Big Data – Análisis de tráfico y optimización de rutas con machine learning

Gabriel Adrian Margineanu

Master en Ingeniería de Telecomunicación
Smart Cities

Tutor/a de TF
Rubén Molina Casasnovas

Profesor/a responsable de la asignatura
Carlos Monzo Sánchez

Fecha Entrega
08/01/2024

Universitat Oberta de Catalunya

Ficha del Trabajo Final

Título del trabajo:	Big Data – Análisis de tráfico y optimización de rutas con machine learning
Nombre del autor/a:	Gabriel Adrian Margineanu
Nombre del Tutor/a de TF:	Rubén Molina Casasnovas
Nombre del/de la PRA:	Carlos Monzo Sánchez
Fecha de entrega:	01/2024
Titulación o programa:	Master en Ingeniería de Telecomunicación
Área del Trabajo Final:	Smart Cities
Idioma del trabajo:	Castellano
Palabras clave	*Big data*, *machine learning*, optimización, computación distribuida

Resumen del Trabajo

En este proyecto se van a analizar un conjunto de datos de acceso libre utilizando sistemas y metodologías *Big Data* sobre el comportamiento del tráfico vehicular en la ciudad de Madrid desde julio de 2022 hasta junio de 2023.

El estudio analizará los datos de tráfico para crear un mapa de calor con las zonas con más tráfico durante cada hora, según el día. Con este análisis se crearán modelos de *machine learning* o inteligencia artificial para optimizar las rutas. De esta forma podremos calcular la ruta optima prediciendo el tráfico que ocurrirá a lo largo del trayecto.

Se van a entrenar varios algoritmos, optimizados para acortar el trayecto para obtener la ruta más ecológica o para encontrar la ruta más rápida. Estos modelos podrían ser utilizados para reducir costes de operación en empresas repartidoras y de transporte, reduciendo el tiempo del trayecto y la polución.

En el proyecto sólo se utilizan datos de tráfico para la ciudad de Madrid durante un año para no aumentar la complejidad de hardware requerido durante el procesado. Solamente sirve como un modelo para demonstrar la viabilidad y el estudio de los sistemas requeridos.

Para el análisis se empleará un entorno virtualizado con contenedores y PySpark para cargar y analizar los datos. Con librerías de Python se crearán los distintos modelos y algoritmos.

Abstract

This project will analyze an open access dataset using Big Data systems and methodologies about the behavior of vehicular traffic in the city of Madrid from July 2022 to June 2023.

The study will analyze the traffic data to create a heatmap with the areas with the most traffic during each hour, depending on the day. This analysis will be used to create machine learning or artificial intelligence models to optimize routes. In this way we will be able to predict the traffic that will occur along the route.

Several algorithms will be trained, optimized to shorten the route to obtain the most environmentally friendly route or to find the fastest route. These models could be used to reduce operating costs for delivery and transport companies by reducing journey times and the amount of pollution.

The project only uses traffic data for the city of Madrid for one year in order to not increase the hardware complexity required during processing. It only serves as a model to demonstrate the feasibility and study of the required systems.

For the analysis, a virtualized environment with containers will be used with PySpark to load and analyze the data. The different machine learning models and algorithms will be created with Python libraries.

Índex

Lista de Figuras

Lista de Tablas

Lista de Gráficas

1. Introducción

En este primer capítulo del proyecto se explicará el contexto y la motivación del trabajo. Se fijarán los objetivos propuestos a alcanzar, el impacto ético social y la planificación con las distintas etapas del proyecto.

1.1. Contexto y justificación del trabajo

En España, el 69% de la población reside en áreas urbanas con más de 50000 habitantes. Esto se debe a la concentración del empleo y a las oportunidades económicas, el acceso a los servicios y la comodidad que estos ofrecen. El desarrollo urbano ha creado redes de transporte, redes de suministros básicos e infraestructura que tiene que soportar una alta densidad de población en general. La rápida expansión de los entornos urbanos requiere una optimización de la infraestructura para aprovechar al máximo los recursos limitados disponibles y bajar los costes de mantenimiento.

Debido a la urbanización y al crecimiento económico, la cantidad de vehículos en uso ha ido incrementando, actualmente hay más de 35,6 millones de vehículos en circulación, según la Dirección General de Tráfico, en España. Esta cantidad de vehículos supone un aumento en la congestión y la polución en las áreas con más habitantes y tráfico. Para mejorar la eficiencia de la infraestructura de transporte, se requiere analizar el tráfico para detectar anomalías y zonas que pueden causar problemas para optimizar la circulación de los vehículos.

La evolución de la tecnología y en concreto de la capacidad de almacenamiento permite recopilar y guardar grandes cantidades de información. Utilizando sistemas de procesamiento *Big Data* es posible analizar grandes bases de datos en tiempo real, permitiendo extraer información detallada y precisa. Debido a las innovaciones de la tecnología, al aumento exponencial en capacidad de cómputo y la asequibilidad de los componentes, la inteligencia artificial se está convirtiendo en un pilar fundamental de la sociedad. Permite optimizar flujos y predecir demanda en tiempo real, de ahí el interés en saber cómo funciona *Big Data* y cómo se implementan estos sistemas.

La implementación de modelos de *machine learning* e inteligencia artificial está acelerándose, siendo Estados Unidos el líder de la investigación y adopción de las mejoras que aporta. La Unión Europea está invirtiendo en varios proyectos, como Europa Digital o *NextGenerationEU*, para promover el desarrollo en los países miembros.

1.2. Objetivos del trabajo

El objetivo del proyecto es estudiar el funcionamiento de los sistemas *Big Data*, analizar las mejores herramientas utilizadas en el procesamiento de datos y realizar un estudio sobre datos reales utilizando las soluciones adecuadas.

En este proyecto nos centraremos en la optimización del tráfico vehicular en la ciudad de Madrid, utilizando datos recogidos durante el último año. Al optimizar las redes de transporte, conseguiremos reducir la polución, ahorrar gastos y reducir la duración de los viajes.

Para optimizar las rutas *GPS* teniendo en cuenta el tráfico, primero crearemos un mapa de calor utilizando datos históricos resaltando las zonas dónde más tráfico hay, según el día y la hora. Tras crear el mapa de calor, utilizaremos *machine learning* e inteligencia artificial para crear varios algoritmos que busquen la ruta más corta o la ruta más rápida para evitar tráfico.

Los datos utilizados en el análisis son los datos publicados por el ayuntamiento de Madrid, de acceso libre a través del portal https://datos.madrid.es/. Usaremos los datos desde junio de 2022 hasta julio de 2023. Se utilizarán los datos de una sola ciudad para reducir la complejidad del entorno necesario para procesar todos los datos. Lo importante es desarrollar un proceso automatizado que analice todos los datos, una vez desarrollado este proceso, se podrá ampliar para utilizar más información o analizar datos en tiempo real, en un entorno más realista, ampliando la utilidad del proyecto.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

Como impactos positivos, el TFM puede ayudar a las personas y a las entidades que usan la aplicación para ahorrar tiempo y recursos, optimizando las rutas elegidas evitando la congestión del tráfico. Al acortar las rutas y optimizando la longitud recorrida, disminuye la polución causada. En este caso, el ahorro energético y mejora de la sostenibilidad contribuimos para lograr los objetivos de desarrollo sostenible marcados para las Smart Cities [1].

Durante la pandemia, una de las medidas drásticas que se planteaban era el seguimiento de las personas a través de una aplicación móvil. Dadas las implicaciones legales y éticas, el uso de estas aplicaciones era opcional, pero en otros países como Singapur o Corea del sur el uso era obligatorio para las personas infectadas. [2]

Otra aplicación en la que ayudaría la gestión de rutas es para un turismo inteligente. Esto ayuda a reducir el impacto ambiental con una distribución más controlada para las ciudades [3].

Para el proyecto, se han utilizado datos tomados con sensores, en vías públicas y no contienen información personal. En otras aplicaciones, se utilizan datos de varias fuentes, no sólo sensores, como por ejemplo datos *GPS* de taxistas, autobuses públicos o de los propios usuarios e información de las redes sociales.

1.4. Enfoque y método seguido

Para decidir qué herramientas se utilizarán en el proyecto, se realizará un estudio del estado actual para valorar las mejores soluciones. Es importante trabajar con herramientas de

acceso libre y código abierto ya que son las tecnologías más asequibles y con más documentación. Se valorarán las ventajas y las desventajas de cada herramienta y se escogerá la que más se ajusta a las necesidades del proyecto.

Para el análisis de los datos se revisarán otros estudios *Big Data* relacionados con el procesado de datos *GPS* y de optimización de rutas mediante *machine learning* e inteligencia artificial. Se programarán distintos modelos para obtener distintos resultados, el trayecto más corto, el viaje más rápido, el mayor ahorro de combustible, por ejemplo.
Para el uso de las herramientas y el procesado de los datos se usará un entorno virtual con contenedores y máquinas virtuales según la aplicación necesaria. De esta forma, el entorno será escalable, seguro y modular, pudiéndose utilizar sobre servidores físicos o en la nube.

1.5. Planificación del trabajo

La planificación del trabajo se define en 5 fases principales, definidas por las entregas PEC. En la Figura 1 aparece el diagrama de Gantt con la duración de cada tarea, no obstante, a continuación, se desglosa cada actividad en un conjunto de actividades:

PEC1 – Definición de objetivos y alcance del TFM 27/09/2023 – 09/10/2023

- Tarea 1.1 Resumen y descripción del proyecto - 1 día
- Tarea 1.2 Motivación y justificación del proyecto - 3 días
- Tarea 1.3 Objetivos esperados del proyecto, impacto sobre la sociedad - 1 día
- Tarea 1.4 Enfoque y método seguido – 1 día
- Tarea 1.5 Creación de la planificación y de la memoria
- **Entregable 1 – Documento con los capítulos introductorios del proyecto con un índice preliminar**

PEC2 – Estado del arte del proyecto 10/10/2023 – 25/10/2023

- Tarea 2.1 Estudio de las distintas tecnologías *Big Data* - 2 día
- Tarea 2.2 Análisis de los sistemas adecuados para procesamiento de datos *GPS* – 2 días
- Tarea 2.3 Ventajas e inconvenientes de cada tecnología – 2 días
- Tarea 2.4 Elección y planteamiento del sistema de análisis - 7 días
- **Entregable 2 – Documento detallando el estado actual de los sistemas *Big Data* y solución planteada para el proyecto**

PEC 3 – Diseño e implementación 26/10/2023 – 17/12/2023

- Tarea 3.1 Configuración del *hardware* y del *software* utilizado – 7 días
- Tarea 3.2 Instalación y configuración del entorno – 8 días
- Tarea 3.3 Análisis de datos y pruebas de rendimiento – 2 días
- Tarea 3.4 Desarrollo del mapa de calor y representación gráfica – 10 días
- Tarea 3.5 Programación y desarrollo de algoritmos de *machine learning* e inteligencia artificial – 6 días
- Tarea 3.6 Análisis y recopilación de los resultados – 10 días

- **Entregable 3 – Descripción del producto configurado y obtención de resultados**

PEC 4 – Desarrollo memoria final 18/12/2023 – 08/01/2024

- Tarea 4.1 Revisión de los resultados - 7 días
- Tarea 4.2 Ajuste de figuras, bibliografía y formato – 3 días
- Tarea 4.3 Conclusiones obtenidas y futuras mejoras - 7 días
- **Entregable 4 – Memoria detallada del proyecto con los resultados obtenidos**

PEC 5 – Presentación del proyecto 09/01/2024 – 17/01/2024

- **Entregable 5 – Presentación del proyecto**

Name	Begin date	End date
TFM - Big Data - Análisis de tráfico y optimización de rutas con machine lear...	27/09/23	17/01/24
PEC 1 - Definición de objetivos y alcance del TFM	27/09/23	09/10/23
PEC 1.1 - Resumen y descripción del proyecto	27/09/23	28/09/23
PEC 1.2 - Motivación	29/09/23	02/10/23
PEC 1.3 - Objetivos del proyecto	03/10/23	04/10/23
PEC 1.4 - Enfoque y método seguido	05/10/23	06/10/23
PEC 1.5 - Creación de la planificación y propuesta de índice	09/10/23	09/10/23
PEC 2 - Estado del arte	10/10/23	25/10/23
PEC 2.1 - Estudio de las distintas tecnologías big data	10/10/23	11/10/23
PEC 2.2 - Análisis de los sistemas adecuados	12/10/23	13/10/23
PEC 2.3 - Planteamiento de ventajas y desventajas	16/10/23	17/10/23
PEC 2.4 - Elección y planteamiento del sistema de procesado	18/10/23	25/10/23
PEC 3 - Diseño e implementación	26/10/23	15/12/23
PEC 3.1 - Entorno hardware y software utilizado	26/10/23	03/11/23
PEC 3.2 Instalación y configuración inicial	06/11/23	14/11/23
PEC 3.3 - Análisis y pruebas de rendimiento	15/11/23	16/11/23
PEC 3.4 - Desarrollo del mapa de calor	17/11/23	27/11/23
PEC 3.5 - Programación de algoritmos machine learning	28/11/23	04/12/23
PEC 3.6 - Análisis de los resultados	05/12/23	15/12/23
PEC 4 - Desarrollo memoria final	18/12/23	08/01/24
PEC 4.1 - Revisión de los resultados	18/12/23	25/12/23
PEC 4.2 - Ajuste de figuras, bibliografía	26/12/23	28/12/23
PEC 4.3 - Conclusiones obtenidas y futuras mejoras	29/12/23	08/01/24
PEC 5 - Presentación y defensa	09/01/24	17/01/24

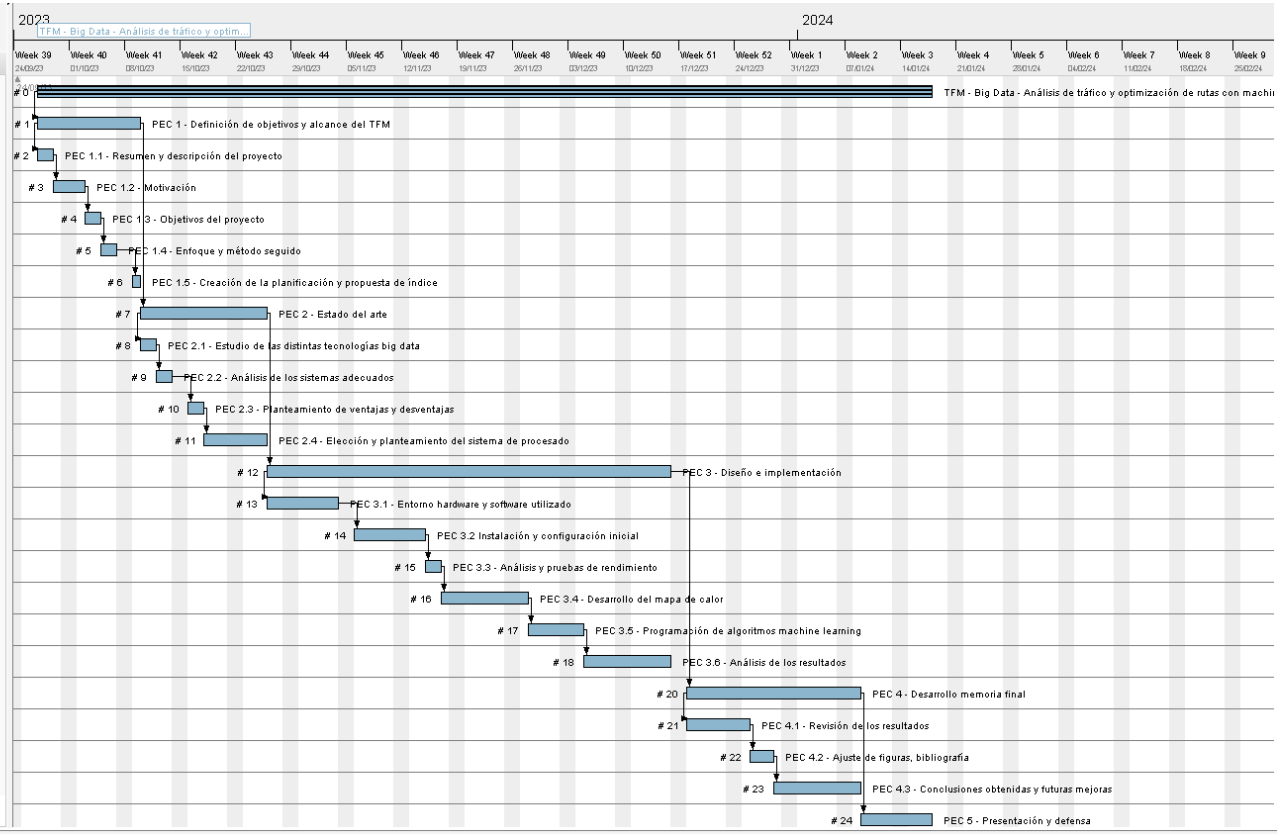


Figura 1. Planificación del proyecto

1.6. Breve sumario de productos obtenidos

En este proyecto se han obtenido los siguientes resultados:

- Análisis de los datos y del formato.
- Mapas de calor con la media del tráfico en Madrid.
- Análisis de rutas más cortas, mediante algoritmos convencionales y optimización *machine learning* mediante modelos bioinspirados.
- Comparación y evaluación de los resultados.

1.7. Breve descripción de los otros capítulos de la memoria

En el segundo capítulo de la memoria se estudia la evolución de los datos y el valor que aporta el análisis. Se revisan las herramientas desarrolladas para procesar los datos, como funcionan los algoritmos de *machine learning* y los distintos tipos de inteligencia artificial. Por último, se mencionan los proyectos actuales y las herramientas de optimización de rutas más conocidas.

En el tercer capítulo se describe la implementación del entorno virtual de trabajo, la configuración de las aplicaciones utilizadas, la carga y el procesado de los datos. Se realizan las pruebas con los distintos algoritmos y se analizan los resultados.

En el cuarto capítulo se evalúan los resultados obtenidos comparados con la planificación inicial y se explican las conclusiones del proyecto.

En el quinto capítulo se plantean futuras mejoras del proyecto y de la infraestructura actual. Para un futuro sostenible, el uso de la tecnología para optimizar procesos será esencial.

Los capítulos restantes incluyen el glosario, la bibliografía utilizada y en anexos, hay partes del código empleado.

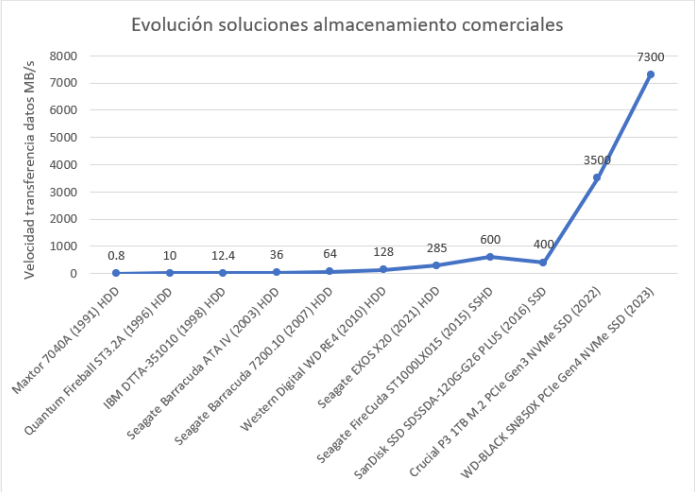
2. Análisis y estudio de las soluciones Big Data, estado del arte

Para tener una visión global de la tecnología *Big Data*, se revisará como nacen estos sistemas y las herramientas desarrolladas para ajustarse a cada caso. Además, se analizarán los distintos tipos de inteligencia artificial y algoritmos *machine learning*. Por último, revisaremos los proyectos y las soluciones actuales relacionadas con *Big Data* e inteligencia artificial aplicadas a optimización de sistemas.

2.1. Introducción *Big Data*

Big Data se refiere a un volumen masivo de datos no estructurados demasiado grande como para ser procesado en una base de datos convencional. Con el desarrollo de las telecomunicaciones, el auge de los dispositivos multimedia y el aumento de sensores conectados a internet, la cantidad de datos generados y almacenados sigue aumentando cada día exponencialmente.

Si bien la capacidad de almacenar datos ha crecido enormemente, la velocidad a la que se pueden leer y escribir los datos no. Por ejemplo, los discos duros (*HDD*) comerciales actuales llegan a velocidades de 300MB/s de escritura y lectura mientras que los nuevos discos de estado sólido (*SSD*) pueden llegar a 7300 MB/s. Los discos híbridos *SSHD* combinan discos *HDD* y memoria cache *NAND* para obtener más velocidad de lectura para ficheros pequeños, si el fichero sobrepasa la memoria *NAND* la velocidad bajará. En la gráfica siguiente se representa la evolución de velocidad máxima de lectura y escritura:



Gráfica 1. Evolución velocidad de escritura y lectura en soluciones de almacenamiento comerciales

En entornos empresariales, hay soluciones específicas que consiguen velocidades aún más altas agrupando el almacenamiento para que funcione en paralelo. En clústeres de alta capacidad de computación se utiliza *hardware* escalable de altas prestaciones, que maximiza el rendimiento de los componentes para obtener las mejores características de cómputo.

Para conseguir leer y grabar grandes cantidades de datos hay que procesar en paralelo los datos. Al fraccionar los *datasets* en distintos discos el sistema debe ser tolerante a fallos de disco, se debe utilizar una configuración que permite replicar los datos en el almacenamiento para proteger los datos ante corrupción. [4, p. 24]

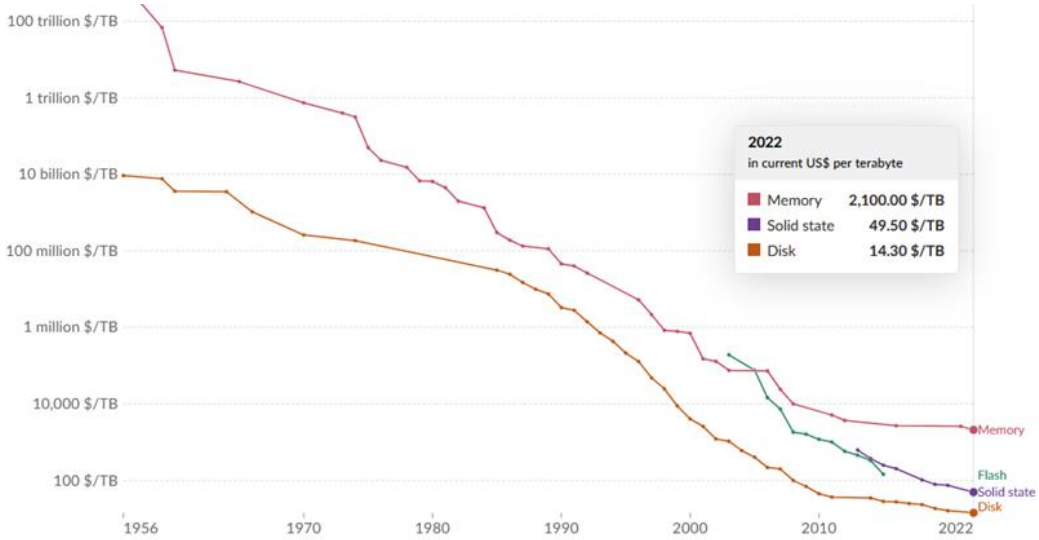
Por ejemplo, los fabricantes actuales de soluciones de almacenamiento no sólo proporcionan discos, sino que proporcionan servidores con capacidad de cómputo, memoria *RAM*, discos *SSD* y discos *HDD*. Estos sistemas se venden como soluciones de almacenamiento *Scale-Out*, siendo posible ampliar la capacidad de almacenamiento y computo añadiendo más nodos. El sistema operativo de estos servidores es capaz de distribuir los ficheros, creando un entorno con alta disponibilidad, redundancia de datos y medidas de seguridad para proteger la información en caso de corrupción o ciberataques. Además, algunos fabricantes ofrecen la posibilidad de comprimir y deduplicar la información, es decir, reducir el tamaño de los datos almacenados en caso de información repetitiva o comprimible.

Big Data se refiere a la gestión de conjuntos masivos de datos, almacenados sin una estructura definida, sin estar previamente procesados. Para explicar mejor que implica *Big Data*, se utilizan los seis “V”, volumen, velocidad, variedad, veracidad, valor y variabilidad:

- Volumen – los sistemas *Big Data* requieren almacenar grandes cantidades de datos, permitir flexibilidad y crecimiento.
- Velocidad – los sistemas deben ser escalables y elásticos para aguantar el aumento en la velocidad de generación y procesado de los datos.
- Variedad – se refiere a los distintos tipos y estructuras de datos. Los sistemas deben ser compatibles con varios formatos.
- Veracidad – los datos almacenados tienen que ser correctos y fiables, los sistemas de almacenamiento deben ser resilientes y resistentes a la corrupción de datos.
- Valor – el sistema debe permitir procesar los datos para extraer el valor y la información requerida.
- Variabilidad – el sistema debe adaptarse a cambios rápidos en la estructura y la cantidad de los datos.

En resumen, para poder almacenar y procesar grandes cantidades de datos en un sistema *Big Data* se deben tener en cuenta tres factores principales. Los sistemas tienen que mover y procesar grandes cantidades de datos en tiempo real, deben ser escalables y resilientes a pérdidas de servicio y el factor más importante, deben tener un precio bajo y operar con componentes eficientes.

El coste del almacenamiento ha ido bajando con el aumento en la densidad de bits almacenados y debido a las mejoras en el proceso de fabricación. En la gráfica siguiente se puede observar cómo cada avance mejora el coste por capacidad de almacenamiento en *Terabytes*:



Gráfica 2. Coste histórico de memoria RAM y almacenamiento HDD y SSD en dólares para 1 TB de capacidad

A la vez, la capacidad de cómputo ha aumentado exponencialmente siendo cada vez más asequibles los procesadores de altas prestaciones con múltiples núcleos. Esto ha generado una expansión hacia el análisis de los datos, generación de algoritmos de aprendizaje automático e inteligencia artificial en empresas privadas y públicas. Este análisis se ha convertido en un componente integral de varias industrias para poder realizar predicciones y optimizar procesos analizando los datos históricos y los datos recibidos de distintos sensores en tiempo real mejorando los costes de operación, las ganancias y maximizando recursos.

2.2. Herramientas Big Data

Dada la creciente necesidad de procesar grandes cantidades de información, se desarrollaron distintas herramientas capaces de almacenar y analizar grandes cantidades de datos de forma eficiente. Debido a los distintos tipos de análisis necesarios, no sólo hay que tener en cuenta los recursos sino también el tipo de herramienta utilizada para aprovechar al máximo la capacidad de cómputo.

Principalmente se utiliza el ecosistema de *Hadoop*, publicado en 2007 como una implementación de código abierto de *MapReduce*, un motor de procesamiento vinculado a un sistema de ficheros distribuidos. Empezó a utilizarse en el motor de búsqueda de *Yahoo* y *Facebook* entre otras grandes empresas. *Hadoop* está formado por tres capas principales, la capa de almacenamiento *HDFS*, la capa de gestión de recursos y programación de tareas *YARN* (*Yet Another Resource Negotiator*) y por último la capa dónde se procesan los datos *MapReduce*.

Hadoop se desarrolla para poder analizar datos escritos una vez y leídos muchas veces, al contrario de una base de datos tradicional dónde la información está indexada y los datos son actualizados continuamente. Una base de datos relacional tiene alta integridad, con una estructura concreta y puede ocupar varios *Gigabytes*, mientras que Hadoop funciona con

datos no estructurados o semiestructurados con tamaños de varios *Petabytes*. Esto aporta más flexibilidad y evita la fase de carga de datos de una base de datos tradicional ya que en Hadoop sólo se tienen que copiar los ficheros. [4, p. 29]. Además de ser más rápido que una base de datos relacional, *Hadoop* aprovecha los recursos de cómputo de alto rendimiento utilizando almacenamiento local.

HDFS, uno de los componentes clave de *Hadoop*, es un sistema de ficheros distribuido que controla el almacenamiento de una red de servidores de almacenamiento. Está diseñado para almacenar grandes cantidades de datos en hardware común, no se necesitan servidores especialmente diseñados para clústeres de alta computación. Se utilizan bloques de datos para simplificar la gestión de la información y por defecto son de 128 *MB* para reducir el número de búsquedas. La gestión de los directorios se realiza a través de *NameNodes*, componentes que gestionan la ubicación, la jerarquía y la estructura de los datos y *DataNodes*, componentes que almacenan los bloques y realizan las operaciones de lectura y escritura de datos.

YARN se utiliza para gestionar los recursos del clúster *Hadoop*, también soporta otras plataformas y aplicaciones de procesamiento de datos. Funciona utilizando un gestor de recursos por clúster y gestores de nodo por cada nodo del clúster para gestionar los contenedores de aplicaciones. Al ejecutarse una tarea, el gestor de recursos de *YARN* envía la tarea al gestor de nodo que crea un contenedor para la tarea asegurándose de que tiene suficientes recursos. Si no hay suficientes recursos o hay algún problema, *YARN* es capaz de detectarlos. [4, p. 98]

Con *MapReduce* se pueden analizar grandes cantidades de datos dividiendo el análisis, primero se mapean los datos, dividiéndolos en fragmentos más pequeños y luego se reducen los datos anteriores realizando varias operaciones en paralelo obteniendo una salida con el resultado. La paralelización de las consultas acelera el cálculo y si hay algún error, *MapReduce* vuelve a reintentar la consulta.

Además de estas herramientas que forman la base del sistema *Hadoop*, se han desarrollado otros módulos según la capa de funcionamiento y el tipo de procesado necesario para maximizar los recursos de cómputo [5]. A continuación, se detallan las herramientas más utilizadas:

Capa de almacenamiento:

- *HBASE* y *Apache Cassandra* son herramientas que almacenan los datos en columnas en lugar de utilizar filas y columnas.
- *Hive* y *Tez* permiten realizar consultas *SQL* interactivas sobre los datos almacenados.
- *Apache Storm*, *Apache Pig*, *Apache Spark* y *Samza* procesan los datos mediante flujos de datos y pueden ejecutar tareas de extracción, transformación y carga.

Capa de procesado, además de procesar los datos esta capa también emplea herramientas de *machine learning* e inteligencia artificial:

- *Flume*, *Kafka* y *Sqoop* son herramientas que integran los datos almacenados en *HDFS* con los motores de consultas.
- *Cascading* implementa una *API (Application Programming Interface)* para procesar los datos mediante los principales lenguajes de programación, *Scala*, *Java* y *Python* entre otros.
- *H2O* permite procesar y analizar los datos en paralelo implementando librerías de *machine learning* con una interfaz *web*.

Capa de gestión y programación:

- *Apache Ambari* permite gestionar y monitorizar varios clústeres *Hadoop*.
- *Apache ZooKeeper* es una herramienta de configuración y coordinación de contenedores y clústeres.

En la siguiente figura se representa el ecosistema actual de *Hadoop* con la herramienta principal de cada función [6]:

Figura 2. Ecosistema de aplicaciones Hadoop

2.3. Algoritmos de aprendizaje automático

El aprendizaje automático o *machine learning* es una rama de la inteligencia artificial que se centra en procesar y analizar los datos mediante algoritmos, aprendiendo de forma iterativa y mejorando los resultados gradualmente [7].

Uno de los primeros sistemas de aprendizaje automático fue desarrollado por Arthur Samuel en 1962, utilizando servidores de IBM para ganarle en ajedrez a un maestro en un partido que duró 5 meses [8]. Los sistemas de aprendizaje automático están formados por tres funciones principales generalmente:

-un proceso de decisión, un algoritmo analiza los datos y busca un patrón

-una función de error, se mide el resultado de las búsquedas y la precisión, comparando con ejemplos conocidos

-un proceso de optimización o actualización, el algoritmo analiza los errores y mejora el proceso de decisión para que disminuya la probabilidad de error

Los sistemas de aprendizaje automático se utilizan para automatizar tareas, analizar grandes cantidades de datos de distintos formatos (texto, voz, imágenes, video) y para extraer información relevante. Para crear los modelos de inteligencia artificial se usan los siguientes algoritmos [9]:

- Regresión lineal, modela la relación entre variables independientes y una variable objetivo.
- Regresión logística, predice la probabilidad de que ocurra un suceso binario.
- Redes neuronales, replicando el pensamiento de un ser humano se analizan patrones y se realizan predicciones. Los datos se analizan por capas creando una red
- Arboles de decisión
- Bosque aleatorio

2.4. Optimización de tráfico mediante Big Data e Inteligencia Artificial

Dada la capacidad de procesamiento de los sistemas *Big Data* se ha invertido en investigación y desarrollo de sistemas capaces de optimizar el tráfico y la movilidad en ciudades utilizando sensores y datos creados por los ciudadanos.

A nivel europeo se está invirtiendo en proyectos de investigación BigData para fomentar el procesado de los datos de forma segura y la competitividad a nivel global [10]. En 2017 se lanzó el proyecto *Transforming Transport* (https://transformingtransport.eu), liderado por Indra con la participación de 9 países y 47 organizaciones. El proyecto incluía varios pilotos aplicados a diferentes medios de transporte integrando datos reales de diversas fuentes (sensores, redes sociales, cámaras…). El proyecto pretendía desarrollar patrones de transporte, mejorar la experiencia de los usuarios optimizando la eficiencia operativa de los servicios y los procesos vinculados con el transporte. [11]

El proyecto no sólo se aplicaba a ciudades sino a aeropuertos, vías ferroviarias, autovías y puertos también. Tras la implementación de los modelos, los resultados han demostrado

reducciones de la polución de 15-25% y de 15% en la reducción de los costes de mantenimiento de la infraestructura. El servicio prestado por los distintos sectores mejoraba, se facilitaba la gestión y surgían nuevos modelos de negocio. [12]

Otro proyecto que utiliza *edge computing* (procesamiento de datos dónde se recogen para reducir costes de transmisión) es EMERALDS *(Extreme-scale Urban Mobility Data Analytics as a Service)*. El proyecto fundado por la Unión Europea intenta optimizar el tráfico y los sistemas de transporte público empleando sensores y dispositivos IoT vinculados mediante *5G* en varias ciudades de Europa. Uno de los casos de uso del proyecto es poder predecir congestiones durante eventos. [13]

Además de optimizar el tráfico y reducir costes, *Big Data* se puede utilizar para elegir las mejores ubicaciones dónde colocar estaciones o paradas de autobús, predecir demanda de servicios de transporte público y calcular la afectación en otros sectores e industrias.

Por ejemplo, el Ayuntamiento de Barcelona y la Dirección General de Tráfico lanzaron en 2019 el proyecto *Autonomous Ready*, un sistema que mejora la infraestructura y los servicios gracias a la información extraída de dispositivos y sensores. El objetivo principal del proyecto es reducir los accidentes en la ciudad y mejorar la seguridad de los ciudadanos, utilizando sistemas que ayudan a la conducción y sensores de visión basados en inteligencia artificial en los vehículos públicos. Los sensores utilizados pueden mejorar la movilidad comunicando incidencias en tiempo real a los usuarios para prevenir accidentes y evitar congestiones. [14]

Actualmente existen varias soluciones comerciales o gratuitas que utilizan *Big Data* para optimizar y mostrar el tráfico en tiempo real. A continuación, se enumeran algunos ejemplos:

- *Google Maps Platform*, utilizando los datos recopilados a través del servicio gratuito de *Google Maps* los usuarios de la aplicación pueden ver la congestión y planificar rutas. Además, *Google* ofrece un servicio de pago a través de su plataforma donde se puede utilizar una *API* para gestionar una flota de vehículos. La plataforma también permite utilizar datos propios y se puede utilizar en aplicaciones o páginas web. [15]

- *Waze*, ofrece los mismos servicios que *Google Maps* pero con algunas mejoras, permite a los usuarios añadir información sobre atascos, accidentes, información sobre puntos de interés. También se integra con aplicaciones y ofrece servicios de pago para empresas. [16]

- *FleetRoot* es una aplicación desarrollada para optimizar rutas y entregas tipo *last-mile* en vehículos de empresas. [17]

3. Diseño, implementación y resultados

En este apartado se detallará el diseño, la configuración y los cálculos realizados con sistemas *Big Data*. Los pasos seguidos son los siguientes:

- Búsqueda de datos relacionados con el tráfico en una ciudad.
- Instalación y configuración de un sistema que analice los datos.
- Pruebas de rendimiento y calculo para determinar las mejores aplicaciones.
- Desarrollo de un mapa de calor relacionado con la intensidad del tráfico.
- Programación y optimización de distintos algoritmos para obtener la mejor ruta.
- Análisis de los resultados obtenidos.

3.1. Datos utilizados en el análisis

Como se comentaba inicialmente, el proyecto es una prueba de concepto, por lo tanto, se utiliza datos limitadas a la zona de Madrid. Usando una cantidad de datos estática y acotada se puede comprobar la funcionalidad de los sistemas en un entorno controlado. Una vez comprobado el funcionamiento de cada sistema, como futura mejora se pueden utilizar datos en tiempo real, ampliando el alcance del proyecto.

Tras revisar varias fuentes de datos, se elige el repositorio del ayuntamiento de Madrid, por ser una fuente fiable, con datos históricos desde 2013 en un formato estandarizado y documentado. Los datos se actualizan cada mes, incrementando la cantidad de sensores y los parámetros tomados. [18]
Desde el portal https://datos.madrid.es se pueden descargar los datos sobre la ciudad de Madrid. En este caso se descargan los datos desde junio de 2022 hasta julio de 2023, de esta forma obtenemos un fichero en formato ***.CSV*** para cada mes. Cada fichero contiene más de 11 millones de registros y los datos tienen el siguiente formato [19]:

Nombre	Tipo	Descripción
Id	Entero	Identificación del punto de medida. Es de tipo secuencial, además de ser único e invariable.
Fecha	Fecha	Fecha y hora oficiales de Madrid con formato dd/mm/yyyy hh:mi:ss
Tipo_elem	Texto	Nombre del tipo de punto de medida: Urbano o M30.
Intensidad	Entero	Número de vehículos en el periodo de 15 minutos, expresada en vehículos/hora. El valor efectivo de vehículos que han circulado en ese periodo se obtiene dividiendo entre cuatro. Un valor negativo implica la ausencia de datos.
Ocupación	Entero	Porcentaje de tiempo que está un detector de tráfico ocupado por un vehículo. Por ejemplo, una ocupación del 50% en un periodo de 15 minutos significa que ha habido vehículos situados sobre el detector durante 7 minutos y 30 segundos. Un valor negativo implica la ausencia de datos.
Carga	Entero	Parámetro de carga del vial en el periodo de 15 minutos. Representa una estimación del grado de congestión, calculado a partir de un algoritmo que usa como variables la intensidad y ocupación, con ciertos factores de corrección. Establece el grado de uso de la vía en un rango de 0 (vacía) a 100 (colapso). Un valor negativo implica la ausencia de datos.
vmed	Entero	Velocidad media de los vehículos en el periodo de 15 minutos (Km./h). Solo para puntos de medida interurbanos M30. Un valor negativo implica la ausencia de datos.

Error	Texto	Indicación de si ha habido al menos una muestra errónea o sustituida en el periodo de 15 minutos. N: no ha habido errores ni sustituciones E: los parámetros de calidad de alguna de las muestras integradas no son óptimos. S: alguna de las muestras recibidas era totalmente errónea y no se ha integrado.
Periodo_integracion	Entero	Número de muestras recibidas y consideradas para el periodo de integración.

Tabla 1. Estructura datos control de tráfico Madrid

Una de las mayores preocupaciones al trabajar con datos es la integridad y el formato de estos. La información que pública el Ayuntamiento de Madrid es de una fuente fiable, contiene datos reales y el formato es siempre el mismo, esto facilitará el procesado de los datos.

Tras obtener los datos del tráfico en Madrid durante un año, se fusionaron los 12 ficheros en uno sólo para abreviar las consultas y los comandos.

3.2. Instalación y configuración del entorno

Al empezar el proyecto se utilizó un ordenador con la distribución *Linux Ubuntu*. Por problemas con la máquina, se utilizó un ordenador con más recursos de cómputo con el Subsistema de *Windows* para *Linux* (*WSL 2*) para poder ejecutar comandos de *Linux Ubuntu*. También se utiliza *Docker Desktop* para ejecutar aplicaciones que funcionan a través de contenedores para balancear la carga.

Para analizar datos se utiliza *Apache Spark* y el lenguaje de programación *Python*. *Apache Spark* funciona con el lenguaje de programación de alto nivel *Scala*, pero también se puede usar con *Python*, *SQL*, *Java* y *R*. Para utilizar el mismo lenguaje para las consultas se configura la versión que funciona con *Python* y consultas *SQL*, *PySpark*. Permite paralelizar las tareas de forma automática para agilizar el procesado de los resultados. Como carga los datos directamente sobre la memoria *RAM* del sistema, las operaciones se agilizan considerablemente. Otra ventaja de *PySpark* es la facilidad con la que se pueden instalar nuevas librerías para el análisis de los datos.

Figura 3. Apache Spark versión 3.1.1 con Python versión 3.10.12

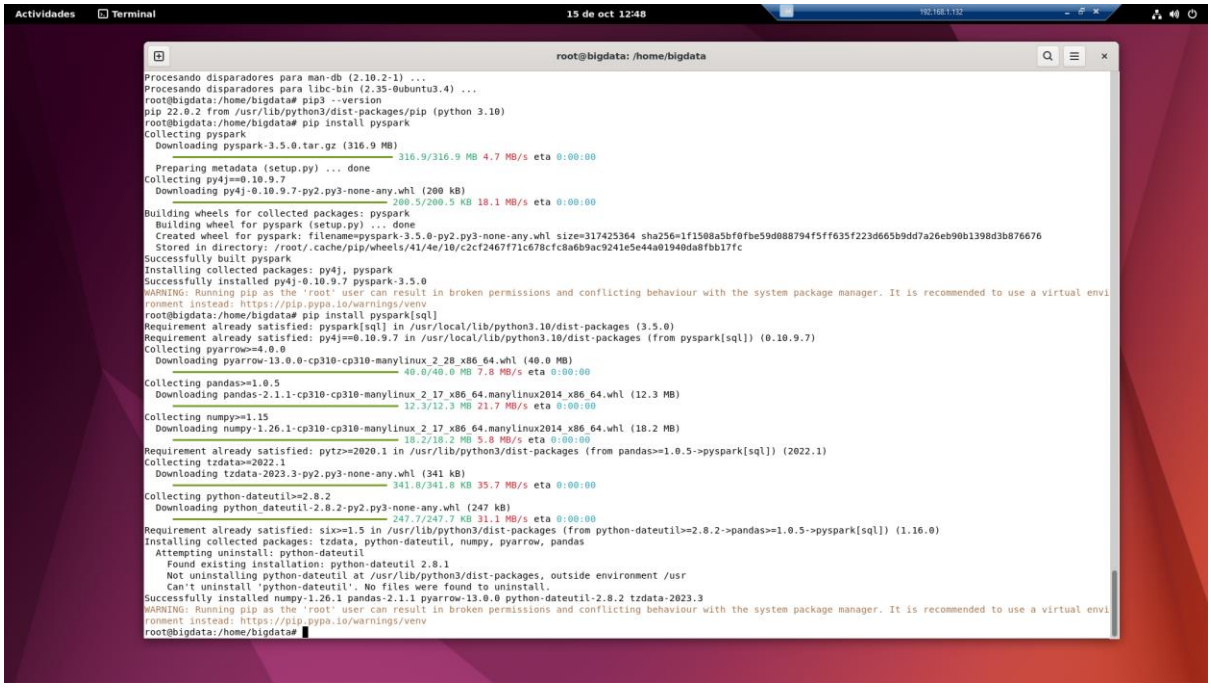


Figura 4. Instalación inicial Java, PIP y Spark

Inicialmente se utilizó *Jupyter Notebooks* sobre contenedores para tener una interfaz gráfica para ver los mapas de calor. En *Jupyter* el *DataFrame* se define con *Pandas* utilizando *Python* como el *Kernel*:

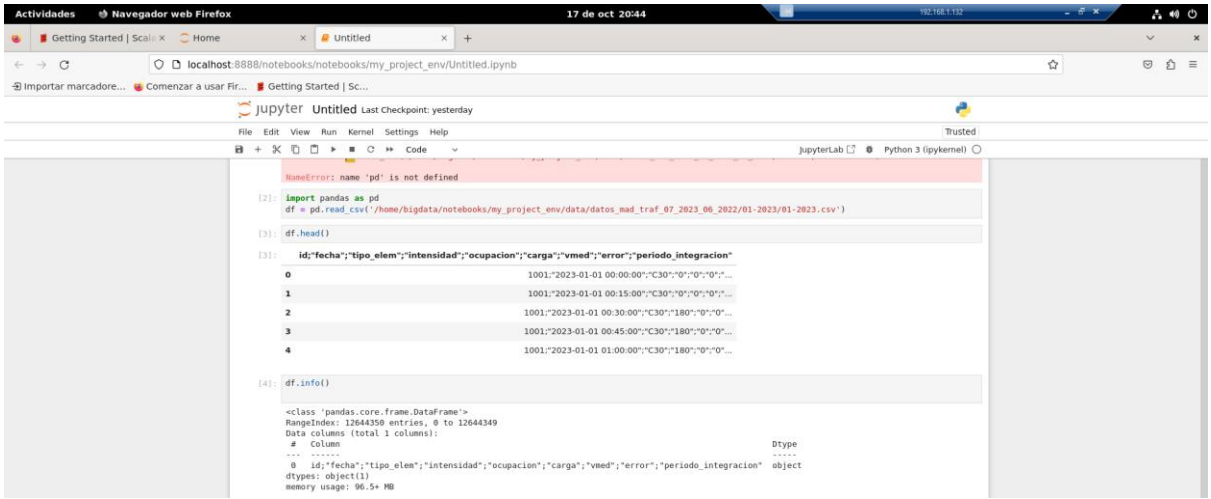


Figura 5. Datos tráfico Julio 2023 importados en Jupyter Notebooks

Como los mapas de calor de *Jupyter Notebooks* utilizando el módulo *Folium* cambiaban al ampliar el mapa y los cambios en la intensidad de tráfico no eran discernibles, se utilizó la versión gratuita del programa *Microsoft Power BI* para representar las variaciones de intensidad en los mapas de calor.

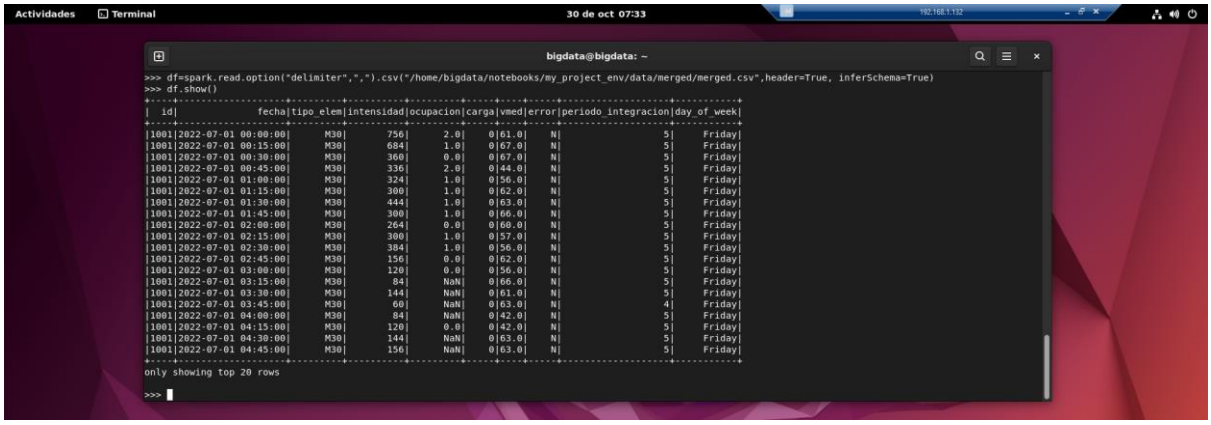


Figura 6. Formato datos

3.3. Procesado de los datos y creación mapas de calor

Al trabajar con la información, *PySpak* carga en la memoria *RAM* el conjunto de datos para agilizar las consultas y para optimizar los trabajos, esto también corta el tiempo de ejecución de las tareas. Como los datos ocupan alrededor de 9 *GB*, para llamar a un único fichero en todas las consultas se fusionan todos los valores de cada mes en un sólo fichero.

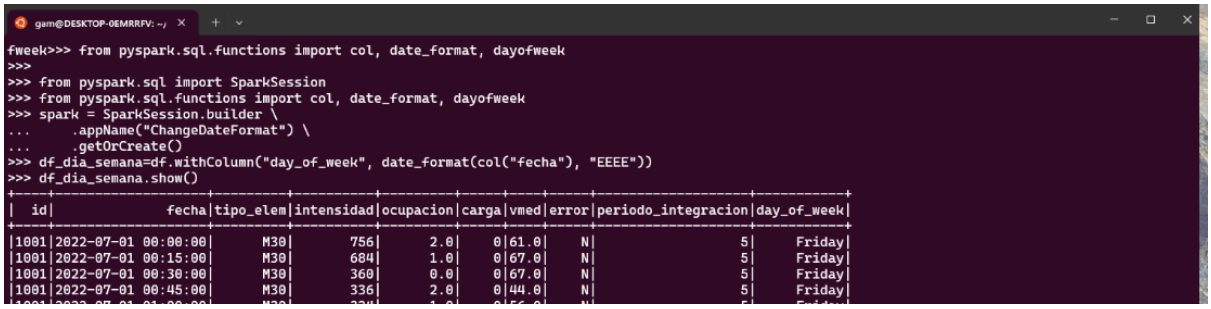


Figura 7. En los datos iniciales se añade una columna adicional con el día de la semana

El objetivo inicial era crear un mapa de calor con el cambio del tráfico, calculando la media de la intensidad por día de la semana y ubicación. Calculando los valores medios con spark, se obtiene la intensidad media por día y por hora en la ubicación de cada sensor. Cruzando estos datos con las coordenadas físicas de cada sensor, se podrá asignar una intensidad de tráfico a cada camino.

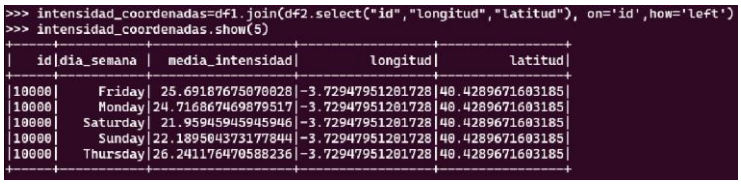


Figura 8. El conjunto de datos intensidad_coordenadas se puede utilizar para crear mapas de calor

Al comienzo del análisis, para dibujar los mapas de calor, se utilizó el entorno gráfico de *Jupyter Notebooks*, una aplicación de código abierto que funciona sobre contenedores *Docker*. Permite realizar consultas usando los lenguajes de programación *Julia*, *Python* y *R*

en un entorno web, flexible y se pueden configurar tareas y flujos de trabajo. Si no tiene un módulo o una característica, se puede instalar con facilidad al ser compatible con el administrador de bibliotecas de *Python*, *PIP*.

Los mapas de calor se crearon usando *Pandas* como la herramienta de gestión de datos y *Folium* para dibujar los puntos, usando como centro del mapa la media de las coordenadas:

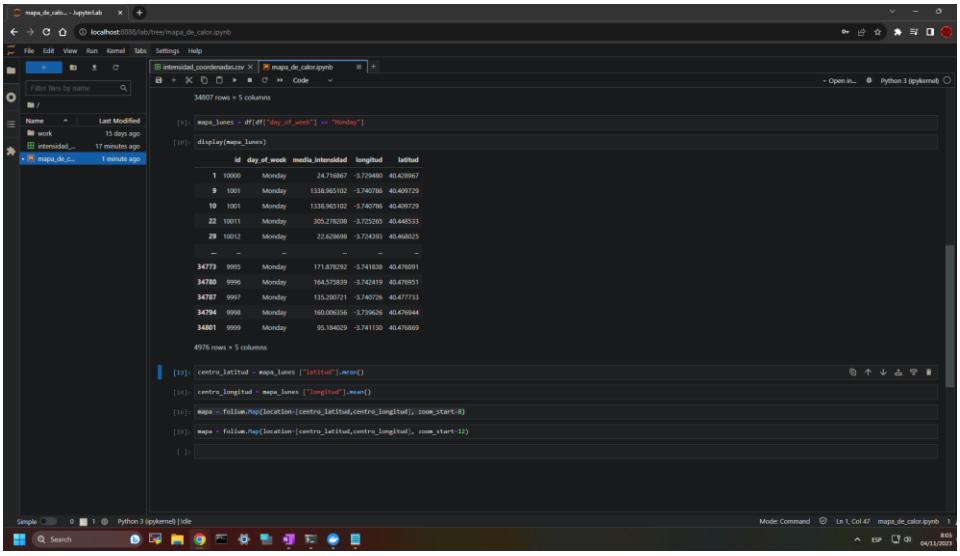


Figura 9. Jupyter Notebook con los datos cargados en el dataframe

Tras ejecutar el código de la Figura 9, obtenemos el siguiente mapa de calor:

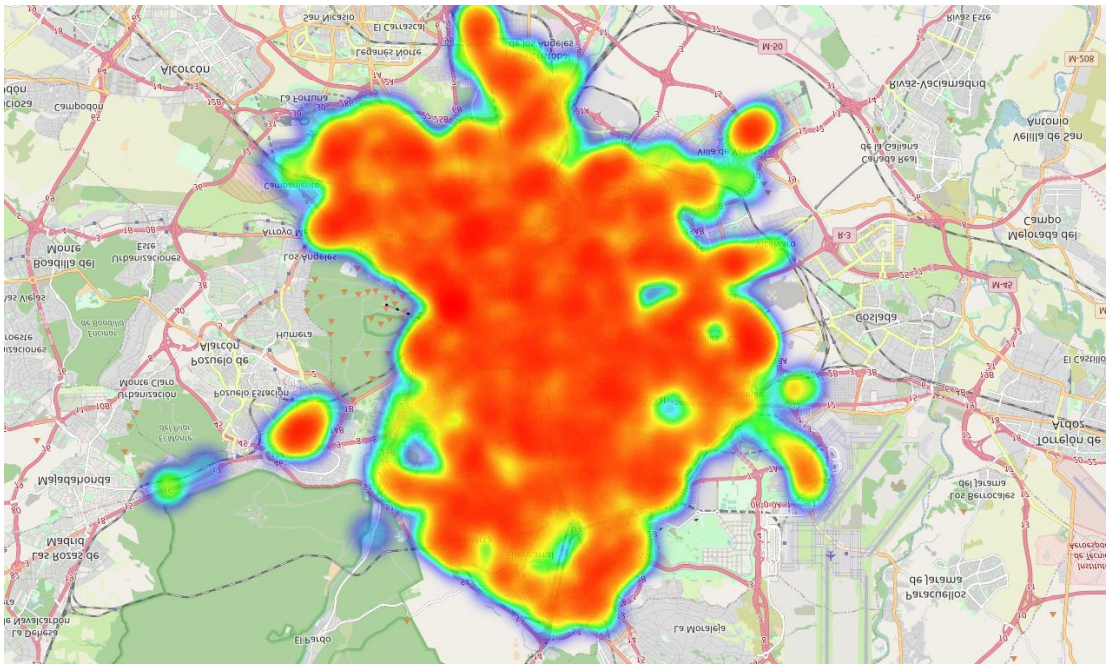


Figura 10. Mapa de calor usando la intensidad de tráfico en Madrid empleando como datos todos los lunes del año

Ampliando el mapa, se pueden ver claramente las zonas dónde están ubicados los sensores y dónde se detecta más intensidad de tráfico:

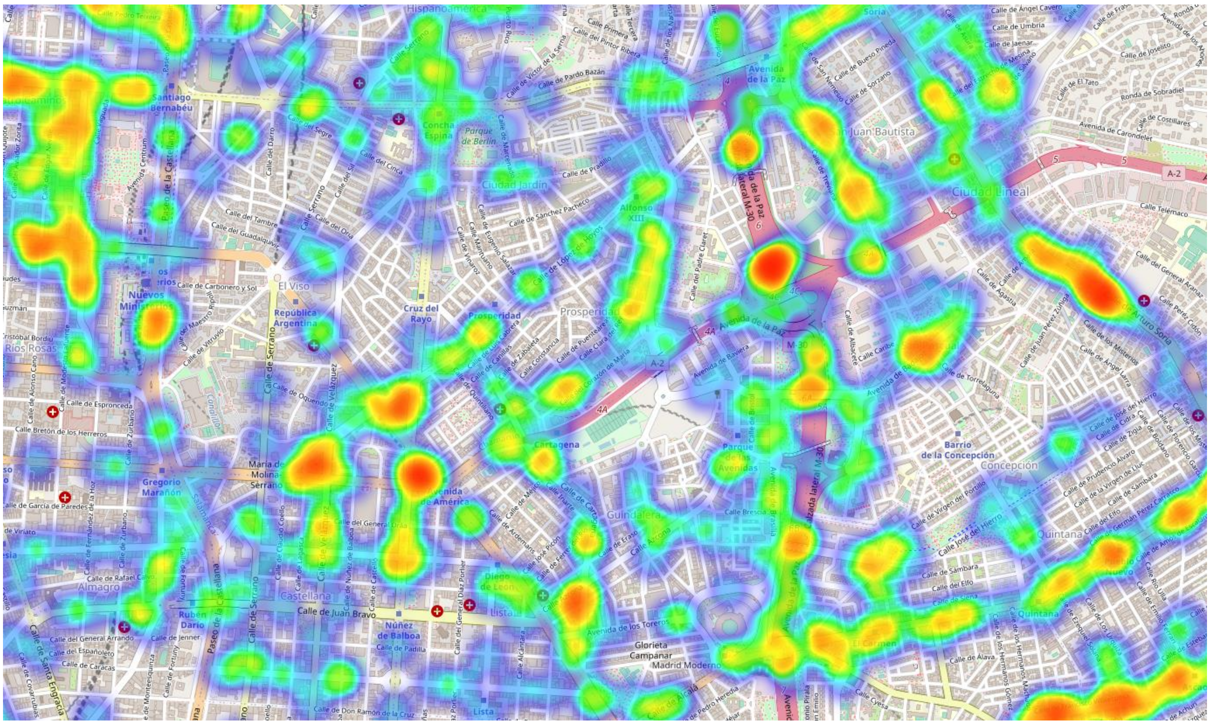
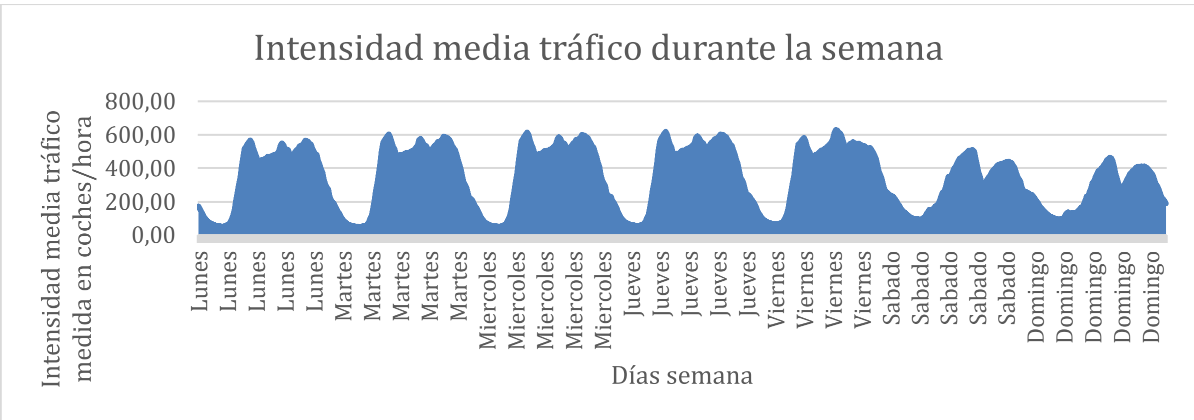


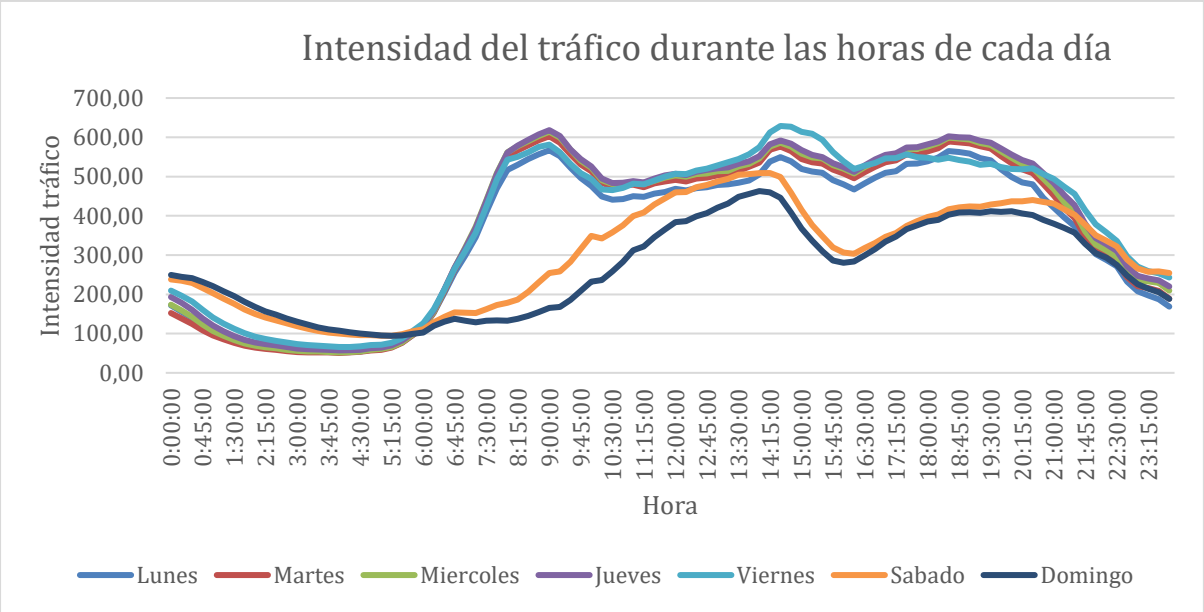
Figura 11. Intensidad de tráfico en las calles de Madrid

El planteamiento inicial era observar la evolución de la intensidad de tráfico según el mapa de calor durante cada día de la semana, pero con Folium, la diferencia entre los mapas de calor obtenidos para cada día de la semana no era discernible. Tampoco era evidente la evolución a lo largo del día. Es por ello por lo que se crean las siguientes gráficas con la intensidad de tráfico por día de la semana y por hora:



Gráfica 3. Intensidad media de tráfico durante la semana, utilizando los datos de todo el año

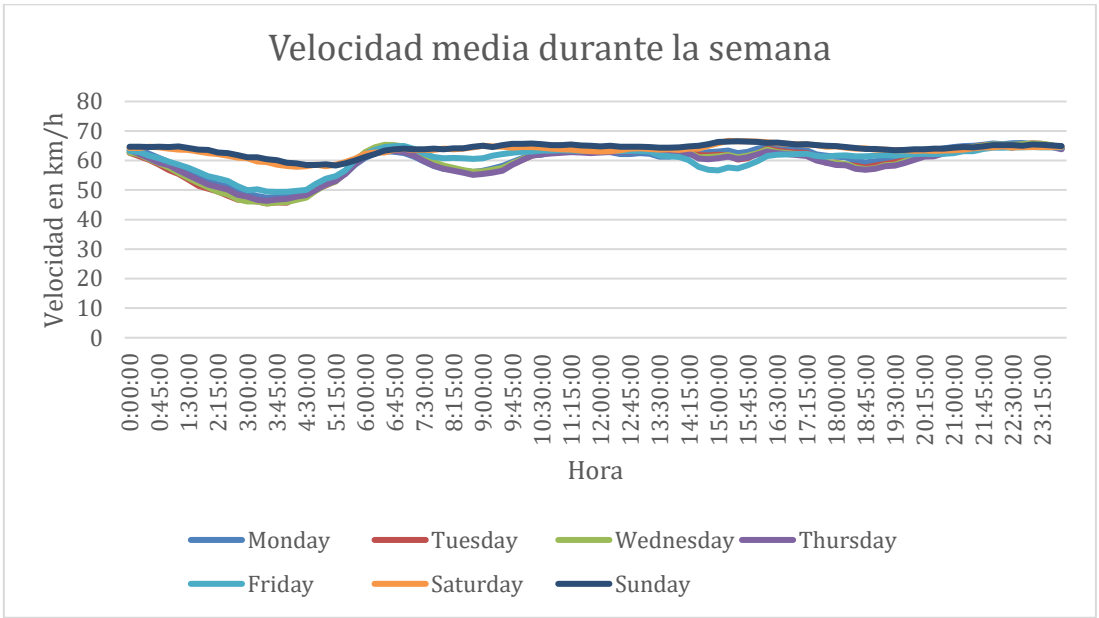
En la siguiente gráfica se puede ver claramente como durante la semana, el tráfico es prácticamente igual y la intensidad baja durante el fin de semana:



Gráfica 4. Intensidad del tráfico durante las horas de cada día

En la Gráfica 4 se nota la diferencia de intensidad de tráfico según la hora. Sumando todos los datos del año y calculando la media, vemos que la mayor cantidad de tráfico ocurre los viernes alrededor de las 15:00h.

Aprovechando que los datos proporcionados contienen también la velocidad media para la autopista de circunvalación M-30 se calcula la siguiente gráfica con la velocidad media:



Gráfica 5. Velocidad media durante la semana sobre la autopista M-30

Los datos de velocidad media se pueden utilizar en los cálculos para optimizar la ruta en los siguientes apartados del proyecto.

Para mejorar la visualización de los mapas de calor, en lugar de utilizar *Folium*, se usa la versión gratuita de *Microsoft Power BI*. Empleando los mismos datos que en las figuras anteriores, se obtiene lo siguiente:

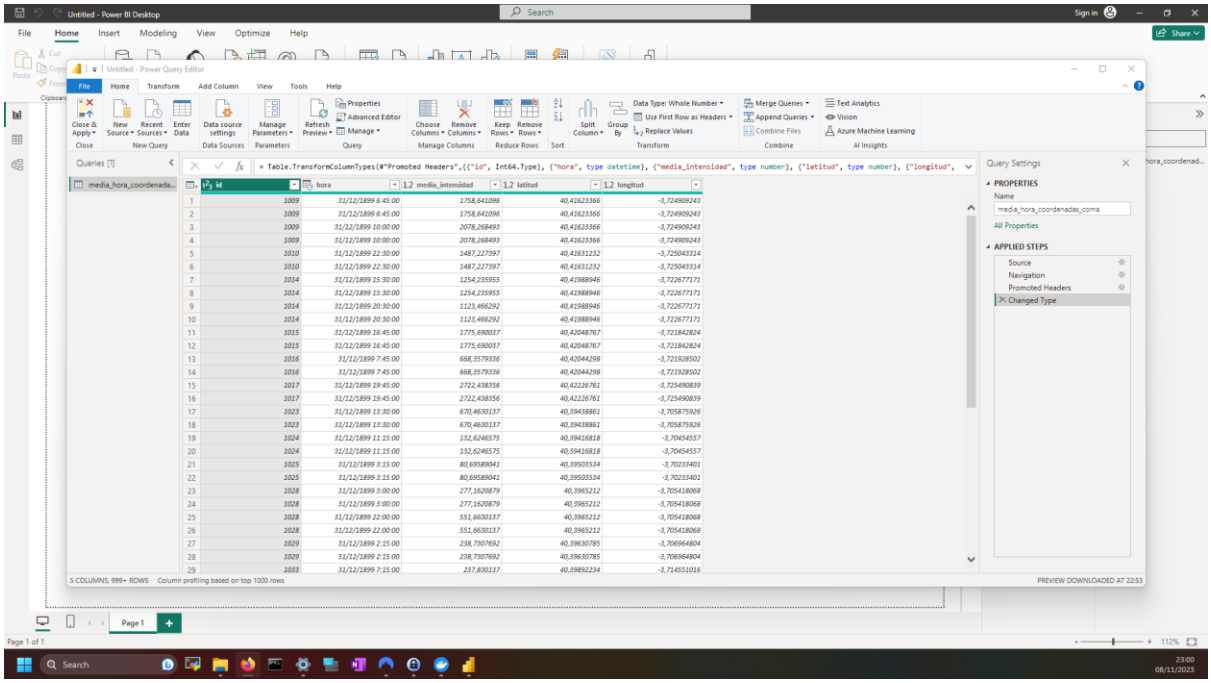


Figura 12. Microsoft Power BI con los datos cargados

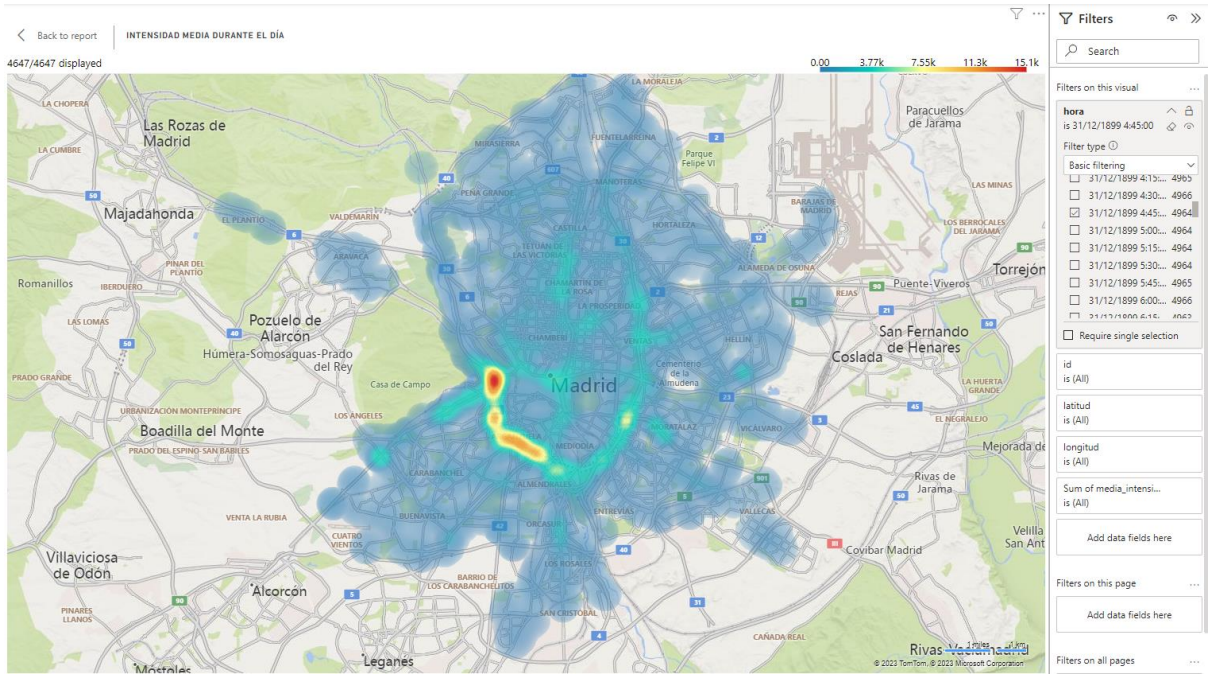


Figura 13. Intensidad media de tráfico en Power BI a las 04:45h

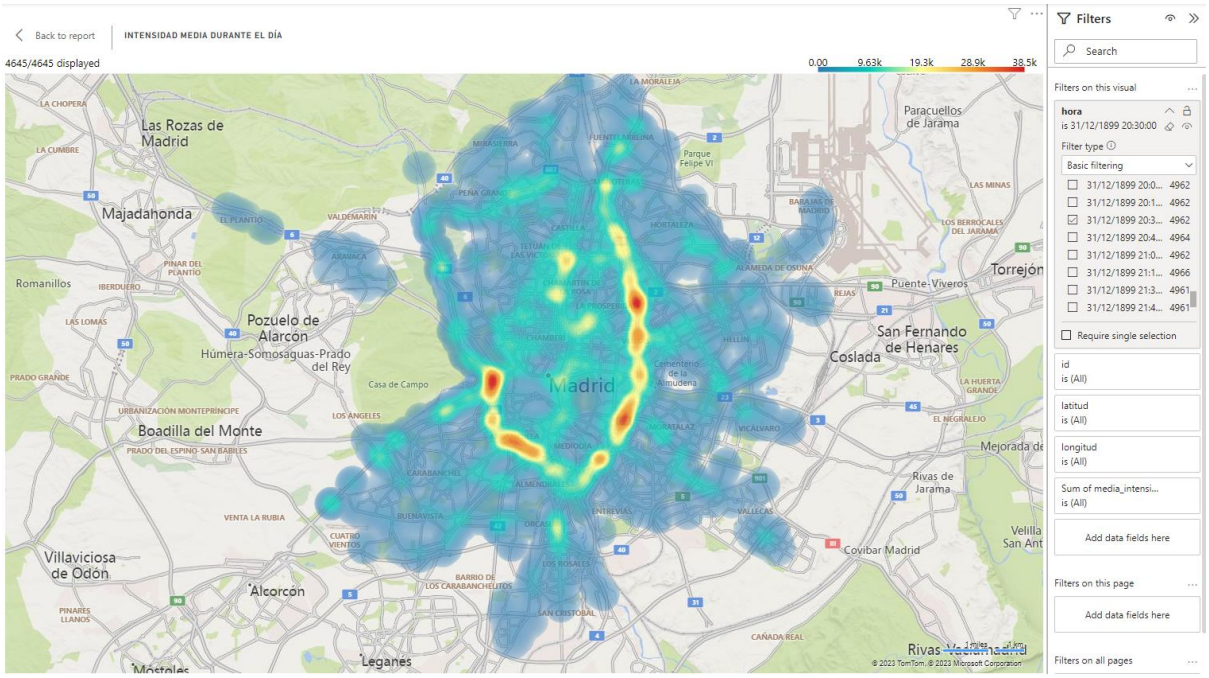


Figura 14. Intensidad media de tráfico en Power BI a las 20:30h

A pesar de ver todo el mapa a la vez, se nota una diferencia de intensidad, pudiéndose observar zonas dónde el tráfico cambia. Ampliando el mapa las diferencias son más notables todavía.

En el video adjunto se puede ver la evolución del mapa de calor, utilizando todos los datos del año. Para preparar los datos del video, se condensaron los 12 meses, en un solo día, manteniendo los datos para cada punto. De esta forma, para cada sensor hay datos con la intensidad de vehículos por hora cada 15 minutos. En el mapa se muestran todos los sensores a la vez en intervalos de 15 minutos, cada 0,5 segundos del video equivalen a 15 minutos:

Mapa de calor
intervalos de 15 min

Durante este análisis, al usar ficheros únicos que ocupaban 8-10 GB, según la cantidad de columnas, se ha observado que todos los datos se cargan en la memoria *RAM* del sistema, el uso de *CPU* y Disco no era intensivo. Las operaciones se realizaban directamente sobre la memoria *RAM*:

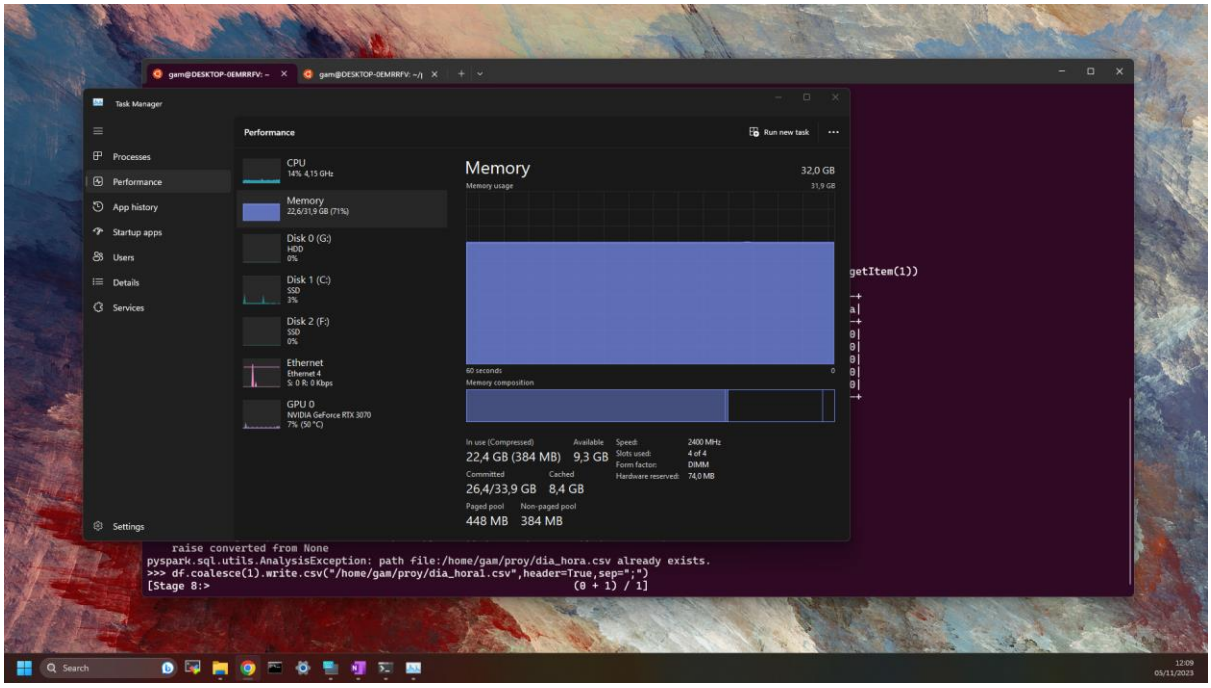


Figura 15. Rendimiento del sistema durante la exportación a un fichero .CSV del dataframe con PySpark ejecutándose sobre WSL

Tras ver la evolución de los datos y obtener los valores medios con la hora y las ubicaciones de los sensores, se pueden usar estos valores para las ubicaciones dónde no tenemos datos. De esta forma, se podrá realizar un análisis completo de todos los caminos, imputando valores cercanos a la realidad.

3.4. Algoritmos de *machine learning* e inteligencia artificial aplicados a enrutamiento de tráfico

Para poder predecir el tráfico y mejorar el enrutamiento para evitar la congestión hay que combinar varias tareas de procesado de datos. Por un lado, se necesita un mapa con los nodos y las intersecciones, dónde todas las ubicaciones y puntos de interés están indexadas. Se necesitan los datos con la intensidad de tráfico en cada camino, información sobre las carreteras y el sentido del tráfico. Una vez estén preparados estos parámetros se tienen que integrar en un único conjunto de datos para ser procesados y analizados con modelos de predicción de tráfico y enrutamiento.

3.4.1 Preparación y colección de datos

Para obtener un mapa con todos los nodos, información detallada sobre los posibles caminos y un entorno visual se utiliza *OSMnx*, una herramienta desarrollada en Python que utiliza los mapas de código abierto de *Open Street Map*. La aplicación se puede configurar en Docker como un contenedor con todas las dependencias instaladas con entorno web a través de la aplicación *Jupyter Notebooks*.

Al tener acceso a los mapas de *Open Street Map*, se puede obtener el grafo de nodos e intersecciones de una zona sólo con realizar una consulta. Además, *OSMnx* viene integrado con *NetworkX*, un conjunto de librerías que permiten analizar y realizar varias operaciones sobre los caminos.

Para crear un mapa con los nodos, inicialmente se descargaron los datos directamente de *Open Street Map* para la ciudad de Madrid y se analizaron estos datos con el programa *QGIS*:

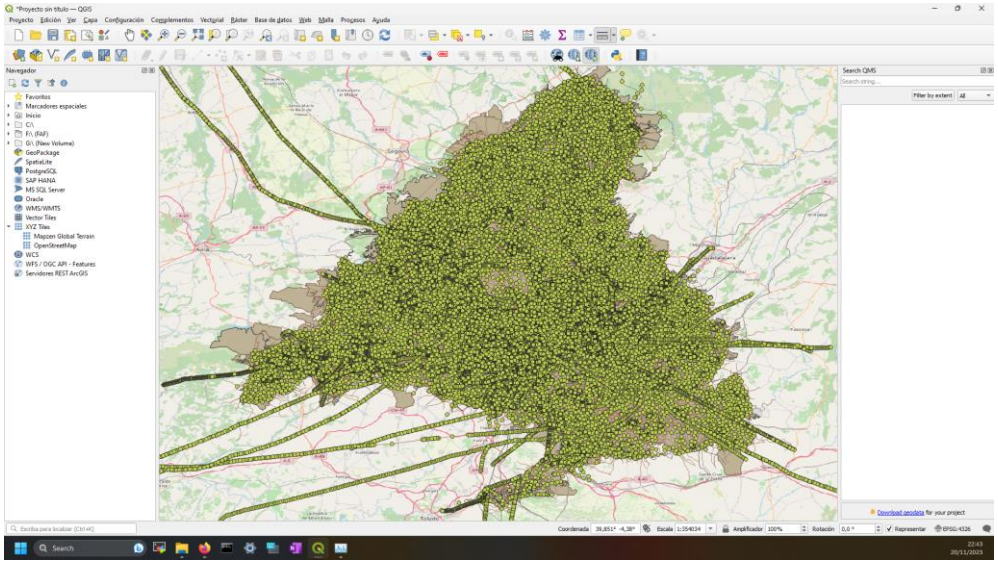


Figura 16. Visualización nodos Open Street Map en QGIS

Como se puede ver en la Figura 17, hay demasiados puntos en *QGIS*, sólo se muestran las intersecciones, sin tener información detallada sobre los caminos. Pero con *OSMnx*, podemos ver los nodos con las posibles conexiones entre ellos directamente, simplificando la visualización y la gestión de los datos:

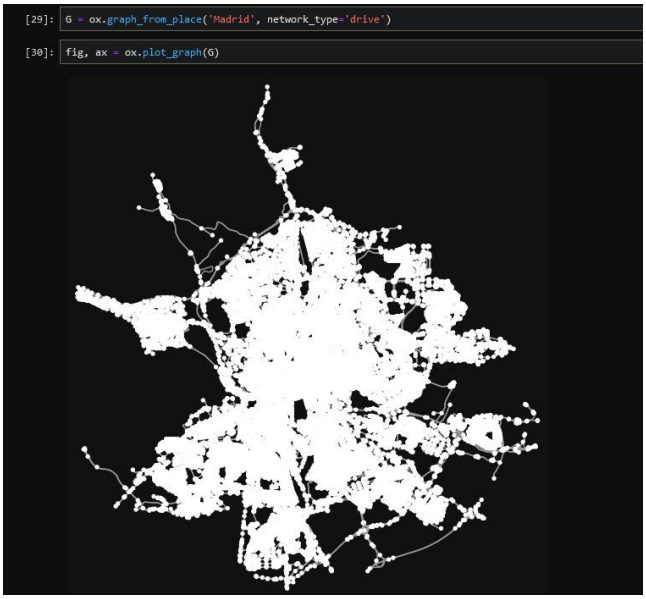


Figura 17. Mapa con las intersecciones y caminos de Madrid en OSMnx

Tras obtener el mapa, convertimos los datos para que sean usables en *GeoDataFrames*, usando la estructura *MultiDiGraph*. En *GeoDataFrames* se pueden procesar los datos topológicos, con los datos del tráfico utilizados en los apartados anteriores.

Para no procesar todos los nodos de Madrid y simplificar el análisis, a partir de ahora se utilizarán sólo los datos del barrio Peñagrande (Figura 19).

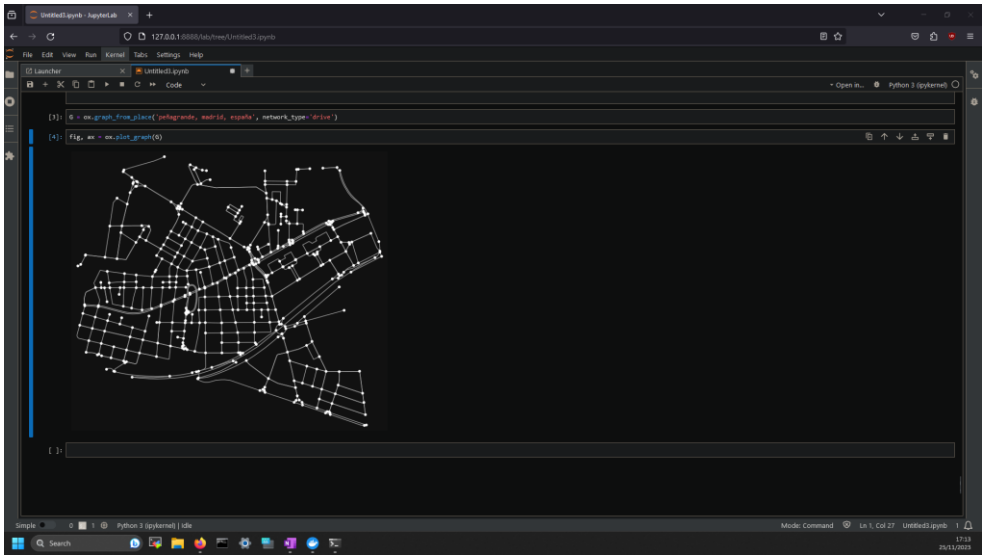


Figura 18. Mapa de nodos en OSMnx del barrio Peñagrande, noroeste de Madrid

Con *OSMnx* se calcula la topología para obtener las intersecciones (*nodes*) y los distintos caminos (*edges*):

	osmid						
21473145	40.473724	-3.725051	3	NaN	NaN	POINT (-3.72505 40.47372)
21682639	40.472810	-3.733468	3	NaN	NaN	POINT (-3.73347 40.47281)
21682643	40.473808	-3.725278	3	27	motorway_junction	POINT (-3.72528 40.47381)
25693384	40.482875	-3.714358	3	NaN	NaN	POINT (-3.71436 40.48288)
25693388	40.482305	-3.717725	3	NaN	NaN	POINT (-3.71773 40.48230)

```
[10]: gdf_margenes.head()
```

[10]:

u	v	key	osmid	oneway	lanes	highway	maxspeed	reversed	length	geometry	ref	name	tunnel	junction	bridge
21682643	**339984133**	**0**	30739560	True	1	motorway_link	70	False	153.339	LINESTRING (-3.72528 40.47381, -3.72562 40.473...	NaN	NaN	NaN	NaN	NaN
	21682639	**0**	[5778602, 33897922, 24049341]	True	4	motorway	90	False	712.436	LINESTRING (-3.72528 40.47381, -3.72545 40.473...	M-30	Calle 30	yes	NaN	NaN
25693388	**2686683465**	**0**	5629759	True	1	residential	NaN	False	22.290	LINESTRING (-3.71773 40.48230, -3.71763 40.482...	NaN	Avenida del Cardenal Herrera Oria	NaN	NaN	NaN
	25693440	**0**	71591654	True	2	secondary	50	False	30.802	LINESTRING (-3.71773 40.48230, -3.71748 40.482...	NaN	Avenida del Cardenal Herrera Oria	NaN	NaN	NaN
25693393	**41867726**	**0**	1002825218	False	NaN	residential	NaN	False	8.075	LINESTRING (-3.73621 40.47715, -3.73619 40.47722)	NaN	NaN	NaN	NaN	NaN

```
[11]: topologia=ox.graph_from_gdfs(gdf_nodos,gdf_margenes, graph_attrs=G.graph)
[14]: print(len(topologia.nodes))
465
[16]: print(len(topologia.edges))
900
```

Figura 19. Formato datos graphml de OSMnx en Jupyter Notebooks

En la Figura 19 se puede ver que, al procesar los datos, se obtienen 465 intersecciones y 900 caminos. Además, se puede apreciar el formato de los datos exportados, dónde los caminos se denotan por el nodo de origen (U) y destino (V). Se incluye información detallada sobre el tipo de carretera, la velocidad máxima, la longitud del camino, las coordenadas y el sentido del tráfico. Estos parámetros se tienen en cuenta a la hora de calcular las posibles rutas disponibles

3.4.2 Integración datos

Para procesar los caminos más cortos, o de menos duración con los valores de intensidad de tráfico proporcionados por el ayuntamiento de Madrid y con los nodos de *OSMnx*, hay que añadir los datos de tráfico a todos los caminos. Para la integración de estos datos se tiene que realizar un procesado exhaustivo de la información.
Para integrar los datos de *OSMnx* con los datos sobre la intensidad del tráfico, exportamos las intersecciones, los caminos y los atributos de las carreteras con el comando:

```
ox.save_graphml(G, filepath='./data/mynetwork.graphml')
```

Obteniendo un fichero ***.XML*** con toda la información necesaria para poder calcular las rutas optimas. Al abrir el fichero observamos un resumen con las definiciones de cada atributo:

```
<?xml version='1.0' encoding='utf-8'?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <key id="d21" for="edge" attr.name="bridge" attr.type="string" />
  <key id="d20" for="edge" attr.name="junction" attr.type="string" />
  <key id="d19" for="edge" attr.name="tunnel" attr.type="string" />
  <key id="d18" for="edge" attr.name="name" attr.type="string" />
  <key id="d17" for="edge" attr.name="ref" attr.type="string" />
  <key id="d16" for="edge" attr.name="geometry" attr.type="string" />
  <key id="d15" for="edge" attr.name="length" attr.type="string" />
  <key id="d14" for="edge" attr.name="reversed" attr.type="string" />
  <key id="d13" for="edge" attr.name="maxspeed" attr.type="string" />
  <key id="d12" for="edge" attr.name="highway" attr.type="string" />
  <key id="d11" for="edge" attr.name="lanes" attr.type="string" />
  <key id="d10" for="edge" attr.name="oneway" attr.type="string" />
  <key id="d9" for="edge" attr.name="osmid" attr.type="string" />
  <key id="d8" for="node" attr.name="highway" attr.type="string" />
  <key id="d7" for="node" attr.name="ref" attr.type="string" />
  <key id="d6" for="node" attr.name="street_count" attr.type="string" />
  <key id="d5" for="node" attr.name="x" attr.type="string" />
  <key id="d4" for="node" attr.name="y" attr.type="string" />
  <key id="d3" for="graph" attr.name="simplified" attr.type="string" />
  <key id="d2" for="graph" attr.name="crs" attr.type="string" />
  <key id="d1" for="graph" attr.name="created_with" attr.type="string" />
  <key id="d0" for="graph" attr.name="created_date" attr.type="string" />
```

Figura 20. Definición de cada atributo utilizado en el fichero XML

En la Figura 20 podemos ver que el ID = D13 se utiliza para conocer la velocidad máxima a la que se puede circular en cada camino. Se puede modificar este parámetro para que cambie según la intensidad de tráfico que se ha utilizado en los apartados anteriores.

Ampliando la zona del barrio Peñagrande en *Power BI*, podemos observar que no todas las calles tienen un valor de intensidad de tráfico:

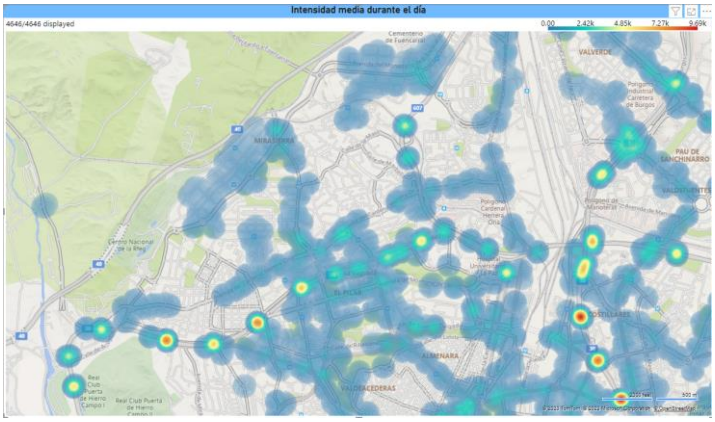


Figura 21. Barrio Peñagrande, Madrid, Intensidad media tráfico a las 09:00h

Como no hay valores de intensidad para todos los caminos, debido al número limitado de sensores, podemos calcular el valor de intensidad media del tráfico solamente en la zona del barrio y aplicar ese valor para los caminos que no tienen un sensor asignado. Para ello obtenemos las coordenadas del rectángulo que engloba al barrio Peñagrande:

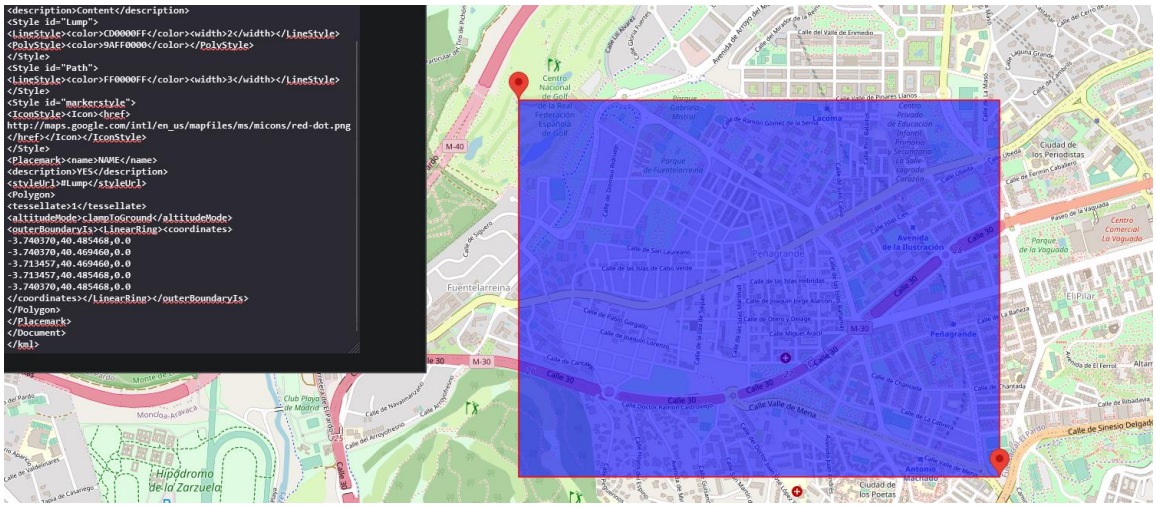


Figura 22. Coordenadas rectángulo datos intensidad tráfico

Con estas coordenadas se filtran los valores de intensidad media de tráfico durante todo el año por hora, obteniendo un fichero con 10710 valores de intensidad:

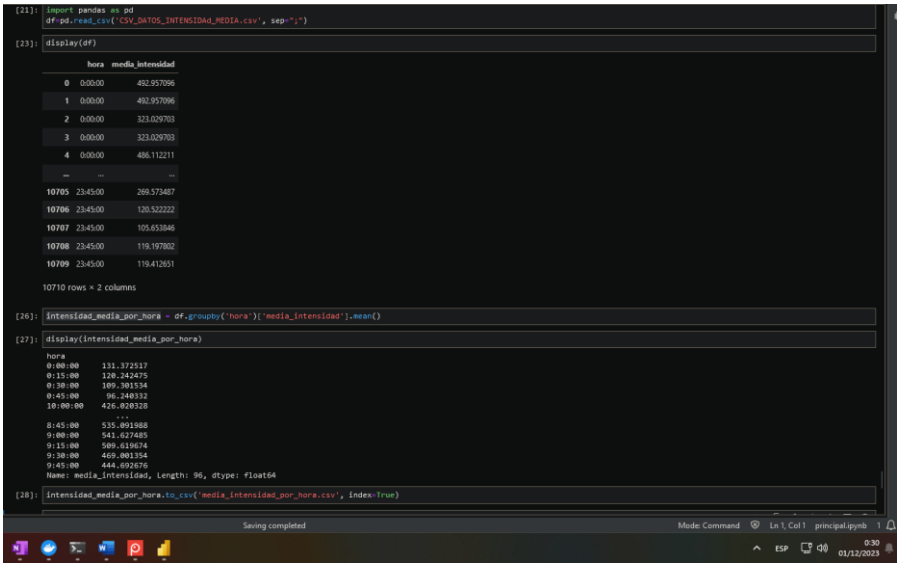


Figura 23. Se calcula el valor medio de la intensidad por hora para el rectángulo anterior

Con los datos anteriores, se puede modificar el valor de velocidad máxima para todos los caminos del fichero **.graphml** exportado desde *OSMnx*.

Para modificar los valores en intervalos de 6 horas, primero se ha buscado el valor de intensidad de tráfico media y el valor de intensidad máximo:

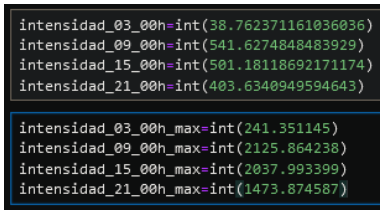


Figura 24. Valores medios y máximos de intensidad tráfico a las 03:00, 09:00, 15:00 y 21:00h barrio Peñagrande

Se crea un algoritmo para recorrer todos los caminos del fichero *network.graphml* y modificar todos los valores de velocidad media, según la intensidad de tráfico. Para cada nodo cercano a un sensor utilizado para medir la intensidad de tráfico. El algoritmo cambia la velocidad de ese camino y si no hay un valor de velocidad asignado, se fija uno a 50 km/h como valor por defecto.

Para cambiar los caminos que están cercanos a un sensor de tráfico multiplicamos la velocidad de ese camino por la intensidad detectada, dividida por la intensidad máxima a esa hora. De esta forma nos aseguramos de que la velocidad nueva siempre será menor o igual a la velocidad original. Si no hay ningún sensor cerca del camino, se utiliza el valor de intensidad media a esa hora.

A partir del grafo original de intersecciones y caminos, se crean 4 ficheros distintos con la velocidad máxima modificada, en intervalos de 6 horas. Se usarán estos datos para comprobar que los algoritmos para calcular la ruta más optima funcionan correctamente.

3.4.3 Optimización de enrutamiento con algoritmos y modelos de *machine learning*

Para optimizar las rutas de tráfico hay varios algoritmos existentes que tienen en cuenta el camino más corto o la ruta más rápido según los parámetros de los caminos.
Por ejemplo, el algoritmo de Dijkstra se utiliza para encontrar el camino más corto entre dos nodos. Este algoritmo se emplea en varios problemas de optimización de rutas, no sólo en tráfico vehicular sino también en enrutamiento de redes.

La propia herramienta *OSMnx* tiene integrada funciones que calculan la ruta más corta entre dos puntos en un mapa. Teniendo en cuenta los nodos de destino y origen, según el peso asignado el algoritmo calculará la ruta más corta o la más rápida.

Si no se asigna ningún peso, *OSMnx* calculará la ruta más corta utilizando el algoritmo de Dijkstra. Como pesos se pueden asignar los atributos de la Figura 22. Si el peso tiene valores negativos se utiliza el algoritmo de Bellman-Ford. De lo contrario se emplea el algoritmo de busqueda *A** (A asterisco), un algoritmo heurístico que encuentra la ruta más corta teniendo en cuenta el camino más corto y el coste desde el nodo actual hasta el nodo destino. El problema de este algoritmo es que requiere más recursos de computación al almacenar la

información de todos los futuros nodos vecinos. Si la información inicial no está procesada correctamente, la complejidad del modelo puedo crecer exponencialmente.

Además de algoritmos puramente matemáticos existen modelos inspirados en la naturaleza, a continuación, se detallan dos de ellos. [20]

3.4.3.1 Algoritmo de la colonia de hormigas

El algoritmo de la colonia de hormigas o *Ant Colony Optimization* en inglés, a partir de ahora ACO, utiliza probabilidades para elegir caminos, inspirándose en el comportamiento de las hormigas para buscar comida utilizando feromonas. Este algoritmo fue diseñado por Marco Dorigo y publicado en 1997 en el artículo "*Ant colony system: a cooperative learning approach to the traveling salesman problem*". [21]

Funcionamiento paso a paso del algoritmo:

1. Inicialización de los parámetros utilizados en el modelo, a cada nodo se le asigna una concentración aleatoria de feromonas.
2. Exploración de los caminos que hay alrededor del origen.
3. Si encuentra una ruta más cercana al destino, aumenta la cantidad de feromonas
4. Las hormigas usarán la ruta que más feromonas tenga, por lo tanto, la ruta más corta será la ruta más utilizada

Podemos aplicar este algoritmo al proyecto para calcular la ruta más corta según la intensidad del tráfico, utilizando los datos del apartado 3.4.2. De esta forma obtenemos lo siguiente:

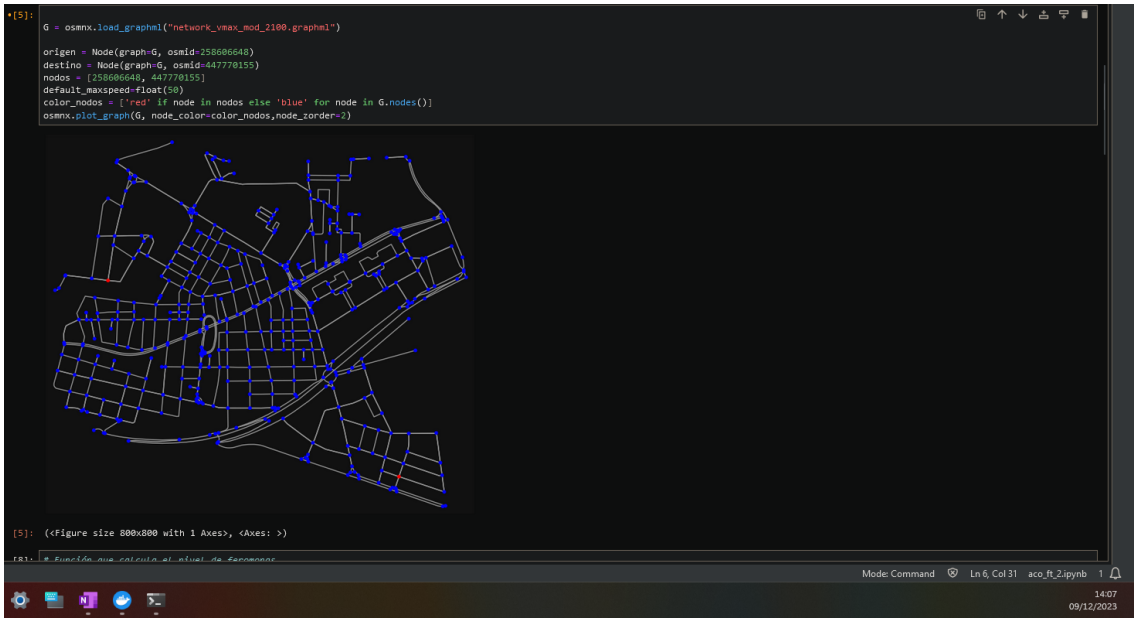


Figura 25. Configuración inicial del modelo

En la Figura 25 se cargan los datos procesados de *OSMnx* y se definen dos nodos de origen y destino, marcados con rojo en el mapa, elegidos aleatoriamente y la velocidad por defecto, en caso de que un camino no tenga asignado el valor.

La fórmula para calcular la concentración de feromonas es:

$$feromonas = concentración^{\alpha \cdot (\frac{1}{distancia})^{\beta}}$$

Siendo α=3 y β=3 en este caso, concentración y distancia dependerán de los nodos vecinos al origen:

```
# Función que calcula el nivel de feromonas
def feromonas(concentracion, distancia, alfa, beta):
    return concentracion ** alpha * ((1/distancia)) ** beta
```

Figura 26. Función utilizada durante el algoritmo ACO para calcular el nivel de feromonas

Al ejecutar el algoritmo se puede obser ver que realiza las 500 iteraciones correctamente, en menos de un segundo. Esto puede ser un factor importante a la hora de integrar el algoritmo con una aplicación que debe funcionar en tiempo real.

```
100% 500/500 [00:00<00:00, 898.16it/s]
```

Figura 27. Ejecución del algoritmo ACO

En el anexo del proyecto se puede observar el código completo del algoritmo, incluyendo la modificación para incluir la velocidad máxima calculada en el apartado anterior de cada camino. De esta forma tenemos en cuenta la intensidad de tráfico a la hora de calcular los caminos. Cuando hay menos intensidad de tráfico, la velocidad estará más cercana al límite de velocidad legal, por lo tanto, incentivamos con más feromonas a los caminos por dónde se puede viajar más rápido. Ejecutando el algoritmo se obtiene lo siguiente:

```
mejor_ruta = max(rutas_conocidas, key=rutas_conocidas.get)
ruta = list(mejor_ruta)
print("Longitud de la ruta:", cost(G, ruta),"metros")
osmnx.plot_graph_route(G, ruta, route_color='red')
```

```
Longitud de la ruta: 3148.323 metros
```

Figura 28. Mejor ruta obtenida con el algoritmo de la colonia de hormigas

Usando la función de *Networkx* con el algoritmo de Dijkstra, obtenemos el siguiente camino más corto, sin tener en cuenta la intensidad de tráfico:

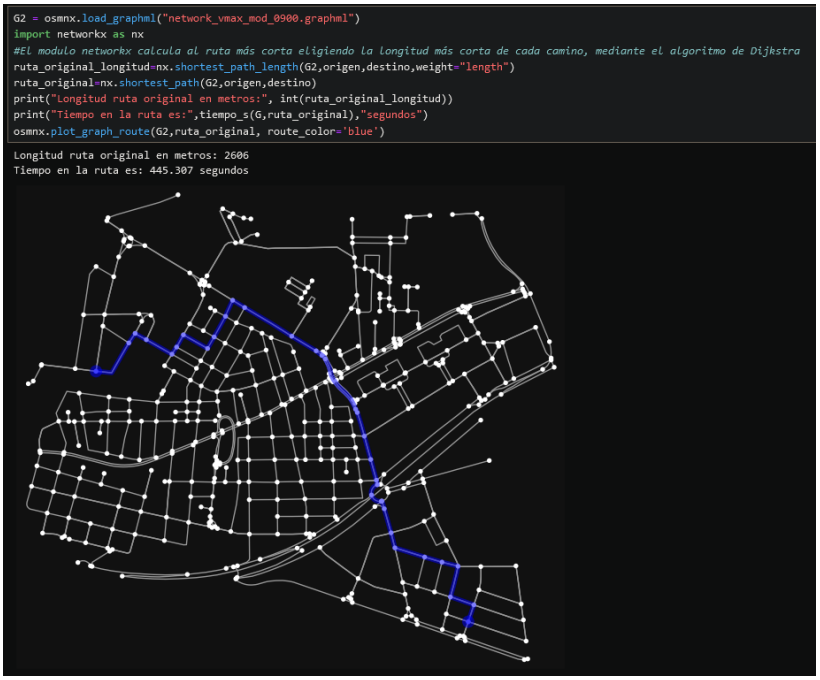


Figura 29. Ruta más corta obtenida con el algoritmo de Dijkstra

Con el algoritmo de la colonia de hormigas, encontramos otra forma de calcular el camino más corto mediante probabilidades aleatorias, teniendo en cuenta la velocidad máxima en cada camino. En el siguiente subapartado se verá otro algoritmo que optimiza las rutas utilizando la ruta más corta.

3.4.3.2 Algoritmo de la colonia de abejas

El algoritmo de la colonia de abejas o *Artificial Bee Colony* en inglés, a partir de ahora ABC, utiliza el comportamiento de un enjambre de abejas para encontrar el camino más corto a las fuentes de polen. El algoritmo fue propuesto por Dervis Karaboga en 2005 en el artículo "*An idea based on honey bee swarm for numerical optimization*" [22].

En el algoritmo se utilizan tres tipos de abejas, y cada una realiza una función distinta:

- Abejas empleadas, la mitad del total de las abejas buscan nuevos caminos vecinos para llegar a la fuente de alimentación según la ruta más corta. De esta forma se define una ruta optima que tenga menos coste.
- Abejas observadoras, se calculan probabilidades según la longitud de las rutas encontradas por las abejas empleadas. Las abejas observadoras calculan nuevas rutas según estas probabilidades, si la ruta final no es mejor, se buscan nuevas rutas. Si se encuentra una ruta mejor, se actualiza el camino más corto.
- Abejas exploradoras, si una fuente de alimento no se tiene en cuenta estas abejas exploran aleatoriamente las fuentes de alimento no visitadas. Se añade el coste de la ruta para ser evaluado.

De esta forma, se elige la ruta más corta teniendo en cuenta las rutas actuales, probando nuevos caminos y empleando caminos que no se utilizan. Para implementar este algoritmo en Python se utilizan varias librerías, a continuación, se adjuntan capturas del código y de los resultados:

```
import osmnx
from smart_mobility_utilities.common import Node, cost, randomized_search
from smart_mobility_utilities.children import get_children
import networkx as nx
from tqdm.notebook import tqdm
import random
import matplotlib.pyplot as plt
```

***Figura 30. Librerías utilizadas en el algoritmo ABC** [23]*

Las librerías fueron desarrolladas por los investigadores en *Smart Mobility* y *Machine Learning* Yinan Wang y Alaa Khamis para optimizar los cálculos de *OSMnx* en problemas de enrutamiento, planificación y coste de rutas.
Cargamos el grafo con los datos de tráfico a las 21:00h, usando los mismos puntos de origen y destino que en el apartado anterior, para comprobar los resultados:

```
G = osmnx.load_graphml("network_vmax_mod_2100.graphml")

origen = Node(graph=G, osmid=258606648)
destino = Node(graph=G, osmid=447770155)
nodos = [258606648, 447770155]

color_nodos = ['red' if node in nodos else 'blue' for node in G.nodes()]
osmnx.plot_graph(G, node_color=color_nodos,node_zorder=2)
```

Figura 31. Carga inicial del fichero graphml, definición de origen y destino

En el anexo del proyecto se encuentra el código completo del algoritmo. Como el algoritmo necesita realizar más cálculos por iteración, sólo se lanzan 50 iteraciones y se empieza con 15 abejas. Tras ejecutar el código, vemos que tarda más tiempo que el algoritmo ACO en realizar los cálculos, obteniendo 12.25 iteraciones/segundo:

Figura 32. Resultados del algoritmo ABC

Como sólo tiene en cuenta el coste de la ruta, obtenemos una longitud más cercana a la longitud optima obtenida con *OSMnx*. En la Figura 33 se puede ver como mejora la longitud total de la ruta, al avanzar en iteraciones:

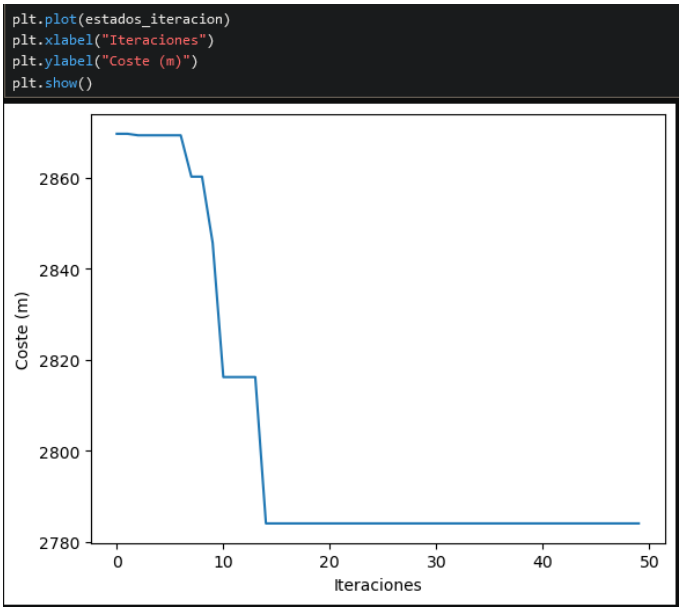


Figura 33. Evolución de la longitud de la ruta

En este caso sólo necesita 14 iteraciones para encontrar la ruta más optima, esto puede ser porque la ruta es demasiado corta y no hay suficiente variación, no encuentra suficientes posibles caminos adicionales. Como en este caso, converge a la solución de forma lineal, la evolución del algoritmo es gradual y no hay una oscilación entre los resultados obtenidos.

3.4.4 Evaluación de los algoritmos utilizados

Para comprobar la efectividad de los algoritmos utilizados en los apartados anteriores, en la librería ***smart_mobility_utilities.common*** se crea la función tiempo_s con el siguiente código:

```python
def tiempo_s(G, route):
    tiempo_total = 0
    for u, v in zip(route, route[1:]):
        datos_camino=G[u][v][0]
        longitud=datos_camino["length"]
        if "maxspeed" in datos_camino:
            velocidad_kmh = float(datos_camino["maxspeed"])
            velocidad_ms = velocidad_kmh/3.6
        else:
            velocidad_ms= 50/3.6
        tiempo_seg=longitud/velocidad_ms if velocidad_ms > 0 else 0
        tiempo_total+=tiempo_seg
    return round(tiempo_total,1)
```

Figura 34. Función que calcula la duración de la ruta

Con la ruta calculada, obtiene la longitud y la velocidad de cada camino, convierte la velocidad en metros por segundo y obtiene el tiempo empleado en viajar la ruta en segundos. Si el camino no tiene una velocidad asignada, se le asigna una velocidad por defecto de 50 km/h.

Aplicando el algoritmo ACO teniendo en cuenta la intensidad de tráfico para la misma ruta que en los apartados anteriores obtenemos los siguientes resultados:

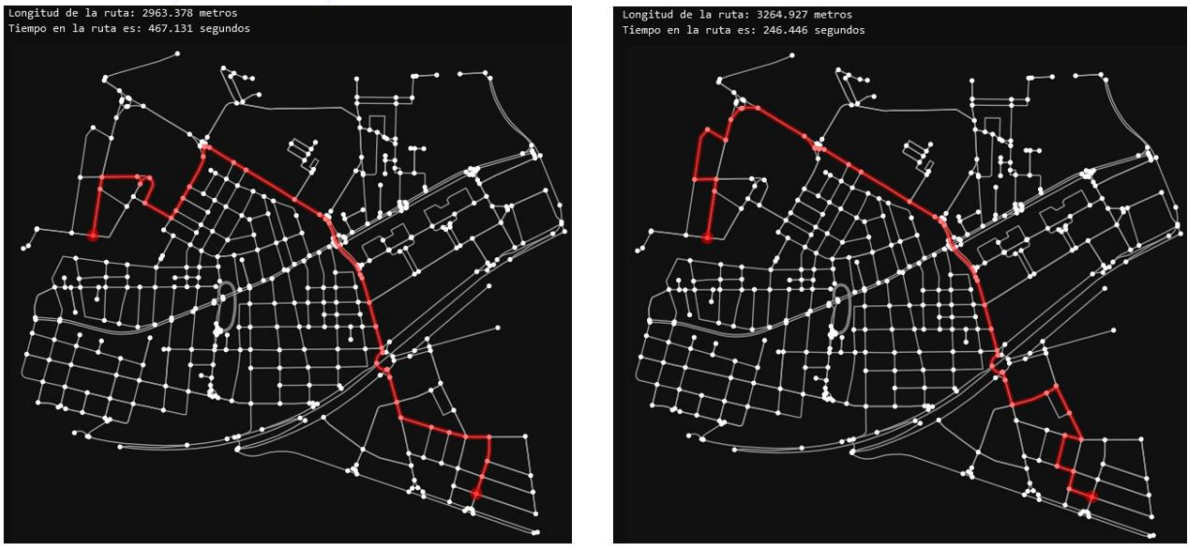


Figura 35. Longitud y duración de trayecto con algoritmo ACO

Al ejecutar de nuevo la función aleatoria que distribuye las feromonas iniciales, se obtiene una ruta totalmente distinta pero que es más efectiva, ya que acorta la duración del trayecto.

Calculando el mismo trayecto con el algoritmo de Dijkstra del módulo *NetworkX* , obtenemos la ruta más corta, pero la duración del trayecto es más larga:

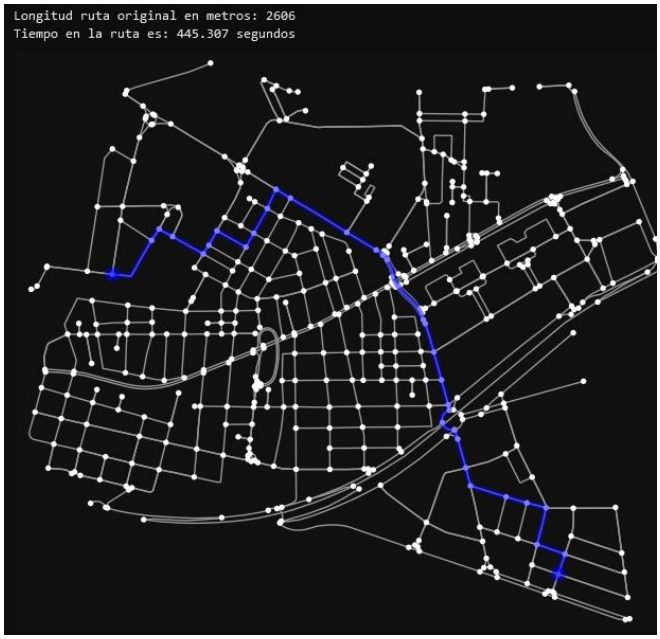


Figura 36. Cálculo de la misma ruta con algoritmo de Dijkstra

Por otro lado, aplicando el algoritmo ABC, teniendo en cuenta sólo la longitud de los caminos, obtenemos lo siguiente:

Figura 37. Longitud y duración de trayecto con algoritmo ABC

Igual que en el caso del algoritmo ACO, al usar probabilidades para elegir los caminos, cada vez que se ejecuta se obtiene un resultado distinto.
Usando el algoritmo de Dijkstra, sin haber modificado los valores de velocidad media, obtenemos la siguiente duración:

Figura 38, Cálculo de la misma ruta con algoritmo de Dijkstra sin modificar la velocidad

De este análisis se puede concluir que el algoritmo ABC se puede emplear para calcular la ruta óptima a nivel de longitud, pero tarda más tiempo en procesar todas las iteraciones.
Por otro lado, el algoritmo ACO es mucho más ágil a la hora de procesar los caminos, pero no obtiene siempre la ruta más corta. Sí que tiene más utilidad para calcular la ruta teniendo otros parámetros en cuenta, como la intensidad de tráfico.

Por último, para comprobar la efectividad de los dos modelos, se amplía el análisis a toda la ciudad de Madrid. Al ampliar el mapa, se modifica el destino para tener una ruta más larga:

```
G = osmnx.load_graphml("network_madrid.graphml")

origen = Node(graph=G, osmid=258606648)
destino = Node(graph=G, osmid=562786877)
nodos = [258606648, 562786877]
color_nodos = ['red' if node in nodos else 'blue' for node in G.nodes()]
tamaño_nodos = [50 if node in nodos else 1 for node in G.nodes()]
osmnx.plot_graph(G, node_size=tamaño_nodos, node_color=color_nodos,node_zorder=2)
```

Figura 39. Nuevo origen y destino

Con el algoritmo ACO, se obtiene una longitud variable de 24.2 o 24.6 km y la duración del viaje es alrededor de 23 minutos.

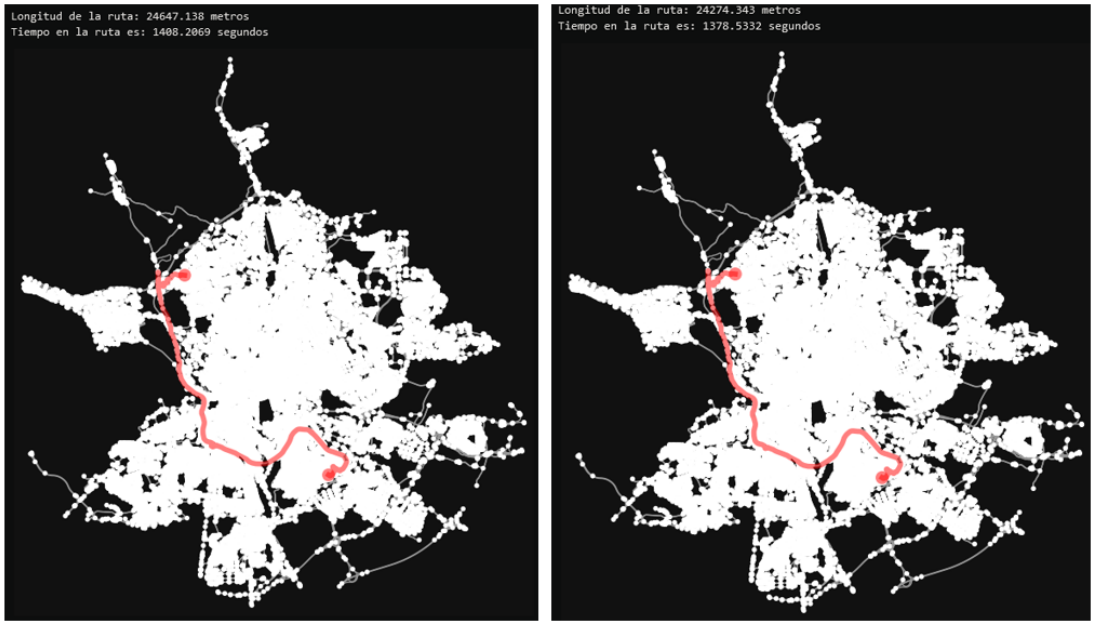


Figura 40. Algoritmo ACO, nueva ruta

Aplicando los algoritmos que sólo tienen en cuenta la longitud del trayecto, Dijkstra y ABC se obtiene:

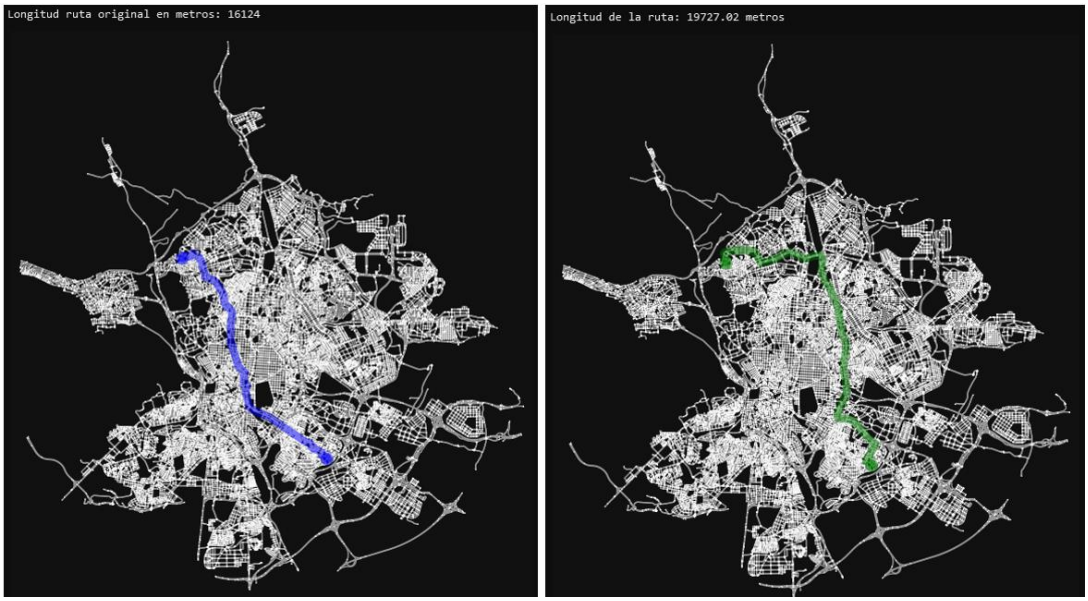


Figura 41. Algoritmo de Dijkstra(azul), 16.1 km y modelo ABC(verde) 19.7 km

En este caso, el algoritmo ABC no optimiza correctamente el camino, esto es porque al ejecutar el algoritmo, sólo se tuvieron en cuenta 50 iteraciones. Como el algoritmo ABC necesita realizar más cálculos, ha tardado 52 minutos en calcular las 50 iteraciones:

Figura 42. Iteraciones algoritmo ABC, para la nueva ruta

Ampliando la cantidad de abejas e iteraciones, vemos que no es viable utilizar este algoritmo:

Figura 43. Modelo ABC con 200 iteraciones y 100 abejas

El modelo tardaría más de 18 horas, aunque no se utilicen todos los recursos de cómputo se podría paralelizar aún más la ejecución, pero aun así no es una opción ya que la ruta debería estar calculada en segundos.

Con el algoritmo ACO ocurre un problema parecido, al ampliar el número de cálculos e iteraciones, al principio tarda mucho más tiempo en obtener los caminos y al avanzar y acortar el camino, avanza de forma acelerada:

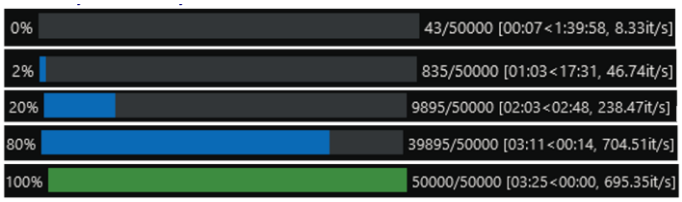


Figura 44. Evolución de las iteraciones por segundo del modelo ACO

Al final, cuando llega a 100% se observa que el número medio de iteraciones por segundo se aproxima al número iteraciones obtenido cuando la cantidad de datos era menor.

Con esto se puede concluir que los dos modelos tienen utilidad, según el caso y según la cantidad de datos que hay que analizar. Si la cantidad de datos es pequeña conviene utilizar ABC, pero si se amplía el foco de análisis, ACO tiene mejores prestaciones y además permite incluir otros parámetros de los caminos para optimizar aún más los resultados.

Como futuras mejoras se podría estudiar las posibles mejoras para agilizar los cálculos y maximizar el uso de los recursos de cómputo. El problema del algoritmo ABC es que tiene en cuenta todos los caminos a la vez, si estos no se usan, después de cada iteración tiene que volver a actualizar la distribución de comida de cada camino. Al aplicarlo a un problema con tantos caminos, aumenta demasiado la complejidad del sistema.

4. Conclusiones

En el análisis inicial de los datos mediante los mapas de calor se puede representar la congestión de forma sencilla para detectar los puntos con más intensidad de tráfico. Se puede observar que hay zonas dónde el tráfico es mucho más denso que la media. En estas zonas en concreto es fundamental analizar la causa del tráfico e intentar mejorar la congestión y las ineficiencias detectadas.

Inicialmente se utilizó Folium para dibujar los mapas de calor, pero esta herramienta no tiene en cuenta un peso, en este caso la intensidad de tráfico, sino que dibuja los colores según los datos que tiene alrededor. Es por ello por lo que, al ampliar el mapa, cambiaba el color de la intensidad de tráfico. Por otro lado, con Microsoft Power BI la representación es correcta y además permite ver los datos según las columnas de los datos, pudiendo separar por horas la visualización de los datos.Tras analizar la intensidad de tráfico, se vio que seguía un patrón cíclico que depende de la hora debido a las rutinas diarias.

Para los siguientes capítulos, para simplificar el análisis de los datos, la mejor opción fue calcular la media por hora de todos los datos, en cada ubicación. De esta forma se consigue una muestra de datos para cada punto a cada hora. La limitación de calcular la media es que no se tienen en cuenta los eventos especiales, dónde la intensidad de tráfico puede ser atípica. Al obtener los valores máximos de intensidad, en algunas zonas se puede ver que estos valores son múltiples veces mayores que la media calculada.

Una vez analizadas las zonas con más intensidad de tráfico, se analizaron modelos de enrutamiento en una zona en concreto, el barrio Peñagrande. Esto es porque se necesitaba una zona constante, pequeña para no complicar demasiado el análisis, pero con suficiente información, con distintos caminos e intersecciones. Tras analizar los algoritmos típicos utilizados para encontrar la ruta más corta, el mejor adaptado a la necesidad del proyecto era el algoritmo de Dijkstra. Para el algoritmo de Bellman-Ford no hay caminos con longitud negativa y en el caso del algoritmo A*, no busca necesariamente los caminos más cortos, por lo tanto, no se puede utilizar para optimizar la ruta más corta.

Tras revisar los algoritmos tradicionales, se aplicaron dos modelos de optimización inspirados en el comportamiento de las colonias de hormigas (modelo ACO) y en los enjambres de abejas (modelo ABC). Ambos modelos se entrenan con unas probabilidades aleatorias iniciales, funcionan mediante iteraciones y van aprendiendo los caminos más cortos entre el origen y el destino. Una vez encontrado el camino más corto buscan formas de acortarlo todavía más. Cada algoritmo tiene sus ventajas e inconvenientes, el algoritmo ACO encuentra rutas con facilidad y realiza los cálculos en poco tiempo, pero los resultados dependen de la distribución inicial aleatoria de los pesos asignados a cada camino y esto afecta a la longitud final calculada del trayecto. Modificando el peso aleatorio para que sea la intensidad de tráfico calculada con los datos del ayuntamiento de Madrid, se mejoró el comportamiento del

algoritmo cambiando el valor de la velocidad máxima de cada camino según la intensidad de tráfico.

En el caso del algoritmo ABC, al aplicarlo sobre el barrio Peñagrande se obtuvieron resultados muy buenos aproximándose a los obtenidos con el algoritmo de Dijkstra, pero al ampliar el mapa a toda la ciudad de Madrid, los caminos obtenidos eran más largos y tardaba mucho más que el algoritmo ACO en realizar los cálculos. Modificando los parámetros iniciales para añadir más abejas o ampliar la cantidad de comida disponible se multiplicó aún más el tiempo que tardaba en realizar los cálculos siendo inaplicable este algoritmo.

En resumen, se han estudiado los sistemas actuales Big Data, las herramientas más utilizadas según el tipo de aplicación y los proyectos corrientes de investigación y desarrollo. Tras este análisis, el proyecto se ha enfocado sobre algoritmos de optimización de rutas. Los modelos utilizados encuentran la ruta optima de formas distintas, el algoritmo ACO es más robusto y más ágil, pero el algoritmo ABC tiene más versatilidad aunque es menos práctico cuando aumenta la cantidad de posibles caminos a analizar.

5. Trabajos futuros

En el proyecto sólo se tuvo en cuenta la intensidad de tráfico como parámetro adicional a la hora de optimizar las rutas, si tuviésemos más información adicional de tráfico se podrían utilizar otros parámetros como la velocidad media del tráfico, cantidad de accidentes, tipos de vehículos, rutas completas. Teniendo todos estos parámetros, se podrían aplicar metodologías *deep learning* que calculen la ruta más corta y rápida teniendo en cuenta varios factores a la vez.

Para optimizar aún más los resultados, se podrían combinar los algoritmos para mejorar los cálculos. Se podría automatizar el cálculo de la ruta, utilizando primero ACO para encontrar el camino más rápido teniendo en cuenta los parámetros de cada camino y luego fragmentar el mapa en pequeños sectores y crear un origen y un destino para cada sector y aplicar el algoritmo ABC para optimizar aún más la ruta. De esta forma se podría obtener la mejor ruta, en cada sector a la vez, paralelizando y agilizando los cálculos. En la Figura 45, se puede observar una posible implementación, dónde sólo se tendrían en cuenta los cuadrantes resaltados:

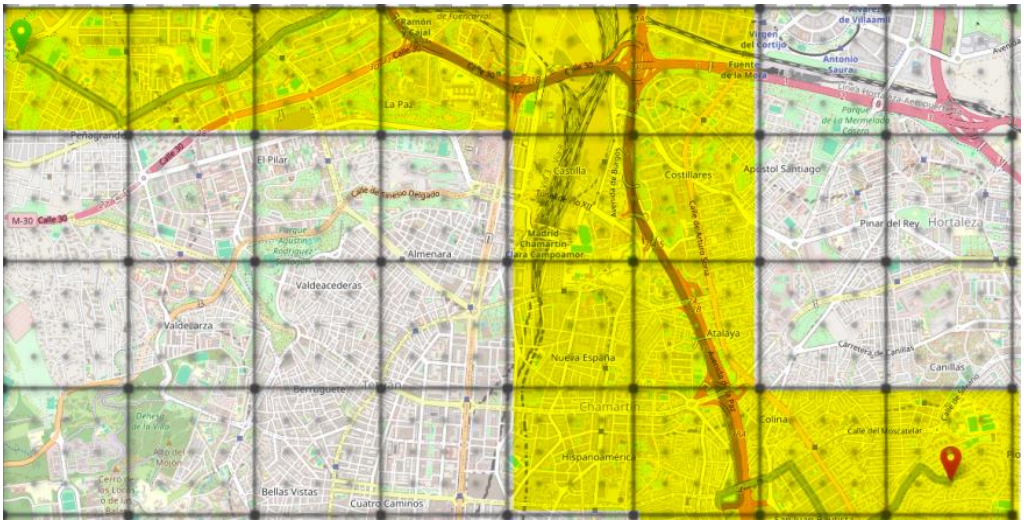


Figura 45. Posible mejora para paralelizar los cálculos, fragmentando la ruta inicial en pequeños bloques

Sólo se ha analizado una zona en concreto debido a que los datos de tráfico disponibles son limitados y para facilitar el trabajo. El objetivo final sería crear una aplicación que funcione sobre *Open Street Maps* o mediante la *API* de *Google Maps* para ofrecer las mejores rutas que funcione con datos en tiempo real, utilizando información de los usuarios para detectar cambios en el tráfico. Además, el procesado de los datos se debe realizar de forma automatizada, la selección de los datos, el cálculo de los valores medios y la zona dónde se tiene que calcular la ruta.

En este caso sólo se utilizan los datos históricos, pero al utilizar datos en tiempo real, mejoraría la veracidad de los resultados. Para poder utilizar datos en tiempo real habría que ampliar la infraestructura de cómputo, usar sensores, cámaras e información de usuarios. Toda la información aportada se debe tener en cuenta para poder calcular una probabilidad de ocupación y congestión. Estas variables se pueden aplicar a los algoritmos de optimización, teniendo en cuenta los datos históricos y la información en tiempo real sobre los vehículos.

En el *Big Data Value Forum* se presentaron varios proyectos financiados por el plan *Next GenerationEU* como *EMERALDS Horizon Europe* y *GREEN.DAT.AI* ,entre otros, que están implementando sensores e infraestructura *IoT* para mejorar la gestión del tráfico y la movilidad urbana en diversas ciudades europeas.

Mediante estos proyectos se dispondrá de una base de datos más amplia, con más sensores y mediciones. Teniendo más parámetros adicionales relacionados con el tráfico se obtendrán modelos más realistas y precisos. En lugar de utilizar probabilidades aleatorias en los modelos de *machine learning*, se podrán utilizar parámetros reales para representar la utilización real de las carreteras en tiempo real.

Al crear una aplicación usada por muchos usuarios, mediante inteligencia artificial y el aprendizaje continuo, la calidad de las rutas propuesta mejorará. Retroalimentando los algoritmos con las rutas propuestas anteriormente y validando las estimaciones con los datos reales, se podrán crear modelos que puedan predecir trayectos con alta precisión.

El trabajo aquí propuesto puede ser ampliado a otras aplicaciones en las que la gestión de rutas optimiza muchos procesos de mejora en las ciudades inteligentes, como por ejemplo en los sistemas de alquiler de bicicletas como el expuesto en [24] que ayudaría también a reducir el tráfico.

6. Glosario

5

5G
Quinta generación de redes móviles, con mejoras en la capacidad, el ancho de banda y la eficiencia energética y espectral. 15

A

API
Application Programming Interface, permite la comunicación entre programas mediante peticiones. 13, 15, 43

B

Big Data
Conjuntos de datos que son tan grandes, rápidos o complejos que resultan difíciles de procesar utilizando métodos de procesamiento de datos tradicionales. 1, 4, 5, 3, 4, 5, 9, 10, 11, 14, 15, 16, 42, 43

C

contenedores
Entorno que permite la ejecución de aplicaciones sin la necesidad de un sistema operativo. 5, 12, 13, 17, 18, 19

CPU
Central Processing Unit, se refiere al procesador del sistema. 24

CSV
Valores separados por comas, es un tipo de formato usado para almacenar datos. 16, 25

D

DataFrame
Estructura de datos formada por una tabla con filas y columnas. 18

DataNodes
Nodos dónde se almacena y gestiona la información. 12

datasets
Conjunto de datos, generalmente organizados. 10

deduplicar
Técnica utilizada para eliminar datos duplicados, reduciendo el tamaño ocupado por los datos y mejorando la eficiencia del almacenamiento. 10

deep learning
Aprendizaje automático mediante abstracciones para transformar y procesar los datos 42

Docker
Plataforma que permite virtualizar y gestionar contenedores. 17, 19, 25

F

Folium
Permite visualizar datos manipulados en Python, en este caso Pandas. 18, 20, 21, 23

G

Gigabytes
Unidad de almacenamiento equivalente a 10^9 bytes. 11
GPS
Global Positioning System o Sistema de Posicionamiento Global. 4, 5
graphml
Formato utilizado para representar grafos (nodos y caminos) utilizando el lenguaje XML. . 27, 28, 29, 30, 34

H

Hadoop
Apache Hadoop, software de código abierto que permite juntar varios recursos de cómputo para resolver problemas con grandes cantidades de datos en paralelo. 11, 12, 13, 49
hardware
Parte física o tangible de los sistemas informáticos. 4, 5, 10, 12
HDD
Disco Duro, almacenamiento de datos mediante discos metálicos y partes móviles. El almacenamiento es más estable pero con menos prestaciones. 9, 10, 11
HDFS
Sistema de archivos distribuido que almacena y gestiona grandes cantidades de datos en un entorno con alta disponibilidad. 11, 12, 13

I

IBM
Empresa de hardware y software dedicada a soluciones de cómputo, almacenamiento y redes. 14, 49
inteligencia artificial
Tareas de análisis y razonamiento mediante reglas, tomas de decisiones y reconocimiento de patrones parecidas a la inteligencia humana 4, 3, 4, 5, 8, 9, 11, 13, 14, 15, 25, 43
IoT
Internet of Things, se refiere a la integración de sensores o dispositivos conectados a internet 15, 43

J

Jupyter Notebooks
Aplicación que permite generar y compartir libretas con código, gráficas e imágenes. Permite integrar varios lenguajes de programación de alto nivel. 18, 19, 25, 27

L

last-mile
Se refiere al último tramo en la entrega de servicios o bienes. 15
Linux Ubuntu
Distribución de Linux, un sistema operativo. 17

M

machine learning
Aprendizaje automático mediante algoritmos y tareas. Permite analizar datos y realizar predicciones según los datos analizados. 1, 4, 5, 3, 4, 5, 8, 9, 13, 25, 30, 43

7. Bibliografía

[1] C. Tarazona Lizarraga, «Análisis de las Necesidades de una Smart City en el Marco de un Desarrollo Sostenible.,» *Master's Thesis, Universitat Oberta de Catalunya, Barcelona, Spain,* 2020.

[2] A. V. Gómez, «https://noticias.juridicas.com/,» Noticias Juridicas, 16 04 2020. [En línea]. Available: https://noticias.juridicas.com/conocimiento/articulos-doctrinales/15060-seguimiento-de-dispositivos-a-traves-de-geolocalizacion-y-covid-19/. [Último acceso: 20 10 2023].

[3] Guitart Roch Montserrat , «Turismo y sostenibilidad en una ciudad inteligente.,» 2020.

[4] T. White, Hadoop The Definitive Guide 4th Edition, 2009.

[5] T. M. K. A. N. R. &. T. H. Sara Landset, «https://journalofbigdata.springeropen.com/,» 2015. [En línea]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-015-0032-1. [Último acceso: 12 10 2023].

[6] E. Mete, «https://dzone.com,» 2019. [En línea]. Available: https://dzone.com/articles/example-of-etl-application-using-apache-spark-and. [Último acceso: 14 10 2023].

[7] IBM, «https://ibm.com,» IBM, [En línea]. Available: https://www.ibm.com/topics/machine-learning. [Último acceso: 2023 10 17].

[8] M. B. C. K. Yngvi Bjornsson, «http://webdocs.cs.ualberta.ca/,» [En línea]. Available: http://webdocs.cs.ualberta.ca/~chinook/project/legacy.html. [Último acceso: 2023 10 17].

[9] Berkeley School of Information, «https://ischoolonline.berkeley.edu,» 26 6 2020. [En línea]. Available: https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/. [Último acceso: 17 10 2023].

[10] Comisión Europea, «https://digital-strategy.ec.europa.eu/,» 19 06 2023. [En línea]. Available: https://digital-strategy.ec.europa.eu/es/policies/strategy-data. [Último acceso: 22 10 2023].

[11] Indra, «https://indracompany.com,» 2017. [En línea]. Available: https://www.indracompany.com/es/noticia/indra-lidera-proyecto-transforming-transport-utilizara-big-data-mejorar-movilidad-europa. [Último acceso: 20 10 2023].

[12] Union Europea, «https://transformingtransport.eu/,» 2020. [En línea]. Available: http://transformingtransport.eu/TT_Project%20Highlights.pdf. [Último acceso: 20 10 2023].

[13] Horizon Europe, «https://emeralds-horizon.eu/,» 2023. [En línea]. Available: https://emeralds-horizon.eu/. [Último acceso: 25 10 2023].

[14] Dirección General de Tráfico, «https://autonomousready.org/,» DGT, 2019. [En línea]. Available: https://autonomousready.org/. [Último acceso: 11 10 2023].

[15] GOOGLE, «https://mapsplatform.google.com/solutions/offer-efficient-routes/,» [En línea]. Available: https://mapsplatform.google.com/solutions/offer-efficient-routes/. [Último acceso: 11 10 2023].

[16] WAZE, «https://www.waze.com/company,» [En línea]. Available: https://www.waze.com/company. [Último acceso: 11 10 2023].

[17] FleetRoot, «https://www.fleetroot.com/,» 2022. [En línea]. Available: blog/how-to-do-route-optimization-across-large-fleets/. [Último acceso: 11 10 2023].

[18] Datos Madrid, «https://datos.madrid.es/portal/site/egob,» Ayuntamiento de Madrid, [En línea]. Available: https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=33cb30c367e78410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default. [Último acceso: 1 09 2023].

[19] Ayuntamiento de Madrid, «https://datos.madrid.es/FWProjects/egob/Catalogo/Transporte/Trafico/ficheros/Estructura_DS_Contenido_Trafico_Historico.pdf,» [En línea]. Available: https://datos.madrid.es/FWProjects/egob/Catalogo/Transporte/Trafico/ficheros/Estructura_DS_Contenido_Trafico_Historico.pdf. [Último acceso: 10 11 2023].

[20] A. K. a. Y. Wang, «https://smartmobilityalgorithms.github.io,» MIT, 2021. [En línea]. Available: https://smartmobilityalgorithms.github.io/book/content/SwarmIntelligenceAlgorithms/index.html. [Último acceso: 24 11 2023].

[21] M. Dorigo, «https://ieeexplore.ieee.org,» 1997. [En línea]. Available: https://ieeexplore.ieee.org/document/585892. [Último acceso: 1 12 2023].

[22] D. Karaboga, «https://www.semanticscholar.org,» 2005. [En línea]. Available: https://www.semanticscholar.org/paper/AN-IDEA-BASED-ON-HONEY-BEE-SWARM-FOR-NUMERICAL-Karabo%C4%9Fa/cf20e34a1402a115523910d2a4243929f6704db1. [Último acceso: 1 12 2023].

[23] W. Yinan y A. Khamis, 20 11 2021. [En línea]. Available: https://github.com/SmartMobilityAlgorithms/smart_mobility_utilities#readme. [Último acceso: 1 12 2023].

[24] J. L. M. Puyol and V. M. Baeza, «Bicycle Sharing System Using an IoT Network,» *2021 Global Congress on Electrical Engineering (GC-ElecEng), Valencia, Spain,* doi: 10.1109/GC-ElecEng52322.2021.9788465, pp. 131-135, 2021.

[25] J. C. McCallum, «Our World in Data,» 2022. [En línea]. Available: https://ourworldindata.org/grapher/historical-cost-of-computer-memory-and-storage. [Último acceso: 04 10 2023].

[26] T. H. Rutger Claes, «Ant Colony Optimization Applied to Route Planning Using Link Travel Time Predictions,» 2011.

8. Anexos

A continuación, se presentan los bloques de código utilizados durante el proyecto, cada bloque gris representa una ejecución por separado.

8.1. Separar los datos por día de la semana según la fecha

```
from pyspark.sql.functions import date_format,col,dayofweek
from pyspark.sql import SparkSession


df=df.read.option(“delimiter”,”;”).csv("datos_trafico_fusionados.csv", header=True,inferSchema=True)
df_dia_semana=df.withColumn(“día_semana.csv”,date_format(col(“fecha”),”EEEE”))
df_dia_semana.show()
```

```
>>> df_dia_semana.show()
+----+-------------------+---------+----------+---------+-----+----+-----+-------------------+-----------+
|  id|              fecha|tipo_elem|intensidad|ocupacion|carga|vmed|error|periodo_integracion|dia_semana|
+----+-------------------+---------+----------+---------+-----+----+-----+-------------------+-----------+
|1001|2022-07-01 00:00:00|      M30|       756|      2.0|    0|61.0|    N|                  5|     Friday|
|1001|2022-07-01 00:15:00|      M30|       684|      1.0|    0|67.0|    N|                  5|     Friday|
|1001|2022-07-01 00:30:00|      M30|       360|      0.0|    0|67.0|    N|                  5|     Friday|
|1001|2022-07-01 00:45:00|      M30|       336|      2.0|    0|44.0|    N|                  5|     Friday|
|1001|2022-07-01 01:00:00|      M30|       324|      1.0|    0|56.0|    N|                  5|     Friday|
|1001|2022-07-01 01:15:00|      M30|       300|      1.0|    0|62.0|    N|                  5|     Friday|
```

Para exportar los datos en un solo fichero hay que especificarlo, ya que spark crea varios ficheros pequeños para paralelizar la tarea:

```
df_dia_semana.coalesce(1).write.csv(datos_días_semana.csv_path,header=True)
```

8.2. Calcular media intensidad de tráfico según el día de la semana

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import date_format
from pyspark.sql.functions import col

spark =SparkSession.builder.appName("Media_dia_semana").getOrCreate()

df=df.read.csv("día_semana.csv ", header=True,inferSchema=True)
df=df.withColumn(“dia_semana”,date_format(col(“dia_semana”),”intensidad”))

df_agrupado=df.groupBy(“día_semana”)
df_media=df_agrupado.agg({“intensidad”:”media_intensidad”})
df_media.show()
```

```
+-----+-----------+------------------+
|   id|dia_semana |  media_intensidad|
+-----+-----------+------------------+
|10000|     Friday| 25.69187675070028|
|10000|     Monday|24.716867469879517|
|10000|   Saturday| 21.95945945945946|
|10000|     Sunday|22.189504373177844|
|10000|   Thursday|26.241176470588236|
+-----+-----------+------------------+
```

8.3. Agrupar las medias con las coordenadas

De esta forma, a cada id se le asignan las coordenadas, para luego utilizar los datos de intensidad de tráfico en los algoritmos de optimización:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import date_forma,col

spark =SparkSession.builder.appName("Combinar_CSV").getOrCreate()
df1=spark.read.option(“delimiter”,”;”).csv(“media_intensidad_por_dia_semana.csv”,header=True, inferSchema=True)
df2=spark.read.option(“delimiter”,”;”).csv(“ubicaciones_sensores.csv”,header=True, inferSchema=True)
```

```
>>> df1.show(5)
+-----+----------+------------------+
|   id|dia_semana|  media_intensidad|
+-----+----------+------------------+
|10000|    Friday| 25.691876750700280|
|10000|    Monday|24.716867469879517|
|10000|  Saturday| 21.959459459459460|
|10000|    Sunday|22.189504373177844|
|10000|  Thursday|26.241176470588236|
+-----+----------+------------------+
only showing top 5 rows

>>> df2.show(5)
+---------+--------+----+--------+--------------------+------------------+------------------+------------------+------------------+
|tipo_elem|distrito|  id|cod_cent|              nombre|              utm_x|              utm_y|          longitud|          latitud|
+---------+--------+----+--------+--------------------+------------------+------------------+------------------+------------------+
|      URB|       4|3840|    1001|Jose Ortega y TLC ..|441.615.343.346.657| 44.757.679.421.385| -36.883.232.754.856|404.305.018.691.825|
|      URB|       4|3841|    1002|Jose Ortega y TLC ..|441.705.882.339.595|447.576.968.733.175|-368.725.610.365.613|404.305.239.406.404|
|      URB|       1|3842|    1003|Pº Recoletos M-N ...|441.319.371.258.338|447.484.115.423.716| -36.917.268.449.043|404.221.320.929.972|
|      URB|       4|3843|    1004|Pº Recoletos K-S ...| 44.130.163.298.559| 44.747.637.275.141|-369.192.878.230.734|404.214.333.442.836|
|      URB|       4|3844|    1005|(AFOROS) Pº  Crew ...|441.605.765.071.902|447.613.213.924.728|-368.846.965.033.102|404.337.820.578.943|
+---------+--------+----+--------+--------------------+------------------+------------------+------------------+------------------+
only showing top 5 rows
```

Figura 46. Procesado de los datos

```
Intensidad_trafico_coordenadas=df1.join(df2.select(“id”,”longitud”,”latitud”),on=”id”,how=”left”)
```

Para no arrastrar información no necesaria, solo se mantienen las coordenadas, el id de cada sensor, el día de la semana y la media de intensidad de tráfico. Luego se crearán varios ficheros, con el día de la semana, por cada hora, agrupando todos los datos en un día en intervalos de 15 minutos. Para guardar los resultados, igual que en el caso anterior, hay que utilizar el comando **coalesce**.

```
>>> media_hora_coordenadas=media_por_hora.join(coordenadas.select("id","latitud","longitud"), on="id",how="left")
>>> media_hora_coordenadas.show(5)
+----+--------+------------------+---------------+---------------+
|  id|    hora| media_intensidad|         latitud|         longitud|
+----+--------+------------------+---------------+---------------+
|1009|06:45:00|1758.641095890411| 40.416233663841|-3.724909242726 42|
|1009|06:45:00|1758.641095890411| 40.416233663841|-3.724909242726 42|
|1009|10:00:00|2078.268493150685| 40.416233663841|-3.724909242726 42|
|1009|10:00:00|2078.268493150685| 40.416233663841|-3.724909242726 42|
|1010|22:30:00|1487.227397260274|40.4163123231285| -3.7250433138461|
+----+--------+------------------+---------------+---------------+
only showing top 5 rows

>>> media_hora_coordenadas.coalesce(1).write.csv("/home/gam/proy/media_hora_coordenadas",header=True,sep=";")
[Stage 26:>                                                         (0 + 16) / 66]
```

Figura 47. Exportación de datos

Inicialmente, al no saber cómo calcular el camino más corto, he intentado crear redes aleatorias, mediante las librerías ***NetworkX***:

```
import networkx as nx
import numpy as np
import osmnx as ox
import pandas as pd
```

```
G = nx.watts_strogatz_graph(n=100, k=5, p=0.1, seed=0)
```

```
nx.draw(G)
```

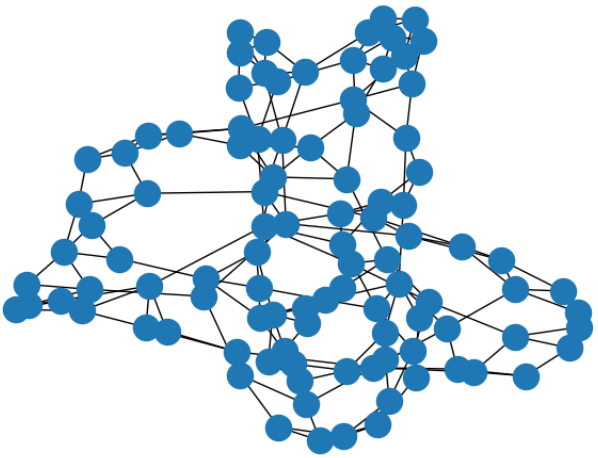


Figura 48. Pruebas iniciales con NetworkX, representación del modelo Watts-Strogatz

La función Watts-Strogatz es un grafo aleatorio circular que forma un clúster, con distancias y valores distintos para cada nodo.

```
#Calcular distancia entre nodos
camino1 = nx.shortest_path(G, source=0, target=50)
```

```
[0, 2, 17, 18, 20, 87, 50]
```

```
camino2 = nx.shortest_path(G, source=0, target=50, weight='distance')
```

```
[0, 99, 97, 34, 36, 37, 39, 40, 50]
```

Figura 49. Pruebas funciones NetworkX

Después de realizar estas pruebas para ver los posibles cálculos que se pueden realizar en Python, encontré los mapas de nodos de *OSMNx*.

8.4. Modificar la velocidad media de cada nodo según la intensidad de tráfico

```
import osmnx as ox
import pandas as pd
from shapely.geometry import Point  #Point se utiliza para definir y calcular las coordenadas de los caminos más cercanos

def busca_camino_mas_cercano(G,punto):
    camino_mas_cercano=ox.nearest_edges(G,punto.x,punto.y) #nearest_edges es la función de NetworkX que calcula las coordenadas de los caminos más cercanos
    return camino_mas_cercano
```

```
df=pd.read_csv("intensidad_madrid_0300.csv",sep=";")
G=ox.load_graphml("network_madrid.graphml")
display(df)
```

```
for index, row in df.iterrows():
    punto = Point(row['longitud'], row['latitud'])
    intensidad = float(row["media_intensidad"])
    camino = busca_camino_mas_cercano(G, punto)
    if camino in G.edges:
        vmax_actual = G.edges[camino].get("maxspeed")
        if vmax_actual is None:
            vmax_actual = 50  # Fija un valor para la velocidad máxima
        else:
            vmax_actual = float(vmax_actual)
            vmax_nueva = vmax_actual * (1 - (intensidad / intensidad_03_00h_max))
            G.edges[camino]["maxspeed"] = vmax_nueva
    else:
        vmax_actual = G.edges[camino].get("maxspeed")
        vmax_nueva = vmax_actual * (1 - (intensidad_03_00h / intensidad_03_00h_max))
        G.edges[camino]["maxspeed"] = vmax_nueva

ox.save_graphml(G, 'network_vmax_mod_2100.graphml')
```

Con el algoritmo anterior, nos aseguramos de que se reducirá la velocidad máxima de cada camino, según la intensidad de tráfico.

8.5. Algoritmo de la colonia de hormigas

Incluye la modificación del algoritmo original para tener en cuenta la velocidad máxima de cada camino a la hora de distribuir las feromonas.

```
# Función que calcula el nivel de feromonas, iniciación de variables
alfa=3
beta=3
def feromonas(level, distancia, alfa, beta):
    return level ** alfa * ((1/distancia)) ** beta
rutas_conocidas=dict()
tiempo_ruta=float(0)
concentraciones_feromonas = {(u,v):random.uniform(0,0.5) for [u,v] in G.edges()}
```

```
for hormiga in tqdm(range(500)):
    # Coloca la hormiga en el mapa
    frontera = [origen]
    explorados = set()
    ruta = []
    encontrada = False
    while frontera and not encontrada:
        padre = frontera.pop(0)
        explorados.add(padre)
        hijos = []
        feromonas_hijos = []
        for hijo in padre.expand():
            # Si detecta el destino, ignora todas las feromonas
            if hijo == destino:
                encontrada = True
                ruta = hijo.path()
                continue
            if hijo not in explorados:
                hijos.append(hijo)
                feromonas_hijos.append(
                    feromonas(
                        concentraciones_feromonas[(padre.osmid, hijo.osmid)],
                        hijo.distance,
                        alfa,
                        beta,
                    )
                )
        if len(hijos) == 0:
            continue  # La hormiga está atrapada, regresa.
        probabilidad_transicion = [feromonas_hijos[i] / sum(feromonas_hijos)
                                   for i in range(len(feromonas_hijos))]
        # Elige probabilísticamente un hijo para explorar, ponderado por la probabilidad de transición
        elegido = random.choices(hijos, weights=probabilidad_transicion, k=1)[0]
```

```
        # Agrega todos los caminos no explorados en caso de que necesitemos explorarlos más tarde
        hijos.pop(hijos.index(elegido))
        frontera.extend(hijos)

        # Establecer el camino elegido como el siguiente nodo a explorar
        frontera.insert(0, elegido)

    #Una vez encontrado un camino, se añaden más feromonas
    for u, v in zip(ruta[:-1], ruta[1:]):
        longitud_camino = G[u][v][0]["length"]
        velocidad_max= float(G[u][v][0].get("maxspeed",default_maxspeed))
                    #Se usa el parámetro maxspeed para la concentración de feromonas
        velocidad_max_ms=velocidad_max/3.6
        tiempo_ruta+=longitud_camino/velocidad_max_ms
        concentraciones_feromonas[(u, v)] += velocidad_max/longitud_camino

    # Si la ruta es recién descubierta, se añade a la lista
    ruta = tuple(ruta)
    if ruta in rutas_conocidas:
        rutas_conocidas[ruta] += 1
    else:
        rutas_conocidas[ruta] = 1
```

```
mejor_ruta = max(rutas_conocidas, key=rutas_conocidas.get)
ruta = list(mejor_ruta)
print("Longitud de la ruta:", cost(G, ruta),"metros")
print("Tiempo en la ruta es:",tiempo_s(G,ruta),"segundos")
osmnx.plot_graph_route(G, ruta, route_color='red')
```

8.6. Algoritmo de la colonia de abejas

```
import osmnx
from smart_mobility_utilities.common import Node, cost, randomized_search, tiempo_s
from smart_mobility_utilities.children import get_children
import networkx as nx
from tqdm.notebook import tqdm
import random
import matplotlib.pyplot as plt
```

```
G = osmnx.load_graphml("network_vmax_mod_2100.graphml")
origen = Node(graph=G, osmid=258606648)
destino = Node(graph=G, osmid=447770155)
nodos = [258606648, 447770155]
color_nodos = ['red' if node in nodos else 'blue' for node in G.nodes()]
osmnx.plot_graph(G, node_color=color_nodos,node_zorder=2)
```

```
# Parámetros del algoritmo
limite_iteraciones = 50  # Número máximo de iteraciones
total_abejas = 15  # Total de abejas en la simulación

# Inicializa listas para almacenar datos de las abejas y sus rutas
fuentes_alimento = []
coste_fuentes_alimento = []
fuentes_observadoras = []
coste_fuentes_observadoras = []
mejor_coste = float(1000000)  # Mejor coste encontrado, inicializado a 1000000
mejor_ruta = None
estados_iteracion = []  # Contador iteraciones

for _ in range(total_abejas // 2):
    ruta = randomized_search(G, origen.osmid, destino.osmid)  # Genera una ruta aleatoria
    fuentes_alimento.append(ruta)  # Añadir la ruta a la lista de fuentes de alimento
    coste_fuentes_alimento.append(cost(G, ruta))

# Inicializar listas para las abejas observadoras
fuentes_observadoras = [[] for _ in range(len(fuentes_alimento))]
coste_fuentes_observadoras = [float(1000000) for _ in fuentes_alimento]
```

```
for _ in tqdm(range(limite_iteraciones)):
# Fase abejas empleadas
    for i in range(total_abejas//2):
        vecino = get_children(G, fuentes_alimento[i], 1)[0]
        coste_vecino = cost(G, vecino)
        if coste_vecino < coste_fuentes_alimento[i]:
            fuentes_alimento[i] = vecino
            coste_fuentes_alimento[i] = coste_vecino
        if coste_vecino < mejor_coste:
            mejor_coste = coste_vecino
            mejor_ruta = vecino
# Fase abejas observadoras
    total = sum(1 / x for x in coste_fuentes_alimento)
    probabilidades = [(1 / coste_fuentes_alimento[i]) / total for i in range(total_abejas // 2)]
    fuentes_a_visitar = []
    for i in range(total_abejas // 2):
        eleccion = random.choices(population=fuentes_alimento, weights=probabilidades)[0]
        # Si la elección de la abeja no es óptima, explora un vecino de la actual en su lugar.
        if cost(G, eleccion) > coste_fuentes_observadoras[i]:
            vecino = get_children(G, fuentes_observadoras[i], 1)[0]
            coste_vecino = cost(G, vecino)
            if coste_vecino < coste_fuentes_observadoras[i]:
                fuentes_observadoras[i] = vecino
                coste_fuentes_observadoras[i] = coste_vecino
                if coste_vecino < mejor_coste:
                    mejor_coste = coste_vecino
                    mejor_ruta = vecino
            continue

        # De lo contrario, genera un vecino de la solución seleccionada por la abeja observadora
        fuentes_a_visitar.append(eleccion)
        vecino_observador = get_children(G, eleccion, 1)[0]
        coste_vecino_observador = cost(G, vecino_observador)
        if coste_vecino_observador < mejor_coste:
            mejor_coste = coste_vecino_observador
            mejor_ruta = vecino_observador
        if coste_vecino_observador < cost(G, eleccion):
            fuentes_observadoras[i] = vecino_observador
            coste_fuentes_observadoras[i] = coste_vecino_observador
        else:
            fuentes_observadoras[i] = eleccion
            coste_fuentes_observadoras[i] = cost(G, eleccion)

# Fase de abejas exploradoras
    abejas_exploradoras = []
    for i in range(total_abejas // 2):
        if fuentes_alimento[i] not in fuentes_a_visitar:
            abejas_exploradoras.append(i)
```

```
    for i in abejas_exploradoras:
        fuentes_alimento[i] = randomized_search(G, origen.osmid, destino.osmid)
        coste_fuentes_alimento[i] = cost(G, fuentes_alimento[i])

    estados_iteracion.append(mejor_coste)
```

```
print("Longitud de la ruta:", mejor_coste,"metros")
print("Tiempo en la ruta es:",tiempo_s(G,mejor_ruta),"segundos")
osmnx.plot_graph_route(G, mejor_ruta, route_color='green')
```

```
plt.plot(estados_iteracion)
plt.xlabel("Iteraciones")
plt.ylabel("Coste (m)")
plt.show()
```

8.7. Código desarrollado y añadido a la función smart_mobility_utilities/common.py

Se utiliza para poder evaluar la duración de cada ruta calculada en segundos.

```
def tiempo_s(G, route):
    tiempo_total = 0
    for u, v in zip(route, route[1:]):
        datos_camino=G[u][v][0]
        longitud=datos_camino["length"]
        if "maxspeed" in datos_camino:
            velocidad_kmh = float(datos_camino["maxspeed"])
            velocidad_ms = velocidad_kmh/3.6
        else:
            velocidad_ms= 50/3.6
        tiempo_seg=longitud/velocidad_ms if velocidad_ms > 0 else 0
        tiempo_total+=tiempo_seg
    return round(tiempo_total,1)
```