¿ES VIABLE UNA RED DE AUTOBUSES SIN LÍNEAS?

Miguel Franco Medina
Ignacio Yazid Guerra de las Peñas
Sergio Sánchez Ruiz
GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO EN INGENIERÍA DEL SOFTWARE
2016-2017
Director: Ismael Rodríguez Laguna

Autorización de difusión y utilización

Nosotros, Miguel Franco Medina, Ignacio Yazid Guerra de las Peñas y Sergio Sánchez Ruiz, alumnos matriculados en el Grado de Ingeniería del Software impartido por la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo de Fin de Grado, realizado durante el curso académico 2016-2017 y bajo la dirección de Ismael Rodríguez Laguna en el Departamento de Sistemas Informáticos y Computación, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet, y garantizar su preservación y acceso a largo plazo.

Miguel Franco Medina, Ignacio Yazid Guerra de las Peñas y Sergio Sánchez Ruiz

Prólogo

¿Qué es una red de autobuses sin líneas?

Esta es la primera cuestión que nos asalta. La dependencia actual que nuestra sociedad tiene de los combustibles fósiles para los vehículos motorizados genera un gran número de inconvenientes para nuestra salud y nuestro entorno. El cambio climático es algo que preocupa a una gran parte de la población y la necesidad de disminuir el consumo energético proveniente de cualquier fuente es real.

Con una red de autobuses sin líneas, queremos conseguir que el transporte público terrestre más utilizado siga estando a la cabeza sin descuidar nuestra salud y nuestro planeta. La solución que proponemos se basa en que los autobuses de nuestra red (además de ser eléctricos, o en su defecto, híbridos) estén gestionados a través de un servidor que calculará la ruta óptima en cada momento para cada autobús de nuestra red en función de la demanda de nuestros usuarios. Al no existir rutas preestablecidas, ningún autobús estará realizando un movimiento pasivo por el mero hecho de cumplir un horario, siempre tendrán un objetivo fijo, el usuario.

Con este método pretendemos mejorar el sistema de transporte. Cada usuario podrá realizar una petición a nuestro servidor a través de la aplicación móvil de nuestra red, recibiendo el número del autobús, la hora, el lugar donde debe presentarse para ser recogido y el destino, siempre lo más cercano posible a donde lo solicite el usuario. Esto reducirá el número de autobuses en funcionamiento simultáneamente, ya que siempre tendemos a minimizar el número de vehículos en movimiento acorde con la demanda. O alternativamente, mejorar el servicio con los autobuses mismos disponibles.

ÍNDICE

Índice de figuras

Índice de abreviaturas

ACO: Ant Colony Optimization.

API: Application Programming Interface.

App: Aplicación.

BBDD: Base de datos.

CSS: Cascading Style Sheets.

FCM: Firebase Cloud Messaging.

GCM: Google Cloud Messaging.

GPX: GPS eXchange Format (Formato de Intercambio GPS).

HTML: HyperText Markup Language.

IDE: Integrated Development Environment.

JSON: JavaScript Object Notation.

NPM: NodeJS Package Manager.

RAD: Red de Autobuses dinámica.

REST: Representational State Transfer.

RFD: River Formation Dynamics.

SQL: Structured Query Language.

TSP: Travelling Salesman Problem.

TW: Time Window.

Resumen

El objetivo de este proyecto es el estudio de la viabilidad de una red de transporte público de autobuses sin líneas preestablecidas. En la memoria describiremos los diferentes estudios realizados para conseguir información útil.

Para ello, hemos desarrollado una aplicación capaz de calcular las rutas a seguir por los diferentes autobuses basadas en las peticiones de los usuarios de la red mediante una aplicación móvil.

El servidor será capaz de recibir la solicitud de un usuario (cliente) con la geolocalización del lugar donde requiere el servicio y la hora específica, computar las opciones de recoger a dicho usuario, eligiendo la que considere mejor, y finalmente informar al usuario del lugar y la hora donde deberá estar para ser recogido y llevado a su destino.

Detallaremos cómo ha sido el proceso de desarrollo de la aplicación y del servidor desde las investigaciones iniciales hasta el resultado final.

Tras los experimentos y pruebas realizadas, los resultados obtenidos han sido satisfactorios pero no esclarecen que una red de autobuses dinámicos sea una opción real para las ciudades, ya que en algunos casos obtiene mejores resultados que redes estáticas pero en otros, como se refleja en las pruebas, obtiene peores resultados. Esto, sumado a la necesidad de concienciar a la sociedad para el uso de una red dinámica y la adaptación de las infraestructuras actuales para el desarrollo de las actividades hace que solo pueda llevarse a la práctica en determinadas condiciones.

Abstract

The goal of this project is the study of the feasibility of a bus public transportation network without a preset paths. In this report, we will describe different studies carried out to obtain useful information. For this, we have developed an application able to calculate the paths to be followed by the buses based on the requests made by the users of the net.

The application along with the server will be capable to receive the request of each user with the geolocation of the place where he needs the service and the specific time. When the server receives the request, the server will compute the different options for picking up this user (choosing the one it thinks that is the best). Finally the server will send back to the user the place and the time where he will be picked up and taken to his destination.

In this report, we will describe how the process of the application and the server have been developed since the very beginning until the final result.

After the experiments and tests carried out, the results have been satisfactory, but it does not claridy that a dynamic bus network is a real option for the cities nowadays because in some cases it obtains better results that the static bus network but, in others, as shown in tests, the obtained results are worse. This, coupled with the need to make society aware of the use of a dynamic bus network and the adaptation of the nowadays infraestructures for the development of the new activities makes it only posible to put into practice on certain conditions.

Palabras clave:

- Red de Autobuses Dinámica
- TSP
- TSPTW
- Agrupamiento3
- Servidor web
- Cliente
- C++

Keywords:

- Dynamic Bus Network
- TSP
- TSPTW
- Clustering
- Web server
- Client
- C++

Capítulo 1. Introducción a RAD

1.1 Introducción

Cada día, millones de personas en el mundo usan el transporte público como alternativa para moverse dentro de su ciudad para ir a trabajar, estudiar, comprar o ir al médico entre otras muchas cosas. Esto genera complejos problemas de logística en las ciudades, hasta tal punto que hoy en día sería inimaginable la expansión de una ciudad, municipio o barrio sin tener que pensar en cómo se va a comunicar con el resto de la civilización ya existente en la zona. De hecho, en la mayoría de los casos, antes incluso de finalizar la construcción de un nuevo polígono de oficinas, casas o fábricas, se instalan las carreteras por las que miles de personas tendrán acceso a ellas.

Con la construcción de estas carreteras, las empresas de transporte empiezan a planificar la forma de llegar hasta allí para dar soporte a todo aquel que requiera transporte eficiente, y por eso las líneas de autobús son las primeras en llegar.

Una solución bastante habitual a este problema suele ser la creación de nuevas paradas de autobús en la zona, nuevas líneas cuyas rutas cubren estas paradas, y añadir estos nuevos autobuses a unos posiblemente ya masificados intercambiadores de líneas de autobús que en su día no fueron pensados para soportar el tránsito de millones de personas cada día.

¿Qué es RAD?

La Red de Autobuses Dinámica (la llamaremos RAD a partir de este momento) es una red de autobuses que no dispone de unas rutas específicas. Es decir, la ruta seguida por cada autobús, las paradas a realizar y los viajeros a recoger no estarán determinados hasta el momento en el que el primer usuario realiza una petición.

El sistema procesa dicha petición y pondrá en funcionamiento el primer autobús, haciendo que el sistema comience a andar. El usuario solicitará un servicio desde un lugar concreto, a una hora concreta y con un destino, entonces el sistema calculará la mejor forma de llegar hasta dicho punto y qué autobús debe modificar su ruta para recoger a este nuevo usuario, reduciendo siempre al mínimo los posibles retrasos ocasionados a todos los demás usuarios de dicho autobús.

La RAD, dispone de un sistema de transbordos. Dichos transbordos se calculan con las rutas de todos los autobuses. Este sistema se encarga de comprobar en qué punto dos (o más) rutas estarán dentro de una distancia (menos de 300m) y un rango de tiempo aceptable(entre 2 y 15 minutos de espera entre autobuses) para que un usuario pueda beneficiarse de las rutas de otros autobuses que no podría usar de ningún otro modo.

¿Por qué RAD?

El sistema RAD nos permitirá realizar peticiones de servicio con antelación. Esto resolverá muchos de nuestros problemas de movilidad a nivel usuario, ya que seremos recogidos dónde y cuándo queramos. Esto nos evitará tiempos de espera y desplazamientos a las paradas convencionales asegurándonos llegar a nuestro destino de una forma rápida y eficiente. Gracias a RAD podremos cruzar la ciudad sin tener que realizar un gran número de transbordos y cambios de localización para ir adaptándonos a los distintos puntos de intercambio convencionales.

A nivel empresarial, es una alternativa magnífica para poder reducir costes, ya que podremos recoger a todos los usuarios que lo necesiten y planificar mejor nuestra flota gracias a las solicitudes realizadas con antelación, lo que nos permitirá hallar las mejores rutas para que ningún autobús consuma recursos innecesarios y así eliminar uno de los problemas generados por la expansión de las ciudades.

Como sociedad, disminuirá la contaminación, ya que menos vehículos implican menor contaminación, o alternativamente, a igualdad de vehículos, se proporcionará un mejor servicio. Un mejor sistema de transportes hará que la sociedad se decante más por su uso, siendo posible con esto la mejora de todas estas infraestructuras, disminuyendo los atascos y en general, facilitando la vida de todos los individuos.

1.2 Objetivos

Los objetivos principales del proyecto consisten en:

- Desarrollar un sistema funcional basado en RAD consistente en: una aplicación servidor que enruta a los usuarios con la posibilidad de asignarles un transbordo y una aplicación móvil para usuarios y conductores.
- Desarrollar una simulación de la toma de decisiones que permita llevar a cabo diferentes ejecuciones del sistema introduciendo usuarios en puntos arbitrarios.
- Analizar los resultados de aplicar los algoritmos de enrutamiento en diferentes condiciones, basándonos en datos tanto demográficos como de redes de transporte público ya existentes.
- Comparar nuestros resultados con los de un sistema estático o semi-dinámico, y obtener conclusiones de en qué condiciones y por qué sería beneficioso o no aplicar una RAD.
- Implementación de un sistema de transbordos entre autobuses mediante el cual, los usuarios recibirán notificaciones cuando deban de cambiar de autobús para llegar a su destino de forma más óptima.

1.3 Partes del sistema

El sistema está formado por diversas partes, que se explican brevemente para una mejor comprensión de la memoria.

1.3.1 División del mapa

Se decidió usar el mapa de la ciudad de Madrid por la familiaridad que tenemos con el mismo, además de las características especiales de la ciudad, como número de habitantes, dimensión, etc… con el fin de obtener unos datos lo más realistas posibles. La división explicada a continuación se puede adaptar a cualquier ciudad. La manera en la que se decidió dividir el mapa fue en forma de cuadrícula. Esta está formada por 100 filas y 100 columnas de cuadrados de dimensiones iguales abarcando un área aproximada de 12.5 kilómetros cuadrados.

Cada uno de los cuadrados tiene un lado de 125 m en línea recta teniendo en cuenta la curvatura de la tierra, gracias a un método obtenido de la librería libosmscout (que explicaremos más adelante). Esta cuadrícula o matriz va a permitir al algoritmo de agrupamiento funcionar de una manera más efectiva, de tal modo que se usará cada esquina de los cuadrados como una posible parada para los usuarios que soliciten el servicio. Si en algún caso uno de estos puntos no fuese accesible por los autobuses (por cualquier tipo de restricción de circulación), el sistema reubicará cualquier petición en un punto accesible para el autobús (una calle).

La matriz crea una rejilla sobre el mapa y lo divide en 10.000 pequeñas zonas que se usarán como referencias espaciales para cualquier necesidad del sistema, ya sea controlar las desviaciones de los autobuses sobre sus rutas iniciales o el desplazamiento que deberán acometer los usuarios para llegar a la ubicación donde serán recogidos/dejados.

Esta cuadrícula NO ejerce una restricción determinante sobre las ubicaciones para el sistema, ya que este se encarga de asegurar que los puntos escogidos donde los autobuses realizarán una parada sean accesibles y óptimos para cada autobús. Por tanto, se usarán a modo de guía que en muchos casos se seguirá, pero no será una necesidad adaptarse a ella.

1.3.2 Algoritmo de agrupamiento

A partir de ahora, referido como clúster. Este algoritmo se encarga de recibir todas las solicitudes de los usuarios y agrupar a los mismos en base a la ubicación de origen, la hora a la que se solicita el servicio y la ubicación de destino. El objetivo de este algoritmo es conseguir que el número de paradas globales que los autobuses tendrán que realizar sea el mínimo posible, disminuyendo así tanto el número de autobuses necesarios como el tiempo perdido en cada parada (deceleración de un autobús hasta su detención, apertura y cierre de puertas y aceleración hasta la velocidad inicial).

1.3.3 Algoritmo de enrutamiento

El algoritmo seleccionado para calcular la mejor solución posible basándose en las restricciones del problema ha sido un algoritmo de tipo genético, resolviendo diferentes instancias de un problema similar al TSP en cada iteración. Este algoritmo recibe a los usuarios agrupados en paradas por el clúster y calcula cual es el mejor modo de dar servicio a todas y cada una de las solicitudes recibidas en el servidor. Una solución válida para el problema está compuesta por el conjunto de las rutas de cada autobús al finalizar cada ejecución completa del programa. El algoritmo, además de recibir como entrada el clúster de

paradas, recibirá la solución (compromisos actuales de los autobuses) actual sobre la que está trabajando el sistema, con el fin de poder reutilizar el mayor número de autobuses en movimiento sin alterar la solución anterior, es decir, el algoritmo siempre busca que, con los nuevos usuarios, la nueva solución contenga nuevas líneas de autobús o que las existentes den servicio a más usuarios sin perjudicar demasiado a los demás usuarios.

1.3.4 Servidor de cálculo de rutas

Para poder conocer cuánto se tarda de un punto a otro, y como llegar al mismo necesitamos una librería que calcule rutas que pasen por los puntos que nosotros precisamos.

El servidor que calcula las rutas es OSRM. Este servidor recibe peticiones HTTP en las que se solicita ruta de varios puntos (latitud, longitud) y envía la solución mediante una respuesta HTTP con datos formateados en JSON.

1.3.5 Servidor para la gestión de peticiones

Este servidor se encarga de la gestión de las peticiones. Es el enlace entre la aplicación móvil de cada usuario y la base de datos, de donde el programa principal, formado por el clúster y el algoritmo de enrutamiento, obtiene los datos para poder generar las soluciones. Tras recibir cada una de las solicitudes, actualizará la BBDD con los datos concretos de cada petición. Además, se encarga de notificar a cada usuario los detalles acerca del servicio de recogida.

Una petición está formada por los siguientes datos:

- Identificación del usuario que la realiza.
- Ubicación (o parada) de origen donde el usuario solicita el servicio, que puede (o no) coincidir con la ubicación actual del usuario en el momento de realizar la solicitud.
- Ubicación (o parada) de destino donde el usuario solicita ser depositado por el autobús.
- Hora a la que se solicita el servicio en la parada de origen, esta hora se tomará de referencia para el sistema de ventanas de tiempo del sistema.

1.3.6 Base de datos

La base de datos de tipo relacional contiene todos los datos relevantes de los usuarios registrados, los autobuses, las rutas y las solicitudes de servicio. Mediante la misma tenemos capacidad para representar la simulación del sistema en una interfaz web.

1.3.7 Aplicación móvil

La aplicación móvil es la conexión entre el cliente del servicio y el sistema. Su función es facilitar al usuario la solicitud del servicio y conectar las peticiones de los usuarios con el servidor que gestiona las peticiones, para que estas queden registradas en la BBDD y entren en el sistema.

1.3.8 Interfaz web

Aplicación web que representa el mapa de Madrid con las diferentes paradas de origen y destino de los usuarios. Así como la posición actual de los autobuses y sus rutas.

1.3.9 Fichero JSON datos de usuarios

Fichero con formato JSON donde se guardarán los datos a tiempo real de los usuarios. Será el intermediario entre la aplicación móvil y el cálculo de rutas.

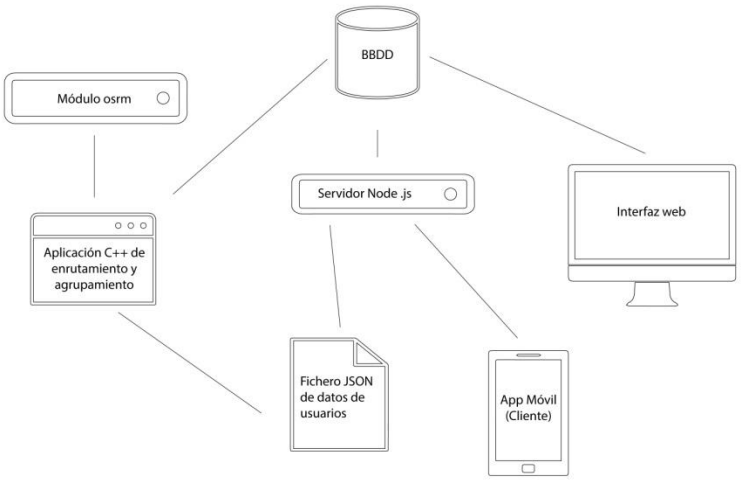

Figura 1: Estructura general del sistema

Capítulo 2: Fases del proyecto

2.1 Fase de investigación

Durante los dos primeros meses del proyecto, se desarrolló en exclusiva la fase de investigación. Durante este periodo se llevaron a cabo las labores de investigación acerca del TSP y la búsqueda de una librería que prestase el servicio de cálculo de las rutas de los autobuses a partir de coordenadas. Además, se tomó una decisión determinante para el desarrollo del proyecto, se eligió C++ como el lenguaje de programación a utilizar para el grueso de la fase de desarrollo del proyecto. Posteriormente y durante el resto de fases del proyecto, se siguieron realizando tareas de investigación sobre cualquier área necesaria para una mejor comprensión y desarrollo del proyecto, como el servidor para la gestión de aplicaciones, la base de datos o la aplicación móvil.

2.1.1 Algoritmo de enrutamiento

Enrutar a una población de usuarios hacia una población de autobuses es una tarea que depende en gran medida de cálculos de soluciones a problemas TSP. En particular cada autobús tiene que elegir la mejor ruta para recoger al mayor número de usuarios posibles, eligiendo solo usuarios que tengan un destino coherente en cuanto a distancia a la ruta actual del autobús.

El TSP presenta un problema principal, pues su complejidad es NP-duro. Por tanto, se deben aplicar métodos heurísticos para conseguir una solución suficientemente buena en un tiempo razonable.

Rápidamente se acotó que el problema concreto que se debía resolver era una versión del VRPTW, el cual incluye ventanas de tiempo (TW) que es necesario satisfacer. Esta es una restricción básica para las soluciones de nuestro problema, ya que dichos nodos sólo son accesibles durante un intervalo de tiempo determinado (fuera de esta ventana, el nodo concreto no puede ser añadido a una solución posible).

La conclusión de este estudio determinó que sería necesario implementar diferentes algoritmos para la resolución del problema y de este modo, determinar cuál de ellos se adaptaba mejor a las necesidades del proyecto.

Los algoritmos estudiados han sido:

- Algoritmo voraz.
- Algoritmo ACO[21].
- Algoritmo RFD[21].
- Algoritmo Dijkstra.
- Algoritmos genéticos.

Se decide implementar un algoritmo voraz, un algoritmo basado en permutaciones (que genera todas las opciones de órdenes posibles) y un algoritmo genético durante la fase de desarrollo para las diferentes pruebas en busca de la mayor adaptabilidad y rendimiento para el proyecto.

La primera implementación del enrutamiento se basaba en un algoritmo voraz. El funcionamiento se basaba en que, para cada iteración del bucle para cada autobús, se elegían a los usuarios más cercanos al mismo, y este los recogía. Esta solución, implementada internamente con una cola de prioridades variables (los elementos de la cola son los usuarios, y la prioridad la distancia al autobús) ofrecía soluciones muy rápidas en tiempo de cálculo.

En concreto, se tomó la decisión de resolver el VRTPW que se nos planteaba mediante un algoritmo heurístico. El más sencillo de aplicar fue el algoritmo del vecino más próximo. Su método para resolver un TSP aplicado en nuestro sistema es el siguiente:

1- Elección de un usuario aleatorio desde el autobús. Usuario marcado.

2- Elección del usuario más cercano al actual.

3- Marcamos el usuario más cercano como el actual.

4- Si siguen existiendo usuarios sin asignar volver al punto 2.

Desde el punto de vista de la optimización del camino calculado para N usuarios aleatoriamente distribuidos, el algoritmo en promedio retorna un camino un 25% más largo que el menor camino posible. También puede devolver el camino peor en muchos casos. En general, si se cumple la desigualdad triangular, según Rosenkrantz[22] el tiempo de ejecución está en orden de **O (log V)**.

Este primer sistema permitió tener una estructura general del proyecto funcionando, estructura que permitía ver cómo el resto de componentes, como la interfaz o la base de datos, estaban funcionando adecuadamente respondiendo ante los eventos que sucedían en el servidor. Aun así, las soluciones que se obtenían no eran buenas: mediante este sistema no se satisfacían TW en ningún caso, ya que la ruta recorrida por el autobús no era la adecuada.

Después de trabajar con el algoritmo voraz, se estudió un algoritmo muy directo, un algoritmo exacto, como es el de permutaciones. El modo de resolver el TSP se basaba en obtener todas las permutaciones posibles del camino y quedarnos con la menor en tiempo y distancia. Para caminos con pocos nodos, como 3 o 4, funcionaba bien ya que el tiempo de cálculo era de pocos segundos, pero al aumentar el número de nodos esto cambiaba mucho, hasta irse a tiempos de cálculo de horas y días. El tiempo de ejecución del mismo es un factor polinómico de orden **O (N!)**, siendo N el número de destinos.

Además, estudiamos el algoritmo Held-Karp[23], también algoritmo de tipo exacto, cuya eficiencia está en el orden de $\mathbf{O\ (n^2 2^n)}$, pero no se fue capaz de implementar en un tiempo razonable en nuestro sistema.

Lo positivo de trabajar temporalmente con el algoritmo de permutaciones fue percatarse de que resolviendo el TSP con una mayor precisión, se obtenían mejores caminos, por lo que él sistema era capaz de funcionar adecuadamente, cumpliendo los TW de recogida y destino de los usuarios.

Finalmente, se decidió estudiar un algoritmo genético e implementarlo. En particular, se buscaban soluciones del mismo tipo que las estudiadas con los algoritmos anteriores.

En el punto 2.1.2 se analiza cómo se agrupan a los usuarios para saber si son estudiados en la solución de uno u otro autobús.

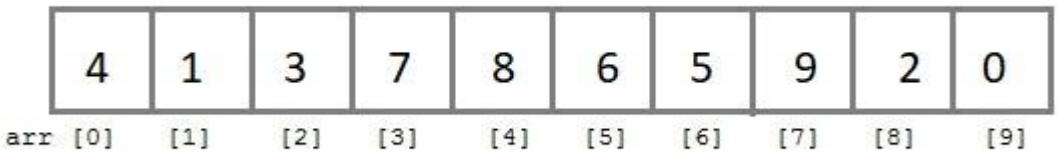

Figura 2: Vector que representa el orden de paradas

En el algoritmo genético, cada cromosoma (solución candidata) es un vector que representa el orden que tiene que seguir el autobús para recorrer las paradas, tratando de maximizar el número de usuarios cumpliendo siempre los TWs de los mismos (y también minimizando la distancia recorrida por el mismo, pero dándole prioridad a recoger más usuarios).

Dimos con un algoritmo en C++ para resolver variantes del TSP genéticamente, por lo que aplicamos el mismo, con severas modificaciones, en nuestro sistema:

https://github.com/marcoscastro/tsp_genetic

La solución al TSP con un algoritmo genético funciona del siguiente modo: en primer lugar, se crean una serie de soluciones que forman la base de la población, entendiendo solución como un vector que mantiene el orden en el que pasar por el mayor número posible de nodos. Estas primeras soluciones son aleatorias, en nuestro caso la primera es el orden 1, 2, 3, 4, 5, 6, 7, 8, 9…

Después se mezclan las soluciones entre sí, evitando seleccionar dos veces al mismo nodo, obteniendo padres e hijos. Como último paso se produce una mutación, que modifica ligeramente las soluciones para obtener cierto componente de aleatoriedad. Como se puede observar en la Figura 3, entre cada fase llamamos a una función isValidSolution(), que comprueba para cuántos usuarios es válida la solución. Estas comprobaciones se explican más detalladamente en el desarrollo del algoritmo de enrutamiento (sección 2.2.1).

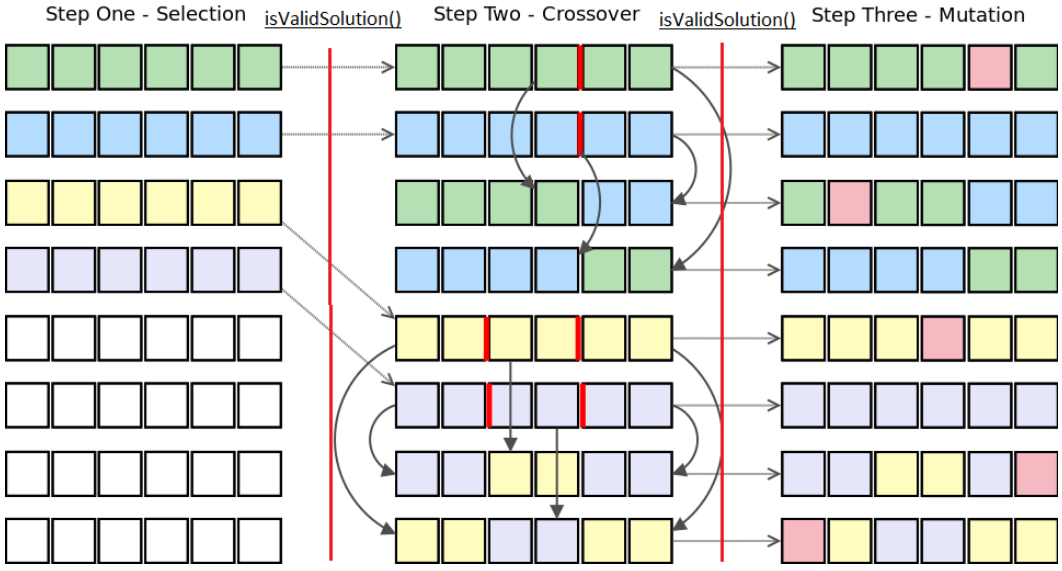

Figura 3: Ejemplo de función isValidSolution

Este proceso acaba con una población con las mejores soluciones, quedándonos nosotros con la que satisfaga al mayor número de usuarios posible, y dentro de estas, si hay varias, la que implique menor tiempo en recorrerse desde el punto de vista del autobús.

2.1.2 Algoritmo de agrupamiento

Tras ser tomada la decisión de convertir el mapa de la ciudad elegida (Madrid en el caso de este proyecto) en una cuadrícula de tamaño 100 por 100 de 12,5 kilómetros cuadrados (denominada rejilla de ahora en adelante) se decidió llevar a cabo una estrategia de clustering para las solicitudes de servicio, con el objetivo de conseguir un agrupamiento de la mayor cantidad de usuarios y, con ello, reducir el número de compromisos globales del sistema, pudiendo con ello optimizar al máximo la utilidad de los autobuses en circulación y dando servicio a todos los usuarios en tiempos aceptables.

Cada cuadrícula tiene un tamaño de 125 m^2 (como se explica en el apartado 1.3.1). Esto quiere decir que, aunque el punto concreto no esté situado en una calle accesible por un autobús, puede ser objeto de una parada, ya que el sistema de cálculo de rutas convertirá dicho punto en una ubicación accesible para el autobús, siendo esta la posición accesible para el autobús más cercana al punto inicial. La función principal de esta cuadrícula es que cada vértice de cada uno de los 10.000 cuadrados que forman la cuadrícula equivale a una posible parada de para nuestro sistema, de tal modo que hay una amplia variedad de puntos en donde se podrá recoger/dejar a cualquier usuario a lo largo de toda la zona de servicio.

Una vez se dividió el mapa, se decidió el funcionamiento de las ventanas de tiempo. Estas ventanas de tiempo comprenden un tiempo máximo de 15 minutos, de tal modo que, por ejemplo, si un usuario solicita un servicio a las 11:05, su ventana de tiempo comprenderá desde las 11:05 hasta las 11:20, siendo cualquier momento, dentro de esta ventana, cuando el autobús que va a realizar el servicio efectuará su parada en el lugar indicado.

Los criterios de restricción para agrupar las diferentes solicitudes se basan en tres criterios fundamentales.

La primera restricción tenida en cuenta es la distancia, de tal modo que todas las paradas que se encuentren en un radio inferior a 500 metros desde la ubicación de la solicitud serán candidatas a recibir al nuevo usuario.

La segunda restricción a la hora de agrupar las solicitudes es que estas compartan una ventana de tiempo, ya que si esta no coincide, no se estaría produciendo un agrupamiento real. Compartir una ventana de tiempo implica que ambas solicitudes coincidan de forma total o parcial en la hora de la solicitud.

La tercera y última restricción es que el autobús (o los autobuses) que en su recorrido pasen exactamente (o suficientemente cerca) de la parada de origen del usuario, y dentro de la ventana de tiempo del mismo, contengan en cualquier punto de su recorrido (el recorrido posterior al punto de recogida del usuario) un punto suficientemente cercano a la ubicación de la parada de destino del usuario o un transbordo con cualquier otro autobús que sí tenga programado en su ruta un punto cercano al destino de la solicitud tratada. Se entiende por suficientemente cercano a una distancia igual o inferior a 400 metros, para evitar cualquier tipo de retraso excesivo sobre la ruta original del autobús con el consiguiente perjuicio de todos los demás usuarios.

En caso de no cumplir estas restricciones, este usuario se añadirá a la ubicación más cercana a su ubicación para que, posteriormente, el algoritmo de enrutamiento le asigne un autobús.

2.1.3 Cálculo de rutas

El proceso de investigación para poder calcular rutas entre dos posiciones dadas dio comienzo en octubre. Analizando las opciones que existen en C++, se optó por buscar una librería que, incluida en el proyecto, permitiera llevar a cabo esos cálculos.

El requisito primordial de este proyecto era poder hacer muchos de esos cálculos, sin límites de uso. La velocidad de dichos cálculos era un factor secundario, buscando principalmente poder llevar a cabo el número de cálculos que fuesen necesarios sin límites de peticiones en un tiempo dado.

La primera solución con la que se trabajó fue Google Maps, que permitía obtener datos de rutas mediante JavaScript con su API de direcciones, haciendo llamadas al servidor de Google en Internet, aunque no disponía de una librería en C++ como tal.

Google no facilita en ningún caso su API sin límites de uso o con unos límites suficientes para que, en este caso, se pudiera trabajar cómodamente, ni dispone de permisos especiales para universidades españolas para trabajar con esta API. Si se querían usar las rutas de Google, eran necesario comprar una licencia cuyo coste no se contempló durante el desarrollo del proyecto. Por lo tanto, esta opción quedó descartada. Además, el tiempo que pasaba desde la petición hasta la respuesta era demasiado alto, por lo que buscamos otras alternativas.

En cuanto a mapas, la alternativa de código abierto a Google Maps es OpenStreetMaps, una muy buena fuente de mapas, cuyos datos son introducidos de modo colaborativo por la comunidad. El acceso a los mismos es ilimitado, con licencias de código abierto y en local mediante ficheros .OSM o .OSM.PBF, pudiendo acceder a datos del país o ciudad que fuese necesario.

Una de las fuentes de estos mapas es https://www.geofabrik.de/, web que frecuentemente actualiza los datos de mapas con ficheros con extensión .OSM.PBF para ciudades y países. Estos datos los rellenan mediante información que aportan los usuarios.

Teniendo acceso a datos de mapas, se buscó la manera de poder calcular rutas en local sobre los mismos. Las librerías para C++ que mejor valoraba la comunidad eran:

- Libosmscout
- Libosmium
- Mapnik
- CartoType

Tras intentos de compilar todas ellas sin éxito, la primera decisión que se tomó fue usar libosmscut, librería que se puede compilar tanto dinámica como estáticamente.

Una librería estática es una librería que "se copia" en nuestro programa cuando lo compilamos. Una vez que tenemos el ejecutable de nuestro programa, el fichero original de la librería no es necesario. Podríamos borrarla y nuestro programa seguiría funcionando, ya que tiene copia de todo lo que necesita. Sólo se copia aquella parte de la librería que se necesite. Por ejemplo, si la librería tiene dos funciones y nuestro programa sólo llama a una, sólo se copia esa función.

Una librería dinámica no se copia en nuestro programa al compilarlo. Cuando tengamos nuestro ejecutable y lo estemos ejecutando, cada vez que el código necesite algo de la librería, irá a buscarlo a ésta. Si borramos la librería, nuestro programa dará un error de que no la encuentra.

En este caso únicamente se tenía acceso al código fuente de la misma, no a los ficheros compilados. Se decidió hacer uso de esta librería ya que es la más recomendaba por la comunidad y; por ende, la que más soporte tenía.

En primer lugar, se intentó compilar la librería dinámicamente para hacer uso de ella, pero los fallos de compilación que surgían impedían hacer uso de la misma, y en el apartado de soporte de la web no aparecía información de cómo compilar en modo estático, así que, de nuevo, se buscó otra alternativa.

Se encontró CartoType, con el que se podían visualizar mapas y calcular rutas sin problemas, además de disponer del código fuente al completo. En este caso el problema radicó en que el software era propietario. Tras contactar con el desarrollador, comentándole las características de nuestro proyecto y aportándole pruebas de que el mismo era de corte universitario y sin intención de obtener beneficios, su respuesta fue negativa ya que solicitaba, al igual que el servicio de Google, el pago de una licencia.

Viendo que no se podía hacer uso de CartoType, se volvió a recurrir a Libosmscout, que fue con la que más se avanzó en el proceso de compilación, al contactar con uno de los desarrolladores, lo que facilitó la información necesaria de cómo compilarla en modo estático.

El grupo de desarrolladores en activo solo tenía acceso al código fuente en diferentes versiones de Linux, pero ninguno en Windows. La ayuda que se recibió únicamente sirvió para compilar la librería en modo estático, lo que implicaba referenciar la licencia GPL en el código y al código fuente en todo el proyecto.

Después de superar esos problemas, se consiguió compilar el código fuente e incluirlo en nuestro proyecto, a mediados de noviembre.

Después de compilar el código fuente se disponía de cuatro ficheros ejecutables que componen Libosmscout, el primero de ellos Import.exe, para importar un mapa en formato OSM.PBF a la estructura interna que maneja la librería. Teniendo el mapa en la estructura interna funcionado, por primera vez se pudieran calcular rutas en C++ sin necesidad de software externo.

Uno de los cuatro componentes de Libosmscout es Libosmcout-map-qt, librería que permite la visualización de mapas en diferentes plataformas mediante QT. Este componente no se consiguió compilar y la ayuda que se brindó por parte de los desarrolladores estaba limitada a la ya prestada, ya que no tenían acceso a máquinas locales en Windows.

Una vez en este punto, se decidió que, aunque no se pudiesen visualizar los mapas mediante C++, usando los mismos datos de mapas en otra plataforma (con una interfaz web) se podían visualizar los mapas, por tanto, se continuaría con la librería libosmscout.

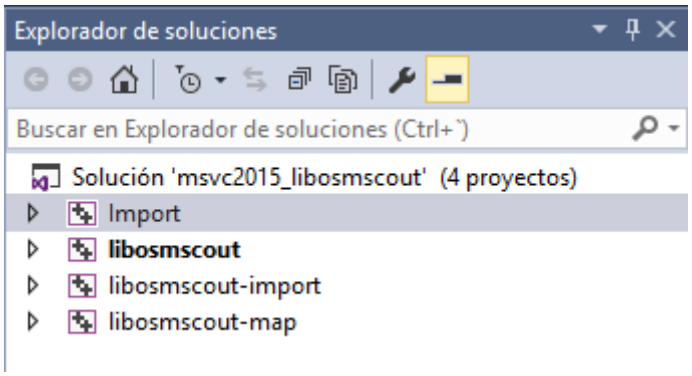

Figura 4: Librerías utilizadas en el proyecto C++

El componente principal de la librería Libosmscout es libosmscout, que permite calcular rutas y asociar calles a posiciones dadas y es el que integra el framework general que dispone de estructuras de datos y funciones básicas para trabajar con mapas, estructuras de datos como el tipo Point (para almacenar posiciones) o funciones como GetEllipsoidalDistance(), que permite obtener distancias en línea recta, teniendo en cuenta la curvatura de la tierra.

La función primordial en este caso era la de calcular rutas:

calculoRutas(Point & a, Point & b)

Necesitábamos una función que con esos parámetros nos devolviese un vector con posiciones y tiempos, esta librería devolvía un vector de Points que correspondía con la ruta más rápida entre los dos puntos.

Las primeras impresiones al usar esta función junto al algoritmo eran que la velocidad de cálculo de una ruta era algo lenta. Cada cálculo que se solicitaba a la librería tardaba unos 4 segundos, lo que provocaba que al usarla para resolver diferentes instancias de TSP y llevar a cabo cálculos de diferentes caminos mínimos, la ejecución del algoritmo fuera algo lenta.

Resolver cada instancia de un TSP implica en cualquier caso disponer de distancias o tiempos entre varios puntos, obtener esos datos de distancias o tiempos entre puntos es un proceso que solo se puede llevar a cabo conociendo la ruta, lo que implica hacer un uso muy intensivo de cálculos de las mismas.

Por ejemplo, desde un punto de vista de tiempo real en el sistema, solicitar los tiempos entre 13 puntos dados en un orden implicaba unos 52 segundos, tiempo asumible por el proyecto.

El problema reside en que calcular un camino mínimo no se basa en conocer el orden, sino en establecerlo, por lo que hay que probar diferentes combinaciones para dar con la mejor solución posible. Cada combinación implica una nueva ruta, lo que en tiempo de ejecución provocaba que el algoritmo que hiciera uso de esta librería fuese un sistema muy lento. En un caso como el del ejemplo mencionado anteriormente, conocer la distancia/tiempo de cada punto con respecto a todos los demás conllevaba un tiempo total de unos 10 minutos, tiempo que no es aceptable en ningún caso.

Aun así, se decidió seguir usando libosmscout realizando optimizaciones propias con las que se trató de mejorar la velocidad en la que se accedía a los datos de las rutas.

Una de estas optimizaciones consistió en evitar usar la librería para realizar resoluciones de las diferentes instancias de TSP en tiempo de ejecución, idea que se nos ocurrió gracias a la ayuda de José Alberto Verdejo López (profesor de la Universidad Complutense de Madrid). Esta idea derivó en la implementación de la cuadrícula o matriz (explicada en el apartado 1.3.1 de esta memoria): calculando la matriz una vez no era necesario volver a hacerlo en tiempo de ejecución.

Una vez que se tuvo claro que estructura de datos usar para guardar la información necesaria, se comenzó a calcular los datos, proceso que solo debería llevarse a cabo una vez.

Uno de los inconvenientes de este sistema era que hacía uso de demasiada memoria. Las dos matrices en total ocupaban 8 GB estáticos, que no se vaciaban durante la ejecución del servidor, ya que eran datos que iba a usar entre otros el proceso de resolver TSPs para cada autobús.

Analizando los datos guardados en el fichero .txt se observó un problema, ya que había demasiadas rutas que libosmcout no era capaz de calcular, lo que reducía enormemente la capacidad de funcionamiento de dos partes claves del sistema:

- El movimiento de los autobuses.
- Los cálculos para resolver distintas instancias de TSP.

Estas dos partes son vitales en la ejecución del servidor, por lo que, aun estando en el mes de marzo, se decidió buscar una alternativa lo más rápida posible. Las características básicas que se buscaron fueron:

- No fallar en ningún cálculo de ruta.
- Lograr que la velocidad de ejecución fuera bastante mayor de la actual.

Teniendo en cuenta esto, se halló una librería llamada OSRM, que parecía tener buena comunidad y un funcionamiento sencillo. Se encontraron ficheros compilados de la misma, lo que facilitó mucho el proceso para usarla junto a nuestro proyecto. La ejecución de la librería es en modo de servidor HTTP, recibe peticiones por GET y envía la respuesta con datos mediante un JSON, JSON que se puede ver en la siguiente figura:


```
{
    "waypoints": [
        {
            "location": [
                -3.703889,
                40.416667
            ],
            "name": "Calle Mayor",
            "hint": "IgUAgP___39KAAAAbQAAAAAAAAAAAAAASgAAAG0AAAAAAAAAAAAAAH8CAACve8f_m7VoAq97x_-btWgCAAAPALorEnU="
        },
        {
            "location": [
                -3.70031,
                40.414756
            ],
            "name": "Calle del Príncipe",
            "hint": "CAgAgP___38IAAAACgAAACABAAAkAAAACAAAAAoAAAAgAQAAJAAAAH8CAACqicf_JK5oApmJx_8jrmgCBwAPALorEnU="
        },
        {
            "location": [
                -3.694325,
                40.44074
            ],
            "name": "Calle de Fernández de la Hoz",
            "hint": "8jwAgP___38sAAAAZAAAAAAAAAAAAAAAALAAAAGQAAAAAAAAAAAAAAH8CAAALocf_pBNpAq2fx_-zE2kCAAAPALorEnU="
        }
    ],
    "routes": [
        {
            "distance": 4867.9,
            "duration": 574.7,
            "weight": 3530.7,
            "weight_name": "routability",
            "geometry": {
                "type": "LineString",
                "coordinates": [
                    [
                        -3.703889,
                        40.416667
                    ],
                    [
                        -3.703709,
                        40.416688
                    ],
```


Figura 5: Ejemplo de respuesta a una petición HTTP en formato JSON

Se ejecutó el servidor correctamente, sin ningún tipo de fallo. En nuestro servidor de C++ se instaló una librería para llevar a cabo peticiones HTTP desde C++ de un modo muy sencillo, la cual pertenece a las oficiales que ofrece Visual mediante el gestor de paquetes NuGet. Esta librería es cpprestsdk, en su versión 1.40.

Teniendo el servidor OSRM y un sistema sencillo para ejecutar peticiones HTTP, en un par de días se consiguió un sistema que devolvía rutas sin fallos impredecibles y, además, lo hacía en un tiempo considerablemente menor al ofrecido por libosmscout.

El proceso para calcular rutas es el que se describe en la siguiente figura:

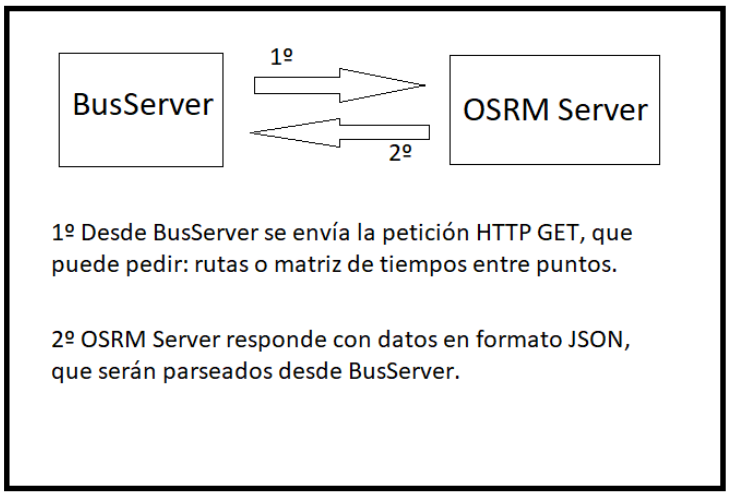

Figura 6: Proceso para el cálculo de rutas

2.1.4 Servidor para la gestión de peticiones

Para el servidor de gestión de peticiones, finalmente se decidió desarrollar una aplicación servidor en Node.js que cuenta con un motor de ejecución de JavaScript diseñado y utilizado por Google Chrome, de código abierto y multiplataforma. Está enfocado a la creación de servidores web altamente escalables y cuenta con un gran número de módulos implementados por terceros. Se escogió Node.js sobre otros frameworks web tradicionales como PHP o Django ya que la realización de múltiples tareas al mismo tiempo, como acceso a ficheros y bases de datos, sigue un modelo I/O asíncrono. Esto permite que el sistema destine menos recursos a la gestión de procesos y pueda atender más peticiones.

PHP y JavaScript, además, son lenguajes interpretados. Esta característica añade portabilidad a las aplicaciones desarrolladas y tiene repercusiones negativas en el rendimiento respecto a lenguajes compilados. Sin embargo, el motor V8 que utiliza implementa una estrategia de compilación Just-In-Time, permitiendo que las regiones de código en las que hay un nivel de tránsito alto sean compiladas para obtener una mejora de rendimiento.

Otros de los beneficios de Node.js es que se puede utilizar JavaScript como lenguaje de scripting en tu consola (como bash, perl, etc.). Dispone de NPM, gestor de paquetes de Node.js, además de muy buena documentación. Con esto ganaremos mayor limpieza, organización y simplificación en el despliegue de la aplicación, ya que únicamente tendremos lo que realmente nos hace falta para el proyecto. Cuenta con un gran número de módulos sobre webapps, enrutado, aplicaciones móviles u otras, implementados por terceros y de fácil instalación. Además, NPM también puede ser utilizado como gestor de dependencias para nuestro propio proyecto.

Con Node.js se dispone de toda la funcionalidad que necesitamos (acceso a ficheros, bases de datos y conexión con el cliente). Permite implementar fácilmente todas las funcionalidades de esta parte del proyecto. Está basado en eventos, y para utilizar Node.js no es necesario instalar ni configurar ningún servicio en el sistema. Basta con instalar el framework y éste se encargará del manejo de los procesos cuando ejecutemos nuestra aplicación servidor.

Al principio empezamos a desarrollar un servidor en C++ utilizando sockets, ya que podría ser mejor para el código de enrutamiento de C++ al estar todo en el mismo lenguaje. Pero los sockets se programan a bajo nivel, y actualmente hay mejores métodos para implementar un servidor, por lo que esa idea fue descartada.

Se ha decidido desarrollar el servidor en local, ya que, entre otros, disponemos de todo el sistema en local por motivos de velocidad al realizar las pruebas. Entonces para leer ya sea del fichero JSON o de la BBDD en local será más rápido y sencillo.

2.1.5 Base de datos

Se investigó la distribución de Apache XAMPP (X (para cualquier sistema operativo), Apache, MariaDB, PHP, Perl). Esta aplicación permite la gestión de base de datos MySQL para el almacenamiento de todos los datos relevantes. Este sistema permite almacenar las BBDD en un entorno local, lo que facilita el acceso a la información y minimiza los posibles retrasos causados a partir de los accesos a un sistema en un entorno remoto.

2.1.6 Aplicación móvil

La aplicación cliente se desarrolló en Android. Debatimos si desarrollar la aplicación con tecnologías web o en nativo Android. Nos decantamos por Android por diversos motivos. Por un lado, a nivel de usuario, la experiencia en una aplicación desarrollada en nativo es mejor que la de una aplicación web. Por otro lado, las aplicaciones web requieren conectividad, y en nativo la dependencia de conectividad es reducida.

En nativo se dispone de servicios avanzados como localización, acelerómetro, notificaciones... Además, la velocidad y respuesta en una aplicación nativa es más rápida que en web. Este factor es bastante importante en esta aplicación, ya que los usuarios esperan una respuesta de la aplicación inmediata. Por ello este factor dio muchos puntos a elegir desarrollar una aplicación nativa. Otro punto a favor es que en nativo no se dispone de límite de almacenamiento, mientras que en una aplicación móvil web sí.

La desventaja de desarrollar la aplicación en nativo es que únicamente sirve para una plataforma. Pero al elegir Android recogemos a más del 64% de los usuarios con dispositivos móviles (según Netmarketshare, 2017). Por otro lado, una vez desarrollada por completo la app Android, montado el servidor y BBDD, no sería muy costoso el desarrollo de la app nativa para iOS, ya que está todo preparado y la estructura de la app es clara.

Otras de las razones por las que hemos escogido Android y no por ejemplo iOS, es que el desarrollo de aplicaciones Android puede realizarse bajo cualquier sistema operativo (Windows, Linux o Mac). En cambio, el desarrollo de aplicaciones iOS sólo se realiza a través de sistema operativo Mac. Además, otra de las virtudes de Android es su facilidad de desarrollo, debido a su extensa documentación gracias a la comunidad web de Android.

Alguno de los problemas encontrados en el desarrollo de la app móvil fue la conexión con el servidor, ya que no se disponía de ninguna experiencia en ello, así que se tuvieron que investigar las mejores maneras de conectar con él. Se debe conectar la app con el servidor principalmente para conectar con la BBDD. Por motivos de seguridad de datos no se debe de hacer llamadas a la BBDD directamente desde la app (primero lo desarrollamos así, más adelante se cambió). Estudiando la documentación oficial de Android, se llegó a la conclusión de que actualmente la mejor manera (y más sencilla) de hacerlo es utilizando la librería llamada Volley (explicada con detalle más adelante).

También requirió bastante investigación realizar bastante investigación el sistema de notificaciones desde el servidor a la app. Se logró establecer las notificaciones de los cambios de rutas, notificando a cada usuario al dispositivo en el que está logueado, gracias a Google Cloud Messaging Client App para Android (explicado detalladamente más adelante y en el Anexo). Se tuvo que estudiar detalladamente su documentación para hacer uso correcto de su funcionamiento.

2.1.7 Interfaz web

Ha sido desarrollada en lenguaje PHP, Javascript, JQuery y HTML. Se decidió desarrollar la interfaz web en PHP ya que se disponía de experiencia en dicho lenguaje. Pero además tiene diversas ventajas.

PHP es de código abierto, es muy popular por lo que dispone de muchas referencias y guías disponibles en la web. Hay librerías muy útiles como la que se ha usado para la representación del mapa, OpenLayers que ofrece un API para acceder a diferentes fuentes de información cartográfica en la red: Web Map Services, Mapas comerciales Web Features Services, distintos formatos vectoriales, mapas de OpenStreetMap, etc.

Puede ser fácilmente insertado en HTML. Esto hace muy fácil convertir un sitio web estático existente en uno nuevo y más dinámico. Por lo que para desarrollar la interfaz es muy útil.

PHP es muy rápido de desarrollar y el tiempo de respuesta es rápido también. Dispone de múltiples extensiones y es extremadamente escalable.

El acceso a BBDD es sencillo y rápido. Algo muy necesario, ya que está constantemente realizando peticiones a la BBDD para mostrar los datos de usuarios y autobuses.

Principalmente se ha investigado más sobre la utilización de la librería OpenLayers que ha sido la más utilizada y de gran ayuda para mostrar la interfaz.

2.1.8 Fichero JSON datos de usuarios

Para la relación entre el cálculo de rutas y la app móvil se decidió que la aplicación de cálculo de rutas genere y actualice un fichero de formato JSON que mantendrá actualizado con los datos de ruta de los usuarios. Al tener actualmente el sistema en local, el fichero útil a la hora de notificar a los usuarios a tiempo real de los cambios sobre su ruta. JSON proporciona mayor velocidad, bastante

Además el formato JSON es muy sencillo basado en clave-valor para almacenar los datos bien estructurados y fácilmente legibles.

2.2 Fase de desarrollo

En la fase de desarrollo, se ha implementado todo el software necesario para el proyecto. Reflejo del estudio y análisis llevado a cabo durante la fase de investigación. Cada uno de los módulos ha sido implementado y probados por separado para después ser probados conjuntamente.

El método utilizado ha sido 'Cluster first, Route second', es decir, primero se organizan todas las nuevas peticiones recibidas por el servidor, para después aplicar el algoritmo genético elegido para solucionar el problema planteado y, finalmente, solicitar a la aplicación de cálculo de rutas el recorrido final.

El funcionamiento del algoritmo del sistema servidor (main en Routing.cpp) se puede comprender con este pseudocódigo:

```
While(true)
{
   usuarios = usuariosBBDD && usuariosIteracionAnterior
   cluster(usuarios)
   buscaTransbordos()
   usuariosIteracionAnterior = unirUsuariosDesasignados(usuarios)
   buscaTransbordos()
}
```

Se cargan los nuevos usuarios de la BBDD y los no asignados de la iteración anterior y se tratan de asignar en el cluster, analizando si existe posibilidad de transbordos. Para saberlo llamamos a `buscaTransbordos()`, función que busca puntos de cruce de las rutas actuales de los autobuses. Se unen los usuarios en grupos y se resuelven diferentes instancias de TSP, dando lugar a una serie de usuarios asignados a buses y otros tantos sin asignar, que pasarán a la siguiente iteración. Por último, se vuelven a buscar transbordos.

La simulación del movimiento de autobuses se ejecuta en paralelo en otro hilo, haciendo uso de las mismas variables del algoritmo.

2.2.1 Algoritmo de enrutamiento

Desarrollar el algoritmo de enrutamiento ha sido una de las tareas principales de este proyecto. El primer desarrollo que llevamos a cabo fue el del algoritmo voraz, con una implementación muy sencilla.

Cada autobús disponía de una cola de prioridad, con prioridades variables, siendo cada elemento el usuario y su prioridad la distancia desde el autobús al mismo.

Este sistema ofrecía una implementación muy sencilla, pero su ejecución conllevaba soluciones peores que el algoritmo genético.

Nuestro sistema de enrutamiento actual funciona del siguiente modo:

Antes de que se lleve a cabo la búsqueda de soluciones de instancias de TSP, se deben separar a los usuarios en autobuses. Este proceso comienza en Cluster.cpp, en la función:

```
map<int,Evento> unirUsuariosDesasignados(map<int, Evento> & usuarios, Matriz<Parada> & matrizParadas, map<int, Autobus> & autobuses)
```

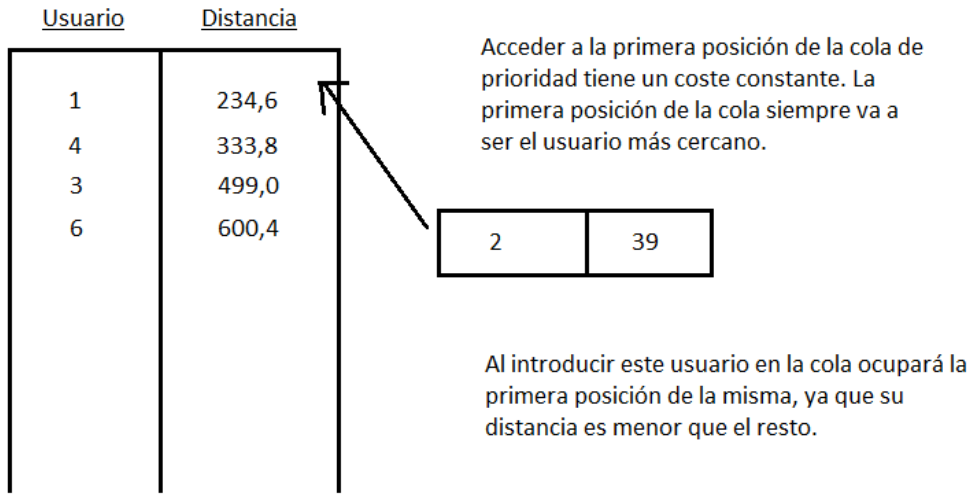

Figura 7: Ejemplo de uso de la cola de prioridad

Todos los usuarios, tanto como los que ya tienen autobús asignado en la fase de clúster como los que no, se clusterizan en función de origen y destino, obteniendo varios grupos diferentes de usuarios.

Posteriormente, para cada grupo se ejecuta el método:

```
map<int, Evento> caminoCluster(map<int, Evento> & usuariosCluster, map<string, Parada>& darsenas, int& numeroAutobuses, Matriz<Parada> & matrizParadas)
```

En esta función se establece si para el grupo de usuarios que llega merece la pena sacar un autobús nuevo de la dársena más cercana o usar uno existente. Posteriormente se introducen en el autobús elegido los usuarios y las paradas con sus TW, y se llama a:

```
Bool IniciarTSP(Autobus &bus, Matriz<Parada> & matrizParadas, vector<int> & usuariosNoRecogidos)
```

`IniciarTSP()` crea los datos necesarios y llama a la resolución de un instancia del problema TSP con el algoritmo genético. En primer lugar, se crea un grafo cuyos nodos representan al autobús y al resto de usuarios elegidos. Para poder solucionar el TSP necesitamos el tiempo de rutas entre estos puntos, por lo que se solicitan esos tiempos a OSRM, que devuelve una matriz con el tiempo de ruta entre todos los elementos. También se crea una se crea una estructura de datos adicional, para guardar correspondencia entre nodos y paradas llamada paradaGrafo.

El método devuelve true si han sido recogidos todos los usuarios.

Con el grafo creado se llama a:

```
void Genetic::run(vector<Parada::ParadaNodo> & paradasNodos, Matriz<Parada> & matrizParadas, Autobus & bus, vector<int> & usuariosNoRecogidos, vector<double> & ordenRecogidaReturn)
```

Recibe:

- `vector<Parada::ParadaNodo> paradaGrafo`: estructura de datos que asocia los nodos del grafo (variables de tipo double) a paradas.
- `Matriz<Parada> matrizParadas`: matriz de paradas.
- `Autobus bus`: bus actual, puede ser uno existente o uno nuevo que salga de dársena.
- `vector<int> & usuariosNoRecogidos`: almacena las ids de usuarios no recogidos a tiempo.
- `vector<double> & ordenRecogidaReturn`: almacena el mejor camino que recorre el grafo.

Con estos datos de entrada se resuelve la instancia actual del TSP con un algoritmo genético que funciona del siguiente modo, dentro de tsp.cpp:

Se crea la población inicial en el método `initialPopulation()`. Para crear soluciones siempre factibles (soluciones que mantienen un orden de primero recogida, después destino, y que llega en los TW dados a una serie de usuarios) llamamos a la función `isValidSolution()`, que internamente trata la solución para que mantenga un orden coherente y establece en ese orden a cuantos usuarios es capaz de recoger, devolviendo el tiempo total de la ruta. Prim ero comprueba si las soluciones son válidas, y si no, las adapta para que lo sean.

Con una población de soluciones válidas ya creadas se trabaja sobre las mismas un número de generaciones dado, en nuestro caso 10. En cada iteración se escogen pares pertenecientes a la población y se mezclan en la función `crossOver`(), donde se las aplica cierto componente de aleatoriedad, lo que da lugar a soluciones que son factibles y prácticamente siempre diferentes. Al igual que en la población inicial, estas soluciones son adaptadas para que cumplan los principios de factibilidad; orden y llegada a tiempo a los TW.

Finalmente tenemos una serie de soluciones, de las cuales nos quedamos con la que más usuarios alcanza a tiempo, y dentro de las que tengan el mismo número de usuarios, la que lo haga en un menor tiempo total.

Si la solución actual es mejor a la anterior para el autobús (en número de usuarios), se enruta al mismo para que su ruta pase por los sitios indicados. Si existían usuarios comprometidos en la iteración actual que ya no son recogidos, pasarán a ser asignados a otro autobús. Para los usuarios no existe consecuencia visible de esto, ya que hasta que la iteración del while principal no acaba, no recibirán la notificación de qué asignación tienen.

Los usuarios sin recoger se pasan al bucle main, para que en la siguiente iteración el Cluster trate de enrutarlos, sino es capaz de hacerlo vuelven al método para unir usuarios desasignados y son tratados en cualquier caso (si es posible con un autobús existente, sino con uno nuevo que salga de dársena).

2.2.1 Algoritmo de agrupamiento

El desarrollo del algoritmo de agrupamiento se realizó de forma iterativa. Se comenzó con un desarrollo simple, los *eventos* que llegaban al sistema se trataban de forma individual y cada uno de ellos era asignado a la parada de la rejilla correspondiente, siendo elegida siempre la esquina superior izquierda de la cuadrícula en la que está ubicada la parada de origen de la solicitud del usuario. Esto se consigue gracias a un método que, a partir de las coordenadas, de origen y de destino, del usuario y la *matriz*, calcula la parada (formada por la fila y la columna dentro de la *matriz*) de origen y destino.

Tras tener una base sólida, sobre la cual cada usuario era añadido a la parada correspondiente con respecto a su ubicación, se comenzó a implementar un sistema para agrupar a la mayor cantidad de usuarios dentro de un radio que se tomó como aceptable. Este radio es de 500m. Antes de añadir al usuario que está siendo tratado, se comprueban todas las paradas dentro de ese radio en busca de usuarios ya asignados que compartan ventana de tiempo y dirección con el usuario que se está tratando. De cumplirse estas condiciones, el autobús que contiene en su recorrido la parada del usuario ya establecido podrá llevar a su destino al usuario nuevo.

En el caso de que no haya ningún usuario que cumpliese dichas condiciones, al nuevo usuario se le añade a la mencionada parada superior izquierda de su celda en la rejilla, y se comprueban todas las rutas de todos los *autobuses* en busca de los que pasen a una distancia inferior a 500m del origen del usuario.

Funcionalidad sin transbordos: a lo mencionado hemos de añadirle la restricción de que, en el resto de la ruta de dicho autobús, algún punto esté a menos de 500m del destino del usuario. Si tras realizar dicha búsqueda ningún autobús fuera seleccionado como candidato, el usuario quedaría en espera hasta completar un segundo paso por el algoritmo o para ser enviado al algoritmo de enrutamiento como usuario sin asignar, para ser tratado y asignado a un autobús en él. Si por el contrario algún autobús cumplía los requisitos para recoger al usuario, este recibiría la parada de origen y destino del usuario, que quedarían establecidas como compromisos para el autobús.

Funcionalidad con transbordos: si el autobús que contiene en su ruta un punto suficientemente cercano al origen del usuario no contiene ningún punto cercano al destino, el sistema de *transbordos* se pone en marcha. Se buscarán, dentro del *Mapa de transbordos*, todos los autobuses que tienen un posible transbordo con el autobús candidato y se comprobarán las rutas de estos en busca de generar el posible cambio de línea para llegar al destino. Sólo se ha contemplado un grado de transbordo, es decir, solo cambiar de línea una vez independientemente del *evento*.

A continuación se detalla el algoritmo a un nivel más bajo.

La función que engloba al algoritmo es la siguiente:

```
void clusterPorPuntos100X100(Evento &usuario, Matriz<Parada> &matriz, map<int, Autobus> &autobuses,
    vector<int> &paradasSinAutobus, bool &rutaModificada, int vueltas, map<int, vector<Trasbordo>> &mapaTrasbordoCliente)
```

Además de los atributos ya mencionados anteriormente, el vector *paradasSinAutobus* almacena todas las paradas de los eventos que quedan sin asignar a un autobús existente para que, posteriormente en el algoritmo de enrutamiento, se traten. El 'bool' rutaModificada hace referencia a una modificación de alguna de las rutas existentes para un autobús. Esto sirve para repetir el proceso una vez más o no.

Lo primero que se lleva a cabo en el algoritmo es buscar la ubicación del usuario en la *matriz* y las paradas dentro del radio aceptable.

```
//calcula latitud y longitud del usuario
buscaZona(latitud, longitud, matriz, ubicacion);

//busca paradas con la misma TW que el usuar
paradaMismasHoras(matriz, latitud, longitud, usuario.getHora(), diccionario);
```

Después, se ordenan los *autobuses* por cercanía a dicho punto para buscar primero sobre sus rutas, ya que la probabilidad de que estos autobuses cumplan las restricciones son mayores.

Posteriormente, para cada autobús se comprueba toda su ruta con:

```
idParadaRecogida = compruebaPuntoConRuta(autobuses[autobusCercano[i].id].getPuntoRuta(), usuario.getCoordInicial(),
    matriz, usuario.getHora(), horaFinUsuario, puntoMejorRuta, false, desviado, distanciaAndando, puntoMejorRutaDestino);
```

Si idParadaRecogida tiene un valor distinto de -1, significa que el autobús es válido y repetiremos la función para buscar si es igualmente válido para la parada de destino; si no, ese autobús se descarta y se analiza el siguiente.

```
idParadaDestino = compruebaPuntoConRuta(autobuses[autobusCercano[i].id].getPuntoRuta(), usuario.getCoordFinal(),
    matriz, usuario.getHora(), horaFinUsuario, puntoMejorRuta, true, desviado, distanciaAndando, puntoMejorRutaDestino);
```

Si idParadaDestino tiene un valor -1, se procederá a buscar un autobús con el que hacer transbordo para que el usuario pueda llegar al destino; si no, significa que dicho autobús puede recoger y dejar al usuario.

Cada ruta válida se almacena en un vector de soluciones posibles, para posteriormente ser elegida la mejor de ellas. La mejor será la que suponga menor desvío del autobús y menor distancia de recorrido (caminando) de un usuario hasta su destino.

2.2.3 Cálculo de rutas

El desarrollo de un sistema de cálculo de rutas no es parte de este proyecto como tal. Implementar un sistema de cálculo de rutas en sí es un proyecto a desarrollar externo a este, por lo que trabajamos con una librería ya existente (OSRM) y guardamos los datos que nos devuelve para usarlos en nuestro algoritmo.

Independientemente de la librería usada, se necesitaba que la ruta devolviera no solo la posición, sino también el tiempo total de la ruta, y los tiempos relativos a cada punto. La estructura de datos utilizada para guardar los mismos, es un vector std de c++, que almacena en orden elementos *PuntoRuta* con la siguiente estructura:

```
struct PuntoRuta
{
        Point   ubicacion;
        Hora    hora;
        int     posicionParada;
};
```

Con estos datos se conocen la posición (mediante la latitud y la longitud de cada punto de la ruta), la hora a la que se pasa por la misma y la posición en el vector de paradas del autobús, si el *PuntoRuta* dado es una parada.

En primer lugar, es necesario ejecutar OSRM, aplicación de consola que se queda abierta recibiendo peticiones.

Hay dos tipos de cálculos que se solicitan al servidor:

- Rutas completas.
- Matriz de tiempos entre varios puntos.

Todo el código que asocia el servidor de rutas y nuestro algoritmo se encuentra en Route.h y Route.cpp.

Rutas completas

El código para calcular una ruta completa entre dos puntos se encuentra en la función:

```
vector<Autobus::PuntoRuta> rutaTiempoEntreDosPuntosHTTP(Point & start, Point & end, int hora, int minuto, double & tiempo)
```

Recibe:

- Point start: punto de origen desde el que se quiere calcular la ruta.
- Point end: punto de destino al que se quiere calcular la ruta.
- Int hora: hora desde la que empieza a calcular la ruta.
- Int minuto: minuto desde el que empieza a calcular la ruta.
- Double & tiempo: variable que devuelve el tiempo total en segundos de la ruta.

Devuelve:

- Vector<Autobus::PuntoRuta>: vector que contiene en orden todos los elementos PuntoRuta que definen la ruta.

Dentro de la función calculamos el string que va a formar la petición HTTP:

```
const string a = "http://localhost:5000/route/v1/driving/" + startLon + "," + startLat + ";" + endLon + "," + endLat + "?steps=false&geometries=geojson&annotations=duration&overview=full";
```

Los parámetros de la petición siguientes al '?' corresponden a opciones de OSRM, en particular:

- steps = false:
- gemotries = geojson:
- annotations = duration
- overview = full:

En concreto, la versión actual del sistema para calcular las rutas de los autobuses usa una función que no solo recibe un punto origen y uno destino, sino que recibe varios en un vector. Todos ellos componen las paradas que tiene que llevar a cabo el autobús:

```
vector<Autobus::PuntoRuta> rutaTiempoEntreVariosPuntosHTTP(vector<Point> & listaPuntos, vector<double> & tiempoEntreDestinosLocal, int hora, int minuto, double tiempo)
```

Internamente, el funcionamiento de la función es igual al definido anteriormente. La única diferencia es que se agregan todos los Points recibidos a la consulta en el orden que se ha calculado previamente:

```
for (Point p : listaPuntos)
{
        a += std::to_string(p.GetLon());
        a += ",";
        a += std::to_string(p.GetLat());
        a += ";";
}
```

Matriz de tiempos entre varios puntos

Uno de los puntos positivos de usar OSRM como sistema para calcular rutas es que, además de calcular rutas, permite conocer de un modo muy rápido una matriz de tiempos que corresponde al tiempo de la ruta entre los valores pasados a la consulta.

En una consulta con 3 puntos: A, B, C tenemos el tiempo que se tarda en llegar de cada punto a cualquier otro:

A → B B → A

A → C C → A

B → C C → B

Internamente, la petición se lleva a cabo con la función:

```
vector<vector<double>> matrizEntreVariosPuntosHTTP(vector<Point> & listaPuntos)
```

Recibe:

vector<Point> listaPuntos: vector con todos los puntos que componen la consulta.

Devuelve:

vector<vector<double>>: vector de vectores que contiene el tiempo que de cada a punto al resto.

La diferencia en la petición es que tiene una estructura que pide los datos en tipo table y añade a la consulta todos los puntos recibidos por parámetro.

```
string a = "http://localhost:5000/table/v1/driving/";
for (Point p : listaPuntos) {
        a += std::to_string(p.GetLon());
        a += ",";
        a += std::to_string(p.GetLat());
        a += ";";
}
```

2.2.4 Servidor para la gestión de peticiones

El servidor está escuchando constantemente cualquier petición que puede hacerse desde la app móvil. Y, además, en otro hilo lee constantemente el fichero JSON por si hay cualquier modificación de rutas para notificar a los usuarios.

El servidor recibe de la app cualquier petición. Este lo procesa, realiza las funciones/llamadas a la BBDD necesarias y responde a la app con la información si la operación ha tenido éxito o no.

Express es un módulo de NPM que simplifica el proceso de implementar un servidor web. Implementar un servidor HTTP utilizando la funcionalidad básica de Node.js exige que el desarrollador implemente la lógica de enrutado, la gestión de cookies y filtrado de las peticiones por POST/GET/UPDATE/DELETE/PUT cuando sea necesario. Express proporciona toda esta lógica sin necesidad de tener que implementarlo. Tan sólo hay que asociar la app, que es como se denomina en este módulo al objeto JavaScript que engloba toda la funcionalidad del servidor, a un puerto de red abierto y configurarla como sea necesario.

Las dos dependencias/paquetes destacadas que se han utilizado son mysql y node-gcm. La dependencia mysql ha servido para conectar el servidor a nuestra BBDD, y así poder realizar las llamadas y peticiones que necesitamos. Se decidió realizar la conexión cliente- servidor con la app móvil a través del protocolo HTTP. La dependencia node-gcm se encarga del sistema de notificaciones entre cliente-servidor, es decir, cuando el servidor debe notificar al cliente de cualquier novedad, aviso, etc. se le envía un mensaje mediante HTTP con toda la información. Esta dependencia y la ayuda de Firebase Cloud Messaging de Google han permitido notificar a los usuarios cuando hay una nueva ruta disponible o cualquier modificación de su ruta. GCM significa Google Cloud Messaging, aunque actualmente se llama Firebase Cloud Messaging (FCM). FCM es una solución multiplataforma que permite enviar, de forma gratuita y segura, mensajes y notificaciones a través del protocolo HTTP y la app cliente. Es necesario crear una nueva clave de API de servidor e ID de remitente. Estas dos claves se encuentran en el fichero del servidor constants.js (fichero donde se encuentran las claves y textos constantes que se van a utilizar en el servidor).

Para generar el fichero Excel con todas los tiempos y distancias de las rutas de los casos de pruebas, se ha creado una función que lee todos los puntos (en algunas pruebas aleatorios) generados de dos en dos (una ruta, origen y destino) de un fichero .txt. Al leer los dos puntos, se realiza una petición HTTP a Google Maps Directions API (explicado en el Anexo) pasándole por parámetro las coordenadas de origen, destino, el modo de transporte (en nuestro caso, el único modo de transporte es en autobús) y la API Key de nuestra aplicación web. Esta petición nos devolverá una respuesta en formato JSON con todos los datos de la ruta. Únicamente se queda con la distancia total y el tiempo de viaje. Estas dos variables, obtenidas junto con las coordenadas de origen y destino, son insertadas en una variable de tipo JSON. Al contener el JSON toda la información de todas las rutas se genera a partir del fichero Excel toda la información recopilada necesaria para realizar la comparación entre los datos del sistema estático y los resultados de nuestro posterior estudio (más adelante se verá la comparativa).

2.2.5 Base de datos

El desarrollo de la BBDD se basa en una implementación sencilla de MySQL proporcionada por el software XAMPP. La interacción con la base de datos es muy sencilla, por medio de una interfaz web accesible a través del navegador.

En el servidor de C++ era necesaria una librería para interactuar con mysql, por lo que incluimos mysqlcppconn.lib. Creamos una clase (BaseDeDatos.cpp) encargada de realizar CRUD (lectura, escritura, edición y borrado) de las diferentes entidades que tenemos. Para llevar a cabo las consultas hacemos uso de consultas preparadas del tipo:

"INSERT INTO eventos (idUsuario, coordX, coordY, coordXFinal, coordYFinal, tratado, hora, minutos) VALUES (?,?,?,?,?,?,?,?)"

En la consulta tenemos una serie de parámetros que serán los que establecemos dinámicamente en la llamada al método.

Mediante un patrón singleton, nos aseguramos de que si existe una instancia de conexión con la BBDD se use la misma, y si no creamos una nueva. Esto reduce la sobrecarga de la BBDD, teniendo como máximo una sola conexión en todo momento.

La BBDD está compuesta por varias tablas donde se guarda toda la información sobre los usuarios, los dispositivos logueados, los eventos que generan y la ubicación actual de los autobuses. En la siguiente Figura se muestra el esquema de la BBDD:

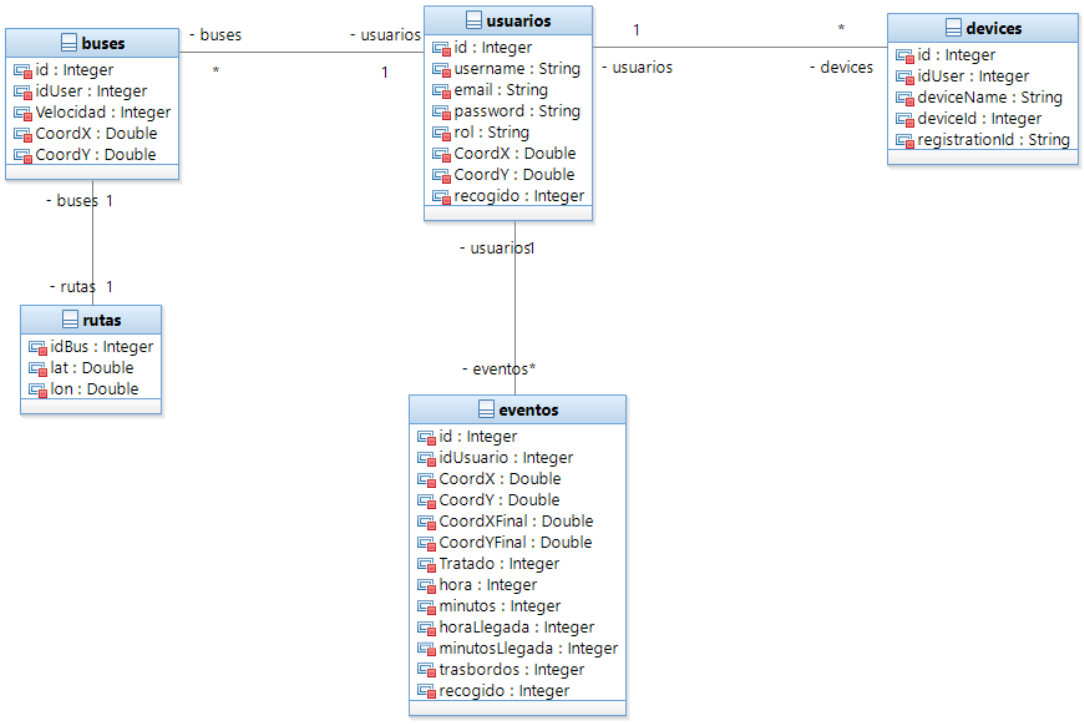

Figura 8: Esquema de la BBDD

Para gestionar la BBDD y ver la información, crear tablas, etc. más cómodamente hacemos uso de phpmyadmin.

2.2.6 Aplicación móvil

Para transmitir los datos en la red cliente/servidor se hace uso de Volley.

Volley es una librería HTTP que hace que la creación de redes para las aplicaciones Android sea más fácil y rápida. Volley nos ofrece los siguientes beneficios, por lo que hemos decidido usarla:

- Automatización de las solicitudes en red.
- Múltiples conexiones en red simultáneas.
- Memoria caché de disco transparente y memoria con coherencia de caché HTTP estándar.
- Soporte para priorización de solicitudes.
- API de solicitud de cancelación. Puede cancelar una sola solicitud o puede establecer bloques o varias solicitudes para ser canceladas.
- Facilitación de personalización.
- Muy buena ordenación de funciones, lo que da mayor facilidad a la hora de crear la interfaz de usuario con los datos asíncronos extraídos de la red.
- Herramientas de depuración y rastreo.
- Dispone de buena documentación muy útil en la página de Android developer.
- Volley se integra fácilmente con cualquier protocolo y puede hacer uso de cadenas de texto, imágenes y JSON.

La aplicación realiza varios tipos de peticiones HTTP: GET, POST, PUT y DELETE.

Las peticiones GET se usan en la aplicación al pedir la información de ruta del usuario, enviar por parámetro el id de usuario y recibir en la respuesta la hora de llegada al origen y destino del autobús y el autobús que recogerá al usuario. Además la información de ruta para el conductor del autobús se hará por medio de la petición GET, pasando por parámetro el id del autobús y devolviendo la ruta. Las peticiones POST se utilizan para enviar las nuevas peticiones de las rutas deseadas por los usuarios, con respuesta por parte del servidor con la ruta establecida (como la petición GET del usuario). Las peticiones PUT se utilizan para actualizar un recurso del servidor, en este caso para establecer el id de usuario conductor en un autobús. Las peticiones DELETE sirven para eliminar un recurso del servidor. Por ejemplo, al hacer logout la app realiza una petición DELETE para eliminar el dispositivo logueado de la BBDD, y así no enviarle nuevas notificaciones.

Para desarrollar la app móvil se ha utilizado Android Studio, un entorno de desarrollo integrado (IDE) desarrollado por JetBrains y en colaboración con Google, que utiliza una licencia de software libre. En la Figura 12 podemos ver una vista del entorno. Está programado en Java y es multiplataforma. Android Studio nos proporciona muchas ventajas a la hora de desarrollar, como por ejemplo la vista en tiempo real de los layouts a la hora de editar una vista, o dividir nuestro proyecto por módulos sin necesidad de estar cambiando el espacio de trabajo. Utilizamos Gradle de Android Studio, que es una herramienta integrada que nos ayuda y automatiza la construcción de proyectos Android. Facilita mucho desplegar nuevas librerías en el proyecto, pruebas o tareas de compilación. Una de sus principales características es la de verificar si hubo algún cambio en el código fuente desde la última compilación. En el caso de que hubiera cambios, recompila todo el código fuente, y en caso contrario no realiza nada.

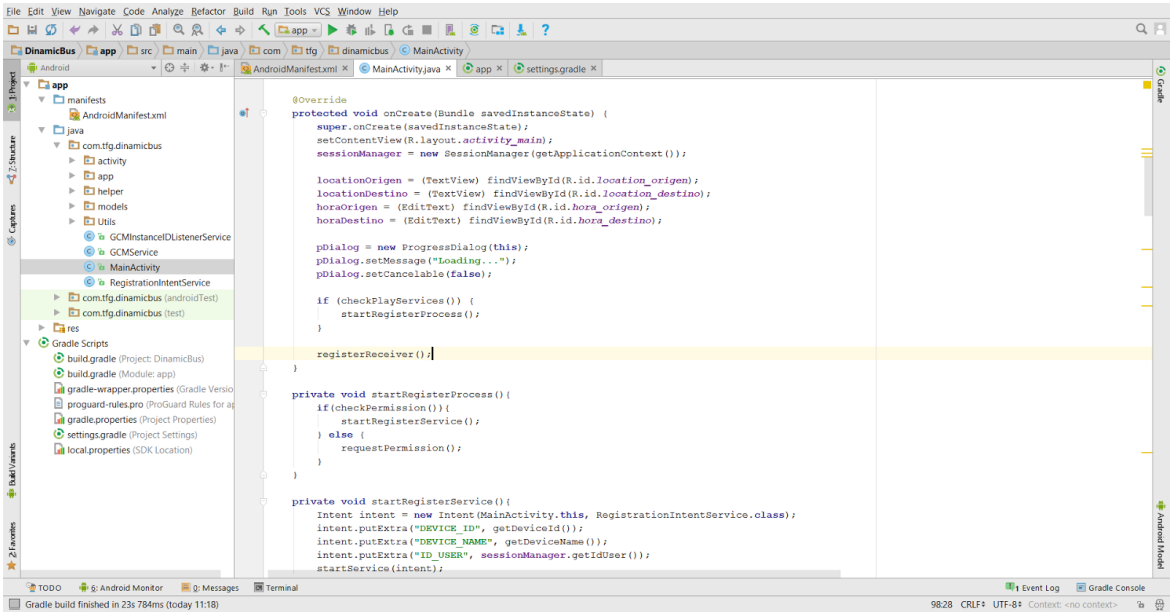

Figura 9: Vista del entorno de desarrollo Android Studio

La estructura de un proyecto Android usando Gradle contiene los siguientes ficheros:

Settings.gradle: Unifica todos los subproyectos dentro de este fichero, único para todo el proyecto, para que podamos acceder desde cualquier subproyecto a la información o funcionalidad que nos aporta otro subproyecto que hayamos añadido.

Build.gradle: En este fichero se escriben las dependencias con otras librerías o la versión del sdk dónde nuestro proyecto va a compilar. En la Figura 13 se muestra este fichero.

AndroidManifest.xml: Proporciona información esencial sobre nuestra aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la app.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"
    defaultConfig {
        applicationId "com.tfg.dinamicbus"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.google.android.gms:play-services:10.2.4'
    compile 'com.android.support:design:25.3.1'
    compile 'com.jakewharton:butterknife:6.1.0'
    compile 'com.mcxiaoke.volley:library-aar:1.0.0'
    compile 'com.android.support:support-v4:25.3.1'
    compile 'com.squareup.retrofit2:retrofit:2.0.0'
    compile 'com.squareup.retrofit2:converter-gson:2.0.0'
    compile "com.google.firebase:firebase-messaging:9.6.1"
    compile 'com.google.firebase:firebase-core:9.6.1'
    compile "com.google.android.gms:play-services-gcm:10.2.4"
}

apply plugin: 'com.google.gms.google-services'
```

Figura 10: Build.gradle del proyecto

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tfg.dinamicbus">

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <permission
        android:name="com.learn2crack.gcmdemo.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
    <uses-permission android:name="com.learn2crack.gcmdemo.permission.C2D_MESSAGE" />

    <application
        android:name=".app.AppController"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Dinamic Bus"
        android:theme="@style/AppTheme">
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
            android:name=".activity.LoginActivity"
            android:label="Dinamic Bus">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

Figura 11: AndroidManifest.xml del proyecto

Para el sistema de notificaciones, se hace uso de Google Cloud Messaging Client App para Android. Gracias a GCM, el servidor envía las notificaciones necesarias al cliente, tales como los cambios de ruta (se explica el funcionamiento de GCM en el Anexo). La aplicación hace uso de sesiones de usuario. Esto sirve para que el usuario no tenga que hacer el login en la app cada vez que la inicia. Se ha creado una clase llamada SessionManager que se encarga de todo el sistema de sesión de usuario. Guarda si hay un usuario logueado, el id del usuario logueado, su rol y el id del autobús que conduce si se trata de un conductor. La app está conectada con el servidor mediante la red local, donde se encuentra el servidor. Al estar en una red local, el servidor limita enormemente la escalabilidad del sistema, por lo que esto no sería viable para un sistema real. Para un sistema real habría que desplegar el servidor en red remota, para que la app se pueda conectar al servidor desde diferentes dispositivos y conexiones de Internet, ya que si está en local, el dispositivo únicamente se podría conectar al servidor si está en su misma red.

Por lo tanto, para que estén conectados el servidor y el dispositivo donde se encuentra la app, ambos deben estar conectados a la misma red Wifi. Hay que indicar la IP del servidor a la app. Se obtiene esta información en la consola mediante el comando “ipconfig” y apuntando la IP obtenida de “Dirección IPv4”.

Se dispone de una clase Const.java (Figura 15) donde almacenamos variables de tipo String que van a ser utilizadas como url para realizar las llamadas al servidor como HTTP, dónde se indica la ip del servidor local, el puerto y la función que se llama. Y así poder reutilizar código y no repetir al hacer la misma llamada desde diferentes puntos de la app.

Const.java

```
package com.tfg.dinamicbus.Utils;

public class Const {
    public static final String URL_ROUTE_REQ = "http://192.168.1.36:5000/info-ruta";
    public static final String URL_INFO_USER_REQ = "http://192.168.1.36:5000/info-usuario";
    public static final String URL_POST_EVENTO_REQ = "http://192.168.1.36:5000/enviar-evento";
    public static final String URL_POST_SIGNUP = "http://192.168.1.36:5000/nuevo-usuario";
    public static final String URL_POST_LOGIN = "http://192.168.1.36:5000/login";
    public static final String URL_HOME = "http://192.168.1.36:5000/";
    public static final String URL_DELETE_DEVICE = "http://192.168.1.36:5000/delete_device";
    public static final String URL_PUT_USERBUS = "http://192.168.1.36:5000/put_user_bus";
}
```

Figura 12: Clase Const.java

Android proporciona una alternativa para el diseño de interfaces de usuario: los ficheros de diseño basados en XML. Son usados en nuestro proyecto para todas las interfaces de usuario.

XML, traducido como "Lenguaje de Marcado Extensible", es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Por ejemplo, aquí mostramos parte del fichero activity_info.xml que muestra la información de ruta del usuario junto con el mapa:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map_info"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activity.InfoActivity"/>
```

2.2.7 Interfaz web

Desarrollada en PHP, Javascript, JQuery, Ajax y PHP, además de un fichero HTML que muestra los datos obtenidos haciendo los cálculos necesarios y peticiones a la BBDD.

Principalmente la estructura se dispone de un fichero principal, 4 ficheros que son llamados por el principal para obtener datos de la BBDD y el fichero HTML.

Se han usado las librerías OpenLayers, jQuery y Bootstrap.

OpenLayers ha permitido que se muestre el mapa de Madrid con exactitud. Haciendo las peticiones necesarias a la BBDD se obtienen las coordenadas de los usuarios, sus destinos, la posición actual de los autobuses y sus rutas. Para cada elemento que queremos añadir creamos una nueva Feature, le añadimos el estilo deseado con la imagen adecuada y lo incluimos en el mapa. Para pintar la línea de ruta que van a realizar los autobuses se utiliza una polyline.

jQuery es una biblioteca multiplataforma de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

Bootstrap es un framework para diseño de aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript.

2.3 Fase de Pruebas

En este apartado se muestran las pruebas que se han realizado sobre el sistema de redes de autobuses dinámicas en las que hemos comparado el rendimiento de dicho sistema con el de la red (estática) de líneas de autobús existente.

Para ello se han realizado diferentes casos de prueba, comparado los mismos casos con la red de autobuses actual.

Cada caso de prueba consta de mil usuarios y rutas en diferentes puntos de la zona de Madrid, donde hemos ubicado el estudio. Para cada ejecución del caso de prueba con unos parámetros dados hemos llevado a cabo ocho repeticiones.

La zona de Madrid donde se ha realizado el estudio (Figura 16) forma un rectángulo con los siguientes límites (en latitudes y longitudes de la Tierra):

- Límite norte: 40.48710200
- Límite sur: 40.37519088
- Límite oeste: -3.75345700
- Límite este: -3.64154588

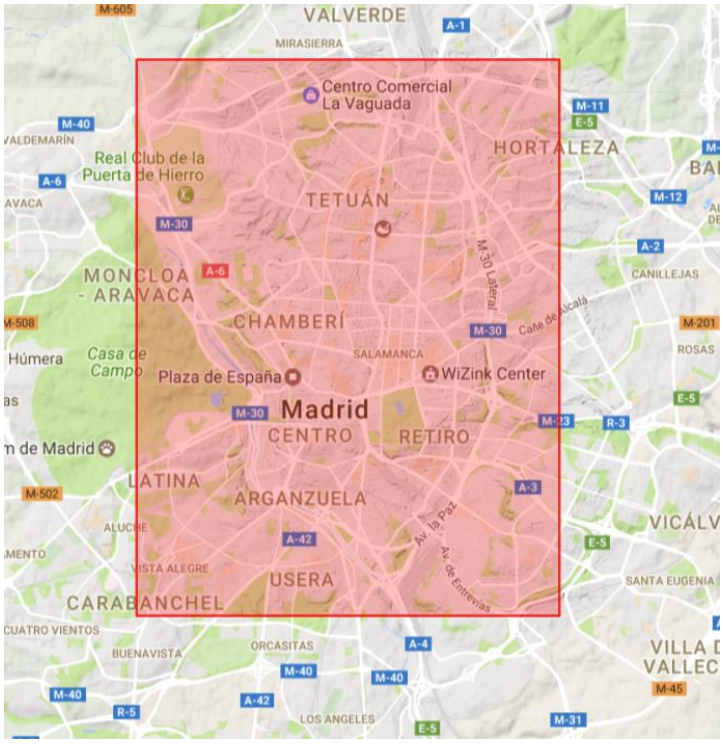

Figura 13: Zona de Madrid donde se ha realizado el estudio

2.3.1 Casos de prueba

2.3.1.1 Punto caliente en el centro de Madrid

Concentramos una zona determinada en el centro de Madrid. Y para usuarios dentro de dicha zona, creamos destinos totalmente al azar hacia fuera del "punto caliente". Para este test generamos 1000 usuarios y hacemos uso de 80 autobuses.

El rectángulo rojo es la zona de origen de los usuarios que se desplazarán hacia cualquier dirección dentro del mapa.

Los límites del punto caliente son:

- Límite norte: 40.424086
- Límite sur: 40.404283
- Límite oeste: -3.712565
- Límite este: -3.688974

Figura 14: Prueba punto caliente en el centro de Madrid

Los resultados obtenidos en este caso fueron los siguientes:

Variables	Sistema estático	Sistema dinámico	Sistema semi dinámico
Número de autobuses utilizados	187	187	187
Tiempo en el autobús medio	56 minutos	69 minutos	61 minutos
Tiempo en el autobús peor	93 minutos	84 minutos	102 minutos
Media de transbordos	0.42	0.08	0.23
Tiempo de espera medio	13 minutos	16 minutos	18 minutos
Tiempo de espera peor	25 minutos	20 minutos	25 minutos
Varianza sobre el tiempo medio en el autobús	2 minutos	3 minutos	1 minuto
Número de repeticiones del experimento	8	8	8
Rango horario	11:00 – 14:00	11:00 – 14:00	11:00 – 14:00
Número generaciones algoritmo genético	-	4	4
Ventana de tiempo para recogida	-	20	20
Ventana de tiempo para destino	-	100	100

2.3.1.2 Cuatro puntos formando un rombo

Establecemos cuatro zonas con tamaños y distancias unas de otras diferentes. Para este test generamos 1000 usuarios y hacemos uso de 80 autobuses.

Generamos rutas de unas zonas a otras, con mayor posibilidad de generar transbordos para comparar su eficacia.

Los límites de las cuatro zonas son:

Izquierda Límite norte: 40.41190200 Límite sur: 40.40919088 Límite oeste: -3.72040700 Límite este: -3.712965	**Derecha** Límite norte: 40.41710200 Límite sur: 40.40519088 Límite oeste: -3.679974 Límite este: -3.64954588
Abajo Límite norte: 40.398283 Límite sur: 40.39519088 Límite oeste: -3.70145700 Límite este: -3.68854588	**Arriba** Límite norte: 40.46710200 Límite sur: 40.457086 Límite oeste: -3.69845700 Límite este: -3.67154588

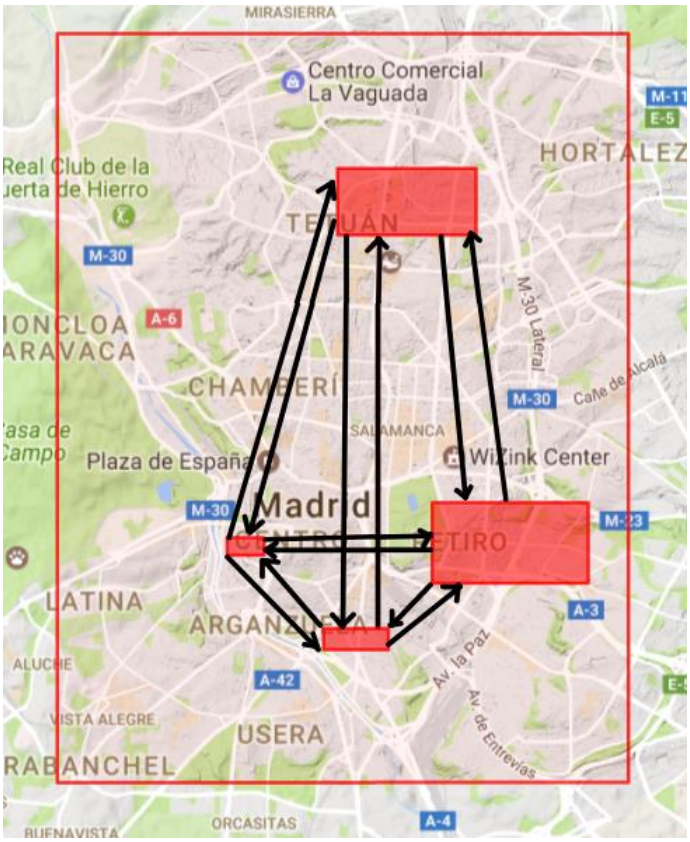

Figura 15: Prueba cuatro puntos formando un rombo

Los resultados fueron los siguientes:

Variables	Sistema estático	Sistema dinámico
Número de autobuses utilizados	115	115
Tiempo en el autobús medio	53 minutos	48 minutos
Tiempo en el autobús peor	68 minutos	98 minutos
Media de transbordos	0.81	0.76
Tiempo de espera medio	14 minutos	11 minutos
Tiempo de espera peor	19 minutos	15 minutos
Varianza sobre el tiempo medio en el autobús	12 minutos	18 minutos
Número de repeticiones del experimento	8	8
Rango horario	11:00 – 14:00	11:00 – 14:00
Número generaciones algoritmo genético	-	4

Ventana de tiempo para recogida	-	15
Ventana de tiempo para destino	-	100

2.3.1.3 Formando una Y con diferentes puntos

Generamos cuatro zonas diferentes formando una “Y”, y se da la posibilidad de generar transbordos entre autobuses. Las zonas inferiores serán los orígenes hacia las zonas superiores, y a su vez la zona situada más al sur será destino de las zonas superiores. Para este test generamos 1000 usuarios y hacemos uso de 80 autobuses.

Los límites de las cuatro zonas son:

Inferior Límite norte: 40.404283 Límite sur: 40.38519088 Límite oeste: -3.72845700 Límite este: -3.704565	**Superior izquierda** Límite norte: 40.45190200 Límite sur: 40.43959088 Límite oeste: -3.72040700 Límite este: -3.707965
Central Límite norte: 40.424086 Límite sur: 40.404283 Límite oeste: -3.712565 Límite este: -3.688974	**Superior derecha** Límite norte: 40.45190200 Límite sur: 40.43959088 Límite oeste: -3.679974 Límite este: -3.64954588

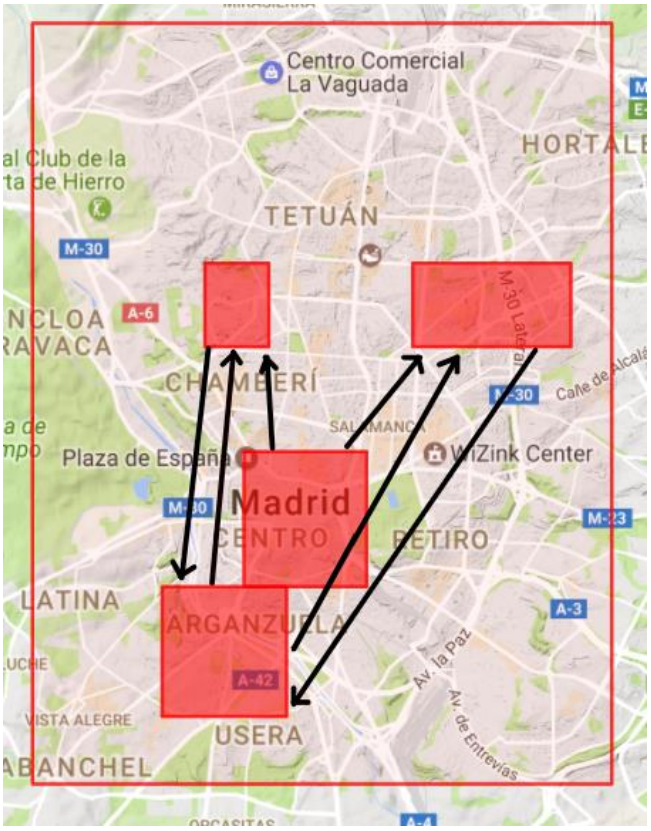

Figura 46: Prueba formando una Y con diferentes puntos

Resultados:

Variables	Sistema estático	Sistema dinámico
Número de autobuses utilizados	80	80
Tiempo en el autobús medio	64 minutos	43 minutos
Tiempo en el autobús peor	93 minutos	84 minutos
Media de transbordos	0.88	0.35
Tiempo de espera medio	14 minutos	11 minutos
Tiempo de espera peor	24 minutos	15 minutos
Varianza sobre el tiempo medio en el autobús	4 minutos	6 minutos
Número de repeticiones del experimento	8	8
Rango horario	11:00 – 14:00	11:00 – 14:00
Número generaciones algoritmo genético	-	5
Ventana de tiempo para recogida	-	15 minutos
Ventana de tiempo para destino	-	90 minutos

2.3.1.4 Mil usuarios por todo el rectángulo

Generación de mil usuarios con orígenes y destinos totalmente aleatorios.

Resultados:

Variables	Sistema estático	Sistema dinámico
Número de autobuses utilizados	160	160
Tiempo en el autobús medio	67 minutos	72 minutos
Tiempo en el autobús peor	85 minutos	107 minutos
Media de transbordos	0.82	0.47
Tiempo de espera medio	14 minutos	17 minutos
Tiempo de espera peor	20 minutos	105 minutos
Varianza sobre el tiempo medio en el autobús	5 minutos	7 minutos
Número de repeticiones del experimento	5	10
Rango horario	15:00-18:00	15:00-18:00
Número generaciones algoritmo genético	-	5
Ventana de tiempo para recogida	-	20 minutos
Ventana de tiempo para destino	-	110 minutos

2.3.1.5 Ciento veinte usuarios por todo el rectángulo

Generación de ciento veinte usuarios con orígenes y destinos totalmente aleatorios.

Resultados:

Variables	Sistema estático	Sistema dinámico	Sistema semi dinámico
Número de autobuses utilizados	15	15	15
Tiempo en el autobús medio	64.4 minutos	85 minutos	60 minutos
Tiempo en el autobús peor	83 minutos	110 minutos	75 minutos
Media de transbordos	0.77	0.1	0.6
Tiempo de espera medio	13 minutos	12 minutos	11 minutos
Tiempo de espera peor	16 minutos	20 minutos	15 minutos
Varianza sobre el tiempo medio en el autobús	3 minutos	5 minutos	2.5 minutos
Número de repeticiones del experimento	6	15	8
Rango horario	11:00-12:00	11:00-12:00	11:00-12:00
Número generaciones algoritmo genético	-	10	10
Ventana de tiempo para recogida	-	20 minutos	20 minutos
Ventana de tiempo para destino	-	120 minutos	120 minutos

2.3.2 Casos de prueba de la app Android junto con el servidor de peticiones

La app Android y el servidor de peticiones se han probado al mismo tiempo que se ejecutaban los casos anteriores y por separado para comprobar con exactitud su funcionamiento.

Se ha probado a crear múltiples usuarios como usuarios viajeros, hacer login en cada dispositivo con un usuario diferente y crear un nuevo evento. Al usuario le aparece por pantalla un mensaje de si el evento se ha creado con éxito (han tenido éxito las pruebas realizadas). Mientras tanto, al servidor le llegan todas estas peticiones, realiza las operaciones debidas y responde a cada usuario en su dispositivo. Al ser tratados los eventos y modificarse el fichero JSON de usuarios, el servidor envía correctamente a cada dispositivo la notificación de cambio de ruta de cada usuario.

Por otro lado, también han sido creadas cuentas con usuarios, conductores y establecido el autobús que va a ser usado por cada usuario. Mostrándose así en el dispositivo el mapa con la ruta del bus seleccionado por el usuario.

2.3.3 Casos de prueba de la interfaz web

Todas las funcionalidades de la interfaz web han sido probadas mientras se ejecutaban las demás partes del sistema.

Con éxito, la web muestra el mapa de Madrid con todos los usuarios (ya sea origen o destino), autobuses, las rutas de los autobuses y los carteles al pinchar sobre cualquier icono indicando la información debida.

Los iconos se actualizan correctamente a medida que pasa el tiempo, pudiéndose ver claramente el movimiento de cada autobús y la ruta que sigue gracias a la polyline dibujada sobre la carretera.

3. Funcionalidad

3.1 Componentes del sistema

Los componentes principales del sistema son:

· Aplicación cálculo de ruta

· Servidor para la gestión de peticiones

· Aplicación móvil

· Interfaz web

· Base de datos

· Fichero JSON datos de Usuarios

3.1.1 Aplicación cálculo de ruta

El funcionamiento de la aplicación está basado en algoritmos de enrutamiento que tratan de minimizar la distancia recorrida por los autobuses. La obtención de rutas más cortas entre puntos dados la llevamos a cabo mediante la librería OSRM.

Estos enrutamientos se calculan a partir de un vector de coordenadas ordenado por el cluster y el algoritmo de enrutamiento donde las coordenadas corresponden a las paradas a realizar por el autobús.

3.1.2 Servidor para la gestión de peticiones

Se encarga de ser el intermediario entre la aplicación de cálculo de ruta C++, la base de datos y la aplicación cliente mediante el fichero JSON de datos de rutas de usuarios.

El servidor estará escuchando constantemente las posibles llamadas de la app móvil (cliente). Al recibir una llamada, hará las operaciones necesarias conectando con la BBDD dependiendo de la petición (operaciones explicadas con más detalle más adelante) y responderá con la información debida y si la operación se ha finalizado con éxito o no.

Además, el servidor se encarga de las notificaciones a usuarios y conductores acerca de los cambios de ruta realizados por la aplicación cálculo de ruta. Para ello, está constantemente comprobando si se producen cambios en el fichero JSON. Si hay cambios, se notificará al usuario correspondiente.

Las funcionalidades que tiene el servidor son:

- Lee constantemente el fichero JSON de usuarios, comprobando si se ha producido algún cambio. Si encuentra algún cambio, mira el id del autobús o del usuario para buscar en la base de datos el id correspondiente al dispositivo de dicho usuario y mandarle una notificación sobre el cambio de ruta producido.
- Mientras el servidor está ejecutándose, está escuchando cualquier petición que pudiera hacer un cliente.
- Información de usuario: petición GET pasándole por parámetro el id del usuario, buscando en el fichero de usuarios dicho id y respondiendo la ruta del usuario si la hubiera.

- Información de ruta: petición GET pasándole por parámetro el id del autobús, buscando en el fichero de autobuses dicho id y respondiendo la ruta del autobús si la hubiera.
- Enviar evento: petición POST para añadir un nuevo evento de usuario en la BBDD. Se le pasa por parámetro el id del usuario, las coordenadas latitud y longitud de origen y destino y la hora a la que le gustaría ser recogido.
- Nuevo usuario: petición POST para añadir un nuevo usuario en la BBDD al hacer Signup. Se le pasa por parámetro el email, la contraseña, el nombre y el rol (conductor o viajero).
- Login: petición GET para loguearse en la aplicación. Se le pasa por parámetro el email y la contraseña y responde si es correcto o no.
- Dispone de una interfaz web con una lista de todos los dispositivos registrados en la BBDD con la posibilidad de enviar una notificación a cualquier dispositivo de la lista.
- Añadir dispositivo: petición POST para añadir un dispositivo a la BBDD que pueda recibir notificaciones de las rutas. Se le pasa por parámetro el nombre del dispositivo, el id del dispositivo, el id de registro del dispositivo y el id de usuario.
- Borrar dispositivo: petición DELETE para borrar un dispositivo de la BBDD para que no reciba más notificaciones. A esta función se le llama cuando el cliente hace logout en la aplicación.
- Añadir un usuario a un autobús: petición PUT para añadir el id de usuario a un autobús en la BBDD. Actualiza la BBDD cambiando de conductor de autobús para saber a qué dispositivo enviar las notificaciones.
- Enviar mensajes al dispositivo correspondiente. Se le pasa por parámetro el texto del mensaje y el id de registro del dispositivo.
- Conexión a la BBDD remota.

Además, se ha hecho uso del servidor web para crear ficheros Excel con orígenes y destinos de usuarios aleatorios para generar casos de prueba del proyecto e insertar en la BBDD como eventos, para que el servidor de cálculo de rutas pueda hacer uso de ello.

3.1.3 Aplicación móvil

Aplicación desarrollada para dispositivos móviles Android, con la posibilidad de funcionar en dos modos dependiendo del usuario que la utilice. Al iniciar la app por primera vez, aparecerá una ventana de Login (Figura 20) pidiendo al usuario el email y la contraseña. Si aún no se dispone de cuenta de usuario o el usuario quiere crear una nueva cuenta, tiene la posibilidad de pulsar sobre el link de signup, que le llevará a un nuevo formulario para crear la nueva cuenta (Figura 21).

Al crear una nueva cuenta, se deben de completar todos los campos (username, email, rol y password). El email debe de ser válido y el rol a introducir debe ser "viajero" o "conductor".

Una vez creada la cuenta, se podrá iniciar sesión desde la vista de Login. Una vez hecho login, el usuario entrará directamente en su perfil cada vez que inicie la app, ya que se hace uso de sesiones.

También el usuario podrá hacer logout cuando desee salir de su cuenta.

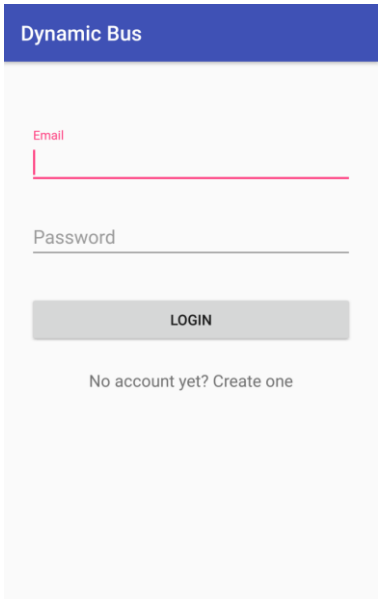

Figura 17: Login de usuario

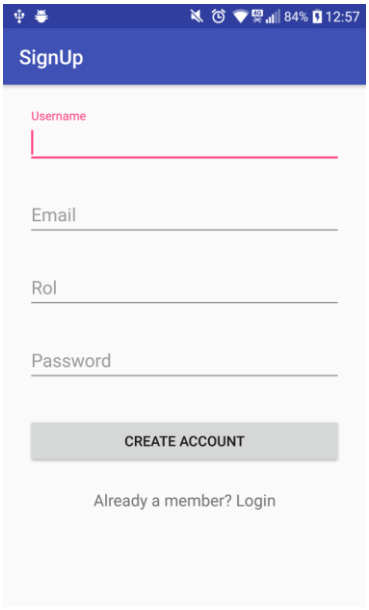

Figura 18: SignUp de usuario

Usuario viajero: El viajero tendrá la opción de solicitar una ruta nueva o consultar la información de su ruta ya establecida previamente (Figura 22). Para solicitar una nueva ruta se deberá introducir la localización donde le gustaría que se le recogiese, el destino y la hora de salida. Se enviará la petición al servidor con los datos y este devolverá la información de la ruta y se mostrará en una nueva vista. En la vista de información de ruta (Figura 26), el usuario podrá ver el id del autobús que le recogerá, la hora a la que será recogido, la hora a la que llegará al destino, la calle donde será recogido, la calle donde se le dejará y debajo un mapa con los puntos exactos donde será recogido y dejado.

Si el usuario solamente desea ver la información, al pulsar el botón "Datos usuario", se irá a la vista de información de la ruta.

Para introducir la localización de recogida y la de destino (Figura 23), el usuario deberá pulsar sobre el botón “Mapa”, que le llevará a una nueva vista con dos inputs para introducir la dirección de origen y la de destino. Al introducir el origen, el usuario deberá pulsar sobre el botón "SET ORIGEN" para establecer el punto de recogida. Al pulsar, el mapa indicará el punto exacto en el mapa para que el usuario se asegure de que lo ha puesto correctamente. El funcionamiento será el mismo para el destino y el botón "SET DESTINO" (Figura 24).

Una vez establecidos los puntos, se pulsará sobre el icono de check negro que guarda los datos y vuelve a la vista principal, y se enviará el nuevo evento del usuario, mostrándose la localización seleccionada de origen y destino.

Al enviar el nuevo evento, se mostrará un mensaje de éxito o de error, para indicar al usuario si el servidor ha recibido la petición (Figura 25). Una vez el usuario tiene establecida su ruta, si esta cambia, el servidor enviará al usuario una notificación de que su ruta ha sido modificada. La notificación aparecerá en la barra de notificaciones del dispositivo (Figuras 27 y 28), tanto si la app está iniciada como si no. Si la aplicación está cerrada, pulsando sobre la notificación se abrirá la aplicación. En cambio, si la aplicación se encuentra abierta, aparecerá además un mensaje de que la ruta ha sido modificada y recomendará ir a la vista de información de ruta.

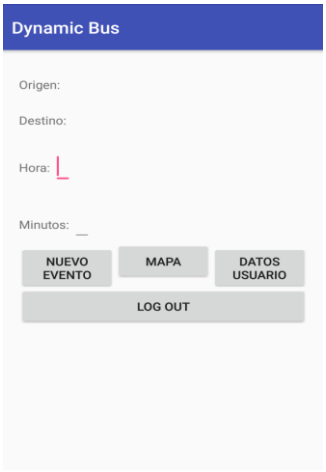

Figura 19: Vista principal del viajero

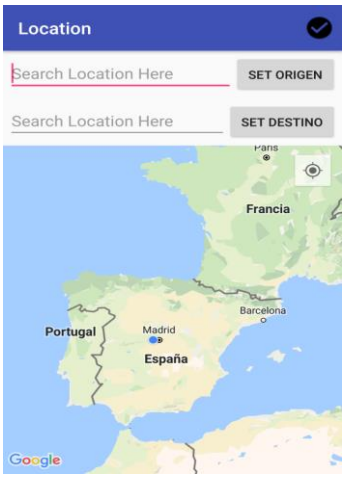

Figura 20: Introducir localización de origen y destino

Figura 21: Set origen y destino

Figura 22: Evento enviado con éxito

Figura 23: Vista de información de ruta

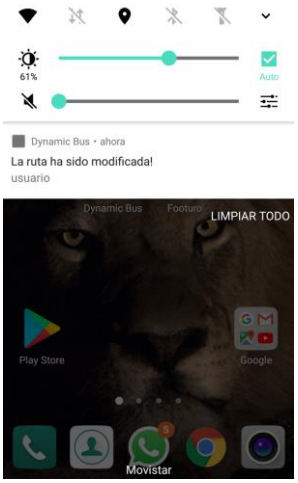

Figura 24: Notificación en la barra de notificaciones

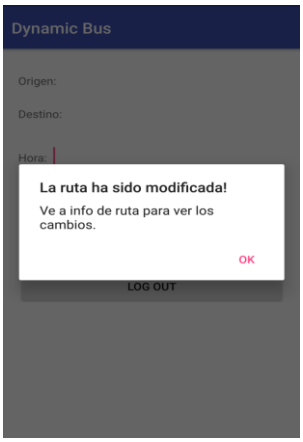

Figura 25: Notificación dentro de la app

Figura 26: Vista de información de ruta del conductor

Usuario conductor: El conductor de autobús hará uso de la aplicación para conocer la ruta que tiene que seguir. La aplicación muestra el mapa con la ruta que debe de hacer el autobús y la posición actual del mismo.

El conductor, al hacer login, accede directamente a la vista de información de ruta (Figura 29), con un cuadro para introducir el id del autobús en el que se encuentra y un botón para confirmar. Al introducir el id del autobús en el que se encuentra, si hay ruta ya establecida se mostrará en pantalla (Figura 30). Si se modifica la ruta del autobús, se actualizará automáticamente la ruta.

Figura 27: Ruta del autobús establecido

3.1.4 Interfaz web

Interfaz web para visualizar el estado del sistema en tiempo real. La interfaz muestra el estado actual de todos los autobuses y usuarios del sistema en un mapa de la ciudad.

Se muestra un icono de usuario que indica la parada de dicho usuario. Si se pincha sobre el icono aparecerá un cartel indicando el id de usuario. El usuario al ser recogido desaparece su icono del mapa.

El icono de destino de usuario es igual al de origen pero con un círculo alrededor suyo. También al pinchar sobre el icono se mostrará el id del usuario seleccionado.

El icono de autobús se muestra en azul y se va actualizando su posición a tiempo real a medida que pasa el tiempo. Al pinchar sobre su icono aparecerá un cartel indicando el id de autobús y además una línea roja mostrando la ruta que debe de llevar el autobús. Al pinchar otra vez sobre él la ruta se dejará de mostrar.

Los iconos se actualizan cada 2 segundos para seguir con detalle sus posiciones a tiempo real.

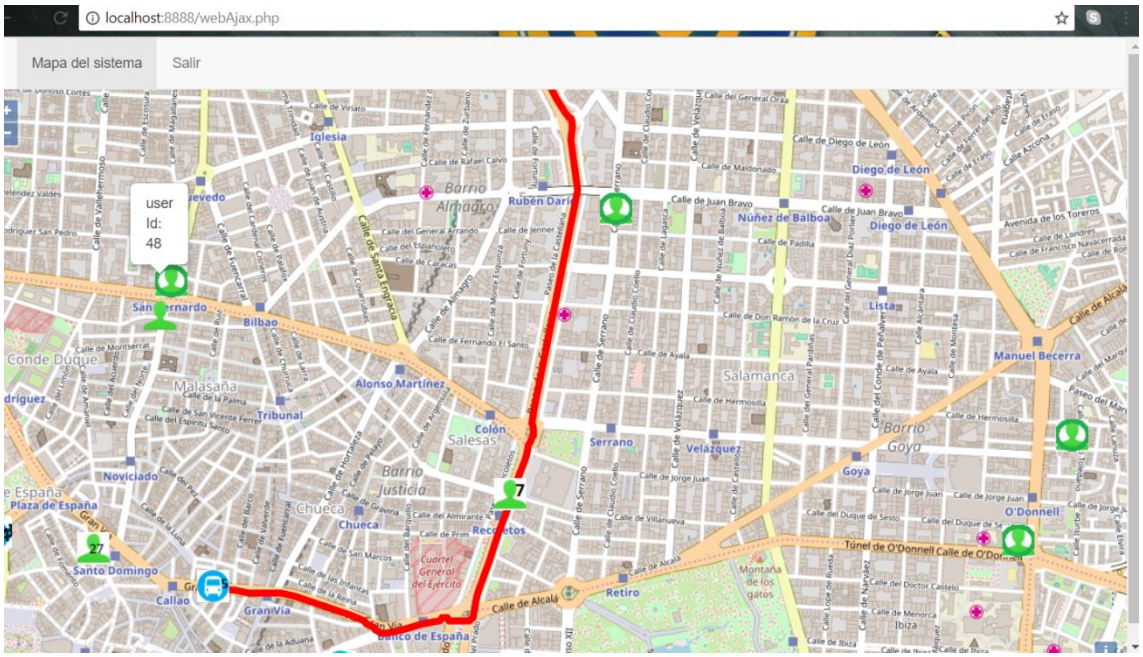

Figura 28: Vista de la interfaz web

3.1.5 Base de datos

La funcionalidad de la BBDD es guardar toda la información necesaria acerca de los usuarios, autobuses y eventos. El servidor de peticiones, la aplicación de cálculo de rutas y la interfaz web hacen uso de la BBDD con esta funcionalidad:

- Servidor de peticiones:
    - Comprobar los datos del usuario para hacer login en la app móvil.
    - Insertar un usuario al registrarse en la app móvil.
    - Insertar un nuevo evento creado por el usuario desde la app móvil.
    - Insertar nuevos eventos a partir de los puntos aleatorios para los casos de prueba.

    - Insertar un nuevo dispositivo cuando un usuario hace login, y borrarlo cuando hace logout.
    - Comprobar si un usuario está logueado en un dispositivo, para mandarle una notificación.
    - Obtener la posición y ruta de cada autobús.
- Aplicación de cálculo de rutas:
    - Comprobar si existen nuevos eventos de usuarios.
- Interfaz web:
    - Ver las coordenadas actuales de los autobuses, usuarios y rutas.

3.1.6 Fichero JSON datos de usuarios

Se trata de un fichero con formato JSON que hace de intermediario entre la aplicación de cálculo de rutas y la aplicación servidor. Este fichero contiene la información que necesita el usuario: contendrá el id del usuario, el id del autobús, la hora que será recogido el usuario, la hora a la que llegará al destino y la posición donde será recogido.

La aplicación de cálculo de ruta escribe las nuevas rutas y las modificaciones de paradas de los usuarios. El servidor lee dicho fichero constantemente para notificar al cliente en caso de cualquier cambio de su ruta lo antes posible.

El fichero consta de un array de elementos. Cada elemento es la información de ruta de cada usuario. Tiene las siguientes claves:

- id: El id único de cada usuario.
- idAutobus: El id único de cada autobús.
- horaOrigen: La hora a la que llegará el autobús al origen establecido.
- horaDestino: La hora a la que llegará el autobús al destino establecido.
- calleOrigen: La calle donde va a ser recogido el usuario.
- calleDestino: La calle donde se le va a dejar al usuario.
- coordXOrigen: La latitud del origen.
- coordYOrigen: La longitud del origen.
- coordXDestino: La latitud del destino.
- coordYDestino: La longitud del destino.

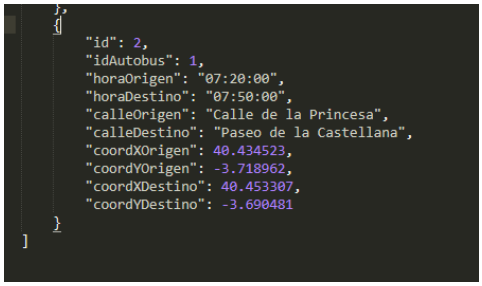

Figura 29: Parte del fichero JSON de datos de usuarios

4. Conclusiones

4.1 Conclusiones

Tras muchas pruebas, diferentes casos de pruebas, y diferentes números de usuarios en cada prueba llegamos a la conclusión de que actualmente este sistema para grupos grandes (mil usuarios simultáneos) da mejores resultados que las rutas estáticas actuales, como se puede observar en la prueba de centro caliente. El tiempo de espera medio es menor que en un sistema estático, ya que el desplazamiento del usuario a la parada se reduce enormemente. El tiempo en el autobús medio es peor (alrededor de un 15%) que en los sistemas estáticos, pero no es necesaria una flota de autobuses tan grande.

En el caso de prueba de Y se puede observar como nuestro sistema dinámico mejora el tiempo medio en el autobús en comparación con uno estático ya que las nuevas rutas que va generando el sistema van adaptándose a las existentes para obtener un mejor resultado sin necesidad de que los usuarios hagan uso de los transbordos.

En la prueba donde los usuarios están distribuidos en cuatro zonas formando un rombo, el sistema dinámico obtiene mejores resultados, disminuyendo el número de transbordos medio necesario en comparación con el sistema estático ya que genera rutas para cambiar entre las zonas de una forma directa disminuyendo el tiempo de viaje; pero, generando los transbordos necesarios para que, tras llegar a la zona objetivo, pueda haber un intercambio de personas por los autobuses para poder llegar a sus destinos finales sin empeorar los márgenes de tiempo.

En cuanto al caso de los mil puntos repartidos aleatoriamente por todo el mapa, el sistema estático obtiene mejores resultados ya que el sistema dinámico no es capaz de encontrar rutas suficientemente buenas cuando los usuarios se encuentran muy separados. En estas situaciones, un mayor número de usuarios sería beneficioso para el sistema ya que al haber menos autobuses y más distancia entre los usuarios, los autobuses necesitan recorrer mayor distancia para satisfacer las peticiones empeorando el tiempo medio de viaje ligeramente y en gran medida, el tiempo de viaje en el caso peor.

Así mismo, se observa como el sistema dinámico se queda muy rezagado en cuanto a media de tiempo en el autobús y tiempo de espera para pruebas con pocos usuarios a pesar de ampliar las ventanas de tiempo de los usuarios y su margen de llegar al destino. En estos casos, el sistema semi-dinámico arroja mejores resultados, reduciendo los tiempos de espera de los usuarios y mejorando el tiempo invertido en el desplazamiento de los sistemas estático y dinámico.

Sin embargo, para grupos muy grandes de usuarios es necesario mejorar el sistema ganando mayor velocidad de cálculo. Sería necesario hacer los cálculos de rutas en máquinas más potentes y así mismo realizar dicho cálculo en múltiples máquinas y después juntando las soluciones para ganar mayor velocidad de respuesta para el usuario ya que es algo muy importante que el usuario sepa cuanto antes que autobús debe coger y dónde.

Además, nos hemos percatado que la mejor manera de resolver las instancias del TSP que se nos plantean no es minimizar el camino a recorrer si no que, por el contrario, se debe tender a maximizar el número de usuarios que puede satisfacer cada autobús.

En general, el sistema dinámico tiene ciertas ventajas y ciertos inconvenientes dependiendo de la situación en comparación con un sistema estático como se observa en las pruebas realizadas. Podríamos decir que para poder determinar cuál de los dos es mejor, sería necesario realizar experimentos reales, en una ciudad tomada como base, para poder añadir a los cálculos variables tales como, el tráfico, vías con mayor regulación de semáforos, errores humanos (como que el conductor se pueda equivocar de calle, los usuarios no estén en la ubicación indicada en el momento acordado, etc…) Además, sería necesario realizar más simulaciones para poder probar mejor todas las funcionalidades y posibilidades de un sistema dinámico, ya que para poder llevarlo a la práctica en la vida real, deben tener un funcionamiento y rendimiento mejor que los sistemas estáticos implantados en la actualidad.

En un futuro cercano sería muy difícil implantar únicamente redes de autobuses dinámicas, ya que actualmente nuestra sociedad no está preparada para ello, ya que no todo el mundo dispone de un dispositivo electrónico adecuado o conocimientos para su uso correctamente.

Por supuesto, este sistema también depende de la ciudad donde se quiera establecer y el número de usuarios que tiene, además de los márgenes de tiempo que se consideren aceptables, ya que en nuestras pruebas hemos usado como referencia ventanas de tiempo de entre 15 y 20 minutos para la recogida del usuario y de entre 90 y 120 minutos para dejar al usuario en su destino.

4.2 Conclusions

After many tests, different test cases and different number of users in each test we got to the conclusion that actually this system applied for big groups (a thousand simultaneous users) delivers better results than the static systems, as can be seen in the crowded center test. The average waiting time is smaller than the static system one, because the distance from the user to the bus stop is much less. The average time in the bus is worse (like a 15%) than the static system average, but its not necessary a float as big as the static one needs.

In the Y test case it is visible that our dynamic system improves the average time in the bus from the static one, because the new paths that are being generated by the system get adapted to the existing ones, to get a better result without making transfers to another bus by the users.

In the test where users are distributed in four different areas making a diamond, the dynamic system get better results, decreasing the average number of transfers in comparation with the static system, because it generates paths to change between the four areas in a straight way decreasing the time of travel, but generating the necessary transfers to reach the objective area, achieving an exchange between users in the buses to get to their final destinations without aggravate the margins of time.

As for the case of the thousand points randomly distributed throughout the map, the static system obtains better results since the dynamic system is not able to find good enough routes when the users are very separated. In these situations, a greater number of users would be beneficial to the system, since with fewer buses and more distance between users, buses need to travel more distances to meet the requests, deteriorating the average travel time slightly and the time of travel in the worst case.

It is observed how the dynamic system is worse in terms of average time on the bus and waiting time for tests with few users despite widening users time windows and their margin of reaching the destination. In these cases, the semi-dynamic system obtains better results, reducing the waiting times of the users and improving the time invested in the displacement

of static and dynamic systems. However, for very large groups of users it is necessary to improve the system gaining greater calculation speed. It would be necessary to do route calculations on more powerful machines and maybe split the tasks between differents machines, one calculating the clustering, transfers and adding the new users to the system and others for calculating the routing algorithm of each group and then putting all the solutions together to gain greater speed of response for the user since it is very important that the user knows as soon as possible that bus should catch and where.

In addition, we have noticed that the best way to solve the TSP instances we are facing is not to minimize the way to go, but rather to maximize the number of users that each user can satisfy every bus.

In general, the dynamic system has some advantages and some disadvantages deppending on the situation compared with a static system as is shown on the done tests. We could say that in order to determine which of the two is better, it would be necessary to perform real experiments, in a taken city, to be able to add more variables to the calculations such as traffic, roads with greater regultarion of signals or traffic lights, human errors (like the driver can be mistaken of the Street, the users are not at the indicated location at the agreed momento, etc…) Also, it would be necessary to perform more simulations to be able to test more the funcionalities and the possibilities of a dynamic system since in order to put it into practice in real life, it must have a better performance tan the static system nowadays.

In the near future it would be so difficult to use an only dynamic bus network since nowadays our society is not prepared for it because not everyone has a suitable electronic device or enough knowledge for using it properly.

Definetly, the dynamic bus network also deppends on the specific circumstances of the city where it will be implanted and the flow of users that it will have. Due to this, it will also be important the acceptable time window that will be applied on the system because on the done experiments the time windows that have been used were between 15 and 20 minutes to pick up each user and between 90-120 minutes to leave the user at his destination.

5. Trabajo futuro

Como trabajo en un futuro, nos gustaría mejorar la estructura global del proyecto. Es decir no tener tantos servidores corriendo a la vez y juntar la interfaz web implementada en PHP junto con el servidor web implementado en Node js para tener sólo uno. También pensamos en generar un ejecutable del cálculo de rutas desarrollado en C++ e incluirlo en el servidor web, con lo que nos evitaríamos la conexión entre el servidor web, los dos ficheros JSON de usuarios y autobuses y la aplicación de cálculo de rutas.

Además, podríamos implementar una app móvil desarrollada en lenguaje Swift para que los usuarios con un dispositivo iOS también puedan realizar peticiones de autobuses, ya que actualmente sólo podrían los dispositivos Android.

Para realizar el sistema más realista y escalable, habría que tener ejecutándose constantemente el servidor en una máquina más potente para realizar las operaciones con mayor velocidad. Además para mayor eficacia sería interesante tener una máquina principal que gestione el algoritmo de agrupamiento junto con las nuevas peticiones y los transbordos acompañada de máquinas enfocadas a resolver los algoritmos de enrutamiento de los diferentes grupos dentro del sistema.

También, por supuesto no tener el servidor de peticiones en local, sino en remoto para que desde cualquier dispositivo y lugar pueda utilizar la app móvil.

Mejorar el algoritmo y probar muchos más casos de eventos y usuarios. Ya que para que se ponga en funcionamiento este sistema se debe de asegurar su correcto funcionamiento en muchas situaciones.

Realización de múltiples pruebas con casos reales. Es decir con autobuses reales para asegurar mejor el margen de error que puedan tener los conductores, ya que unos conductores circulan más rápido que otros y así mismo podrían equivocarse de calle y retrasar el sistema. Esos casos habría que tratarlos para mejor funcionamiento del sistema.

6. Organización del trabajo

6.1 Modelo de desarrollo

A la hora de trabajar, nos hemos basado en un modelo de desarrollo ágil.

El desarrollo ágil de software supone un enfoque para la toma de decisiones en los proyectos de software basado en métodos de ingeniería del software de desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. El trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo.

Cada iteración del ciclo de vida incluye: planificación, análisis de requisitos, diseño, codificación, pruebas y documentación, tiene gran importancia el concepto de "Finalizado", ya que el objetivo de cada iteración no es agregar toda la funcionalidad para justificar el lanzamiento del producto al mercado, sino incrementar el valor por medio de "software que funciona" (sin errores).

En nuestro proyecto, estábamos expuestos a que los requisitos cambiasen. Por ejemplo empezamos a desarrollar el cálculo de rutas con las APIs de Google Maps, pero posteriormente decidimos mejor hacerlo con diferentes librerías de cálculo de rutas.

Procuramos enseñarle algo funcional al profesor cada dos semanas.

La comunicación entre los integrantes del proyecto era cara a cara.

Somos un equipo auto-organizado (con ayuda del profesor para guiarnos).

A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Realización de pruebas automáticas.

6.2 Herramientas de comunicación

La comunicación en el proyecto ha sido mediante correo electrónico, teléfono móvil, reuniones entre los integrantes del grupo en persona o Skype y reuniones entre los integrantes del grupo y el profesor. Además, la información de cada integrante se actualizaba al hacer un commit en el repositorio.

Principalmente cada dos semanas nos reunimos con el profesor para ir viendo los progresos durante esas dos semanas y establecer las próximas tareas. Durante los últimos meses de proyecto nos reunimos cuando era necesario, ya que los avances y resultados iban siendo cada vez más rápidos.

6.3 Herramientas de control de versiones

Al inicio del proyecto empezamos utilizando Google Drive, donde disponemos de espacio ilimitado con la cuenta de la UCM. Ahí hemos ido almacenando diferentes ficheros y documentos útiles o de estudio para el proyecto, como ficheros pdf explicando algoritmos, enlaces de interés para investigar, tareas pendientes, la matriz de tiempos y distancias, etc.

Al empezar a desarrollar código, creamos un repositorio en Bitbucket que hemos usado durante todo el periodo del proyecto. Es bastante intuitivo y simple llevar un control de versiones y trabajar varias personas al mismo tiempo. En Bitbucket almacenamos todo el proyecto, salvo el servidor web, que lo almacenamos en Heroku por motivos de espacio del repositorio.

No hemos tenido grandes problemas con este apartado, ya que si trabajamos al mismo tiempo sobre mismos ficheros se crean ramas nuevas, y más adelante se une el código mediante un merge.

Hemos utilizado por mayor comodidad SourceTree. Se trata de un cliente visual de Git. SourceTree simplifica la forma de interactuar con los repositorios de Git y Mercurial para centrarnos en el código únicamente. Además, con los diagramas es muy sencillo ver los progresos y trabajar con las diferentes ramas.

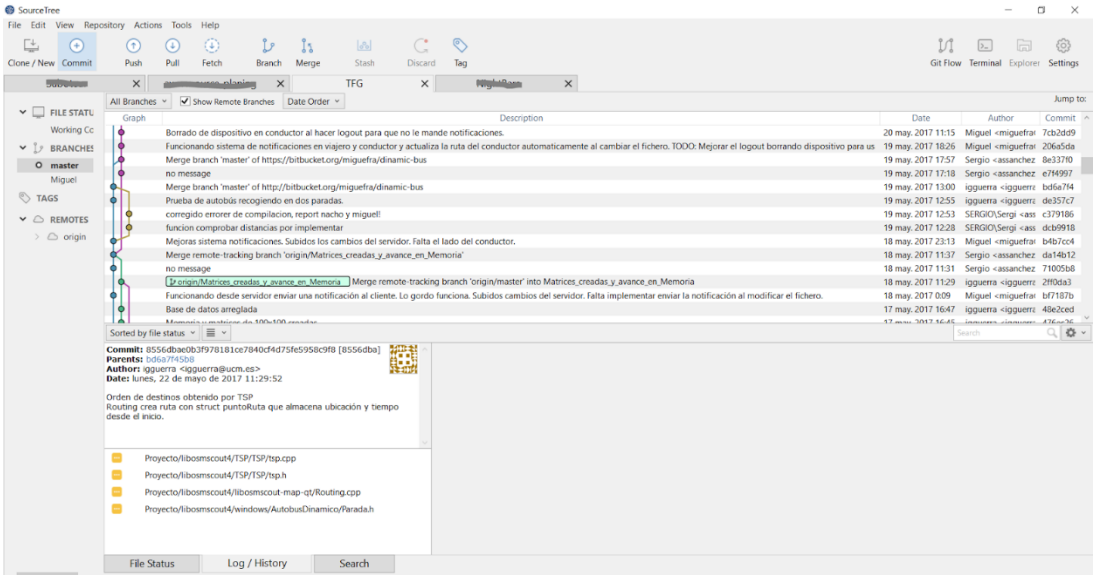

Figura 30: SourceTree

6.4 Trabajos realizados por cada miembro del grupo

6.4.1 Miguel

Con el comienzo del curso, a principios de octubre, cuando decidimos comenzar el desarrollo del Trabajo de Fin de Grado. Tuvimos una primera reunión con el director, nos explicó los objetivos del trabajo y las principales líneas a seguir. A partir de ahí, con las ideas más claras, empezamos a tomar decisiones sobre como comenzar el proyecto, lenguajes a utilizar, aplicaciones a desarrollar, etc.

Empecé a desarrollar una app Android con la API de Google Maps visualizando un mapa y mostrando puntos para tener una primera toma de contacto. Pero pronto nos dimos cuenta de que hacer el cálculo de rutas dentro de la app Android es inviable. Por lo que pensamos otro lenguaje de programación más adecuado. Decidimos implementar el cálculo de rutas de usuarios en C++ junto con el director.

Mientras tanto, junto con mis compañeros, estudiamos la posibilidad de usar unos tipos de algoritmos u otros. Buscando el más adecuado para nuestro proyecto. El director nos facilitó dos PDF con los algoritmos River formation y The ant colony optimization, además de buscar otros tipos de algoritmos similares al problema del viajante (TSP). Al buscar información, volvimos a tener reunión con el director comentándole las diferentes posibilidades. A partir de ahí comenzamos a quedar más o menos cada dos semanas para ir comentándole al director los progresos y que nos fuera guiando sobre lo que hacíamos.

Al decidir que desarrollaríamos una aplicación en C++ para el cálculo de rutas, además se pensó que haría falta una aplicación móvil para los usuarios generar las rutas que deseen y los conductores saber el trayecto que deben seguir. Yo me centré en desarrollar la aplicación móvil.

Para tener mejor organizado el proyecto, necesitábamos un repositorio de código, por lo que valoré las diferentes posibilidades de repositorios y decidí que la manera más sencilla sería usar Bitbucket y SourceTree ya que me parece bastante sencillo y útil.

Para el estudio y desarrollo de la app móvil Android he hecho uso de lo aprendido durante la carrera. Aunque además he tenido que investigar y aprender bastantes cosas que no conocía ni había utilizado nunca. Como la utilización de la API de Google Maps, Google Cloud Messaging y la librería Volley. Además de la investigación de la mejor manera de conectar la app a un servidor de peticiones. Al darme cuenta que necesitaba un servidor para que la app pueda realizar peticiones a una BBDD y estar conectada a la app de cálculo de rutas en C++, decidí desarrollarlo en Node Js a la vez que la app Android a medida que iba necesitando las peticiones. También tuve que estudiarme la documentación de Google para hacer uso de Google Cloud Messaging en web, esta parte fue una de las más tediosas y que más tiempo le tuve que dedicar a la hora de desarrollar, junto con el uso de la librería Volley ya que no tenía nada de experiencia con ella. También investigué el uso de Heroku, para desplegar el servidor en remoto, pero finalmente se decidió realizar en local.

A la vez que lo explicado anteriormente, junto con mis compañeros, creamos las principales tablas de la BBDD y yo a medida que iba necesitando nuevas tablas para la app móvil y el servidor web las creaba como por ejemplo la tabla de usuarios y la de dispositivos registrados. Decidimos usar phpMyAdmin en local mediante Xampp ya que teníamos experiencia en ello. Además investigue la manera de establecer la BBDD remota para usarla en conjunto todos (un tiempo estuvimos usándola remota), pero finalmente se mantuvo en local por motivos explicados anteriormente.

Creación y actualización del fichero JSON de datos de usuarios, desde la app en C++ de cálculo de rutas se crea el fichero JSON y actualiza a medida que se va ejecutando la aplicación. Además de leer dicho fichero desde el servidor de peticiones en Node para mantener informado al usuario en todo momento.

Ayudar a mis compañeros con diferentes partes de la aplicación C++ de cálculo de rutas como creación de hilos para la simulación del tiempo, actualización de las rutas y posiciones de los autobuses. Además de corrección de algún error dados durante la ejecución del programa.

Nacho creó la interfaz web, estudié y entendí como lo había implementado ya que yo nunca había hecho nada de PHP en la carrera para poder hacer mejoras y nuevas implementaciones. Además estudié el uso de la librería OpenLayer empezando a hacer cambios y nuevas implementaciones en el diseño y funcionamiento de la interfaz.

Aprovechando la aplicación web en Node, implementé en ella métodos para el cálculo de estadísticas de los casos reales con ayuda de la API de Google Maps. Investigué y desarrollé los usos de la API para calcular estadísticas de rutas en autobús para generar las pruebas con los casos de líneas estáticas actuales. Además, de generar las rutas aleatoriamente, meter los datos en la BBDD y generar un JSON con las estadísticas para volcar esa información sobre un fichero Excel para visualizar mejor los datos obtenidos.

Junto con los compañeros, realizar las pruebas de la aplicación C++ e interfaz web y creación del fichero JSON (en la aplicación C++) que devuelve las estadísticas de las pruebas. También realicé diversas pruebas para el correcto funcionamiento de la app Android junto con el servidor probando todos los casos de uso de los diferentes tipos de usuario y comprobando que los datos obtenidos y enviados fueran los correctos y con sentido.

Investigué las aplicaciones relacionadas con nuestro proyecto para añadir dicha información a la memoria. Realicé correcciones globales de la memoria en base a las correcciones del director sobre ella. Además, de la realización de la memoria junto con los compañeros centrándome más en las partes que he ido desarrollando.

6.4.2 Ignacio

El proyecto comenzó en octubre, cuando nos reunimos con el director por primera vez. Establecimos junto a Ismael una serie de objetivos y requisitos a cumplir además de cómo iba a ser el contacto con él a lo largo del curso. Quedamos en vernos cada dos semanas, incrementando la periodicidad de las reuniones en la fase final del proyecto.

En primer lugar, llevamos a cabo la fase de investigación. En la misma nos percatamos de que había que resolver instancias de TSPs de algún modo, por lo que decidimos investigar sobre el mismo. Me encargue de estudiar diferentes algoritmos para resolver TSPs, entre ellos el de Held-Karp, el voraz y algoritmos genéticos. Entre otros también entre en contacto con el algoritmo ACO y el RFD, información facilitada por nuestro director. Además, en esta fase de investigación también establecimos con que tipos de tecnologías íbamos a trabajar y que riesgos o problemas podían conllevar el uso de las mismas. El reto tecnológico principal era desarrollar la aplicación principal en C++ y acceder a datos reales de mapas en él en modo local.

Decidimos usar C++ como lenguaje principal de la aplicación, por lo que, en primera instancia, necesitábamos una fuente fiable de información de mapas y posibilidad de calcular rutas. Por tanto, busque opciones para trabajar con mapas en C++, lo que me llevó a encontrar una librería llamada Libosmscout, además de otras que no pudimos implementar, como Osmium o Mapnik. Entré en contacto con desarrolladores de software privativo relacionado con mapas, pidiéndoles ayuda para un proyecto académico, pero no accedieron a darnos una licencia sin coste.

Una de las opciones que tuve en cuenta fue Google Maps, pero la licencia gratuita tiene un límite de uso de uso que no nos permiten llevar a cabo todos los cálculos que necesitamos: 2500 solicitudes al día, además de un máximo de 23 puntos intermedios de paso. Trabajando con autobuses de 30 usuarios, nos interesa poder trabajar con 60 puntos intermedios de paso (origen y destino para cada usuario), por lo que está API fue descartada.

Instalar Libosmscout en nuestro proyecto no fue sencillo, la compilación en Windows derivó en una gran cantidad de fallos. Para poder compilarla necesité ayuda de sus desarrolladores, por lo que me puse en contacto con los mismos, y tras varios días conseguí que la librería funcionara.

Por lo que mis labores principales durante la fase de investigación fueron: investigar diferentes algoritmos para resolver TSPs además de investigar que opciones y librerías podíamos usar para trabajar con mapas en C++.

Una vez que teníamos el sistema para calcular rutas funcionando, decidimos implementar una versión inicial del sistema, con todos sus componentes, para evitar posibles fallos en el futuro. Estudiamos que tipos de datos nos interesaban de cada entidad y creamos las mismas en una base de datos.

En esta fase desarrollé un primer algoritmo que era capaz de enrutar, recoger y dejar usuarios. Este algoritmo consistía en resolver instancias de TSPs de modo voraz, cada autobús disponía de una cola de prioridad con una serie de usuarios y tenía que satisfacer a los mismos en orden de prioridad. La prioridad era el tiempo necesario para llegar al compromiso, por tanto, los autobuses iban recogiendo y dejando a los compromisos en orden temporal.

El sistema funcionaba, pero sin interfaz visual, por lo que me encargue de desarrollar una interfaz web basada en la librería OpenLayers 3, PHP, HTML y Javascript, que recogía la posición de usuarios y autobuses de la base de datos y los representaba en un mapa con datos de OpenStreetMaps. Al ver el sistema en funcionamiento sobre un mapa real pude ver como los resultados no eran tan buenos como los esperados, dando lugar a rutas de autobús que pecaban de ser muy largas para pocos usuarios.

Poco después de tener este sistema funcionando nos percatamos de que la librería libosmscout no estaba devolviendo siempre datos correctos, por lo que investigué buscando una alternativa y encontré OSRM, que es la librería usada actualmente en el sistema. La comunicación con la misma reside en un protocolo HTTP, por lo que decidí implementar un cliente HTTP en C++, librería disponible en el gestor de paquetes de Visual Studio.

Teniendo ya un sistema básico funcionando, decidimos buscar una solución mejor a la que teníamos, que, si bien nos demostraba que técnicamente el proyecto era factible, no daba lugar a resultados interesantes. Tanto los tiempos medios de espera como los de viaje en el autobús eran altos y desproporcionados para viajes que eran cortos, de pocos km.

En esta fase de desarrollo final decidimos buscar una mejor elección a la hora de asignar usuarios a autobuses. Tomamos la decisión de dividir nuestro problema en dos fases: agrupamiento y enrutamiento. Decidimos clusterizar a los usuarios y asociarles paradas, desarrollo del que se encargó Sergio. Participé en el desarrollo de la clusterización implementada por Sergio aportando ideas y resolviendo fallos que surgían.

Principalmente me encargué de la fase de enrutamiento, esta implicaba resolver instancias de TSPs. Atendiendo a que íbamos a tener grupos grandes de usuarios en cada grupo y al mal resultado de la solución voraz decidimos resolver los TSPs con algoritmos genéticos. Estos algoritmos se basan en partir de una población inicial de soluciones y por medio de evolución de padres a hijos obtener soluciones mejores.

En esta fase me encargué de estudiar cómo funcionan los algoritmos genéticos, cómo evolucionan y como podría funcionar en nuestro sistema. Encontré una implementación básica para resolver este tipo de problemas en C++ y la rediseñé para satisfacer Time Windows, además de no buscar literalmente el camino más corto, sino de buscar el que satisficiera a mayor número de usuarios. El uso, implementación y adaptación del algoritmo genético derivó en la realización de un número de pruebas muy alto hasta dar con un número de generaciones y una tasa de mutación interesantes para grupos de unos 20-30 usuarios.

También me encargué junto a Sergio, de estudiar cómo integrar un sistema estático y uno semi-dinámico en nuestro sistema, además de implementarlo, para poder comparar los datos contra los de nuestra solución. Estudiamos que líneas serian interesantes, además de dónde colocar las paradas para que el número de cruces entre líneas fuera muy alto, con el fin de provocar numerosos transbordos.

Finalmente, mi última tarea fue el desarrollo de la memoria, haciendo énfasis en las secciones relacionadas con el enrutamiento y cálculo de rutas. Además, llevé a cabo una corrección global de la memoria derivada de las últimas reuniones con el director.

6.4.3 Sergio

Tras el comienzo del curso y la primera reunión con el director, acotamos los requisitos iniciales para el proyecto, objetivos principales y secundarios. Decidimos que la parte principal del proyecto debía ser el algoritmo de enrutamiento y el cálculo de las rutas de los autobuses.

Durante las dos siguientes semanas previas a la segunda reunión estuve investigando diferentes algoritmos que podían dar solución al problema planteado, el TSP, además de investigar más a fondo sobre la naturaleza y el fondo del TSP, ya que, salvo un breve contacto con el mismo en una asignatura de la carrera, no sabía mucho más de él.

Tras la segunda reunión fijamos el uso de C++ para el desarrollo de la aplicación así como la necesidad de un algoritmo de agrupación para poder mejorar el rendimiento del futuro sistema. Además del uso de la ciudad de Madrid como base de pruebas debido a la familiaridad y características propias de la ciudad (tamaño, disposición, número de habitantes, etc...)

Mis actividades principales durante el proyecto han sido el desarrollo del algoritmo de agrupamiento, el desarrollo de la matriz que representa el mapa, el desarrollo del sistema de transbordos y el desarrollo de las ventanas de tiempo, así como la optimización de todos ellos para mejorar el rendimiento del sistema de forma global.

Al comienzo de la implementación de la aplicación fuimos añadiendo estructuras de datos, tales como la estructura de los autobuses, los usuarios y los eventos que posteriormente remodelé para convertirlos en objetos y que nuestra aplicación tuviese los beneficios de una programación orientada a objetos tales como herencia, cohesión, acoplamiento, etc…

La matriz fue un desarrollo sencillo. Se basa en una matriz cuadrada de 100 unidades por lado. La distancia de una unidad se acotó en 125 metros reales, es decir, se tuvo en cuenta la curvatura de la tierra para calcularlo, esto hace que las dimensiones reales del área de la matriz sean de 12.5 kilómetros de Oeste a Este y 9.4 kilómetros de Norte a Sur. Cada nodo de la matriz sería la representación de una parada que se usaría como referencia, añadiendo a la misma sus coordenadas, lista de usuarios, etc…

En cuanto al desarrollo del algoritmo lo realicé de forma iterativa y sencilla, añadiendo cada vez nuevas restricciones. Al comienzo, sólo existían 16 grupos distintos. El mapa estaba dividido en 4 partes más o menos iguales y las peticiones se agrupaban, en primera instancia, por coincidencias de su zona de origen y zona de destino para posteriormente ser agrupadas en puntos compartidos por distintos usuarios en función de la distancia que les separaba para evitar un número de paradas muy grande en una distancia corta. Posteriormente y con la incorporación de la matriz, el sistema tuvo que ser reestructurado casi por completo, la forma de agrupar a la gente modificó sus parámetros, ahora no sólo importaba el origen y el destino de las peticiones, sino además se tenían en cuenta todas las líneas de autobuses cuyo recorrido pasase a una distancia suficientemente próxima de la ubicación de origen de la petición y fuese en dirección a su destino, siendo siempre premiado el caso en el que el mismo autobús al que era asignada una petición para ser recogida, fuese el encargado de trasladar dicha petición a su destino. Además, las peticiones que estuviesen cercanas unas de otras, se juntarían en el mismo punto, (aunque no compartiesen los parámetros anteriores) para favorecer al futuro sistema de transbordos (a pesar de que su implementación no se llevó a cabo hasta mayo).

Con el sistema de agrupamiento funcionando bajo los términos mencionados, se empezó a probar que todo funcionaba correctamente y las distancias que se consideraban aceptables para la agrupación de peticiones, etc… Además de implementarse un sistema de cambio de ubicación para evitar que las nuevas peticiones que llegaban tuviesen que desplazarse en exceso para agruparse con otras con características similares. Esto hacía que la gente como máximo sufriese un cambio en la ubicación asignada para su recogida una vez, pero mejoraba el tamaño de los grupos que se formaban.

Tras esto, creé el sistema de las ventanas de tiempo, estas ventanas especifican el plazo en el que se debe recoger a un usuario por la ubicación indicada. Esto dio lugar a una nueva forma de agrupar a la gente, que se añadía a las anteriormente mencionadas pero con una prioridad mayor que las anteriores. Las peticiones que compartiesen una ventana de tiempo, tenderían a agruparse en primera instancia, ya que si dos peticiones tenían ventanas de tiempo dispares en una misma área, agruparlas no tendría ningún sentido.

El sistema de las ventanas de tiempo (o Time Windows) trajo consigo más problemas de los esperados. Fue necesario revisar varias veces su funcionamiento ya que, a pesar de que las peticiones obtenían correctamente su TW, a la hora de agruparlos con el algoritmo de agrupamientos y ser modificados los TW, los errores surgidos fueron excesivos. Este fue el motivo principal por el que el sistema de transbordos pasó a un segundo plano hasta el mes de mayo.

El sistema de transbordo significó un gran beneficio a todo el sistema en cuanto a calidad de resultados pero fue uno de los mayores retos en cuanto a los problemas de tiempos de ejecución que generó ya que, con ellos, las opciones de agrupación aumentaron considerablemente así como las posibilidades de llevar a los usuarios desde su origen a su destino. El gran problema que esto generó fue que los transbordos debían recalcularse para todos los autobuses que sufrían modificaciones en su ruta original, además de con los autobuses con la que las líneas modificadas tenían algún transbordo existente y con los nuevos autobuses añadidos al sistema debidos a la alta demanda. Este sistema se basó en que dos líneas de autobuses generaban una posibilidad de hacer un transbordo cuando ambas rutas cruzaban o pasaban suficientemente cerca la una de la otra en un punto concreto y dentro de un determinado rango horario, ya que no es suficiente que ambas rutas estén conectadas si no, que conecten en el momento preciso.

La complejidad de este sistema, sumado al incremento de los autobuses, degeneró en la necesidad de aplicar podas al algoritmo que se encargaba de calcular estos transbordos. Fue una tarea dura, pero se consiguió reducir de unos 10 minutos iniciales con 10 autobuses para el cálculo de transbordos a 3 minutos de media con más de 30 autobuses.

También estuve desarrollando la base de datos, junto con el resto del equipo ya que cada uno adaptaba las tablas a sus necesidades. Implementé junto con Ignacio cambios en la base del código del sistema para mejorar su funcionamiento tales como semáforos para los accesos a la base de datos, cambios de vectores utilizados inicialmente en el sistema por mapas para mejorar el sistema de forma global. Además colaboré en el desarrollo de algoritmo de enrutamiento de forma breve, ya que el peso del mismo fue llevado a cabo por Ignacio.

Además, implementé junto con Ignacio el sistema de autobuses semi-dinámico y el estático para la realización de los test. Llevé a cabo numerosas pruebas para comprobar el funcionamiento de la aplicación y obtención de datos, además de sus análisis.

Junto con mis compañeros también completé la memoria con los apartados en los que he estado más implicado, así como el formato de la misma y su estructura.

Bibliografía

[1] R. A. Day and B. Castel, *Cómo escribir y publicar trabajos científicos*, 4ª ed. ed. Washington: Organización Panamericana de la Salud, 2008.

[2] Universidad de la Laguna. (12 feb. 2013). *Cómo elaborar un trabajo* [en línea]. Disponible: http://www.ull.es/view/institucional/bbtk/Redaccion

[3] J. Riquelme, *Canon de presentación de trabajos universitarios: modelos académicos y de investigación*. Alicante: Acuallara, D. L. 2006.

[4] M. D. Borgoños Martínez, *Cómo redactar referencias bibliográficas en un trabajo de investigación: aplicación práctica del "Harvard Style"*. Madrid: ANABAD, 2007.

[5] C. Arroyo Jiménez and F. J. Garrido Díaz, *Libro de estilo universitario*. Madrid: Acento editorial, 1997.

[6] E. Casilari Pérez. (2007, 8 feb. 2013). *Breves notas de estilo para la redacción de proyectos fin de carrera* [en línea]. Disponible: http://www.poptelecomunicacion.uma.es/menu/Manual_de_estilo.pdf [7] J. Zobel, *Writing for computer science*, 2nd ed. ed. Heidelberg: Springer, 2004.

[8] Real Academia Española (Madrid), *Ortografía de la lengua española*. Madrid: Espasa, 2000.

[9] Real Academia Española (Madrid). (12 feb. 2013). *Página web de la Real Academia Española* [en línea]. Disponible: http://www.rae.es/rae.html

[10] FECYT. (Versión 17/01/07, 12-02-2013). *Propuesta de manual de ayuda a los investigadores españoles para la normalización del nombre de autores e instituciones* [en línea]. Disponible: http://www.accesowok.fecyt.es/wpcontent/uploads/2009/06/normalizacion_nombre_autor.pdf

[11] G. Múñoz Alonso, *Estructura, metodología y escritura del Trabajo de Fin de Máster*. Madrid: Escolar y Mayo, 2011.

[12] G. Norman, *Cómo escribir un artículo científico en inglés*. Madrid: Hélice, 1999.

[13] Universidad Carlos III (Biblioteca). (2008, 12 feb. 2013). *Como citar bibliografía* [en línea]. Disponible: http://www.uc3m.es/portal/page/portal/biblioteca/aprende_usar/como_citar_bibliografia

[14] A. Estivill and C. Urbano. (12 feb. 2013). *Cómo citar recursos electrónicos (Versión 1.0 ed.)* [En línea]. Disponible: http://www.ub.es/biblio/citae-e.htm 6

[15] Documentación de Google. Disponible: https://developers.google.com

[16] Documentación de Android. Disponible: https://developer.android.com

[17] Wikipedia. Disponible: https://es.wikipedia.org

[18] Paquetes NPM, documentación. Disponible: https://www.npmjs.com/

[19] Documentación oficial sobre Google Cloud Messaging. Disponible: https://developers.google.com/cloud-messaging/

[20] Documentación oficial sobre Volley. Disponible: https://developer.android.com/training/volley/index.html

[21] Pablo Rabanal, Ismael Rodríguez, Fernando Rubio Ismael, *Solving Dynamic TSP by Using River Formation Dynamics.*

[22] Daniel J. Rosenkratz, *An Analysis of Several Heuristics for the Traveling Salesman Problem*, 1977.

http://epubs.siam.org/doi/10.1137/0206041

[23] M. Held and R.M. Karp, *The traveling-salesman problem and minimum spanning trees: part II*, 1971.

https://www.researchgate.net/publication/200035312_The_traveling-salesman_problem_and_minimum_spanning_trees_Part_II

Anexo

Node js

Node.js es un entorno de ejecución multiplataforma que permite ejecutar Javascript fuera del navegador. Se distribuye bajo licencia MIT.

Principalmente dirigido a servidores web, pero puede utilizarse para implementar cualquier tipo de aplicación.

Las herramientas que se usan para gestionar proyectos son: Desde la línea de comandos npm o desde un IDE como NetBeans.

NPM es una herramienta que se incluye con la distribución de Node. Permite:

· Crear proyectos vacíos (así creamos nuestro proyecto).

· Gestionar las librerías de las que hace uso un proyecto, consultando una base de datos pública (NPM Registry) que indica desde dónde descargarlas.

· Gestionar scripts: arranque del sistema, parada, ejecución de tests, etc.

Node.js funciona con un modelo de evaluación de un único hilo de ejecución, usando entradas y salidas asíncronas las cuales pueden ejecutarse concurrentemente en un número de hasta cientos de miles sin incurrir en costos asociados al cambio de contexto. Este diseño de compartir un único hilo de ejecución entre todas las solicitudes atiende a necesidades de aplicaciones altamente concurrentes, en el que toda operación que realice entradas y salidas debe tener una función callback. Un inconveniente de este enfoque de único hilo de ejecución es que Node.js requiere de módulos adicionales como cluster para escalar la aplicación con el número de núcleos de procesamiento de la máquina en la que se ejecuta.

Google Cloud Messaging

Google Cloud Messaging (GCM) es un servicio de notificación móvil desarrollado por Google que permite a los desarrolladores de aplicaciones de terceros enviar datos de notificación o información de servidores desarrollados por el desarrollador a aplicaciones que se orientan al sistema operativo Google Android, así como a aplicaciones o extensiones desarrolladas para el navegador de Internet de Google Chrome. Ha sido reemplazado por el Firebase Cloud Messaging de Google (FCM). Por ejemplo, un mensaje descendente del servidor a la app podría informar a la aplicación cliente de que hay nuevos datos que se deben extraer del servidor, como en el caso de una notificación de cambio de ruta.

El servicio GCM se encarga de todos los aspectos de la cola de mensajes y la entrega hacia y desde la aplicación cliente de destino.

Una implementación de GCM incluye un servidor de conexión de Google, un servidor de aplicaciones en su entorno que interactúa con el servidor de conexión a través del protocolo HTTP o XMPP y una aplicación cliente.

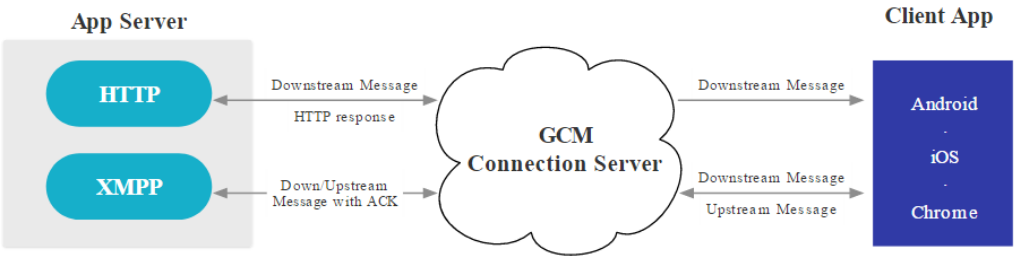

Figura 31: Estructura de conexión GCM

Componentes:

- GCM Connection Servers: Servidores de Google implicados en el envío de mensajes entre el servidor de aplicaciones y la aplicación cliente.
- Client App: Aplicación cliente habilitada para GCM. Para recibir y enviar mensajes GCM, esta aplicación ha sido registrada con GCM y un identificador único denominado token de registro.
- App Server: se implementa el protocolo HTTP para comunicarse con los servidores de conexión de GCM. El servidor envía mensajes descendentes a un servidor de conexión GCM. El servidor de conexión almacena el mensaje y, a continuación, lo envía a la aplicación cliente.

Credenciales:

- Sender ID: Un valor numérico único creado al configurar el proyecto. El identificador del remitente se utiliza en el proceso de registro para identificar un servidor con permiso para enviar mensajes al cliente.
- Server Key: Clave guardada en el servidor que proporciona al servidor un acceso autorizado a los servicios de Google. En HTTP, la clave del servidor se incluye en el encabezado de las solicitudes POST que envían mensajes.
- Application ID: La aplicación cliente registrada. Se utiliza el identificador del paquete de la aplicación.
- Registration Token: ID emitido por los servidores de conexión de GCM a la aplicación cliente que le permite recibir mensajes.

Google Maps Directions API

Google Maps Directions API es un servicio que calcula indicaciones entre ubicaciones usando una solicitud HTTP.

Con la Directions API, puedes:

- Buscar indicaciones para diferentes medios de transporte.
- Mostrar indicaciones en varias partes usando una serie de waypoints.
- Especificar orígenes, destinos y waypoints como strings de texto, como coordenadas de latitud y longitud o ID de sitios.

Una solicitud de Google Maps Directions API debe respetar la siguiente forma: https://maps.googleapis.com/maps/api/directions/outputFormat?parameters

Cuando la Google Maps Directions API devuelve resultados, los coloca en una matriz (JSON) routes.

Cada elemento de la matriz routes contiene un solo resultado para el origen y el destino especificados. Esta ruta puede consistir en una o más etapas, según se hayan especificado waypoints o no.

Bitbucket

Bitbucket es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de revisiones Mercurial y Git.

Git es un software de control de versiones, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

Mercurial es un sistema de control de versiones multiplataforma, para desarrolladores de software.

Los conceptos básicos que hemos utilizado son:

- Merge: Unión de dos ramas diferentes.
- Pull: Descarga del código sobre la rama en la que te encuentras.
- Commit: Subir el código modificado (con opción a poner un comentario) en local.
- Push: Actualizar el repositorio con los commits realizados en local.
- Branch: Creación de una rama nueva para trabajar sobre ella.
- Revert: Volver a una versión anterior.

Heroku

Heroku es una plataforma como servicio de computación en la Nube que soporta distintos lenguajes de programación. Nosotros hemos utilizado Node js para el servidor web.

HTTP:

Hypertext Transfer Protocol o HTTP es el protocolo de comunicación que permite las transferencias de información en la World Wide Web. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse.

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. El cliente realiza una petición enviando un mensaje, con cierto formato al servidor. El servidor le envía un mensaje de respuesta.

Los mensajes HTTP, son en texto plano lo que lo hace más legible y fácil de depurar. Esto tiene el inconveniente de hacer los mensajes más largos.

Los mensajes tienen la siguiente estructura:

- Línea inicial
- Para las peticiones: la acción requerida por el servidor (método de petición) seguido de la URL del recurso y la versión HTTP que soporta el cliente.
- Para respuestas: La versión del HTTP usado seguido del código de respuesta (que indica que ha pasado con la petición seguido de la URL del recurso) y de la frase asociada a dicho retorno.
- Las cabeceras del mensaje que terminan con una línea en blanco. Son metadatos. Estas cabeceras le dan gran flexibilidad al protocolo.
- Cuerpo del mensaje. Es opcional. Su presencia depende de la línea anterior del mensaje y del tipo de recurso al que hace referencia la URL. Típicamente tiene los datos que se intercambian cliente y servidor. Por ejemplo para una petición podría contener ciertos datos que se quieren enviar al servidor para que los procese. Para una respuesta podría incluir los datos que el cliente ha solicitado.

Métodos de petición

HTTP define una serie predefinida de métodos de petición (algunas veces referido como "verbos") que pueden utilizarse. El protocolo tiene flexibilidad para ir añadiendo nuevos métodos y para así añadir nuevas funcionalidades. El número de métodos de petición se han ido aumentando según se avanzaba en las versiones. Cada método indica la acción que desea que se efectúe sobre el recurso identificado. Lo que este recurso representa depende de la aplicación del servidor. Por ejemplo el recurso puede corresponderse con un archivo que reside en el servidor. Los métodos que hemos utilizado en la aplicación son: GET y POST.

GET: Pide una representación del recurso especificado. La petición puede ser simple, es decir en una línea o compuesta de la manera que muestra el ejemplo.

POST: Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.

JetBrains

Compañía de desarrollo software especializada en diseño de IDEs.

Formato JSON

JSON, acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. Los tipos de datos disponibles en JSON son: Números, Cadenas, Booleanos, null, Array y Objetos.

Ejemplo de JSON:

Figura 32: Ejemplo de formato JSON

Aplicaciones relacionadas o de utilidad

Lo primero que hicimos al empezar el proyecto es investigar si existen aplicaciones similares a lo que vamos a realizar. Existen diferentes aplicaciones que te calculan una ruta de un punto a otro en el mapa. Pero no encontramos ninguna que te calculase rutas dinámicas que es lo que estábamos buscando, todo son rutas estáticas sin cambios.

Google Maps API

Google Maps API ofrece una gran variedad de servicios para trabajar con mapas, calcular distancias de un punto a otro, información sobre los puntos, etc. Pero el problema que tuvimos, debido al cual no lo utilizamos en el algoritmo de cálculo de ruta, es que dispone de un número limitado de peticiones al servidor. Y no podemos depender de un límite de peticiones, por lo que descartamos esta opción.

En cambio, si nos ha sido útil en la app móvil para seleccionar los puntos de origen y destino ya que no son tantas peticiones como el cálculo de rutas.

Uber, Cabify, Lyft

Se trata de aplicaciones que proporcionan a sus clientes una red de transporte privado a través de su software de aplicación móvil, que conecta los pasajeros con los conductores de vehículos registrados en su servicio, los cuales ofrecen un servicio de transporte a particulares. Su funcionalidad es de la siguiente manera:

1. Solicitar un trayecto o un conductor. Gracias a la geolocalización de tu smartphone esto resulta muy sencillo.

2. Recibir feedback del conductor a través de la App. Podrás ver cómo se acerca tu vehículo, e información acerca del conductor. De esta forma tendrás todo bajo control.
3. Realizar el viaje y pagar. En función del servicio que elijas tendrás más o menos comodidades. Pero el pago desde la propia aplicación móvil siempre será una de ellas.

La idea es similar a nuestro proyecto, pero únicamente disponen de un usuario y las rutas las escoge el conductor, nuestra idea se basa en varios usuarios por autobús y rutas dinámicas.