por los resultados de la heurística *Random*, por lo que se puede apreciar que esta heurística obtiene los peores resultados.
* Las heurísticas *CMT, SaveMatch* y *Kilby* tienen un comportamiento regular, pues no quedan entre los mejores resultados, pero tampoco entre los peores.
* Las heurísticas *SaveParall, SaveMatch* y *Kilby* son deterministas, pero al utilizar el operador 3-opt como método de post-optimización deja de serlo.
* La heurística *MJ* obtiene buenos resultados, pero también es el que mayor tiempo emplea en su ejecución. Aunque, mientras aumenta la cantidad de clientes, las que obtienen mayores tiempos son *SaveParall, SaveMatch* y *Kilby.* Sin embargo, *Random* obtiene los peores resultados en las soluciones, pero el menor tiempo de ejecución en todos los casos.

A continuación, en la Figura 18 y Figura 19, se presentan las gráficas que reafirman las conclusiones mencionadas anteriormente:

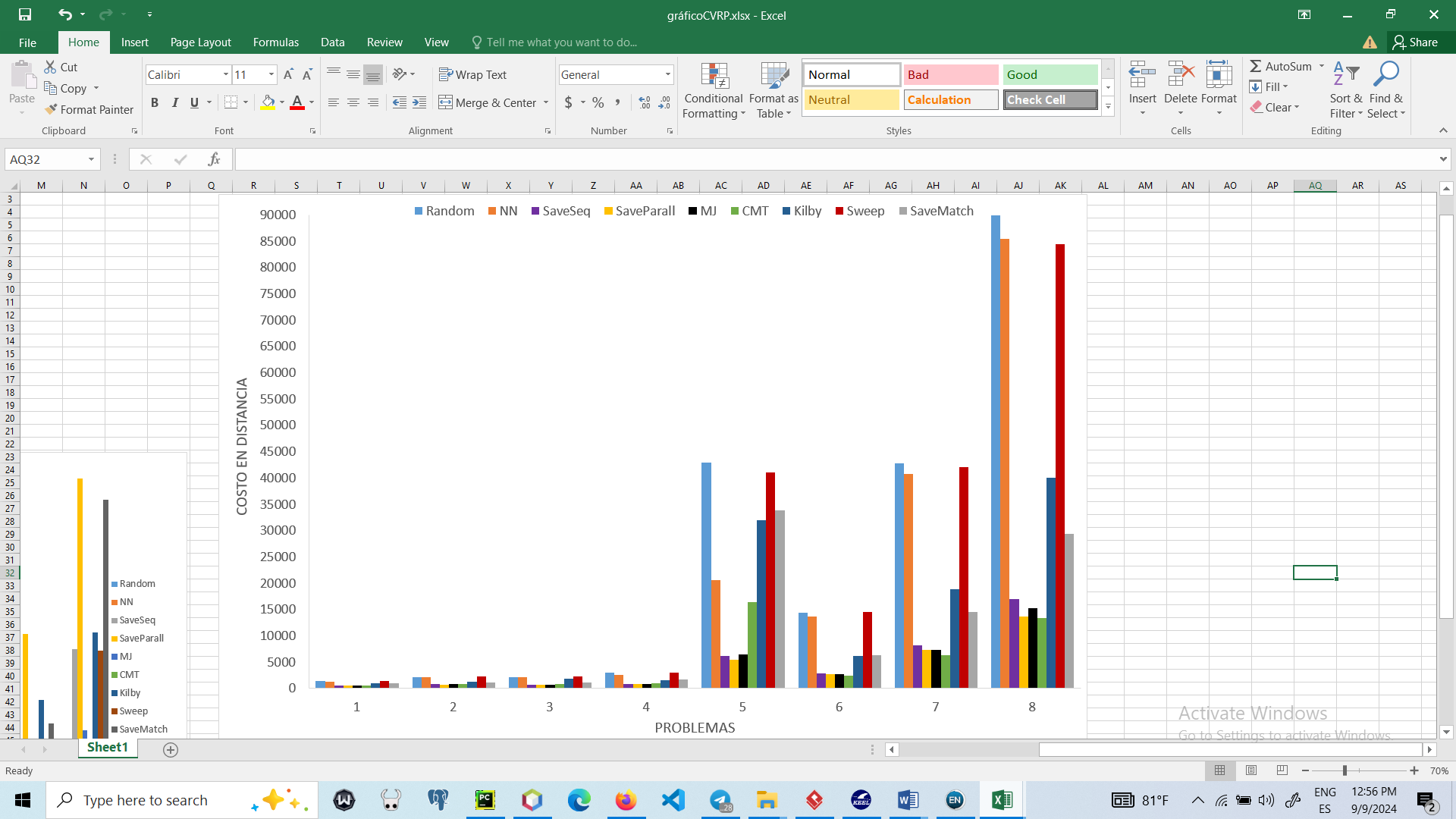


Figura 18: Gráfica de barras para el costo de las heurísticas en la variante CVRP.

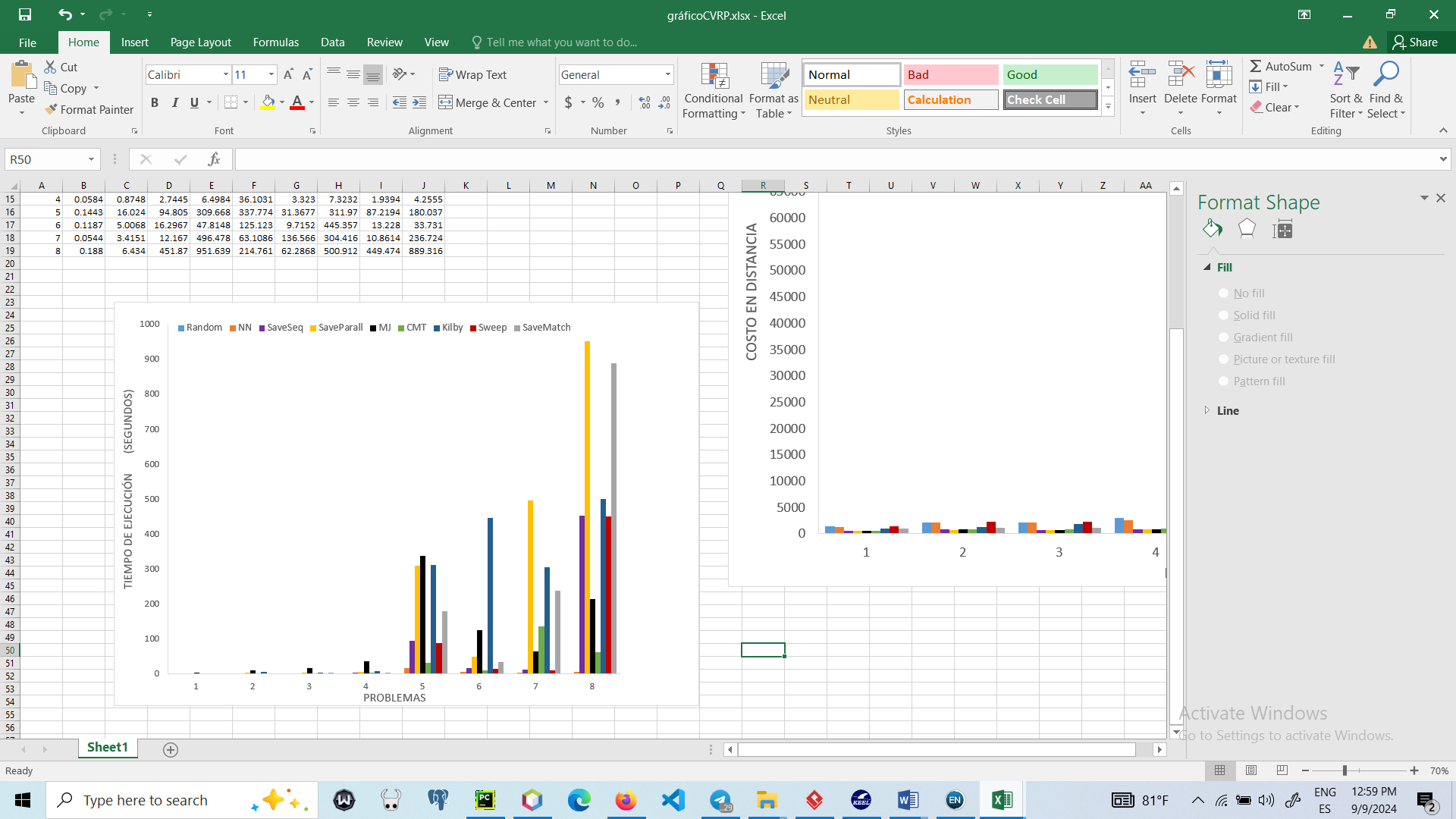


Figura 19: Gráfica de barras para el tiempo de ejecución de las heurísticas en la variante CVRP.

#### **3.3.1.2 Análisis de los resultados para la variante HFVRP**

En la Tabla 18 y la Tabla 19 se resumen los resultados obtenidos sobre los ocho problemas de la variante HFVRP. Para esta variante se realizan experimentos para el método de ordenamiento *Descending* de la lista de capacidades, por obtener los mejores resultados en la versión anterior [39].

Tabla 18: Resumen de los resultados en las ocho instancias HFVRP (I).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *Random* | | | *NN* | | | *SaveSeq* | | | *SaveParall* | | | *SaveMatch* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 1152.25 | **0.0351** |  | 917.40 | 0.1540 |  | 566.53 | 0.4820 |  | 497.99 | 1.0935 |  | 922.51 | 1.7534 |
| 2 |  | 2061.52 | **0.0471** |  | 1696.16 | 0.4395 |  | 835.85 | 1.4416 |  | 851.96 | 1.181 |  | 1185.49 | 7.1024 |
| 3 |  | 1594.74 | **0.0456** |  | 1171.20 | 0.4109 |  | 698.18 | 1.3554 |  | 658.68 | 4.1937 |  | 1213.99 | 3.7284 |
| 4 |  | 3123.60 | **0.0721** |  | 1960.17 | 0.8415 |  | 774.38 | 2.6611 |  | 990.28 | 13.602 |  | 1527.92 | 9.8802 |
| 5 |  | 44459.46 | **0.2249** |  | 22355.98 | 33.670 |  | 6255.36 | 24.611 |  | 777.43 |  |  | 35622.27 | 425.6863 |
| 6 |  | 13923.44 | **0.2639** |  | 13096.29 | 6.1536 |  | 2680.11 | 20.9691 |  | 2620.46 | 102.1874 |  | 6320.81 | 572.2572 |
| 7 |  | 39743.58 | **0.2715** |  | 38047.66 | 35.7714 |  | 5934.57 | 126.7698 |  | 5970.02 | 4.4937 |  | 13838.25 | 8460.7132 |
| 8 |  | 86007.76 | **0.5714** |  | 83576.92 | 119.4991 |  | 15882.32 | 220.9266 |  | 18945.3 | 7.9899 |  | 31177.03 | 40309.355 |

Tabla 19: Resumen de los resultados en las ocho instancias HFVRP (II).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *MJ* | | | *CMT* | | | *Kilby* | | | *Sweep* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 582.13 | 3.9872 |  | 625.13 | 0.6129 |  | 1065.77 | 1.3930 |  | 1121.32 | 0.2746 |
| 2 |  | 816.30 | 9.3635 |  | 851.58 | 1.3590 |  | 1255.69 | 5.7586 |  | 2067.59 | 0.8829 |
| 3 |  | 698.27 | 16.6713 |  | 913.49 | 1.7733 |  | 1896.52 | 3.0008 |  | 1578.88 | 0.8762 |
| 4 |  | 784.41 | 32.1689 |  | 1242.09 | 3.4130 |  | 1556.23 | 7.3406 |  | 2123.70 | 1.8406 |
| 5 |  | 6395.63 | 6855.8267 |  | 18917.08 | 112.6921 |  | 32125.54 | 85.1255 |  | 44260.40 | 80.9879 |
| 6 |  | 2408.97 | 371.9073 |  | 2163.73 | 19.9983 |  | 6231.82 | 490.645 |  | 13096.29 | 6.1536 |
| 7 |  | 5503.95 | 217.3151 |  | 4905.91 | 121.9479 |  | 18890.44 | 6833.156 |  | 40799.19 | 213.4071 |
| 8 |  | 13940.0 | 183.9995 |  | 12345.32 | 1003.677 |  | 37395.33 | 1392.285 |  | 86365.48 | 336.3865 |

A partir de los resultados alcanzados se puede concluir que:

* Las mejores soluciones se encontraron con las heurística *SaveSeq, SaveParall* y *MJ,* mientras que las soluciones de mayor costo fueron encontradas con los algoritmos *Random, Sweep* y *NN.*
* Las heurísticas *CMT, SaveMatch* y *Kilby* no obtienen los mejores resultados, pero tampoco los perores, por lo cual su comportamiento es regular.
* Se puede observar que Random obtiene los peores resultados en las soluciones, pero el menor tiempo de ejecución en todos los casos.
* La heurística *MJ* obtiene buenos resultados, pero también es el que mayor tiempo emplea en su ejecución. Aunque, mientras aumenta la cantidad de clientes, las que obtienen mayores tiempos son *SaveMatch* y *Kilby.*

A continuación, en las Figura 20 y Figura 21, se presentan las gráficas que reafirman las conclusiones mencionadas anteriormente:

#### 

Figura 20: Gráfica de barras para el costo de las heurísticas en la variante HFVRP.

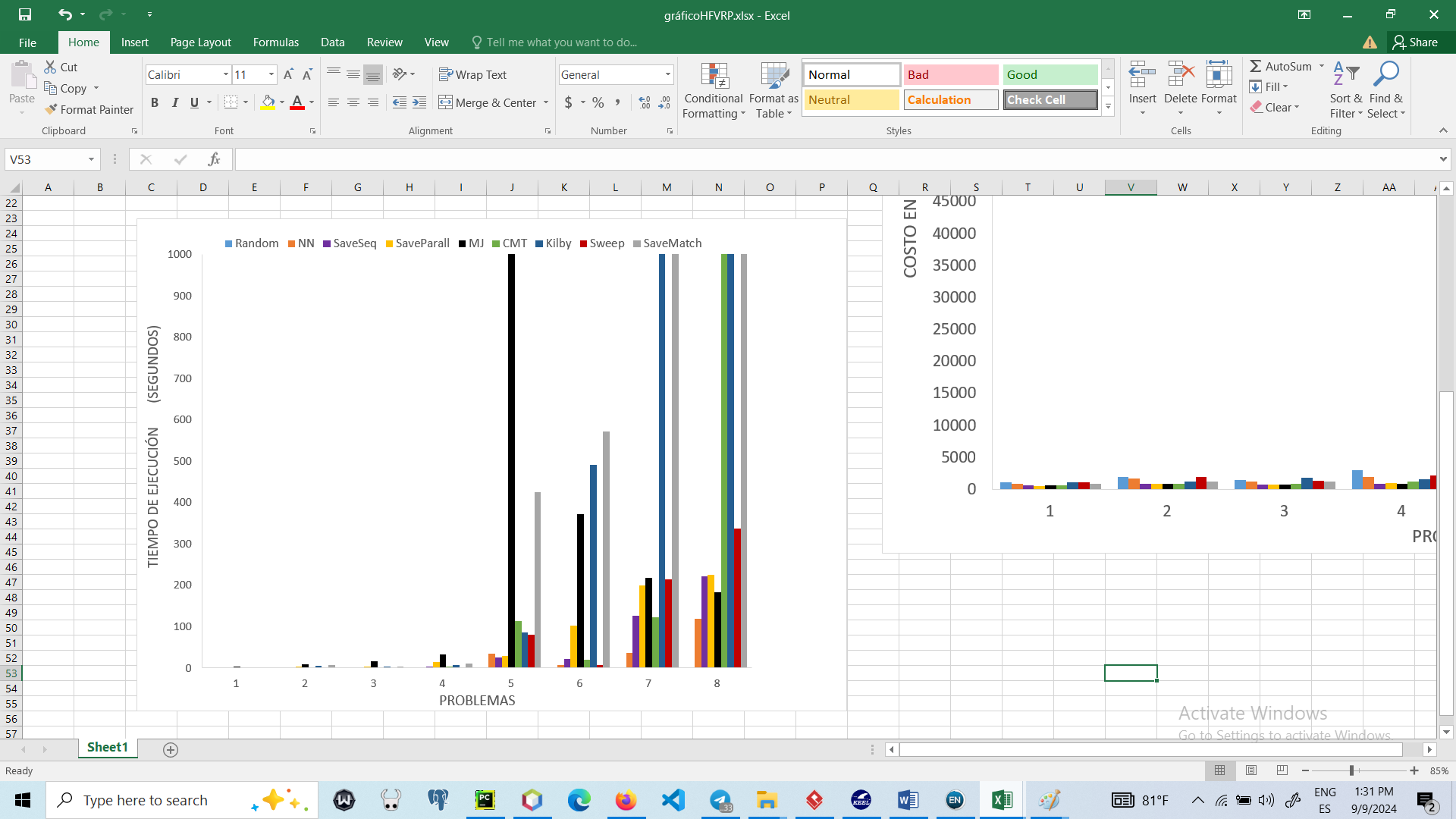


Figura 21: Gráfica de barras para el tiempo de ejecución de las heurísticas en la variante HFVRP.

#### **3.3.1.3 Análisis de los resultados para la variante MDVRP**

En la Tabla 20 y la Tabla 21 se resumen los resultados obtenidos sobre los ocho problemas de la variante MDVRP seleccionados de la literatura. Para esta variante se realizan experimentos para el método de asignación *BestNearest*, por obtener los mejores resultados en la versión anterior [39].

Tabla 20: Resumen de los resultados en las ocho instancias MDVRP (I).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *Random* | | | *NN* | | | *SaveSeq* | | | *SaveParall* | | | *SaveMatch* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 1094.55 | 0.0900 |  | 1014.82 | **0.0828** |  | 1062.98 | 0.2845 |  | 786.07 | 0.4551 |  | 1418.77 | 0.2841 |
| 2 |  | 2552.84 | **0.0788** |  | 2359.61 | 0.5166 |  | 1144.25 | 1.4595 |  | 974.44 | 3.3595 |  | 1406.76 | 2.2098 |
| 3 |  | 8215.06 | **0.1177** |  | 8386.06 | 0.8904 |  | 5698.81 | 2.7322 |  | 3318.6 | 4.7684 |  | 8403.63 | 4.2784 |
| 4 |  | 18917.24 | **0.3887** |  | 19071.19 | 3.3262 |  | 18886.34 | 12.5386 |  | 7466.85 | 16.923 |  | 20070.49 | 20.0164 |
| 5 |  | 11242.51 | **0.2773** |  | 8965.72 | 3.4387 |  | 5039.68 | 13.976 |  | 4024.24 | 23.3302 |  | 6659.06 | 16.0464 |
| 6 |  | 15516.12 | **0.2783** |  | 13761.88 | 2.5329 |  | 8420.09 | 9.2313 |  | 6021.26 | 15.7345 |  | 10448.35 | 8.4764 |
| 7 |  | 936.43 | 0.0925 |  | 925.82 | 0.1336 |  | 755.19 | 0.2717 |  | 602.23 | 0.3698 |  | 872.21 | 2.1052 |
| 8 |  | 6009.17 | **0.1951** |  | 4966.42 | 0.6425 |  | 2761.28 | 2.3026 |  | 2639.18 | 3.6211 |  | 3911.75 | 4.2291 |

Tabla 21: Resumen de los resultados en las ocho instancias MDVRP (II).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *MJ* | | | *CMT* | | | *Kilby* | | | *Sweep* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 645.99 | 0.3784 |  | 748.74 | 0.2281 |  | 1157.68 | 0.2965 |  | 1027.11 | 0.1339 |
| 2 |  | 803.09 | 9.9856 |  | 1030.56 | 1.8661 |  | 1434.98 | 4.9874 |  | 2572.76 | 0.5599 |
| 3 |  | 2790.24 | 18.0643 |  | 3752.61 | 3.4766 |  | 4756.19 | 8.8169 |  | 8219.78 | 0.6854 |
| 4 |  | 6392.52 | 82.2219 |  | 8677.32 | 16.2640 |  | 10937.99 | 43.0887 |  | 18875.29 | 1.5296 |
| 5 |  | 3736.03 | 107.2054 |  | 4139.97 | 15.0633 |  | 7028.82 | 65.3264 |  | 11196.25 | 2.6812 |
| 6 |  | 5133.44 | 48.9223 |  | 5427.71 | 9.4167 |  | 8769.58 | 54.9366 |  | 15591.27 | 2.0606 |
| 7 |  | 367.82 | 0.7900 |  | 635.25 | 0.3275 |  | 862.37 | 0.4989 |  | 947.57 | **0.0618** |
| 8 |  | 1894.69 | 17.2066 |  | 2761.5 | 3.4065 |  | 3890.21 | 5.7184 |  | 5966.67 | 0.4829 |

A partir de los resultados alcanzados se puede concluir que:

* Las mejores soluciones se encontraron con la heurística *MJ,* mientras que las soluciones de mayor costo fueron encontradas con las heurísticas *Random, SaveMatch, NN* y *Sweep*.
* Las heurísticas *SaveParall* y *CMT* obtienen buenos resultados*.*
* Las heurísticas *SaveSeq* y *Kilby* tienen un comportamiento regular, pues no quedan entre los mejores ni peores resultados.
* En los casos donde coinciden los valores mínimos y el promedio, significa que tuvo un comportamiento determinista y que el operador *3opt* no obtuvo mejoras.
* Los mejores tiempos se obtienen con la heurística *Random,* *NN* y *Sweep,* mientras que los peores tiempos fueron con *MJ.*

A continuación, en la Figura 22 y Figura 23, se presentan las gráficas que reafirman las conclusiones mencionadas anteriormente:

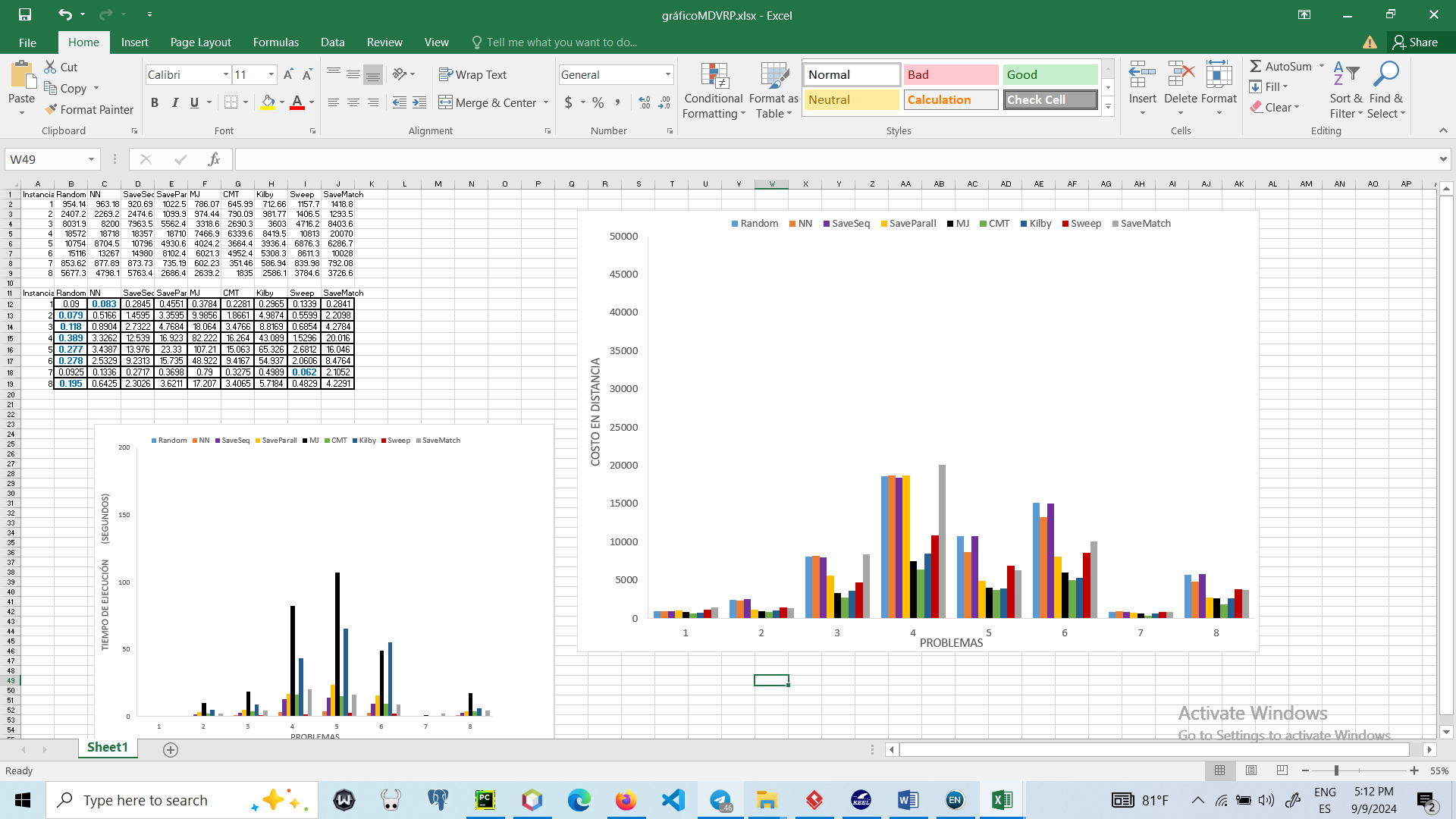


Figura 22: Gráfica de barras para el costo de las heurísticas en la variante MDVRP.

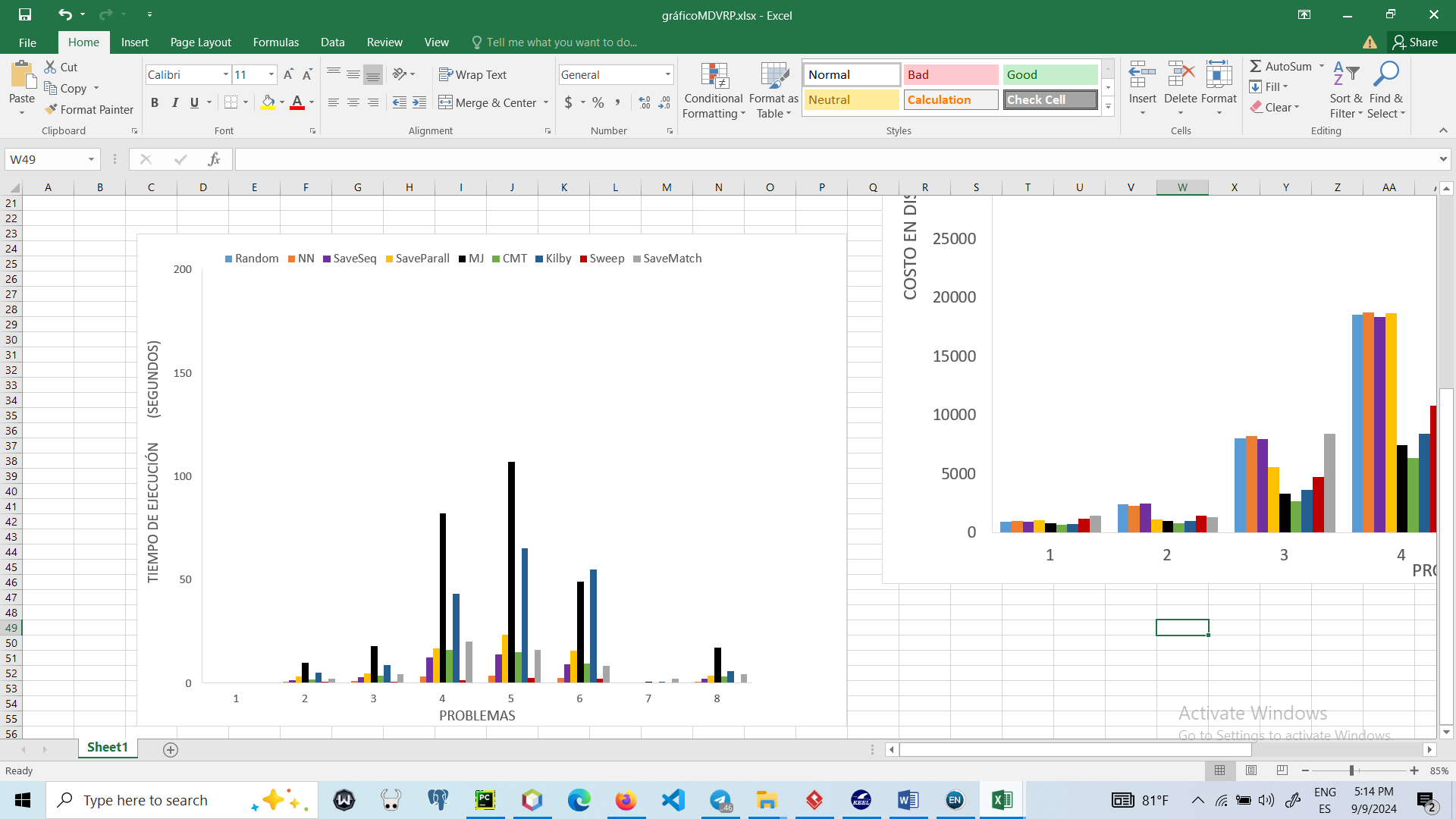


Figura 23: Gráfica de barras para el tiempo de ejecución de las heurísticas en la variante MDVRP.

#### **3.3.1.4 Análisis de los resultados para la variante TTRP**

En la Tabla 22 y la Tabla 23 se resumen los resultados obtenidos sobre los ocho problemas de la variante TTRP seleccionados de la literatura.

Tabla 22: Resumen de los resultados en las ocho instancias TTRP (I).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *Random* | | | *NN* | | | *SaveSeq* | | | *SaveParall* | | | *SaveMatch* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 1561.81 | **0.045** |  | 1524.49 | 0.1817 |  | 610.36 | 0.417 |  | 580.79 | 0.5273 |  | 956.92 | 0.4926 |
| 2 |  | 2545.77 | **0.0508** |  | 2381.66 | 0.3972 |  | 1038.77 | 1.273 |  | 820.58 | 1.4296 |  | 1411.35 | 2.4013 |
| 3 |  | 3327.62 | **0.0630** |  | 3075.71 | 0.7898 |  | 1106.50 | 2.391 |  | 879.66 | 2.701 |  | 1678.23 | 1.9087 |
| 4 |  | 6376.07 | **0.0799** |  | 3217.73 | 1.3613 |  | 998.48 | 4.411 |  | 1070.66 | 5.8528 |  | 1590.34 | 5.8976 |
| 5 |  | 3886.42 | **0.0825** |  | 2308.14 | 0.9214 |  | 1101.46 | 3.323 |  | 747.81 | 3.8310 |  | 1316.76 | 4.9598 |
| 6 |  | 6569.20 | **0.1228** |  | 6191.25 | 5.0142 |  | 1888.82 | 11.138 |  | 1308.28 | 17.616 |  | 1993.44 | 20.5612 |
| 7 |  | 1583.77 | **0.0405** |  | 1563.25 | 0.1554 |  | 786.99 | 0.3428 |  | 597.02 | 0.4937 |  | 901.23 | 0.3565 |
| 8 |  | 4918.28 | **0.0837** |  | 4535.62 | 2.4486 |  | 1189.75 | 5.613 |  | 977.43 | 7.9899 |  | 1876.91 | 6.4523 |

Tabla 23: Resumen de los resultados en las ocho instancias TTRP (II).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *MJ* | | | *CMT* | | | *Kilby* | | | *Sweep* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 616.11 | 1.8323 |  | 881.23 | 0.3237 |  | 1208.06 | 1.1457 |  | 1535.37 | 0.2584 |
| 2 |  | 967.12 | 6.4166 |  | 1401.16 | 1.3781 |  | 1712.69 | 4.9588 |  | 2529.72 | 0.7926 |
| 3 |  | 1150.70 | 14.5486 |  | 1334.63 | 1.8367 |  | 2124.76 | 5.0726 |  | 3323.73 | 1.8182 |
| 4 |  | 1230.55 | 56.5215 |  | 1686.62 | 2.4692 |  | 3385.09 | 14.4167 |  | 6381.68 | 3.0313 |
| 5 |  | 1016.63 | 22.0210 |  | 1288.28 | 1.6385 |  | 2682.64 | 10.1838 |  | 3873.64 | 1.8079 |
| 6 |  | 1904.92 | 105.1122 |  | 2125.39 | 13.2150 |  | 3858.2 | 135.5542 |  | 6634.31 | 12.2426 |
| 7 |  | 645.83 | 1.8917 |  | 891.55 | 0.2515 |  | 1199.14 | 1.0958 |  | 1601.22 | 0.2534 |
| 8 |  | 1284.69 | 71.0268 |  | 1716.66 | 4.8927 |  | 2733.57 | 45.2076 |  | 4944.19 | 5.7698 |

A partir de los resultados alcanzados se puede concluir que:

* Las mejores soluciones se encontraron con la heurística *SaveParall,* mientras que las soluciones de mayor costo fueron encontradas con las heurísticas *Random, NN* y *Sweep*.
* Las heurísticas *MJ* y *CMT* obtienen buenos resultados, similar a *SaveParall.*
* Las heurísticas *SaveSeq* y *Kilby* tienen un comportamiento regular, pues no quedan entre los mejores ni peores resultados.
* En los casos donde coinciden los valores mínimos y el promedio, por ejemplo, la heurística *Kilby,* significa que tuvo un comportamiento determinista y que el operador *3opt* no obtuvo mejoras.
* Los mejores tiempos se obtienen con la heurística *Random,* mientras que los peores tiempos fueron con *MJ.*

A continuación, en la Figura 24 y Figura 25, se presentan las gráficas que reafirman las conclusiones mencionadas anteriormente:

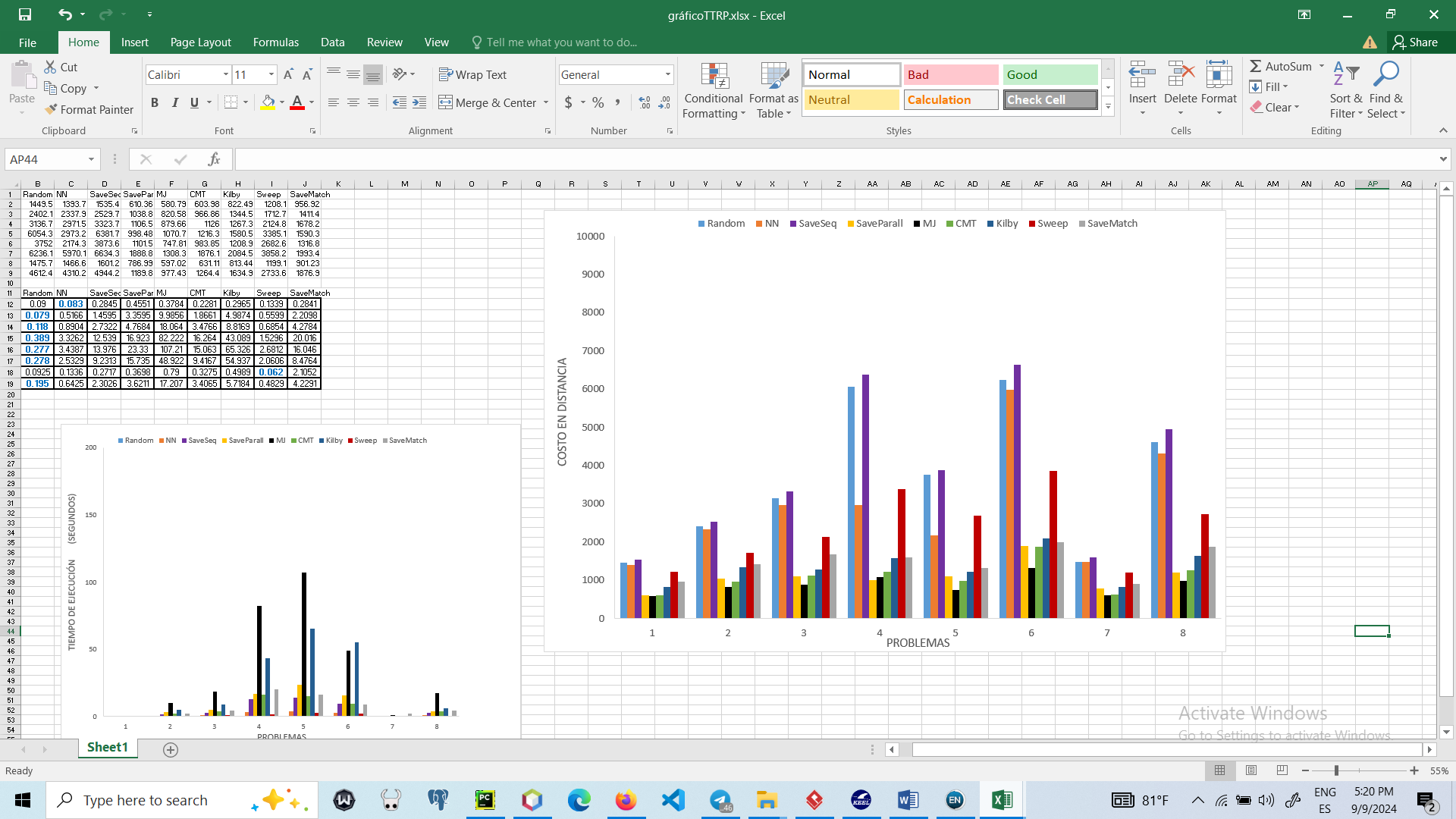


Figura 24: Gráfica de barras para el costo de las heurísticas en la variante TTRP.

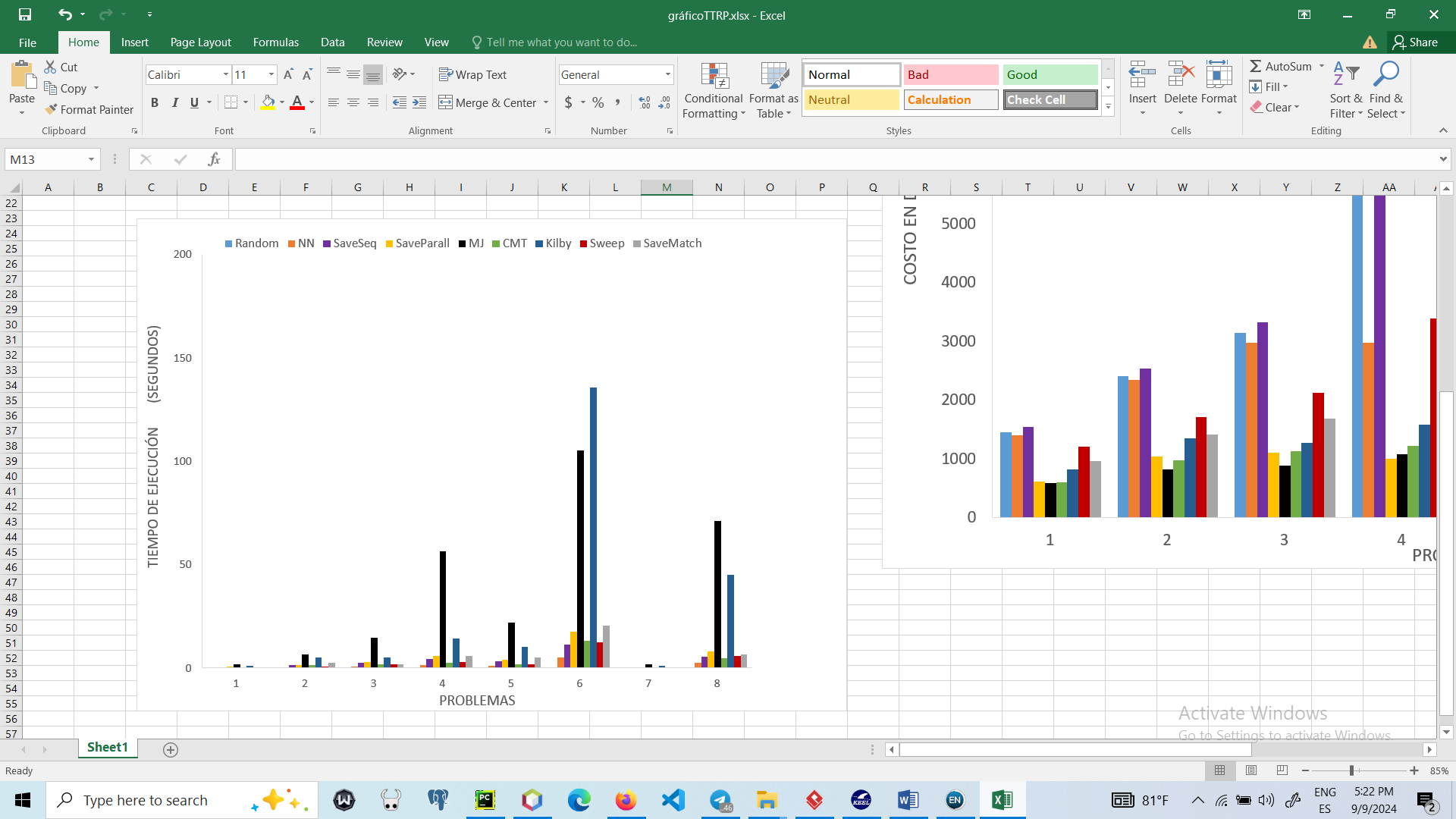


Figura 25: Gráfica de barras para el tiempo de ejecución de las heurísticas en la variante TTRP.

#### **3.3.1.5 Análisis estadístico**

Aunque las gráficas confirman los resultados anteriores, para mayor certeza se decide utilizar la prueba estadística no paramétrica de Friedman [100]. Esta permite comprobar si los resultados obtenidos con las heurísticas de construcción presentan diferencias significativas. Se utiliza el costo promedio de los problemas como medida para comparar el rendimiento de los nueve algoritmos heurísticos.

Tabla 24: Rankings de las heurísticas para las cuatro variantes VRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algoritmos | *Rankings* | | | |
| CVRP | HFVRP | MDVRP | TTRP |
| *Random* | 8.375 | 8.5 | 7.25 | 8 |
| *NN* | 7 | 6.1875 | 7.375 | 6.75 |
| *SaveSeq* | 2.875 | 2.5 | 4.875 | 2.5 |
| *SaveParall* | **1.875** | 2.625 | 2.625 | **1.125** |
| *SaveMatch* | 5.5 | 6.25 | 6.625 | 4.875 |
| *MJ* | 2.375 | **2.25** | **1** | 2.375 |
| *CMT* | 2.875 | 2.625 | 2.375 | 4.125 |
| *Kilby* | 5.75 | 6.25 | 5.625 | 6.25 |
| *Sweep* | 8.375 | 7.8125 | 7.25 | 9 |

En la Tabla 24, se resalta en color azul que para las variantes CVRP y TTRP la heurística *SaveParall* se posiciona como el mejor algoritmo con un *p-value* que sugiere la existencia de diferencias significativas entre los restantes algoritmos heurísticos considerados. Sin embargo, para HFVRP y MDVRP se destaca la heurística *MJ.*

Respecto al tiempo, para mayor certeza de las afirmaciones anteriores se aplica igualmente la prueba estadística no paramétrica de Friedman [100]. Se utiliza el tiempo promedio de los problemas como medida para comparar el rendimiento de las ocho heurísticas. Como resultado queda *Random* como la heurística que menos tiempo demora en su ejecución para las cuatro variantes de planificación de rutas de vehículos con diferencias significativas con las restantes heurísticas.

### **3.3.2 Comparación de los resultados con la versión anterior de BHCVRP**

En las Tabla 25, Tabla 26, Tabla 27, Tabla 28, Tabla 29, Tabla 30, Tabla 31 y Tabla 32, se resumen los resultados obtenidos de aplicar las siete heurísticas implementadas en la versión anterior de BHCVRP sobre las ocho instancias de cada variante VRP. En cada problema se marca en negrita la mejor solución obtenida por las siete heurísticas. En los valores de costo y tiempo de ejecución se consideran dos cifras significativas.

Tabla 25: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias CVRP (I).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *Random* | | | *NN* | | | *SaveSeq* | | | *SaveParall* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 1702.14 | **0.034** |  | 1233.61 | 0.105 |  | 575.52 | 0.087 |  | 561.64 | 0.196 |
| 2 |  | 2392.85 | **0.047** |  | 1749.63 | 0.087 |  | 786.85 | 0.149 |  | 724.73 | 0.692 |
| 3 |  | 2310.81 | **0.032** |  | 632.28 | 0.123 |  | 671.75 | 0.137 |  | 631.85 | 0.690 |
| 4 |  | 3256.04 | **0.039** |  | 2049.58 | 0.133 |  | 859.02 | 0.175 |  | 762.48 | 1.644 |
| 5 |  | 58923.64 | **0.049** |  | 20534.06 | 1.400 |  | 6258.79 | 1.435 |  | 5374.23 | 233.309 |
| 6 |  | 14832.38 | **0.039** |  | 9341.01 | 0.361 |  | 2936.79 | 0.376 |  | 2629.57 | 22.909 |
| 7 |  | 42051.26 | **0.077** |  | 27979.17 | 1.437 |  | 7648.00 | 2.016 |  | 6871.46 | 377.605 |
| 8 |  | 87593.49 | **0.088** |  | 58727.53 | 6.434 |  | 17235.64 | 451.87 |  | 13678.65 | 1951.639 |

Tabla 26: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias CVRP (II).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *MJ* | | | *CMT* | | | *Sweep* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 611.59 | 0.244 |  | 854.59 | 0.114 |  | 1367.20 | 0.045 |
| 2 |  | 848.90 | 0.366 |  | 978.99 | 0.132 |  | 1851.70 | 0.079 |
| 3 |  | 664.63 | 0.482 |  | 1004.02 | 0.201 |  | 1740.60 | 0.057 |
| 4 |  | 835.15 | 0.867 |  | 1286.96 | 0.168 |  | 2158.63 | 0.068 |
| 5 |  | 6266.93 | 61.637 |  | 17953.30 | 1.182 |  | 70053.21 | 0.157 |
| 6 |  | 2947.41 | 1.732 |  | 3582.78 | 0.506 |  | 7745.33 | 0.081 |
| 7 |  | 7420.82 | 9.114 |  | 9773.22 | 3.424 |  | 22205.85 | 0.150 |
| 8 |  | 14948.77 | 41.346 |  | 19176.97 | 18.213 |  | 46104.44 | 0.361 |

Tabla 27: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias HFVRP (I).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *Random* | | | *NN* | | | *SaveSeq* | | | *SaveParall* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 1457.71 | **0.024** |  | 1001.56 | 0.077 |  | 527.10 | 0.059 |  | 536.53 | 0.270 |
| 2 |  | 2275.20 | **0.025** |  | 1411.39 | 0.080 |  | 652.98 | 0.093 |  | 851.96 | 1.181 |
| 3 |  | 2264.74 | **0.026** |  | 1429.92 | 0.106 |  | 662.28 | 0.094 |  | 879.66 | 2.701 |
| 4 |  | 3123.60 | **0.036** |  | 1962.89 | 0.266 |  | 774.38 | 0.409 |  | 1070.66 | 5.8528 |
| 5 |  | 61785.35 | **0.037** |  | 22736.22 | 1.033 |  | 6397.67 | 1.092 |  | 7470.81 | 3.8310 |
| 6 |  | 14755.53 | **0.091** |  | 9131.46 | 0.547 |  | 2588.6 | 0.673 |  | 1308.28 | 17.616 |
| 7 |  | 41402.19 | **0.123** |  | 26688.21 | 2.158 |  | 5597.72 | 2.527 |  | 5970.02 | 4.4937 |
| 8 |  | 87507.12 | **0.070** |  | 57878.02 | 7.989 |  | 15672.3 | 20.9242 |  | 977.43 | 7.9899 |

Tabla 28: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias HFVRP (II).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *MJ* | | | *CMT* | | | *Sweep* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 538.77 | 0.186 |  | 690.15 | 0.076 |  | 1280.31 | 0.019 |
| 2 |  | 686.02 | 0.358 |  | 972.90 | 0.102 |  | 1723.95 | 0.059 |
| 3 |  | 665.11 | 0.372 |  | 993.36 | 0.098 |  | 1731.74 | 0.037 |
| 4 |  | 784.41 | 4.810 |  | 1242.09 | 0.867 |  | 2123.70 | 0.087 |
| 5 |  | 6537.06 | 51.711 |  | 20324.56 | 1.245 |  | 70279.88 | 0.138 |
| 6 |  | 2714.86 | 2.892 |  | 3454.43 | 0.698 |  | 7904.14 | 0.221 |
| 7 |  | 5545.46 | 15.457 |  | 8646.55 | 2.866 |  | 21087.88 | 0.271 |
| 8 |  | 13952.11 | 50.595 |  | 22242.66 | 17.180 |  | 45417.50 | 0.448 |

Tabla 29: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias MDVRP (I).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *Random* | | | *NN* | | | *SaveSeq* | | | *SaveParall* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 957.80 | 0.032 |  | 826.44 | 0.037 |  | 646.41 | 0.048 |  | 626.2 | 0.058 |
| 2 |  | 2563.63 | **0.040** |  | 1501.49 | 0.082 |  | 901.40 | 0.115 |  | 833.09 | 0.324 |
| 3 |  | 8193.67 | **0.043** |  | 5110.09 | 0.132 |  | 3312.51 | 0.133 |  | 3302.04 | 0.280 |
| 4 |  | 18825.05 | **0.064** |  | 11864.58 | 0.211 |  | 7543.33 | 0.262 |  | 7429.57 | 0.583 |
| 5 |  | 10529.67 | **0.065** |  | 6387.67 | 0.189 |  | 3437.04 | 0.335 |  | 3016.17 | 1.888 |
| 6 |  | 14134.88 | **0.062** |  | 8066.29 | 0.160 |  | 4727.56 | 0.261 |  | 3993.25 | 1.241 |
| 7 |  | 902.52 | **0.029** |  | 740.36 | 0.032 |  | 529.71 | 0.048 |  | 528.52 | 0.059 |
| 8 |  | 5609.60 | **0.040** |  | 3621.49 | 0.081 |  | 2123.68 | 0.115 |  | 1951.45 | 0.226 |

Tabla 30: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias MDVRP (II).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *MJ* | | | *CMT* | | | *Sweep* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 639.01 | 0.062 |  | 759.23 | 0.044 |  | 850.86 | **0.030** |
| 2 |  | 894.12 | 0.216 |  | 1155.09 | 0.106 |  | 1691.91 | 0.052 |
| 3 |  | 3237.39 | 0.303 |  | 4288.36 | 0.141 |  | 6106.68 | 0.059 |
| 4 |  | 7284.11 | 0.995 |  | 10024.89 | 0.308 |  | 12862.69 | 0.077 |
| 5 |  | 3287.15 | 1.320 |  | 4602.53 | 0.374 |  | 7632.53 | 0.078 |
| 6 |  | 4343.88 | 0.701 |  | 5815.21 | 0.240 |  | 9156.59 | 0.101 |
| 7 |  | 526.21 | 0.065 |  | 643.19 | 0.057 |  | 773.11 | 0.033 |
| 8 |  | 2089.05 | 0.348 |  | 2879.68 | 0.143 |  | 4110.23 | 0.051 |

Tabla 31: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias TTRP (I).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *Random* | | | *NN* | | | *SaveSeq* | | | *SaveParall* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 1627.30 | **0.052** |  | 1259.35 | 0.078 |  | 602.8 | 0.117 |  | 765.25 | 0.196 |
| 2 |  | 2672.96 | **0.127** |  | 1877.47 | 0.256 |  | 1033.49 | 0.273 |  | 1820.58 | 1.4296 |
| 3 |  | 3423.11 | **0.051** |  | 2321.50 | 0.231 |  | 1106.50 | 0.391 |  | 1879.66 | 2.701 |
| 4 |  | 6371.67 | **0.072** |  | 2152.60 | 0.368 |  | 998.48 | 0.411 |  | 1070.66 | 5.8528 |
| 5 |  | 4085.60 | **0.086** |  | 1733.76 | 0.316 |  | 1101.46 | 0.323 |  | 1747.81 | 3.8310 |
| 6 |  | 6679.67 | **0.066** |  | 4553.98 | 1.191 |  | 1888.82 | 1.138 |  | 2308.28 | 17.616 |
| 7 |  | 1632.47 | **0.040** |  | 1242.58 | 0.104 |  | 782.74 | 0.128 |  | 1597.02 | 0.4937 |
| 8 |  | 5140.10 | **0.056** |  | 3440.13 | 0.618 |  | 1166.62 | 0.613 |  | 1977.43 | 7.9899 |

Tabla 32: Resumen de los resultados obtenidos por BHCVRP [39] en las ocho instancias TTRP (II).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problema | *MJ* | | | *CMT* | | | *Sweep* | | |
| Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) | Min | Prom | Tiempo (s) |
| 1 |  | 793.95 | 0.236 |  | 934.13 | 0.217 |  | 1535.37 | 0.2584 |
| 2 |  | 1269.96 | 0.455 |  | 1417.89 | 0.421 |  | 2529.72 | 0.7926 |
| 3 |  | 1185.81 | 1.049 |  | 1423.57 | 0.326 |  | 3323.73 | 1.8182 |
| 4 |  | 1646.97 | 2.480 |  | 1835.43 | 0.524 |  | 6381.68 | 3.0313 |
| 5 |  | 1202.0 | 1.071 |  | 1442.07 | 0.439 |  | 3873.64 | 1.8079 |
| 6 |  | 1945.89 | 5.621 |  | 2353.39 | 1.770 |  | 6634.31 | 12.2426 |
| 7 |  | 874.40 | 0.255 |  | 917.22 | 0.150 |  | 1601.22 | 0.2534 |
| 8 |  | 1603.21 | 3.825 |  | 1986.04 | 0.741 |  | 4944.19 | 5.7698 |

A partir de los resultados alcanzados y aplicando la prueba estadística no paramétrica de Friedman [100], se obtiene que la heurística *SaveParall* es la que da mejores resultados para las variantes CVRP, MDVRP y TTRP, mientras que para HFVRP es la heurística *SaveSeq*. Por otra parte, las heurísticas *Random* y *Sweep* obtienen las soluciones más costosas.

Después de las modificaciones realizadas, se desea comparar el comportamiento de las heurísticas implementadas en la versión anterior de BHCVRP. Con el objetivo de lograr una comparación detallada, en las Figura 26, Figura 27, Figura 28 y Figura 29, se muestra una gráfica que refleja el comportamiento por cada par de heurísticas (actual y anterior) en cada variante VRP.

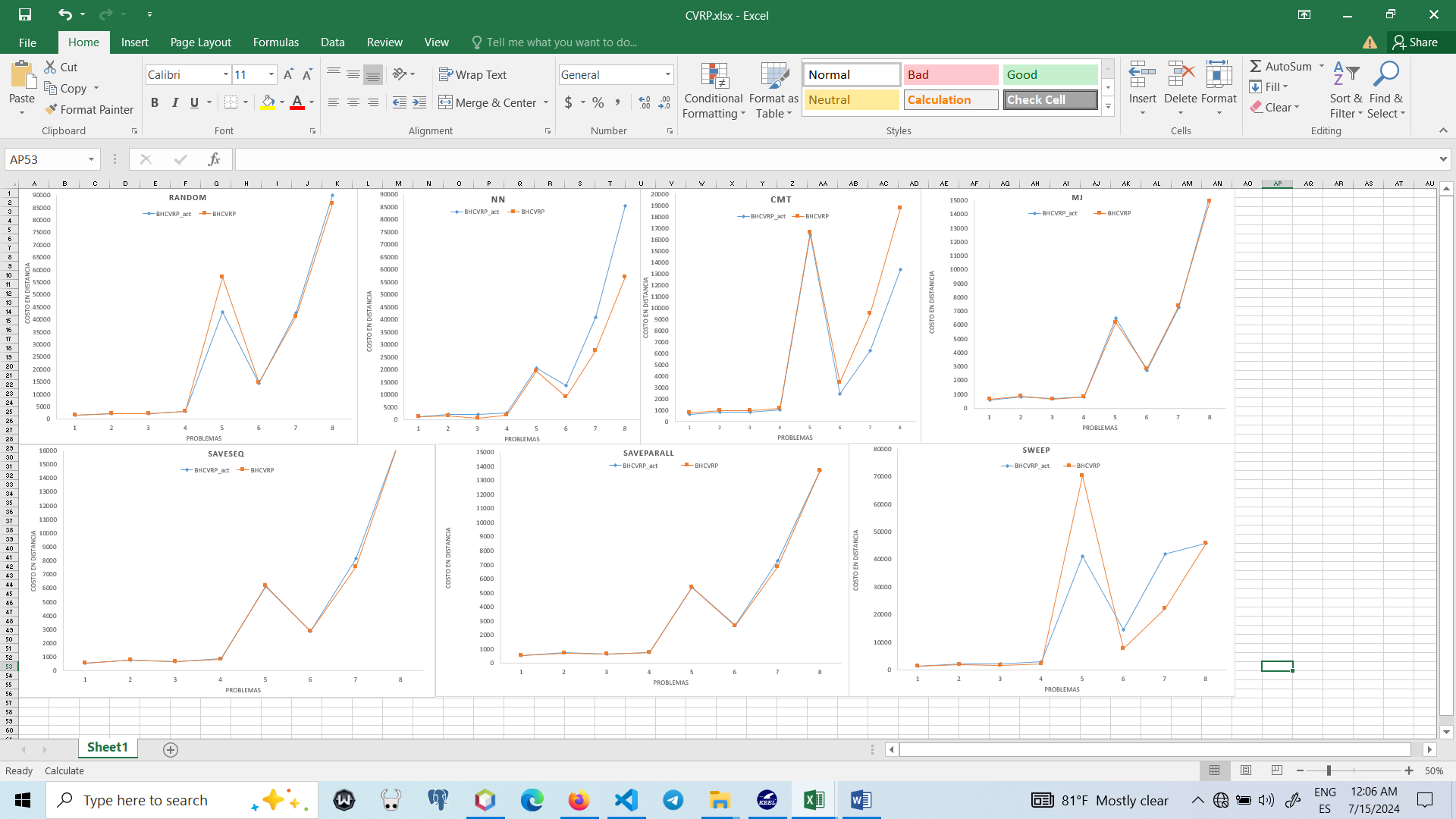


Figura 26: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en distancia de las soluciones para la variante CVRP.

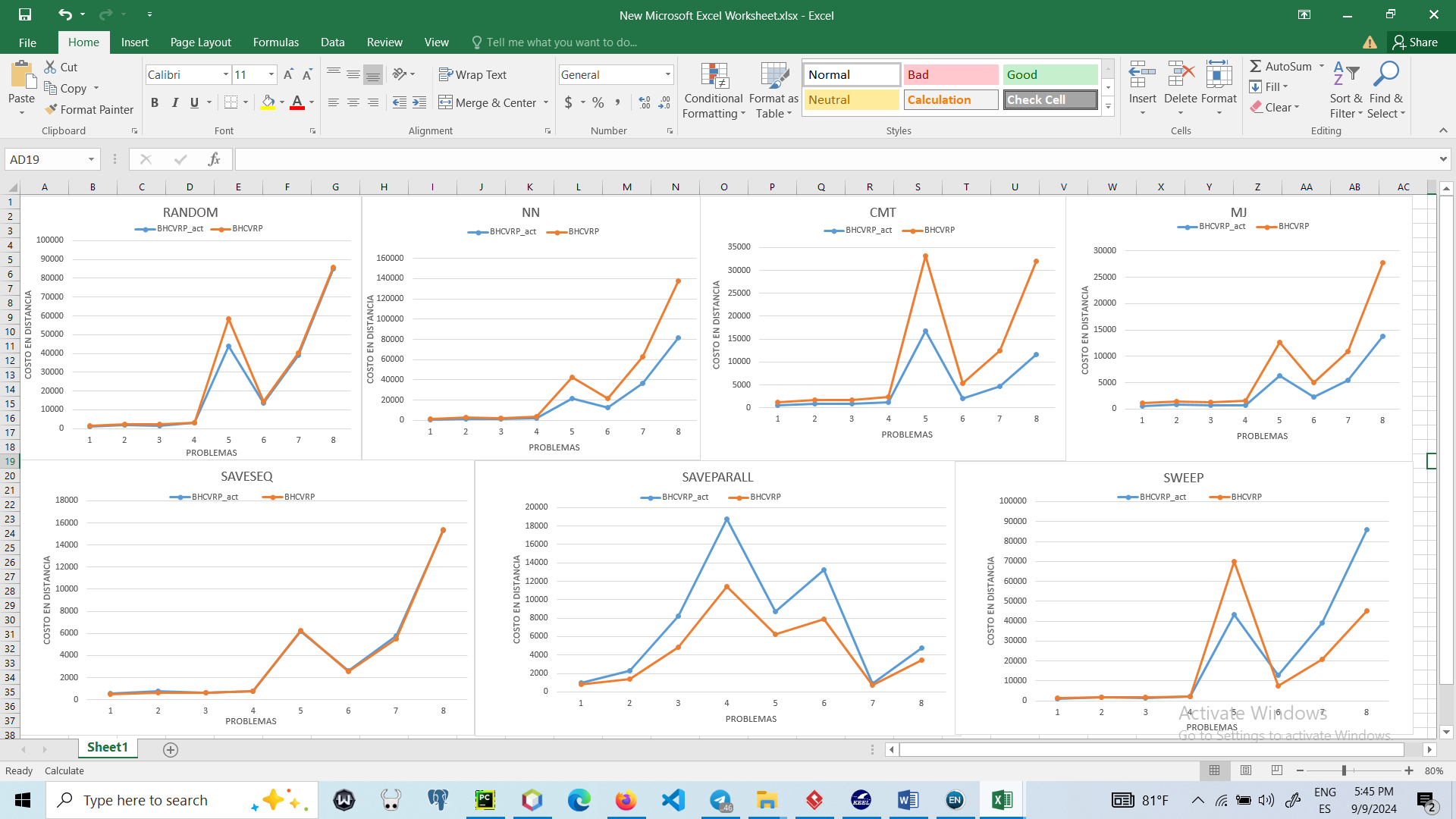


Figura 27: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en distancia de las soluciones para la variante HFVRP.

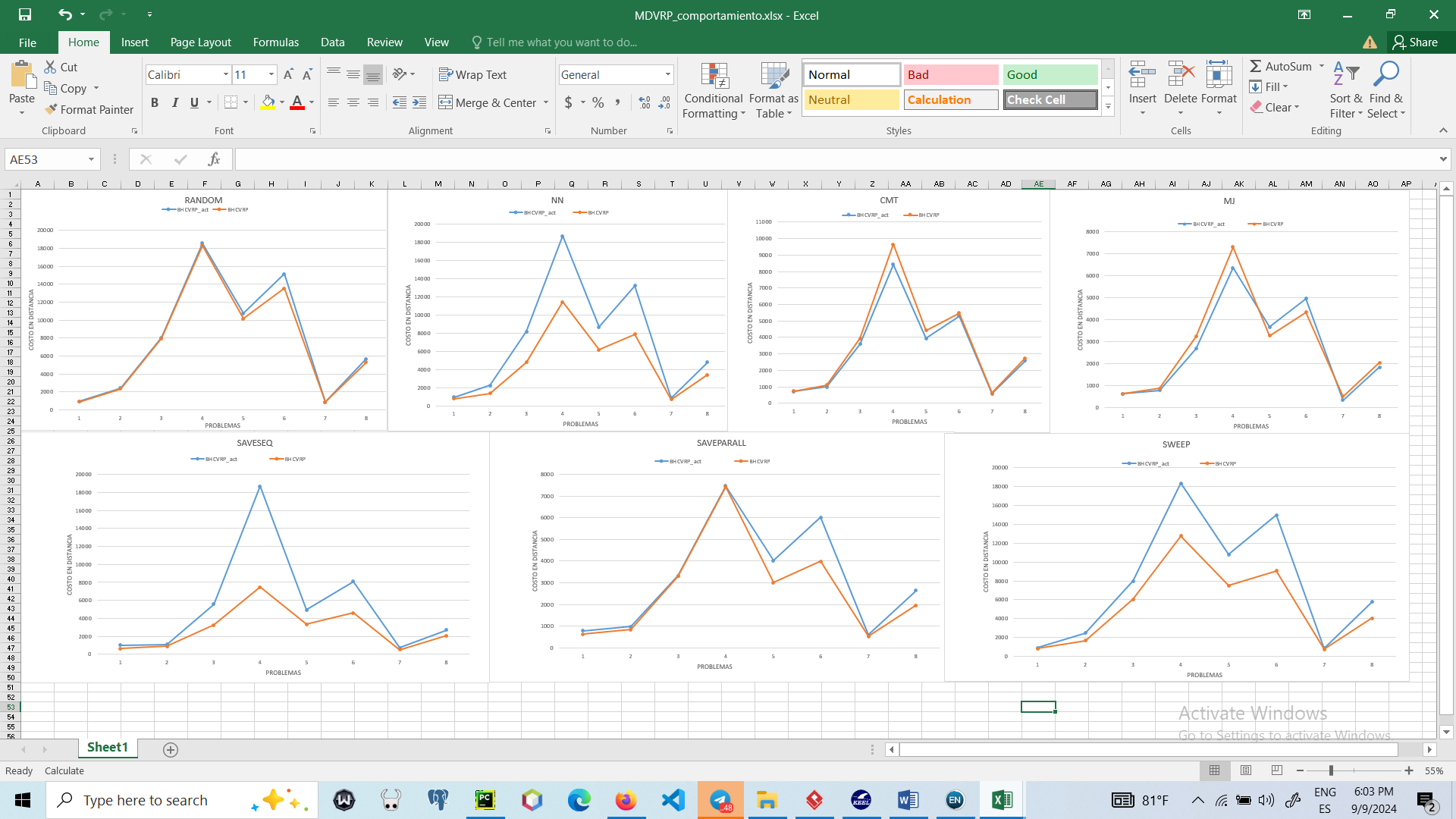


Figura 28: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en distancia de las soluciones para la variante MDVRP.

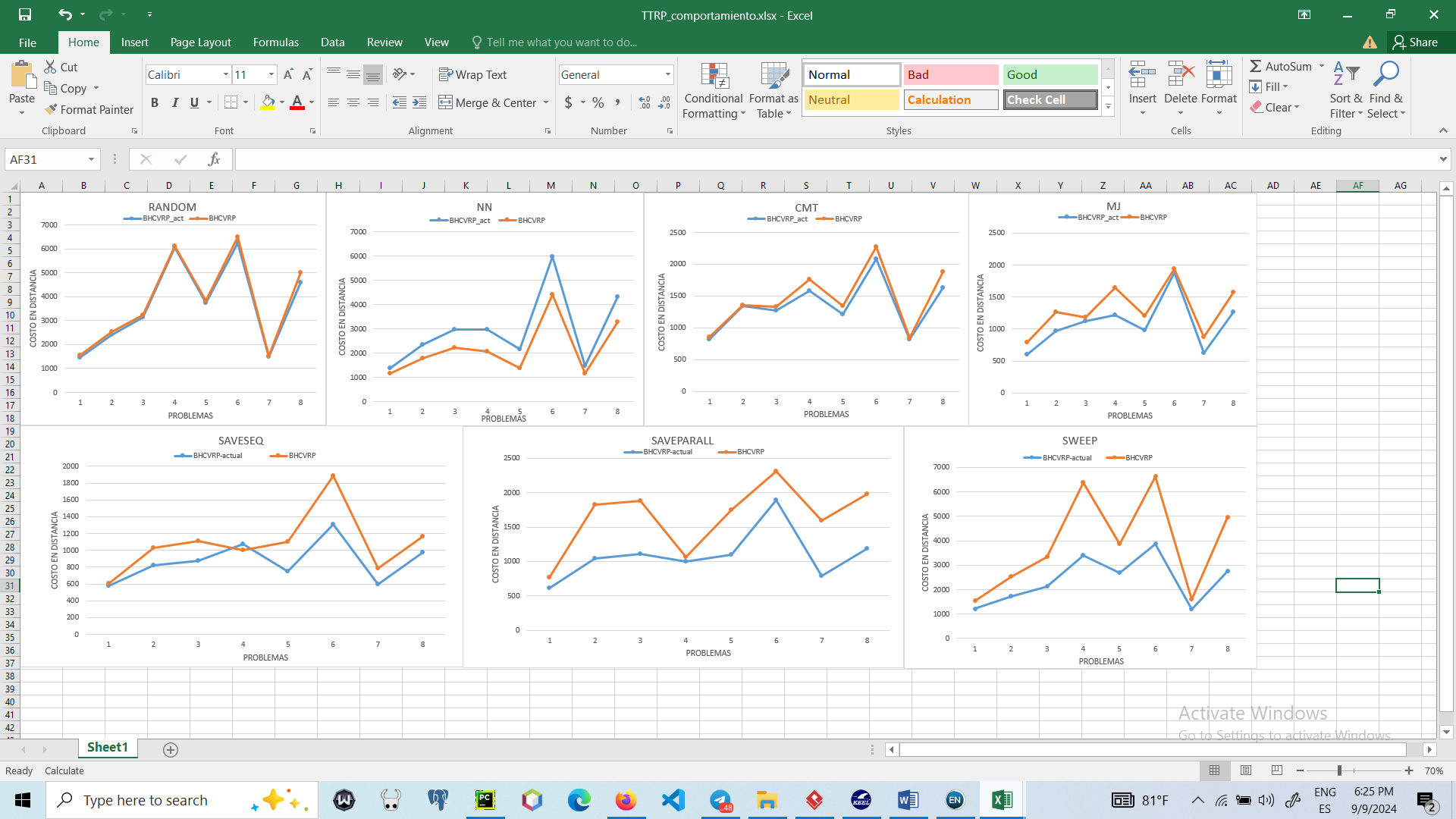


Figura 29: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en distancia de las soluciones para la variante TTRP.

Con el objetivo de lograr un mayor grado de certeza en la comparación de los algoritmos ya existentes en BHCVRP, se decide aplicar la prueba estadística no paramétrica de Wilcoxon [101]. Esta prueba se utiliza para determinar si hay diferencias significativas entre los resultados alcanzados por cada par de algoritmos.

Tabla 33: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a costo para la variante CVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *p-value* | *p-value* asintótico |
| *Random\_act vs Random*  *Random vs Random\_act* | 14.0  14.0 | 14.0  14.0 | ≥0.2  ≥0.2 | 0.932647  0.932647 |
| *Sweep\_act vs Sweep*  *Sweep vs Sweep\_act* | 7.0  21.0 | 21.0  7.0 | ≥0.2  ≥0.2 | 1  0.204894 |
| *SaveSeq\_act vs SaveSeq*  *SaveSeq vs SaveSeq\_act* | 3.0  25.0 | 25.0  3.0 | ≥0.2  ≥0.2 | 1  0.051913 |
| *SaveParall\_act vs SaveParall*  *SaveParall vs SaveParall\_act* | 4.0  24.0 | 24.0  4.0 | ≥0.2  ≥0.2 | 1  0.075927 |
| *MJ\_act vs MJ*  *MJ vs MJ\_act* | 20.0  16.0 | 16.0  20.0 | ≥0.2  ≥0.2 | 0.726286  1 |
| *NN\_act vs NN*  *NN vs NN\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *CMT\_act vs CMT*  *CMT vs CMT\_act* | 36.0  0.0 | 0.0  36.0 | ≥0.2  ≥0.2 | 0.009583  1 |

Los resultados obtenidos en la Tabla 33 muestran que las heurísticas *Random, Sweep, SaveSeq, SaveParall* y *MJ* tienen un comportamiento similar en ambas versiones de la biblioteca, ya que no presentan diferencias significativas en sus respectivos *p-values.* Sin embargo, la heurística *NN* obtiene un peor rendimiento en cuanto a costo de las soluciones y la heurística *CMT* obtuvo mejoras relevantes.

Tabla 34: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a costo para la variante HFVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *p-value* | *p-value* asintótico |
| *Random\_act vs Random*  *Random vs Random\_act* | 28.0  0.0 | 0.0  28.0 | ≥0.2  ≥0.2 | 0.014248  1 |
| *Sweep\_act vs Sweep*  *Sweep vs Sweep\_act* | 11.0  17.0 | 17.0  11.0 | ≥0.2  ≥0.2 | 1  0.554113 |
| *SaveSeq\_act vs SaveSeq*  *SaveSeq vs SaveSeq\_act* | 9.0  19.0 | 19.0  9.0 | ≥0.2  ≥0.2 | 1  0.352542 |
| *SaveParall\_act vs SaveParall*  *SaveParall vs SaveParall\_act* | 14.0  14.0 | 14.0  14.0 | ≥0.2  ≥0.2 | 0.932647  0.932647 |
| *MJ\_act vs MJ*  *MJ vs MJ\_act* | 18.0  10.0 | 10.0  18.0 | ≥0.2  ≥0.2 | 0.446873  1 |
| *NN\_act vs NN*  *NN vs NN\_act* | 3.0  25.0 | 25.0  3.0 | ≥0.2  ≥0.2 | 1  0.051913 |
| *CMT\_act vs CMT*  *CMT vs CMT\_act* | 24.0  4.0 | 4.0  24.0 | ≥0.2  ≥0.2 | 0.075927  1 |

Los resultados obtenidos en la Tabla 34 muestran que todas las heurísticas, excepto *Random,* tienen un comportamiento similar en ambas versiones de la biblioteca, ya que no presentan diferencias significativas en sus respectivos *p-values.*

Tabla 35: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a costo para la variante MDVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *p-value* | *p-value* asintótico |
| *Random\_act vs Random*  *Random vs Random\_act* | 1.0  35.0 | 35.0  1.0 | ≥0.2  ≥0.2 | 1  0.014266 |
| *Sweep\_act vs Sweep*  *Sweep vs Sweep\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *SaveSeq\_act vs SaveSeq*  *SaveSeq vs SaveSeq\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *SaveParall\_act vs SaveParall*  *SaveParall vs SaveParall\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *MJ\_act vs MJ*  *MJ vs MJ\_act* | 23.0  13.0 | 13.0  23.0 | ≥0.2  ≥0.2 | 0.441209  1 |
| *NN\_act vs NN*  *NN vs NN\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *CMT\_act vs CMT*  *CMT vs CMT\_act* | 36.0  0.0 | 0.0  36.0 | ≥0.2  ≥0.2 | 0.009583  1 |

Los resultados obtenidos en la Tabla 35 muestran que la heurística *MJ* es la única que permace con un comportamiento similar en ambas versiones de la biblioteca, ya que no presenta diferencias significativas en sus respectivos *p-values.* La heurística *CMT* obtiene un mejor rendimiento en cuanto a costo de las soluciones, mientras que el resto poseen un rendimiento desfavorable.

Tabla 36: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a costo para la variante TTRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *p-value* | *p-value* asintótico |
| *Random\_act vs Random*  *Random vs Random\_act* | 36.0  0.0 | 0.0  36.0 | ≥0.2  ≥0.2 | 0.009583  1 |
| *Sweep\_act vs Sweep*  *Sweep vs Sweep\_act* | 23.0  13.0 | 13.0  23.0 | ≥0.2  ≥0.2 | 0.441209  1 |
| *SaveSeq\_act vs SaveSeq*  *SaveSeq vs SaveSeq\_act* | 11.0  17.0 | 17.0  11.0 | ≥0.2  ≥0.2 | 1  0.554113 |
| *SaveParall\_act vs SaveParall*  *SaveParall vs SaveParall\_act* | 9.0  19.0 | 19.0  9.0 | ≥0.2  ≥0.2 | 1  0.352542 |
| *MJ\_act vs MJ*  *MJ vs MJ\_act* | 36.0  0.0 | 0.0  36.0 | ≥0.2  ≥0.2 | 0.009583  1 |
| *NN\_act vs NN*  *NN vs NN\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *CMT\_act vs CMT*  *CMT vs CMT\_act* | 36.0  0.0 | 0.0  36.0 | ≥0.2  ≥0.2 | 0.009583  1 |

Los resultados obtenidos en la Tabla 36 muestran que las heurísticas *Sweep, SaveSeq* y *SaveParall* tienen un comportamiento similar en ambas versiones de la biblioteca, ya que no presentan diferencias significativas en sus respectivos *p-values.* Sin embargo, la heurística *NN* obtiene un peor rendimiento en cuanto a costo de las soluciones y el restoobtuvo mejoras relevantes.

En cuanto al tiempo de ejecución, se muestra un ejemplo en la Figura 30, con el rendimiento por cada par de heurísticas (actual y anterior) en la variante CVRP. Como se puede apreciar en los siete casos, el tiempo presenta diferencias significativas destacándose la versión anterior de BHCVRP con los mejores resultados. Para el resto de las variantes el comportamiento es similar al descrito.

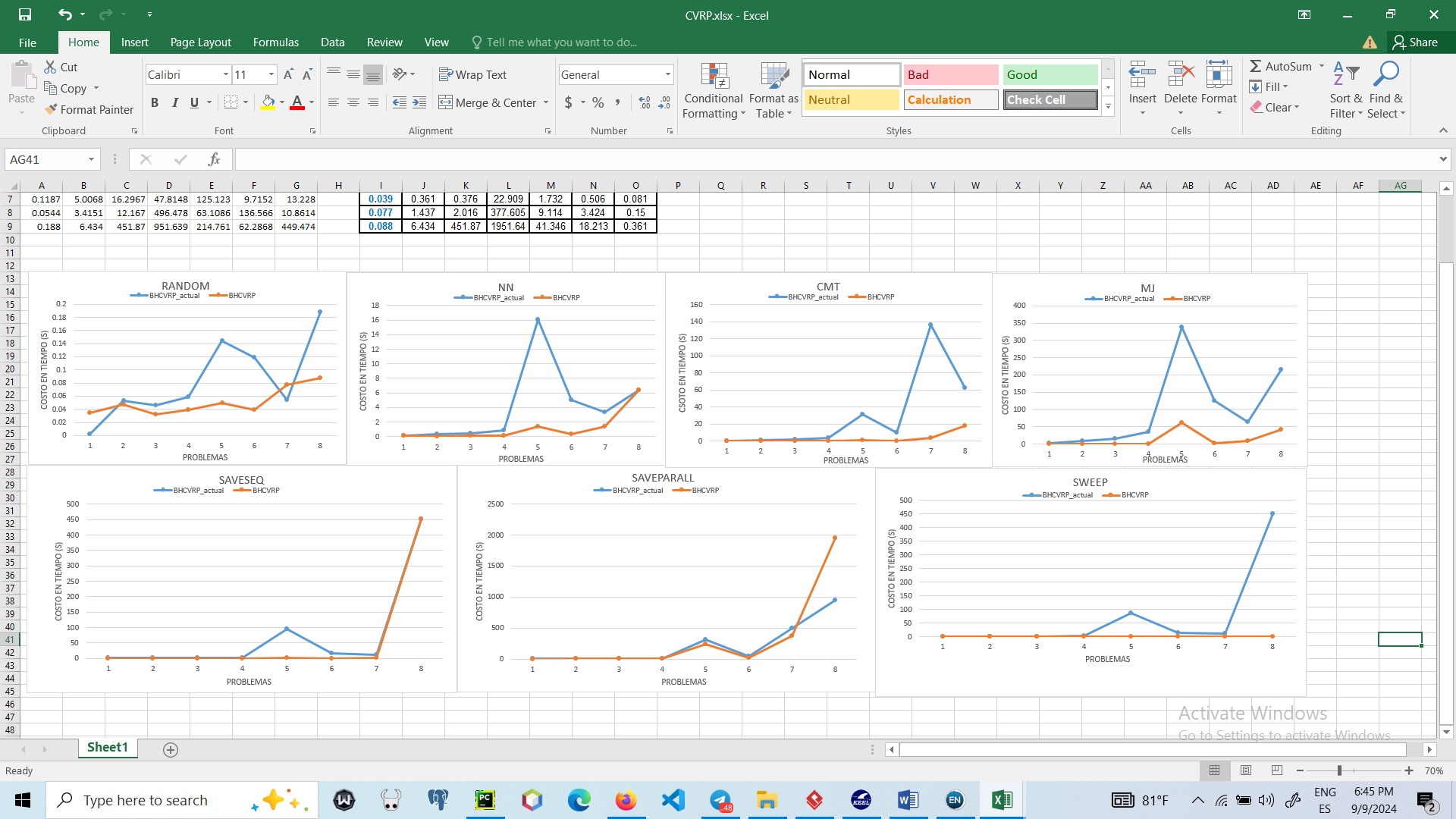


Figura 30: Comparación del comportamiento de las heurísticas de BHCVRP en ambas versiones en cuanto al costo en tiempo de las soluciones para la variante CVRP.

Por último, se decide aplicar la prueba estadística no paramétrica de Wilcoxon [101] (ver Tabla 37, Tabla 38, Tabla 39 y Tabla 40) para comparar los tiempos de las ocho instancias para cada variante VRP en ambas versiones de BHCVRP.

Tabla 37: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a tiempo para la variante CVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *p-value* | *p-value* asintótico |
| *Random\_act vs Random*  *Random vs Random\_act* | 9.0  27.0 | 27.0  9.0 | ≥0.2  ≥0.2 | 1  0.183431 |
| *Sweep\_act vs Sweep*  *Sweep vs Sweep\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *SaveSeq\_act vs SaveSeq*  *SaveSeq vs SaveSeq\_act* | 0.0  28.0 | 28.0  0.0 | ≥0.2  ≥0.2 | 1  0.014248 |
| *SaveParall\_act vs SaveParall*  *SaveParall vs SaveParall\_act* | 8.0  28.0 | 28.0  8.0 | ≥0.2  ≥0.2 | 1  0.141482 |
| *MJ\_act vs MJ*  *MJ vs MJ\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *NN\_act vs NN*  *NN vs NN\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *CMT\_act vs CMT*  *CMT vs CMT\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |

Tabla 38: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a tiempo para la variante HFVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *p-value* | *p-value* asintótico |
| *Random\_act vs Random*  *Random vs Random\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *Sweep\_act vs Sweep*  *Sweep vs Sweep\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *SaveSeq\_act vs SaveSeq*  *SaveSeq vs SaveSeq\_act* | 0.0  28.0 | 28.0  0.0 | ≥0.2  ≥0.2 | 1  0.014248 |
| *SaveParall\_act vs SaveParall*  *SaveParall vs SaveParall\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *MJ\_act vs MJ*  *MJ vs MJ\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *NN\_act vs NN*  *NN vs NN\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *CMT\_act vs CMT*  *CMT vs CMT\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |

Tabla 39: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a tiempo para la variante MDVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *p-value* | *p-value* asintótico |
| *Random\_act vs Random*  *Random vs Random\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *Sweep\_act vs Sweep*  *Sweep vs Sweep\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *SaveSeq\_act vs SaveSeq*  *SaveSeq vs SaveSeq\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *SaveParall\_act vs SaveParall*  *SaveParall vs SaveParall\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *MJ\_act vs MJ*  *MJ vs MJ\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *NN\_act vs NN*  *NN vs NN\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *CMT\_act vs CMT*  *CMT vs CMT\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |

Tabla 40: Resultados de la prueba de Wilcoxon en la comparación con los algoritmos de BHCVRP [39] en cuanto a tiempo para la variante TTRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *p-value* | *p-value* asintótico |
| *Random\_act vs Random*  *Random vs Random\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *Sweep\_act vs Sweep*  *Sweep vs Sweep\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *SaveSeq\_act vs SaveSeq*  *SaveSeq vs SaveSeq\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *SaveParall\_act vs SaveParall*  *SaveParall vs SaveParall\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *MJ\_act vs MJ*  *MJ vs MJ\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *NN\_act vs NN*  *NN vs NN\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| *CMT\_act vs CMT*  *CMT vs CMT\_act* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |

A partir de los resultados alcanzados se puede arribar a las siguientes conclusiones: existen diferencias significativas ya que, para todas las variantes en la mayoría de las heurísticas, la versión desarrollada en Java obtiene mejores tiempos que la versión de Python.

### **3.3.3 Comparación de los resultados con OR-Tools**

Debido a la relevancia de la herramienta OR-Tools, sobre todo para la comunidad en Python, se decide realizar un experimento para determinar la calidad de las soluciones en cuanto a costos y tiempo de ejecución. En las Tabla 41 y Tabla 42, se resumen los mejores resultados en la versión de BHCVRP actual y OR-Tools, para las variantes CVRP y HFVRP. En cada problema se marca en negrita color azul la mejor solución obtenida y el menor tiempo de ejecución. En los valores de costo y tiempo de ejecución se consideran dos cifras significativas.

Tabla 41: Resumen de los resultados obtenidos por BHCVRP y OR-Tools en las ocho instancias CVRP.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problema | BHCVRP | | | OR-Tools | | |
| Costo en distancia | Total de rutas | Tiempo de ejecución (s) | Costo en distancia | Total de rutas | Tiempo de ejecución (s) |
| 1 | **558.89** | 4 | 0.54 | 2152.00 | 4 | **0.13** |
| 2 | **787.61** | 6 | 1.25 | 3916.00 | 6 | **0.33** |
| 3 | **666.86** | 3 | 15.75 | 4133.00 | 3 | **0.27** |
| 4 | **814.49** | 4 | 36.10 | 4534.00 | 4 | **0.57** |
| 5 | **6134.72** | 4 | 94.80 | 9298.00 | 4 | **15.00** |
| 6 | **2391.66** | 18 | 9.71 | 2597.00 | 19 | **1.61** |
| 7 | **6239.59** | 36 | 136.57 | 6970.00 | 37 | **9.12** |
| 8 | **13386.91** | 56 | 62.29 | 13881.00 | 59 | **15.00** |

Tabla 42: Resumen de los resultados obtenidos por BHCVRP y OR-Tools en las ocho instancias HFVRP.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problema | BHCVRP | | | OR-Tools | | |
| Costo en distancia | Total de rutas | Tiempo de ejecución (s) | Costo en distancia | Total de rutas | Tiempo de ejecución (s) |
| 1 | **550.34** | 4 | 0.48 | 3225.00 | 4 | **0.14** |
| 2 | **799.87** | 6 | 9.36 | 3730.00 | 4 | **0.30** |
| 3 | **666.31** | 3 | 1.35 | 4030.00 | 3 | **0.40** |
| 4 | **768.28** | 3 | 2.66 | 1512.00 | 3 | **0.71** |
| 5 | **6125.32** | 4 | 24.61 | 9327.00 | 4 | **12.03** |
| 6 | **2118.76** | 15 | 19.99 | 2597.00 | 19 | **1.61** |
| 7 | **4705.29** | 23 | 121.95 | 6970.00 | 37 | **9.12** |
| 8 | **11564.10** | 51 | 1003.68 | 13881.00 | 59 | **15.00** |

Con el objetivo de lograr un mayor grado de certeza en la comparación de los resultados obtenidos por BHCVRP y OR-Tools, se decide aplicar la prueba estadística no paramétrica de Wilcoxon [101] como se aprecia en la Tabla 43. Esta prueba se utiliza para determinar si hay diferencias significativas entre los resultados alcanzados por cada componente de software.

Tabla 43: Resultados de la prueba de Wilcoxon en la comparación con los resultados de BHCVRP y OR-Tools en cuanto a costo para las variantes CVRP y HFVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *pvalue* | *p-value* asintótico |
| Variante CVRP |  |  |  |  |
| *BHCVRP vs OR-Tools*  *OR-Tools vs BHCVRP* | 36.0  0.0 | 0.0  36.0 | ≥0.2  ≥0.2 | 0.009583  1 |
| Variante HFVRP |  |  |  |  |
| *BHCVRP vs OR-Tools*  *OR-Tools vs BHCVRP* | 36.0  0.0 | 0.0  36.0 | ≥0.2  ≥0.2 | 0.009583  1 |

Los resultados obtenidos con la prueba de Wilcoxon muestran diferencias significativas en cuanto al costo en distancia entre ambos componentes de software. Se obtienen notablemente mejores soluciones con BHCVRP.

En cuanto al tiempo de ejecución, en las gráficas de las Figura 31 y Figura 32, se muestra el rendimiento por cada par de soluciones (BHCVRP y OR-Tools). Como se puede apreciar en los dos casos, OR-Tools obtiene mejores tiempos de ejecución.



Figura 31: Comportamiento de las soluciones obtenidas por BHCVRP y OR-Tools en cuanto a tiempo de ejecución para la variante CVRP.

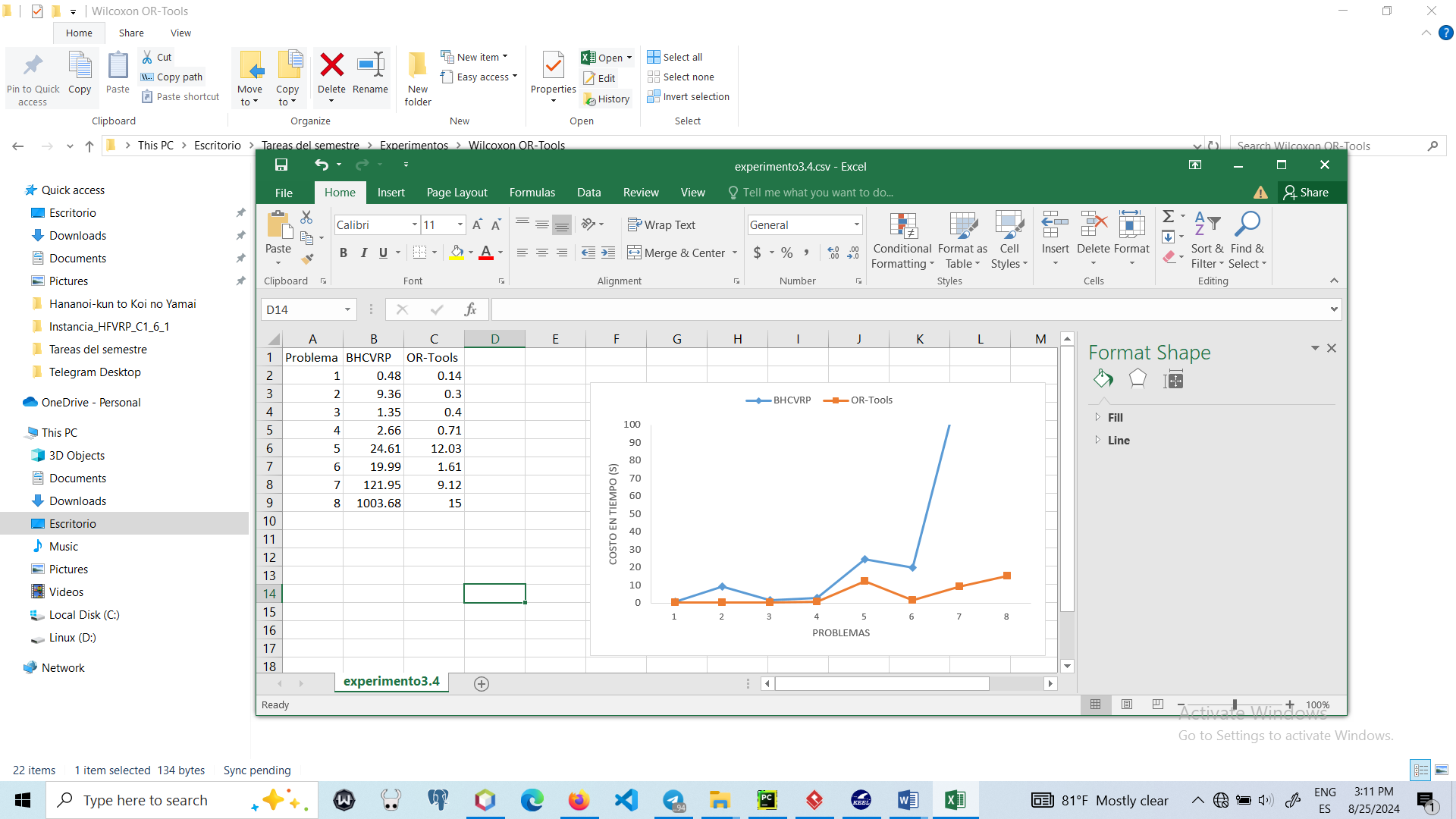


Figura 32: Comportamiento de las soluciones obtenidas por BHCVRP y OR-Tools en cuanto a tiempo de ejecución para la variante HFVRP.

Por último, se decide aplicar la prueba estadística no paramétrica de Wilcoxon [101] (ver Tabla 44) para comparar los tiempos promedios de las ocho instancias para CVRP y HFVRP, en ambas herramientas.

Tabla 44: Resultados de la prueba de Wilcoxon en la comparación con los resultados de BHCVRP y OR-Tools en cuanto a tiempo de ejecución para las variantes CVRP y HFVRP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hipótesis | R+ | R- | *pvalue* | *p-value* asintótico |
| Variante CVRP |  |  |  |  |
| *BHCVRP vs OR-Tools*  *OR-Tools vs BHCVRP* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |
| Variante HFVRP |  |  |  |  |
| *BHCVRP vs OR-Tools*  *OR-Tools vs BHCVRP* | 0.0  36.0 | 36.0  0.0 | ≥0.2  ≥0.2 | 1  0.009583 |

A partir de los resultados alcanzados se puede arribar a las siguientes conclusiones: existen diferencias significativas en cuanto a tiempo de ejecución para ambas variantes. La herramienta OR-Tools obtiene los resultados en considerablemente menor tiempo que BHCVRP.

## **3.4 Conclusiones parciales**

Después de la realización de los experimentos definidos previamente con el objetivo de analizar las heurísticas de construcción y su comportamiento en cuatro variantes de VRP, se obtienen las siguientes conclusiones:

* Los resultados obtenidos para las variantes CVRP y TTRP sugieren que las heurísticas que alcanzan mejores soluciones son *SaveParall* y *MJ*. Sin embargo, para la variante HFVRP los mejores resultados son con las heurísticas *MJ* y *SaveSeq,* y para la variante MDVRP son *MJ* y *CMT.*
* Las pruebas estadísticas realizadas confirman que para las variantes CVRP y TTRP, la heurística *SaveParall* es la de mejor rendimiento y para las variante HFVRP y MDVRP es más recomendable *MJ*.
* En cuanto al parámetro tiempo, se evidencia que las heurísticas *Kilby, SaveMatch* y *MJ* son las que mayor cantidad de tiempo consumen, solo esta última obteniendo buenos resultados. Sin embargo, la heurística *Random* obtiene tiempos pequeños de ejecución alcanzando los peores resultados en todos los casos.
* En la comparación realizada con la versión anterior de BHCVRP, se puede apreciar que casi todas las heurísticas mantienen un comportamiento similar para la variante CVRP. En el caso de la heurística *CMT* se evidencia mejor rendimiento en la versión actual, mientras que para la heurística *NN* es desfavorable.
* En la comparación realizada con la versión anterior de BHCVRP, se puede apreciar que casi todas las heurísticas mantienen un comportamiento similar para la variante HFVRP, excepto *Random.*
* En la comparación realizada con la versión anterior de BHCVRP, se puede apreciar que casi todas las heurísticas tienen un comportamiento desfavorable para la variante MDVRP. En el caso de la heurística *CMT* se evidencia mejor rendimiento en la versión actual, mientras que la heurística *MJ* mantiene un comportamiento similar.
* En la comparación realizada con la versión anterior de BHCVRP, se puede apreciar que las heurísticas *MJ, Random* y *CMT* obtienen un mejor comportamiento para la variante TTRP. En el caso de las heurística *Sweep, SaveSeq* y *SaveParall* se evidencia un comportamiento similar en ambas versiones, mientras que para la heurística *NN* es desfavorable.
* En la comparación realizada con la herramienta OR-Tools, se evidencia que BHCVRP obtuvo absolutamente mejores resultados en cuanto a costo en distancia, aunque OR-Tools presentó menores tiempo de ejecución para todos los casos.

# **Conclusiones**

Después de dar cumplimiento a los objetivos trazados en este trabajo y a partir de los resultados alcanzados se puede concluir que los Problemas de Planificación de Rutas de Vehículos son problemas de optimización combinatoria y una buena alternativa para solucionarlos son los algoritmos heurísticos y metaheurísticos. Las heurísticas de construcción presentan buenos resultados para estos problemas y son aplicables como punto de partida en el contexto de las metaheurísticas. Actualmente, existen escasos componentes de software que implementen métodos heurísticos para resolver variantes de VRP. De los componentes de software estudiados solo BICIAM, VRPH, BHCVRP, JSprit, VROOM y OR-Tools resuelven problemas de planificación de rutas de vehículos. Sin embargo, solo Hipster y BHCVRP implementan heurísticas de construcción exclusivamente. Es importante destacar que este último resuelve solamente las variantes CVRP, MDVRP, HFVRP y TTRP.

La nueva versión de BHCVRP resuelve algunas de las deficiencias encontradas en la versión anterior. Se incorporan dos nuevas heurísticas de construcción: el Algoritmo de Ahorro basado en *Matching* y la Heurística de Inserción de Kilby. Además, se plasman un conjunto de mejoras en la arquitectura y funcionamiento de la biblioteca como son: la creación de la primera versión que cubre los lenguajes de programación Python y Java, el tratamiento de excepciones propias relacionadas con la distancia, capacidad de los vehículos, velocidad y tiempo de ejecución, así como la aplicación de principios de diseño como encapsulación de la variabilidad, *Hollywood*, cohesión y acoplamiento, *Open/Close* y *Don´t Repeat Yourself*, y el patrón de diseño *Template*.

Los resultados alcanzados en los experimentos haciendo uso de la prueba estadística de Friedman permite afirmar que en las variantes CVRP y TTRP, la heurística *SaveParall* es la de mejor rendimiento. Por otra parte, para las variantes HFVRP y MDVRP, la heurística más apropiada es *MJ.* En cuanto al tiempo de ejecución, se evidencia que la heurística *MJ* es la que emplea mayor cantidad, pero alcanza mejores resultados. Sin embargo, el Método Aleatorio obtiene un tiempo de ejecución pequeño con los peores resultados para todos los casos. A partir de la comparación de las dos versiones de BHCVRP se aprecia que la mayoría de las heurísticas mantienen un comportamiento similar en las cuatro variantes VRP. Aunque, es importante destacar que la heurística *NN* de manera general fue desfavorable y *CMT* mejoró su rendimiento. Por último, en el experimento comparativo realizado con la herramienta OR-Tools, BHCVRP obtiene mejores resultados en cuanto a costo de las soluciones a pesar de tomar mayor cantidad de tiempo.

# **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:

* Añadir las modificaciones realizadas en cuanto a arquitectura y código fuente, en la versión de BHCVRP en Python, a la versión existente en Java.
* Incorporar otras variantes de VRP que reflejen nuevas características de la vida real. Por ejemplo, SBRP, OVRP, VRPTW, entre otras.
* Extender la biblioteca a partir de la incorporación de nuevas heurísticas de construcción, como puede ser la Heurística de Asignación Generalizada de Fisher y Jaikumar, el Algoritmo de Pétalos, entre otras.
* Con el objetivo de proporcionar mayor flexibilidad a la biblioteca permitir al usuario el uso de pasos de post-optimización a su conveniencia.
* Incorporar nuevas excepciones propias, por ejemplo, para las heurísticas que poseen parámetros como las versiones del Algoritmo de Ahorros, la heurística del vecino más cercano con lista de candidatos restringidos y las heurísticas de inserción *Mole & Jameson, CMT* y *Kilby.*
* Respecto a la variante MDVRP sería conveniente incorporar nuevos métodos de asignación y admitir por parte del usuario una asignación predeterminada.
* Mejorar el funcionamiento de heurísticas como *NN, Kilby* y *SaveMatch,* así como los resultados para la variante MDVRP*.* Además, incorporar experimentos con el resto de las configuraciones que brindan las variante HFVRP (orden *Ascending* y *Random*) y MDVRP (todas las asignaciones que brinda BHAVRP, excepto *BestNearest*).
* Realizar nuevos experimentos y compararlos con los resultados publicados en la literatura.

# **Referencias bibliográficas**

[1] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[2] P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

[3] B. Eksioglu, A. V. Vural, and A. Reisman, "The vehicle routing problem: A taxonomic review," *Computers & Industrial Engineering,* vol. 57, no. 4, pp. 1472-1483, 2009.

[4] S. N. Kumar and R. Panneerselvam, "A survey on the vehicle routing problem and its variants," 2012.

[5] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science,* vol. 6, no. 1, pp. 80-91, 1959.

[6] D. L. Applegate, *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[7] G. Clarke and 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.

[8] M. Zirour, "Vehicle routing problem: models and solutions," *Journal of Quality Measurement and Analysis JQMA,* vol. 4, no. 1, pp. 205-218, 2008.

[9] 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, 1992.

[10] G. D. Konstantakopoulos, S. P. Gayialis, and E. P. Kechagias, "Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification," *Operational research,* vol. 22, no. 3, pp. 2033-2062, 2022.

[11] E. Arifta and F. Rakhmawati, "Analysis of Book Distribution Routes Using the Capacity Vehicle Routing Problem (CVRP) Method Using the Sweep Algorithm," *Sinkron: jurnal dan penelitian teknik informatika,* vol. 8, no. 1, pp. 360-367, 2023.

[12] F. Morsidi and I. Y. Panessai, "Overview of the Integral Impact of MDVRP Routing Variables on Routing Heuristics," *Applied Information Technology And Computer Science,* vol. 4, no. 1, pp. 1723-1738, 2023.

[13] V. R. Máximo, J.-F. Cordeau, and M. C. Nascimento, "An adaptive iterated local search heuristic for the Heterogeneous Fleet Vehicle Routing Problem," *Computers & Operations Research,* vol. 148, p. 105954, 2022.

[14] L. Accorsi and D. Vigo, "A hybrid metaheuristic for single truck and trailer routing problems," *Transportation Science,* vol. 54, no. 5, pp. 1351-1371, 2020.

[15] Y. Conde, "Biblioteca de heurísticas de construcción para el problema de ruteo de camiones y remolques," Universidad Tecnológica de La Habana José Antonio Echeverría CUJAE., La Habana, 2014.

[16] Rita M. Newton and W. Thomas, "Design of School Bus Routes by Computer," *Socio-Economic Planning Sciences,* vol. 3, pp. 75-85, 1969.

[17] V. Y. Piqueras, "Optimización heurística económica aplicada a las redes de transporte del tipo VRPTW," Phd, Departamento de Ingeniería de la Construcción y Proyectos de Ingeniería Civil, Universidad Politécnica de Valencia, Valencia, España, 2002.

[18] M. A.-e. Asghari and S. M. J. Mirzapour, "Green vehicle routing problem: A state-of-the-art review," *International Journal of Production Economics,* vol. 231, p. 107899, 2021.

[19] B. H. O. Rios, "Heuristics for vehicle routing problems with uncertainty= Heurísticas para problemas de roteamento de veículos com incertezas," [sn], 2023.

[20] E. Ruiz y Ruiz, I. García-Calvillo, and S. Nucamendi-Guillén, "Open vehicle routing problem with split deliveries: mathematical formulations and a cutting-plane method," *Operational Research,* vol. 22, no. 2, pp. 1017-1037, 2022.

[21] H. Qin, X. Su, T. Ren, and Z. Luo, "A review on the electric vehicle routing problems: Variants and algorithms," *Frontiers of Engineering Management,* vol. 8, pp. 370-389, 2021.

[22] D. S. Hochba, "Approximation algorithms for NP-hard problems," *ACM Sigact News,* vol. 28, no. 2, pp. 40-52, 1997.

[23] E.-G. Talbi, *Metaheuristics: from design to implementation*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2009.

[24] M. L. Fisher and R. Jaikumar, "A generalized assignment heuristic for vehicle routing," *Networks,* vol. 11, no. 2, pp. 109-124, 1981.

[25] M. H. Romanycia and F. J. Pelletier, "What is a heuristic?," *Computational intelligence,* vol. 1, no. 1, pp. 47-58, 1985.

[26] R. Martı, "Procedimientos metaheurısticos en optimización combinatoria," *Matemátiques, Universidad de Valencia,* vol. 1, no. 1, pp. 3-62, 2003.

[27] W. J. Clancey, "Heuristic classification," *Artificial intelligence,* vol. 27, no. 3, pp. 289-350, 1985.

[28] G. W. Nurcahyo, R. A. Alias, S. M. Shamsuddin, and M. N. M. Sap, "Sweep algorithm in vehicle routing problem for public transport," *Jurnal Antarabangsa Teknologi Maklumat,* vol. 2, pp. 51-64, 2002.

[29] L. B. R. Medina, E. C. G. La Rota, and J. A. O. Castro, "Una revisión al estado del arte del problema de ruteo de vehículos: Evolución histórica y métodos de solución," *Ingeniería,* vol. 16, no. 2, pp. 35-55, 2011.

[30] R. Mole and S. Jameson, "A sequential route-building algorithm employing a generalised savings criterion," *Journal of the Operational Research Society,* vol. 27, no. 2, pp. 503-511, 1976.

[31] A. Olivera, "Heurísticas para problemas de ruteo de vehículos," *Reportes Técnicos 04-08,* 2004.

[32] D. A. González Restrepo and D. Y. Gómez Veloza, "Solución al problema de ruteo de vehículos con entregas y recogidas aplicando el algoritmo de pétalos y la heurística del vecino más cercano," 2019.

[33] A. C. P. Pérez, E. S. Ansola, and A. Rosete, "A metaheuristic solution for the school bus routing problem with homogeneous fleet and bus stop selection," *Ingeniería,* vol. 26, no. 2, pp. 233-253, 2021.

[34] A. L. Infante Abreu, R. Díaz Hernández, M. André Ampuero, A. Rosete Suárez, J. Fajardo Calderín, and K. Escalera Fariñas, "Solución al problema de conformación de equipos de proyectos de software utilizando la biblioteca de clases BICIAM," *Revista Cubana de Ciencias Informáticas,* vol. 9, pp. 126-140, 2015.

[35] M. Maischberger, "COIN-OR METSlib a Metaheuristics Framework in Modern C++," *Firenze University. Recuperado de* [*https://projects*](https://projects)*. coin-or. org/metslib,* 2011.

[36] E. Alba *et al.*, "MALLBA: A library of skeletons for combinatorial optimisation," in *European Conference on Parallel Processing*, 2002, pp. 927-932: Springer.

[37] C. Groër, B. Golden, and E. Wasil, "A library of local search heuristics for the vehicle routing problem," *Mathematical Programming Computation,* vol. 2, pp. 79-101, 2010.

[38] K. Florios and G. Mavrotas, "Generation of the exact pareto set in multi-objective traveling salesman and set covering problems," *Applied Mathematics and Computation,* vol. 237, pp. 1-19, 2014.

[39] L. Díaz, "Nueva versión de la Biblioteca de Heurísticas de Construcción para Problemas de Planificación de Rutas de Vehículos," Tesis de Diploma, Instituto Superior Politécnico José Antonio Echeverría, 2016.

[40] I. Torres, "Componente de software: BHCVRP," *Transferencia Tecnológica CITI,* 2016.

[41] F. Didier, L. Perron, S. Mohajeri, S. A. Gay, T. Cuvelier, and V. Furnon, "OR-Tools’ vehicle routing solver: a generic constraint-programming solver with heuristic search for routing problems," 2023.

[42] E. Lima, "CommuteVRP: otimização de serviços privados de transporte contínuo," in *Anais do XIII Simpósio Brasileiro de Sistemas de Informação*, 2017, pp. 547-554: SBC.

[43] A. Mahmoud, T. Chouaki, S. Hörl, and J. Puchinger, "Extending JSprit to solve electric vehicle routing problems with recharging," *Procedia Computer Science,* vol. 201, pp. 289-295, March 22-25 2022.

[44] M. Karkula, J. Duda, and I. Skalna, "Comparison of capabilities of recent open-source tools for solving Capacitated Vehicle Routing Problems with Time Windows," *Carpathian Logistics Congress,* Accessed on: December 2-4, 2019

[45] M. PAREJO *et al.*, "Desarrollo de un algoritmo para seleccionar la ubicación de puntos de abastecimiento minimizando los tiempos y costos de transporte," *Revista Espacios,* vol. 41, no. 17, pp. 1-15, 2020.

[46] O. E. Calderón Calderón, "Diseño e implementación de modelo matemático para la optimización del recorrido de camiones recolectores de residuos sólidos en el municipio de León," 2020.

[47] P. Offermann, S. Blom, M. Schönherr, and U. Bub, "Artifact types in information systems design science–a literature review," in *Global Perspectives on Design Science Research: 5th International Conference, DESRIST 2010, St. Gallen, Switzerland, June 4-5, 2010. Proceedings. 5*, 2010, pp. 77-92: Springer.

[48] F. S. Hillier, "Investigación de operaciones," 2015.

[49] J. Puchinger and G. R. Raidl, "Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification," in *International work-conference on the interplay between natural and artificial computation*, 2005, pp. 41-53: Springer.

[50] M. Cuadrado and V. Griffin, "Modelos Matemáticos para la Optimización de la Distribución de Vehículos Nuevos en Venezuela. Caso: Clover International CA," *Ingeniería Industrial. Actualidad y Nuevas Tendencias,* vol. 1, no. 1, pp. 53-65, 2009.

[51] H. S. Salazar, "Optimizacion Multiobjetivo Aplicado a un Problema de Ruta Corta Estocastico," 2004.

[52] R. Matai, S. P. Singh, and M. L. Mittal, "Traveling salesman problem: an overview of applications, formulations, and solution approaches," *Traveling salesman problem, theory and applications,* vol. 1, no. 1, pp. 1-25, 2010.

[53] K. L. Hoffman, M. Padberg, and G. Rinaldi, "Traveling salesman problem," *Encyclopedia of operations research and management science,* vol. 1, pp. 1573-1578, 2013.

[54] M. R. Garey and D. S. Johnson, "A Guide to the Theory of NP-Completeness," *Computers and intractability,* pp. 37-79, 1990.

[55] S.-W. Lin, F. Y. Vincent, and S.-Y. Chou, "Solving the truck and trailer routing problem based on a simulated annealing heuristic," *Computers & Operations Research,* vol. 36, no. 5, pp. 1683-1692, 2009.

[56] B. Golden, A. Assad, L. Levy, and F. Gheysens, "The fleet size and mix vehicle routing problem," *Computers & Operations Research,* vol. 11, no. 1, pp. 49-66, 1984.

[57] S. Baptista, R. C. Oliveira, and E. Zúquete, "A period vehicle routing case study," *European Journal of Operational Research,* vol. 139, no. 2, pp. 220-229, 2002.

[58] J.-F. Cordeau and G. d. é. e. d. r. e. a. d. décisions, *The VRP with time windows*. Citeseer, 2000.

[59] R. Baldacci, P. Toth, and D. Vigo, "Exact algorithms for routing problems under vehicle capacity constraints," *Annals of Operations Research,* vol. 175, pp. 213-245, 2010.

[60] J. L. Franco and S. N. Isaza, "Heurística para la Generación de un Conjunto de Referencia de Soluciones que Resuelvan el Problema de Ruteo de Vehículos con Múltiples Depósitos MDVRP," 2012.

[61] D. I. G. MATEOS, A. G. Gómez, D. D. A. MARTINO, J. P. GARCÍA, and N. G. Fernández, "Desarrollo de un método híbrido para la resolución del MDVRP," *Revista de la Escuela Jacobea de Posgrado* [*http://revista*](http://revista)*. jacobea. edu. mx,* no. 5, pp. 45-64, 2013.

[62] L. Algarra and D. A. Pelta, "DVRP-OCR: Un Método Cooperativo Basado en Heurısticas Simples para el DVRP."

[63] A. F. Tangarife Álvarez, "Revisión del estado del arte del problema de ruteo abierto (OVRP)," 2017.

[64] K. Braekers, K. Ramaekers, and I. v. Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review," *Comput. Ind. Eng.,* vol. 99, pp. 300-313, 2016.

[65] M. Dror, G. Laporte, and P. Trudeau, "Vehicle routing with split deliveries," *Discrete Applied Mathematics,* vol. 50, no. 3, pp. 239-254, 1994.

[66] G. Laporte and F. V. Louveaux, *Solving stochastic routing problems with the integer L-shaped method*. Springer, 1998.

[67] J. F. Bard, L. Huang, M. Dror, and P. Jaillet, "A branch and cut algorithm for the VRP with satellite facilities," *IIE transactions,* vol. 30, no. 9, pp. 821-834, 1998.

[68] F. Narducci, "Programación de talleres intermitentes flexibles, por medio de la heurística del margen de tolerancia. 117 p," Tesis de maestría (Ingeniería Industrial). Universidad del Norte. Barranquilla, 2005.

[69] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, *Modern heuristic search methods*. Wiley, 1996.

[70] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & operations research,* vol. 13, no. 5, pp. 533-549, 1986.

[71] I. M. Chao, "A tabu search method for the truck and trailer routing problem," *Computers & Operations Research,* vol. 29, pp. 33-51, 2002.

[72] J. M. Daza, J. R. Montoya, and F. Narducci, "Resolución del problema de enrutamiento de vehículos con limitaciones de capacidad utilizando un procedimiento metaheurístico de dos fases," *Revista EIA,* no. 12, pp. 23-38, 2009.

[73] B. E. Gillett and L. R. Miller, "A heuristic algorithm for the vehicle-dispatch problem," *Operations research,* vol. 22, no. 2, pp. 340-349, 1974.

[74] H. Gabow, "An Efficient Implement at ion of Edmonds’ Maximum-Matching Algorithm," Stanford University, 1972 (Technical Report No. 31). <ftp://reports>. stanford …1972.

[75] N. Christofides, "The vehicle routing problem," *Combinatorial optimization,* 1979.

[76] J. Bramel and D. Simchi-Levi, "A location based heuristic for general routing problems," *Operations research,* vol. 43, no. 4, pp. 649-660, 1995.

[77] V. R. d. Souza, "Uma análise do framework OptaPlanner aplicado ao problema de empacotamento unidimensional," *Research Gate*, 2 de febrero de 2020.

[78] I. Sommerville and A. Wesley, "Introducción a la Ingeniería de Software," *I. Sommerville, Ingeniería de Software. Pearson Educación,* 2011.

[79] K. Arnold, J. Gosling, and D. Holmes, *The Java programming language*. Addison Wesley Professional, 2005.

[80] N. Pérez, "Incorporación de algoritmos de agrupamiento en la Biblioteca de Heurísticas de Asignación para Problemas de Planificación de Rutas de Vehículos con Múltiples Depósitos.," Trabajo de diploma para optar por el título de Ingeniería en Informática, Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), La Habana, 2022.

[81] R. Nerey, "Herramienta para la generación de rutas en el problema del transporte obrero: Trans-o," Tesis de Diploma, Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), La Habana, 2017.

[82] M. Machado, "Biblioteca de Heurísticas de Asignación para el Problema de Planificación de Rutas de Vehículos con Múltiples Depósitos.," Trabajo de diploma para optar por el título de Ingeniería en Informática, Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), La Habana, 2019.

[83] P. Kilby, P. Prosser, and P. Shaw, "Dynamic VRPs: A study of scenarios," *University of Strathclyde Technical Report,* vol. 1, no. 11, 1998.

[84] P. Kilby, P. Prosser, and P. Shaw, "Guided local search for the vehicle routing problem with time windows," *Meta-heuristics: Advances and trends in local search paradigms for optimization,* pp. 473-486, 1999.

[85] M. Desrochers and T. Verhoog, "A new heuristic for the fleet size and mix vehicle routing problem," *Computers & Operations Research,* vol. 18, no. 3, pp. 263-274, 1991.

[86] K. Altinkemer and B. Gavish, "Parallel savings based heuristics for the delivery problem," *Operations research,* vol. 39, no. 3, pp. 456-469, 1991.

[87] P. Wark and J. Holt, "A repeated matching heuristic for the vehicle routeing problem," *Journal of the Operational Research Society,* vol. 45, no. 10, pp. 1156-1167, 1994.

[88] C. B. Reynoso, "Introducción a la Arquitectura de Software," *Universidad de Buenos Aires,* vol. 33, 2004.

[89] V. K. Madasu, T. Venna, T. Eltaeib, M. Moalla, N. Almuslet, and A. Badaoui, "Solid principles in software architecture and introduction to resm concept in oop," *Studies,* vol. 35, no. 38, p. 8, 2015.

[90] P. Lalanda, "Style-specific techniques to design product-line architectures," ed: Thomson-CSF Corporate Research Laboratory, Domaine de Corbeville, Orsay, France, 1999.

[91] R. Capilla Sevilla, "Temas Diseño y Arquitectura del Software," 2022.

[92] G. Torossi, "Diseño de sistemas," *Universidad Tecnológica Nacional Disponible en* [*http://exa*](http://exa)*. unne. edu. ar/informatica/anasistem2/public\_html/apuntes/de1. pdf*.

[93] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[94] B. Eckel, *Thinking in Patterns with Java*. President, MindView, Inc., 2003.

[95] F. Martínez and J. Cueva, "Guía de Construcción de Software en Java con Patrones de Diseño, 2000," *Universidad de Oviedo: Escuela Universitaria de Ingeniería Técnica Informática*.

[96] E. Alba, "Networking and emerging optimization,"Available: <http://neo.lcc.uma.es/vrp/vrp-instances/>

[97] PyCharm. (2024, 25 de agosto). *Install PyCharm | PyCharm Documentation*. Available: <https://www.jetbrains.com/help/pycharm/installation-guide.html>

[98] ApacheNetBeans. (2024, 25 de agosto). *Welcome to Apache NetBeans*. Available: <https://netbeans.apache.org/front/main/index.html>

[99] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical test as methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation,* vol. 1, no. 1, pp. 3-18, 2011.

[100] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association,* vol. 32, no. 200, pp. 674-701, 1937.

[101] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin,* vol. 1, no. 6, pp. 80-83, 1945.