Universitat Oberta de Catalunya

uoc.edu

Optimización de rutas para ride-pooling

Eduardo Salguero López
Máster universitario de Ingeniería Informática
Computación de altas prestaciones

Belén Bermejo Gonzalez
Josep Jorba Esteve

15 de Enero de 2021

A Marta por aturarme cando se aproximan as entregas.
Ás miñas pequerrechas por sacarme de paseo.
A Abel por axudarme a buscar ideas.
A todos os que me ofreceron o seu valioso tempo para acadar a meta.

Grazas e grazas.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Optimización de rutas para ride-pooling*
Nombre del autor:	*Eduardo Salguero López*
Nombre del consultor/a:	*Belén Bermejo Gonzalez*
Nombre del PRA:	*Josep Jorba Esteve*
Fecha de entrega (mm/aaaa):	01/2021
Titulación:	*Máster universitario de Ingeniería Informática*
Área del Trabajo Final:	Computación de altas prestaciones
Idioma del trabajo:	Español
Palabras clave	VRP, Ride-pooling, Optimización

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Podemos definir *ride-pooling* como el servicio de movilidad en el cual varios pasajeros comparten un conductor profesional. El objetivo de *ride-pooling* es proporcionar una solución que permita reducir el coste económico de los viajes (menos viajes con más pasajeros), tratar de minimizar los viajes en vacío (trayectos sin pasajero) y de ese modo ayudar a lograr ciudades más sostenibles medioambientalmente y con menos coches.

El trabajo se centra el desarrollo de una solución tecnológica que permita obtener rutas de transporte óptimas mediante la agrupación de usuarios. El problema que se trata de abordar es una especialización de VPR o problema de enrutamiento de vehículos.

Para lograr el objetivo el trabajo contextualiza e investiga el problema para abordar finalmente de forma iterativa e incremental la construcción en lenguaje Go de un servicio de optimización de rutas para *ride-pooling*. Para ello, se apoya en la implementación de un algoritmo heurístico que soluciona nuestro problema.

El resultado del trabajo incluye como entregables un servicio funcional en forma de API HTTP que permite enviar problemas de VRP y obtener su solución de forma escalable y eficiente. Este *solver* permite agrupar usuarios en vehículos proporcionando como resultado rutas óptimas, tanto de forma síncrona como asíncrona, atendiendo a las restricciones del problema.

Abstract (in English, 250 words or less):

We can define ride-pooling as the mobility service in which several passengers share a professional driver. The objective of ride-pooling is to provide a solution that allows to reduce the economic cost of journeys (fewer journeys with more passengers), try to minimize empty journeys (journeys without a passenger) and thus help to achieve more environmentally sustainable cities and with fewer cars.

The work focuses on the development of a technological solution that allows obtaining optimal transport routes by grouping users. The problem they are trying to address is a VPR (vehicle routing problem) specialization.

To achieve the objective, the work contextualizes and investigates the problem to finally approach in an iterative and incremental way the construction in Go language of a route optimization service for ride-pooling. For this, it relies on the implementation of a heuristic algorithm that solves our problem.

The result of the work includes as deliverables a functional service in the form of an HTTP API that allows to send VRP problems and obtain their solution in a scalable and efficient way. This solver allows users to be grouped into vehicles, resulting in optimal routes, both synchronously and asynchronously, taking into account the constraints of the problem.

Índice

Lista de figuras

Lista de tablas

1. Introducción

En este capítulo se da a conocer el contexto del proyecto, su justificación y los objetivos del mismo. Además se introduce el método seguido y la planificación, los productos obtenidos y la estructura del trabajo.

1.1 Contexto y justificación del Trabajo

Cabify es una plataforma tecnológica de movilidad, un MaaS (*Mobility as a Service*), que pone en contacto a usuarios, particulares y empresas, con las formas de transporte que mejor se adaptan a sus necesidades. Dentro de sus servicios ofrece *ride-hailing*. *Ride-hailing*, de forma resumida, puede entenderse como el servicio de movilidad que permite la reserva de viajes, con un coste estimado a priori, pago por uso y con un conductor profesional, a través de una aplicación móvil o web.

Actualmente provee de un servicio para empresas que permite agrupar viajeros en su plataforma de *ride-hailing*. Los objetivos son dos, por un lado reducir el coste económico de los viajes, y por otro minimizar los viajes en vacío (trayecto sin pasajero o carga más allá del propio conductor) y de ese modo avanzar en su misión de ayudar a lograr ciudades más sostenibles medioambientalmente y con menos coches.

Una parte fundamental para realizar una agrupación eficiente es disponer de un sistema que permita optimizar las rutas. Esto es, un artefacto que permita obtener un conjunto óptimo de rutas para una flota de vehículos que debe satisfacer las demandas de un conjunto dado de clientes y con unas restricciones previamente establecidas.

Esta familia de problemas es conocida como VRP *(Vehicle Routing Problem)* y existen múltiples variaciones con diferentes restricciones. Un Solver VRP es un solucionador de este tipo de problemas.

Este proyecto tiene como objetivo el estudio e implementación de un Solver VRP que permita realizar esas agrupaciones. Además de ser la base para un

solucionador de problemas de optimización de rutas relacionados con *ride-pooling* y *delivery*.

1.2 Objetivos del Trabajo

Podemos definir *ride-pooling* como el servicio de movilidad en el cual varios pasajeros comparten un conductor profesional. Por ejemplo, la persona X ha contratado un servicio de *ride-hailing* para ir de A a B. Fortuitamente, la persona Y hace una solicitud al mismo proveedor de servicios para la misma ruta o una similar. Un algoritmo combina ambas rutas y la persona Y se sube en el vehículo de ride-hailing junto con la persona X o viceversa. Es posible que el trayecto total dure algo más, pero la empresa de *ride-hailing* puede ofrecer tarifas más económicas al compartir el coche. Además, como hemos comentado, los trayectos compartidos alivian el tráfico y benefician el medio ambiente.

Actualmente la plataforma da soporte a empresas con un gran número de empleados, dichos empleados necesitan desplazarse desde múltiples orígenes a un mismo destino o viceversa. Todas estas empresas buscan un ahorro de costes económicos y optimización de tiempos, además de facilitar la vida a sus empleados mediante un transporte más eficiente y que mejore su experiencia. Una manera de lograr esto es mediante una planificación óptima (en tiempo, coste y otros parámetros) de la agrupación de pasajeros. De aquí en adelante denominaremos este problema de planificación como offline.

Por otro lado, se plantea extender el trabajo a la optimización de rutas privadas compartidas. Esto es, permitir encontrar rutas óptimas que habiliten la compartición de parte del viaje y, por lo tanto, minimizar los gastos y maximizar la eficiencia de transporte. Esta variante será denominada como online dado que el problema a resolver está basado en información en tiempo real y no por una planificación previa.

Es decir, podemos especificar el problema offline como un problema de planificación en cuanto a que las rutas se optimizan antes de ser iniciadas. Por otro lado, el problema online requiere ser resuelto de forma reactiva, esto es, cuando un

nuevo par origen-destino aparece y este viaje puede ser compartido la ruta debe poder ser re-planificada para dar servicio a un nuevo viajero.

A pesar de las diferencias el propósito es abstraerse del caso concreto y resolver ambos problemas de ride-pooling mediante una única abstracción. El objetivo del trabajo es permitir **obtener rutas de transporte óptimas mediante la agrupación de usuarios**. Tanto desde un punto de vista offline como online, de forma escalable y eficiente.

En resumen, el trabajo busca cumplir los siguientes objetivos:

- Desde una perspectiva de utilidad, generar un Solver de VRP para problemas de *ride-pooling*.
- Desde una perspectiva de futuro, desarrollar un producto funcional que sirva de punto de partida y marco de desarrollo para la solución de problemas de optimización de rutas.
- Desde un punto de vista didáctico:
    - Explorar el estado del arte de VRP.
    - Fomentar el aprendizaje de Golang.
    - Explorar técnicas de paralelización.

1.3 Enfoque y método seguido

El trabajo se inicia con un estudio de trabajos relacionados, incluida la exploración de productos existentes, esta revisión, como se indica en el capítulo 2, concluye con la necesidad de la creación de un producto nuevo.

Inicialmente para el desarrollo del proyecto se había planteado un modelo de ciclo de vida en cascada que parecía adaptarse perfectamente al cumplimiento de los hitos predeterminados por el calendario. Pronto se replantea el enfoque.

Finalmente podemos decir que el proceso de desarrollo o implementación de la solución tecnológica que nos ha permitido cumplir los objetivos del trabajo ha sido un proceso iterativo e incremental. Este modelo buscaba reducir el riesgo y la incertidumbre derivados de la falta de conocimiento sobre el dominio del problema. Nos permitiría obtener un crecimiento progresivo tanto de la funcionalidad como del conocimiento necesario para su desarrollo. Permitiéndonos, de este modo, contar

cuanto antes con una pieza de software útil que se pudiese validar y asegurar cubrir unos objetivos parciales, mucho antes de contar con un producto completo.

1.4 Planificación del Trabajo

Se programa una planificación con hitos directamente relacionada con las fechas clave de las fases del TFM:

Fecha	Actividad	Descripción
09/10/2020	PEC 1: Plan de trabajo	Elaboración del plan de trabajo, detalle de requisitos y funcionalidades y planificación del proyecto.
09/11/2020	PEC 2: Elaboración del estado del arte	Elaboración del estado del arte
20/12/2020	PEC 3: Implementación	Desarrollo e implementación del proyecto
13/01/2021	PEC 4: Entrega de la memoria del TFM	Fecha de finalización del proyecto con la entrega de la memoria, presentación final, vídeo y código depurado (producto final).

Tabla 1. Fechas claves del TFM

Las diferentes tareas se planifican del siguiente modo:

Nombre	Fecha de inicio	Fecha de fin
Trabajo de fin de Máster	24/09/2020	**15/01/2021**
Plan de trabajo	24/09/2020	09/10/2020
Brainstorming inicial	24/09/2020	05/10/2020
Definición del contexto y objetivos	04/09/2020	07/10/2020
Calendarización	07/09/2020	09/10/2020
Boceto arquitectónico	06/09/2020	09/10/2020
Elaboración del estado del arte	10/10/2020	09/11/2020
Especificación de las restricciones del problema	10/10/2020	26/10/2020
Estudio y elaboración de documentación con el estado del arte	26/10/2020	09/11/2020

Implementación	23/10/2020	31/12/2020
Análisis y diseño	23/10/2020	01/11/2020
Especificación de requisitos funcionales	23/10/2020	25/10/2020
Análisis del dominio	28/10/2020	30/10/2020
Boceto arquitectónico	30/10/2020	01/11/2020
MVP	01/11/2020	08/12/2020
Bootstrapping del servicio	01/11/2020	08/11/2020
Implementación de algoritmo usando Go	09/11/2020	29/11/2020
API-ficación de la solución	30/11/2020	08/12/2020
Versión final	09/12/2020	15/12/2020
Paralizando mediante el uso de Goroutines	09/12/2020	15/12/2020
API asíncrona	16/12/2020	31/12/2020
Memoria y presentación	30/11/2020	**15/01/2021**
Notas y agregación de los documentos generados	30/11/2020	03/01/2021
Ampliación y revisión de la memoria	03/01/2021	10/01/2021
Elaboración de la presentación	11/01/2021	12/01/2021
Video de presentación y demo	11/12/2021	13/01/2021
Memoria final	10/01/2021	**15/01/2021**

Tabla 2. Planificación del trabajo

1.5 Breve sumario de productos obtenidos

Una vez finalizado el trabajo se ha obtenido:

1. Los entregables intermedios de las PECs
2. Una implementación estable y funcional del Solver formado por:
    - Un repositorio Git
    - Un servicio escrito en Golang
    - Una imagen de Docker con el ejecutable
    - Un Workflow de GitHub Actions para CI
3. Especificación y documentación de la API
4. Una presentación
5. Esta memoria

1.6 Estructura del trabajo

El capítulo 2 contiene el estado del arte de VRP, durante el capítulo 3 se contextualiza la aproximación escogida para la resolución del problema y se describe formalmente el problema. En el capítulo 4 se presenta una breve descripción de conceptos relacionados tanto con cuestiones técnicas como de implementación que ayudará al lector a poder contextualizar los siguientes apartados.

Durante el capítulo 5 se describe el análisis funcional y los requisitos de la plataforma. Estos se completan con la descripción de la API que tiene lugar en el capítulo sexto. El capítulo séptimo contiene toda la información de diseño técnico de la solución. El octavo capítulo relata el enfoque y método de desarrollo seguido. Los capítulos 9 y 10 contienen las conclusiones y posibles líneas de trabajo. Le siguen las referencias bibliográficas, y para acabar un listado de anexos.

2. Estado del arte

El objetivo de este capítulo es presentar soluciones de software libre, productos comerciales y trabajos de investigación sobre solucionadores de VRP y especialmente sobre optimización de rutas para ride-pooling.

2.1 Metodología

Para realizar este estudio del estado último de la materia es necesaria una revisión sistemática de la literatura. Se define y se realiza un proceso de selección para estudiar el estado del arte del problema que estamos tratando. Esta sección presenta una descripción detallada del proceso de selección de literatura basado en [1].

2.1.1 Búsqueda de palabras clave

El proceso de selección tiene lugar en Google Search y Google Scholar utilizando las siguiente palabras clave: “VRP”, “VRP solver”, “ride-pooling VRP”, “carpooling VRP”, “VRP Algorithms”, “VRP solver GPU”, “VRP solver API”, “VRP pickup drop-off”. De forma concreta las palabras clave son descritas utilizando las siguientes expresiones lógicas: (ride-pooling OR carpooling) AND VRP, VRP AND Ride-Hailing, VRP AND Algorithms, VRP AND SOLVER AND (GPU OR API), VRP AND PICKUP AND (DROP-OFF OR DELIVERY). Resultando más de 350 referencias.

Además el proceso de selección incluye búsquedas por tema en GitHub. El tema escogido es "vehicle-routing-problem"[1] con 94 repositorios públicos que coinciden con este tema.

2.1.2 Filtrado de resúmenes y descripciones

Teniendo en cuenta el número inicial de resultados se ha realizado un cribado mediante la lectura de resúmenes y descripciones con el objetivo de filtrar solo los resultados más relevantes para el problema específico. Además, se han clasificado los trabajos resultantes en tres grupos: soluciones de software libre, soluciones o

[1] https://github.com/topics/vehicle-routing-problem

productos software comerciales y artículos académicos tal y como se indica en la figura [1].

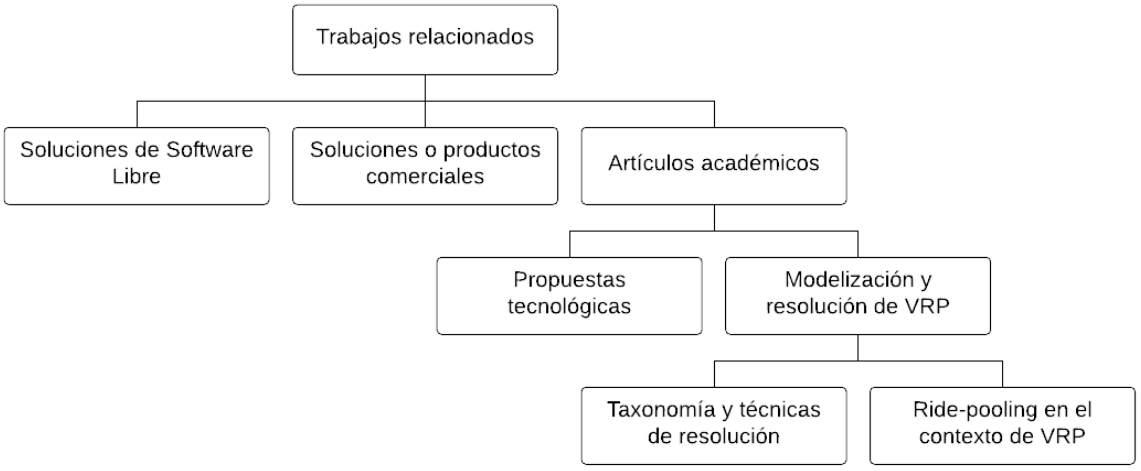

Figura 1. Clasificación de los trabajos relacionados

2.2 Trabajos relacionados

Si algo caracteriza este siglo es la movilidad, tanto de pasajeros como de bienes (desde el punto de vista de la logística de distribución). Es por lo tanto obvio que organizaciones e individuos han dedicado grandes esfuerzos con el fin de generar ventajas competitivas mediante la reducción de costes. Es por ello que el establecimiento de las rutas para vehículos de la manera más óptima ha generado un gran interés. Como resultado se han propuesto un gran número de modelos, artículos académicos, soluciones de software libre y comercial que abordan este problema con el fin de mejorar el desempeño.

En este caso nos enfocamos en el aspecto de la movilidad de pasajeros. En él se realiza una revisión del estado del arte en cuanto al Problema de Enrutado de Vehículos (VRP, por sus siglas en inglés, Vehicle Routing Problem) persiguiendo la optimización de rutas para ride-pooling.

2.2.1 Soluciones de software libre

Esta sub-sección busca proporcionar una visión sobre las soluciones de software libre relacionadas con el problema. Pretende presentar una serie de artefactos de software que permitan solucionar problemas VRP de diferentes tipologías.

2.2.1.1 An open source Spreadsheet Solver for VRP

"VRP Spreadsheet Solver" [2] es una herramienta de código abierto para representar, resolver y visualizar los resultados de los problemas de generación de rutas de vehículos (VRP). Unifica Excel, GIS público y metaheurísticas. Puede resolver problemas de generación de rutas de vehículos con hasta 200 clientes.

Desde el punto de vista de los objetivos de este trabajo esta solución plantea varios problemas. El primero es la limitación a 200 clientes (pasajeros), en segundo lugar un problema de escalabilidad y rendimiento por cuestiones tecnológicas y en tercer lugar, y no menos importante, no permite resolver las restricciones de nuestro problema.

2.2.1.2 OR-Tools

Google Optimization Tools u OR-Tools [3] es un paquete de software de código abierto, rápido y portable para para la optimización, diseñado para abordar los problemas de enrutamiento de vehículos, flujos, la programación de números lineal y entera y la programación basada en restricciones.

OR-Tools está escrito en C++, pero cuenta con wrappers para Python, C # y Java. OR-Tools permite modelar un problema en uno de los lenguajes de programación mencionados y después utilizar alguno de los solvers proporcionados, tanto comerciales como de software libre.

OR-Tools permite modelar y resolver muchos tipos de VRP:

- Problema del vendedor ambulante (TSP), el clásico problema de enrutamiento en el que solo hay un vehículo
- VRP con limitaciones de capacidad, en el que los vehículos tienen capacidades máximas para los artículos que pueden transportar.
- VRP con ventanas de tiempo, donde los vehículos deben visitar las ubicaciones en intervalos de tiempo específicos.

- VRP con limitaciones de recursos, como espacio o personal para cargar y descargar vehículos en el depósito (el punto de partida de las rutas).
- VRP con visitas canceladas, donde los vehículos no están obligados a visitar todas las ubicaciones, pero deben pagar una multa por cada visita que se cancela.
- VRP con recogidas y entregas, donde cada vehículo recoge artículos en varios lugares y los deja en otros.

OR-Tools es, a día de hoy, una de las mejores herramientas del mercado para la resolución de problemas de optimización combinatoria, pero no permite modelar ciertas restricciones de nuestro caso, o no de manera inmediata. Aún salvando este gap de conocimiento, la solución podría utilizarse como *solver* o *backend* para la resolución de nuestro problema pero sin llegar a cubrir los objetivos de este trabajo, que requieren otros niveles de abstracción, como por ejemplo la interfaz pública del servicio.

2.2.1.3 jsprit

jsprit [4] es un conjunto de herramientas de código abierto basado en Java para resolver TSP y VRP. Según los promotores es ligero, flexible y fácil de usar, y se basa en una única meta heurística para resolver los problemas.

Si bien parece una buena solución, y a priori podría llegar a modelarse nuestro problema con jsprint, esto requeriría realizar cambios en el código lo requiere una gran curva de aprendizaje, principalmente motivado por la mala documentación del proyecto. Por otra parte, y dado que emplea una única metaheurística, parece que una solución ad-hoc debería ser más eficiente y adecuada para nuestro problema.

2.2.1.4 OptaPlanner

OptaPlanner [5] es un *Constraint Satisfaction Solver*, es decir, posee un motor que permite optimizar la planificación de recursos. Resuelve los problemas clásicos de los CSP - del inglés Constraint satisfaction problem o problema de satisfacción de restricciones-: asignar un conjunto limitado de recursos limitados (empleados, activos,tiempo y dinero) para proporcionar productos o servicios a los clientes, VRP, turnos de los empleados de turnos, planificación de trabajo, etc.

OptaPlanner está escrito en Java, y en principio, permite a los programadores resolver problemas de optimización de un modo eficaz. OptaPlanner oculta al

programador sofisticados algoritmos de optimización y heurística. Es software *open-source*, y está publicado bajo la licencia Apache Software. Además está pensado para integrarse y embeberse con otras tecnologías Java.

De forma similar a jsprit esta solución es demasiado generalista y resulta complicado/imposible modelar ciertas restricciones de nuestro problema sin realizar adaptaciones en el código.

2.2.1.5 Open-VRP

Open-VRP [6] es un *framework open-source* para el modelado de VRP. La librería es extensible y está escrita en *Common Lisp Object System*. Permite añadir algoritmos propios escritos desde cero o utilizar una implementación retocada de Búsqueda Tabú.

Open-VRP tiene una orientación académica, su objetivo es permitir experimentar con metaheurísticas desde un punto de vista de las matemáticas/OR (investigación operativa) sin requerir el conocimiento en C++ o Java, lenguajes en los que se encuentran implementadas la mayoría de soluciones libres de VRP.

La visión de Open VRP es ser un *toolkit* para la comunidad de OR que permita ocuparse de la aplicación de métodos analíticos avanzados para ayudar en la toma de decisiones.

En esta ocasión la falta de conocimientos de Lisp es una barrera importante que hace descarta la solución por una cuestión tecnológica.

2.2.1.6 VROOM - Vehicle Routing Open-source Optimization Machine

VROOM [7] es un motor de optimización de código abierto escrito en C ++ 17, tiene como objetivo proporcionar soluciones a varios problemas de enrutamiento de vehículos (VRP) con tiempo de cálculo reducido.

VROOM nace en VERSO, una pequeña empresa que se dedica a resolver problemas de optimización, para construir su API de optimización de rutas.

De nuevo el lenguaje es una barrera importante que hace descarta la solución por una cuestión tecnológica.

2.2.1.7 A Vehicle Routing Problem solver

El proyecto [8], escrito en Rust, nace con el foco además de en el rendimiento, con la idea de ser diseñado para ser extensible con la intención de soportar un amplio conjunto de variaciones de VRP. Esto se logra a través de varios puntos de extensión: restricciones personalizadas, funciones objetivo, criterios de aceptación, etc.

Por defecto provee una única metaheurística basada en un algoritmo evolutivo multiobjetivo.

El proyecto es experimental por lo que no se aconseja el uso en producción.

2.2.1.8 Consideraciones generales

En general las soluciones aportadas ofrecen solvers solventes, pero más o menos difíciles de integrar, en todos los casos se requieren de capas de abstracción adicionales para cumplir el objetivo del proyecto. Desde la traducción del problema entre los diferentes dominios, a todo lo relacionado con la interfaz pública de nuestro optimizador de rutas para *ride-pooling*.

En algunos casos, dichos artefactos podrían llegar a ser utilizados como piezas de nuestra solución, pero en ningún caso sustituirla por completo. Por otro lado, y dada la naturaleza y las restricciones de nuestro problema, la complejidad de modelar el problema con estas soluciones resulta harto complicada.

Por otro lado, nuestra propuesta debe intentar aportar una solución tecnológica más moderna, específica y por lo tanto simple. Esto permitirá poder iterar y mantener el proyecto de forma más sencilla y sin partir con la enorme deuda técnica y de conocimiento, a bajo nivel, de las implementaciones listadas.

Para acabar considero que dado que se necesita optimizar el rendimiento, una solución ad-hoc siempre (debería) ser más eficiente que usar una solución genérica.

2.2.2 Soluciones o productos software comerciales

En esta sub-sección proporcionaremos una visión sobre las soluciones comerciales. Existen más productos de software comercial de los mencionados pero se encuentran centrados en la gestión de flotas y el delivery. Por otro lado, la mayoría no ofrecen APIs que permitan integrar sus solvers.

2.2.2.1 Routific

Routific [9] es una plataforma de software de optimización de rutas que ayuda a las empresas de reparto a planificar sus rutas de manera más eficiente, ahorrando tiempo y hasta un 40% en combustible.

Routific es un SaaS y cuenta con la *Routific Engine API*. Dicha API permite automatizar y optimizar las operaciones de enrutado y planificación mediante la integración de sus algoritmos mediante llamadas http.

Desde la perspectiva de los objetivos de este trabajo, los principales problemas de *Routific* son, por un lado está centrado en el *delivery*, no en el *ride-pooling*. Por otro lado se tarifica por número de visitas (direcciones únicas en 24h) por lo cual no es escalable económicamente para un problema con millones de direcciones diferentes al mes. Por otro lado, se han realizado pruebas y se han visto agrupaciones raras con un alto volumen de “pasajeros”. Además utiliza su propio sistema de rutas para el cálculo de tiempos y distancias por lo que no permite utilizar otro *backend* más especializado para el cálculo de costes. Para acabar no permite modelar ciertas restricciones de nuestro problema.

2.2.2.2 Highway

Highway [10] es un software de planificación de la última milla y optimización de rutas para empresas de distribución. *Highway* es un producto de *SmartMonkey*, empresa fundada en Barcelona en el año 2015.

Del mismo modo que *Routific*, *Highway* proporciona una API para la optimización de rutas que permite integrar su solver en otras soluciones. Mediante esta API se obtiene una asignación óptima de trabajos a vehículos.

El principal inconveniente de *Highway Route Optimization API* es que no permite modelar algunas de las restricciones de nuestro problema. Además el coste es por

vehículo/mes lo que hace que no sea escalable económicamente cuando hablamos de flotas de centenares de miles de vehículos.

2.2.2.3 HERE Tour Planning

HERE Technologies [11] es una compañía multinacional que desarrolla el servicio de mapas y geolocalización del mismo nombre. Fundada en 1985, *HERE Technologies* cuenta con más de 9.000 empleados en 54 países. La compañía es la mayor empresa de geolocalización a nivel global.

HERE Tour Planning [12] es una API REST que se encarga de la optimización de rutas de múltiples vehículos. Proporciona una forma de resolver diversas tipologías de VRP. Puede utilizarse para planificar una flota de vehículos de *delivery* o servicio, teniendo en cuenta varias limitaciones como ventanas de tiempo, capacidad, coste, tiempo/distancia máxima de viaje, descansos para el conductor, etc.

Tour Planning soporta las siguientes variaciones de VRP:

- VRP con capacidad (CVRP)
- VRP con ventanas de tiempo (VRPTW)
- VRP de varios depósitos (MDVRP)
- Open VRP (OVRP)
- VRP con flota heterogéneo o mixta (HFVRP)
- VRP con recogida y entrega con ventanas de tiempo (PDPTW)

HERE Tour Planning posee tres planes de precios, el mayor permite gestionar hasta 250 vehículos por mes con un límite de 1 millón de transacciones por un precio bastante razonable, pero que se queda corto para nuestras necesidades. Por otro lado, y a pesar de permitir solucionar más variaciones de VRP que los productos anteriores, no permite modelar las restricciones de nuestro problema al 100%.

2.2.2.4 Solvice OnRoute API

Solvice [13] es una empresa que proporciona productos y servicios de optimización y planificación para ayudar a las organizaciones a ofrecer valor a sus clientes de forma sostenible.

OnRoute API [14] es un grupo de potentes algoritmos de enrutamiento para resolver problemas de VRP *NP-hard* (o NP-complejo). Permite resolver VRP con ventanas temporales y con restricciones de capacidad. De nuevo tiene una política de planes de precios que permiten escalar los costes a medida de las necesidades de la organización, pero de nuevo los planes superiores parecen no estar pensados para la escala de nuestro problema. Por otro lado *OnRoute API* no permite modelar nuestras restricciones y por lo tanto resolver el problema que pretende abordar este trabajo.

2.2.2.5 Consideraciones generales

Todas y cada una de las soluciones descritas tiene un coste económico que crece de forma exponencial, lo cual no es escalable para *ride-pooling* con decenas o centenares de miles de vehículos. Por otro lado, y no menos importante, resuelven problemas similares pero no permiten modelar las restricciones de nuestro problema al 100%.

2.2.3 Artículos académicos

Durante las últimas décadas el *Vehicle Routing Problem* (VRP) y sus variantes se han vuelto cada vez más populares en la literatura académica. Podemos encontrar publicaciones sobre las características del problema, los supuestos y sus taxonomía, la formulación de modelos matemáticos y los métodos de solución utilizados para resolverlos, benchmarks, paralelización, etc.

Como se ha comentado nuestro problema tiene dos vertientes: una tecnológica, en cuanto a que requiere de software como habilitador de la solución, y otra formal, referente a la modelización del problema y sus restricciones y la construcción de una solución (creación del conjunto de rutas). Para una mejor compresión clasificaremos los artículos académicos en ambos grupos:

- Soluciones tecnológicas
- Modelización y resolución de VRP

Por otro lado, se debe tener en cuenta que cada referencia es un índice a los artículos citados, los cuales son también relevantes.

2.2.3.1 Propuestas tecnológicas

2.2.3.1.1 Carpooling, a vehicle routing approach

Carpooling, a vehicle routing approach [15] es un artículo del año 2013. El objetivo de esta tesis es implementar diferentes métodos automatizados que puedan agrupar usuarios en vehículos compartidos.

El documento presenta una definición formal de carpooling y del problema a resolver y describe las diferentes herramientas que se utilizan para la implementación de las soluciones. El artículo describe tres soluciones diferentes al problema de los viajes compartidos.

A nivel tecnológico la solución es implementada en lenguaje Scala utilizando OscarR. OscarR es un conjunto de herramientas de Scala para resolver problemas de investigación operativa (OR). Utiliza uno de sus componentes para modelar el problema como un problema de programación de restricciones. OscaR permite implementar fácilmente heurísticas de búsqueda y heurísticas de relajación adaptadas a cada problema en particular.

Una de las ventajas del artículo es que modela y propone soluciones a un problema similar al objetivo de este trabajo. Podemos considerarlo como un buen punto de partida a la hora de implementar nuestra solución, no tanto desde el punto de vista tecnológico sino desde una perspectiva de la solución formal del mismo. La principal desventaja es que el repositorio de código referenciado se encuentra off-line.

2.2.3.1.2 VRP solution using GPGPUs

VRP solution using GPGPUs [16], es un estudio del año 2017. El artículo propone determinar los requisitos que son fundamentales para un sistema de viajes compartidos, mediante el análisis de investigaciones anteriores en los campos de problemas de generación de rutas de vehículos y vehículos compartidos.

Una vez establecidos un conjunto de requisitos para el sistema, propone la implementación de un prototipo de software heterogéneo basado en tareas que puede procesar una representación del mundo real de una red de carreteras, procesar un lote de solicitudes de clientes para determinar si sus itinerarios coinciden con algunos alcance y asignar una solicitud compatible a los vehículos.

El trabajo se centra en analizar el impacto del uso de GPGPU en el diseño, la implementación y el rendimiento del sistema y del impacto positivo en el rendimiento de los sistemas VRP.

El objetivo del trabajo se enfoca en *ride-sharing*, pretende resolver un problema de agrupación de viajeros con itinerarios y horarios similares, es decir hacer coincidir vehículos y clientes en un entorno dinámico. Este punto de vista, salvo pequeñas restricciones, coincide con el problema formal que se plantea resolver en este trabajo, pero no permite cubrir los objetivos del mismo. Podemos decir que puede utilizarse como una base o inspiración en cuanto a cómo aborda el problema pero no ofreciendo una solución completa que cumpla con los objetivos del TFM.

2.2.3.2 Modelización y resolución de VRP

2.2.3.2.1 Taxonomía y técnicas de resolución

VRP engloba problemas que se relacionan con el transporte de mercancías, o personas, entre los depósitos y los clientes, donde el objetivo es diseñar un horario y ruta óptimos para que uno o más vehículos atiendan a los clientes con el mínimo coste operativo posible y la máxima satisfacción del cliente.

Por ejemplo, el plan de ruta puede intentar minimizar el número de vehículos utilizados, la distancia total recorrida, el tiempo de servicio; al mismo tiempo, puede intentar maximizar las solicitudes y los volúmenes atendidos por unidad de distancia. En la práctica, normalmente se añaden varias restricciones al modelo básico. Estas pueden incluir respetar la capacidad del vehículo, aplicar un cierto orden de visitas o adherirse a los horarios de servicio de los clientes y a las horas máximas de trabajo de los conductores. Otras restricciones incluyen los tipos de servicio para los clientes (entrega y/o recogida), número de depósitos (uno/dos/múltiples), tipos de vehículos operativos (homogéneos/heterogéneos), disponibilidad de conductores (horarios de inicio fijos/variables), la necesidad de que los vehículos deban regresar al depósito (o no), y si la información relativa a la ruta y a las solicitudes se conocen de antemano o se descubren en tiempo real.

Como se ha comentado y dada la gran cantidad de investigaciones publicadas que abordan los diversos tipos de VRP [17][18][19], resulta cada vez más difícil realizar un seguimiento de todos los tipos de problemas y métodos de solución subyacentes.

En la figura [2] mostramos una clasificación general de los problemas básicos de enrutamiento y planificación y sus interconexiones:

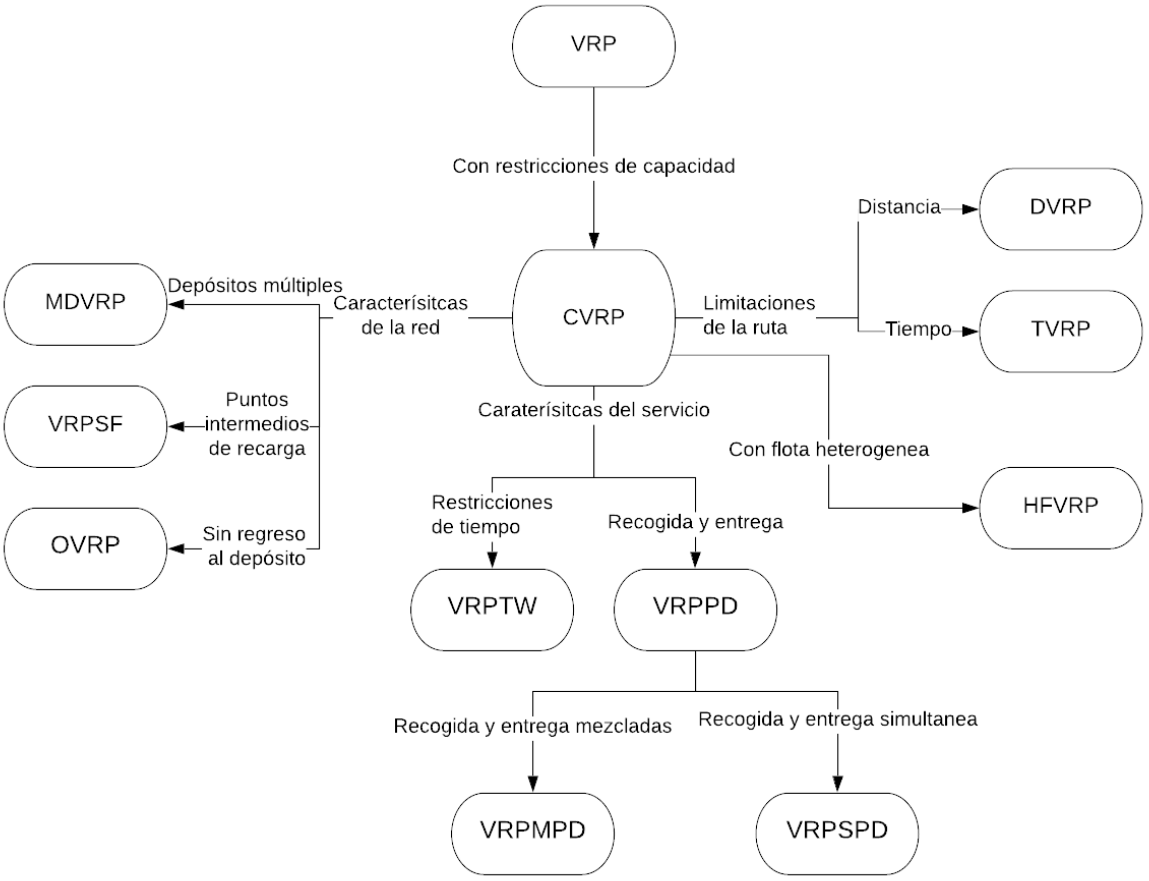

Figura 2. Taxonomía VRP

Como generalización del famoso TSP, el VRP es un problema *NP-hard* [20] [21]. Este hecho, junto con las restricciones y consideraciones prácticas antes mencionadas, aumenta la dificultad de manejar las diferentes variantes de los problemas de enrutamiento de vehículos y contribuye a la existencia de una gran cantidad de modelos y algoritmos que tratan este problema [22][23][24][25][26][27].

Presentamos en la figura [3] una clasificación sobre las técnicas para la solución de problemas de VRP [28][29]:

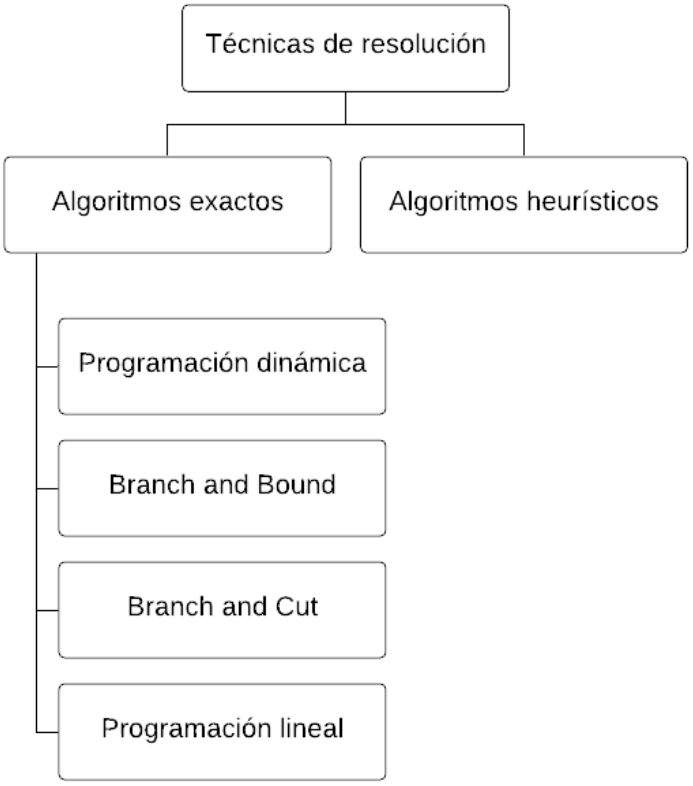

Figura 3. Algoritmos exactos.

En la figura [4] se expande la rama de heurísticas simple y en la figura [5] la rama de metaheurísticas para mejor visibilidad.

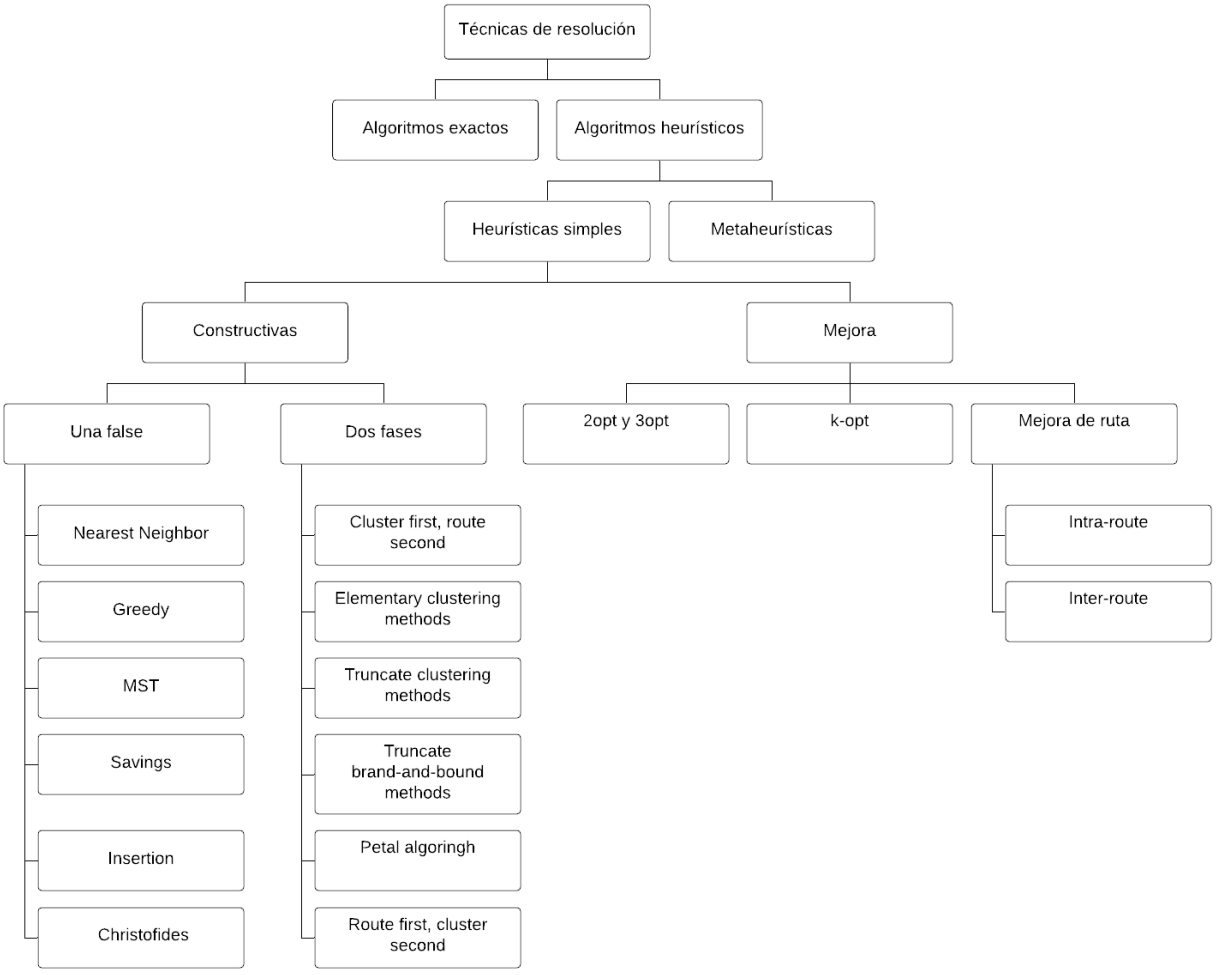

Figura 4. Heurísticas simples

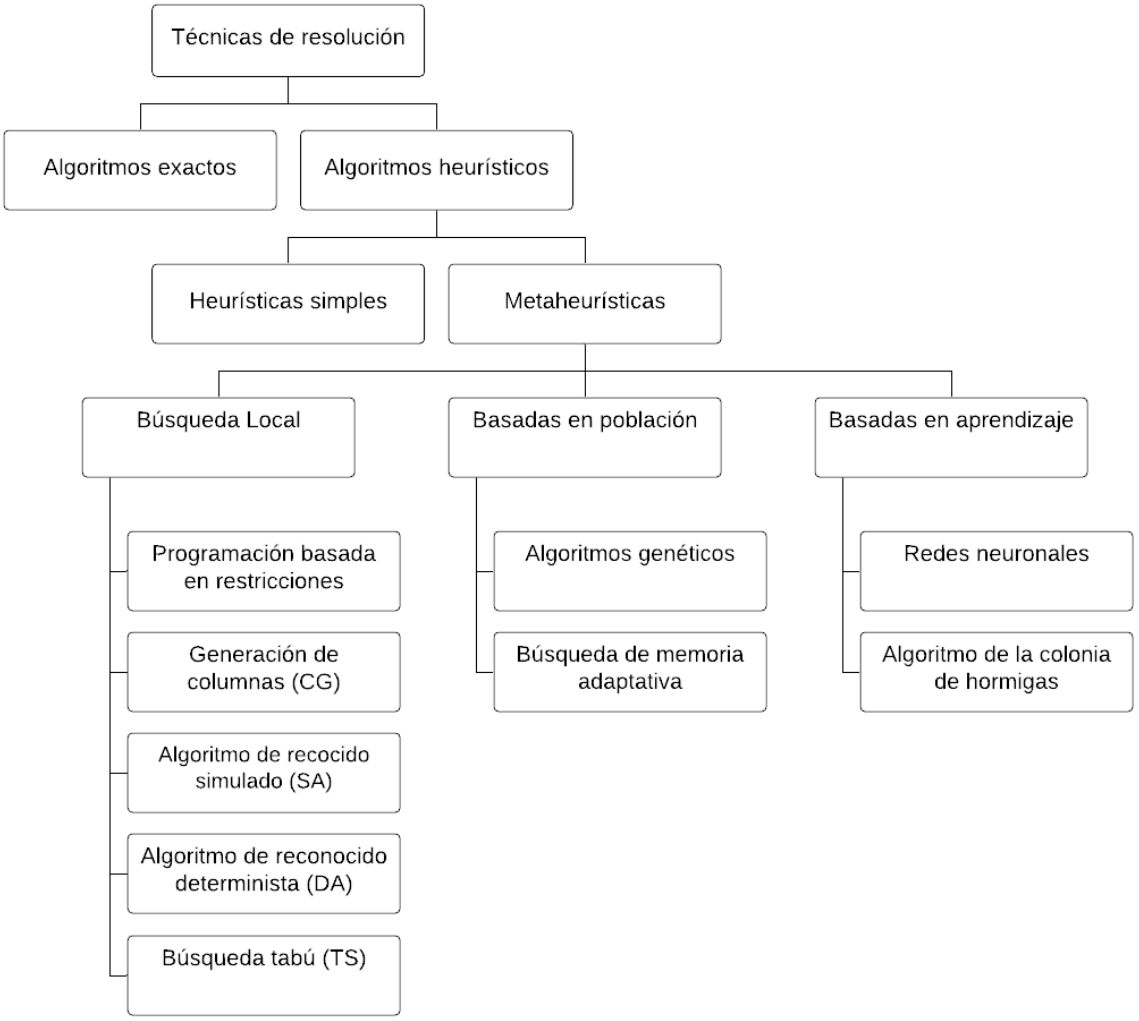

Figura 4. Metaheurísticas

2.2.3.2.2 Ride-pooling en el contexto de VRP

Una vez alcanzado este punto de generalización es necesario contextualizar formalmente nuestro problema, para ello lo describiremos utilizando la taxonomía mencionada. Nuestro caso es un tipo de VRP con recogida y entrega (PD) con ventanas temporales (TW) y flota heterogénea (HF), multi-depósito (MD) y abierto (O).

Es de recogida y entrega por razones obvias, con ventanas temporales dado que el nivel de servicio implica que el pasajero no puede superar un tiempo máximo de viaje, esto es, debe llegar a su destino dentro de una ventana temporal. De flota heterogénea en cuanto a que en la flota existen vehículos con diferentes capacidades y el número es limitado. Multi-depósito dado que los vehículos pueden partir de orígenes diferentes (lo más probable que cada vehículo tenga su propia

procedencia). Es abierto dado que los vehículos no tienen que regresar a la base después de prestar el servicio.

En [30] los autores proponen una investigación de los enfoques heurísticos y metaheurísticos para resolver VRPPDTW (VRP con pickup y delivery con ventanas de tiempo). [23] presenta un framework exacto para resolver VRP con capacidad, VRP con ventanas temporales, VRP con pickup y delivery y ventanas temporales y VRP con flota heterogénea (incluido multi-depósito). [22] propone una heurística unificada capaz de resolver cinco variantes de VRP: el problema de enrutamiento de vehículos con ventanas de tiempo (VRPTW), con capacidad (CVRP), multi-depósito (MDVRP), dependiente del sitio (SDVRP) y el problema abierto de enrutamiento de vehículos (OVRP). Con [31] volvemos a abordar VRPPDTW con un algoritmo híbrido mejorado.

2.2.3.3 Consideraciones generales

Hemos presentado un breve resumen de la enorme producción académica relacionada con VRP en general y en particular con los subtipos que caracterizan nuestro problema. En este estado del arte se exponen las bases y el marco formal que permite modelar nuestro problema, proponer una solución y en última instancia guiar la implementación de nuestro *solver*. Por otro lado, no se proporciona una solución completa que permita lograr el objetivo del trabajo desde un punto de vista tecnológico.

3. Propuesta para la solución

Este capítulo pretende abordar las posibles aproximaciones para solucionar el problema de VRP subyacente, justifica la técnica de resolución elegida y presenta formalmente el problema.

3.1 Una aproximación mediante VRP

Como hemos comentado nuestro problema se resume en obtener rutas de transporte óptimas mediante la agrupación de usuarios.

El problema de enrutamiento de vehículos o VRP data del año 1959 y fue introducido por Dantzig y Ramser [32], quienes describieron una aplicación real de la entrega de gasolina a las estaciones de servicio y propusieron una formulación matemática.

VPR consiste en encontrar rutas para un conjunto de vehículos que deben visitar una serie de localizaciones cumpliendo un conjunto de reglas para satisfacer las demandas de un grupo dado de clientes. VRP es un problema de optimización combinatoria y de programación de enteros.

Las soluciones óptimas a este tipo de problema para pequeñas instancias (20 a 50 clientes) pueden ser encontradas en tiempo razonable (polinomial) mediante programación lineal entera. Sin embargo, determinar la solución óptima es un problema *NP-hard* [33][34] lo que implica que no se disponga, para grandes instancias, de algoritmos exactos que encuentren soluciones óptimas en un tiempo polinomial determinístico. El tiempo de resolución de estos algoritmos crece exponencialmente.

Debido a que la resolución exacta basada en una formulación lineal podría generar dificultades para encontrar una solución sencilla y factible, podría modelarse este tipo de problemas mediante el uso del paradigma de programación con restricciones *(constraint programming)*. La programación por restricciones es un paradigma de programación donde las relaciones entre las variables son expresadas en términos de restricciones (ecuaciones). La programación con restricciones se ocupa primero de reducir el espacio de posibles soluciones y, luego, de ejecutar métodos específicos de búsqueda.

Otra posible aproximación para la resolución de un problema de optimización combinatoria como VRP sería la utilización de un algoritmo heurístico o metaheurístico. En estos casos lo habitual es que la búsqueda de una buena solución del problema se divida en dos fases: construcción de la solución y mejora de la solución. La construcción de la solución se refiere al proceso de crear una o más soluciones factibles iniciales que actuarán como punto de partida. En esta fase, una heurística de construcción comienza desde una solución vacía y agrega gradualmente componentes de la solución a la solución parcialmente construida, hasta que se genera una solución factible. Por otro lado, la mejora de la solución intenta modificar gradualmente la solución de inicio, en función de alguna métrica predefinida, hasta que se obtenga una determinada calidad de la solución o haya transcurrido un período de tiempo determinado.

En el contexto de la búsqueda de una solución de buena calidad, usualmente usamos el término espacio de búsqueda para referirnos al espacio de estados de todas las soluciones/estados factibles que son accesibles desde la solución/estado actual. Para encontrar una solución de buena calidad, un algoritmo heurístico o metaheurístico se mueve de un estado a otro, es decir, de una solución candidata a otra, a través de un proceso que a menudo se denomina Búsqueda Local (*Local Search* o LS).

Durante LS, la nueva solución generalmente se genera dentro del vecindario de la solución anterior. Un vecindario cercano N(x) de una solución es un subconjunto del espacio de búsqueda que contiene uno o más óptimos locales, las mejores soluciones en este vecindario. Para evaluar cada solución candidata x y determinar su costo en comparación con otras soluciones se utiliza una función de coste f(x). La mejor solución dentro del espacio de búsqueda general se llama la solución globalmente óptima, o simplemente la solución óptima.

Además, se han formulado algunos métodos de búsqueda local bajo el marco metaheurístico, en el que un conjunto de reglas predefinidas o técnicas algorítmicas ayudan a guiar una búsqueda heurística hacia la búsqueda de soluciones de alta calidad para problemas complejos de optimización. Entre las técnicas metaheurísticas más famosas se encuentran: Hill Climbing (HC), Annealing Simulado (SA), Algoritmos genéticos (GA), Optimización de colonias de hormigas (ACO), Búsqueda tabú y Búsqueda de vecindad variable (VNS).

Por la naturaleza de nuestro problema sabemos que no necesitamos obtener el resultado exacto o la ruta óptima con un alto coste temporal, lo que necesitamos es una buena solución en un tiempo razonable. Debido a esto en lugar de tratar de resolver nuestro problema *NP-hard* con un programa de programación lineal entera o mediante programación con restricciones hemos implementado una metaheurística de construcción basada en datos.

Nuestra heurística está basada en [35] y básicamente agrupa secuencialmente a los clientes descartando a los que produzcan una violación de las restricciones. Cuando una agrupación es factible se guarda y se inicia un nuevo clúster, el proceso se repite hasta que se enrutan todos los clientes. Por otro lado, y dada la naturaleza heterogénea de nuestra flota, el algoritmo guía la solución buscando un correcto bin-packing cuando el asset o vehículo está infrautilizado mediante una restricción adicional.

Cabe destacar que la metaheurística propuesta es solo una de las posibles y que el objetivo del trabajo no es la implementación de un algoritmo concreto, sino dar un primer paso y preparar el marco de trabajo para resolver problemas de optimización de rutas de *ride-pooling* fijando un contrato para nuestro *solver* y abstrayendo la solución del algoritmo implementado.

3.2 Descripción formal del problema

Nuestro caso es un tipo de VRP con recogida y entrega (PD) con ventanas temporales (TW) y flota heterogénea (HF), multi-depósito (MD) y abierto (O).

Es de recogida y entrega en cuanto a que un viajero sube y baja del vehículo en distintas localizaciones, con ventanas temporales dado que el nivel de servicio implica que el pasajero no puede superar un tiempo máximo de viaje, esto es, debe llegar a su destino dentro de una ventana temporal. De flota heterogénea en cuanto a que en la flota existen vehículos con diferentes capacidades y el número es limitado. Multi-depósito dado que los vehículos pueden partir de orígenes diferentes (lo más probable que cada vehículo tenga su propia procedencia) y abierto dado que los vehículos no tienen que regresar a la base después de prestar el servicio.

Una instancia de nuestro problema se caracteriza por los siguientes datos:

- Una flota de vehículos $F = \{1,....n\}$, $F^+ = F \cup \{0\}$, cada uno de capacidad Q
- Un conjunto de clientes $I = \{1,....n\}$, $I^+ = I \cup \{0\}$
- Para cada cliente $i \in I$ se conoce la demanda q_i, tal que $q_i > 0$ para una ubicación de recogida, $q_j < 0$ para una ubicación de entrega y $q_i + q_j = 0$ para las ubicaciones de recogida y entrega del mismo cliente $(q_0 = 0)$.
- Costo de viaje c_{ij} entre cada par de elementos $i,j \in I^+$ en donde el coste es un par tiempo, distancia.

Podemos formalizar nuestro problema como un grafo dirigido $G = (N,A)$ donde N es el conjunto de nodos y A el conjunto de arcos. A viene como $A = \{(i,j): i,k \in N\}$.

El conjunto de nodos N está formado por $N = A \cup C$. En donde A representa las localizaciones iniciales de los assets y C los pares de localizaciones de los clientes o request.

Una ruta $[R,f]$ es una secuencia ordenada de elementos de I^+ asociados a un vehículo específico f. El costo de la ruta denotado por $[R,f]$ es la suma de los costos de todos los arcos presentes en R y que denotaremos $C([R,f])$. Una solución es un conjunto de rutas $s=\{[R_1,t_1],...,[R_m,t_m]\}$

4. Definición de conceptos

En este capítulo presentaremos una serie de conceptos del dominio necesarios para poder contextualizar los siguientes apartados.

Asset

Un *asset* es un recurso con un identificador único que puede utilizarse para satisfacer un servicio. Es un vehículo o medio de locomoción que permite el traslado de un lugar a otro de personas. Un conjunto de *assets* forman una flota.

Cada *asset* dispone de una capacidad, la capacidad es la carga máxima que puede soportar. Tiene una localización inicial, un punto, establecido en la definición del problema y una final. Este es el punto en el que se realiza el *drop-off* del último viajero, si el vehículo ha realizado un servicio.

Request

Una *request* es una solicitud de servicio, posee un identificador único y un par de puntos que indican la localización de *pickup* (recogida e inicio del servicio) y un punto que indica la localización del *drop-off* (entrega y fin del servicio). Una request tiene además asignada una carga. La carga es la capacidad que consume al iniciar el servicio y en sentido contrario al finalizar el mismo.

Una *request* puede no haber sido satisfecha y por lo tanto no haberse incluido en ninguna ruta.

Restricciones

Una restricción es una limitación que se impone para la solución de un problema. Existen restricciones implícitas, como las limitaciones de orden y capacidad, y restricciones explícitas como el tiempo de servicio máximo.

El tiempo de servicio máximo especifica una restricción temporal en forma de multiplicador sobre el tiempo de la ruta directa para una *request*. Es decir, especifica que una solicitud no puede superar un tiempo máximo de viaje, esto es, debe llegar a su destino dentro de una ventana temporal especificada mediante:

$$max\ time\ =\ time\ ruta\ directa\ *\ factor$$

En donde *ruta directa* es el trayecto entre el *asset* seleccionado, el *pickup* y el *drop-off* de una solicitud.

Punto

Representa un punto codificado como una matriz de dos elementos: latitud y longitud. Siendo latitud y longitud coordenadas geográficas.

Problema

Un problema es el enunciado, el planteamiento de de la situación a resolver, en forma de conjunto de *assets* y *request* y unas serie de restricciones explícitas. Es la entrada a nuestro sistema. Es la definición del problema de enrutado que se desea resolver. Posee un identificador único.

Solución

Una solución es un conjunto de rutas y de *request* no asignadas. Está asociado a un problema mediante su identificador y cuenta con una serie de métricas:

- Número de *assets*: es el número de assets que intervienen en la construcción de la solución.
- Número de *requests*: es el número de de requests satisfechas en la solución
- Número de *requests* no asignadas: es el número de request a las que no se presta servicio.
- Duración: es la suma del tiempo (en nanosegundos) de todas las rutas que componen la solución.
- Distancia: es la suma de las distancias (en metros) de todas las rutas que componen la solución.
- Tiempo de resolución: es el tiempo de ejecución del algoritmo para la resolución del problema. No incluye el tiempo invertido en la construcción de la matriz de costes.

Ruta

Una ruta es una secuencia ordenada de *waypoints* en la que interviene un *asset* el cual presta un servicio a una serie de *requests*. Una ruta posee unas métricas relacionadas, estas son la distancia (en metros) y la duración (en nanosegundos) total del trayecto.

Waypoint

Un *waypoint* es un punto con una carga concreta en el que tienen lugar actividades relacionadas con el servicio.

Una actividad es una tupla con un identificador, que referencia a un asset o request, y un tipo. Los tipos de acciones pueden ser:

- PickUp: recogida de un viajero, inicio del servicio de una *request*.
- DropOff: entrega de un viajero, fin del servicio de una *request*.
- Start: inicio de una ruta, hace referencia al primer punto en el que el asset inicia el trayecto para ofrecer un servicio.

Estimador ETA

Un estimador de ETA (estimated time of arrival) o tiempo estimado de llegada, es un servicio que proporciona una estimación de tiempo y distancia entre dos puntos. Es el servicio que permite construir la matriz de costes necesaria para realizar las evaluaciones en los algoritmos o heurísticas de optimización.

Matriz de costes

Proporciona la distancia y el tiempo de viaje para una matriz de orígenes y destinos. Dicha distancia y tiempo son los costes de la ruta.

La matriz de costes se encuentra desacoplada del algoritmo o heurística de resolución del problema de VRP. Es un paso previo a la ejecución del algoritmo.

5. Análisis funcional y requisitos

En este capítulo se describen las funcionalidades del sistema, las interacciones con los actores externos.

5.1 Requisitos del producto

Un requisito es una característica observable de un sistema que expresa una necesidad o restricción que un *stakeholder* tiene sobre él. Los requisitos funcionales hacen referencia a la funcionalidad que debe proporcionar el sistema y los datos que tiene que conocer y guardar.

Los requisitos funcionales en forma de historias de usuario se enumeran a continuación:

1. Como usuario quiero poder obtener una solución con rutas optimizadas para un problema dado.
2. Como usuario quiero poder resolver planificaciones de forma asíncrona.
3. Como usuario quiero poder obtener una solución a un problema previamente enviado.

También nos encontramos con una serie de requisitos no funcionales que indican calidades esperadas del sistema, en concreto requisitos operacionales y de entorno:

- La API debe devolver datos en formato JSON
- La API debe ser accesible a través de HTTP

El conjunto de requisitos definen necesidades o restricciones sobre el producto. Es decir, en necesario que una vez desarrollado los satisfaga.

5.2 Casos de uso

Un caso de uso recoge el contrato entre el sistema y los stakeholders mediante la descripción del comportamiento observable del mismo. Dado que un actor es alguien que tiene comportamiento, y no solo intereses en el sistema, sabemos que de todos esos *stakeholders* o interesados, solo una parte de ellos son actores.

En nuestro caso identificamos un único actor, el usuario. Un usuario es cualquier entidad que puede interactuar con el sistema y realizar acciones.

En la figura [6] presentamos el diagrama de casos de uso:

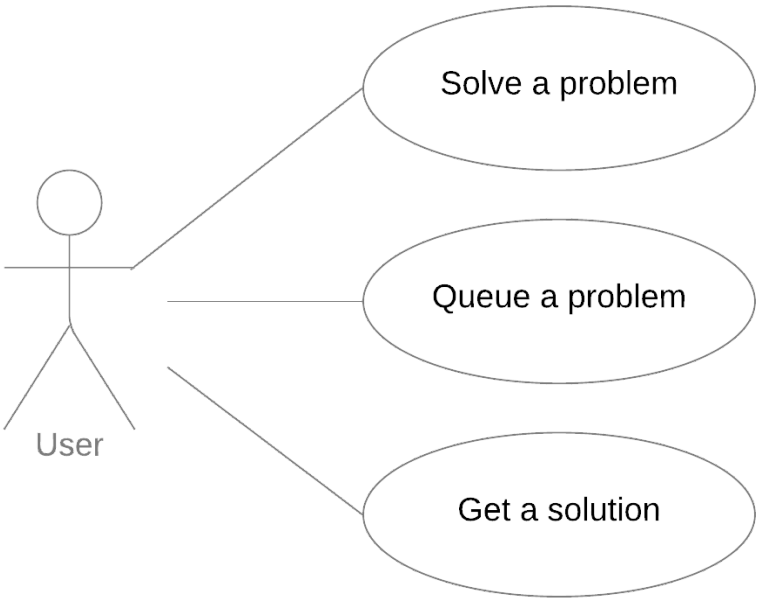

Figura 6. Diagrama de casos de uso

6. La interfaz del servicio: la API como propuesta de solución

En esta sección se presenta brevemente como el producto se expone para poder interactuar con él, o dicho de otro modo, su API pública.

6.1 Descripción

El servicio es visto desde fuera como una simple API REST que permite interactuar a través de HTTP con la plataforma. La API está compuesta por un total de 3 rutas. Para su definición se ha seleccionado la versión 3.0 de la *OpenAPI Specification* [36] –formalmente conocido como la especificación *Swagger*– , el cual es considerado el estándar de la industria para definición de APIs REST. A continuación se indicará un resumen con las rutas y detalles principales, dejando el documento completo en formato YAML en el Anexo I.

6.2 Seguridad

La API no implementa ningún sistema de autenticación. Al resultar en un servicio que no estará expuesto a internet, solo será consultable desde la red interna por lo que no se considera necesario.

Por otro lado, la implementación de algún mecanismo de autenticación como JWT o el uso de API keys es trivial y está fuera del alcance del proyecto.

6.3 Respuestas y errores

La API consume y produce objetos JSON. En el caso de los errores todos las excepciones controladas generarán una respuesta con un mensaje de error significativo y un formato como el que sigue:

```
{
 "error": "meaningful error message"
}
```

6.4 Rutas

Url	Método	Descripción
/problem		
	POST	Resuelve un problema de VRP con la descripción dada. El *endpoint* resuelve de forma sincrónica el problema de enunciado en el cuerpo de solicitud.
/problem-long		
	POST	Lanzará una tarea en segundo plano para la solución de un problema previsiblemente más costoso. Mediante el endpoint de recuperación podría hacerse ping para obtener el resultados o el estado de la tarea. Es necesario para solicitudes más grandes (más requests) que tarden más de 30 segundos. El *endpoint* devolverá inmediatamente un código HTTP 202 y el problem_id
/solution/{problem_id}		
	GET	Permite recuperar la solución de un problema previamente resuelto o un problema previamente encolado. Cuando el problema aún no se ha resuelto, el código de estado es 102 (Procesando). Al finalizar, el código de estado es 200. La respuesta es la misma que se obtendría del endpoint síncrono.

Tabla 3. Rutas de la API

En el Anexo II se detalla un ejemplo real de solicitud y respuesta de la API.

7. Diseño técnico de la solución

En este capítulo se da a conocer el diseño técnico del producto.

7.1 Arquitectura del sistema

En la figura [7] se incluye un diagrama de alto nivel que representa la arquitectura del sistema y sus relaciones con el exterior:

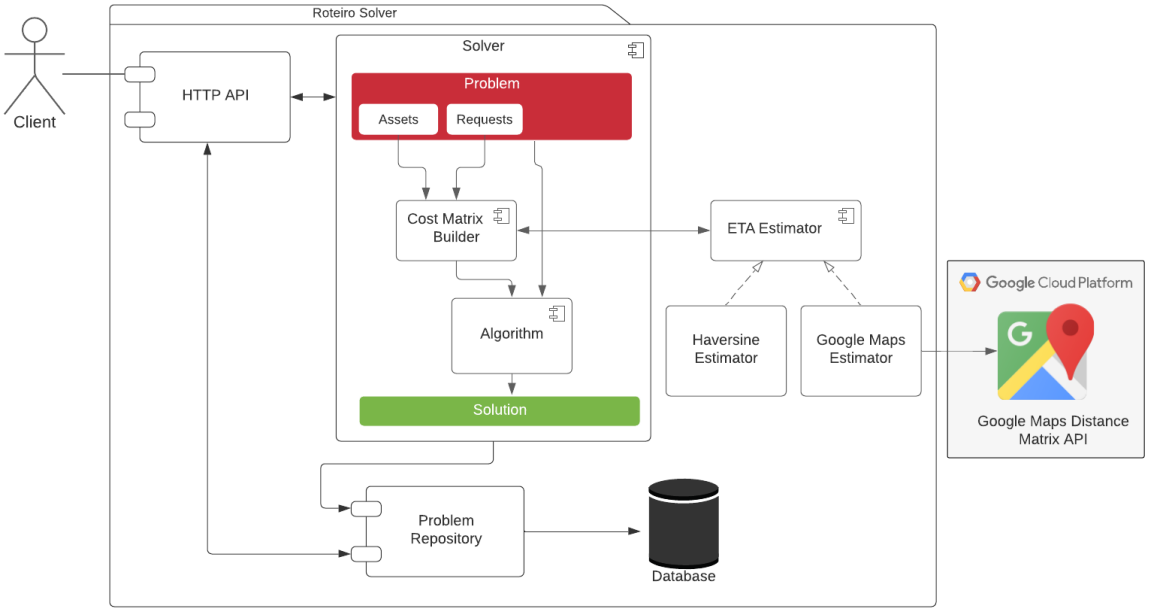

Figura 7. Diagrama a alto nivel del sistema

El sistema posee un estilo arquitectónico heterogéneo. Implementa un sistema cliente-servidor en donde la parte servidor, escrita en Golang, se compone de diferentes paquetes o componentes; los componentes se orquestan respetando el principio de inversión de dependencias.

Gracias a esto podemos hacer que el Solver, que es el núcleo de nuestra aplicación, no dependa de los detalles de implementación, como pueden ser Google Maps Distance Matrix API, un algoritmo concreto, la base de datos... Todos estos aspectos se especifican mediante interfaces.

El sistema expone su interfaz y contrato externo mediante objetos JSON a través de una API HTTP completamente especificada.

7.2 Solucionando un problema

Llamamos *solver* a la pieza de software que resuelve un problema matemático. Un solver toma la descripción de un problema de forma genérica y calcula su solución. En nuestro dominio un solver trata de resolver un problema de VRP.

El Solver es el núcleo de nuestra solución tecnológica, es nuestro componente principal. Es un orquestador que mediante una serie de colaboradores proporciona una solución al enunciado que recibe como entrada.

Lo primero que hace el Solver es crear la *matriz de costes*. La operación está paralizada desplegando *goroutinas* [37] y se apoya en una de las implementaciones de *ETA Estimator*. Esta matriz de costes se utilizará para calcular el coste de rutas y de distancias entre pares de puntos dentro del algoritmo.

El paso siguiente consiste en instanciar y ejecutar el algoritmo adecuado con los parámetros de nuestro problema, el resultado de dicha ejecución será la solución al problema y será la salida de nuestro Solver.

Para una mejor comprensión de la secuencia de acciones y procesos se añade el diagrama de secuencia [8] iniciado con la entrada de un problema al sistema:

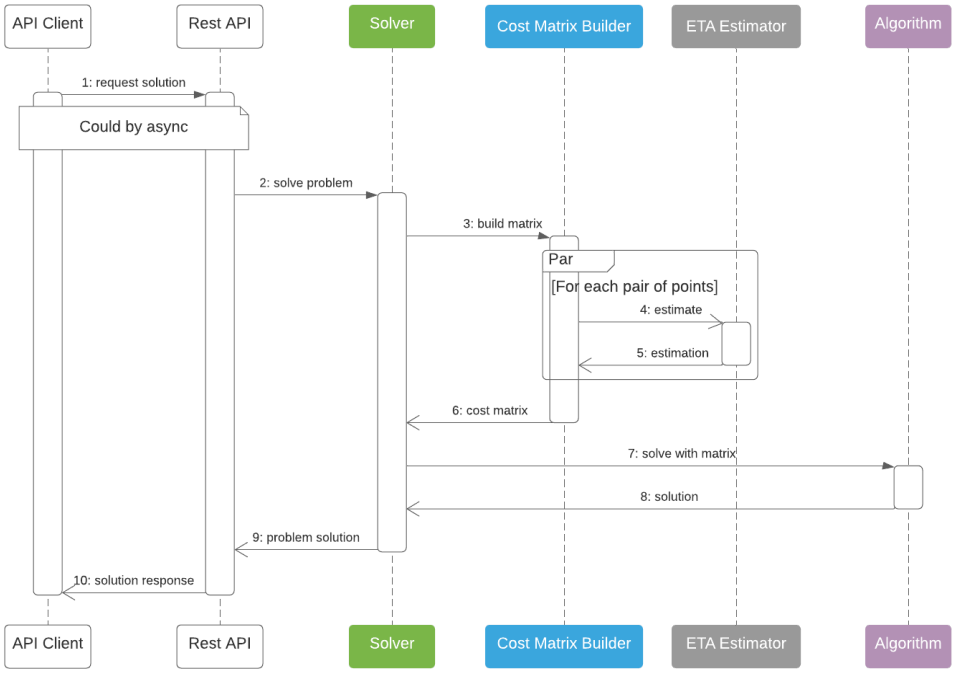

Figura 8. Diagrama del proceso de resolución

8. El proceso de desarrollo del proyecto

Esta sección da a conocer el proceso de desarrollo e implementación del producto.

Podemos decir que el proceso de desarrollo o implementación de la solución tecnológica que nos ha permitido cumplir los objetivos del trabajo ha sido un proceso iterativo e incremental. Este modelo buscaba reducir el riesgo y la incertidumbre derivados de la falta de conocimiento sobre el dominio del problema, nos permitiría obtener un crecimiento progresivo tanto de la funcionalidad como del conocimiento necesario para su desarrollo. Permitiéndonos contar cuanto antes con una pieza de software útil que se pudiese validar y asegurar cubrir unos objetivos parciales mucho antes de contar con un producto completo.

Todo comienza con un problema y unos requisitos claros y completamente especificados pero con un alto riesgo en la ejecución debido a principalmente a dos factores:

1. El problema se sale del área de conocimiento profesional.
2. Un tiempo de ejecución limitado y en un momento de máxima incertidumbre (COVID 19).

El desarrollo empieza con el proceso de descubrimiento del contexto, el estado del arte y el estudio de otras soluciones tecnológicas. Ese proceso comienza con la exploración inicial de soluciones basadas en programación lineal entera y posteriormente a la programación con restricciones, para acabar concluyendo que el problema podría ser abordado mediante la implementación de algoritmos heurísticos o metaheurísticos.

A continuación se relatan las diferentes fases del desarrollo con sus hitos o aprendizajes más importantes, el proceso se ha condensado en el gráfico [9]:

Figura 9. El proceso de desarrollo

Fase 0: Bootstrapping del servicio y búsqueda del heurístico

Esta fase se compone de dos hitos sin una relación de orden entre ellos. Uno sería el primer paso en la construcción de la solución tecnológica, el bootstrap de servicio, esto es, preparar el arranque del mismo, creación del repositorio de git, selección de las herramientas básicas y preparación del esqueleto del proyecto.

El otro hito tenía como objetivo la búsqueda de un algoritmo heurístico que estuviese alineado con las restricciones de nuestro problema, además de su comprensión y validación.

Fase 1: Implementación del algoritmo

Esta segunda etapa pretendía realizar una implementación rudimentaria del algoritmo con el lenguaje seleccionado, fue básicamente la traducción del algoritmo a Golang y su validación mediante el uso con un conjunto de test sencillo y revisable de forma manual. En este proceso de validación se fueron añadiendo casos de test más complejos a medida que la implementación iba cumpliendo las especificaciones.

Fase 2: Un caso real

Hasta este momento se habían testeado casos con pocos puntos, alejados decenas de kilómetros en donde tanto el coste de las distancias como su validación eran algo trivial al ojo humano.

El paso siguiente sería probar un caso real, esto es, un caso de ejemplo concreto en una ciudad utilizando estimaciones de tiempo y distancia más precisas. Por otro lado, se hacía urgente poder representar las rutas sobre un mapa para poder validar de forma visual la solución al problema.

Hasta este punto el servicio utilizaba una implementación del estimador de ETAs que realizaba cálculos de distancias utilizando la *fórmula del semiverseno* o *Haversine* *[38]* y sobre la cual aplicaba una estimación de velocidad media en ciudad de 30km/h para el cálculo de tiempos.

Para continuar con el plan lo primero era encontrar y utilizar un estimador de ETAs más preciso. De entre las posibles soluciones se decide por adoptar Google Maps Distance Matrix API [39] por los siguientes motivos:

1. Cuenta con un SDK en Golang [40] que permite realizar una integración con poco esfuerzo.
2. A pesar de ser de pago, y con alto coste, cuenta con una capa gratuita que permite realizar pruebas sin incurrir en grandes gastos.

3. Sigue un modelo SaaS [41]. No requiere ningún tipo de despliegue o configuración adicional, solo una tarjeta de crédito y cubrir un formulario para obtener la *API Key*.
4. Google Maps posee la información, los datos y los mecanismos necesarios para ser la solución más precisa del mercado.

Una vez contamos con la nueva implementación del estimador el paso siguiente fue la construcción de un componente capaz de renderizar sobre en mapa una solución de un problema. El artefacto utiliza *go-staticmaps* [42]. *go-staticmaps* es una biblioteca de go para renderizar imágenes de mapas estáticos usando mosaicos de OpenStreetMap.

La figura [10] representa el escenario de un problema de enrutado en Bogotá. El problema consta de 10 assets con una capacidad de 3 y 26 requests con una carga de 1 por solicitud. El resultado, son por lo tanto 10 rutas que dan servicio a las 26 solicitudes:

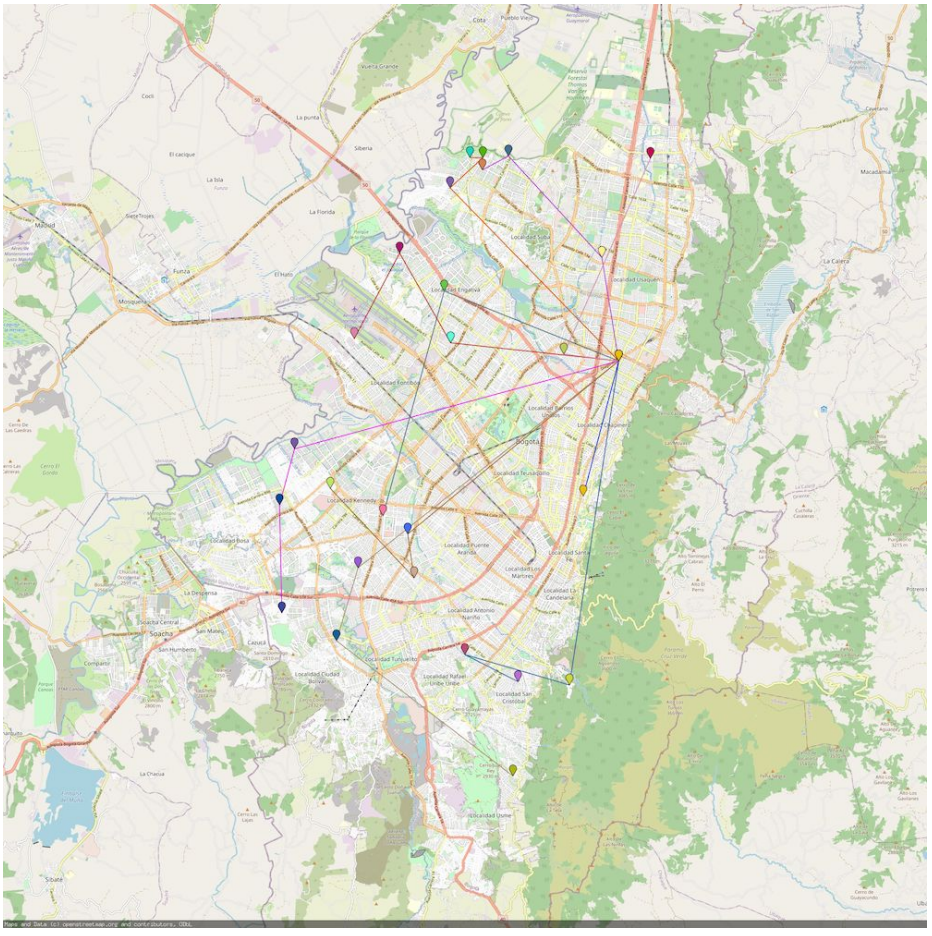

Figura 10. Problema sin requests sin asignar

En la figura [11] se muestra un escenario similar pero con una flota de 8 vehículos. En este caso no se pueden dar servicio a 3 de las *requests.*

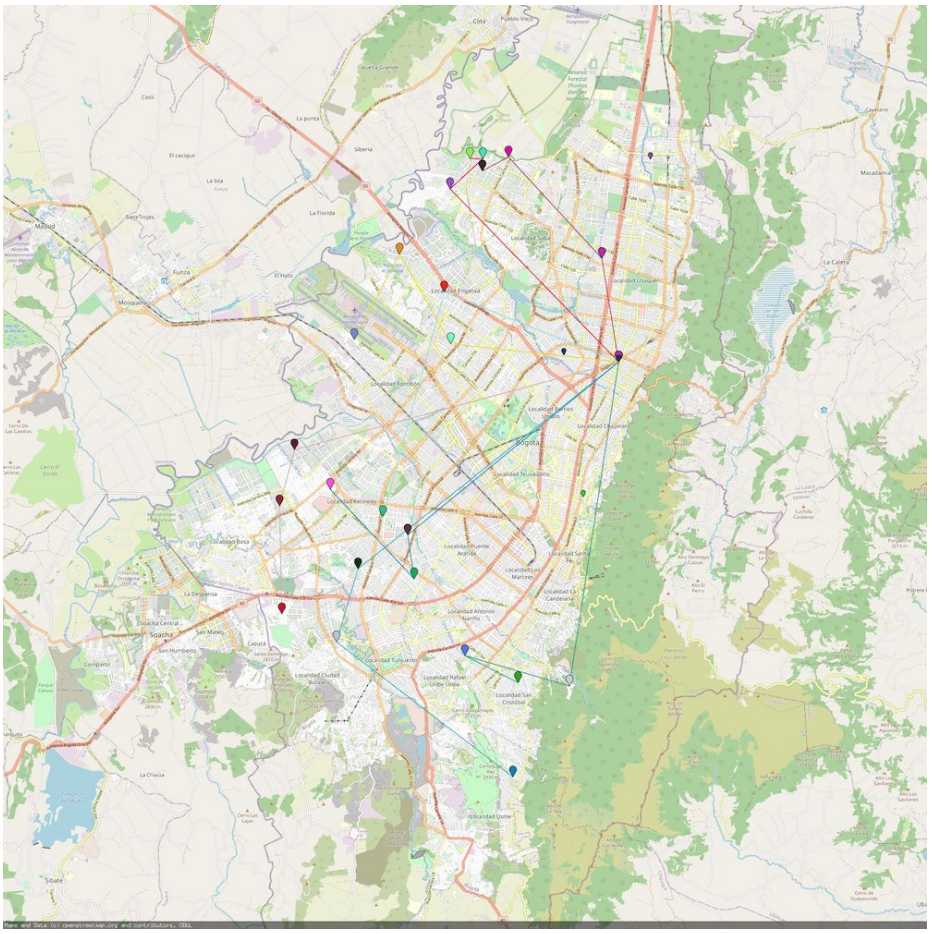

Figura 11. Problema con request sin asignar

El resultado de esta fase es por lo tanto Solver solvente y una representación gráfica de las soluciones.

Fase 3: Construyendo la API

Llegados a este punto es el momento de exponer el Solver mediante la API HTTP. Esta fase incluye la especificación de la API y la implementación del método síncrono de resolución de problemas.

El resultado de esta fase es por lo tanto la especificación de la API y el *endpoint* de resolución funcionando. Con esto ya es posible resolver problemas enunciados a través de objetos JSON.

Fase 4: Mejoras de rendimiento

Durante la fase anterior se experimentaron grandes diferencias en los tiempos de ejecución dependiendo del *Estimador ETA* utilizado. Como la consulta de las estimaciones tenían lugar en tiempo de ejecución del algoritmo era complicado afirmar rotundamente donde se encontraba el cuello de botella. Con la intención de ganar visibilidad del rendimiento del algoritmo (era además una idea contemplada inicialmente) se decide desacoplar el algoritmo del cálculo de los costes, para ello se decide construir la matriz de costes.

Una vez realizado el refactor y la extracción de dichas operaciones se confirma la sospecha. El ejemplo anteriormente citado utilizando *Haversine* [38] para calcular las distancias entre los puntos, el algoritmo resuelve el problema en 164.611341ms. Utilizando la API de Google se va a los 4 minutos principalmente por las latencias que introduce la API de Google (son muchas distancias a calcular).

El paso siguiente consiste en agrupar los cálculos para aprovechar el modo 1 a N de la API de Google. Esto implica mejoras de tiempos sustanciales (se reduce a la mitad de tiempo el cómputo) pero se considera que sigue siendo demasiado elevado.

Continuando con la búsqueda de mejoras de rendimiento se deciden encontrar puntos susceptibles de paralelizar. Paralelizar se refiere a convertir código secuencial en multihilo o vectorizado con objeto de utilizar múltiples procesadores simultáneamente en una máquina con multiprocesador de memoria compartida. En ese proceso se decide que de nuevo el punto que mayores mejoras puede proporcionar es la construcción de la matrix. Se paraleliza la construcción utilizando goroutinas consiguiendo una mejora de varios órdenes de magnitud. Se logra reducir el tiempo de cómputo de los 4 minutos iniciales a unos 14 segundos logrando resolver el problema completo en menos de 15 segundos.

Fase 5: Completar la API

Es la última etapa del desarrollo del servicio. Los cambios incluyen la agregación del endpoint asíncrono de solución de problemas así como la adición del endpoint que permite obtener problemas previamente enviados.

Para soportar el funcionamiento asíncrono se decide en primera instancia implementar un repositorio en memoria y lograr la asincronía mediante una cola implementada de nuevo utilizando goroutinas.

Queda como mejora futura la implementación de un repositorio sobre algún sistema de persistencia como MySQL o Redis y delegar la ejecución de los trabajos a algún sistema de colas desacoplado del servicio.

Con esta fase se completan los requisitos funcionales y operacionales del sistema. Cabe destacar que el artefacto tiene una cobertura de test del 84%, con test de integración, *end-to-end* y unitarios.

9. Conclusiones y trabajo futuro

En este capítulo se dan a conocer las conclusiones del proyecto y las conclusiones personales asociadas al desarrollo del mismo, así como las líneas de trabajo futuro.

9.1. Conclusiones

Todo proyecto, entendido éste como una secuencia de actividades que se desarrolla durante un tiempo para obtener un resultado único, requiere de una base, tanto histórica como actual, de conocimientos teóricos y prácticos, de esfuerzo, trabajo y motivación. Todo esto, además de generar un producto, proporciona una maravillosa oportunidad para incrementar las capacidades y habilidades de quien en él se involucre.

Desde antes de empezar tuve claro que mis objetivos personales en relación con el TFM iban más allá de la superación del mismo. He entendido el desarrollo del trabajo como la oportunidad de crear un producto útil cuya construcción fuese posible dentro de los límites temporales del TFM, así como de vehículo para explorar un nuevo dominio de conocimiento del cual no conocía más que el enunciado del problema.

El proceso me ha afianzado en la convicción de que un mismo problema puede afrontarse y solucionarse de diferentes maneras. Lo importante es conocer qué se quiere solucionar, conceptualizar tanto el problema de cómo la solución, extraer los conceptos y las relaciones entre ellos y, en este caso concreto, explorar la gran cantidad de artículos académicos que dan soporte a la solución más allá del aspecto tecnológico.

Considero, como se expone en la memoria, que el trabajo logra los objetivos planteados. A pesar de ello creo que, como se indica en el capítulo siguiente, hay todavía un camino por recorrer antes de contar con un producto listo para el uso en producción en un entorno de altas prestaciones.

En cuanto a la planificación, desde el primer momento se sabía que por las características del trabajo, el contexto sociohistórico, el conjunto de mis destrezas y el marco temporal limitado plantear la ejecución basada en un ciclo de vida en cascada era un error. Era necesario un planteamiento ágil que habilitase un desarrollo iterativo e incremental para posibilitar manejar y afrontar el riesgo del

proyecto permitiendo jugar, si fuese necesario, con el alcance del mismo. Asegurando que se pudiese contar cuanto antes con un producto que cumpliese los objetivos generales, pero sobre el que poder seguir iterando

Desde mi punto de vista, este enfoque centrado en los objetivos y no en el medio ha permitido que el resultado del trabajo aporte valor aún sin llegar a explorar técnicas basadas en el uso de GPU o la implementación de algoritmos más complejos. Ambos puntos parecían imprescindibles en el esbozo inicial de la solución, hecho incluso antes de comenzar el proceso de descubrimiento del problema que tiene lugar durante el desarrollo del trabajo. Basándonos en eso podemos decir que ha habido cambios en cuanto al plan previo, pero estos tiene que ver más con detalles de implementación que con modificaciones a nivel funcional o de los objetivos del trabajo.

Finalmente, y teniendo en cuenta que el resultado es mejorable y que, como suele ser habitual, una vez acabado lo ideal sería poder volver a empezar de cero con los aprendizajes del proceso, me considero satisfecho y contento con el resultado final.

9.2. Sostenibilidad del proyecto

Como se ha comentado la finalidad de este proyecto no termina con la redacción de la memoria del trabajo. Este es solo un paso más para poder cumplir el objetivo final del mismo, agrupar viajeros en un entorno real.

A pesar de que el sistema es funcional hay varias líneas de trabajo que se deben continuar. La primera está orientada al despliegue en producción del servicio y requiere:

- Mejorar la observabilidad, es decir, añadir métricas que permitan inferir el estado del sistema, su rendimiento y su operación.
- Incorporar una solución de almacenamiento que permita a la solución ser multi-instancia y por lo tanto escalable horizontalmente. Actualmente el servicio almacena los problemas y sus soluciones en memoria, esto implica que los objetos no pueden ser compartidos y por lo tanto no existen fuera del runtime de la instancia en ejecución. Por otro lado, y por la misma causa, los objetos se eliminarán al reiniciar el servicio. Utilizar una solución de persistencia desacoplada basada en MySQL o Redis solucionaría el problema.

A mayores sería recomendable mover los trabajos de solución a un sistema de colas que permita tener un mejor control sobre el número de procesos. La idea sería implementar una cola de problemas y una serie de *workers* que en paralelo los fueran solucionando teniendo en cuenta las restricciones de capacidad, prioridades, etc...

Además, se debería explorar posibles mejoras en el *bin-packing* del algoritmo implementado. Esto es, el algoritmo no busca la solución más eficiente en cuanto a la capacidad del asset. Si una ruta es factible se asigna al resultado pudiendo quedar asignaciones poco eficaces desde el punto de vista de la carga del vehículo. Sobre todo cuando en la misma flota existen assets con capacidades muy diferentes (x2, x3). Este problema es, de nuevo, un problema de optimización combinatoria que debe permitir maximizar el valor de la carga respetando el resto de restricciones que impone el problema de VRP.

Otra línea de trabajo estaría enfocada en la búsqueda de alternativas a Google Maps como *backend* para el cálculo de estimaciones de tiempo/distancia. Como se ha visto el estimador es una pieza fundamental del Solver dado que los valores que este servicio ofrece son los que constituyen la matriz de costes. Una solución propia es eficiente en cuanto a tiempo y coste pero no proporciona los datos más exactos. Google Maps API es preciso pero muy caro. Una solución basada en Open Source Routing Machine [43] construida sobre los datos de OpenStreetMap [44] podría ser más que aceptable.

Para acabar, y no menos importante, otras actuaciones serían el onboarding final del servicio a producción y la integración con los componentes downstream y upstream existentes. Así como la realización de test de rendimiento y comparación de los resultados de la optimización de nuestra propuesta frente a otras soluciones comerciales similares.

10. Glosario

API: una API (Application Programming Interface en inglés) es un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas de la misma manera en que la interfaz de usuario facilita la interacción humano-software.

API HTTP: es una API accesible de forma remota usando el protocolo HTTP.

Instancia: una instancia de un programa es una copia de una versión ejecutable del programa que ha sido escrito en la memoria del computador.

JSON: es un formato ligero de intercambio de datos simple de leer y escribir para humanos, y simple de interpretar y generar para las máquinas. Está basado en un subconjunto del Lenguaje de Programación JavaScript. JSON es un formato de texto que es completamente independiente del lenguaje.

Pick-up: hace referencia al acto de recoger a algo o alguien. En nuestro caso es el acto de recoger a un viajero.

Delivery: delivery (reparto o entrega) es una actividad parte de la función logística que tiene por finalidad colocar bienes, servicios en el lugar de consumo o uso (el cliente final).

Drop-off: hace referencia al acto de entregar algo o alguien. En nuestro caso es la acción que ocurre cuando un viajero llega a su destino y baja del vehículo.

Heurística: la heurística es una técnica diseñada para resolver un problema más rápidamente cuando los métodos clásicos son demasiado lentos, o para encontrar una solución aproximada cuando los métodos clásicos no logran encontrar una solución exacta.

Meta-heurística: una metaheurística es un método heurístico para resolver un tipo de problema computacional general, usando los parámetros dados por el usuario sobre unos procedimientos heurísticos genéricos y abstractos de una manera que se espera eficiente.

Ride-hailing: puede entenderse como el servicio de movilidad que permite la reserva de viajes, con un coste estimado a priori, pago por uso y con un conductor profesional, a través de una aplicación móvil o web.

Ride-pooling: es el servicio de movilidad en el cual varios pasajeros comparten un conductor profesional. Por ejemplo, la persona X ha contratado un servicio de ride-hailing para ir de A a B. Fortuitamente, la persona Y hace una solicitud al mismo proveedor de servicios para la misma ruta o una similar. Un algoritmo combina ambas rutas y la persona Y se sube en el vehículo de ride-hailing junto con la persona X o viceversa.

SaaS: Software como un Servicio, abreviado SaaS, es un modelo de distribución de software donde el soporte lógico y los datos que maneja se alojan en servidores de una compañía de tecnologías de información y comunicación, a los que se accede vía Internet desde un cliente.

Servicio: es un tipo de proceso ejecutado en segundo plano. Es un software que realiza tareas automatizadas, responde a eventos de hardware o escucha solicitudes de datos de otro software.

Stakeholder: es cualquier agente (humano u otro sistema) que tiene algún interés en el sistema software.

VRP: *Vehicle Routing Problem*, es uno de los problemas de optimización combinatoria más importantes y ampliamente estudiado, con muchas aplicaciones del mundo real, en la logística de distribución y del transporte. El objetivo de este problema trata de minimizar la distancia total recorrida por un conjunto de vehículos para satisfacer la demanda de un determinado conjunto de clientes.

Worker: un *worker* es un proceso en segundo plano que se encarga de ejecutar una tarea de larga ocupación sin necesidad de interrumpir la respuesta del usuario.

YAML: es un formato de serialización de datos legible por humanos.

11. Referencias

- [1] Kitchenham B, Brereton OP, Budgen D, Turner M, Bailey J, Linkman S (2009) Systematic literature reviews in software engineering — a systematic literature review. Inf Softw Technol 51(1):7–15
- [2] Erdogan, Günes. (2017). An Open Source Spreadsheet Solver for Vehicle Routing Problems. Computers & Operations Research. 84. 10.1016/j.cor.2017.02.022.
- [3] Google OR-Tools. (Octubre 2020). https://developers.google.com/optimization
- [4] jsprit | java toolkit for rich VRPs and TSPs. (Octubre 2020). https://jsprit.github.io/
- [5] OptaPlanner - Constraint satisfaction solver (Java™, Open Source). (Octubre 2020). https://www.optaplanner.org/
- [6] Open-source framework for modeling Vehicle Routing Problems. (Octubre 2020). https://github.com/mck-/Open-VRP
- [7] VROOM - Vehicle Routing Open-source Optimization Machine. (Octubre 2020). http://vroom-project.org/
- [8] A Vehicle Routing Problem solver. (Octubre 2020). https://github.com/reinterpretcat/vrp
- [9] Delivery Route Optimization Software & App | Routific. (Octubre 2020). https://routific.com/
- [10] SmartMonkey.io – Plan your routes efficiently. (Octubre 2020). https://smartmonkey.io/
- [11] HERE Technologies | The world's #1 location platform. (Octubre 2020). https://www.here.com/
- [12] Tour Planning API from HERE: Multi-vehicle route optimization | HERE. (Octubre 2020). https://developer.here.com/products/tour-planning
- [13] Advanced planning software. (Octubre 2020). https://www.solvice.io
- [14] OnRoute API. (Octubre 2020). https://www.solvice.io/onroute/api
- [15] Corentin Mulders, Carpooling, a vehicle routing approach. (2012) https://www.info.ucl.ac.be/~pschaus/assets/master_thesis/2013_corentin_mulders.pdf

- [16] M. Onofrei. VRP solution using GPGPUs. (2017). https://scripties.uba.uva.nl/search?id=634852
- [17] B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem: A taxo-nomic review. Computers & Industrial Engineering, 57(4), Pages 1472–1483, (2009). ISSN 0360-8352
- [18] Jairo R. Montoya-Torres, Julián López Franco, Santiago Nieto Isaza, Heriberto Felizzola Jiménez, Nilson Herazo-Padilla. A literature review on the vehicle routing problem with multiple depots, Computers & Industrial Engineering, Volume 79, Pages 115-129, (2015), ISSN 0360-8352.
- [19] Reza Moghdani, Khodakaram Salimifard, Emrah Demir, Abdelkader Benyettou. The green vehicle routing problem: A systematic literature review. Journal of Cleaner Production, Volume 279, (2020), ISSN 0959-6526
- [20] M. R. Garey and D. S. Johnson.Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, (1979).
- [21] P. Toth and D. Vigo, editors.The vehicle routing problem. Society for Industrial And Applied Mathematics, Philadelphia, PA, USA, (2001).
- [22] David Pisinger, Stefan Ropke, A general heuristic for vehicle routing problems, Computers & Operations Research, Volume 34, Issue 8, Pages 2403-2435, (2007), ISSN 0305-0548
- [23] Baldacci, R., Bartolini, E., Mingozzi, A. et al. An exact solution framework for a broad class of vehicle routing problems. Comput Manag Sci 7, 229–268, (2010).
- [24] Roberto Baldacci, Aristide Mingozzi, Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. European Journal of Operational Research. Volume 218, Issue 1,Pages 1-6, (2012), ISSN 0377-2217
- [25] Gilbert Laporte, Michel Gendreau, Jean-Yves Potvin, Frédéric Semet. Classical and modern heuristics for the vehicle routing problem. International Transactions in Operational Research, Volume 7, Issues 4–5, Pages 285-300, (2000), ISSN 0969-6016
- [26] Prins C., Bouchenoua S. A Memetic Algorithm Solving the VRP, the CARP and General Routing Problems with Nodes, Edges and Arcs. In: Hart W.E., Smith J.E., Krasnogor N. (eds) Recent Advances in Memetic Algorithms. Studies in Fuzziness and Soft Computing, vol 166. Springer, Berlin, Heidelberg. (2005)

- [27] Toro O. Eliana M., Escobar Z. Antonio H., Granada E. Mauricio. Literature Review of the Vehicle Routing Problem in the Green transportation Context. (2016)
- [28] Eldrandaly, Khalid & Ahmed, AbdelHadi. Routing Problems: A Survey. (2008)
- [29] Bansal, Sandhya. Issues in Solving Vehicle Routing Problem with Time Window and its Variants using Meta heuristics - A Survey. (2013)
- [30] Hosny, Manar. Vehicle Routing with Pickup and Delivery: Heuristic and Meta-heuristic Solution Algorithms. (2012)
- [31] Carlo S. Sartori, Luciana S. Buriol. A study on the pickup and delivery problem with time windows: Matheuristics and new instances. Computers & Operations Research, Volume 124, ISSN 0305-0548, (2020)
- [32] G. B. Dantzig and D. R. Fulkerson. Minimizing the number of tankers to meet a fixed schedule.Naval Research Logistics Quarterly, 1:217–222, (1954).
- [33] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, (1979).
- [34] P. Toth and D. Vigo, editors. The vehicle routing problem. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, (2001).
- [35] Manar I. Hosny, Christine L. Mumford. Constructing initial solutions for the multiple vehicle pickup and delivery problem with time windows, Journal of King Saud University - Computer and Information Sciences, Pages 59-69, ISSN 1319-1578, (2012) https://doi.org/10.1016/j.jksuci.2011.10.006
- [36] Open API Specification (Noviembre 2020). http://spec.openapis.org/oas/v3.0.3
- [37] Go by Example: Goroutines (Noviembre 2020). https://gobyexample.com/goroutines
- [38] Fórmula del Semiverseno (Noviembre 2020). https://es.wikipedia.org/wiki/F%C3%B3rmula_del_semiverseno
- [39] Google Maps Distance Matrix (Noviembre 2020). https://developers.google.com/maps/documentation/distance-matrix/overview
- [40] Google Maps SDK (Noviembre 2020). https://github.com/googlemaps/google-maps-services-go

- [41] Software como Servicio (Software as a Service / SaaS) (Noviembre 2020). https://es.wikipedia.org/wiki/Software_como_servicio
- [42] Go Static Maps (Noviembre 2020). https://github.com/flopp/go-staticmaps
- [43] Open Source Routing Machine (Diciembre 2020). http://project-osrm.org/
- [44] OpenStreetMap (Diciembre 2020). https://www.openstreetmap.org/about

12. Anexos

12.1 Especificación de la API mediante OpenAPI Specification

```
openapi: 3.0.0
info:
 version: v1
 title: Roteiro API
servers:
 - url: 'http://localhost:8080/api/v1/'
paths:
 '/problem':
  post:
     summary: "Solve a problem with the given description"
     operationId: problemPost
       description: "The endpoint can solve the vehicle routing
problem stated in the request body synchronously."
     tags:
       - Solver
     requestBody:
       description: The problem
       required: true
       content:
         application/json:
           schema:
             $ref: "#/components/schemas/ProblemRequest"
     responses:
       200:
         description: "Success"
         content:
           application/json:
             schema:
               $ref: "#/components/schemas/SolutionResponse"
       400:
         description: "Error"
         content:
           application/json:
             schema:
               $ref: "#/components/schemas/ErrorResponse"
 '/problem-long':
   post:
        summary: "Queue a long running problem with the given
description"
     operationId: problemPostAsync
        description: "It will trigger a long-running background
task. The GET solution endpoint could be used to ping to fetch
the results.
This is necessary for larger requests that take longer than 30
seconds.
```

```
When you POST to the above URL, it will return immediately with a
202 HTTP code and the problem_id"
      tags:
        - Solver
      requestBody:
        description: The problem
        required: true
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/ProblemRequest"
      responses:
        202:
          description: "Problem queued"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/ProblemId"
        400:
          description: "Error"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/ErrorResponse"
  '/solution/{problem_id}':
    get:
      summary: "Get the solution for the given problem_id"
      operationId: solutionGet
          description: "It allows to recover the solution of a
previously solved problem or a previously queued problem.
When the problem has not yet been resolved, the status code is
102 (Processing). Upon completion, the status code is 200.
The response that you would normally get directly from the
synchronous endpoint is now in the output."
      tags:
        - Solver
      parameters:
        - name: problem_id
          in: path
          description: ID of related problem
          required: true
          schema:
            type: string
            format: uuid
      responses:
        200:
          description: "Success"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/SolutionResponse"
```

```
        102:
          description: "Processing. The problem is not solved yet"
        404:
          description: "Not found"
        400:
          description: "Error"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/ErrorResponse"
components:
 schemas:
   ErrorResponse:
     type: object
     properties:
       error:
         type: string
         description: "The error message"
   ProblemRequest:
     type: object
     properties:
       assets:
         type: array
         items:
           $ref: "#/components/schemas/Asset"
       requests:
         type: array
         items:
           $ref: '#/components/schemas/Request'
       constraints:
         type: object
         properties:
           max_journey_time_factor:
             type: number
             format: float
   SolutionResponse:
     type: object
     properties:
       problem_id:
         type: string
         format: uuid
       metrics:
         type: object
         properties:
           num_assets:
             type: integer
             format: int32
           num_requests:
             type: integer
             format: int32
           num_unassigned:
```

```
            type: integer
            format: int32
          duration:
            type: integer
            format: int32
              description: "Duration of the complete solution in
nanoseconds"
          distance:
            type: integer
            format: int32
            description: "Distance in meters"
          solved_time:
            type: integer
            format: int32
                    description: "Time to solve the problem in
nanoseconds"
      routes:
        type: array
        items:
          type: object
          properties:
            asset:
              $ref: '#/components/schemas/Asset'
            requests:
              type: array
              items:
                $ref: '#/components/schemas/Request'
            waypoints:
              type: array
              items:
                $ref: '#/components/schemas/Waypoint'
            metrics:
              type: object
              properties:
                requests:
                  type: integer
                  format: int32
                duration:
                  type: integer
                  format: int32
                distance:
                  type: integer
                  format: int32
      unassigned:
        type: array
        items:
          $ref: '#/components/schemas/Request'

  Asset:
    type: object
    properties:
```

```
    asset_id:
      type: string
    location:
      $ref: '#/components/schemas/Point'
    capacity:
      type: integer
      format: int32
Request:
  type: object
  properties:
    requester_id:
      type: string
    load:
      type: integer
      format: int32
    pick_up:
      $ref: '#/components/schemas/Point'
    drop_off:
      $ref: '#/components/schemas/Point'
Point:
  type: object
  properties:
    lat:
      type: number
      format: float
    lon:
      type: number
      format: float
Waypoint:
  type: object
  properties:
    location:
      $ref: '#/components/schemas/Point'
    load:
      type: integer
      format: int32
    activities:
      type: array
      items:
        type: object
        properties:
          activity_type:
            type: string
          ref:
            type: string
ProblemId:
  type: object
  properties:
    problem_id:
      type: string
      format: uuid
```

12.2 Ejemplo entrada y salida de la API

A continuación se expone un ejemplo real de entrada y salida del sistema. El caso utiliza Google Maps API como *backend* de Estimación ETA arrojando los siguientes tiempos de ejecución:

- Construcción de la matriz de costes: 1.072761515s
- Solución del problema: 1.279268ms

El ejemplo se ha ejecutado en un MacBook Pro con un procesador 2,7 GHz Intel Core i7 de 4 núcleos y 16 GB 2133 MHz LPDDR3 de memoria RAM.

12.2.1 Especificación del problema

Ejemplo de JSON con un problema con 3 *assets* y 8 *requests*


```
{
 "assets": [
  {
     "asset_id": "a1",
     "location": {
       "lat": 4.68295,
       "lon": -74.04965
     },
     "capacity": 3
   },
   {
     "asset_id": "a2",
     "location": {
       "lat": 4.68295,
       "lon": -74.04965
     },
     "capacity": 3
   },
   {
     "asset_id": "a3",
     "location": {
       "lat": 4.68295,
       "lon": -74.04965
     },
     "capacity": 3
   }
 ],
 "requests": [
   {
     "requester_id": "1",
     "pick_up": {
```

```
      "lat": 4.68295,
      "lon": -74.04965
    },
    "drop_off": {
      "lat": 4.52924,
      "lon": -74.08894
    },
    "load": 1
  },
  {
    "requester_id": "2",
    "pick_up": {
      "lat": 4.68295,
      "lon": -74.04965
    },
    "drop_off": {
      "lat": 4.57929,
      "lon": -74.1545
    },
    "load": 1
  },
  {
    "requester_id": "3",
    "pick_up": {
      "lat": 4.68295,
      "lon": -74.04965
    },
    "drop_off": {
      "lat": 4.60634,
      "lon": -74.1465
    },
    "load": 1
  },
  {
    "requester_id": "4",
    "pick_up": {
      "lat": 4.68295,
      "lon": -74.04965
    },
    "drop_off": {
      "lat": 4.58963,
      "lon": -74.1748
    },
    "load": 1
  },
  {
    "requester_id": "5",
    "pick_up": {
      "lat": 4.68295,
      "lon": -74.04965
    },
```

```
      "drop_off": {
        "lat": 4.62963,
        "lon": -74.17566
      },
      "load": 2
    },
    {
      "requester_id": "6",
      "pick_up": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "drop_off": {
        "lat": 4.65039,
        "lon": -74.17011
      },
      "load": 1
    },
    {
      "requester_id": "7",
      "pick_up": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "drop_off": {
        "lat": 4.56418,
        "lon": -74.08705
      },
      "load": 1
    },
    {
      "requester_id": "8",
      "pick_up": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "drop_off": {
        "lat": 4.57419,
        "lon": -74.10678
      },
      "load": 1
    }
  ],
  "constraints": {
    "max_journey_time_factor": 1.5
  }
}
```

12.2.2 Especificación de la solución

A continuación se presenta la salida del Solver en formato JSON:


```
{
 "problem_id": "8104c80a-f752-4972-8c84-619af2a0de03",
 "metrics": {
   "num_assets": 3,
   "num_requests": 8,
   "num_unassigned": 0,
   "duration": 10996000000000,
   "distance": 75627,
   "solved_time": 1269033
 },
 "routes": [
   {
     "asset": {
       "asset_id": "a1",
       "location": {
         "lat": 4.68295,
         "lon": -74.04965
       },
       "capacity": 3
     },
     "requests": [
       {
         "requester_id": "1",
         "pick_up": {
           "lat": 4.68295,
           "lon": -74.04965
         },
         "drop_off": {
           "lat": 4.52924,
           "lon": -74.08894
         },
         "load": 1
       },
       {
         "requester_id": "7",
         "pick_up": {
           "lat": 4.68295,
           "lon": -74.04965
         },
         "drop_off": {
           "lat": 4.56418,
           "lon": -74.08705
         },
         "load": 1
       },
       {
         "requester_id": "8",
```

```
      "pick_up": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "drop_off": {
        "lat": 4.57419,
        "lon": -74.10678
      },
      "load": 1
    }
  ],
  "waypoints": [
    {
      "location": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "load": 3,
      "activities": [
        {
          "activity_type": "Start",
          "ref": "a1"
        },
        {
          "activity_type": "PickUp",
          "ref": "1"
        },
        {
          "activity_type": "PickUp",
          "ref": "7"
        },
        {
          "activity_type": "PickUp",
          "ref": "8"
        }
      ]
    },
    {
      "location": {
        "lat": 4.57419,
        "lon": -74.10678
      },
      "load": 2,
      "activities": [
        {
          "activity_type": "DropOff",
          "ref": "8"
        }
      ]
    },
    {
```

```
        "location": {
          "lat": 4.56418,
          "lon": -74.08705
        },
        "load": 1,
        "activities": [
          {
            "activity_type": "DropOff",
            "ref": "7"
          }
        ]
      },
      {
        "location": {
          "lat": 4.52924,
          "lon": -74.08894
        },
        "load": 0,
        "activities": [
          {
            "activity_type": "DropOff",
            "ref": "1"
          }
        ]
      }
    ],
    "metrics": {
      "requests": 3,
      "duration": 3801000000000,
      "distance": 24286
    }
  },
  {
    "asset": {
      "asset_id": "a2",
      "location": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "capacity": 3
    },
    "requests": [
      {
        "requester_id": "2",
        "pick_up": {
          "lat": 4.68295,
          "lon": -74.04965
        },
        "drop_off": {
          "lat": 4.57929,
          "lon": -74.1545
```

```
      },
      "load": 1
    },
    {
      "requester_id": "4",
      "pick_up": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "drop_off": {
        "lat": 4.58963,
        "lon": -74.1748
      },
      "load": 1
    },
    {
      "requester_id": "3",
      "pick_up": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "drop_off": {
        "lat": 4.60634,
        "lon": -74.1465
      },
      "load": 1
    }
  ],
  "waypoints": [
    {
      "location": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "load": 3,
      "activities": [
        {
          "activity_type": "Start",
          "ref": "a2"
        },
        {
          "activity_type": "PickUp",
          "ref": "2"
        },
        {
          "activity_type": "PickUp",
          "ref": "4"
        },
        {
          "activity_type": "PickUp",
          "ref": "3"
```

```
        }
      ]
    },
    {
      "location": {
        "lat": 4.60634,
        "lon": -74.1465
      },
      "load": 2,
      "activities": [
        {
          "activity_type": "DropOff",
          "ref": "3"
        }
      ]
    },
    {
      "location": {
        "lat": 4.57929,
        "lon": -74.1545
      },
      "load": 1,
      "activities": [
        {
          "activity_type": "DropOff",
          "ref": "2"
        }
      ]
    },
    {
      "location": {
        "lat": 4.58963,
        "lon": -74.1748
      },
      "load": 0,
      "activities": [
        {
          "activity_type": "DropOff",
          "ref": "4"
        }
      ]
    }
  ],
  "metrics": {
    "requests": 3,
    "duration": 3854000000000,
    "distance": 27555
  }
},
{
  "asset": {
```

```
    "asset_id": "a3",
    "location": {
      "lat": 4.68295,
      "lon": -74.04965
    },
    "capacity": 3
  },
  "requests": [
    {
      "requester_id": "5",
      "pick_up": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "drop_off": {
        "lat": 4.62963,
        "lon": -74.17566
      },
      "load": 2
    },
    {
      "requester_id": "6",
      "pick_up": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "drop_off": {
        "lat": 4.65039,
        "lon": -74.17011
      },
      "load": 1
    }
  ],
  "waypoints": [
    {
      "location": {
        "lat": 4.68295,
        "lon": -74.04965
      },
      "load": 3,
      "activities": [
        {
          "activity_type": "Start",
          "ref": "a3"
        },
        {
          "activity_type": "PickUp",
          "ref": "6"
        },
        {
          "activity_type": "PickUp",
```

```
            "ref": "5"
          }
        ]
      },
      {
        "location": {
          "lat": 4.65039,
          "lon": -74.17011
        },
        "load": 2,
        "activities": [
          {
            "activity_type": "DropOff",
            "ref": "6"
          }
        ]
      },
      {
        "location": {
          "lat": 4.62963,
          "lon": -74.17566
        },
        "load": 0,
        "activities": [
          {
            "activity_type": "DropOff",
            "ref": "5"
          }
        ]
      }
    ],
    "metrics": {
      "requests": 2,
      "duration": 3341000000000,
      "distance": 23786
    }
  }
 ],
 "unassigned": []
}
```

12.2.3 Representación de la solución

Esta respuesta se representa en el espacio de geográfico tal y como se indica en la figura [12]:

Figura 12. Representación de la solución de ejemplo