ULPGC Universidad de Las Palmas de Gran Canaria | Escuela de Ingeniería Informática

Trabajo de Fin de Grado

Optimización de la logística de retirada de enseres voluminosos

Un enfoque multicriterio

TITULACIÓN: Grado en Ciencia e Ingeniería de Datos

AUTOR: Krish Sadhwani Mansukhani

TUTORIZADO POR:
José Juan Hernández Cabrera

Junio 2024

Agradecimientos

Antes que nada, a José Juan Hernandez, tutor de este trabajo, por ser siempre un gran pilar en la enseñanza obtenida.

A Jose Belizón, socio-director de Monentia S.L., por el apoyo en la elaboración del proyecto.

A Jose Évora y Octavio Roncal, que han servido como respaldo ante cualquier duda y problemas enfrentados.

Por supuesto también, a los miembros de la comunidad que forman la Escuela de Ingeniería Informática, tanto profesores, como el personal y los alumnos que han instruido y guiado en el camino hasta llegar al punto actual de mi carrera profesional.

Resumen

La gestión eficiente de enseres voluminosos supone un desafío crucial para servicios públicos y autoridades locales. En Europa, se generan anualmente hasta 30 mil toneladas de colchones en desuso [7] y cerca de 55 millones de toneladas de residuos de madera en 2016 [2], aumentando un 110 % en la última década.

La implementación del proyecto ha generado un sistema innovador para la gestión de enseres voluminosos, mejorando la eficiencia en su retirada y optimizando las rutas. Elaborado con éxito en colaboración con la consultoría tecnológica Monentia, para su aplicabilidad práctica. Los resultados sientan las bases para futuras investigaciones y desarrollos, ofreciendo un enfoque sostenible que mejora la calidad de vida de los ciudadanos y proporciona una infraestructura adaptable para los Ayuntamientos.

Abstract

The efficient management of bulky waste poses a crucial challenge for public services and local authorities. In Europe, up to 30 thousand tons discarded mattresses [7] and approximately 55 million tons of wood waste were generated in 2016 [2], marking a 110 % increase over the past decade.

The implementation of the project has generated an innovative system for the management of bulky waste, improving efficiency in their removal and optimizing routes. Developed successfully in collaboration with the technology consultancy Monentia, for its practical applicability. The results lay the foundation for future research and development, offering a sustainable approach that improves the quality of life of citizens and provides an adaptable infrastructure for city councils.

Índice general

Índice de figuras

Capítulo 1

Fundamentos Teóricos

La civilización avanza al ampliar el número de importantes operaciones que podemos realizar sin pensar en ellas.

Alfred North Whitehead

El presente capítulo sumerge en la compleja tarea de mejorar la gestión de la recogida de residuos voluminosos a través de un enfoque sofisticado, empezando por los conceptos teóricos de la optimización logística.

Siguiendo esta premisa, se explorará la relevancia de la optimización en este contexto y realizando una revisión de la literatura para establecer un fundamento sólido. Desde la introducción hasta la aplicación práctica, este capítulo busca trazar un camino hacia una gestión más eficiente y sostenible de los enseres voluminosos, inspirado por la sabiduría que yace en la mejora constante.

1.1. Introducción a la logística de retirada de enseres voluminosos

El objetivo de este proyecto es el desarrollo de software para optimizar la gestión de la recogida de residuos voluminosos, abordando específicamente los desafíos asociados al **Problema del Agente Viajero** (TSP), **Problema de Rutas de Vehículos** (VRP) y el **Problema de la Mochila** (Knapsack Problem).

La complejidad inherente a estos problemas, clasificados como NP-completos, demanda soluciones inteligentes y eficientes. El desarrollo de software eficiente para la gestión de residuos voluminosos debe abordar tanto desafíos matemáticos como algoritmos avanzados. Esto implica proporcionar soluciones aproximadas que se acerquen a la óptima en tiempos computacionales razonables, asegurando una gestión de rutas que no solo sea eficiente operacionalmente, sino también sostenible desde una perspectiva medioambiental.

Retos enfrentados ante este tipo de problemas:

- ✔ **Complejidad exponencial**: El número de rutas posibles crece factorialmente con el número de puntos, lo que convierte la búsqueda de la solución óptima en un problema computacionalmente desafiante.

- ✔ **Consideración de restricciones**: La inclusión de restricciones de capacidad de vehículos y restricciones temporales añade una capa adicional de complejidad al problema.

- ✔ **Asignación de Vehículos**: Determinar el número óptimo de vehículos y asignar eficientemente un conjunto de peticiones a cada uno, considerando restricciones de capacidad y temporales.

- ✔ **Optimización de múltiples rutas**: Coordinar múltiples vehículos para minimizar la distancia total recorrida y otras variables, lo que agrega complejidad adicional.

1.2. Relevancia de la optimización logística en el contexto

La respuesta a la necesidad imperante de optimizar la logística de recogida de enseres voluminosos es crucial, considerando la creciente demanda y complejidades asociadas. Los problemas tipo TSP, VRP y el Problema de la Mochila con su complejidad exponencial y la necesidad de considerar restricciones variadas, plantean desafíos que van más allá de la capacidad de los métodos tradicionales.

Se observa una tendencia de consumo en constante crecimiento, atribuible a factores como la duración reducida de la vida útil de muebles y enseres, en parte influida por fenómenos como el efecto IKEA.

Ante la situación actual enfrentada en Las Palmas, se pone como ejemplo la **Figura 1.1**, que muestra una fotografía capturada de manera espontánea en una ubicación cerca de la universidad ULPGC, donde se observa condiciones normales de basura y enseres abandonados al lado de un contenedor. La imagen refleja la falta de concienciación social en cuanto al medio ambiente y la gestión adecuada de los enseres. Esta fotografía se tomó de forma aleatoria durante un recorrido por la calle, sin prever su contenido, lo que resalta la presencia cotidiana y natural del problema en el entorno urbano.

Así mismo, para la **Figura 1.2**, se presenta una fotografía extraída de un medio digital periodístico, donde se destaca un gran volumen de enseres tirados en un entorno urbano. A diferencia de la primera imagen, esta fotografía se publica intencionalmente en un medio de comunicación como una noticia relevante para la audiencia. La presencia de esta imagen en una publicación oficial subraya la importancia y la gravedad del problema de la gestión de residuos voluminosos, evidenciando su relevancia a nivel social y mediático.

Ilustración 1.1: Fotografía de basura y enseres abandonados

Ilustración 1.2: Fotografía de enseres destacados en una publicación oficial, fuente [5]

En un entorno donde la asignación deficitaria de la demanda supera a la oferta, se hace evidente la urgencia de abordar la eficiencia logística como un factor crucial para el éxito operativo y económico.

El problema no radica únicamente en la falta de tiempo disponible en la oferta, sino **más bien en la asignación deficitaria de la demanda**, donde esta, supera significativamente a la oferta disponible. Por lo que optar por métodos tradicionales no conlleva a tener un sistema óptimo, ni mucho menos, dadas las cantidades crecientes generados de los residuos de estos tipos anualmente y negligencia del entendimiento de las situaciones particulares de los vehículos en operación.

El ayuntamiento de Las Palmas como ejemplo, actualmente trabaja de una forma donde las peticiones son asignadas ante un acuerdo con el ciudadano mediante llamada telefónica. Donde estas, se registran dependiendo de los huecos que tuvieran libres, no entendiendo la demanda a la cuál pudieran enfrentarse los vehículos y las rutas a las cuales estarían sometidas aunque fueran ineficientes por el consumo de energía.

Dicho esto, es necesario esteblecer un sistema de optimización que pudiera proveer datos actualizados al personal del servicio ante las particularidades de cada vehículo, y que estas a posteriori pudieran luego acordar las fechas y horas con los ciudadanos teniendo la **garantía** que todas las peticiones serán resueltas, además de estar trabajando de una forma eficiente.

Por ello, el proyecto se concentra en resolver problemas concretos en Gran Canaria, para en particular, el ayuntamiento de Las Palmas, estableciéndose como precedente, para otros organismos en la región y el país a su vez. Proponiendo una metodología robusta y un software avanzado que pudiera adaptarse a diversas ubicaciones y contextos logísticos.

1.3. Revisión de la literatura

Desde los primeros intentos por abordar problemas logísticos hasta la era actual de avanzadas tecnologías, la optimización logística ha emergido como un campo dinámico y esencial.

En sus primeras incursiones, la literatura abordó problemas NP-completos, como el Problema del Agente Viajero (TSP), el Problema de Rutas de Vehículos (VRP) y el Problema de la Mochila (Knapsack Problem), reconociendo la complejidad exponencial de estos desafíos. Sin embargo, la evolución de los enfoques ha sido constante, marcada por la necesidad de adaptarse a las cambiantes demandas de la sociedad moderna.

La literatura más reciente refleja una transición hacia la aplicación práctica de algoritmos avanzados y estrategias heurísticas para la gestión de residuos voluminosos. La introducción de tecnologías emergentes, como la inteligencia artificial y el aprendizaje automático, ha revolucionado la forma en que se aborda la optimización logística. La capacidad de procesar grandes volumenes de datos y la mejora continua de algoritmos han allanado el camino hacia soluciones más eficientes y adaptativas.

En la etapa actual, se encuentra estando en un punto crítico donde la literatura converge con la práctica, aprovechando no solo la sabiduría acumulada sino también las herramientas tecnológicas disponibles. La atención se centra en implementaciones específicas, como el caso de Gran Canaria, y se vislumbra un futuro donde la integración de tecnologías avanzadas en la gestión de residuos voluminosos no solo sea necesaria sino imperativa para alcanzar niveles óptimos de eficiencia y sostenibilidad en el campo.

1.4. Estado actual

En respuesta a los desafíos asociados con problemas de enrutamiento de vehículo (VRP), del agente viajero (TSP) y de la mochila (Knapsack), se están desarrollando y aplicando diversos avances tecnológicos.

El uso de algoritmos de optimización, sistemas de información geográfica (SIG) y sensores IoT (Internet de las cosas) está permitiendo una gestión más eficiente de las rutas de recogida, optimizando la asignación de recursos y reduciendo los costos operativos para resolver las tareas genéricas pertenecientes al problema del enrutamiento de vehículo (VRP).

Uno de los casos, es el software de optimización de rutas desarollado por UPS, ORION (On-Road Integrated Optimization and Navigation). Este software, juega un papel fundamental en la gestión eficiente de las operaciones de entrega del gigante del

transporte marítimo. Con aproximadamente 5,5 mil millones de paquetes entregados anualmente utilizando una flota de 125,000 vehículos, UPS confía en ORION para optimizar rutas con múltiples conductores y múltiples paradas [8].

ORION aprovecha algoritmos y tecnologías sofisticados, como cargas de trabajo de cálculo paralelo, algoritmos geoespaciales metaheurísticos y algoritmos de programación y rutas de vehículos. A la vez, aborda el problema de las rutas de vehículos capacitados con limitaciones, garantizando que las rutas se optimicen y se cumplan con diversas limitaciones operativas.

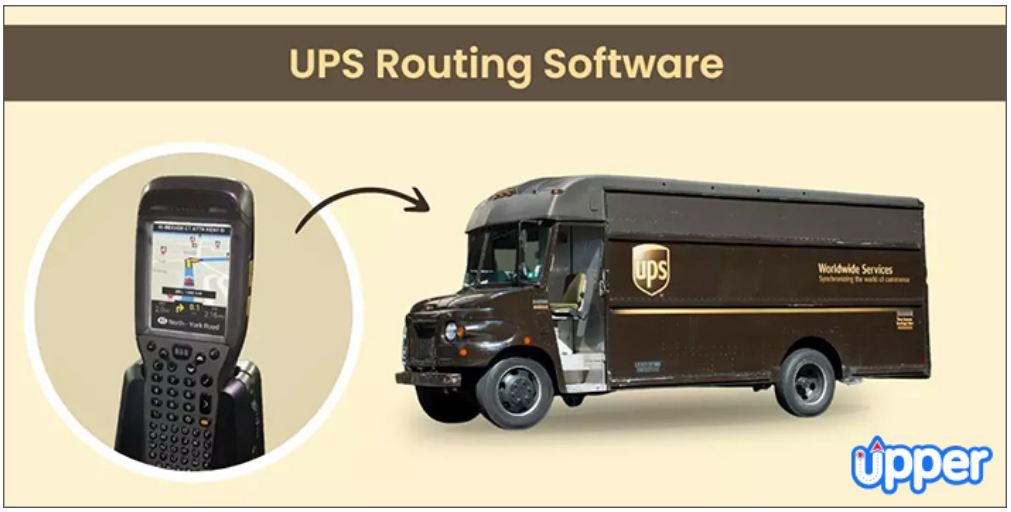

Ilustración 1.3: Integración de ORION en el sistema de UPS, fuente [6]

Una de las características clave de ORION es su capacidad para ajustar rutas dinámicamente en tiempo real, respondiendo a interrupciones o solicitudes específicas de los clientes. Esta flexibilidad mejora la capacidad de UPS para realizar entregas a tiempo y adaptarse a las condiciones cambiantes de manera eficiente.

Si bien ORION ha sido fundamental para mejorar las operaciones de entrega de UPS, se centra principalmente en las entregas de paquetes tradicionales en lugar de los desafíos específicos que plantea la eliminación de artículos voluminosos. A diferencia del DIAD (Dispositivo de adquisición de información de entrega) de UPS y el envío mediante teléfonos inteligentes, que están orientados a las entregas tradicionales, el proyecto para optimizar la logística de retirada de artículos voluminosos **requiere un enfoque único** para abordar las complejidades del problema del viajante (TSP), el problema de las rutas de vehículos (VRP) y el problema de la mochila combinados.

A pesar de su eficacia para optimizar las rutas de entrega, las capacidades de ORION no son directamente aplicables a los desafíos que plantea el proyecto actual. Sin embargo, la implementación exitosa de ORION subraya el potencial de las técnicas de optimización avanzadas para abordar problemas logísticos complejos. Aprovechando los conocimientos adquiridos durante el desarrollo y la implementación de ORION, el proyecto tiene como objetivo desarrollar soluciones innovadoras adaptadas a los **requisitos específicos de la eliminación de artículos voluminosos**, allanando el camino para prácticas de gestión de residuos más eficientes y sostenibles.

Además de la implementación de algún software, cabe destacar las iniciativas para concienciar a la sociedad sobre problemas generados por los enseres voluminosos, financiado por la Unión Europea, URBANREC. Esta iniciativa se trata de proveer una herramienta electrónica (e-tool) y planes de actuación para la valorización de este tipo de residuos, publicando información relevante acerca de los problemas abreviados y cuales son las acciones que se deberían de acometer ante las situaciones. [1]

Desde programas gubernamentales de concienciación hasta proyectos piloto de implementación de nuevas tecnologías, se están explorando diversas estrategias para mejorar la gestión de los residuos voluminosos.

En cuanto a las perspectivas futuras, se espera que el campo de la logística de la retirada de enseres voluminosos pudiera tener una mejora aprovechando los recursos que se disponen. Considerando la falta de atención en el asunto, el proyecto actual sirve incluso para, **captar la atención de servicios gubernamentales y empresariales ante sus métodos tradicionales en el tema**. Se vislumbran oportunidades prometedoras en áreas como la inteligencia artificial, el aprendizaje automático y la analítica de datos para desarrollar soluciones aún más eficientes y sostenibles.

1.5. Competencias específicas

- **ET1** *Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la información y las comunicaciones*

  Dado que el proyecto fue creado en colaboración con la empresa de consultoría tecnológica Monentia, se entendieron sus necesidades y el caso que querían resol-

ver, ante las perspectivas de la clientela y otros aspectos empresariales, de una forma donde se pudiera crear un sistema tecnológico a ser integrado.

- **ED1** *Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.*

  Se han usado técnicas de optimización relativas a la utilización de algoritmos y problemas de minimización/maximización, con restricciones, teniendo en cuenta la dinámica de estas herramientas de una forma práctica.

- **ED2** *Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos de computación, percepción y actuación en ambientes o entornos inteligentes.*

  Ante la elaboración del proyecto y la construcción del software, surgieron problemas y casos que tuvieron que ser resueltas para tener en cuenta aspectos prácticos y de poder ser utilizados en el ámbito empresarial, entendiendo las situaciones particulares de la región y los desafíos asociados ante ello.

- **ED3** *Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volumenes de datos.*

  La aplicación utiliza ciertos datos que se adquieren de forma automatizada haciendo solicitudes a una API que gestiona y procesa la información para luego ser devuelta en una gran cantidad de parámetros, y para la cuál, se extraen y se ajustan para consecuentemente, realizar las operaciones requeridas en el software.

- **ED5** *Capacidad para seleccionar, diseñar, desplegar, integrar, evaluar, construir, gestionar, explotar y mantener las tecnologías de hardware, software y redes para dar soporte a aplicaciones en Ciencia e Ingeniería de Datos.*

El desarollo ha dado lugar a la creación de un servicio web, para la cuál se ha tenido que configurar parámetros asociados a poder lanzar un servicio de este tipo y verificar que los usuarios pudiesen arrancar la aplicación en sus equipos de forma independiente mediante una dirección asociada al *localhost* y un puerto específico definido por el usuario.

- **ED7** *Capacidad para participar activamente en la especificación, diseño, implementación y mantenimiento de los sistemas de información y de las comunicaciones de uso común en Ciencia e Ingeniería de Datos y de diseñar e implementar software de aplicaciones para estos sistemas.*

  La contrucción inicial del software dio lugar a tener que hacer una revisión de las aplicaciones disponibles actualmente en la rama y de participar en asuntos relevantes al tópico enfrentado. Para la cuál, se diseñó la arquitectura del servicio web teniendo estos aspectos en cuenta.

- **ED8** *Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de la información y las comunicaciones para Ciencia e Ingeniería de Datos, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil.*

  El despliegue de la aplicación es realizado a través de una contenerización de ello y para la cuál, la aplicación se dispone a ser ejecutable en cualquier sistema localmente mediante acceso web, realizando solicitudes necesarias para iniciar la computación del programa y de proveer los resultados para poder ser accedidos mediante un enlace de web local.

Capítulo 2

Problemas Destacados

2.1. Problemas NP-Completo

Los problemas *NP-Completo* desempeñan un papel fundamental en la teoría de la complejidad computacional, siendo esenciales para comprender la dificultad inherente de ciertos problemas algorítmicos. Su relevancia en este proyecto sobre la gestión eficiente de la recogida de residuos voluminosos radica en la capacidad de modelar de manera precisa y realista situaciones logísticas complejas.

Estos forman parte de la clase NP (tiempo polinómico no determinista), lo que significa que, si se proporciona una solución candidata, es verificable en tiempo polinómico. Sin embargo, encontrar esa solución candidata en sí misma no se ha demostrado ser computacionalmente eficiente. La notación "NP" proviene de "No Polinómico", indicando que la complejidad de resolver estos problemas no se encuentra dentro de los límites de la eficiencia polinómica.

Presenta una característica fundamental a solucionar, a parte de la complejidad exponencial previamente mencionada:

- **Reducción Polinómica**: Un problema es NP-Completo si es NP y cualquier problema en NP puede reducirse polinómicamente a él. Esta propiedad permite demostrar la NP-Completez de un problema al mostrar que otro problema NP-Completo puede reducirse a él en tiempo polinómico.

El Problema del Agente Viajero (TSP) y Problema de Rutas de Vehículos (VRP), así como el Problema de la Mochila (Knapsack Problem) se presentan como problemas NP-Completos en este asunto.

1. **Problema del Agente Viajero (TSP)**: Consiste en encontrar la ruta más corta que visite todos los nodos de un grafo exactamente una vez y regrese al nodo inicial. Su NP-Completez se deriva de la necesidad de explorar todas las permutaciones posibles para encontrar la solución óptima.

2. **Problema de Rutas de Vehículos (VRP)**: Implica determinar la mejor manera de asignar vehículos para atender un conjunto de ubicaciones, minimizando la distancia total recorrida. Al compartir propiedades con el TSP, el VRP también se clasifica como NP-Completo.

3. **Problema de la Mochila (Knapsack Problem)**: Consiste en seleccionar un conjunto de elementos con valores y pesos dados, de modo que se maximice el valor total sin exceder un peso máximo permitido. Este problema es relevante en la optimización de la carga de los vehículos, asegurando que se utilicen de manera eficiente. Al igual que el TSP y el VRP, el problema de la mochila es NP-Completo, lo que añade complejidad al proyecto.

Relevancia específica ante lo mencionado:

Modelado de Desafíos Logísticos: La elección de abordar el TSP y VRP en el proyecto refleja la realidad de los desafíos logísticos en la gestión de residuos voluminosos. La complejidad de estos problemas captura fielmente la naturaleza intricada de la planificación de rutas y asignación de vehículos en este contexto.

Necesidad de Soluciones Eficientes: La NP-Completez de estos problemas subraya la necesidad de soluciones inteligentes y eficientes. La implementación de estrategias de optimización y heurísticas se convierte en esencial para alcanzar una gestión óptima de rutas en tiempos computacionales razonables.

Aplicación Práctica: La elección de abordar Problemas NP-Completos en el proyecto no solo proporciona un marco teórico sólido, sino que también establece la base para la aplicación práctica de algoritmos avanzados y técnicas de optimización en la gestión concreta de residuos voluminosos.

2.2. Análisis del problema

Ante la visión inicial del proyecto, surgen cuestiones y problemas que se pudieran plantear. Este tipo de asuntos son especialmente importante considerar para la aplicación práctica del proyecto y el enfrentamiento a los retos asociados ante las variantes que pudieran surgir. Abreviando estas problemáticas generaría a plantear una base más sólida de la implementación del modelo a construir.

Se mencionan los siguientes tópicos con sus argumentos y ideas que se manifiestan ante lo previamente expuesto.

2.2.1. Asunto "la vuelta a base"

Ocurre una cuestión importante a destacar, sobre el enrutamiento de los vehículos.

Los vehículos destinados a retirar los enseres que dejan los usuarios a partir de las peticiones, tienen que atender a ellas de forma que la distancia es un factor crucial a considerar y que formaría parte de un coste ante la optimización.

Al estar atendiendo a las peticiones hechas por los usuarios para la retirada de los enseres, los costes cálculados no tomarían en cuenta el coste de vuelta a base si lo que se consideran son los caminos a las peticiones de forma unidireccional.

El aspecto de considerar las rutas de los camiones en el punto de partida (que generalmente es desde un vertedero/punto limpio, base como se refirió) y que luego de completar las solicitudes, se debe regresar a la base como requisito necesario para la disposición de los objetos, considerando también, que los vehículos deberían de partir de un determinado sitio y volver a ello para el adecuado funcionamiento de la organización interna conducidas por la entidad encargada de ofrecer este tipo de recursos en marcha.

No tomar este concepto en cuenta pudiera ocasionar que un vehículo, al haber atendido a unas cuantas peticiones, se sitúe en una ubicación muy alejada (alto coste) del punto limpio, y por necesidad, ha de volver a ella. Esto genera altos consumos de energía y en los peores casos, que el vehículo no pudiera retirar los enseres recogidos y fuera a quedarse inmovilizado en medio de la trayectoria.

2.2.2. ¿Que ocurre si sólo quedase un vehículo disponible para atender a las peticiones?

Se parte del argumento, suponiendo que existen varios vehículos iniciales para la retirada de los enseres y que estos, trabajan cooperativamente. Algunos podrían eliminarse ante la búsqueda de la siguiente petición a atender, simplemente por la razón de que el espacio de ocupación de ellos está al límite por la que no es posible que pudiesen acumular más objetos con la finalidad de ser retiradas. Aunque podrían existir otras razones por la que los vehículos dejasen de atender a las peticiones, en caso de no cumplir las restricciones establecidas.

Ante esto, podría darse la circunstancia de que, al estar los vehículos volviendo a sus bases, se encuentre disponible solo un vehículo para atender a las peticiones restantes.

En este caso, si el sistema trabaja con un algoritmo de asignación, es decir con una técnica de optimización combinatoria. Podrían surgir problemas en la asignaciones a hacer debido a que solo se encuentra un "worker" (trabajador).

Esto da lugar a un concepto, el hecho de que el Vehículo$_i$ tuviera que atender a Petición$_x$ y Petición$_y$ en esa orden o viceversa, no complace a generar alguna mejora en eficiencia si se hace ante un determinado orden. Al atender a las dos peticiones de cualquier forma, el tiempo y energía empleado por el vehículo se vería afectado de la misma manera.

Sería de necesidad crucial establecer algún protocolo para esta teoría, como se comportaría el único vehículo disponible cuando aún existen peticiones pendientes de ser atendidas, y cuáles serían las restricciones a considerar para que dicho recurso no fuera estar atendiendo más de lo que debería.

2.2.3. Consideraciones ante peticiones no resueltas

Asumiendo que el algoritmo garantizaría que todas las peticiones reciban una asignación de los vehículos en servicio; sin embargo, cuando no se cumple alguna de las restricciones especificadas, podrían haber situaciones en las que la aplicación se detenga. Como ejemplo, si el tiempo trabajado para todos los camiones por detección

automatizada, ha superado el límite definido (se podría imaginar el caso del conductor trabajando un máximo de horas), al no haber camiones en servicio, la aplicación tendría que finalizar *(se mencionan camiones concretamente mencionando a los vehículos, se usarán los dos términos para referirse a lo mismo).*

Podría haber casos en los que las peticiones se queden sin resolver y deban transferirse en la siguiente ejecución de la aplicación. Esto lógicamente, produce a una cantidad acumulativa de peticiones, ya que si en la siguiente puesta en marcha de la aplicación, habiendo definido nuevas peticiones a atender y que en su terminación, no se resuelven sus peticiones definidas, esta traslada sus peticiones no resueltas más de la anterior, a la siguiente ejecución.

Sería esencial considerar, adicionalmente, las priorizaciones para determinadas peticiones. Dependiendo de las condiciones de los usuarios y el nivel de urgencia a la que se encuentran para la retirada de sus enseres, podrían encontrarse con mayor o menor nivel de satisfacción. Sobre todo recalcar que las peticiones no atendidas, habiéndose quedado ante una prolongación de los días sin haber sido resuelto producirá a los usuarios que estén insatisfechos.

Así mismo, el hecho de resolver una petición inatendida frente a otra, tendría que venir con algún motivo y causa justificada. La razón por la que uno recibió el tratamiento antes que otro debido a que la aplicación consideró ciertos factores y, no otros estimando el nivel de importancia para los usuarios por sus necesidades. Podría incluso generar un cierto nivel de sesgo si no se tuvieran en cuenta estos aspectos.

2.2.4. Medición del tiempo de trabajo por vehículo

Pudiendo automatizar el proceso de la medición del tiempo para el tiempo que lleva trabajando un vehículo es importante, sobre todo, para registrar las horas empleadas en cargo por los conductores y de tener que quedar dentro de un cierto margen. A parte, estas anotaciones podrían llevar a un posterior análisis llevado a cabo por parte de la entidad encargada de realizar este tipo de optimización, con la finalidad de una continua mejora de sus servicios y calidad ofrecida.

Si el tiempo trabajado por vehículo excede a lo inicialmente configurado en la aplicación, volvería a su base ante la terminación y se quedaría no disponible.

Surge también la propuesta ante el caso de que al tener en dispocición las distancias de las rutas, se podría aplicar unos criterios matemáticos para generar una estimación de tiempo. Pero esta abordación no es adecuada y precisa para su utilización práctica. Dado que las distancias y los tiempos no tienen que estar correlacionados. Un ejemplo; si el lugar de solicitud se encuentra en una carretera con colina (de cuesta arriba), en cuyo caso la distancia registrada puede no ser mucha, pero al ser una pendiente, el vehículo tarda más de lo esperado. Por otro lado, con ese mismo camino en declive, se llega mucho más rápido al destino pero la distancia anotada es igual. Esta problemática en especial se traslada para las regiones montañosas, sobre todo, el caso de estudio del proyecto; Gran Canaria.

Aunque surgen cuestiones técnicas, por un lado, de la posibilidad de calcular tiempos estimados de trabajo para una determinada ruta durante la ejecución del programa. El hecho de tener ese dato es un reto en sí, además de poder incorporarlo para que cada vehículo considere sus propios tiempos, acumulados ante las peticiones atendidas, sin colisionar con los tiempos de otros vehículos es otra dificultad a resolver.

2.2.5. Actualización de ubicaciones de los vehículos

Una abreviación al modelo son los lugares de los vehículos en cada momento. Si la aplicación considera las ubicaciones de forma estática para su realización del cómputo de costes, no considerando las actualizaciones de las coordenadas específicamente de los vehículos. Se produce un mal ajuste.

Cuando un vehículo atiende a una petición, sus coordenadas comparadas con las que tenía inicialmente son actualizadas ante la misma petición que atendío. En las siguientes búsquedas heurísticas, se tendrían que considerar esas nuevas coordenadas como las ubicaciones actuales de los vehículos, usado posteriormente para la puesta en marcha del cálculo a realizar entre las peticiones y el vehículo.

Esta implementación se tiene que aplicar conjuntamente con un tópico mencionado en el siguiente apartado, fruto de esta discusión.

2.2.6. Comprobaciones de asignaciónes únicas por vehículo

El detalle es simple anotar, no pueden haber dos o más vehículos que atiendan a la misma petición, es decir, cada petición debe ser atendida por un único vehículo.

Ante lo referido previamente, las asignaciones únicas son esenciales para que las actualizaciones de las ubicaciones de los vehículos funcione correctamente también. Un ejemplo; si en el caso de dos vehículos (Vehículo$_i$ y Vehículo$_j$) atienden a la Petición$_x$ por no aplicar la teoría de la unicidad, y además se actualizan las coordenadas de los vehículos. Los dos vehículos tendrían las mismas coordenadas (la de Petición$_x$) dando el caso que ante la búsqueda heúristica, se generen los mismos costes para los dos vehículos ya que las trayectorías ante las peticiones son las iguales en los dos por estar localizados en la misma ubicación. Consecuentemente dando lugar a que el algoritmo de asignación no pudiera funcionar bien (aleatoriedad ante la escogida por ejemplo) al no interpretar ninguna diferencia, en cuanto a costes, para los dos vehículos.

Por lo cuál, a cada petición se le debería asignar un solo vehículo único; de lo contrario, se producirían errores de enrutamiento y la optimización fallaría.

2.2.7. Algoritmo de asignación

Dado la complejidad del problema al tratarse como NP-Completo y de tener, ante una primera vista, múltiples criterios y restriciones, prevalece una complicación en cuanto a la utilización de la técnica encargada para la selección del ”worker” frente a otros para resolver una determinada petición.

La técnica o algoritmo, debería de ser capaz de poder solucionar casos ante la duplicidad y errores en la convergencia a la hora de observar los costes obtenidas ante las distintas combinaciones de rutas entre los vehículos y las peticiones.

Un ejemplo; en el supuesto caso donde se encuentran dos mínimos, con el mismo valor, para la misma petición en distintos vehículos. Aquí ocurre la pregunta ante la escogida del vehículo para la atendida a dicha petición, y si se hiciera de una forma aleatoria, se produciría sesgo dando lugar a una optimización ineficiente.

Ocurre también otra cuestión relativa ante el previo apartado sobre las actualizaciones de las coordenadas. El hecho de eliminar a una petición y no considerarla cuando es atendida es importante y lógico. Si el vehículo atiende a la petición, su ubicación debería de actualizarse, pero si aún se considera de alguna forma esa petición, su coste hallado para el vehículo sería nulo ya que se encuentra en el mismo lugar. Podría producirse así mismo, problemas ante la convergencia cuando existen priorizaciones frente a determinadas peticiones siempre por su cercanía y dando el caso, la aplicación no considera asuntos fuera de su ”zona de confort”.

Capítulo 3

Metodología de la Optimización

3.1. Definición de la función objetivo

Para el planteamiento del problema, se abrevia un enfoque multicriterio. Es decir, los costes vendrían abreviados ante una serie de factores, cada uno de ellos independiente del otro, y que sean de alta significación para la optimización dado que en parte, conducen al establecimiento de una determinada ruta.

Cada uno de los factores tendrían un cierto peso, la cuál vendría configurado internamente en la aplicación, se considera de antemano establecer un enfoque equitativo para cada uno.

3.1.1. Costes de parámetros

Ante lo mencionado perviamente, se establecen concretamente tres criterios para calcular posteriormente los costos globales, siendo cada uno de ellos los parámetros definidos:

- **Costos de distancia por carretera entre Vehículo$_i$ y Petición$_x$**

Considerar las distancias para las rutas es imperativo para hallar los costes, un

factor crucial a la hora de establecer una heurística dado que la aplicación se trata principalmente de incorporar las ubicaciones para las peticiones y los vehículos en el mapa, y la distancia es lo primero que uno podría tener en cuenta para nombrar cualquier factor influyente en el modelo.

De igual modo, se menciona especifícamente las distancias por carretera. Debido a que las ubicaciones y la información proporcionada, es reflejado como dato real pudiendo ser representado en un mapa real. Simplemente considerar las distancias por nodos de forma lineal no es representativo de la realidad y carecería de la precisión requerida.

- Costes de ocupación para Vehículo$_i$

Otra variable proviene del peso adquirido por el vehículo ante la inclusión del objeto/s de retirada.

Cada uno de estos conlleva a que pudiera rellenar la capacidad del vehículo de una cierta manera, pudiendo ser reflejado por la medición del enser a retirar (normalmente para considerar todas las dimensiones se especifica la métrica en metros cúbicos).

De esta manera consecuentemente, se podría medir el nivel de urgencia que pudiera tener inheritado la petición dado el volumen del objeto/s a retirar, suponiendo que a mayores dimensiones, más molestias pudiera causarle al usuario intentando retirarla por el servicio.

- Tiempo esperado por el usuario con Petición$_x$

En los anteriores apartados, se había especificado que el tiempo medido no tendría que estar correlacionado con la distancia ante una trayectoria. Simplemente por el hecho de considerar carreteras de cuesta arriba o viceversa, lo cual influyen a la aceleración ganada por puros temas físicos y otros aspectos de la maquinaría del vehículo, que hiciera consumir más o menos energía dada las circustancias a la que se enfrenta.

Por ello, incorporar al factor del tiempo es relevante sobre todo para regiones montañosas como Gran Canaria, con consumos impredecibles ante distintas rutas que pudieran surgir, también aspectos de la compleja y alterada topología de las carreteras en la región.

Conjuntamente los tres componentes abreviados generarían un coste global hallado.

3.1.2. Costes globales

Ante alguna forma, se tiene que establecer un objetivo de minimización/maximización para los factores previamente discutidos y incorporarlos para formar un único coste hallado.

De esta forma y como establecimiento de una teoría que persigue lo siguiente; ***el coste global se obtendrá minimizando las distancias en carretera y el tiempo de espera del usuario, maximizando la ocupación del vehículo.***

Esto significa, que el caso ideal a resolver para el vehículo primero sería un objeto/s de tamaño máximo a retirar, que se ubique a una distancia mínima y utilice el mínimo tiempo esperado. Cubriendo la necesidad de urgencias de los usuarios ante las dimensiones de los enseres a retirar (mayor volumen incrementa la necesidad de retirada), y que estas peticiones localizadas cercas en cuanto a ubicación y en menor tiempo de llegada, serán casos primeros a resolver.

Esta teoría persigue maximizar la satisfacción de los clientes y de resolver peticiones con mayor demanda que resultasen con el consumo mínimo de energía (con menos uso de petróleo por ejemplo y otros recursos del vehículo)

Los pesos se definen para cada parámetro y lógicamente, al incorporar los tres parámetros con distintas escalas en una función, es necesario normalizarlos previamente, por lo que estas listas de arrays, en términos de código, se normalizan en una escala de 0,1.

coste_global= $w_{dist} * X_{dist\_normalizada} - w_{ocup} * X_{ocup\_normalizada} + w_{tiempo} * X_{tiempo\_normalizado}$

El resultado de los costes globales podría ser algo como una matriz con el tamaño de la cantidad de camiones, con tantos valores en cada lista como cantidad de peticiones. Es decir, con un ejemplo de 3 camiones y 10 peticiones, habrán 30 valores para los costes globales.

A partir de estos valores, se utilizará algún algoritmo de asignación que minimice la ruta, finalizando con peticiones seleccionadas por el algoritmo para cada camión y la trayectoría global a seguir.

3.2. Restricciones establecidas

Ante la puesta en marcha de la aplicación, prevalecerían ciertas restricciones que debieran de ser cumplidas durante la ejecución y la búsqueda heurística ante todas las combinaciones posibles de las rutas.

Estas condiciones aseguran que la aplicación funcione como esperado, debido a las integraciones de funcionalidades relativas a las esperadas de casos realísticos.

Se definen las siguientes:

- Número de vehículos en servicio no pueden exceder al límite establecido de vehículos inicialmente.

Se espera que la aplicación anote los vehículos destinados para ser operados pero además y por temas de seguridad (considerado como un cierto respaldo), se recalca que la especificación de un límite en concreto fuera a ser definido para que la aplicación trabaje dentro del margen que se le especifica.

- Cada vehículo tendría un tiempo estimado en trabajo, no debiera de superar el límite de horas de trabajo máximo definido.

El programa se encargará de acumular ciertos tiempos de trabajo calculadas para las rutas dependiendo de las peticiones atendidas y que la funcionalidad de esta se basa en una automatización del proceso del registro de horas.

Normalmente cada conductor tiene que someterse ante un tiempo de trabajo en concreto, y para la cuál no se vendría empeñado a trabajar más de lo que se esperaría. Por ello y para agilizar este proceso, se establece que si algún vehículo fuera a superar el límite de horas establecidas inicialmente, es sometido ante una terminación para la cuál vuelve a su base.

- Para cada petición, solo se encuentra un vehículo único asignado a ella.

En los apartados anteriores se mencionó dicho caso, y como podría causar un impedimento ante la búsqueda debido a la colisión de dos o más vehículos ante una petición.

Aunque este asunto no debería de estar presente en la ejecución de la aplicación, como consideración por si de alguna manera de forma inesperada ocurre dicho problema, se resuelve ante esta misma restricción.

- Existe un tiempo de desatención para cada petición que no fuera a ser atendida ese día (evento), la cuál es dejada para la siguiente. Así, prevalece un máximo de días por la que se deja una petición sin atender y debiera recibir máxima prioridad al llegar cerca del límite del máximo definido.

Esta argumentación trata sobre la satisfacción de los usuarios de la aplicación y la forma a la que se les atiende. Como expectativa, cada cliente esperaría que sus peticiones fueran a ser resueltas en el menor tiempo posible.

Aunque esa posibilidad existe, cuando se encuentran con diversas peticiones en grandes números teniendo en cuenta que los criterios y restricciones debieran de ser cumplidas. Cabe el hecho de que al menos uno quedase sin poder haber sido atendido (lo normal es que la aplicación fuera a resolver todos los casos pero como recalcado, existe de que ocurra el hecho).

Las peticiones no resueltas en el mismo día/evento, son trasladadas al siguiente, por lo que se espera que se resuelva en ese día. Pero, y aunque las probabilidades son mínimas, se podrían quedar sin resolver durante varios días.

Para que exisitera alguna condición que regulase este comportamiento inusual, se destina a establecer un método de priorización para las peticiones que llevan días sin poder haber sido resueltas. El límite es el número de días que pudiera haber una petición sin resolver, si se llega cerca de ello a un cierto margen, se prioriza la petición frente a todas las demás peticiones en el mismo día aunque dichas características de la petición sea muy costosa para llevar a cabo.

De esto se procura, que ninguna petición fuera a llevarse estancado en no ser resuelta.

- Cada véhiculo tiene una capacidad máxima, no debiera de atender a más peticiones si la capacidad llega a un cierto límite, vuelve a su base.

La premisa del concepto es claro, el vehículo solo puede llevar un cierto nivel de carga, que si fuera a ser alcanzado no pudiera a estar disponible para la atención de más peticiones y retorna a su base.

Además de mencionar en concreto que la idea no es especificar lo siguiente; *la capacidad total de los objetos en los vehículos no debería de ser igual o superior a la capacidad maxima de ella*, dado que no es preciso argumentar el caso en este estilo. Es mas bien, que al llegar *cerca* a la capacidad máxima, el vehículo debiera de no encontrarse disponible. Un ejemplo, si la capacidad máxima del vehículo es de diez metros cúbicos, llegar a un cierto límite, como al nueve metros cúbicos, debería de ser suficiente para decir que el vehículo vuelva a su base y no atienda a más peticiones.

Esta analogía permite que los vehículos tengan un cierto margen de espacio, sin tener cantidad excesiva que produzca molestias o que interrumpa su funcionamiento negativamente, para garantizar un adecuado rendimiento en general y retorno seguro a base.

3.3. Herramientas utilizadas

Los componentes principales se configuran y la lógica se implementa en algoritmos hechos mediante el lenguaje Python, destacado por ser una herramienta trabajado sobre todo para aplicaciones de todo tipo, en especial de optimización, y más generales dentro del ámbito de la Inteligencia Artifical y en específico también, el Aprendizaje Automático.

Además el lenguaje facilita el ser usado dados sus entornos de desarollo integrados, destacados como Pycharm, que ofrecen una serie de librerías tratadas en especial para manejar operaciones matemáticas y definiciones matriciales *(NumPy, SciPy, Pandas..etc)*, así como las alteraciones en los índices en distintos elementos para almacenar memoria y colecciones de datos *(listas, arrays NumPy)* que pudieran ser aprovechadas.

Capítulo 4

Adquisición y Procesamiento de Datos

4.1. Planteamiento inicial

Como indicado anteriormente, los datos de entrada para los distintos costes descritos podrían organizarse en dos parámetros para la cuál se prestará atención detallada; la distancia por carreteras y el tiempo esperado dada una cierta ruta. La ocupación del vehículo será incorporado implícitamente durante la ejecución del programa, solicitando ese dato ante los administradores del sistema, ya que en primera resolución, este detalle será incorporado por el propio usuario a la hora de publicar su petición al software.

Dado que la distancia y tiempo tienen que ser recabados de una forma concreta y sobre todo, realísticos. Se aborda el tema en particular.

4.2. Obtención de datos

El enfoque principal es la recopilación de datos que realmente sean representativos ante una determinada ruta entre los vehículos y peticiones.

El asunto no se puede tratar en principio, con librerías gratuitas y de libre uso, por ejemplo; *jsprit*. Debido a que las coordenadas de los vehículos y las peticiones no son exactamente una línea recta distanciada entre sí en cuanto al viaje. Dado que

están mapeadas en ubicaciones reales, se deben considerar las rutas de las carreteras de modo que un vehículo tendría que recorrer esa ruta para llegar a el destino. Por lo que la distancia y tiempo vendría influenciado por ese método.

Existen numerosas API's para poder obtener la información requerida, se enumeran los aspectos a considerar para cada una de ellas y la mejor opción destacada para su selección en el proyecto:

1. **API de Google Maps**

   Fortalezas: Datos completos y ampliamente utilizados, información de tráfico en tiempo real, precisa y confiable.

   Limitaciones: Requiere una clave API y tiene límites de uso. Puede no ser adecuado para uso intensivo o aplicaciones a gran escala.

2. **GraphHopper**

   Fortalezas: Código abierto, personalizable, autohospedado, buen rendimiento.

   Limitaciones: Requiere configuración y mantenimiento, es posible que no tenga datos tan completos como las soluciones comerciales.

3. **OpenRouteService**

   Fortalezas: De código abierto, proporciona una amplia gama de servicios de enrutamiento, incluidas las isócronas.

   Limitaciones: Requiere una clave API, límites de uso, puede haber interrupciones ocasionales en el servicio.

4. **Datos de OpenStreetMap + NetworkX**

   Fortalezas: Código abierto, personalizable, no se requiere clave API, flexibilidad.

   Limitaciones: Requiere preprocesamiento manual de datos, es posible que no tenga

datos en tiempo real y que carezca de algunos atributos de la carretera.

Ante el análisis y la consideración de cada una, se opta por aquello que a la vez de ser preciso, no tuviera límites de uso, con alta disponibilidad y que pudiera ser flexible en cuanto a las configuraciones.

4.3. La herramienta de GraphHopper Routing API

La escogida es realizada ante el servicio de web GraphHopper, especificamente, la API de Routing. Por ser eficiente y confiable en cuanto a los datos que provee, en especial por todas las utilidades y características que forman parte del software.

GraphHopper, es un motor de enrutamiento rápido y eficiente en memoria lanzado bajo la licencia Apache 2.0. Se puede utilizar como una biblioteca Java o un servidor web independiente para calcular la distancia, el tiempo, las instrucciones paso a paso y muchos atributos de la carretera para una ruta entre dos o más puntos.

Más allá de esta ruta ”A to B”, admite ”ajustarse a la carretera”, cálculo isócrono, navegación móvil y más. GraphHopper utiliza datos OpenStreetMap y GTFS de forma predeterminada y también puede importar otras fuentes de datos.

La API de Routing en concreto, calcula la mejor ruta que conecta dos o más puntos, donde el significado de ”mejor” depende del perfil del vehículo y del caso de uso. Además de las coordenadas de la ruta, puede devolver instrucciones paso a paso, elevación, detalles de la ruta y otra información útil sobre la ruta.

Existe un límite de uso para la herramienta que viene condicionado ante las consultas hechas, por sus frecuencias y otros rasgos, aunque dichas limitaciones no impiden su usabilidad en el proyecto y se podría considerar el software como lo suficientemente flexible para las operaciones la cuál realizar.

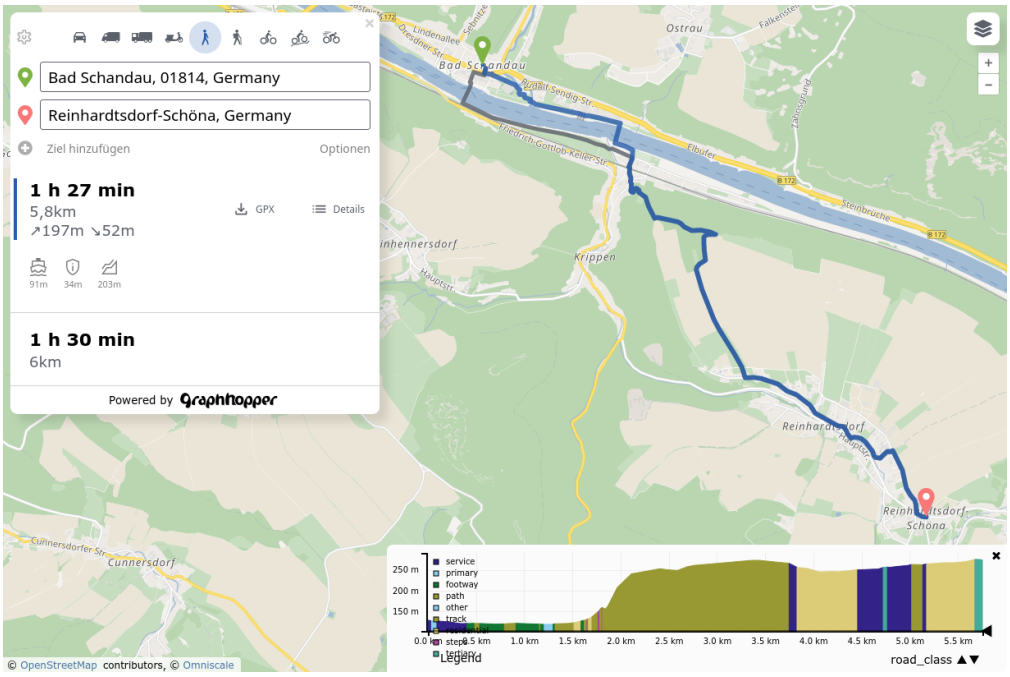

Ilustración 4.1: Ejemplo de consulta en API Explorer, fuente [4]

4.4. Ajustes en el proyecto

Las consultas a la API se realizarán al servicio web que ofrece el GraphHopper, para ello el usuario utilizando el servicio tendrá que hacerlo mediante su clave de API generada dentro de la administración de la cuenta a la hora de inciar sesión en la plataforma.

De esta manera, todas las ”requests” se realizarán dentro del propio programa en ejecución mediante Python, incorporando como dato de entrada, las coordenadas ante el establecimiento de una ruta (ej. las coordenadas del Vehículo$_1$ y Petición$_5$) solicitando la información requerida (distancia y tiempo esperado) ante la respuesta generada a través de un JSON.

Este simple procedimiento se pone en marcha en la construcción del proyecto, y para todas las distintas combinaciones ante las coordenadas de los vehículos y las peticiones, la información obtenida ante el acceso de los parámetros que se quieren obtener del JSON, se guarda luego en contenedores de información *arrays* para su accessibilidad a posteriori, concretando el funcionamiento del cálculo de los costes que se obtendrán.

Capítulo 5

Desarollo del Modelo

5.1. Diseño del software

La arquitectura del proyecto en cuanto a su visualización para la interpretación del ”workflow” o flujo del trabajo la cuál será desempeñada por el software. 5.1

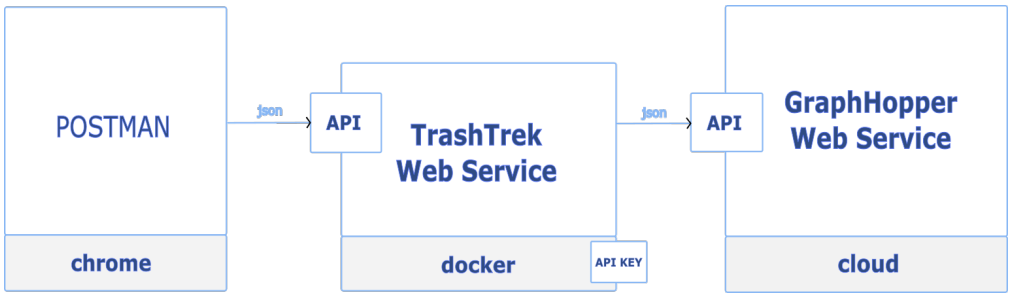

Ilustración 5.1: Arquitectura del proyecto

Los diagramas explican el desarollo de las pruebas a realizar y el uso del servicio. En cuanto al proyecto, vendría figurado como **TrashTrek Web Service**, tratado de poder ser resuelto como un servicio web específicamente y para la cuál, su aplicación será configurado de tál manera.

Las consultas del software se realizarán mediante cualquier herramienta de testeo o pruebas, en este caso, *POSTMAN* como ejemplo y usado en diverso ámbitos, por la

que es de gran popularidad y frecuente uso por los desarolladores debido por parte, a su fácil y gratuito acceso.

Desde POSTMAN, se mandará la información inicial requerida para poner a marcha el software (en formato JSON), a la URL del servicio TrashTrek, la cuál devolverá dentro de un cierto tiempo, la respuesta en el mismo formato, publicando los resultados finales para la interpretación de la heurística encontrada y otros aspectos a discutir posteriormente.

5.2. Organización interna del código

El código elaborado con Python y utilizado en la IDE de Pycharm, es organizado de una forma en concreta para el adecuado adaptación a principios de diseño importantes en cuanto a la creación de un proyecto software.

El esquema proporciona más detalles a percibir.5.2

La estructura se basa en una división en dos funcionalidades distintas a ser procesadas dentro del proyecto TrashTrek. Por una parte, *application* se encargará de la ejecución del programa y las interfaces por las que se definen, así mismo la captación y procesamiento de los datos por el servicio a la hora de tramitar una consulta. La carpeta de *results* almacenará los ficheros de las visualizaciones producidas en HTML, de las cuáles en los resultados finales de las consultas, se podrá acceder mediante la URL definida para el servicio web con su adecuado puerto.

El *model* trata de gestionar los procesos internos y el mecanismo desempeñado por la propia aplicación a la hora de hacer la computación necesaria para el procesamiento.

Dentro de ello, se segmenta por las tareas a realizar, el *display_model* procesará la visualización final ante los datos que se obtienen del resultado, *engine_model* establece distintas funciones, algunas para establecer la gestión de los datos en cuanto a las definiciones de las variables y sus actualizaciones por las iteraciones (*model_data.py*, *model_validator.py* y *model_distance.py*), y otras para hacer el procesamiento requerido específico. Como ejemplo; *el path_calculator* se encarga de cálcular los parámetros de rutas (tiempo y distancia) accediendo a la API de GraphHopper para cada par de

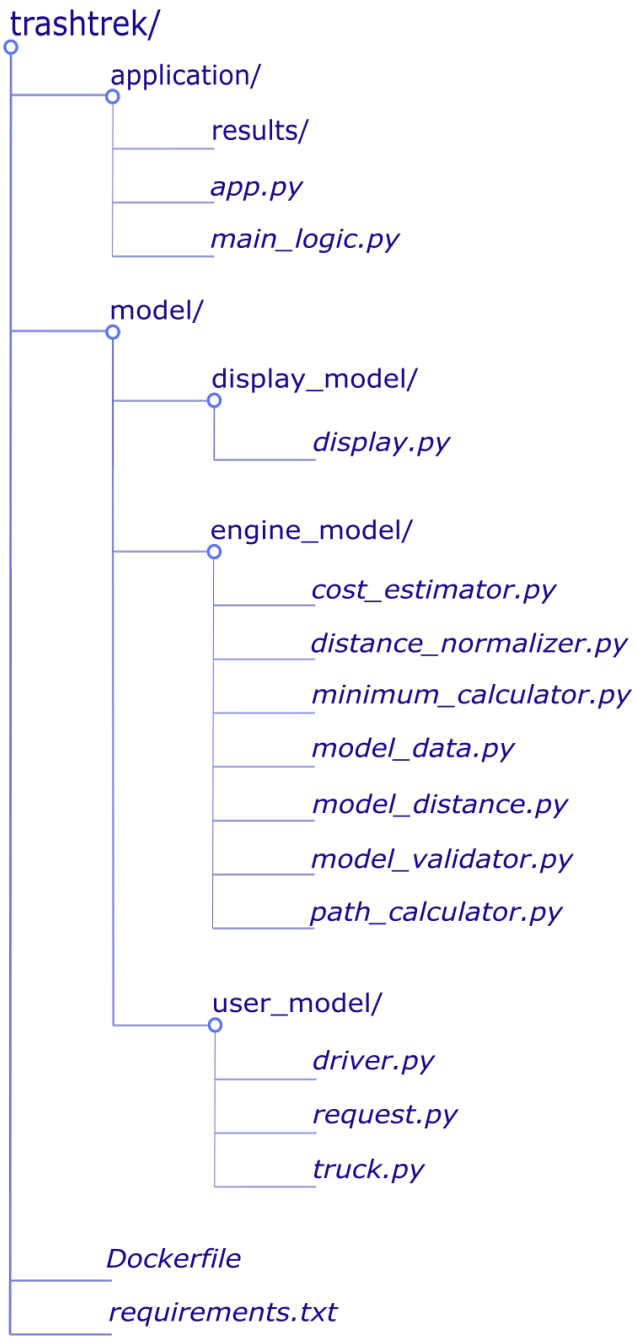

Ilustración 5.2: Jerarquía del proyecto

combinaciones entre las coordenadas proporcionadas de los vehículos y las peticiones.

El *user_model* define los usuarios (dataclass referido en Python), por los objetos de las clases usados en el proyecto. La clase de *driver.py* como ejemplo, influye en cuantos vehículos (camiones) se podrían usar, una especificación menos de las cantidades de vehículos disponibles, hará que la cantidad de ajuste al número de conductores. Estos escenarios podrían servir para casos reales y prácticos, en casos de que, la entidad encargada de gestionar el servicio quisiera proporcionar también la información de los conductores para mejor entendimiento y usabilidad del software, y sobre todo, la organización empresarial.

La lógica del código y utilización de las funciones establecidas en el paquete *model*, serán usadas por el *main_logic.py*, considerado como el programa principal. En ellas, se definen operaciones con un seguimiento ordenado para la cuál utilizará por parte y de manera iterativa, las funcionalidades descritas.

Otros ficheros como el *requirements.txt* y el *Dockerfile* serán de utilización a la hora de construir la imagen en Docker, y desplegar la aplicación con todas sus dependencias (especificadas en el fichero de requisitos).

Las definiciones de todo el código, así como el repositorio Github online del proyecto, vienen especificados en el idioma inglés para su mayor consideración y alcance en el público general, también cumpliendo por parte principios de buen diseño.

5.3. Funcionamiento

5.3.1. Inicialización de los datos

El segmento de la lógica establecida necesita su propio apartado ya que en él, se sitúa el esquema principal de la aplicación en sí.

Se empieza inicializando las funciones y declarándolas en variables apropriados para ello, a partir de los datos proporcionados inicialmente (ej. peticiones, camiones, número máximo de camiones, API de GraphHopper..etc).

```
   model_data = ModelData(requests, drivers, trucks)

   model_distance = ModelDistance(requests)

   model_validator = ModelValidator(model_data.trucks_coordinates,
model_data.requests_coordinates, model_data.volumes_requests,
model_data.assignment_truck_request, model_data.requests_priorities,
model_data.inattention_days)

   display = Display(graphhopper_api_key)

   path_calculator = PathCalculator(graphhopper_api_key)

   distance_normalizer = DistanceNormalizer()

   minimum_calculator = MinimumCalculator()
```

Se almacenan los datos iniciales para las coordenadas de los vehículos, ya que ante la puesta en marcha y en las iteraciones, se actualizarán a ser de las mismas en cuanto a las peticiones que atenderán. Dado que se consideran los costes de vuelta a base, por un lado estarán en *path_calculator*, y para la visualización de las ubicaciones de los vehículos y peticiones iniciales, en *display*:

```
   path_calculator.store_initial_data(model_data.trucks_coordinates)

   display.store_initial_data(model_data.trucks_coordinates)

   model_validator.store_attendance_time(average_attendance_time)

   model_validator.add_limits(maximum_truck_capacity,
limit_work_minutes, maximum_inattention_days, maximum_trucks)
```

Se imprimen los datos iniciales ante el usuario o entidad usando la aplicación, si al estar usando la aplicación desde el Docker y su ejecución del servidor web hecha desde la terminal por ejemplo, se visualizarán los resultados de ejecución en iteraciones como logs creados durante la puesta en marcha:

```
   app.logger.info(f'STARTING APPLICATION.....')

   app.logger.info(f'INITIAL TRUCK COORDINATES:
model_data.trucks_coordinates')

   app.logger.info(f'INITIAL REQUESTS COORDINATES:
model_data.requests_coordinates')

   app.logger.info(f'INITIAL VOLUMES OBJECTS:
model_data.volumes_requests')

   app.logger.info(f'INITIAL ASSIGNMENT MATRIX:
model_data.assignment_truck_request')

   app.logger.info(f'INITIAL INATTENTION DAYS:
model_data.inattention_days')
```

La visualización en primeras instancias estará activa, pero podrá ser modificada para valor establecido para el tipo booleano:

```
   activate_display = True
```

La aplicación se comportará para poder ejecutarse en iteraciones, de esta forma se harán las asignaciones para cada uno de los vehículos asignándoles una petición en dicha iteración, que será repetido hasta resolver todas las peticiones:

```
   for i in range(1, 100):
```

El rango de 100 establece un número a la que nunca se llegará en condiciones normales, pero a la vez se establece algún número para que la aplicación no se vaye a iterar y no se detenga nunca como medida de seguridad.

5.3.2. Procedimiento iterativo

Dentro del bucle, se pone en marcha la lógica principal.

Antes de hacer las operaciones se hace una serie de comprobaciones:

1. Una condición para la validación, que contiene los datos actualizados, cuando todas las peticiones han sido conseguido una asignación:

   ```
   model_validator.verify_termination()
   ```

   Si se da la condición como verdadera, se procede a detener el bucle:

   ```
   if model_validator.terminate:

    app.logger.info(f''FINISHED IN ITERATION: {i}'')

    break
   ```

2. Al seguir, se hace unas comprobaciones para garantizar que se cumplen los valores definidos en la incorporación de datos en el programa, para definir máximos y limitaciones:

   ```
   model_validator.check_limit_trucks()
   model_validator.check_truck_capacity()
   model_validator.check_priorities()
   ```

3. En esta sección, se procede a calcular los minutos trabajados por cada vehículos dependiendo de las asignaciones que haya tenido, verificando que permanecen dentro del límite y a la vez, imprimiendo los datos. Se tiene en cuenta que el tiempo total calculado será de las distancias entre las rutas, el viaje a realizar para atender a la petición pero también el añadiendo el tiempo de vuelta a base porque considerar este último refleja de una mejor manera la situación realística a la que se enfrentará en cuanto al tiempo trabajado. Pero además de la suma de los tiempos de forma bidireccional, se añade ante la atendida a cada petición, un tiempo promedio proporcionado como dato del entorno de la aplicación para resolver dicha petición (cierto margen para cargar los objetos en el vehículo). Entonces, quedaría la fórmula como tál; ($T_{ruta\_para\_atender\_peticionX}$ + $T_{ruta\_para\_volver\_a\_base\_desde\_peticionX}$ + $T_{margen\_para\_atender\_a\_peticionX}$). En la primera iteración no se sigue la premisa de calcular los tiempos por no haber asignaciones:

```
if not model_distance.is_first_iteration:

 model_validator.calculate_limit_minutes_worked()

 model_validator.check_limit_minutes_worked()

 app.logger.info(f''SUM MINUTES PER TRUCK:model_validator.
sum_minutes_per_truck'')
```

4. Si se da la circunstancia de que solo haya un vehículo en operación, la optimización se detiene porque no se infiere alguna lógica para poder llegar a reconocer si es mejor atender a una petición frente a otra.

   Dado que el coste total de la trayectoria se presume ser equivalente. Aunque si cabe mencionar el hecho de que pudiera atender menos peticiones de las presentes teniendo en cuenta las condiciones de ello, pero este aspecto no es cubierto para este primer modelo.

   Se asume que el último vehículo atiende a las peticiones restantes y si no pudiera atender a más, vuelve a base por razonamientos propios:

```
if len(model_validator.assignment_truck_request[0]) == 1:

 app.logger.info(''There is only one truck left in next
iteration, the optimization stops,'' '' as the application
currently works with more than one truck.'')

 app.logger.info(''QUITTING..... '')

 break
```

Al final del bucle, se actualizan los datos permanecidos en la clase de *ModelValidator*. Para la cuál en el principio de la siguiente iteración, se traslada esa actualización en *ModelData* (actuándo como el contenedor raíz de los datos) y *ModelDistance*. Por lo que en dicha iteración, siendo la actual en la reiteración, se trabaje con los nuevos cambios (asignaciones vehículos-peticion, cambios en coordenadas para los vehículos al atender

a peticiones...etc):

```
   model_data.update(model_validator.assignment_truck_request,
model_validator.inattention_days, model_validator.requests
_priorities, model_validator.trucks_coordinates, model_validator.
requests_coordinates, model_validator.volumes_requests)

   model_distance.update(model_data.requests_coordinates,
model_data.volumes_requests,model_data.assignment_truck_request)
```

Para más información en la estructura de datos y detalles de código, consultar el apartado siguiente de la **Utilización del Trashtrek, en la cuál se define el acceso libre al repositorio del proyecto.**

La siguiente sección se compone de las operaciones requeridas para hallar los costes:

```
   path_calculator.calculate_graphhopper(model_data.trucks
_coordinates, model_distance.requests_coordinates)

   cost_estimator = CostEstimator(0.3, 0.4, 0.3)

   cost_estimator.calculate_parameter_costs(model_distance.
volumes_requests, model_distance.assignment_truck_request,
path_calculator.road_distances, path_calculator.time_estimated)
```

Como visto, se utiliza el API de GraphHopper Routing para calcular los parámetros de interés a través de la incorporación de las coordenadas y la función definida para ello. Luego se comienza inicializando los pesos para cada uno de los criterios que se definieron para el problema (*distancia, ocupación y tiempo*), y se le da una importancia adicional a la ocupación del vehículo por dos razones; de suma hasta el 1 y para el equilibrio del asunto en como para ciertos casos podría estar correlacionado el tiempo y la distancia. Por ello, **el peso de la ocupación podría compensar las importancias** para no generar sesgo en aquellas peticiones que estuvieran tán cercas que el tiempo y la distancia son mínimas.

Además se llama a la función encargada de calcular los costes de parámetros.

Antes de calcular los costes globales, es importante definir los siguientes aspectos; por un lado, se añaden los tiempos hallados por la API en un objeto de la clase para su próxima utilización en la reiteración para el cálculo de los tiempos por cada vehículo.

El siguiente paso es normalizar los costes de parámetros, despues de haber verificado los tamaños de las peticiones disponibles a ser asignadas para poder compararlas entre sí considerando que son de escalas distintas. Por lo que si solo se tuviera un dato, no es posible normalizar, se pretende aplicar otra técnica para que los tres parámetros pudieran compararse entre sí. Este, se basa en prefijar valores como máximos normalmente encontrados en las pruebas y dividir ese único valor entre dicho máximo. Al efectuar la operación, se procede a calcular los costes globales con los pesos definidos anteriormente:

```
   model_validator.add_timings(path_calculator.time_estimated)

   distance_normalizer.verify(model_distance.requests_coordinates)

   distance_normalizer.normalize(cost_estimator.distance_costs
_trucks, cost_estimator.occupation_costs_trucks,
cost_estimator.time_costs_trucks)

   cost_estimator.calculate_global_costs(distance_normalizer.
normalized_trucks_distance,distance_normalizer.
normalized_trucks_occupation,distance_normalizer.normalized_
trucks_time)

   app.logger.info(f''GLOBAL COSTS IN ITERATION i FOR UNATTENDED
REQUESTS: cost_estimator.global_costs'')
```

Ante los costes, se obtiene el mínimo con el uso del **algoritmo húngaro** y imprimiendo los resultados de las asignaciones por vehículo.

Unas de las razones por las cuales este algoritmo es bueno es el hecho de que permite analizar los mínimos obtenidos entre todos los "workers", para que no hayan disrupciones. Un ejemplo, si el mínimo local para el Vehículo Nº1 es para Petición Nº2, aunque Vehículo Nº2 también tuviera su mínimo local en Petición Nº2, no se escoge esta, sino el **segundo mínimo local** ya que ha podido determinar que *Petición Nº2 ha de ser*

atendida por el Vehículo Nº1. Así podría ocurrir que para el Vehículo Nº3, si el mínimo local era la Petición Nº2, el segundo mínimo para la petición que tiene que atender Vehículo Nº2, entonces escogerá su **tercer mínimo local**.

```
minimum_calculator.calculate_minimum(cost_estimator.global_costs)
```

```
app.logger.info( f''MINIMUM VALUES OBTAINED FOR UNATTENDED
REQUESTS: TRUCKS: minimum_calculator.row_ind, REQUESTS:
minimum_calculator.col_ind'')
```

El tópico trabajado en lo siguiente es fundamental para guardar los índices de las peticiones que no han sido atendidas, ya que aquellos que la han sido, se eliminan de la lista pero hay que tener en cuenta el número (N^o) asociado a dicha petición que fue eliminada, para hacer las operaciones de los cálculos y las estimaciones obtenidas adecuadamente, y que finalmente se tuvieran las asignaciones correctas en la **matriz de asignaciones vehículo-petición**.

Un ejemplo, si se tienen 10 peticiones definidas. La disponibilidad para ser atendido de ellos es para todos en principio, entonces se crea una lista del 1 al 10 *[1,2,3,...,10]*. Al tener las peticiones Nº 3,4 y 8 atendidas en la **matriz de asignaciones**. Los índices asociados a las peticiones dado que empiezan desde cero, serán 4,5 y 9 para la lista creada anteriormente. Entonces la lista de *modified_indexes* queda actualizada de la siguiente forma: *[1,2,3,6,7,8,10]*. Si en la próxima iteración se tuviera peticiones Nº 1,4 y 5 a ser atendidas ante el array de las **peticiones no asignadas**, estos se mapean de tál forma que la petición *Nº1 se refiere al número 2 en la lista, Nº4 se refiere al número 7 en la lista y Nº5 se refiere a número 8 en la lista.* Entonces se quedan con las peticiones correctas a ser atendidas en la iteración, Nº1,6 y 7 para la matriz de asignaciones (uno menos por empezar los Nº en la matriz desde 0).

Se intuyen los siguientes pasos, los índices asociados a Nº1,6 y 7 son 2,7 y 8. Quitando de la lista de *modified_indexes* se tiene *[1,3,6,10]*. En la siguiente iteración al tener peticiones Nº 0,2 y 3, el mapeo correcto es 0,5 y 9 (*0 se refería a 1, 2 se refería a 6 y 3 se refería a 10, luego se resta 1*). Por lo que al final se queda con la lista de peticiones disponibles *[3]*. La lista se va acortando hasta que se atiendan todas las peticiones y no quede ningún elemento en ella. Estas operaciones se realizan en la función de *check_size*, modificando las asignaciones correctamente (*col_ind*).

Luego, se guardan los resultados en una lista que se irá actualizando, mostrando al

final, de la puesta en marcha de la aplicación, las peticiones asignadas en cada iteración de forma ordenada:

```
minimum_calculator.check_size(model_distance.modified_indexes)

model_distance.extend_assigned_requests(minimum_calculator.
col_ind)

app.logger.info( f''TOTAL REQUESTS INDEXES ASSIGNED (ALSO
ATTENDED): model_distance.assigned_requests_indexes'')
```

Se actualizan las coordenadas de los vehículos para situarlos en las peticiones asignadas a las que tuvieron que atender:

```
model_data.change_truck_coordinates(minimum_calculator.row_ind,
minimum_calculator.col_ind,model_data.requests_coordinates)
```

Luego, se modifican las asignaciones de vehículo-petición (*mencionado previamente, definido como una matriz con valores booleanos de tantas columnas como vehículos y tantas de filas como peticiones, en principio teniendo 0 en todas las entradas*), para asignar un 1 a aquella combinación (*Vehículo$_i$-Petición$_x$*) en la matriz donde se ha conseguido una asignación:

```
model_data.assign_request(minimum_calculator.row_ind,
minimum_calculator.col_ind)
```

En el último segmento del proceso iterativo, se vuelve a enseñar los datos, pero ante las modificaciones hechas en la iteración actual. Así de cómo se guardan los datos ante el orden de atención a peticiones, globales, ya que para estos datos el orden es acumulativo por las iteraciones y a su vez, se comprueba que cada petición si tuviera alguna asignación, solo tuviera una como máxima (*un vehículo único*).

El estado de primera iteración será falsa si se ha llegado al final y se reitera, esta definición puede permanecer así para cualquier iteración restante a seguir:

```
app.logger.info(f''TRUCK COORDINATES:
```

```
model_data.trucks_coordinates'')

   app.logger.info(f''REQUESTS COORDINATES:
model_data.requests_coordinates'')

   app.logger.info(f''ASSIGNMENT MATRIX:
model_data.assignment_truck_request'')

   app.logger.info(f''VOLUME OBJECTS:
model_data.volumes_requests'')

   app.logger.info(f''INATTENTION DAYS:
model_data.inattention_days'')

   app.logger.info(f''REQUESTS PRIORITIES:
model_data.requests_priorities'')

   model_validator.update(model_data.assignment_truck_request,
model_data.inattention_days, model_data.requests_priorities,
model_data.trucks_coordinates, model_data.requests_coordinates,
model_data.volumes_requests)

   model_data.fetch_route_attendance_order()

   model_validator.check_unique_assignment()

   model_distance.is_first_iteration = False
```

5.3.3. Finalización

El último segmento imprime los resultados obtenidos:

```
   app.logger.info(''---------END RESULTS----------')

   app.logger.info(f''GLOBAL ATTENDANCE ORDER:
```

```
model_data.attendance_order '')

   app.logger.info(f''ASSIGNED REQUESTS INDEXES FOR EACH TRUCK
IN ORDER: model_distance.assigned_requests_indexes '')

   app.logger.info(f''ASSIGNMENT MATRIX:
model_data.assignment_truck_request'')

   app.logger.info(''VISUALIZATION IN REAL MAP SAVED. '')
```

En específico para las listas indicando la coordinación, el órden general de las peticiones, *attendance_order*, obtenido ante el recorrido total indicando el flujo de atención de ellas de una cierta manera, facilitando la organización de ellas y una mejor percepción ante la atendida. **Estos se obtienen y son actualizados en cada iteración ante las peticiones asignadas en la matriz de asignaciones**. Por otro lado, se indica el orden seguido por vehículo, esto se establece para resaltar el recorrido por vehículo. Un ejemplo; al tener 3 vehículos en el sistema, el orden de las peticiones por vehículos vendría configurado para los vehículos de forma estructurada *(Petición$_{x1}$-Vehículo 1, Petición$_{x2}$-Vehículo 2, Petición$_{x3}$-Vehículo 3, Petición$_{x4}$-Vehículo 1, Petición$_{x5}$-Vehículo 2..etc).* **Este dato es obtenido y actualizado en cada iteración a través de las asignaciones obtenidas con el algoritmo húngaro para cada vehículo.**

Cuando se tiene el parámetro activado para la visualización (por defecto lo está), se graban las visualizaciones en formato HTML, devolviendo los resultados al final sacados del programa total.

```
   app.logger.info(f''ENDING APPLICATION..... '')

   basic_route_url = display.draw_basic_graph(path_calculator.
initial_trucks_coordinates,model_data.requests_coordinates,
base_url=request.url_root) if activate_display is True else None

   ordered_route_url = display.
draw_graph_order_routes(path_calculator.initial_trucks_coordinates,
model_data.requests_coordinates,model_data.assignment_truck_request,
model_distance.assigned_requests_indexes,base_url=request.url_root)
if activate_display is True else None
```

```
   return  ''routeBasicURL'': basic_route_url,
''routeOrderedURL'': ordered_route_url,
''globalAttendanceOrder'': str(model_data.attendance_order),
''assignedRequestsOrderByTruck'': str(model_distance.assigned_requests
_indexes),
''assignmentMatrixTruckRequest'': str(model_data.assignment_truck_request.
tolist())
```

5.4. Visualización de las rutas

La parte gráfica, con la que se puede llegar a interpretar los resultados recae en dos visualizaciones guardadas en ficheros HTML, como enseñado previamente.

La primera se trata de una visualización básica, donde la idea es de mostrar los datos en el mapa y llegar a percibir las ubicaciones de forma contextualizada. Se interpreta a su vez, todas las posibles combinaciones que pudieran llegar a formarse demostrando la gravedad del problema en cuanto a la búsqueda de la ruta más óptima y para la cuál este proyecto, intenta resolver la problemática desde una perspectiva.

Se muestra un ejemplo de 3 vehículos y 10 peticiones dispersos en zonas de forma aleatoria (los vehículos ubicados en los puntos limpios), obtenido al final de la ejecución del programa como resultado en HTML como mapa interactivo, para reflejar el concepto de esta primera gráfica. 5.3

Ilustración 5.3: Ejemplo de mapa básico

La segunda gráfica proporciona la información de interés, la cuál tiene embebida los resultados de las rutas generados por la aplicación. Las peticiones se reflejan en el mismo color (ej. naranja) y indican el orden a la que se atenderá de forma global y el vehículo encargado de atenderlas, los vehículos tendrían su propio color único y en el mapa, son las ubicaciones iniciales (normalmente puntos limpios) a la cuál están presentes. El mismo ejemplo de antes pero ante la gráfica de este tipo. 5.4

En los siguientes apartados, se analizará este caso en particular en más profundidad para aclarar y resolver dudas que podrían surgir y el por qué se obtuvieron las rutas de la manera que se notan ante la ejecución de la aplicación, también por qué, ante una perspectiva, las rutas y el orden de atendida de peticiones obtenidas son las óptimas ante todas las combinaciones que pudieran surgir.

Ilustración 5.4: Ejemplo de mapa con rutas obtenidas

5.5. Despliegue de la aplicación

El proyecto tiene varías librerías y consecuentemente dependencias, para las cuales es posible que ante las características presentes en otra máquina, esto impida que se pudiera ejecutar la aplicación por distintas versiones o falta de ciertos recursos.

Por lo que la mejor opción en cuanto a tener el programa accesible al público en general sin tener que preocuparse por los componentes que pudiera tener es, como se habrá intuido hasta el momento, la utilización de la herramienta de Docker.

Como compartido previamente en cuanto a la estructuración del proyecto, el Dockerfile tiene que estar presente para indicar el procedimiento que se tendría que seguir en cuanto a la "construcción" (*build*) de la aplicación.

Al tener iniciado el Docker Desktop para el arranque del sistema, se construye la imagen del proyecto y se especifican ciertos argumentos en la línea de comandos para ejecutar la aplicación y tener la disponible en el localhost.

La definición se basa en lo siguiente (indicando la ”xx” como parámetro a definir):

```
# docker run -p xx:5000 -e
GRAPHHOPPER_API_KEY='xx' -e AVERAGE_ATTENDANCE_TIME_IN_MINUTES=xx
trashtrek:latest
```

Por lo que al ejecutar la imagen, se especifica un puerto a criterio del usuario que lo defina, así mismo la clave de la API y el tiempo promedio para atender a una petición (este último como requisito y de no generar ningún sesgo).

Más información en la explicación del funcionamiento de Docker y su relevancia en el asunto. [3]

5.6. Utilización del TrashTrek WebService

El proyecto compilado y despegado como una aplicación será utilizado como un servicio, utilizado para aquellos que quieren llevar a cabo distintas pruebas y adoptado para alguna entidad para incorporarlo en sus tareas a realizar.

El servicio cumple el diseño del software propuesto y sigue ese esquema, la cuál queda como optativa la elección del software o utilidad para hacer consultas, se especificó el Postman como ejemplo.

El acceso es de tipo *open-source* para la cuál es posible encontrar la información discutida teórica y la implementación del código seguido de ello, así como un manual para ejecutar la aplicación y realizar consultas en el siguiente vínculo:

Enlace de Github al repositorio

El proyecto en sí estará actualizado en el repositorio. Así como las indicaciones para poder acceder a la imagen y hacer una acción simple de *pull* está explicítamente mencionado, con la mención del prototipado de la estructura de los datos que se espera definir y recibir durante la ejecución.

Capítulo 6

Ejemplo

Se plantea un esquema reflejado que tiene como objetivo de ser representado como un caso real, aunque es de carácter fictioso, la intención es de mostrar la aplicación de este proyecto ante un cierto nivel práctico.

Ante un determinado día y puesta en marcha del servicio web, se encuentran los siguientes datos para la cuál establecer la optimización:

Vehículos

0. Vehículo- Punto Limpio Maspalomas (27.770183, -15.597304)

1. Vehículo- Punto Limpio Arucas (28.128852, -15.504827)

2. Vehículo- Punto Limpio El Batán (28.093512, -15.425922)

Peticiones

0. Petición- A retirar *mesa de centro* en Las Torres, Las Palmas (28.118482, -15.445237)

1. Petición- A retirar *cama individual* en Tamaraceite, Las Palmas (28.100746, -15.474429)

2. Petición- A retirar *gabinete pequeño* en Arucas, Las Palmas (28.130995, -15.515053)

3. Petición- A retirar *cama matrimonial* en Teror, Las Palmas (28.065085, -15.545675)

4. Petición- A retirar *escritorio* al lado de Playa Canteras, Las Palmas (28.138761, -15.435350)

5. Petición- A retirar *televisión plana* en Arucas, Las Palmas (28.118830, -15.526022)

6. Petición- A retirar *mesa de comedor* en Teror, Las Palmas (28.059831, -15.547753)

7. Petición- A retirar *silla de oficina* en Puerto Mogán, Lomo Quiebre (27.818894, -15.765593)

8. Petición- A retirar *estantería alta* en Vecindario (27.838164, -15.442441)

9. Petición- A retirar *espejo grande* en Puerto Rico de Gran Canaria (27.789653, -15.717291)

Otra información inicial

1. El tiempo máximo de trabajo por vehículo es de 8 horas.

2. Máximo días de peticiones desatendidas son 10.

3. Máximo número de vehículos a poder utilizar son 10.

4. Máxima capacida de vehículos es de 10 m3.

5. Máximo número de conductores disponibles es de 10.

Su visualización sigue la forma. 6.1 6.2

Como puede ser intepretado, los **iconos verdes representan las peticiones**, mientras que **los azules, los vehículos** (*los objetos dibujados solo se presentan en este ejemplo pero no forman parte en la visualización obtenida por la aplicación*).

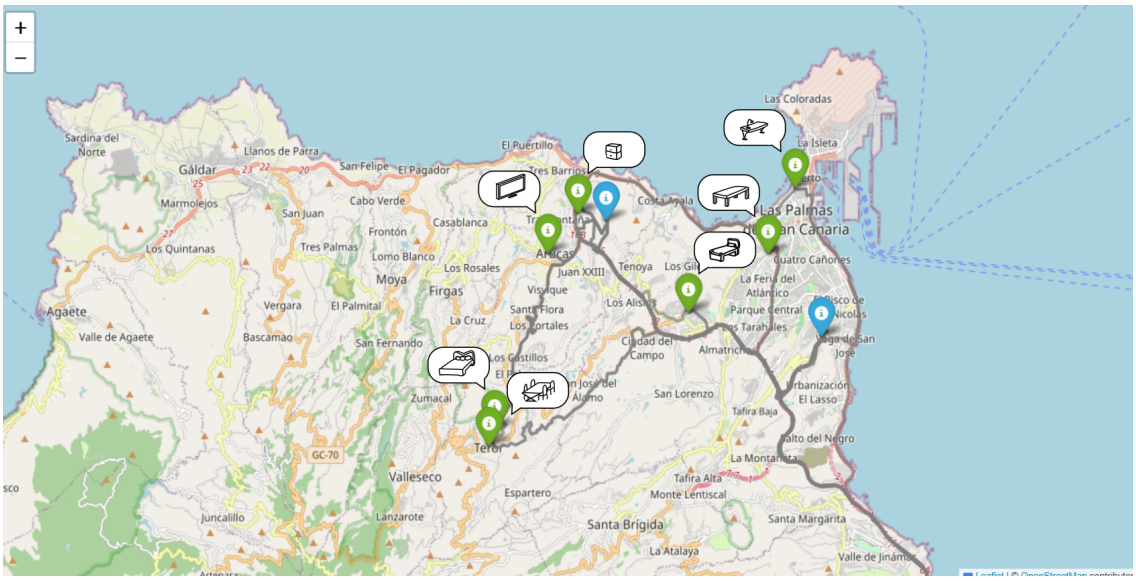

Ilustración 6.1: Mapa básico Petición-Vehículo, parte 1

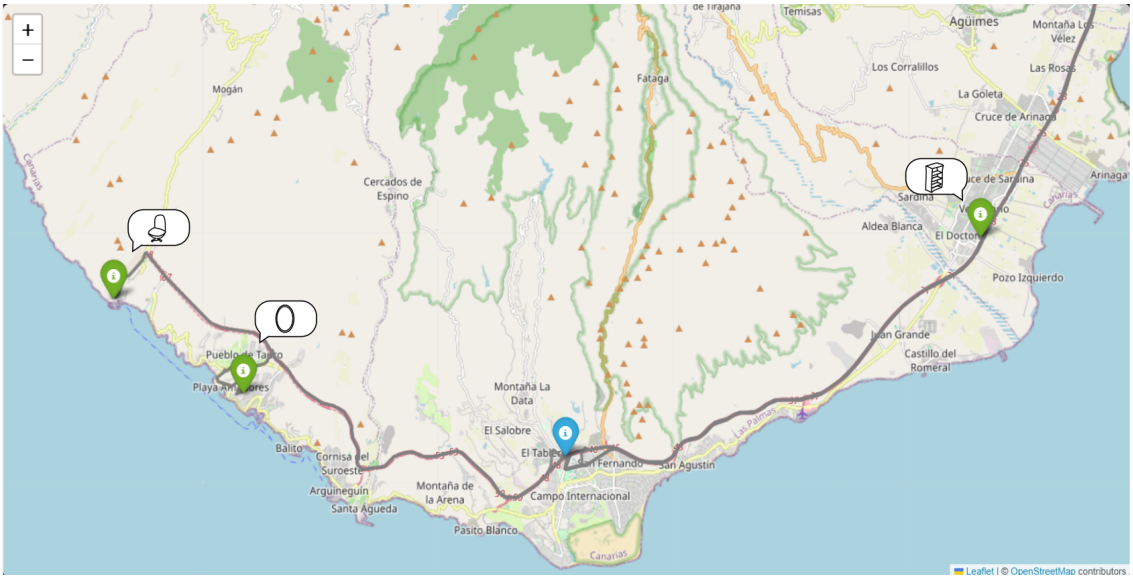

Ilustración 6.2: Mapa básico Petición-Vehículo, parte 2

Ejecutando el algoritmo con los datos iniciales mencionados, genera las siguientes interpretaciones, la matriz de asignaciones comportándose de la siguiente forma:

$$\begin{bmatrix} [1 & 0 & 0] \\ [1 & 0 & 0] \\ [0 & 1 & 0] \\ [0 & 1 & 0] \\ [1 & 0 & 0] \\ [0 & 1 & 0] \\ [0 & 1 & 0] \\ [0 & 0 & 1] \\ [0 & 0 & 1] \\ [0 & 0 & 1] \end{bmatrix}$$

Cada columna representa el vehículo, y las filas, peticiones. La enumeración de "1" significa que para ese par de Vehículo$_x$-Petición$_y$, se establece la asignación.

En resumidas, se figura:

✔ Vehículo 0 *(azul)* (*empezando las asociaciones desde 0*) = Petición 0, 1 y 4.

✔ Vehículo 1 *(rojo)* = Petición 2, 3, 5 y 6.

✔ Vehículo 2 *(verde)* = Petición 7, 8 y 9.

Además de la matriz, se obtiene más información, aquella que cuenta el ordén general a seguir, por ello se obtiene la siguiente lista (datos obtenidos del parámetro *globalAttendanceOrder*):

```
[3, 4, 8, 1, 6, 7, 0, 5, 9, 2]
```

Entonces el mapeo queda de la siguiente forma:

1. Vehículo 1 - Petición 3

2. Vehículo 0 - Petición 4

3. Vehículo 2 - Petición 8
4. Vehículo 0 - Petición 1
5. Vehículo 1 - Petición 6
6. Vehículo 2 - Petición 7
7. Vehículo 0 - Petición 0
8. Vehículo 1 - Petición 5
9. Vehículo 2 - Petición 9
10. Vehículo 1 - Petición 2

En cuanto a los resultados obtenidos de la visualizacion HTML de rutas, el orden es mostrado ***por vehículo*** *y no el atendido global, por ello se mapea de forma secuencial por índice de nombrado de vehículo. Este dato es mostrado en la variable assignedRequestsOrderByTruck en la sección de resultados. La razón por la implementación de esta forma tiene que ver suponiendo que todos los vehículos viajarán a la vez o ante un determinado horario, visualizando las asignaciones ordenadas por índice de vehículo demostrará mejor la ruta que debe seguir cada vehículo independientemente.*

En lo siguiente, se muestran las visualizaciones obtenidas de las rutas. 6.3 6.4

Ilustración 6.3: Mapa rutas Petición-Vehículo, parte 1

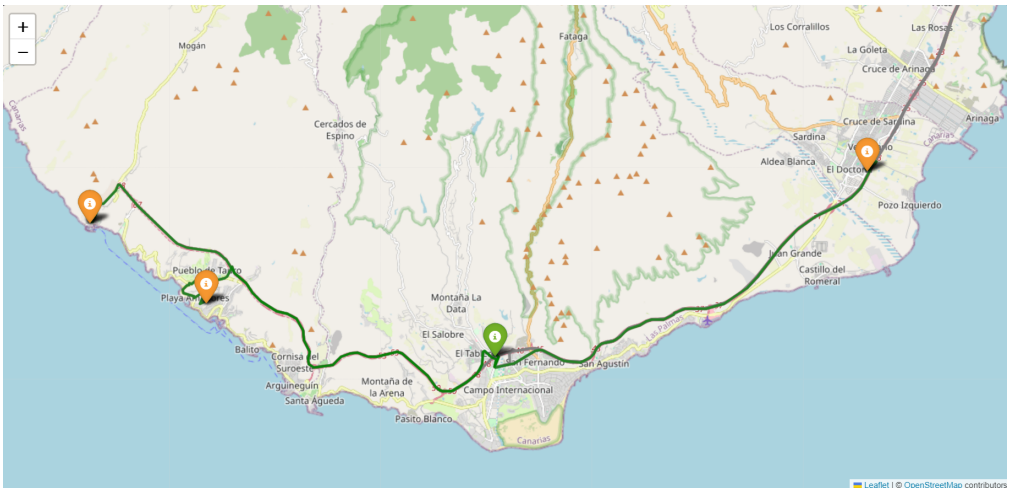

Ilustración 6.4: Mapa rutas Petición-Vehículo, parte 2

Se interpretan los resultados ante un análisis, configurando la explicación de lo obtenido y verificando su funcionalidad correcta:

- ✔ La petición con el objeto más grande a retirar (cama matrimonial en Teror) es atendida primero por el contacto más cercano, el Vehículo 1. Por otro lado, para el Vehículo 0, la ruta que sigue le lleva primero a Las Canteras (escritorio) y luego a Tamaraceite (cama individual), la razón es por priorizar los objetos de gran volumen primero a ser resueltos (dado que objetos de mayor volumen tienden a producir mayor urgencia desde la perspectiva del cliente), y finalmente va hacía Las Torres (mesa de centro, el objeto más pequeño en metros cúbicos de los dos previamente atendidos).

- ✔ Vehículo 1 sigue un camino claro, primero, va hacía la petición ubicada en Teror (cama matrimonial) como previamente dicho, y luego hacía la petición más cercana, también en Teror (mesa de comedor), posteriormente, llega a Arucas (televisión) y la otra petición también en la misma zona (gabinete pequeño), este último el más cercano a la base, terminando el tránsito.

- ✔ Vehículo 2 se mantiene en el sur y trata de atender a las peticiones ubicadas en aquella región, como esperado. Viajando primero al Vecindario para retirar una estantería alta, luego va hacía la petición más alejada de él en el sur, en Puerto Mogán. Esto se debe a que el sistema ha sido capaz de interpretar que pudiera volver a la base de manera eficiente después de atender a la siguiente petición en Puerto Rico, ya que está en el camino a la hora de volcer a la base luego, y también por el hecho de que la silla de oficina ocupa más que el espejo en cuanto al volumen en metros cúbicos.

Objetos de gran volumen son dificilmente tratados y resueltos ante la atendida de las peticiones, no solo por el tamaño que ocupan para el vehículo sino también por otros factores, como el hecho de tener que subirlas y ajustarlas en el vehículo. Aparte de otros factores como los clientes, queriendo retirarlas de la manera más rápida dado que producen molestias (ej. no será una experiencia similar para aquél individuo que tuviera una silla rota en la entrada de su casa que tener un sofá desgastado ocupando todo el pasillo).

Por ello se enfatiza y se declara como crucial, que los objetos de mayor volumen sean resueltos primero, los objetos pequeños luego pueden ser fácilmente tratados incluso si no quedase mucho espacio en el vehículo.

6.1. Problemas resueltos

El proyecto establecido y las distintas implementaciones, engloban un software capaz de solventar el caso de la optimización de rutas para la retirada de enseres voluminosos. Distintos asuntos se manifestaban a ser problemáticos y para la cuál, la aplicación pretende resolver.

Se explican las características del proyecto:

- El tiempo de trabajo acumulado por cada vehículo y los resultados obtenidos ante las rutas involucra el hecho de que **no se alejarán demasiado de las bases**. Ya que, al final del trayecto, deben depositar los objetos acumulados y preferiblemente en el punto limpio de la cuál empezaron (dado que el vehículo podría pertenecer a dicha zona, en cuanto a la organización).

  La forma de haber abordado el caso permanece en el hecho, de que, para cada uno de los costos calculados se agrega el costo para esa ubicación de petición específica regresando a la ubicación base, para esto, mientras se recuperan los datos de la distancia y el tiempo con el GraphHopper Routing API en la lógica principal de esta línea de código:

  ```
  path_calculator.calculate_graphhopper(model_data.trucks
  _coordinates, model_distance.requests_coordinates)
  ```

  Para ello, en vez de calcular las distancias y el tiempo de forma unidireccional para las peticiones en las siguientes líneas:

  ```
  road_distances[i, j] = distance / 1000 # km
  ```

  ```
  time_estimated[i, j] = time / 60000 # minutes
  ```

  Después de la sección, **un factor adicional será añadido**, realizando una nueva solicitud al API, para las distancias y tiempo que llevan las peticiones ante las coordenadas iniciales del vehículo (base) de la siguiente manera:

  ```
  road_distances[i, j] += distance_return / 1000 # km
  ```

  ```
  time_estimated[i, j] += time_return / 60000 # minutes
  ```

  Estos cambios aseguran que la distancia y el tiempo sean considerados para el recorrido total. Para las peticiones localizadas lejos de la base, si el vehículo tuviese que ir a esa ubicación, tendría que volver también dando lugar a un consumo adicional mayor. Generando por lo tanto, mayores costes de parámetros y global asociados con dicha petición.

- Puede ocurrir incluso, en ciertos casos, que solo quedase un vehículo operando, ya que por alguna casualidad los otros ya no se encuentran disponibles (ej. se ha llegado a sobrepasar el umbral de la carga máxima). Para ello, la aplicación no sigue con la optimización.

  Se considera que no hay ninguna razón por la que uno deba continuar explorando la ruta mínima, y tampoco tendría sentido calcular los costes de parámetros y costos globales, seleccionando la mejor ruta para los vehículos interceptores con el algoritmo de hungaró si solo hay una disponible. Entonces en esta instancia y al igual que ocurre con la evidencia empírica de errores ocurridos, se detiene la petición y se brindan los resultados hasta el momento. Un ejemplo, si algún Vehículo_x tuviera que atender a Petición_3 y Petición_7. Se supone que aquél único vehículo que queda por atender, debe viajar a las Peticiones 3 y 7 en cualquier sentido, de 3 a 7 o de 7 a 3, no importando a cuál atiende primero, y que por supuesto al final, debe regresar a la base.

- Por lo general, el algoritmo garantizaría que todas las peticiones reciban una asignación de los vehículos en servicio; sin embargo, cuando alguna de las restricciones no se cumple, podrían haber situaciones en las que la aplicación se detenga. Por ejemplo si el tiempo trabajado para todos los vehículos ha superado el límite definido, al no haber vehículo en servicio, la aplicación finaliza.

  Podría haber casos en las que, **las peticiones se queden sin estar resueltos y deban transferirse en la siguiente ejecución de la aplicación**. El diseño del concepto está implementado *sin su puesta en marcha en el sistema*, debido a que está diseñado para ejecutarse en un solo evento.

  Los problemas, ante peticiones no resueltas, derivan en usuarios insatisfechos, aún más, si los días se prolongan y la petición sigue desatendida.

  Para solucionar esto se hace el esquema básico que es tener una lista de prioridades (ceros) y una lista de días de desatención (ceros), y si una petición queda desatendida ese día, el día siguiente participa en peticiones para ser atendidas y sus días de desatención se suman.

  Si el número de días de falta de atención para una petición específica, excede un límite, por ejemplo; 10, significa que la petición ha sufrido 10 días de falta de atención. Inmediatamente se priorizará (0 convertido a 1 en la lista de prioridades,

pensando en un valor de bit), y antes de atender cualquier petición al volver a ejecutar la aplicación al día siguiente, **la petición prioritaria** (valor 1 en la lista) **será la primera en ser atendida**, sin importar que rasgos tuviera y donde fuera estar ubicado.

Esto asegura la satisfacción de los usuarios, de modo que el sistema resuelve todas las peticiones, es decir, los vehículos resuelven las peticiones, incluso si tardan en llegar y deshacerse del/los objeto/s.

- El total de minutos aproximadamente que trabaja cada vehículo en la iteración dada es automatizado. Estos datos se recuperan a través de la matriz de asignación de los camiones a las peticiones y las estimaciones de tiempo en minutos para cada una de esas asignaciones. Sin embargo, esta es una tarea complicada, ya que la matriz de asignación permanece estática, pero la matriz de tiempo estimado se modifica ya que las estimaciones solo se capturan para peticiones no asignadas en la iteración actual, por lo que se deben asociar los índices correctos para las asignaciones (en la matriz) a las estimaciones de tiempo.

  Se pretende mostrar la teoría del funcionamiento designado para esta función, el ejemplo se define con 3 vehículos y 10 peticiones por resolver.

  Como se mencionó, el array de tiempo estimado (estimaciones de tiempo en minutos) solo se considera para aquellas peticiones que no están asignadas, por lo que tampoco se puede asignar directamente los índices, debido a que se modifica y se tiene que asociar los índices correctos.

  Se recuerda que dentro de cada actualización del cálculo del tiempo, ***el tiempo promedio de atendida en minutos se suma ante la atendida de una petición****. Lo que significa que además de sumar el tiempo empleado en el trayecto, también se suma un tiempo extra para la resolución de la petición (desde el punto de vista del conductor en el manejo del/los objeto/s a retirar), para no generar ningún sesgo, este dato se pregunta como una variable de entorno de la aplicación.*

  Iteración 1

  Se empieza teniendo la siguiente **matriz de asignaciones vehículo-petición** (la columna siendo cada vehículo y las filas, peticiones):

$$\begin{bmatrix} [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \end{bmatrix}$$

Teniendo a la vez, un **array de tiempos estimados** de la siguiente forma (tamaño 3, para cada vehículo, cada uno con 10 valores para cada una de las peticiones):

$$\begin{bmatrix} [x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10}] \\ [y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 & y_9 & y_{10}] \\ [z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 & z_8 & z_9 & z_{10}] \end{bmatrix}$$

La variable x indica todos los valores para Vehículo_1, la y para Vehículo_2, y la z para Vehículo_3, estos se obtendrán a priori con la API y serán dados a la función, así como la matriz de asignaciones.

Como no existen peticiones asignadas en el principio, el array de tiempos estimados será considerado para todas las peticiones. Es decir, que se necesita el tiempo estimado para cada combinación Vehículo-Petición no asignada. Se tendrá además, una lista de índices para mantener el orden de tál forma:

indices = [1,2,3,4,5,6,7,8,9,10]

Iteración 2

Ahora, para la próxima iteración (*segunda, se tiene algunas peticiones asignadas a vehículos, 3 peticiones en cada iteración ya que existen 3 vehículos*) y entonces, la matriz de asignaciones se actualiza de la siguiente forma:

$$\begin{bmatrix} [0 & 0 & 0] \\ [0 & 0 & 1] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [1 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 1 & 0] \end{bmatrix}$$

Para la lista de los índices, se seleccionan los índices de filas de la matriz que tienen un "1", en este caso, será **2,8 y 10**.

Luego, se seleccionan los tiempos estimados para estos índices específicos en el asociado vehículo, como se sabe ya que cada columna en la matriz es un vehículo. Entonces, se selecciona los valores $\mathbf{x_8}$**,** $\mathbf{y_{10}}$**,** $\mathbf{z_2}$, ***x referido a Vehículo***$_1$, ***y a Vehículo***$_2$ e ***z a Vehículo***$_3$ (los lugares en la matriz donde se ubica un "1"). Como resultado, se obtiene una variable llamada *truckMinutesTotal* de esta manera:

truckMinutesTotal=[x_8,y_{10},z_2]

Se actualiza la lista de los índices, eliminando los índices asignados *(estos se refieren a los índices en la matriz de asignaciones donde no hay peticiones asignadas)*:

indices = [1,3,4,5,6,7,9]

Iteración 3

Para la siguiente iteración (tercera), el array de tiempos estimados es actualizado dejando en ello otros valores y de tamaño 7, ya que en las asignaciones previa, 7 peticiones no fueron asignadas, por lo que se tiene:

$$\begin{bmatrix} [x_1 & x_3 & x_4 & x_5 & x_6 & x_7 & x_9] \\ [y_1 & y_3 & y_4 & y_5 & y_6 & y_7 & y_9] \\ [z_1 & z_3 & z_4 & z_5 & z_6 & z_7 & z_9] \end{bmatrix}$$

Los valores son distintos y no tienen que ver con el array de tiempos estimados previo, se pone x_1,x_3..etc no para indicar que los valores del array antes se trasladan a este, sino que ahora se tienen 7 elementos para cada uno de los 3 vehículos y para ponerlo como un ejemplo de explicación del caso, siendo valores distintos.

Se asocian los valores a los índices de la lista, el órden no es 1,2,3,4..etc pero como se indica en la lista de índices: 1,3,4,5,6,7,9

Se tendrá la matriz en esta iteración de esta forma:

$$\begin{bmatrix} [1 & 0 & 0] \\ [0 & 0 & 1] \\ [0 & 0 & 0] \\ [0 & 1 & 0] \\ [0 & 0 & 1] \\ [0 & 0 & 0] \\ [0 & 0 & 0] \\ [1 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 1 & 0] \end{bmatrix}$$

Existen 3 peticiones más asignadas, se vuelve a observar la lista de los índices:

indices = [1,3,4,5,6,7,9]

Se seleccionan los índices que tiene una asignación (índice fila == 1), de esta lista, 1, 4 y 5 tienen un "1" en ello para las filas de la matriz de asignaciones. Se asocian el 1, 4 y 5 al array de tiempos estimados y los vehículos al que corresponde, la cuál se queda con x_1, y_4, z_5.

truckMinutesTotal=[x_8+x_1, y_{10}+y_4, z_2+z_5]

Se actualiza la lista de índices:

indices = [3,6,7,9]

Iteración 4

En esta iteración (cuatro), el array de tiempos estimados:

$$\begin{bmatrix} [x_3 & x_6 & x_7 & x_9] \\ [y_3 & y_6 & y_7 & y_9] \\ [z_3 & z_6 & z_7 & z_9] \end{bmatrix}$$

La matriz de asignaciones ahora tiene 3 peticiones más asignadas:

$$\begin{bmatrix} [1 & 0 & 0] \\ [0 & 0 & 1] \\ [0 & 1 & 0] \\ [0 & 1 & 0] \\ [0 & 0 & 1] \\ [1 & 0 & 0] \\ [1 & 0 & 0] \\ [1 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 1 & 0] \end{bmatrix}$$

indices = [3,6,7,9]

Averiguando cuales son las asignaciones, se concluyen 3,6 y 7. Asociando aquellos a los vehículos y los tiempos estimados se encuentra y_3, x_6, x_7.

Por lo tando, la variable se actualiza:

truckMinutesTotal=[x_8+x_1+x_6+x_7, y_{10}+y_4+y_3, z_2+z_5]

La lista de índices queda ante la eliminación:

indices = [9]

Iteración 5

En la actual iteración (cinco), el array de tiempos estimados queda de la siguiente forma:

$$\begin{bmatrix} [x_9] \\ [y_9] \\ [z_9] \end{bmatrix}$$

La matriz de asignaciones indica:

$$\begin{bmatrix} [1 & 0 & 0] \\ [0 & 0 & 1] \\ [0 & 1 & 0] \\ [0 & 1 & 0] \\ [0 & 0 & 1] \\ [1 & 0 & 0] \\ [1 & 0 & 0] \\ [1 & 0 & 0] \\ [0 & 1 & 0] \\ [0 & 1 & 0] \end{bmatrix}$$

Teniendo un único valor en la lista de índices, se asocia aquello al vehículo correcto, siendo el Vehículo_2, mapeado a ser y_9. Por lo que los minutos totales de los vehículos quedan de la siguiente forma:

truckMinutesTotal= [x_8+x_1+x_6+x_7, y_{10}+y_4+y_3+y_9, z_2+z_5]

Durante cada iteración, a parte de almacenar los tiempos estimados trabajados por cada vehículo de forma automatizada siguiendo la metodología indicada, **la restricción del tiempo siendo menor que el límite diario también es aplicado y comprobado**.

Si ante un ejemplo, el tiempo de trabajo máximo es de 8 horas (480 minutos), si cualquier vehículo excede ese cantidad, será eliminado de la operación de asig-

naciones y no será considerado como disponible a atender peticiones. Volverá entonces, a base.

- El concepto implementado tiene en cuenta las actualizaciones de ubicaciones en cada iteración ante la atendida de una petición, ya que los vehículos viajan y se localizan en distintas zonas dependiendo ante las asignaciones que tuvieran.

  Por lo tanto, si las coordenadas iniciales de los vehículos, ante un ejemplo, se comprende de la siguiente forma:

  ```
  trucks = [Truck(1, 10, (28.093512, -15.425922)), Truck(2, 10,
  (28.128852, -15.504827)), Truck(3, 10, (27.770183, -15.597304))]
  ```

  Si existe una petición en particular de esta manera:

  ```
  request = [Request(1, (27.755366, -15.604631), 0.2),....]
  ```

  Para la cuál, el algoritmo Húngaro detecta que Vehículo 3 tenía que resolver la Petición 1, las coordenadas para dicho vehículo se actualizan despues de haber obtenido la ruta mínima ante el algoritmo. Significando entonces que las coordenadas de los vehículos quedan como visto:

  ```
  trucks = [Truck(1, 10, (28.093512, -15.425922)), Truck(2, 10,
  (28.128852, -15.504827)), Truck(3, 10, (27.755366, -15.604631))]
  ```

  Ante esta modificación, se entiende que para las peticiones que quedan por ser atendidas, las distancias entre vehículos y peticiones en las próximas iteraciones a calcular, **tomarán de base las posiciones de la última modificación de los vehículos actualizados**.

  La metodología genera un sistema más realista ya que los vehículos están en **constante movimiento**, y se tienen que tener en cuenta el desplazamiento que realizan si van a atender peticiones ubicadas en distintas zonas con sus propias coordenadas.

- Como requisito fundamental, cada asociación entre vehículo y petición debe ser **única**. No se debería dar el caso que el Vehículo 1 tuviera que atender a Petición

8 y el Vehículo 2 por ejemplo, también tuviera asignada esa petición.

Cada petición única debería de tener solo un único vehículo asignado a ello, sino, existen errores de enrutamiento y la optimización falla. **El sistema no está diseñado para crear estos errores en primer lugar**, pero para tener una medida de seguridad y un "plan-b", una condición es añadida para corregir el error si de alguna manera surge por razones inesperadas.

La idea básica es de analizar cada fila de la matriz de asignaciones entre vehículos y peticiones, y comprobar cada una de las filas para examinar si **la suma de las fila es mayor que "1"**.

Un ejemplo:

$$\begin{bmatrix} [0 & 0 & 0] \\ [0 & 0 & 0] \\ [1 & 1 & 0] \\ [0 & 0 & 0] \\ [0 & 1 & 0] \\ [0 & 1 & 0] \\ [0 & 0 & 0] \\ [0 & 1 & 0] \\ [0 & 0 & 0] \\ [1 & 0 & 0] \end{bmatrix}$$

La petición 3 es asignada a Vehículo 1 y 2, el cambio a realizar es, de tener a Vehículo 1 encargado de resolver esa petición, dejando entonces la matriz modificada:

$$\begin{bmatrix} [0 & 0 & 0] \\ [0 & 0 & 0] \\ [1 & 0 & 0] \\ [0 & 0 & 0] \\ [0 & 1 & 0] \\ [0 & 1 & 0] \\ [0 & 0 & 0] \\ [0 & 1 & 0] \\ [0 & 0 & 0] \\ [1 & 0 & 0] \end{bmatrix}$$

La idea traducida a código utilizando NumPy con Python es de tál forma:

```
np.array([np.array([1] + [0] * (len(row) - 1)) if sum(row) >1 else row
for row in self.assignment_truck_request])
```

El caso es simple y resuelve el asunto, sin tener que mirar a condiciones externas (ej. la carga de peticiones para cada vehículo) y complicar más la optimización realizada declarando más operaciones.

- Cuando cada combinación de vehículo y petición resulta en un coste global, para la cuál, si el problema fuera a ser representado en 3 vehículos y 10 peticiones como ejemplo, se tendrían como consecuencia 30 costes globales distintos, esto da lugar a un problema de asignación. Se enumeran problemas con ejemplos reales:

  ¿Que ocurre si durante la selección del mínimo en los costes globales, existen dos valores mínimos para el mínimo local para dos o más vehículos?

  La siguiente matriz de costes globales, como ejemplo:

$$\begin{bmatrix} [0{,}55415517 & 0{,}4620469 & 0{,}31047253 & 0{,}03280298 & 0{,}05774575 \\ -0{,}27522498 & -0{,}06035956 & -0{,}4 & 0{,}02281897 & -0{,}27421732] \\ [0{,}51415517 & 0{,}4620469 & 0{,}3604723 & 0{,}05270298 & 0{,}05474575 \\ -0{,}25522498 & -0{,}06135956 & -0{,}4 & 0{,}02181897 & -0{,}27421732] \\ [-0{,}00247533 & -0{,}06634494 & 0{,}20986608 & 0{,}54505636 & 0{,}58169866 \\ 0{,}31074229 & 0{,}48571429 & 0{,}15676352 & 0{,}5271015 & 0{,}07265692] \end{bmatrix}$$

En la matriz, existen dos mínimos, Petición 8 para Vehículo 1 y 2 (-0.4), en la primera iteración que son exactamente iguales.

Se dice, sin embargo, que uno podría ser seleccionado como el mínimo global a atender, por ejemplo Vehículo 1 ante la Petición 8 en la iteración. Aunque, en la próxima iteración el valor mínimo para Petición 8 en Vehículo 2 se queda igual, en -0.4, entonces el Vehículo 2 atenderá a la Petición 8, que no debería de ocurrir ya que no se cumple el criterio de una asignación de vehículo única por petición.

Dado que la restricción es violada, la clase de *ModelValidator* corrige el problema con la función de *check_unique_assignment*, y modifica la matriz de asignaciones para que solo un vehículo atienda a la Petición 8.

¿Entonces, cuál es el problema?

Si solo el Vehículo 1 atiende a Petición 8, en la siguiente iteración, pase lo que pase, **el mínimo global será a favor de nuevo** para Vehículo 2 de atender a Petición 8.

Como ya averiguado, el problema vuelve a repetirse ante un ciclo, **un error de convergencia producido**.

Se puede recabar, que existen situaciones ante la escogida de los mínimos que podrían perturbar o influenciar a surgir errores en las próximas iteraciones si no se tuvieran en cuenta todos los costes y las **inter-relaciones entre ellas**.

Si existen valores mínimos idénticos para los costes globales en dos o más vehículos. **¿Cuál se prioriza?** La idea es que todos los vehículos trabajen cooperativamente, cuando uno atiende a determinadas peticiones, y otros en línea con sus asignacio-

nes, de manera que uno fuera a compensar al otro sin causar disrupciones.

Algoritmo Húngaro al rescate

El algoritmo Húngaro es una técnica eficiente para la búsqueda de la asignación óptima en un problema dado (en el caso, vehículos con peticiones) con costes asociados. Garantiza una solución óptima y trata de resolver el asunto de convergencia que pudiera ser ocasionado en enfoques iterativos.

Aplicación directa y sencilla:

```
# Representation of global costs

cost_matrix = np.array([[0.55415517, 0.4620469, 0.36047253,
0.05280298, 0.05774575, -0.22522498, -0.06035956, -0.4,
0.02281897, -0.27421732], # ... Other rows ... ])

# Using the Hungarian Algorithm

row_ind, col_ind = linear_sum_assignment(cost_matrix)

# Updating the assignment matrix

assignment_matrix = np.zeros_like(cost_matrix)

assignment_matrix[row_ind, col_ind] = 1
```

El algoritmo es especialmente útil para resolver problemas de asignación donde se busca la asignación óptima entre un conjunto de elementos fuente y otro conjunto de elementos de destino, minimizando o maximizando una función de costo asociada.

En este caso, aplicando el algoritmo húngaro a la matriz de costos global que representa los costos asociados con la asignación de vehículos a las peticiones, es capaz de encontrar la asignación óptima que minimice el costo total. Este algoritmo garantiza que cada petición se asigne a un solo vehículo y viceversa,

evitando así los problemas de duplicación y convergencia.

En resumen, el algoritmo Húngaro es una solución eficaz al problema, proporcionando una asignación óptima que resuelva los problemas de duplicación y convergencia observados.

Incorporarlo al enfoque actual mejora la calidad y eficiencia de la solución sin necesidad de cambiar la lógica fundamental del código.

Se puede encontrar una descripción clara y ejemplos de este método en el siguiente enlace. [9]

6.2. Cuestiones particulares

Un aspecto adicional extraño se podría considerar ante otros resultados obtenidos (con otros datos iniciales de lo mostrado en el ejemplo anterior) y la visualización de ello.

¿Por qué en ciertos ejemplos, un vehículo viajaría desde sur al norte (después de todo, también tiene que regresar), mientras que hay otros vehículos cerca en los lugares del norte (Las Palmas)?

La respuesta se debe al enfoque multicriterio, la distancia no es la única métrica a manejar y se debe asegurar que todos los vehículos trabajen cooperativamente. Además, cada camión **debe** funcionar; para que el algoritmo húngaro también realice asignaciones, considerará **todos** los vehículos especificados. Así está diseñada la aplicación, entonces surge una pregunta:

¿Existe la posibilidad de que menos vehículos que el especificado, podrían atender todas las peticiones de manera más eficiente?

Sí, sin embargo, esta implementación no se realiza y puede requerir que se ejecuten múltiples eventos en paralelo para encontrar la mejor búsqueda y brindar soluciones más óptimas tras los hallazgos. Bien sea, de simplemente utilizar otro método de selección más complejo para no considerar todos los vehículos, sino incluso un subconjunto de los

presentes para encontrar la ruta más eficiente. Además, puede parecer que el enfoque también requiere un uso más intensivo computacionalmente, un problema de optimización de mayor complejidad, pero podría ser parte de una *propuesta de investigación adicional*.

Sin embargo, y para contradecir la teoría anterior desde una perspectiva, se considera que los vehículos **deberían estar todos operativos**, por lo que su especificación en los datos iniciales hace que se considere sin importar dónde se encuentren en el mapa.

Debido a esto, los vehículos, aunque estén ubicados en la región sur, deben atender peticiones en la región norte si los demás vehículos de esa región están ocupados.

Una ventaja de este sistema (para todos los vehículos), entre otras, es el **equilibrio de carga**.

Distribuir el peso y el consumo global entre todos los vehículos garantiza un equilibrio óptimo, también da lugar a una estructura mejor organizada y regulada. Garantizando al mismo tiempo, que la la capacidad de los vehículos no altera tanto su velocidad y, por tanto, su tiempo (ya que no uno o algunos, o un subconjunto, están haciendo todo el trabajo y tomando todo el peso de los objetos a retirar de las demandas de peticiones, en cambio, todos).

6.3. Posibles extensiones

Ante este primer modelo orientado a resolver la problemática de optimización de la retirada de enseres voluminosos, un enfoque inicial, se pudiera añadir más criterios y mejorar el código en cuanto a la eficencia y incluso la búsqueda del mínimo.

Siendo importante considerar que la puesta en marcha de forma práctica de la aplicación y su uso constante, daría lugar a encontrar vulnerablidades y/o posibilidades de mejora dada las soluciones generadas.

Como mencionado anteriormente ante la cuestiones surgidas, es posible ante una forma, encontrar incluso un subconjunto de vehículos que pudieran resolver las peticiones en el espacio de búsqueda y no tener que utilizar todos los recursos (vehículos)

que se disponen. Este asunto a formar parte de una propuesta para el siguiente avance a realizar ante este modelo, una posterior versión.

En cada aspecto de implementación, se podría discutir una posible mejora, hasta el momento se puede concluir de una forma, que se ha creado una optimización que en definitiva, mejora el método tradicional de trabajo (atender a peticiones de forma intuitiva y ante suposiciones, con sesgo y sin decisiones conducidas por datos reales, realizadas por los conductores).

Capítulo 7

Implementación Práctica

En primer lugar, se podría crear, ante un diseño incorporativo en el mercado, una plataforma accesible al público en general, a través de una aplicación web o una aplicación móvil descargable.

Esta plataforma permitiría a los ciudadanos de Gran Canaria acceder fácilmente al servicio de retirada de enseres voluminosos de forma gratuita y sencilla, por parte del ayuntamiento de las Palmas. Los usuarios podrían completar solicitudes proporcionando los datos necesarios, lo que permitiría acumular peticiones para el servicio de TrashTrek.

Para los ciudadanos en la región, podrían ser anunciados que existe un nuevo sistema para la retirada de enseres voluminosos, que pudieran acceder a ello.

La aplicación de recogida de datos acumularía peticiones de clientes para poner a marcha el servicio de *TrashTrek*.

Se encargarían los administradores del sistema de la retirada en el ayuntamiento, para ejecutar el servicio de optimización y el proporcionar el encuentro de rutas al personal encargado de atender las peticiones y resolverlas.

Es posible entonces, tener una aplicación integrada en un dispositivo "tablet", como herramienta de visualización simplificada de las rutas para los conductores, una abstracción ante los HTML's proporcionados de Trashtrek, instalado en cada vehículo de

recogida de enseres. Con la finalidad de mostrar de manera más simplificada y intuitiva, las rutas proporcionadas por TrashTrek, facilitando así el trabajo de los conductores.

De esta manera, se podría recabar un entorno de trabajo que engloba un sistema capaz de solucionar la problemática que se enfrentan los ciudadanos ante sus enseres voluminosos. Teniendo en cuenta, que siempre hay posibilidad de escalamiento (integración incluso en otros ayuntamientos en la región) y mejora de la plataforma.

Además, el servicio web puede ser utilizado para entidades que ya trabajen con técnicas de enrutamiento de vehículos o que disponen de los recursos y quisieran resolver el asunto que llevan a cabo, integrando ciertas herramientas utilizadas en el servicio, como la API de GraphHopper y/o la visualización de rutas con Folium. Pudiendo ser parte, el proyecto de un estudio de investigación y resolución de problemas que se enfrentan de tipo Np-Completez (la combinación de VRP, TSP y Problema de la Mochila).

El proyecto se inició teniendo en cuenta, que existe actualmente una gran cantidad de enseres descartados y por lo tanto, generan un impacto negativo sobre el medio ambiente, además de una mayor relevancia si no existen protocolos que pudieran solventar esa gran demanda que se va notando en tiempos actuales.

La preocupación del aumento de residuos generados de carácter general y la consecuencia que tienen, es algo observado en todo el mundo, no solo en las Islas Canarias. Cabe generar y actualizar sistemas que pudieran solucionar eficientemente estas tareas, en especial, con el aprovecho de las últimas tecnologías que se disponen.

Capítulo 8

Conclusiones

El proyecto ha sido un esfuerzo dedicado a abordar uno de los desafíos más apremiantes que enfrentan las ciudades modernas, sobre todo empresas y servicios gubernamentales: la gestión eficiente de los residuos voluminosos. A lo largo de este trabajo, se ha explorado inicialmente en profundidad el problema de la retirada de enseres voluminosos, identificando las complejidades logísticas y los impactos negativos que este problema puede tener en la comunidad y el medio ambiente.

Al desarrollar un sistema de optimización basado en algoritmos y técnicas multicriterio, se ha demostrado que es posible mejorar significativamente la gestión de la retirada de enseres voluminosos actual, ofreciendo soluciones inteligentes y adaptativas que maximizan la eficiencia operativa, a la vez reduciendo el impacto ambiental.

8.1. Resultados

Se abrevian los siguientes resultados obtenidos ante el trabajo empleado para elaborar el proyecto fin de carrera:

- **Despliegue de un servicio web con Docker**

  La aplicación final ha sido construida como una imagen de Docker, permitiendo que todas las dependencias fueran a estar instaladas en la construcción de la

misma, esto ayuda a que los usuarios no tuvieran que tener especificaciones de software concretas en sus equipos y pudieran arrancar el servicio localmente sin problemas.

Adicionalmente, el proyecto está incluido en un respositorio Github. Donde se han sentado las bases para futuras investigaciones y desarrollos en este ámbito, ofreciendo el código fuente y documentación acerca del uso del servicio, así como información adicional de la lógica de las implementaciones. En los apartados anteriores se había adjuntado el enlace, disponible para su consulta y colaboración por parte de la comunidad académica y profesional.

La imagen está disponible tanto en Github, como DockerHub.

- **Creación de un primer modelo para tratar enseres voluminosos**

  La elaboración del proyecto ha generado un sistema que trata a ser una primera solución al método tradicional por la que se trabaja actualmente en el servicio de retirada de enseres voluminosos. El modelo se define como punto de partida, requeriendo una implementación práctica y consecuente mejoras.

- **Desarollo sostenible y eficiencia operativa**

  Uno de los aspectos que mejor se cubre en el proyecto, es el enfoque de sostenibilidad, ayudando a generar un entorno más limpio mediante un sistema inteligente que se basa en proveer una optimización con el empleo de heurística, para la cuál, se estima de generar un impacto duradero en la mejora de la calidad de vida de los ciudadanos.

8.2. Contribuciones

A nivel empresarial, el enfoque innovador ha sido implementado con éxito como un servicio web desplegado en colaboración con la empresa de consultoría tecnológica *Monentia*, lo que demuestra su viabilidad y aplicabilidad en entornos reales ante un primer acercamiento. Se proporciona una infraestructura para los ayuntamientos en cuanto a la integración del sistema.

A nivel personal, comprender el problema y aplicar soluciones fue un proceso que supuso un gran reto. Afortunadamente, se había contado con la valiosa ayuda del tutor, José Juan, y la co-dirección del socio-director de la empresa, Jose Belizón. Además de abordar problemas en un entorno empresarial, este proyecto permitió adquirir una visión integral de la ciencia y la ingeniería de datos aplicadas en proyectos comerciales. Esto no solo enriqueció las habilidades técnicas, sino que también proporcionó una perspectiva práctica sobre la implementación de soluciones tecnológicas en el ámbito empresarial.

8.3. Trabajo Futuro

Se contempla varias líneas de desarrollo y expansión que potenciarán su impacto y eficiencia.

- **Difusión del proyecto**

  Una de las primeras acciones será la difusión del proyecto en jornadas y conferencias relacionadas con la sostenibilidad, el medioambiente y el tratamiento de residuos. Estos eventos proporcionarán una plataforma ideal para presentar los resultados y beneficios del sistema, fomentando el interés y la colaboración entre diferentes entidades y expertos en el campo.

- **Integración con Ayuntamientos**

  El siguiente paso crucial será la integración y adopción del modelo por parte de los Ayuntamientos en la región. Se estima trabajar en estrecha colaboración con las autoridades locales para adaptar y personalizar el sistema a las necesidades específicas de cada municipio. Esta implementación permitirá a los Ayuntamientos mejorar significativamente la eficiencia de sus operaciones de retirada de enseres voluminosos, reduciendo costes y minimizando el impacto ambiental.

- **Mejora continua del modelo**

  La mejora continua del modelo será fundamental para mantener su relevancia y eficacia. A medida que el sistema se ponga en marcha, se generarán numerosos

insights y datos valiosos que servirán para afinar y optimizar el algoritmo. Este ciclo de retroalimentación permitirá ajustes dinámicos y mejoras constantes, garantizando que el sistema evolucione y se adapte a las necesidades cambiantes del entorno urbano y las expectativas de los ciudadanos.

En resumen, el futuro del proyecto se centrará en su promoción y adopción a nivel regional, y en el perfeccionamiento continuo del modelo a través del análisis de los datos generados por su operación. Esto asegurará que el sistema no solo resuelva los desafíos actuales de la logística de retirada de enseres voluminosos, sino que también se adapte y mejore continuamente, contribuyendo de manera significativa a la sostenibilidad y eficiencia en la gestión de residuos.

Bibliografía

[1] ACR+ (2020). Urbanrec guidelines: 'new approaches for the valorisation of urban bulky waste into high added value recycled products'. *www.acrplus.org*.

[2] Commission, E. (2018). D1.1 european wood waste statistics report. *https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5bf1792ce&appId=PPGMS*. Waste Wood Treatment.

[3] Docker (2024). Use containers to build, share and run your applications. *www.docker.com/resources/what-container/*.

[4] GraphHopper (2024). Routing api. *https://docs.graphhopper.com/#tag/Routing-API*.

[5] Moreno, M. (2023). El dilema de tirar los trastos en las palmas de gran canaria: faltan puntos limpios. *www.atlanticohoy.com*.

[6] Patel, R. (2023). Ups routing software (orion): Does it really help drivers manage their work efficiently? *https://www.upperinc.com/blog/ups-route-planning-software/*.

[7] Profesional, R. (2019). Colchones reciclados, otro reto de la economía circular. *www.residuosprofesional.com/colchones-reciclados-economia-circular/*.

[8] Route4Me (2021). Everything you need to know about ups route optimization software (orion). *https://blog.route4me.com/ups-route-optimization-software-orion/*.

[9] Wikipedia (2024). Hungarian algorithm. *https://en.wikipedia.org/wiki/Hungarian_algorithm*.

Glosario

algoritmos Conjunto de instrucciones definidas y ordenadas que permiten llevar a cabo una actividad mediante pasos sucesivos. 2

API Conjunto de definiciones y protocolos que permiten que diferentes aplicaciones de software se comuniquen entre sí. 9

Docker Plataforma de software que permite la creación, despliegue y ejecución de aplicaciones en contenedores, que son entornos aislados que incluyen todo lo necesario para ejecutar un software. 45

efecto IKEA Fenómeno en el cual las personas atribuyen un valor mayor a productos que han ayudado a crear o ensamblar ellos mismos. 3

heurísticas Métodos práctico de resolución de problemas que no garantiza la solución óptima, pero que es suficiente para alcanzar una solución en un tiempo razonable. 6

NP-completos Clase de problemas para los cuales no se conoce ningún algoritmo eficiente que los resuelva, y cuyo tiempo de resolución crece de manera exponencial con respecto al tamaño de la entrada. 2

POSTMAN Herramienta que permite a los desarrolladores probar y trabajar con APIs, facilitando la creación, prueba y documentación de solicitudes HTTP. 29

Problema de la Mochila Problema de combinatoria donde se busca maximizar el valor total de los ítems en una mochila, sin exceder su capacidad. 2

Problema de Rutas de Vehículos Problema de optimización que implica determinar la mejor manera de asignar vehículos para atender un conjunto de ubicaciones, minimizando la distancia total recorrida. 2

Problema del Agente Viajero Problema de optimización que consiste en encontrar la ruta más corta que visite todos los nodos de un grafo exactamente una vez y regrese al nodo inicial. 2

repositorio Github Plataforma de hospedaje de proyectos de software que utiliza el sistema de control de versiones Git. 32

servicio web Sistema software diseñado para soportar la interacción máquina a máquina a través de una red. 10

sostenibilidad Capacidad de satisfacer las necesidades actuales sin comprometer la capacidad de futuras generaciones para satisfacer sus propias necesidades, manteniendo el equilibrio ecológico. 6

URBANREC Iniciativa orientada a concienciar y resolver problemas relacionados con la gestión de residuos voluminosos y sostenibilidad. 8