Universidad de Valladolid

E.T.S. Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Evaluación de soluciones para el cálculo de rutas óptimas con bases de datos

Autor:

Noelia Nestar Vian

Universidad de Valladolid

E.T.S. Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Evaluación de soluciones para el cálculo de rutas óptimas con bases de datos

Autor:
Noelia Nestar Vian

Tutor:
Dr. Diego R. Llanos Ferraris

Agradecimientos

A la primera persona que se lo quiero agradecer es a mi tutor Dr. Diego R. Llanos Ferraris, sin su ayuda y conocimientos no hubiese sido posible realizar este trabajo.

A Rober, por estar siempre ahí cuando nadie más lo esta, pieza clave de que este trabajo vea la luz.

A mis padres, por haberme proporcionado la mejor educación y lecciones de vida. En especial a mi padre, por haberme enseñado que con esfuerzo, trabajo y constancia todo se consigue, y que en esta vida nadie regala nada. Y a mi madre, por hacerme la vida cada día más sencilla y confiar en mis decisiones.

A mi hermana, con la que he compartido grandes momentos. Y como no a mis perros, los únicos capaces de sacarme una sonrisa en mis peores momentos.

Gracias a todos aquellos que siguen estando cerca de mi y que le regalan a mi vida algo de ellos.

Resumen

En el presente trabajo se propone un Sistema de Información Geográfica (SIG) basado completamente en software libre. El fin es evaluar el mismo como posible solución a la demanda tecnológica propuesta por la empresa GMV de una solución para el cálculo óptimo de rutas de transporte, atendiendo al requerimiento de ofrecer una batería de algoritmos, así como que el desarrollo no dependa de componentes comerciales. Se expone sistemáticamente la investigación, que inicia con la instalación y continua con un plan de pruebas y evaluación, cuyos resultados llevaran a la realización de un profundo análisis de proximidad. Se busca analizar y determinar la eficiencia del cálculo de rutas óptimas entre los algoritmos ofrecidos por esta plataforma.

La movilidad es un factor que influye de forma relevante en la competitividad de una empresa, por ello es fundamental encontrar herramientas que permitan innovar y mejorar en este aspecto. La fundamentación dada en las matemáticas, la ingeniería, y la aplicación de métodos adecuados pueden lograr el desarrollo de una estrategia que reduzca la improductividad en el tiempo de trayecto, disminuyendo a su vez los gastos en transporte.

Palabras clave: Shortest Path, SIG, Transporte, Cartografía, Codificación geográfica.

Abstract

In this work, a Geographic Information System (GIS) based entirely on free software is proposed. The purpose is to evaluate it as a possible solution to the technological demand proposed by the company GMV about a solution for optimal calculation by transportation routes, considering the requirement to provide a battery of algorithms and that the development does not depend on commercial components. The investigation is systematically exposed, beginning with installation and continuing with a plan of testing and evaluation, the results lead to the realization of a thorough analysis of proximity. It seeks to analyze and determine the efficiency of calculating optimal routes between the algorithms offered by this platform.

Mobility is an important competitiveness factor of a company, so it is essential to find tools to innovate and improve in this area. The foundations given in mathematics, engineering, and the application of appropriate methods can achieve the development of a strategy to reduce the unproductivity in the time of commuting and simultaneously reduce transport costs.

Keywords: Shortest Path, SIG, Transport, Cartography, Geocoding.

Tabla de Contenidos

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Todas las empresas de transporte y logística se enfrentan diariamente al problema de la obtención del camino más corto, para su utilización en la búsqueda de soluciones al problema del viajante de comercio (TSP), que busca el camino más corto que pase por una serie de puntos. Este problema, que de por sí es computacionalmente intratable en el caso general, se complica aún más cuando se tienen en cuenta restricciones de horarios, de carga o de número de vehículos utilizados.

En teoría de grafos, el problema del camino más corto es el problema de encontrar un camino entre dos vértices o nodos de un grafo, de tal forma que la suma de los pesos de las aristas que los conectan sea el menor posible. Este problema, cuya solución es computacionalmente viable, está en la raíz de muchos otros problemas de recorridos de grafos, como los que aparecen en telecomunicaciones al intentar buscar el camino con menor retraso en la propagación de la señal, o incluso en las conexiones entre personas en redes sociales. En el campo del transporte, el problema del camino más corto está en la raíz del problema del viajante de comercio (Travelling Salesman Problem), un problema NP-Hard de optimización combinatoria, de gran importancia en investigación operativa. El problema TSP tiene una gran importancia práctica, ya que condiciona por completo el funcionamiento de toda empresa destinada al transporte de pasajeros o mercancías.

Este TFG surge de la demanda tecnológica de la empresa GMV de una solución para el cálculo óptimo de rutas de transporte, con el fin de atacar posteriormente el problema TSP. Se requiere una batería de algoritmos, así como que el desarrollo no dependa de componentes comerciales, ni en lo que respecta a los algoritmos, ni en el software subyacente, ni tampoco en los mapas utilizados.

En este trabajo se presenta una solución basada en el cálculo de rutas óptimas con bases de datos, destinada a proporcionar una batería de algoritmos para resolver el problema del camino más corto o Shortest Path entre dos puntos. Dicha solución plantea un proyecto de Sistema de Información Geográfica (SIG) basado completamente en software libre. Se aborda su desarrollo y la evaluación del mismo como posible alternativa a cubrir la demanda tecnológica planteada por la empresa GMV.

1.1. Objetivos

El objetivo fundamental es probar y evaluar una solución existente para el cálculo de rutas óptimas a través de su consulta en bases de datos. De modo que permita aceptar o desechar dicha solución en base a los resultados recogidos de su estudio.

1.2. Resumen de la memoria

Para hacer frente a la demanda tecnológica de la empresa GMV de una solución para el cálculo óptimo de rutas de transporte, se planteo un proyecto de Sistema de Información Geográfica (SIG) basado completamente en software libre bajo las licencias BSD y GPL. Dicho SIG esta compuesto exactamente por tres módulos, cuyas funcionalidades se vinculan para crear un sistema que aporte soluciones al cálculo óptimo de rutas de transporte. Los módulos que lo componen son: PostgreSQL, PostGIS y pgRouting.

PostgreSQL es un Sistema Gestor de Bases de Datos (SGBD) objeto-relacional, con su código fuente disponible libremente. Este primer módulo es el que asume las funciones generales de una base de datos. En segundo lugar tenemos PostGIS, una extensión de la base de datos PostgreSQL que permite gestionar objetos geográficos, de forma que añade la capacidad de utilizar PostgreSQL como base de datos espacial en un Sistema de Información Geográfica. Y en la cima de este sistema, el modulo cuya funcionalidad permite valorar el conjunto como solución a esta demanda tecnológica, pgRouting. PgRouting es una extensión de PostgreSQL/PostGIS que añade la funcionalidad del cálculo de rutas, en concreto se puede utilizar para resolver el camino más corto o Shortest Path entre dos nodos o ejes de la red lineal. PgRouting implementa varios algoritmos para resolver el problema del camino más corto, siendo alguno de ellos: el algoritmo de Dijkstra junto con su versión bidireccional y el algoritmo Astar (también conocido como "A*") que también cuenta con su versión bidireccional.

Es sobretodo por la gran variedad de competencias que plantea pgRouting, por lo que el Sistema de Información Geográfica formado por las herramientas anteriormente comentadas, así como su posible interacción a mayores con otros sistemas, se considero una buena y rentable solución para hacer frente a esta demanda tecnológica. Siendo el objetivo principal de este trabajo probar y evaluar dicha solución, de modo que pueda ser aceptada o rechazada en base a los resultados obtenidos y recogidos en esta memoria.

Se comenzó realizando la instalación de cada uno de los módulos con que cuenta la plataforma, tras lo cual se procedió a la importación de los datos cartográficos que permitieran interactuar con el sistema. Fundamentalmente se trabajo con la cartografía libre OpenStreetMap, cumpliendo así el requisito de no utilizar mapas propietarios. Aunque posteriormente, la propia empresa GMV solicito la carga de la cartografía comercial TeleAtlas, al ser la cartográfica con la que habitualmente trabaja. Todo este proceso se realizo de forma progresiva comenzando a nivel de ciudad, país y finalizando con la carga del continente europeo.

Con la plataforma lista para ser utilizada, se procedió a desarrollar una batería de pruebas y un plan de evaluación, que consistió en la realización de pruebas evolutivas basadas en el cálculo de rutas para el mismo nodo, nodos adyacentes, nodos distantes de una misma ciudad, nodos pertenecientes a ciudades distintas de un mismo país y finalmente nodos pertenecientes a ciudades distintas de diferente país del continente europeo. Siendo la ruta Valladolid-Varsovia la que marcaría el desarrollo de este proceso y posterior evaluación.

Fue en este punto donde surgió un problema que ponía en riesgo la funcionalidad de la plataforma evaluada, así como la viabilidad de este proyecto. El Sistema de Información Geográfica desarrollado, no cumplía el objetivo de ofrecer una batería de algoritmos para atacar el cálculo de rutas óptimas debido al uso intensivo de la memoria por parte de las funciones de pgRouting. Lo que llevo a la necesidad de reali-

zar un profundo análisis de proximidad entre los nodos de la ruta, buscando generar un área de influencia que limitara la carga de datos en memoria y permitiera a las funciones realizar su labor. Proceso largo y basado en multitud de pruebas cuyos resultados se recogen en esta memoria. Pero finalmente se encontró solución a dicho problema, que ademas mejoraba los tiempos de respuesta obtenidos hasta el momento. A partir de este punto, se produjo un minucioso proceso de evaluación y comparativa del rendimiento, que permitió analizar profundamente la solución planteada, así como seguir mejorando ciertos aspectos en cuanto a los tiempos de respuesta obtenidos.

Una vez lograda la satisfacción del cliente con el rendimiento y la respuesta de la plataforma para el cálculo óptimo de rutas, se procedió a favorecer su uso en entornos de producción. Debido a como pgRoutnig implementa sus funciones y a los datos cartográficos importados, no era posible utilizar el sistema sin un procesamiento previo de las vías a tomar como origen y destino. Siendo necesaria la combinación de las cartografías OpenStreetMap y TeleAtlas, de modo que TeleAtlas ofreciera la información referente a la codificación geográfica de las vías y OpenStreetMap se utilizara para realizar los cálculos de ruta, ya que como se había comprobado, el sistema ofrecía mejores tiempos de respuesta con esta cartografía. Logrado este aspecto, el último punto consistió en desarrollar una interfaz web que facilitara la interacción con la plataforma.

Finalmente, en base a los resultados obtenidos a lo largo del desarrollo de la evaluación, puede concluirse que como solución basada en software libre presenta las ventajas e inconvenientes de estos tipos de sistemas. Pero que gracias a las soluciones planteadas a lo largo del desarrollo de este trabajo, se logra ofrecer una plataforma que cumple los requisitos exigidos por GMV y cuyos tiempos de respuesta se reducen notablemente respecto a los tiempos manejados por la empresa hasta el momento con su propio sistema. Convirtiendo así la solución planteada en una posibilidad a tener muy en cuenta a la hora de cubrir la demanda tecnológica planteada.

Capítulo 2

Análisis

En el presente capítulo se ofrece una visión de la metodología utilizada y el plan de trabajo seguido para el desarrollo de este proyecto, recogiendo un diagrama cuyo objetivo es exponer el tiempo de dedicación para las diferentes tareas o actividades a lo largo del tiempo que duro el mismo. También se ofrece un presupuesto estimado del coste real.

2.1. Metodología y plan de trabajo

La metodología aplicada para el desarrollo de este Sistema de Información Geográfica, se basa en el modelo iterativo incremental de desarrollo evolutivo de la ingeniería de software. Se considero el apropiado porque los requisitos centrales son conocidos de antemano, aunque no estén bien definidos a nivel detalle. Además, basar el desarrollo en este modelo permitió conseguir versiones cada vez más completas y complejas, hasta llegar al producto final; incluso evolucionar más allá, durante la fase de operación.

Bajo este modelo se obtuvieron versiones parciales a medida que se fue construyendo el producto final, llamadas incrementos. Cada versión emitida incorporaba, a los anteriores incrementos, los nuevos objetivos/requisitos o mejoras que fueron analizados como necesarios al comienzo de dicho incremento. En general, cada incremento se construyo sobre aquel que ya fue entregado. De esta manera no se dejaba para el final del proyecto ninguna actividad arriesgada relacionada con la entrega de requisitos.

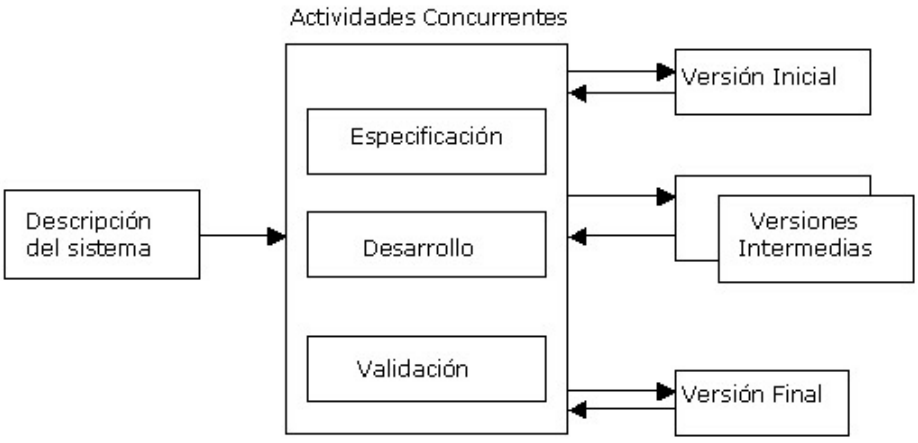


Figura 2.1: Modelo iterativo incremental.

Cada incremento se desarrollo siguiendo un modelo en cascada típico, por lo que al inicio de cada incremento se tuvo una especificación detallada de la funcionalidad esperada que se pretendía desarrollar en ese incremento. Así, cuando un incremento se completaba, se entregaba aportando una parte de la funcionalidad del sistema. Luego de cada integración se logro entregar un producto con mayor funcionalidad que el previo. El proceso se repitió hasta alcanzar el sistema final completo.

Un aspecto fundamental para guiar este tipo de desarrollo es la priorización de los objetivos/requisitos en función del valor que aportan al cliente. Por ello fue vital la disponibilidad del cliente, que se mantuvo constante durante todo el proyecto a través del tutor, dado que participa de manera continua:

- En el inicio de cada iteración, detallando (o habiendo detallado previamente) los requisitos que se iban a desarrollar.
- En la finalización de cada iteración, revisando los requisitos desarrollados.

La idea de la que se partió es que el primer incremento fuera un prototipo, con las funciones básicas. De modo que el cliente evaluara inicialmente este sistema básico, comprobando y valorando el resultado de su uso, de manera que pudiera aportar necesidades o mejoras al plan para el desarrollo de los posteriores incrementos. Además se considero la oportunidad de aportar otros factores, como son la priorización (mayor o menor urgencia en la necesidad de cada incremento en particular) y la dependencia entre incrementos (o independencia) por parte del cliente.

Centrándonos en el plan de trabajo, la dimensión temporal de la realización de este proyecto es de diez meses, durante los cuales se utilizaron iteraciones cortas de 1 a 2 semanas incrementando así la productividad del proyecto, dado que se tiene objetivos a corto plazo. Así mismo, con iteraciones cortas se logro un seguimiento minucioso y continuado de la evolución y resultados del proyecto por parte del tutor, con su correspondiente retroalimentación de las siguientes mejoras a incorporar.

Las tareas vinculadas a los incrementos desarrollados a lo largo de este periodo se detallan a continuación:

Incremento 1: El primer incremento consistió en la instalación del Sistema de Información Geográfica a evaluar, es decir, de los módulos PostgreSQL, PostGIS y pgRouting que lo integran. Realizando las pruebas oportunas que permitían verificar que la instalación y configuración de la plataforma era correcta, así como la funcionalidad para la que fue concebida.

Incremento 2: Tras la presentación del prototipo ante el tutor, se obtuvo realimentación enfocada a sus expectativas en cuanto a la cartografía a utilizar. Se opto por la cartografía libre OpenStreetMap, realizando una carga progresiva que se iniciara con la importación cartográfica a nivel de ciudad, país y finalmente el continente europeo. Se tuvo en cuenta el requerimiento por parte del cliente de probar la importación de la cartografía comercial TeleAtlas, dado que es la cartografía utilizada por la empresa GMV.

Incremento 3: Con el SIG completo en su versión inicial, se procede a testear las funciones que pgRouting implementa para resolver el camino más corto o Shortest Path entre dos nodos o ejes de la red, lo que constituye el tercer incremento. Los resultados obtenidos de una batería de pruebas son documentados y presentados ante el cliente/tutor, esperando que el trabajo se vaya refinado en base a los comentarios o nuevos aspectos requeridos por el cliente.

Incremento 4: Una vez lograda la satisfacción del cliente en cuanto a la solución evaluada para resolver el problema del camino más corto, se procede a abordar el desarrollo de una interfaz web que facilite y haga más agradable la interacción con el sistema en entornos de producción.

Los incrementos comentados recogen a grandes rasgos el proceso desarrollado durante la evaluación del sistema. No entran en detalles, pero permiten obtener una perspectiva del esquema seguido en la realización de este proyecto. De hecho, como se vera a lo largo de esta memoria, alguno de estos incrementos abarco gran parte del tiempo durante el que se realizo este trabajo, es el caso del tercer incremento. Fueron necesarias varias iteraciones para completar su cometido, de las que resultaron versiones cada vez mas eficientes del producto final perseguido.

2.2. Planificación

De acuerdo a lo comentado en el anterior apartado sobre la metodología utilizada y el plan de trabajo seguido, se puede observar gráficamente mediante un diagrama de Gantt, el periodo de tiempo dedicado a cada una de las actividades, al igual que para cada uno de los incrementos que las contienen:

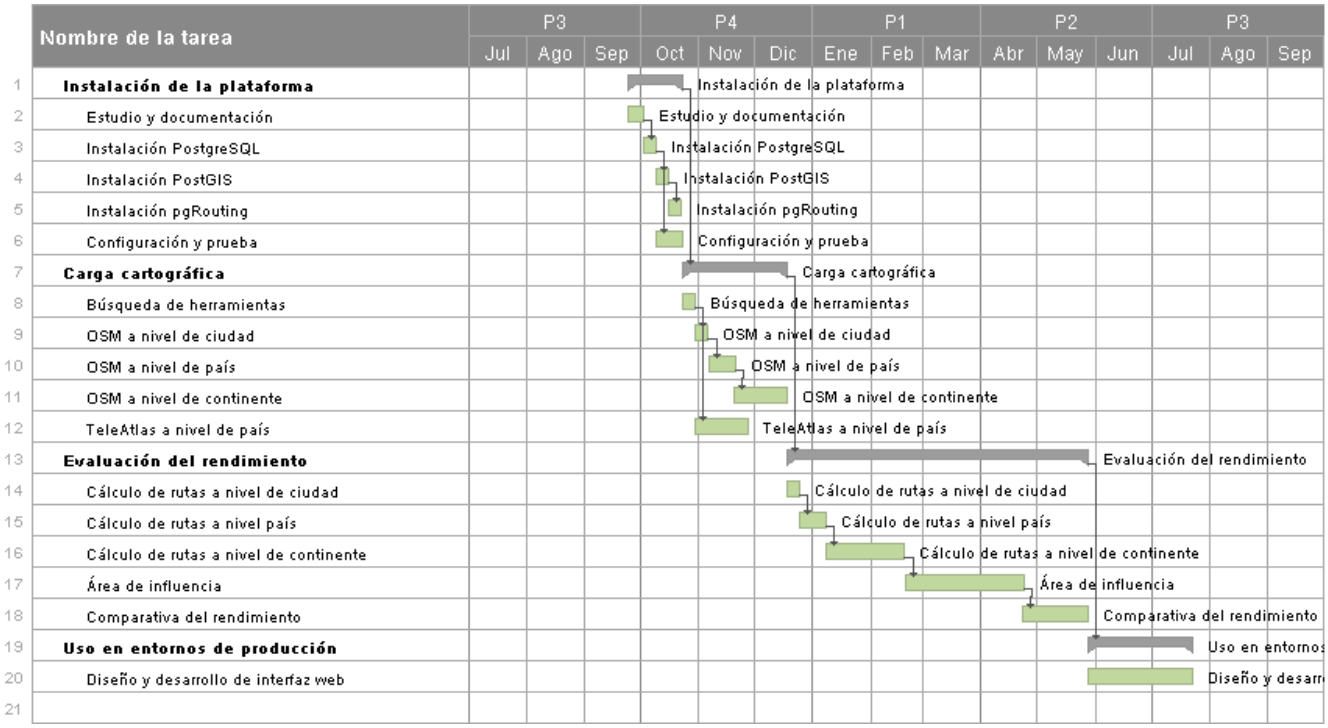


Figura 2.2: Cronograma de actividades.

Siendo la siguiente tabla la que recoge de forma más clara cada una de las tareas que componene cada uno de los cuatro incrementos en que se compone este proyecto. Puede observarse los tiempos invertidos en la realización de cada una, al igual que las relaciones de precedencia que las vinculan y que condicionan el inicio del posterior incremento.

	Nombre de la tarea	Fecha de inicio	Fecha de finalización	Duración	Predecesoras
1	**Instalación de la plataforma**	**24/09/14**	**22/10/14**	**25d**	
2	Estudio y documentación	24/09/14	01/10/14	7d	
3	Instalación PostgreSQL	02/10/14	08/10/14	6d	2
4	Instalación PostGIS	09/10/14	15/10/14	6d	3
5	Instalación pgRouting	16/10/14	22/10/14	6d	4
6	Configuración y prueba	09/10/14	22/10/14	12d	3
7	**Carga cartográfica**	**23/10/14**	**17/12/14**	**48d**	1
8	Búsqueda de herramientas	23/10/14	29/10/14	6d	
9	OSM a nivel de ciudad	30/10/14	05/11/14	6d	8
10	OSM a nivel de país	06/11/14	19/11/14	12d	9
11	OSM a nivel de continente	20/11/14	17/12/14	24d	10
12	TeleAtlas a nivel de país	30/10/14	26/11/14	24d	8
13	**Evaluación del rendimiento**	**18/12/14**	**27/05/15**	**138d**	7
14	Cálculo de rutas a nivel de ciudad	18/12/14	24/12/14	6d	
15	Cálculo de rutas a nivel país	25/12/14	07/01/15	12d	14
16	Cálculo de rutas a nivel de continente	08/01/15	18/02/15	36d	15
17	Área de influencia	19/02/15	22/04/15	54d	16
18	Comparativa del rendimiento	23/04/15	27/05/15	30d	17
19	**Uso en entornos de producción**	**28/05/15**	**22/07/15**	**48d**	13
20	Diseño y desarrollo de interfaz web	28/05/15	22/07/15	48d	

Figura 2.3: Duración y precedencia de actividades.

La duración de cada actividad sigue un patrón semanal, dado que como se comento en el anterior apartado, se utilizaron iteraciones cortas de 1 a 2 semanas para realizar el seguimiento del proyecto. Siendo las reuniones semanales, las que definirían en función de los resultados obtenidos hasta ese momento las siguientes acciones a desarrollar.

La duración aparece especificada en días, coincidiendo con los días desde que se da vía verde al inicio de la tarea hasta que se cierra en base a la conformidad con los resultados. Pero la inversión de tiempo real es de horas semanales, a la razón de 8 horas semanales en el primer cuatrimestre y 12 horas en el segundo. Tomando como días laborales de lunes a viernes, con sábado incluido.

No obstante, este diagrama y su correspondiente tabla se ajustan a los tiempos de dedicación resultantes. De la perspectiva inicial hubo desviaciones debidas a diferentes problemas que se irán desarrollando conforme se avance en esta memoria, lo que supuso un incremento en el coste de tiempo total estimado para la realización del proyecto. En concreto, dichas desviaciones tienen lugar en el incremento tres donde se trata la evaluación del rendimiento, al mismo tiempo que en el incremento cuatro donde se aborda la creación de una interfaz web para el sistema. Inicialmente, el coste de tiempo total estimado se había obtenido en base a tiempos que permitieran presentar el proyecto en el mes de julio, pero las desviaciones provocaron un aumento en los tiempos que supuso que no se cumplieran las fechas planificadas, siendo necesario modificar estas en vista a presentar en el mes de septiembre.

Comentar que ni el apartado anterior ni el diagrama presentado recogen la redacción de esta memoria y el período de tiempo invertido. Se redactó la memoria completa, aunando toda la información recogida

anteriormente junto con el código de la aplicación y los resultados y conclusiones finales, durante el mes de agosto. Pero se intento ir documentando el proyecto a la vez que se iba desarrollando, es por esto que esta fase esta solapada en el tiempo.

2.3. Presupuesto

En este apartado se pretende realizar una estimación del coste de este proyecto si hubiera sido desarrollado profesionalmente. Para ello se toman en cuenta las siguientes consideraciones iniciales:

- El proyecto lo va a desarrollar una sola persona con un nivel profesional equivalente al de Ingeniero Junior.

- El período de desarrollo del proyecto abarca unos diez meses, en los que no se ha seguido una jornada de trabajo continua. Es decir, no se ha seguido por ejemplo una jornada de oficina de 8 horas al día, sino que se ha ido realizando según la disponibilidad de tiempo (incluyendo trabajo en sábados, domingos y festivos). Se han ido apuntando diariamente las horas de trabajo, y gracias a esto podemos tener el cálculo de las horas totales de trabajo.

Vamos a dividir el presupuesto en dos bloques: recursos humanos y materiales. Pasamos a describirlos.

- **Recursos humanos**

Durante la realización del trabajo fin de grado se han ido anotando las horas de trabajo diarias de forma aproximada. El resultado total se ha alcanzado tras unas 454 horas de trabajo.

Para los costes de los recursos humanos se considera el siguiente coste por hora:

*** Ingeniero Junior..... 13€/h

Teniendo en cuenta lo anterior, los gastos en recursos humanos serían:

*** Ingeniero Junior (13€/h x 454 h) ..... 5902€

Resumimos lo anterior en una tabla:

Elemento	Coste(€)
Recursos humanos: Ingeniero Junior	5902
COSTE TOTAL RECURSOS HUMANOS	**5902**

Tabla 2.1: Coste de los recursos humanos.

- **Recursos materiales**

En cuanto a los recursos materiales los podemos desglosar en dos aspectos, uno referido al coste del software utilizado y otro al hardware y consumibles.

Con respecto al software necesario, hay que destacar que todo el software utilizado: PostgreSQL, PostGIS, pgRouting, herramientas de importación, así como la cartografía base para el desarrollo del proyecto, es de libre distribución. Y por tanto, el coste que supone es de 0€. El único coste relativo al software es el del sistema operativo usado, en este caso al tratarse de Ubuntu Server también implica un coste de 0€.

Con respecto al hardware necesario para este proyecto (incluimos también en esta tabla el coste de los consumibles):

Elemento	Coste(€)
Servidor	1400
Electricidad consumida	80
Conexión a Internet ADSL (30€/mes x 10 meses)	300
Material de oficina: papel, tinta impresora, bolígrafos, etc.	40
COSTE TOTAL HARDWARE Y CONSUMIBLES	**1820**

Tabla 2.2: Coste del hardware y los consumibles.

- **Resumen del presupuesto**

Todo el desglose anterior lo resumimos en la siguiente tabla:

Coste	Coste(€)
Recursos humanos	5902
Software	0
Hardware y consumibles	1820
COSTE TOTAL PROYECTO	**7722**

Tabla 2.3: Coste total del proyecto.

Por tanto, el presupuesto estimado para la ejecución material del proyecto, asciende a la cantidad de siete mil setecientos veintidós Euros (7722€).

Capítulo 3

Puesta en marcha del entorno de trabajo

En este capítulo se recoge el proceso inicial llevado a cabo para poder abordar la prueba y evaluación de la solución planteada. Este proceso consiste en generar un prototipo inicial de la propuesta. Se recoge desde el inicio del proyecto, basado en el estudio y documentación de los módulos que integran el SIG, el proceso de instalación, las herramientas utilizadas para la carga cartográfica, hasta el modo de utilización de pgRouting en el cálculo de rutas. Se pretende que el lector obtenga una visión del proceso lo más clara y fiel posible.

3.1. Características de la máquina

Se parte de una máquina virtual para realizar la instalación del prototipo, a continuación se detallan sus características:

- Ubuntu Server (modo comando)
- 6Gb de RAM y 4 cores
- Dos discos, el primero para el sistema 10G. Y el segundo de 400G para datos de usuario y bases de datos, montado en /home
- Acceso vía SSH desde fuera de la Uva
- Dominio: gis.infor.uva.es

3.2. Prototipo basado en software libre

Como se ha comentado en el capítulo anterior, este proyecto pretende probar y evaluar una solución existente para el cálculo de rutas óptimas a través de su consulta en bases de datos.

La solución con la que se ha trabajado a lo largo de este proyecto consiste en un Sistema de Información Geográfica (SIG) basado completamente en software libre bajo las licencias BSD y GPL. Dicho SIG esta compuesto exactamente por tres módulos, cuyas funcionalidades se vinculan para crear un sistema que aporta soluciones para el cálculo óptimo de rutas de transporte. En la siguiente figura se muestra el esquema de módulos que integran este SIG:

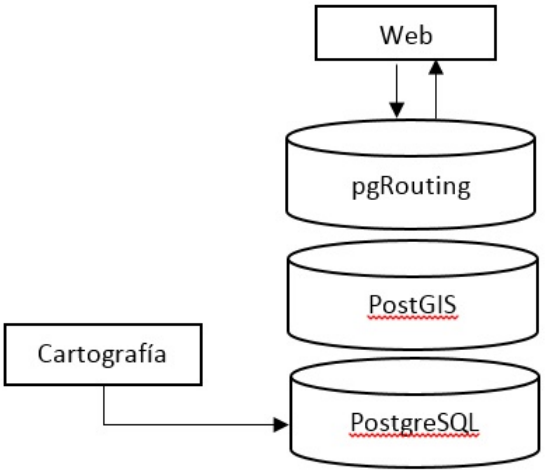


Figura 3.1: Esquema de módulos del SIG.

Como puede observarse, los módulos que lo componen son:

- PostgreSQL: Módulo que recoge las funciones generales de una base de datos.
- PostGIS: Extensión que convierte a PostgreSQL en una base de datos espacial.
- pgRouting: Extiende la base de datos espacial PostgreSQL/PostGIS para proporcionar funcionalidades de enrutado.

A continuación se procede a describir brevemente cada uno de ellos y lograr así una visión global de la funcionalidad final obtenida.

3.2.1. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional (ORDBMS), muy conocido y usado en entornos de software libre porque cumple los estándares SQL92 y SQL99, y también por el conjunto de funcionalidades avanzadas que soporta, lo que lo convierte en el SGBD de código abierto más potente del mercado situándose al mismo o a un mejor nivel que muchos SGBD comerciales.

PostgreSQL se distribuye bajo licencia BSD (Berkeley Software Distribution), lo que permite su uso, redistribución y modificación con la única restricción de mantener el copyright del software a sus autores, en concreto el PostgreSQL Global Development Group y la Universidad de California.

PostgreSQL utiliza un modelo cliente/servidor y se basa en multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Así, un fallo en uno de los procesos no perjudicará al resto y el sistema continuará funcionado. De hecho, sus características técnicas y arquitectura la hacen una de las bases de datos más potentes y robustas del mercado. PostgreSQL ofrece un gran rendimiento con grandes cantidades de datos y una alta concurrencia de usuarios accediendo al mismo tiempo en el sistema. Su desarrollo tiene una antigüedad superior a los 16 años, y durante este tiempo, estabilidad, potencia, robustez, facilidad de administración e implementación de estándares han sido las características que más se han tenido en cuenta durante su desarrollo.

3.2.2. PostGIS

PostGIS es una extensión de la base de datos que permite gestionar objetos geográficos, de forma que añade la capacidad de utilizar PostgreSQL como base de datos espacial en un Sistema de Información Geográfica. Ha sido desarrollado por Refractions Research como proyecto de software libre bajo licencia GPL.

PostGIS está desarrollado en lenguaje C, C++ y PL/PgSQL (lenguaje procedural de PostgreSQL) y utiliza entre otras las bibliotecas de software libre GEOS (análisis espacial) y Proj4 (proyecciones). PostGIS utiliza GEOS para realizar los cálculos geométricos (tipos de objetos geométricos, relaciones y operaciones de análisis espacial entre geometrías, etc.).

Debido a que PostGIS está construido sobre PostgreSQL, se beneficia de forma directa de todas sus características, así como de los estándares abiertos. PostGIS agrega tipos adicionales (geometría, geografía, mapa de bits y otros) a PostgreSQL. También agrega funciones, operadores y mejoras de índice que se aplican a estos tipos espaciales. Estas funciones adicionales, operadores, índices y tipos aumentan la potencia del núcleo PostgreSQL DBMS, haciendo de este un sistema de gestión de base de datos espacial rápido y robusto.

La implementación está basada en geometrías e índices "ligeros" optimizados para reducir el uso de disco y memoria. Utilizando geometrías ligeras ayuda al servidor a incrementar la cantidad de datos migrados desde el almacenamiento físico en el disco hacia la RAM, mejorando el desempeño de las consultas sustancialmente. A día de hoy es un producto afianzado, que ha demostrado su eficiencia en cada versión. En relación con sus competidores, PostGIS ha demostrado ser muy superior a la extensión geográfica de la nueva versión de MySQL, y a juicio de muchos, es muy similar a la versión geográfica de la base de datos Oracle. Es una alternativa real al software propietario superándole en estabilidad y rapidez.

Actualmente la combinación PostgresSQL/PostGIS es la base de datos espacial de código abierto más ampliamente utilizada. Muchas y muy variadas organizaciones de todo el mundo usan esta combinación, incluyendo agencias gubernamentales de riesgos adversos y organizaciones que almacenan terabytes de datos y sirven millones de peticiones web al día.

3.2.3. pgRouting

PgRouting es una extensión de PostgreSQL/PostGIS que añade la funcionalidad del cálculo de rutas, en concreto se puede utilizar para resolver, entre otros, los siguiente problemas:

- Resolver el camino más corto o Shortest Path entre dos nodos o ejes de la red lineal (dispone de tres algoritmos diferentes, que se comentarán más adelante).
- Problema del viajante o Traveling Salesperson Problem (TSP). Si imaginamos un comerciante que debe visitar una serie de ciudades distintas el problema a resolver cosiste en encontrar una ruta óptima que pase una única vez por cada una de las ciudades minimizando la distancia total recorrida por el comerciante.
- Problema de distancia de conducción o Driving Distance calculation (DD).

En este proyecto se va a centrar en resolver el problema del camino más corto. El camino más corto consiste en el cálculo de la ruta entre dos puntos (nodos) de forma que la suma de los costes de los tramos constituyentes es minimizada. El coste de un tramo puede basarse en su longitud, el tiempo en atravesarlo o cualquier otra variable.

pgRouting implementa tres algoritmos diferentes para resolver el problema del camino más corto entre dos puntos o ejes de una red lineal:

- Algoritmo de Dijkstra. Este algoritmo devuelve el camino más corto (de todos los posibles) entre dos nodos de una red. Para ello, no se necesita conocer las coordenadas de los puntos de los nodos de la red, sino simplemente disponer de la numeración de los puntos inicial y final de cada tramo así como su longitud. pgRouting ofrece además la versión bidireccional para este algoritmo.
- Algoritmo Astar (también conocido como "A*"). Además de los números de los nodos inicial y final de cada tramo y su longitud, utiliza para los cálculos las coordenadas X e Y de dichos nodos. Mediante un algoritmo heurístico es capaz de encontrar un camino que tiene una alta probabilidad de ser uno de los más cortos entre dos nodos de la red. Al igual que el algoritmo de Dijkstra, cuenta con su versión bidireccional.
- Algoritmo Shooting Star. Es un algoritmo heurístico que necesita también las coordenadas de los nodos como el Astar, pero añade la posibilidad de añadir restricciones a la red, por ejemplo, giros no permitidos.

pgRouting está disponible bajo la licencia GPLv2 y cuenta con el apoyo de una comunidad cada vez mayor de personas, empresas y organizaciones.

3.3. Instalación de la plataforma

Aclarar que la primera instalación se realizo a partir del código fuente para poder adaptar el sistema a nuestras necesidades. Que se desarrollo un manual de instalación con el proceso llevado a cabo en cada uno de los módulos. Lo cual se considera excesivo añadirlo a este apartado, por lo que dichos manuales se encuentran incluidos en la entrega de esta memoria.

Como aspectos interesantes a destacar, comentar la configuración del módulo PostgreSQL que se realizo para optimizar la plataforma, así como el proceso que debe realizarse a la hora de crear una base de datos espacial sobre la misma.

3.3.1. Optimizar la configuración de PostgreSQL

PostgreSQL se puede empezar a utilizar nada más terminar de instalarlo y después de inicializar nuestro "cluster", sin necesidad de configurar nada. Pero si vamos a utilizar PostgreSQL para algo importante y con cierto volumen de datos y usuarios es imprescindible que lo configuremos para dicho trabajo. No es una gran ayuda tener un servidor con varios GBytes de memoria RAM si le hemos dicho a PostgreSQL, por ejemplo, que no utilice más de 32MBytes.

El comportamiento de PostgreSQL se puede controlar con el fichero "postgresql.conf". En este fichero podemos cambiar todos los parámetros de configuración que afectan al funcionamiento y al comportamiento de PostgreSQL en nuestra máquina. Los cambios que realicemos en este fichero afectaran a todas

las bases de datos que tengamos definidas en nuestro cluster. Para que dichos cambios sean efectivos, bastara con hacer un simple restart del servidor PostgreSQL.

Esta fue una de las tareas realizadas durante el proceso de instalación de la plataforma, explorar las opciones de optimización que tiene PostgreSQL. Se analizó la optimización en la configuración general de PostgreSQL (fichero postgresql.conf) de acuerdo con el hardware disponible, desde lo cual se observo una mejora en el arranque del servicio y en los tiempos de respuesta de las consultas SQL a partir de las modificaciones realizadas.

A continuación vamos a ver los parámetros más importantes que deberíamos cambiar si empezamos a usar PostgreSQL para un uso serio y si queremos sacarle el máximo partido a nuestra máquina. Existen muchos más parámetros, pero aquí solo nos vamos a centrar en los más importantes:

- max_connections: Número máximo de clientes conectados a la vez a nuestras bases de datos.
- shared_buffers: Este parámetro es muy importante y define el tamaño del buffer de memoria utilizado por PostgreSQL. No por aumentar este valor mucho tendremos mejor respuesta. En un servidor dedicado podemos empezar con un 25 % del total de nuestra memoria. Nunca más de 1/3 (33 %) del total.
- work_mem: Usada en operaciones que contengan ORDER BY, DISTINCT, JOINS, etc. En un servidor dedicado podemos usar un 2-4 % del total de nuestra memoria si tenemos solamente unas pocas sesiones (clientes) grandes.
- maintenance_work_mem: Usada en operaciones del tipo VACUUM, ANALYZE, CREATE INDEX, ALTER TABLE, ADD FOREIGN KEY. Su valor dependerá mucho del tamaño de nuestras bases de datos.
- effective_cache_size: Parámetro usado por el "query planner" de nuestro motor de bases de datos para optimizar la lectura de datos. En un servidor dedicado podemos empezar con un 50 % del total de nuestra memoria. Como máximo unos 2/3 (66 %) del total.
- checkpoint_segments: Este parámetro es muy importante en bases de datos con numerosas operaciones de escritura (insert, update, delete). Para empezar podemos empezar con un valor de 64. En grandes databases con muchos GBytes de datos escritos podemos aumentar este valor hasta 128-256.

Es muy importante tener en cuenta que al aumentar los valores por defecto de muchos de estos parámetros, tendremos que aumentar los valores por defecto de algunos parámetros del kernel de nuestro sistema. La información detallada de como hacer esto se encuentra en la documentación oficial de PostgreSQL.

En fin, este es proceso delicado que cuenta con multitud de parámetros cuya función se debe conocer bien para no dañar el funcionamiento del sistema y obtener el mayor rendimiento posible. Pero este proceso puede ser más fácil con la ayuda de la herramienta PgTune, especialmente diseñada para calcular los parámetros óptimos. Su uso e instalación es muy básico. Simplemente obtenemos la herramienta mediante repositorio y con una simple sentencia como:

```
postgres@gis:~$ pgtune -i /postgresql.conf -o postgresql_opt.conf
```

Se genera un nuevo archivo de configuración nombrado postgresql_opt.conf que podemos ya usar para nuestro servicio. Resguardamos la configuración actual, luego sobrescribimos con nuestra nueva configuración y reiniciamos el servicio.

3.3.2. Creación de una base de datos espacial

Tras instalar la plataforma, aún no se puede utilizar su funcionalidad en ninguna base de datos. Para ello, se necesita dotar a la base de datos que queramos utilizar con PostGIS y pgRouting de la funcionalidad espacial y de enrutado requerida.

Para crear una base de datos con soporte PostGIS y pgRouting, se tienen que ejecutar varios comandos desde la consola del sistema operativo. Alguno de estos pasos requiere permisos de administrador de PostgreSQL. Por ello, con el fin de simplificar las cosas vamos a utilizar el usuario postgres que generalmente es administrador del cluster de PostgreSQL, utilizado para detallar el proceso desarrollado.

Bien, en primer lugar es necesario crear una base de datos en PostgreSQL, para lo cual utilizamos el comando "createdb" que al ser un comando del sistema operativo se puede ejecutar directamente desde una terminal o símbolo del sistema.

```
postgres@gis:~$ createdb -U postgres prueba encoding='UTF8'
```

Este comando crea la base de datos "prueba" cuyo dueño es el usuario postgres (al haberla creado). Si el usuario de PostgreSQL tiene el mismo nombre que el usuario que ha abierto el terminal del SO no seria necesario utilizar la opción -U. En tal caso la sentencia anterior sería similar a: createdb prueba encoding='UTF8'.

Conectamos con la base de datos creada a través del cliente psql y ejecutamos el siguiente SQL para habilitar tanto la funcionalidad de PostGIS como la de pgRouting:

```
postgres@gis:~$ psql prueba
psql (9.3.6)
Type "help" for help.

prueba=# create extension postgis;                 - Habilitar PostGIS (raster)
prueba=# create extension postgis_topology;        - Habilitar Topología
prueba=# create extension fuzzystrmatch;           - fuzzy matching para Tiger
prueba=# create extension postgis_tiger_geocoder;  - Habilitar US Tiger Geocoder
prueba=# create extension pgrouting;               - Habilitar pgRouting
```

Mediante el comando psql \dx listamos las extensiones instaladas para la base de datos actual, comprobando que las extensiones anteriormente creadas se encuentran disponibles en la base de datos. Además, podemos comprobar la versión tanto de PostgreSQL como de PostGIS o pgRouting instalada con las sentencias:

```
prueba=# select version();
PostgreSQL 9.3.6 on x86_64-unknown-linux-gnu, compiled by gcc (Ubuntu 4.8.2-19ubuntu1)
4.8.2, 64-bit

prueba=# select postgis_full_version();
POSTGIS="2.1.2 r12389" GEOS="3.4.2-CAPI-1.8.2 r3921" PROJ="Rel. 4.8.0, 6 March 2012"
GDAL="GDAL 1.10.1, released 2013/08/26" LIBXML="2.9.1" LIBJSON="UNKNOWN" TOPOLOGY RASTER
```

```
prueba=# select version, boost from pgr_version();
(2.0.0,v2.0.0,3,fbbaa2a,master,1.54.0)
```

Las funciones anteriores devuelven la versión del módulo instalado así como las versiones de las bibliotecas incluidas. Es una buena forma de comprobar que todo el proceso se ha realizado correctamente.

3.4. Importación de cartografía

Una cuestión relevante en este trabajo ha sido considerar y percatarse de la importancia que tiene para estos sistemas la información geográficamente referenciada. El SIG implementado no esta vinculado a ninguna fuente de datos cartográficos concreta. Existiendo así la posibilidad de utilizar mapas propietarios o libres. Solo es necesario que la estructura y diseño de la cartografía con que se desea trabajar pueda ser adapta a los requisitos y necesidades del SIG planteado.

Una misma cartografía se puede ver de formas diferentes, con atributos diferentes y aspectos bastante diferenciados en función del tratamiento al que vaya destinada, en este caso, al cálculo de rutas. Actualmente los cuatro proveedores de mapas más populares en el mundo son cuatro: Google Maps, NavTeq, TeleAtlas y OpenStreetMap. De entre estas cuatro alternativas se importo la cartografía proveniente de TeleAtlas y de OpenStreetMap. La primera al ser un requisito priorizado por la empresa GMV, ya que actualmente es la cartografía con la que trabaja. Y la segunda por que nos ofrece las ventajas de ser una alternativa libre frente al resto de alternativas comerciales, además de aparecer múltiples referencias a la misma en la documentación de los distintos módulos que integran la plataforma. Lo que indica que es una fuente de datos cartográficos afianzada para el uso de esta plataforma.

Esta sección pretende ofrecer una visión general de los conocimientos necesarios que tuvieron que ser adquiridos para realizar esta tarea. Así como los inconvenientes y aspectos a destacar durante el desarrollo de la misma. Inicialmente se presenta el proceso de importación de los datos geográficos procedentes de TeleAtlas, para posteriormente detallar más profundamente el largo y complicado proceso que supuso la importación de la cartografía de OSM.

3.4.1. TeleAtlas: Importar cartografía a PostGIS

Cartografía de TeleAtlas

TeleAtlas es sin duda el líder europeo en la creación de cartografía digital con una clara estrategia de neutralidad que le permite concentrarse en la única tarea que desarrolla, la creación y comercialización de mapas digitales. Dicha cartografía tiene un carácter comercial y es con la que actualmente trabaja la empresa GMV, de ahí la importancia de comprobar la posibilidad de cargar dicha cartografía en el SIG desarrollado.

TeleAtlas cuenta con dos tipos de productos disponibles en la actualidad: Multinet y StreetNet. Multinet es su producto de gama alta y contiene muchas características que fueron creadas para los navegadores GPS en automóvil. Una copia de dicho producto fue facilitada por la empresa GMV para desarrollar esta tarea.

La información geográfica contenida en la base de datos geográfica MultiNet esta estructurada de acuerdo con un modelo de datos en capas y en un formato SIG estándar como es shapefile (ver el siguien-

te apartado). Los mapas se almacenan de acuerdo a áreas geográficas, estructuradas en una jerarquía de directorios, dividiendo sus datos en capas organizadas conceptualmente en 13 unidades temáticas. Estas son: Road and Street Network, Ferry Connections, Address Area Boundaries, Railways, Points of Interest, Settlement, Water Areas and Water Lines, Land Use and Land, Built-up Areas, Administrative Areas and Places, Postal Districts, Other Named Areas y Structures (bridges and tunnels).

Dicha distribución no fue nada fácil de entender y menos aún trabajar con ella, pero gracias a los manuales que se distribuyen con este producto, aunque muy esquemáticos, se pudo sacar adelante. Para las funciones de enrutado, la unidad que nos interesa es "Road and Street Network". Esta unidad contiene una serie de capas, siendo la capa "NW Network, Geometry with Basic Attributes" la que esta especialmente destinada a las aplicaciones de enrutado de vehículos.

Al importar esta capa en un SIG, se crea una tabla cuya estructura cuenta con multitud de campos para las operaciones de enrutado. En la siguiente tabla se recogen algunos de los atributos necesarios para poder realizar estas operaciones:

Campo	Tipo	Descripción
gid	integer	Clave primaria
id	double precison	Identificador de vía
frc	smallint	Tipo de vía
feattyp	smallint	Tipo de elemento de transporte
f_jnctid	double precision	Identificador del nodo origen
t_jnctid	double precision	Identificador del nodo destino
name	character varying(100)	Nombre oficial de la vía
namelc	character varying(3)	Código de idioma
meters	double precision	Longitud (en metros)
kph	smallint	Límite de velocidad (en km/h)
minutes	double precision	Tiempo de viaje (en minutos)
geom	geometry(MultiLineString,4326)	Geometría (coordenadas)

Tabla 3.1: NW Network, Geometry with Basic Attributes.

Es a partir de dicha estructura de información, mediante la que se obtienen los datos cartográficos que permiten realizar funciones de enrutado en el SIG desarrollado, utilizando la cartografía TeleAtlas.

Formato Shapefile

El formato Shapefile (SHP) es un formato vectorial de almacenamiento digital donde se guarda la localización de los elementos geográficos y los atributos asociados a ellos. Es el formato más extendido y popular entre la comunidad SIG, de hecho es el formato utilizado por TeleAtlas para distribuir su cartografía. Es un formato propiedad de la compañía ESRI, pero es difícil encontrar un SIG que no lea este sistema de archivos. Aspecto por el cual se ha convertido en el formato estándar de facto para el intercambio de información geográfica.

Es un formato multiarchivo, es decir se compone de varios archivos que un cliente SIG lee como uno único. El mínimo requerido es de tres y tienen las siguientes extensiones:

- Shape (.shp) : Se trata del archivo principal y almacena la información geométrica de los elementos de la capa en formato vectorial. Pueden contener puntos, líneas o polígonos y cada vértice lleva

implícitas sus coordenadas en un sistema de referencia concreto (que por lo general se especifica en el archivo project). Se componen de una cabecera con información general sobre el tipo de shapefile y un número variable de registros, que a su vez pueden estar compuestos por varias entidades geométricas independientes.

- Shape Index (.shx) : Consiste en un índice de las entidades geométricas que permite refinar las búsquedas dentro del archivo .shp.

- dBase (.dbf) : Es la base de datos, en formato dBASE. Se trata de una tabla de datos en la que se registran los atributos de cada elemento. Es un formato con larga historia, muy compatible y sencillo que nos permite almacenar datos estructurados. En los shapefiles, las tablas dBase se emplean para asignar atributos numéricos, de texto o de fecha a los registros contenidos en el archivo principal. Cada registro debe estar asociado con una única entrada en la tabla, ambos archivos se vinculan mediante un número de registro en el archivo principal y el código en la tabla (OBJECTID).

Además de estos tres archivos requeridos, opcionalmente se pueden utilizar otros para mejorar el funcionamiento en las operaciones de consulta a la base de datos, información sobre la proyección cartográfica, o almacenamiento de metadatos. Estos archivos son: .prj, .sbn, .sbx, .fbn, .fbx, .ain, .aih, .shp.xml.

Habitual entre los anteriores suele ser el archivo Projection (.pjr). No es indispensable como se ha mencionado, pero permite georeferenciar los datos geométricos que poseemos en el Shape. Con el archivo Shape (.shp) definimos geométricamente una serie de elementos en el espacio , pero si queremos situar dicho elemento sobre el terreno necesitamos referir los datos a un sistema de coordenadas. Los datos necesarios por lo general están contenidos en este fichero.

Un shapefile se puede convertir a otros tipos de formatos con relativa facilidad. Podemos importarlos a una base de datos espacial como PostGIS.

Importación de un fichero shapefile con las utilidades de PostGIS

Este apartado aborda cómo importar shapefiles a una base de datos PostGIS. Para ello hay varias opciones, las más extendidas son:

- Mediante el uso del comando shp2pgsql, que se invoca desde línea de comandos. PostGIS incluye la herramienta shp2pgsql por defecto.

- Mediante el uso de la versión gráfica del importador shp2pgsql (el plugin pgShapeLoader).

- Mediante el uso de un cliente SIG de escritorio (por ejemplo, QGIS con el complemento PostGIS manager).

De entre estas opciones, se trabajo con la utilidad shp2pgsql que permite convertir shapefiles a tablas PostGIS (y viceversa con pgsql2shp). El comando shp2pgsql se invoca desde la consola y aunque puede resultar menos agradable que la versión gráfica, al contar con más opciones es más potente.

Procedimiento

shp2pgsql convierte un archivo shapefile en una serie de comandos SQL para crear una tabla espacial y añadir las geometrías importadas del shape mediante órdenes Insert. La salida de este comando puede

ser capturada en un archivo SQL o canalizada directamente al comando psql, que ejecutará los comandos contra una base de datos de destino.

Preparación

1. Se selecciona el archivo shapefile que se desea cargar, junto con todos los archivos asociados al mismo: .shp, .shx y .dbf y así sucesivamente.

2. Se identifica el SRID (ïdentificador de referencia espacial") de los datos.

   Un problema común es averiguar qué número SRID utilizar para los datos, es decir, el sistema de coordenadas. Todo lo que se tiene es un archivo .prj asociado al shapefile. Pero, ¿cómo se traduce un archivo .prj en el número SRID correcto?

   La respuesta es usar una herramienta realmente simple llamada Prj2EPSG (`http://prj2epsg.org`) que permite cargar y convertir el contenido del archivo .prj a un código SRID.

3. Se identifica la base de datos de destino en la que se desea cargar los datos.

Cargando datos

1. Ejecutar el comando shp2pgsql y canalizar la salida al comando psql para cargar el archivo shapefile en la base de datos en un solo paso. La sintaxis recomendada es:

   ```
   shp2pgsql -I -a -s <SRID> <PATH/TO/SHAPEFILE> <DBTABLE> | psql -d <DATABASE>
   ```

   Los parámetros de comando son:

   - <SRID> Identificador de referencia espacial.
   - <PATH/TO/SHAPEFILE> Ruta de acceso al archivo shapefile.
   - <DBTABLE> Nueva tabla espacial que será creada.
   - <DBASE> Base de datos donde se creará la tabla.

   ```
   shp2pgsql -I -a -s 4326 ../espe02_______nw.shp esp_eur2014 | psql -d teleatlas
   ```

   La opción -I creará un índice espacial después de crear la tabla. Esto es muy recomendable para mejorar el rendimiento. Por defecto, shp2pgsql crea código SQL para crear una nueva tabla y poblar esta con los datos contenidos en el archivo shapefile. La opción -a indica que solo se proceda a añadir datos, sin generar la tabla. Se recomienda que dicha tabla sea previamente generada con esta misma sentencia, sustituyendo la opción -a por la opción -p que sólo produce el código SQL que crea la tabla, sin añadir datos. Así se evita que en posteriores importaciones se generen errores.

2. Si se desea capturar los comandos SQL, se debe canalizar la salida a un archivo:

   ```
   shp2pgsql -I -a -s <SRID> <PATH/TO/SHAPEFILE> <DBTABLE> > SHAPEFILE.sql
   ```

   El archivo se puede cargar en la base de datos más tarde mediante la siguiente ejecución:

   ```
   psql -d <DATABASE> -f SHAPEFILE.sql
   ```

El shapefile finalmente se ha importado como una tabla en la base de datos PostGIS. Se puede verificar ejecutando la siguiente consulta en la línea de comandos:

```
psql -U <USERNAME> -d <DATABASE> -c "\d"
```

Carga por lotes

Aunque es posible ejecutar el comando shp2pgsql tantas veces como sea necesario, puede ser más eficiente crear un archivo por lotes para cargar varios shapefiles. Para ello se crea un shell script, por ejemplo loadfiles.sh, que recorre los directorios donde se ubican los shapefiles y para cada shape crea el correspondiente código SQL. Posteriormente, dicho código SQL se ejecuta en la base de datos.

A continuación se muestra el contenido del script generado en el desarrollo de este trabajo. Con dicho script se realizo la carga de los shapefiles de la capa "NW Network, Geometry with Basic Attributes" de TeleAtlas, destinada a las aplicaciones de enrutado de vehículos.

```
#!/bin/bash
for dir in /home/postgres/eur2014_03/shpd/mn/esp/*/
do
    dir=${dir%*/}
    cd $dir/
    for f in *nw.shp
    do
        shp2pgsql -I -s 4326 -a $f esp_eur2014 > /home/postgres/Teleatlas/${f}.sql
    done
done

for f in /home/postgres/Teleatlas/*.sql
do
    psql -d teleatlas -f $f
done
```

Este script asume que todos los archivos tienen la misma proyección.

3.4.2. Importación de datos OSM en PostGIS

OpenStreetMap

El proyecto OSM (OpenStreetMap) es un proyecto colaborativo para crear cartografía gratuita y de libre uso. Actualmente se distribuye bajo Licencia Abierta de Bases de Datos (ObdL), basada en la licencia Creative Commons. Por lo tanto, cualquier persona tiene derecho a copiar, distribuir, comunicar, transformar y comercializar dicha cartografía con la única condición de decir quien la ha producido y utilizar una licencia igual o compatible en la creación de cualquier producto derivado.

Dado el carácter voluntario del proyecto, la cobertura de información varía por zonas. No obstante, su base de datos guarda una inmensa cantidad de información geográfica que crece constantemente. Para

hacernos una idea, en la última reseña que se consulto durante la documentación de esta memoria, el archivo Planet.osm que se genera semanalmente con los datos del proyecto (todos los nodos, vías y relaciones que conforman el mapa) almacenado en formato XML tenia un tamaño de 43Gbytes comprimido.

OSM integra en una única base de datos los datos que terceros han liberado, combinando en un único lugar y con un único formato datos de todo tipo y de todos los países del mundo: desde calles y carreteras, hasta edificios y parques, comercios, lugares naturales, rutas de transporte público, tendido eléctrico, y todo lo que uno pueda imaginar.

Hay varias maneras de obtener la información de la base de datos de OpenStreetMap. Por una parte, es posible descargarla como un único fichero de Planet OSM (`http://planet.osm.org/`. Este fichero se actualiza con una periodicidad semanal, y se ofrece en los formatos PBF ("Protocolbuffer Binary Format") y XML. Pero se trata de un fichero de gran tamaño (28 GB en formato PBF, 43 GB en forma de fichero XML comprimido).También están disponibles extractos limitados por contenido o por el área geográfica que cubren, pudiéndose encontrar en Geofabrik (`http://download.geofabrik.de/`) o desde la web principal de OSM, desde el menú exportar, podemos seleccionar el área a exportar en formato XML.

Además, existen multitud de herramientas diseñadas e implementadas para la importación de la cartografía OSM a un Sistema de Información Geográfica compuesto por la estructura PostgreSQL/PostGIS. La elección de cual utilizar dependerá de la finalidad con que se vaya a manipular la información cargada.

Estructura de la base de datos OSM

En este apartado se refleja brevemente como es la estructura de la base de datos OSM, para entender como es posible seleccionar capas de información a partir de los datos estándar OSM.

OpenStreetMap representa características físicas sobre el terreno (por ejemplo, carreteras o edificios) mediante etiquetas asociadas a sus elementos básicos (nodos, vías y relaciones). Cada etiqueta describe un "atributo geográfico" de la función que se muestra en ese nodo específico, vía o relación.

- Elementos

  La base de datos OSM esta llena de elementos (objetos) que definen calles, puntos de interés, parques, rutas... Son de 4 tipos:

Tipo	Nombre	Descripción	Ejemplos
	Nodo	Punto individual.	Punto de interés, parada de autobús, semáforo...
	Vía (way)	Una línea que une varios nodos.	Calle,camino, muro...
	Área	Una vía cerrada. Con etiquetas (tags) que indican que es un área.	Parque, polígono industrial, bosque...
	Relación	Un conjunto de los anteriores con etiquetas (tags) de relación.	Ruta de autobús (conjunto de calles por las que pasa dicha ruta)

Tabla 3.2: Tipos de elementos OSM.

Como vemos en realidad solo hay nodos y vías, los otros dos tipos son como las anteriores pero con etiquetas concretas.

- Etiquetas (tags)

  Para saber que tipo de elemento es cada cual siempre deben tener asignados etiquetas (tags) que describen sus características. Las etiquetas están formadas por pares {clave=valor} como p.e. {highway=residential}, esta etiqueta aplicada a un elemento vía (way) nos indica que se trata de un elemento perteneciente a la red en enrutado. Dentro de la tipología o familia highway (carreteras) nos encontramos ante una de tipo residential (calle).

  Como vemos la clave describe la tipología del objeto y el valor concreta el objeto. La clave highway no siempre describe objetos de vía (way) como calle o autopista sino que también hay objetos de nodo como {highway=traffic_signal} o {highway=bus_stop}.

Clave	Valor	Descripción
highway	motorway	autopista
	residential	calle urbana
	service	vía de acceso
	track	camino
	pedestrian	calle peatonal
	footway	acera dedicada
	cycleway	vía ciclista
	steps	escalones
	bus_stop	parada de bus
	traffic_signal	semáforo

Tabla 3.3: Ejemplo de etiqueta OSM.

  Tenemos multitud de claves y valores, como vemos están todos en inglés. Para no perdernos existe una lista que recopila las etiquetas más aceptadas por la comunidad OSM, con su correspondencia y traducción en castellano: `http://wiki.openstreetmap.org/wiki/ES:Map_Features`.

  Para conseguir definir con precisión una realidad a base de etiquetas se pueden combinar múltiples pares {clave=valor}, pero con cuidado, no se pueden repetir claves.

Veamos un poco más de cerca como se representan los elementos principales (nodos y vías) junto con las etiquetas que describen su "atributo geográfico".

Un nodo es simplemente un par de coordenadas (latitud, longitud) que definen una localización en el mapa. Los tags asociados permiten saber si se trata de una ciudad, un restaurante, un hospital, etc. Un ejemplo de nodo es el siguiente:

```
<node id="26517895" version="6"  timestamp="2011-10-18T23:50:43Z" uid="399394" user="egrn"
changeset="9595967" lat="43.5558289" lon="-5.921961">
<tag k="source:name" v="Nomenclátor Geográfico de Municipios y Entidades de Población"/>
<tag k="name:ja" v="hfsafhf"/>
<tag k="is_in:province_code" v="33"/>
```

```
<tag k="source:file" v="http://centrodedescargas.cnig.es/CentroDescargas/equipamiento/
BD_Municipios-Entidades.zip"/>
<tag k="is_in:municipality" v="Avilés"/>
<tag k="is_in" v="Asturias;Principado de Asturias;España;Europe"/ >
<tag k="popu1ation:date" v="2009"/ >
<tag k="capital" v="7"/>
<tag k="admin_level" v="7"/>
<tag k="source" v="lnstituto Geográfico Nacional"/>
<tag k="name" v="Avilés"/>
<tag k="source:ele" v="MDT5"/>
<tag k="place" v="town"/>
<tag k="source:date" v="2011-06"/>
<tag k="ref:ine" v="33004010100"/>
<tag k="population" v="79105"/>
<tag k="ele" v="5"/>
</node>
```

Como vemos, el tag "place" identifica el nodo como una ciudad (town), y el tag "name" indica que se trata de Avilés. Por su parte, los tags "is_in", "is_in:province_code" y "is_in:municipality" contienen la clasificación jerárquica de la población dentro de un municipio, provincia, etc. Otros tags, como "population", dan información adicional sobre el nodo.

Una vía es una serie ordenada de nodos que define una calle, camino, carretera o autopista. Por ejemplo:

```
<way id="3996195" version="2" timestamp="2008-09-30T23:10:42Z" uid="41263" user="afernandez"
changeset="722980">
<nd ref="20952923"/>
<nd ref="20952943"/ >
<nd ref="20960696"/>
<nd ref="20952925"/ >
<nd ref="20963055"/>
<nd ref="21067928"/>
<tag k="source:name" v="local knowledge"/>
<tag k="highway'' v="residential"/>
<tag k="name'' v="Calle de Manchester"/>
<tag k="oneway'' v="yes"/>
</way>
```

Los atributos "ref" que aparecen en los elementos <nd>de la vía son referencias a los nodos que la componen. El tag "highway" indica el tipo de vía, en este caso "residential", lo cual indica que es una calle de un núcleo urbano.

Para concluir este apartado, como se ha podido comprobar, la forma en que las características geográficas son representadas por OpenStreetMap permite seleccionar los elementos que componen una determinada capa atendiendo a su tipo y etiquetado, a partir de la base de datos OSM estándar. Es decir, si lo que nos interesa es obtener una capa con las principales vías de transporte para utilizar en el calculo

de rutas, bastara con seleccionar aquellos elementos (vías y relaciones principalmente) que posean como etiqueta asociada "highway" indicando el tipo de vía.

Proceso de importación de datos OSM en PostGIS

En esta sección del documento se recoge el proceso llevado a cabo para la importación de los datos geográficos de OpenStreetMap en una base de datos espacial PostGIS.

Existen varias herramientas que podemos utilizar para la importación de datos OSM a bases de datos espaciales. En la realización de este proyecto fueron tres las herramientas de importación utilizadas:

- osm2pgsql
- osm2pgrouting
- osm2po

Estas herramientas importan datos en formato .osm a bases de datos y están pensadas para organizar los datos siguiendo esquemas diseñados para unos usos muy concretos. Por ejemplo, la herramienta osm2pgsql está diseñada para que genere una bases de datos muy eficiente de cara a la generación de imágenes (renderización), pero no destinada al cálculo de rutas mediante SQL.

En la realización de este proyecto, se cometió el error de comenzar la importación de datos empleando osm2pgsql. Se opto en primer lugar por seleccionar un área pequeña como la provincia de León para probar la funcionalidad del prototipo instalado. Obteniendo un resultado satisfactorio tanto en la importación de los datos como en la generación del correspondiente esquema en la base de datos.

Osm2pgsql se caracteriza por agrupar en una misma tabla las entidades que tienen el mismo tipo de geometría (puntos, líneas o polígonos). Estas tablas son:

- planet_osm_point: Entidades de tipo punto.
- planet_osm_line: Entidades de tipo linea (excluyendo las carreteras).
- planet_osm_polygon: Entidades de tipo polígono.
- planet_osm_roads: Carreteras.

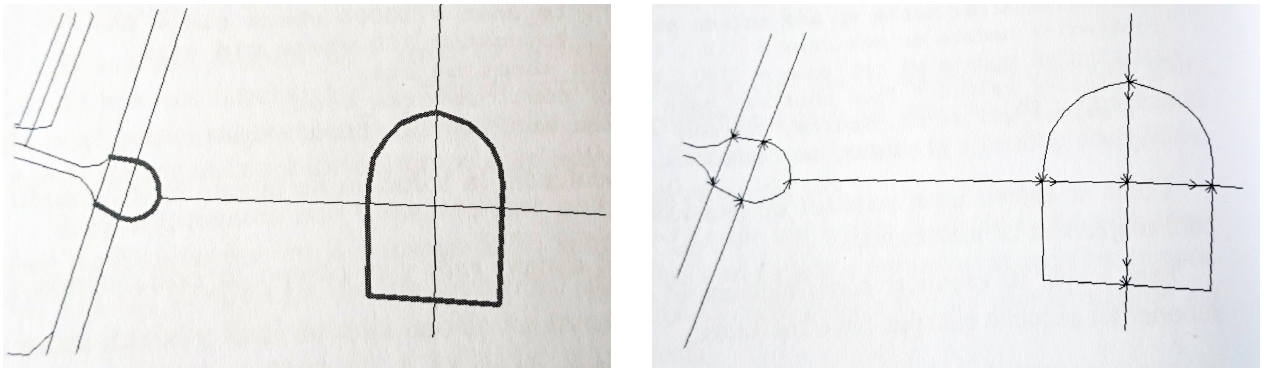
Con osm2pgsql disponemos de un fichero "default.style" que permite personalizar ligeramente el proceso de importación. Este fichero nos permite definir qué datos (elemetos con etiquetas concretas) queremos importar. Pero pese a todo ello, la estructura generada por esta herramienta no cumple el objetivo perseguido: poder utilizar las funcionalidades proporcionadas por pgRouting para el cálculo de rutas. El esquema generado por osm2pgsql no se ajusta a los requisitos para poder aplicar dichas funciones.

Pero fue gracias a cometer este error por el cual se descubrieron las otras dos herramientas anteriormente mencionadas, osm2pgrouting y osm2po. Dichas herramientas están pensadas para generar esquemas especialmente orientados a la aplicación del cálculo de rutas con SQL. En los dos siguientes apartados se describe detalladamente las aportaciones ofrecidas por estas herramientas, su uso y el esquema final generado para ser utilizado en el cálculo de rutas.

Importación mediante osm2pgrouting

Un tramo (way) OSM puede estar formado por más de dos nodos, donde los nodos intermedios pueden formar parte de otros tramos, por lo tanto es necesario partir dichos tramos para que únicamente sus nodos inicial y final sean los que conectan con otros tramos.

La figura siguiente (izquierda) muestra como en la cartografía OSM importada con oms2pgsql los tramos lineales OSM (ways) se cruzan entre sí sin crear nodos en las intersecciones, por el contrario la cartografía de la figura de la derecha muestra como la cartografía OSM importada con osm2pgrouting ha partido dichos tramos lineales creando nuevos registros en la tabla. Los dos tramos lineales (2 filas) de la figura de la izquierda (tramos resaltados) se han transformado en 9 tramos lineales (9 filas) en la figura de la derecha.



(a) Importación con osm2pgsql (b) Importación con osm2pgrouting

Figura 3.2: Nodificación de cartografía OSM.

Es por ello que osm2pgrouting es la herramienta indicada para importar datos OpenStreetMap a una base de datos PostgreSQL/PostGIS cuando su uso esta dirigido al cálculo de rutas con pgRouting.

Una vez instalada la herramienta osm2pgrouting, a través de su código fuente, sólo es necesario disponer de una base de datos espacial para proceder a la carga cartográfica. Osm2pgrouting se ejecuta desde línea de comandos en una terminal del sistema. Las opciones del comando son:

- -file <fichero osm>: Nombre del fichero xml OSM a importar.
- -conf <fichero de configuración>: Ruta al archivo de configuración de osm2pgrouting. Generalmente este archivo es mapconfig.xml y se distribuye al instalar osm2pgrouting.
- -dbname <base de datos>: Nombre de la base de datos en la cual cargar las capas OSM.
- -user <usuario>: Usuario de la base de datos.

Opcionales:

- -host <host>: Host de la máquina donde está el servidor de PostgreSQL. Por defecto es 127.0.0.1.
- -port <puerto>: Número de puerto para la conexión. Por defecto es 5432.
- -passwd <contraseña>: Contraseña de la conexión a la base de datos.
- -clean: Elimina las tablas instaladas en un proceso de osm2pgrouting anterior.

Antes de realizar la importación hay que localizar el fichero de configuración de osm2pgrouting. La distribución de osm2pgrouting incluye un fichero de configuración llamado "mapconfig.xml" ya configurado que conviene localizar en nuestro disco duro, generalmente se encontrará en el directorio de instalación de osm2pgrouting o en Ubuntu en "/usr/share/osm2pgrouting/mapconfig.xml".

Osm2pgrouting sólo importará aquellos objetos ways (vías) OSM cuyas etiquetas aparezcan definidas en mapconfig.xml, por defecto este fichero importa únicamente aquella información OSM relativa a los viales. En la wiki de OSM aparecen varios artículos con información sobre las etiquetas utilizables en aplicaciones de routing.

Un ejemplo de sentencia de importación con la herramienta osm2pgrouting para la cartografía de la provincia de León a la base de datos espacial "leon" sería:

```
postgres@gis:~$ /usr/share/bin/osm2pgrouting -file /home/postgres/leon.osm -conf
/usr/share/osm2pgrouting/mapconfig.xml -dbname leon -user postgres -clean
```

Tras la ejecución se nos informaría por consola sobre el éxito de la importación, así como los ways cargados y el número de tramos en los que se han dividido.

```
Create Topology success
######################
size of streets: 3474
size of splilled ways: 8432
finished
postgres@gis:~$
```

Tras la importación aparecerán cuatro nuevas tablas en la base de datos espacial utilizada. Las tablas "classes" y "types" almacenan información sobre las etiquetas OSM. La tabla "nodes" contiene las coordenadas latitud y longitud de los nodos (WGS 84), y la tabla "ways" contiene los nuevos tramos lineales tras el proceso de partición de los originales OSM.

Hay que tener en cuenta que osm2pgrouting elimina la información OSM necesaria para relacionar las vías con los nodos que las forman o el campo "osm_id" identificativo de OSM. Es decir, no se podrá relacionar la información importada con los datos OSM iniciales. Cuando se quiera actualizar la red mediante datos OSM habrá que eliminar todos los datos actuales y realizar una nueva importación.

Por el contrario, aporta importantes ventajas:

- Crea nodos en las intersecciones de la red OSM calculando y partiendo los tramos lineales adecuadamente.
- Importa únicamente (fichero mapconfig.xml) la cartografía OSM necesaria aplicando filtros de etiquetas.
- Añade a la tabla los campos adecuados para utilizar las funciones de pgrouting, como x1, y1, x2, y2, length (cost), reverse_cost, rule, to_cost, source y target. Aunque algunos de ellos se deben rellenar tras la importación por el usuario.

- Calcula en función de la etiqueta "highways->oneway" de OSM si los tramos son de dirección única, y en tal caso establece un "reverse_cost" apropiado.

Los resultados obtenidos para la ciudad de León fueron satisfactorios, tanto en la importación cartográfica como en el esquema generado para su utilización con pgRouting. Lo mismo ocurrió cuando se probo a importar la cartografía de toda España. Pero el problema llego con la cartografía europea.

Importación de datos Europe OSM

Hay algunas limitaciones en la importación de datos con osm2pgrouting, sobre todo en relación con el tamaño de la red. La versión actual de osm2pgrouting carga todos los datos en la memoria, lo que hace que sea rápido, pero también que requiera una gran cantidad de memoria suponiendo un gran problema para grandes conjuntos de datos. Una herramienta alternativa a osm2pgrouting sin la limitación de tamaño de la red es osm2po (`http://osm2po.de`), disponible en "Licencia Freeware".

Este apartado recoge los pasos que fueron necesarios para utilizar osm2po. Dicha herramienta es a la vez, un convertidor y un motor de enrutamiento. Analiza los datos de OpenStreetMap y hace que sea enrutable. Mientras osm2pgrouting parece estar limitado por la cantidad de memoria disponible en el sistema, osm2po es capaz de convertir grandes conjuntos como europe.osm.

Su instalación es simple, se basa en descargar la última versión estable de osm2po desde la página oficial y descomprimir el archivo, generando así un directorio donde encontraremos el archivo jar junto con la documentación de la herramienta (sólo disponible en alemán). osm2po requiere tener instalado Java^{TM} 6 (o superior).

Gracias a osm2po importamos la cartografía OSM de Europa, obteniendo esta a través del servicio que ofrece Geofabrik: `http://download.geofabrik.de`. Se dispone de tres formatos diferentes: .osm.pbf, .shp.zip y .osm.bz2. La herramienta osm2po sólo importa datos en formato .osm por lo que el formato .shp.zip queda descartado. En cuanto a los otros dos archivos .osm comprimidos, osm2po permite importar directamente sin necesidad de descomprimirlos. Este aspecto es interesante dado que nos ahorra bastante espacio de almacenamiento, dado que una vez descargado ronda los 15,6Gb en .osm.pbf y los 23,7Gb en .osm.bz2, rondando los 271Gb descomprimido.

Ahora veamos su funcionamiento. El único requisito es que osm2po este desempaquetado en cualquier directorio y que Java se pueda llamar desde línea de comandos. Situados en el directorio donde se encuentra desempaquetado, haremos uso del archivo osm2po-xxx.jar, la sintaxis es la siguiente:

```
java [-Xmx] -jar osm2po-xxx.jar [param=value]* [inputfile]
```

Donde:

- -Xmx: Parámetro Java importante para la reserva de memoria. Por ejemplo: -Xmx2g (2Gb).
- param=value: Parámetros osm2po. No deben contener ningún espacio.
- inputfile: Ruta archivo de entrada.

En cuanto a los parámetros osm2po, existen dos que requieren cierta atención:

- tileSize: El parámetro más importante para la conversión. Controla el equilibrio entre la memoria disponible y el tamaño de los datos. tileSize acepta dos sub-valores, siendo el patrón:[<Y>x<X>|x] [,<buffer>]. Por defecto es tileSize=1x1,0.1 (en grados) donde Y, X son números enteros y buffer un decimal. 1x1,0.1 es un valor pesimista y debe aumentarse para los países más grandes. Ya que si una vía (way) es demasiado larga y el buffer demasiado pequeño, no se puede resolver y será omitida en la importación. Es un parámetro difícil de ajustar, pero consultando los ejemplos de la página oficial de osm2po podemos determinarlo.

- prefix: Otro parámetro importante. Se debe ajustar con el fin de anteponer dicho prefijo a los nombres generados para archivos o tablas. Si no se especifica, el valor predeterminado es prefix=osm. Debe se breve y razonable, por ejemplo "eu" para Europa.

Ajustando la sintaxis a nuestras necesidades, la sentencia resultante para cargar la cartografía europea sería semejante a:

```
postgres@gis:~/osm2po-4.8.8$ java -Xmx29g -jar osm2po-xxx.jar prefix=eu tileSize=50x50,1,c
''/home/postgres/europe-latest.osm''
```

Para realizar esta importación fue necesario ampliar la memoria principal con que se contaba inicialmente en la máquina virtual de instalación. En un principio dicha máquina contaba con 6Gb de RAM, suficientes si se ajustan correctamente los parámetros osm2po, pero suponiendo un coste de tiempo de procesamiento de casi un día. Por lo que fue necesario asignar una memoria de 32Gb para agilizar todo este proceso y descartar fallos en la importación.

Ejecutada la sentencia, el proceso realizado por osm2po siempre es el mismo:

1. Convertir los datos para pgRouting
2. Generar un script de inserción SQL
3. Inicio del servicio de oms2po (routing)

Si todo ha ido bien, al final del procesamiento aparece el siguiente mensaje:

```
Waiting for requests at http://localhost:8888/Osm2poService
```

Esto nos indica que el proceso se completo con éxito y que osm2po inicio su propio servicio web de enrutamiento. Podemos cerrar dicho servicio mediante Ctrl+C. En el directorio de osm2po, podemos comprobar que se ha creado una carpeta cuyo nombre corresponde con el valor otorgado al parámetro "prefix", en el ejemplo, "eu". Los siguientes archivos de texto vale la pena echarles un vistazo rápido:

- osm2po/osm2po.config: Archivo de configuración para osm2po.
- osm2po/eu/eu_2po.log: Archivo de registro que fue escrito durante la conversión.
- osm2po/eu/eu_2po_4pgr.sql: Script SQL que crea una tabla e insertar los datos convertidos para PostGIS.

Generado el script de inserción (eu_2po_4pgr.sql), creamos en PostgreSQL una nueva base de datos espacial para la carga de datos con osm2po. Crearemos una nueva base de datos en PostgreSQL llamada "europe". Tras crear en PostgreSQL una nueva base de datos espacial, es el momento de utilizar el script generado por osm2po para poblarla. Si hemos estado siguiendo los mensajes de la consola durante el procesamiento de osm2po, nos daremos cuenta de que se nos proporciona una plantilla para realizar dicha tarea a través de psql. La plantilla mostrada es:

```
psql -U [username] -d [dbname] -q -f "/path/to/my/dir/eu/eu_2po_4pgr.sql"
```

Sólo tenemos que reemplazar [username] y [dbname] con los nombres que usamos. En nuestro caso, la base de datos "europe" y el usuario "postgres". Siendo el comando que debemos utilizar para importar los datos correspondientes en la base de datos:

```
postgres@gis:~$ psql -U postgres -d europe -q -f "/home/postgres/osm2po/eu/eu_2po_4pgr.sql"
```

Tras la ejecución, contamos con la tabla "eu_2po_4pgr" creada y poblada en nuestra base de datos "europe", la cual sera utilizada para realizar las consultas de enrutamiento por Europa con pgRouting.

3.5. Rutas mediante pgRouting

El siguiente apartado se va a centrar en resolver el problema del camino más corto utilizando las funciones que pgRouting aporta para tal fin, aplicadas a la cartografía de OSM.

3.5.1. Camino más corto

El camino más corto consiste en el cálculo de la ruta entre dos puntos (nodos) de forma que la suma de los costes de los tramos constituyentes es minimizada. El coste de un tramo puede consistir en su longitud, el tiempo en atravesarlo o cualquier otra variable.

pgRouting implementa tres algoritmos diferentes para resolver el problema del camino más corto entre dos puntos o ejes de una red lineal.

1. Algoritmo de Dijkstra. Este algoritmo devuelve el camino más corto (de todos los posibles) entre dos nodos de una red. Para ello, no se necesita conocer las coordenadas de los puntos de los nodos de la red, sino simplemente disponer de la numeración de los puntos inicial y final de cada tramo así como su longitud. pgRouting ofrece además la versión bidireccional para este algoritmo.

2. Algoritmo Astar (también conocido como "A*"). Además de los números de los nodos inicial y final de cada tramo y su longitud, utiliza para los cálculos las coordenadas X e Y de dichos nodos. Mediante un algoritmo heurístico es capaz de encontrar un camino que tiene una alta probabilidad de ser uno de los más cortos entre dos nodos de la red. Al igual que el algoritmo de Dijkstra, cuenta con su versión bidireccional.

3. Algoritmo Shooting Star. Es un algoritmo heurístico que necesita también las coordenadas de los nodos como el Astar, pero añade la posibilidad de añadir restricciones a la red, por ejemplo, giros no permitidos.

3.5.2. Topología de red

pgRouting necesita que la capa lineal utilizada en todos sus algoritmos tenga topología de red. Esto implica las siguientes condiciones:

- Los tramos de la red deben tener sus puntos inicial y final numerados de forma que se asegure que dos tramos están conectados si su número de nodo inicial o final es el mismo.

- Si hay dos tramos que se cruzan, el punto de intersección no será considerado en el cálculo de la ruta. Por lo tanto, es necesario que si en la realidad el cruce representa una intersección donde la ruta puede bifurcarse éste se debe modelar con un nodo y partir los dos tramos que se cruzan en cuatro tramos con la inserción/actualización de nuevas geometrías en la tabla.

Para cumplir las dos condiciones requeridas contamos con las herramientas osm2pgrouting y osm2po. Como ya se comento en la sección anterior, osm2pgrouting y osm2po son unas herramientas de línea de comandos que permiten importar los datos de OpenStreetMap en una base de datos espacial para su uso con pgRouting. Se construye la topología de red de enrutamiento de forma automática y se crea tablas de tipos de entidad y clases de tráfico.

3.5.3. Problema del camino más corto

A continuación se definen las funciones que pgRouting proporciona para resolver el problema del camino más corto y que han sido utilizadas en el desarrollo de este trabajo. Y en el posterior apartado podrá ver un ejemplo simple de la utilización básica de una de estas funciones.

Dijkstra y B.Dijkstra

La función pgr_dijkstra devuelve el camino más corto usando el algoritmo de Dijkstra, mientras que la función pgr_bdDijkstra devuelve el recorrido más corto "bidireccional" usando el algoritmo de Dijkstra bidireccional.

- Sinopsis

El algoritmo de Dijkstra, fue concebido por el científico computacional holandés, Edsger Dijkstra en 1956. Se trata de un algoritmo de búsqueda gráfica que resuelve el problema del camino más corto de una sola fuente con costos no negativos, generando un árbol de ruta más corta. Por su parte, el algoritmo de búsqueda bidireccional de Dijkstra realiza una búsqueda desde la fuente hacia el destino y, al mismo tiempo, desde el destino hacia el origen, terminando donde estas búsquedas se reúnen en el centro.

Ambos algoritmos devuelven un conjunto de registros pgr_costResult (seq, id1, id2, cost) que conforman un camino. De hecho el formato de dichas funciones es el mismo, solo es necesario cambiar su nombre a la hora de referenciar una u otra.

```
pgr_costResult[] pgr_dijkstra(text sql, integer source, integer target, boolean directed,
boolean has_rcost);
```

```
pgr_costResult[] pgr_bdDijkstra(text sql, integer source, integer target, boolean directed,
boolean has_rcost);
```

- Descripción

Los parámetros a proporcionar en el uso de ambas funciones son:

sql Consulta SQL, que debe proporcionar un conjunto de registros con los siguientes campos:

```
SELECT id, source, target, cost [,reverse_cost] FROM edge_table
```

- id int4 Identificador del borde
- source int4 Identificador del vértice inicial del borde
- target int4 Identificador del vértice final del borde
- cost float8 Valor del costo del recorrido sobre el borde
- reverse_cost float8 (opcional) El costo para el recorrido inverso del borde. Esto sólo se utiliza cuando los parámetros directed y has_rcost son True.

source int4 Identificador del punto de partida

target int4 Identificador del punto de llegada

directed true Si la gráfica es direccional

has_rcost Si es True, el campo reverse_cost del conjunto de registros generados se utiliza para calcular el costo de la travesía del borde en la dirección opuesta.

Devuelven un conjunto del tipo de datos pgr_costResult[]:

seq Secuencia de registros

id1 Identificador del nodo visitado

id2 Identificador del borde (-1 para el último)

cost Costo del recorrido desde el nodo id1 usando el borde id2 hasta el otro extremo del borde.

A* y B.A*

La función pgr_astar retorna el camino más corto usando el algoritmo A*, mientras que la función pgr_bdAstar retorna el camino más corto usando el algoritmo A* bidireccional.

- Sinopsis

El algoritmo A* ("A Star") se basa en el algoritmo de Dijkstra con una heurística que evalúa sólo un subconjunto de la gráfica general, permiténdole resolver la mayoría de los problemas del camino más corto. Por su parte, el algoritmo de búsqueda bidireccional A* busca desde la fuente hasta el destino y al mismo tiempo desde el destino al origen y termina cuando ambas búsquedas se reúnen en el centro.

Ambos algoritmos devuelven un conjunto de registros pgr_costResult (seq, id1, id2, cost) que conforman un camino. De hecho el formato de dichas funciones es el mismo, solo es necesario cambiar su nombre a la hora de referenciar una u otra.

```
pgr_costResult[] pgr_astar(text sql, integer source, integer target, boolean directed,
boolean has_rcost);

pgr_costResult[] pgr_bdAstar(text sql, integer source, integer target, boolean directed,
boolean has_rcost);
```

- Descripción

Los parámetros a proporcionar en el uso de ambas funciones son:

sql Consulta SQL, que debe proporcionar un conjunto de registros con los siguientes campos:

```
SELECT id, source, target, cost [,reverse_cost] FROM edge_table
```

- id int4 Identificador del borde
- source int4 Identificador del vértice inicial del borde
- target int4 Identificador del vértice final del borde
- cost float8 Valor del costo del recorrido sobre el borde
- x1 float8 Coordenada x del punto inicial del borde
- y1 float8 Coordenada y del punto inicial del borde
- x2 float8 Coordenada x del punto final del borde
- y2 float8 Coordenada y del punto final del borde
- reverse_cost float8 (opcional) El costo para el recorrido inverso del borde. Esto sólo se utiliza cuando los parámetros directed y has_rcost son True.

source int4 Identificador del punto de partida

target int4 Identificador del punto de llegada

directed true Si la gráfica es direccional

has_rcost Si es True, el campo reverse_cost del conjunto de registros generados se utiliza para calcular el costo de la travesía del borde en la dirección opuesta.

Devuelven un conjunto del tipo de datos pgr_costResult[]:

seq Secuencia de registros

id1 Identificador del nodo visitado

id2 Identificador del borde (-1 para el último)

cost Costo del recorrido desde el nodo id1 usando el borde id2 hasta el otro extremo del borde.

3.5.4. Ejemplo básico de uso

Como ejemplo se va a calcular el camino más corto desde un nodo X a un nodo Y utilizando el algoritmo de Dijkstra que nos asegura la mejor ruta posible. La función a utilizar se llama pgr_dijkstra, comentada anteriormente. Vamos a ver cómo utilizarla.

Disponemos de la base de datos espacial "spain" en la que se importo la cartografía de España mediante la herramienta osm2po. Gracias a esta aplicación, en dicha base de datos ya hay una tabla llamada "sp_2po_4pgr" preparada para ejecutar consultas de routing.

Calculemos la ruta más corta entre dos puntos dentro de una misma ciudad, seleccionamos como origen "Calle Prado de la Lana" y como destino "Calle Pintor Oliva" de la ciudad de Palencia. Podemos consultar los datos recogidos en la tabla "sp_2po_4pgr" para ambas localizaciones:

```
spain=# select id, osm_id, osm_name, source, target from sp_2po_4pgr where osm_id=X;
```

id	osm_id	osm_name	source	target
1975135	192505339	Calle Prado de la Lana	1434779	1434780
775586	72795578	Calle Pintor Oliva	544132	544133

Tabla 3.4: Algunos de los datos recogidos en sp_2po_4pgr.

La sentencia para el uso de la función pgr_dijkstra resultante es:

```
spain=# select seq, id1 AS node, id2 AS edge, cost from pgr_dijkstra('select id, source,
target, cost from sp_2po_4pgr', 1434779, 544132, false, false);
```

Siendo el resultado obtenido:

seq	node	edge	cost
0	1434779	775637	0.0062252698
1	544176	775638	0.0017699131
2	544158	775639	0.0017806096
3	544196	775640	0.0038258135
4	544169	1637284	0.0016537159
5	544210	1637295	0.00024356964
6	544205	1637281	0.0031830075
7	273368	1637266	0.001708908
8	544132	-1	0

Tabla 3.5: Resultado de la función pgr_dijkstra.

Siguiendo la tabla de nodos, podemos ver la secuencia calculada. La columna "node" y "edge" se corresponden con los campos "source" e "id" de una vía, respectivamente. La columna de coste nos dice el coste de cada salto.

seq	node	edge	Vía
0	1434779	775637	Calle del Labrador
1	544176	775638	Calle del Labrador
2	544158	775639	Calle del Labrador
3	544196	775640	Calle del Labrador
4	544169	1637284	Avenida de Valladolid
5	544210	1637295	Avenida de Valladolid
6	544205	1637281	Avenida de Valladolid
7	273368	1637266	Avenida Cardenal Cisneros
8	544132	-1	Calle Pintor Oliva

Tabla 3.6: Vías de la ruta de la función pgr_dijkstra.

Que se comprobó, coincide con la ruta que nos ofrecería Google Maps para el mismo caso:

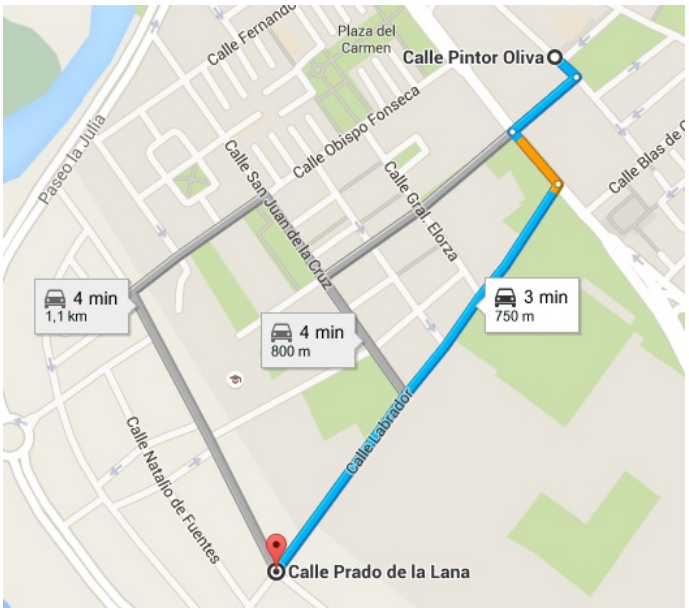


Figura 3.3: Ruta en Google Maps.

Capítulo 4

Evaluación inicial del rendimiento

Como se ha pretendido reflejar hasta el momento, una vez instalada la plataforma se procedió a cargar cartografía OSM comenzando a nivel de ciudad, después de país y finalmente del continente europeo. Tras lograr cada una de estas cargas cartográficas se realizaron una serie de pruebas, con el objetivo de comprobar que el cálculo de rutas óptimas mediante el modulo pgRouting funcionaba.

Así fue como se detecto que el esquema generado por la herramienta osm2pgsql no estaba destinado para tal fin, mientras que si era el caso de las herramientas de importación osm2pgrouting y osm2po, como ya se ha reflejado. Tanto el esquema generado por la herramienta osm2pgrouting como el de la herramienta osm2po, pueden ser utilizados para el cálculo de rutas con pgRouting. La diferencia radica en la limitación de la carga cartográfica, siendo posible cargar cartografía pesada como es el continente europeo únicamente por osm2po.

En este capítulo se recoge el plan de pruebas y evaluación desarrollado a lo largo de este proyecto. Se recogen las pruebas realizadas tras la carga con la herramienta osm2po, a nivel de país (España) y continente (Europa). Los resultados son similares para osm2pgrouting, pero limitándose a las pruebas realizadas sobre la cartografía de España, por lo que se descarta su inclusión.

4.1. Batería de pruebas y resultados para la cartografía española

En las siguientes ilustraciones y tablas se ofrece una visión de las rutas calculadas con pgRouting a nivel de país. Se recogen aquellas pruebas cuya distancia dentro del propio país resultaba relevante para el fin de este proyecto. Omitiéndose las pruebas referentes a un mismo nodo, nodos adyacentes o nodos de una misma ciudad, dado que su obtención no implico ningún problema.

También se muestra la sintaxis de la consulta básica utilizada para llevar a cabo estas pruebas y una tabla donde se recogen los tiempos obtenidos para las mismas.

4.1.1. Sintaxis de la consulta

```
select seq, id1 as node, id2 as edge, cost from pgr_astar('select id,
source, target, cost, x1, y1, x2, y2 from sp_2po_4pgr', source, target,
false, false);
```

Como se puede observar, la sintaxis utilizada para obtener estas rutas se corresponde exactamente con la ofrecida por pgRouting para tal fin. En función del algoritmo que se desee para calcular la ruta, se deberá modificar la función resaltada en color de acuerdo con la especificación para dicho algoritmo que proporcione pgRouting y que se indico en la correspondiente sección del capitulo anterior.

Tener en cuenta que los campos id, source y target no mantienen sus valores en posteriores importaciones de la cartografía, ya que son campos que generan automáticamente las propias herramientas de importación de datos OSM.

4.1.2. Casos de prueba

España	**Vía**		**Nodo**			
Provincia	osm_name	osm_id	osm_source_id	osm_target_id	source	target
Carballo (A Coruña)	Calle Estrella	39248986	470267639	693166925	280758	451609
Cartagena (Murcia)	Calle Ramón y Cajal	23171548	390992085	386828426	51255	1830820

Tabla 4.1: Prueba 1 - Spain.

Figura 4.1: Prueba 1 - Spain.

España	**Vía**		**Nodo**			
Provincia	osm_name	osm_id	osm_source_id	osm_target_id	source	target
Irún (Guipúzcoa)	Colon pasealekua	39980180	458451172	458445546	276985	276972
Algeciras (Cádiz)	La Rocha	268295715	2445784167	2445784138	1830937	1830938

Tabla 4.2: Prueba 2 - Spain.

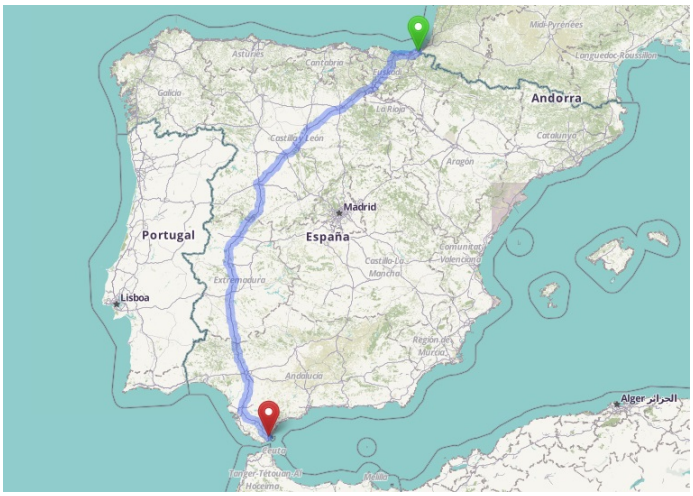


Figura 4.2: Prueba 2 - Spain.

España	**Vía**		**Nodo**			
Provincia	osm_name	osm_id	osm_source_id	osm_target_id	source	target
Tudela (Navarra)	Calle de Manuel Moneo	296735207	3005439503	3005439502	1891990	1891991
Cáceres (Extremadura)	Calle Piedad	35438368	357338013	415695863	245117	366277

Tabla 4.3: Prueba 3 - Spain.

Figura 4.3: Prueba 3 - Spain.

España	**Vía**		**Nodo**			
Provincia	osm_name	osm_id	osm_source_id	osm_target_id	source	target
Barcelona	Vía Augusta	47373955	228725691	229063644	37132	365165
Almería	Calle Altamira	204539433	298004170	298003927	194441	1515559

Tabla 4.4: Prueba 4 - Spain.

Figura 4.4: Prueba 4 - Spain.

España	Vía		Nodo			
Provincia	osm_name	osm_id	osm_source_id	osm_target_id	source	target
Málaga	Calle Hilera	196873633	2071665284	1143527901	1466261	754840
Tomelloso (Ciudad Real)	Plaza Malaga	207438958	2176933544	2176933543	1534788	1534813

Tabla 4.5: Prueba 5 - Spain.

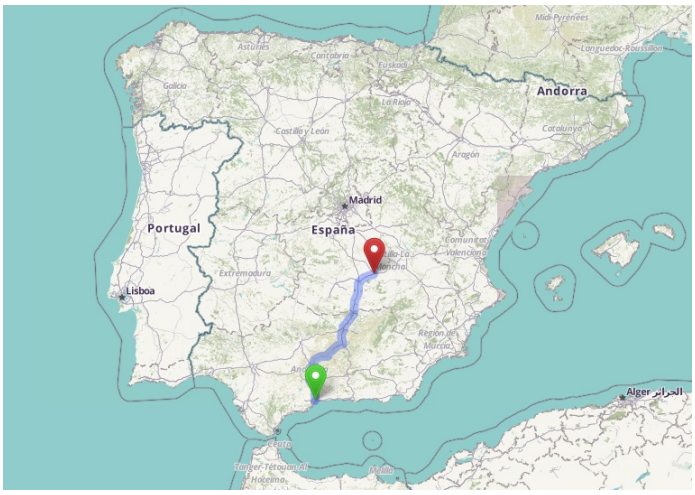


Figura 4.5: Prueba 5 - Spain.

4.1.3. Resultados obtenidos

La siguiente tabla recoge el tiempo de ejecución de cada una de las pruebas presentadas en el apartado anterior. Aparecen las cinco pruebas junto con sus resultados para las cuatro funciones pgRouting con las que se trabaja. Para cada una de estas funciones se midió el tiempo de cinco ejecuciones y se obtuvo su media, todo ello en la unidad de milisegundos. Dicha media se convierte a segundos decimales para facilitar su observación y evaluación.

Se recomienda analizar estos resultados comparando inicialmente cada algoritmo con su versión bidireccional. Tras lo cual se proceda a evaluar las versiones de Dijkstra contra su correspondiente versión A*.

Prueba \ Algoritmo		*Dijkstra*	*B.Dijkstra*	*A**	*B.A**
1	1ª	10664,945	12179,843	11190,625	39809,277
	2ª	9891,725	11580,647	10092,717	39485,773
	3ª	9651,078	12048,067	11023,786	38753,654
	4ª	10772,547	10582,141	10888,379	35750,319
	5ª	8918,525	11989,769	10923,529	39053,396
	Media:	9979,764	11676,093	10823,807	38570,484
	Segundos:	9,98	11,68	10,82	38,57
2	1ª	10860,977	12894,177	11217,172	13838,341
	2ª	9181,525	11664,761	10550,383	10031,191
	3ª	8405,579	11576,420	9905,810	13653,921
	4ª	10277,819	10794,883	9747,957	12684,152
	5ª	10233,149	11606,574	11380,575	13949,457
	Media:	9791,810	11707,363	10560,379	12831,412
	Segundos:	9,79	11,71	10,56	12,83
3	1ª	10333,757	12715,434	9671,586	9843,352
	2ª	8782,199	12003,287	8238,402	8324,903
	3ª	10302,333	11994,917	9123,699	9419,596
	4ª	9556,381	11863,140	9138,458	10376,822
	5ª	9883,125	12301,548	9228,766	8618,832
	Media:	9771,559	12175,665	9080,182	9316,701
	Segundos:	9,77	12,18	9,08	9,32
4	1ª	12180,621	12434,038	10644,256	20503,698
	2ª	9875,577	9527,342	10307,179	17253,204
	3ª	10492,379	11442,332	10403,896	19345,982
	4ª	9856,675	12176,534	10316,597	20062,894
	5ª	9460,166	12138,309	10338,343	18134,428
	Media:	10373,084	11543,711	10402,054	19060,041
	Segundos:	10,37	11,54	10,40	19,06
5	1ª	10581,737	12237,109	9470,910	13310,412
	2ª	10279,629	12223,413	9366,782	13183,359
	3ª	10395,018	11952,470	8961,985	13167,241
	4ª	10149,958	12003,853	8519,362	13236,482
	5ª	10385,328	12079,758	9474,044	12308,201
	Media:	10358,334	12099,321	9158,617	13041,139
	Segundos:	10,36	12,10	9,16	13,04

Tabla 4.6: Resultados de las pruebas para la cartografía española.

Como se puede observar, el tiempo de ejecución obtenido para la función de Dijkstra y A* en todas las pruebas, presentan mejores resultados que sus versiones bidireccionales. Aspecto que nos extraño, dado que teóricamente debería ser al contrario. Aunque suponemos que se deba a algún aspecto de como implementa estas funciones bidireccionales pgRouting.

Por otro lado, hay un resultado que se aleja notablemente del resto de los valores, el tiempo de ejecución de la función A* bidireccional para la primera prueba. Sobre esta prueba y con este algoritmo, ante dicho resultado, se tomaron nuevas medidas en posteriores días y momentos, pero el resultado se mantuvo en valores semejantes.

En cuanto a la comparativa del resultado de Dijkstra contra el de A*, la diferencia de sus tiempos

para los casos de prueba no permiten determinar que uno sea mejor que otro, dado que se mantienen en un rango de valores semejantes variando en cada prueba cual emite mejor respuesta. En sus versiones bidireccionales, si podríamos aventurar que B.Dijkstra ofrece mejores resultados que B.A*.

4.2. Batería de pruebas y resultados para la cartografía europea

En las siguientes ilustraciones y tablas se ofrece una visión de las rutas calculadas con pgRouting a nivel de continente. Se selecciono una serie de pruebas que surgieron de modo aleatorio, pero que debían ajustarse al siguiente patrón (resultando un total de 15 pruebas):

- Pruebas 01, 02 y 03: Usando como "source" y "target" el mismo nodo: tres pruebas en tres países diferentes.
- Pruebas 11, 12 y 13: Usando como "source" y "target" dos nodos consecutivos de una misma vía: tres pruebas en tres países diferentes.
- Pruebas 21, 22 y 23: Usando como "source" y "target" dos nodos de dos vías dentro de una misma ciudad: tres pruebas en tres países diferentes.
- Pruebas 31, 32 y 33: Usando como "source" y "target" dos nodos de dos vías de diferentes ciudades dentro del mismo país: tres pruebas en tres países diferentes.
- Pruebas 41, 42 y 43: Usando como "source" y "target" dos nodos de dos vías de diferentes ciudades de diferente país: tres pruebas en tres países diferentes.

También se muestra la sintaxis de la consulta básica utilizada para llevar a cabo estas pruebas y una tabla donde se recogen los tiempos obtenidos para las mismas.

4.2.1. Sintaxis de la consulta

```
select seq, id1 as node, id2 as edge, cost from pgr_astar('select id,
source, target, cost, x1, y1, x2, y2 from eu_2po_4pgr', source, target,
false, false);
```

Como se puede observar, la sintaxis utilizada para obtener estas rutas se corresponde exactamente con la ofrecida por pgRouting para tal fin. En función del algoritmo que se desee para calcular la ruta, se deberá modificar la función resaltada en color de acuerdo con la especificación para dicho algoritmo que proporcione pgRouting y que se indico en la correspondiente sección del capitulo anterior.

Tener en cuenta que los campos id, source y target no mantienen sus valores en posteriores importaciones de la cartografía, ya que son campos que generan automáticamente las propias herramientas de importación de datos OSM.

4.2.2. Casos de prueba

id	osm_id	osm_source_id	osm_target_id	source	target
10197465	49252333	977015471	995270013	8324560	17383662
España, Palencia, vía: 'Calle Don Pelayo'					

Tabla 4.7: Prueba 01 - Europe.

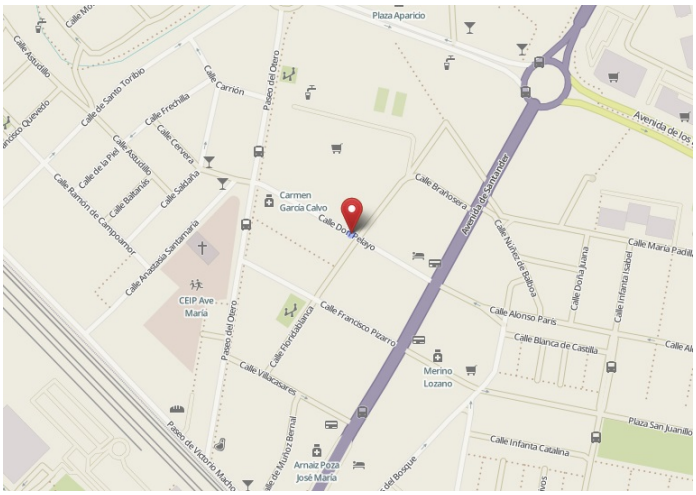


Figura 4.6: Prueba 01 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
20635178	161973090	358950598	1739249272	5514019	16563132
Francia, Condom, vía: 'Rue Dutoya'					

Tabla 4.8: Prueba 02 - Europe.

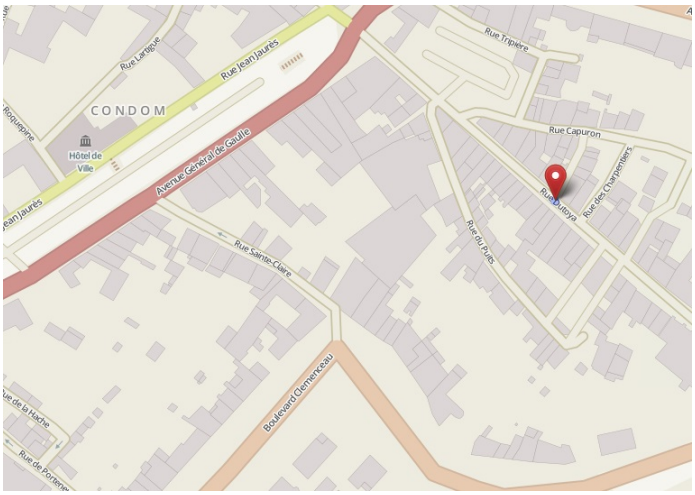


Figura 4.7: Prueba 02 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
31797540	311539528	534402312	274703485	7591325	3158834
Italia, Prato, 'Via Galcianese'					

Tabla 4.9: Prueba 03 - Europe.

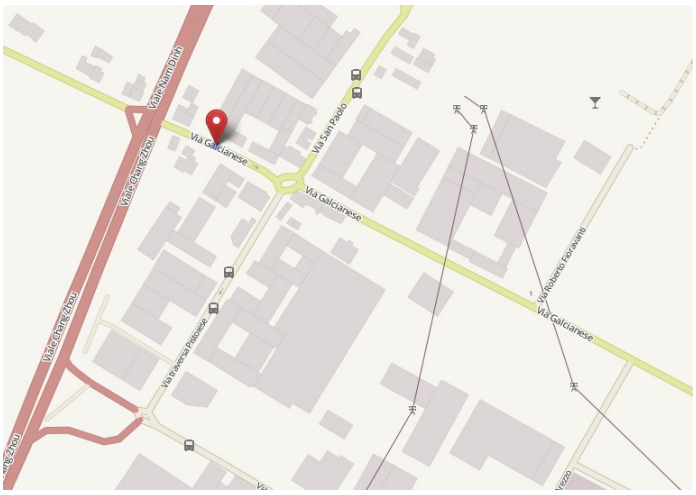


Figura 4.8: Prueba 03 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
10197465	49252333	977015471	995270013	8324560	17383662
10197466	49252333	995270013	625500059	17383662	8324561
España, Palencia, vía: 'Calle Don Pelayo'					

Tabla 4.10: Prueba 11 - Europe.

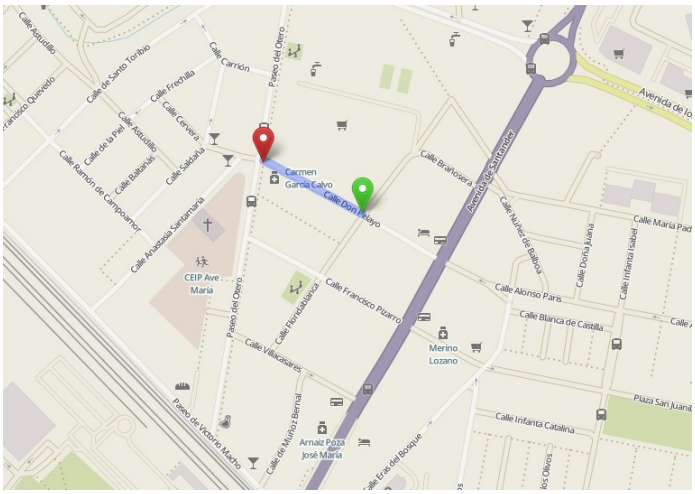


Figura 4.9: Prueba 11 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
20635178	161973090	358950598	1739249272	5514019	16563132
20635179	161973090	1739249272	1739249259	16563132	24869728
Francia, Condom, vía: 'Rue Dutoya'					

Tabla 4.11: Prueba 12 - Europe.

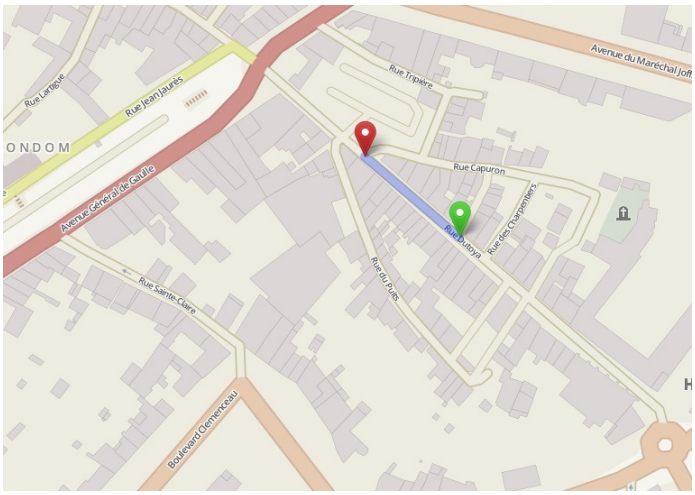


Figura 4.10: Prueba 12 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
31797540	311539528	534402312	274703485	7591325	3158834
31797541	311539528	274703485	1861664778	3158834	17661478
Italia, Prato, 'Via Galcianese'					

Tabla 4.12: Prueba 13 - Europe.

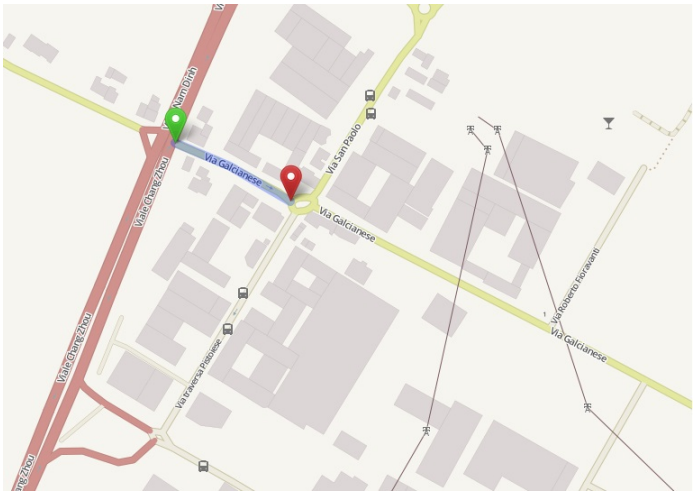


Figura 4.11: Prueba 13 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
10197465	49252333	977015471	995270013	8324560	17383662
España, Palencia, vía: 'Calle Don Pelayo'					
12379884	73285768	504738866	1641089517	10073950	15765299
España, Palencia, vía: 'Avenida de Madrid'					

Tabla 4.13: Prueba 21 - Europe.

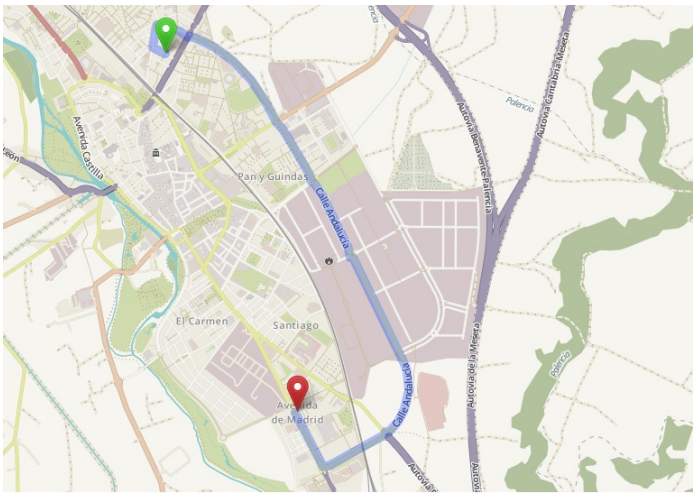


Figura 4.12: Prueba 21 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
20635178	161973090	358950598	1739249272	5514019	16563132
Francia, Condom, vía: 'Rue Dutoya'					
6825906	32610563	358950381	366707574	5664587	5664588
Francia, Condom, vía: 'Rue Albert Camus'					

Tabla 4.14: Prueba 22 - Europe.

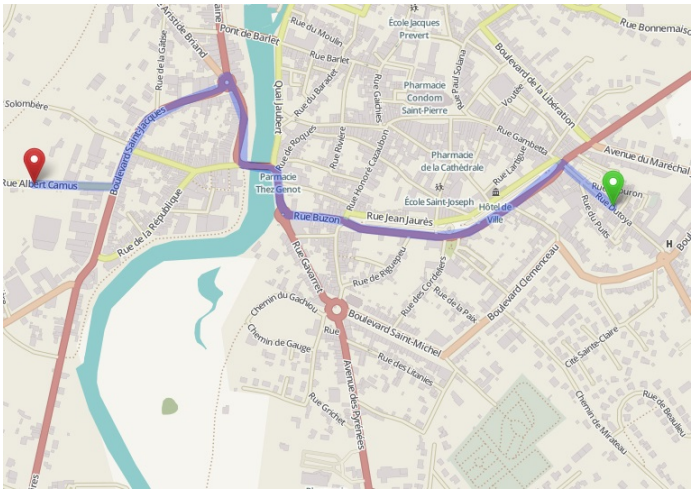


Figura 4.13: Prueba 22 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
31797540	311539528	534402312	274703485	7591325	3158834
Italia, Prato, 'Via Galcianese'					
20410758	159688972	429705930	484760610	7183195	16393296
Italia, Prato, 'Via Antonio Stradivari'					

Tabla 4.15: Prueba 23 - Europe.

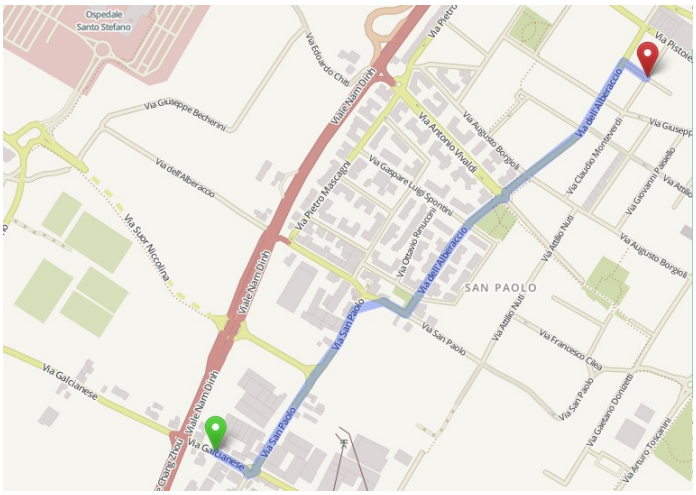


Figura 4.14: Prueba 23 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
10197465	49252333	977015471	995270013	8324560	17383662
España, Palencia, vía: 'Calle Don Pelayo'					
22677085	181616261	1427444914	1427444760	14246280	18134845
España, Granada, Peligros, vía: 'Calle Cuba'					

Tabla 4.16: Prueba 31 - Europe.

Figura 4.15: Prueba 31 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
20635178	161973090	358950598	1739249272	5514019	16563132
Francia, Condom, vía: 'Rue Dutoya'					
5882316	29459645	355221271	461268283	4892891	4892892
Francia, Poitiers, vía: 'Rue Girouard'					

Tabla 4.17: Prueba 32 - Europe.

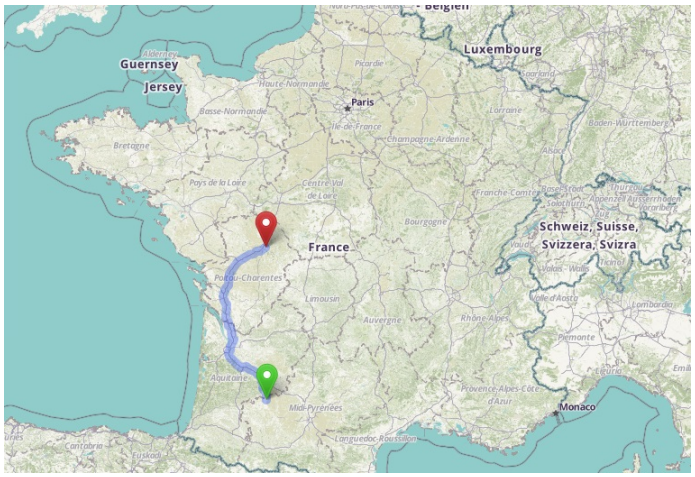


Figura 4.16: Prueba 32 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
31797540	311539528	534402312	274703485	7591325	3158834
Italia, Prato, 'Via Galcianese'					
11933915	67519115	305814833	305814824	9718233	1723028
Italia, Siena, vía: 'Via delle Terme					

Tabla 4.18: Prueba 33 - Europe.

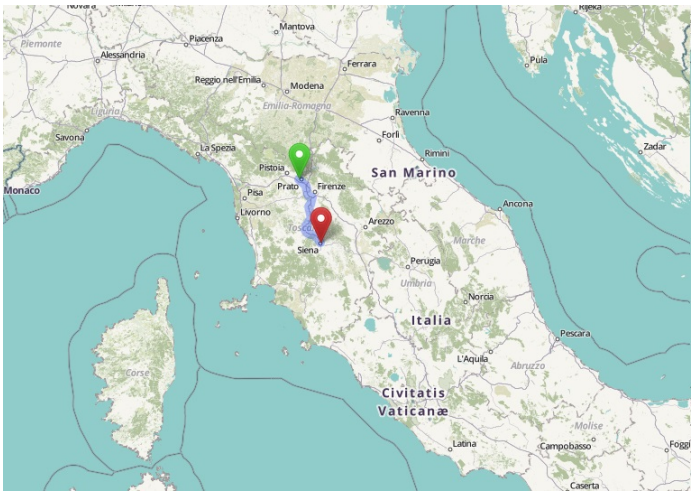


Figura 4.17: Prueba 33 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
10197465	49252333	977015471	995270013	8324560	17383662
España, Palencia, vía: 'Calle Don Pelayo'					
5945858	29899074	329515966	329515983	4950271	4950272
Francia, Pau, vía: 'Rue du Midi'					

Tabla 4.19: Prueba 41 - Europe.

Figura 4.18: Prueba 41 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
11448166	60759251	247492031	349327489	2937407	5371476
España, Castilla y León, Valladolid, vía: 'Paseo del Cauce'					
18766205	143461234	1569797853	32596647	463705	1597149
Polonia, Varsovia, vía: 'Twarda'					

Tabla 4.20: Prueba 42 - Europe.

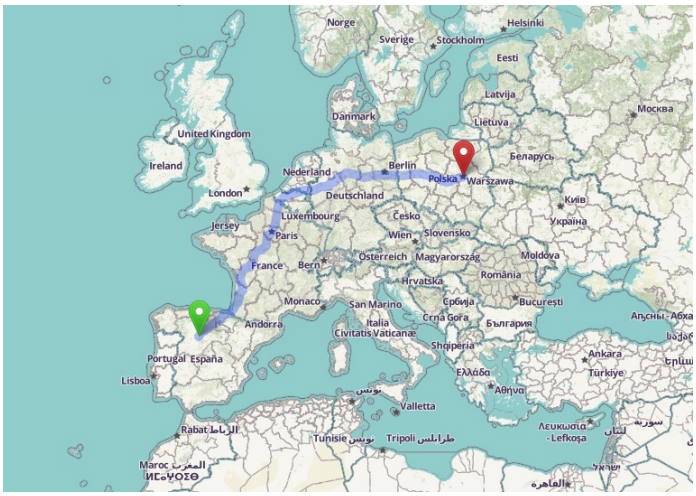


Figura 4.19: Prueba 42 - Europe.

id	osm_id	osm_source_id	osm_target_id	source	target
11448166	60759251	247492031	349327489	2937407	5371476
España, Castilla y León, Valladolid, vía: 'Paseo del Cauce'					
6901861	32821113	3212623436	74985297	1533414	1454980
Luxemburgo, Luxemburgo, vía: 'Rue Willy Goergen'					

Tabla 4.21: Prueba 43 - Europe.

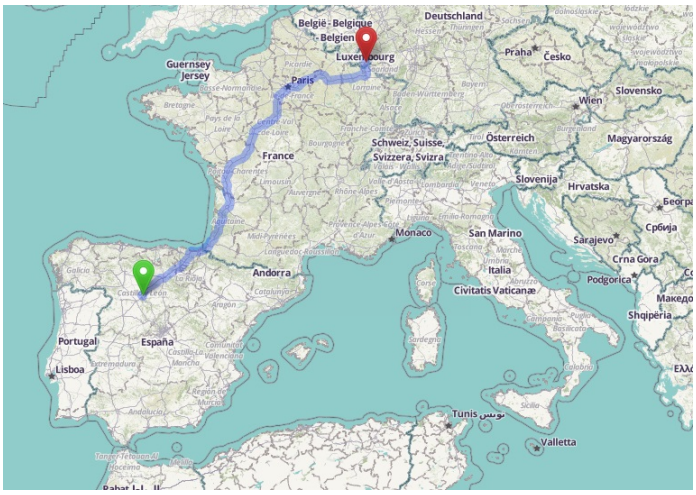


Figura 4.20: Prueba 43 - Europe.

4.2.3. Resultados obtenidos

La siguiente tabla recoge el tiempo de ejecución de cada una de las pruebas presentadas en el apartado anterior. Aparecen las quince pruebas junto con sus resultados para las cuatro funciones pgRouting con las que se trabaja. Para cada una de estas funciones se midió el tiempo de tres ejecuciones y se obtuvo su media, todo ello en la unidad de milisegundos. Dicha media se convierte a segundos decimales para facilitar su observación y evaluación.

Comenzaremos analizando cada algoritmo. Lo primero que se puede apreciar es como, tanto el algoritmo A* como el algoritmo B.A*, no reflejan ningún tipo de resultado. Esto se debe a que a la hora de utilizar ambos algoritmos sobre la cartografía europea, no se obtiene ninguna ruta. De hecho, dichas funciones retornaban siempre el mismo mensaje:

```
ERROR: invalid memory alloc request size 1073792000
```

Este problema fue trascendental para la viabilidad de este proyecto. Por esta razón, el posterior capitulo esta dedicado íntegramente a su análisis y búsqueda de solución. Por consiguiente, no es necesario entrar en más detalles en este apartado.

Descartados A* y su versión bidireccional, retomemos el análisis observando los valores que recogen Dijkstra y B.Dijkstra. Como puede apreciarse para ambos algoritmos se obtiene solución en todas las pruebas realizadas. Pero son valores muy elevados, que van desde los 142,68 a los 206.56 segundos decimales. Es decir, estamos hablando de que calcular una ruta óptima sobre la cartografía europea puede tardar entre 2 min 23 s y 3 min 27 s.

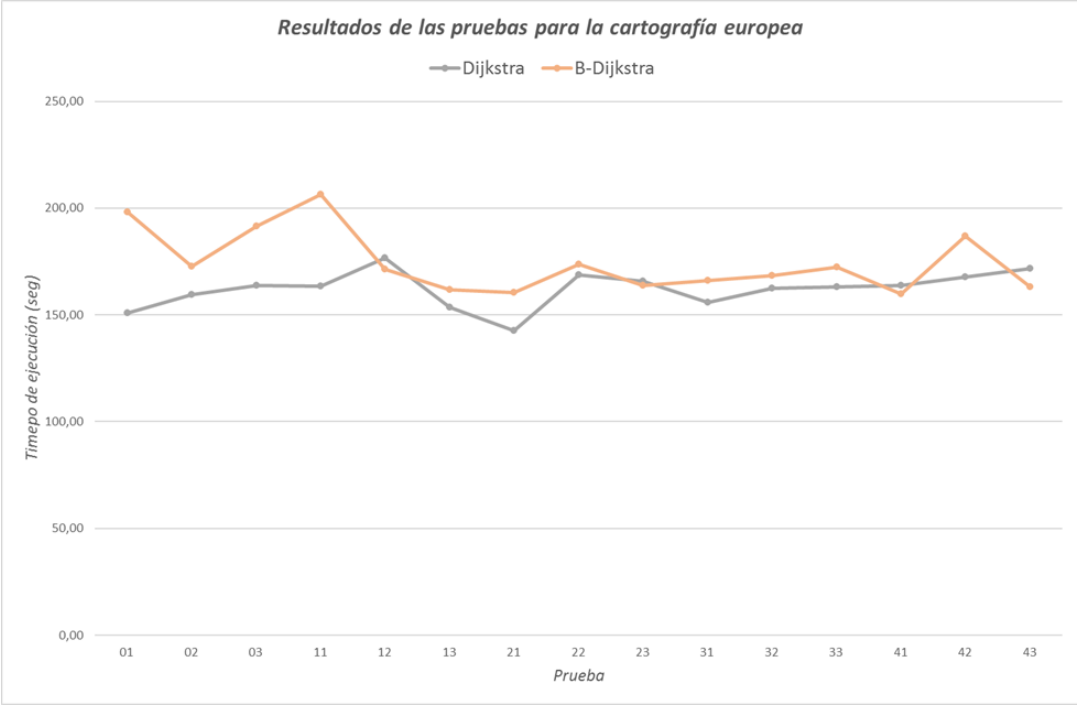


Figura 4.21: Resultados de las pruebas para la cartografía europea.

El tiempo que pgRouting tarda en calcular la ruta tomando como origen y destino el mismo punto, y el tiempo que tarda cuando origen y destino pertenecen a dos ciudades distintas de distintos países, no muestran apenas diferencia. Incluso en lo que respecta a los resultados de aplicar una u otra función. Sólo se observa que el algoritmo B.Dijkstra presenta una peor respuesta para los casos de prueba básicos, donde origen y destino son el mismo nodo o nodos adyacentes.

Lógicamente estos tiempos de respuesta obtenidos deben ser mejorados, es impensable que un usuario del sistema espere dichos tiempos para obtener respuesta a su consulta. Este hecho también se aborda en el posterior capítulo.

Algoritmo \ *Prueba*		01	02	03	11	12	13	21	22	23
A*	1ª	*	*	*	*	*	*	*	*	*
	2ª	*	*	*	*	*	*	*	*	*
	3ª	*	*	*	*	*	*	*	*	*
	Media:									
	Segundos:									
B.A.*	1ª	*	*	*	*	*	*	*	*	*
	2ª	*	*	*	*	*	*	*	*	*
	3ª	*	*	*	*	*	*	*	*	*
	Media:									
	Segundos:									
Dijkstra	1ª	130079,653	116629,294	192384,010	136657,726	205068,029	140163,525	134499,504	160168,427	194351,109
	2ª	162913,776	147531,456	153856,292	159370,618	148684,542	153081,831	132804,491	181648,517	143811,049
	3ª	159749,598	214247,663	145342,242	194352,206	176095,893	167662,166	160749,304	163866,461	159233,170
	Media:	150914,342	159469,471	163860,848	163460,183	176616,155	153635,841	142684,433	168561,135	165798,443
	Segundos:	150,91	159,47	163,86	163,46	176,62	153,64	142,68	168,56	165,80
B. Dijkstra	1ª	179583,746	185425,058	204526,026	180660,94	164512,503	196788,702	154544,578	167278,145	162567,116
	2ª	190302,452	164332,632	168586,597	166088,771	166877,993	126298,664	167183,69	179623,229	155067,764
	3ª	224724,212	168603,582	201488,673	272927,936	182513,854	161822,645	159796,196	173823,691	173625,707
	Media:	198203,470	172787,091	191533,765	206559,216	171301,450	161636,670	160508,155	173575,022	163753,529
	Segundos:	198,20	172,79	191,53	206,56	171,30	161,64	160,51	173,58	163,75

Tabla 4.22: Resultados de las pruebas para la cartografía europea (1/2)

Algoritmo \ *Prueba*		21	22	23	31	32	33	41	42	43
A*	1ª	*	*	*	*	*	*	*	*	*
	2ª	*	*	*	*	*	*	*	*	*
	3ª	*	*	*	*	*	*	*	*	*
	Media:									
	Segundos:									
B.A.*	1ª	*	*	*	*	*	*	*	*	*
	2ª	*	*	*	*	*	*	*	*	*
	3ª	*	*	*	*	*	*	*	*	*
	Media:									
	Segundos:									
Dijkstra	1ª	134499,504	160168,427	194351,109	162248,800	143906,471	153564,702	165610,770	164848,643	170857,522
	2ª	132804,491	181648,517	143811,049	147727,582	164117,202	155869,517	155510,496	163456,030	175707,119
	3ª	160749,304	163866,461	159233,170	157846,779	179338,445	180014,453	169947,935	174687,771	168876,985
	Media:	142684,433	168561,135	165798,443	155941,054	162454,039	163149,557	163689,734	167664,148	171813,875
	Segundos:	142,68	168,56	165,80	155,94	162,45	163,15	163,69	167,66	171,81
B. Dijkstra	1ª	154544,578	167278,145	162567,116	184430,304	168208,41	196206,942	203906,129	179275,356	156872,049
	2ª	167183,69	179623,229	155067,764	149724,121	163320,602	158794,951	122960,822	190894,064	131549,883
	3ª	159796,196	173823,691	173625,707	163790,835	173555,318	162276,141	152941,428	181739,27	200467,353
	Media:	160508,155	173575,022	163753,529	165981,753	168361,443	172426,011	159936,126	187021,161	162963,095
	Segundos:	160,51	173,58	163,75	165,98	168,36	172,43	159,94	187,02	162,96

Tabla 4.23: Resultados de las pruebas para la cartografía europea (2/2)

Capítulo 5

La importancia del área de influencia

A la hora de aplicar los algoritmos A* y su versión bidireccional sobre la cartografía europea, surgio un error inesperado. No retornaban la ruta para ninguno de los casos de prueba utilizados. La unica salida optenida de la colsulta era:

```
ERROR: invalid memory alloc request size 1073792000
```

Este problema supuso un auténtico rompecabezas. Inicialmente se sospecho de la falta de memoria, ampliando la RAM de la máquina virtual donde se encuentra instalada la plataforma a un valor tan elevado como 32Gb. Aún así este error se mantuvo en las consultas con A* y B.A*.

Se sospecho de la configuración de PostgreSQL, por lo que se probo a modificar los parámetros de dicha configuración. Enfocando especialmente nuestra atención en aquellos parámetros que juegan un papel importante en las operaciones de asignación de memoria, pero no se consiguió ninguna mejora.

Consultando en la web, personas familiarizadas con este tipo de sistemas SIG, comentaban la posibilidad de que se tratara de un error debido la existencia de algún dato corrupto antes de realizar la importación a PostgreSQL. Así que se descargo la cartografía europea OSM nuevamente, comprobando su integridad mediante su hash MD5 que es facilitado desde los distintos puntos de descarga OpenStreetMap. Pero nuevamente el error no procedía de ahí.

Finalmente, preguntamos por nuestro error en el soporte de incidencias que pgRouting pone a disposición de sus usuarios. La contestación de uno de sus desarrolladores no tardo en llegar, confirmando nuestras sospechas iniciales: el error procedía de la falta de memoria. Al parecer el algoritmo A* y B.A* cargan todos los registros almacenados en la tabla de enrutado (eu_2po_4pgr, tabla que osm2po genera tras la importación cartográfica de Europa) en memoria, por lo que si se estamos trabajando con grandes áreas geográficas y no disponemos de suficiente memoria, no es posible utilizar dichos algoritmos.

Este inconveniente no es reflejado por pgRouting en su documentación, aspecto que deberían señalar. Pero a pesar de ello, nos ofrecieron una solución: limitar la búsqueda de la ruta óptima a un área de influencia limitada por origen y destino.

5.1. Nociones básicas

Antes de entrar en los detalles de las soluciones que se plantearon para abordar el problema, es necesario saber los tipos de geometría ofrecidos por PostGIS, así como analizar el campo "geom_way" importado en nuestras tablas. Con esto se pretende facilitar el entendimiento de las áreas de influencias presentadas posteriormente.

5.1.1. Tipos de geometría

Los tipos de geometría que PostGIS soporta con plena funcionalidad son principalmente los tipos: Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon y GeometryCollection. Estos tipos han sido soportados por PostGIS desde su inicio y además son los tipos básicos en los que se apoya cualquier SIG. Además, se definen dos formas de representación espacial de las geometrías: WKT (Well-Known Text) y WKB (Well-Know Binary). Ambas formas guardan información del tipo de objeto y sus coordenadas.

La representación WKT es una codificación en formato ASCII estandarizada diseñada para describir objetos espaciales expresados de forma vectorial. Su especificación ha sido promovida por un organismo internacional, el Open Geospatial Consortium (OGC), siendo su sintaxis muy fácil de utilizar, de forma que es muy generalizado su uso en la industria geoinformática. Muchas de las bases de datos espaciales, y en especial PostgreSQL/PostGIS, utilizan esta codificación.

Por su parte, la representación WKB es una variante de WKT expresada de forma binaria, la cual es también utilizada por estos gestores espaciales, pero con la ventaja de que al ser compilada en forma binaria la velocidad de proceso es muy elevada. Algunos ejemplos de representación WKT de estos siete tipos de geometrías son:

Tipo de geometría	Representación	Comentario
POINT	"POINT (2 2)"	un Point
LINESTRING	"LINESTRING (4 1, 1 4, 1 1, 4 3)"	un LineString con 3 Point
POLYGON	"POLYGON ((1 4, 1 1, 4 1, 4 2, 3 2, 3 4, 1 4))" "POLYGON ((1 1, 5 1, 5 5, 1 5, 1 1), (2 2, 4 2, 4 4, 2 4, 2 2))"	un Polygon con 1 exteriorRing y 0 interiorRing (isla) un Polygon con 1 exteriorRing y 1 interiorRing (isla)
MULTIPOINT	"MULTIPOINT (1 1, 2 4, 3 2)" "MULTIPOINT ((1 1), (2 4), (3 2))"	un MultiPoint con 3 Point. Ambas sintaxis son aceptadas por PostGIS.
MULTILINESTRING	"MULTILINESTRING ((1 4, 1 1, 3 1), (4 5, 2 2, 4 2, 2 4))"	un MultiLineString con 2 LineString
MULTIPOLYGON	"MULTIPOLYGON (((1 4, 4 4, 4 1, 1 1, 1 4), (3 2, 2 2, 2 3, 3 3, 3 2)), ((1 6, 4 6, 4 5, 1 5, 1 6)))"	un MultiPolygon con 2 Polygon (1 de ellos con una isla)
GEOMETRY COLLECTION	"GEOMETRYCOLLECTION (LINESTRING (1 4, 2 4, 2 2), LINESTRING (1 3, 1 1, 2 1), POINT (3 4), POINT (3 1), POLYGON ((4 4, 4 1, 7 1, 7 4, 4 4), (5 3, 5 2, 6 2, 5 3)), POLYGON ((8 4, 8 1, 9 1, 9 4, 8 4)))"	una Geometrycollection formada por 2 Points, 2 LineString y 2 Polygon. También puede estar formada por elementos de tipo Multi así como de otras GeometryCollection incluso de forma anidada.

Tabla 5.1: Representación WKT de objetos espaciales.

En la figura siguiente se muestra de forma gráfica la representación de estos siete tipos de geometrías:

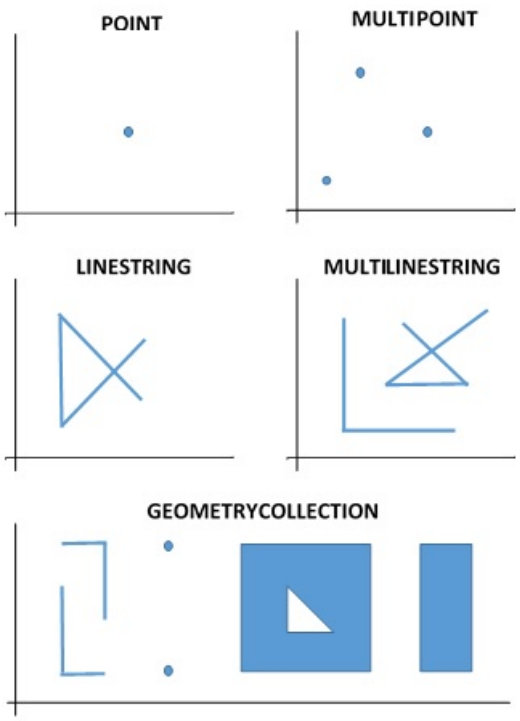


Figura 5.1: Tipos básicos de geometrías en PostGIS.

5.1.2. Campo geom_way o geom

PostGIS almacena la información geográfica en una columna del tipo GEOMETRY, donde por lo general se almacena la geometría en formato WKB (Well-Known Binary).

Todas nuestras tablas generadas a partir de las herramientas de importación destinadas a utilizar pgRouting, crean una columna del tipo GEOMETRY, que se ajusta a la siguiente especificación:

Columna	Tipo
geom_way	geometry(LineString, 4326)

Tabla 5.2: Especificación de la columna de geometría.

Lo que nos indica que la información geográfica almacenada se basa en una geometría de tipo LineString, lo cual concuerda con que la información que estamos almacenando tiene que ver con la red de vías (calles, carreteras, etc). También se especifica el SRID, Identificador del Sistema de Referencia Espacial, que especifica un valor único para el sistema de coordenadas empleado, en este caso 4326.

PostGIS ofrece multitud de funciones que permiten transformar el sistema de referencia espacial utilizado, así como el tipo de geometría siempre que la conversión sea posible.

5.2. Propuesta de los desarrolladores

Identificado el origen del problema, la falta de memoria, se procedió a probar la solución planteada por los desarrolladores de pgRouting: cargar sólo los nodos comprendidos en un cierto área de influencia limitado por los puntos de origen y destino.

Con frecuencia las operaciones SIG requieren la generación de áreas de influencia (buffering) en ciertos análisis. La forma más simple de área de influencia es la que se genera en torno a datos puntuales ya que el proceso implica tan solo la creación de un polígono "circular" en torno al punto, de radio equivalente a la distancia del buffer.

El modelo de consulta propuesta por los desarrolladores de pgRouting fue la siguiente:

```
select id, source, target, cost from eu_2po_4pgr where ST_Dwithin(geom, (select
ST_MakeLine(geom) from vista_vertice_table where id in (8324560, 17383662)), radius)
```

Esta consulta se corresponde con la sentencia sql que se introduce en la función de pgRouting a utilizar, es decir, como se vio en la sección "3.5. Rutas mediante pgRouting" apartado "3.5.3 Problema del camino más corto", dichas funciones reciben como primer parámetro una consulta SQL que se correspondería con esta. Siendo el resultado final de la consulta:

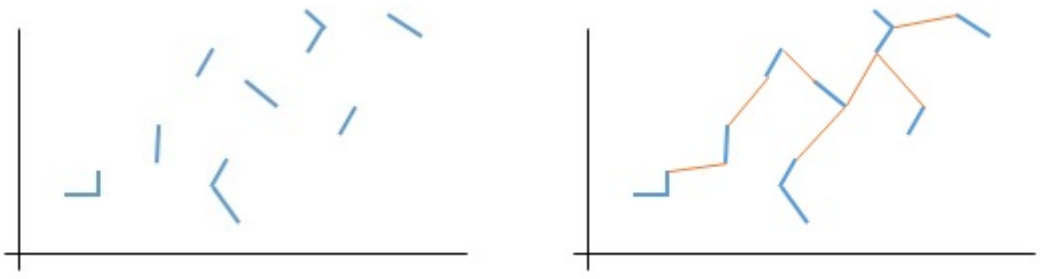
```
select seq, id1 AS node, id2 AS edge, cost from pgr_dijkstra('select id, source, target,
cost from eu_2po_4pgr where ST_Dwithin(geom, (select ST_MakeLine(geom) from vista_vertice
_table where id in (8324560, 17383662)), radius)', 8324560, 17383662, false, false);
```

Ahora, ¿que hace esta consulta?. Primero analicemos las funciones PostGIS que aparecen en ella:

- ST_MakeLine

  ```
  geometry ST_MakeLine(geometry set geoms);
  ```

  Crea un LineString a partir de la geometría que recibe como argumento. Es decir, en este caso, devuelve una línea formada por todas las vías que recibe. En la ilustración queda mas claro:



(a) Conjunto de geometrías LineString (b) Resultado de la función ST_MakeLine

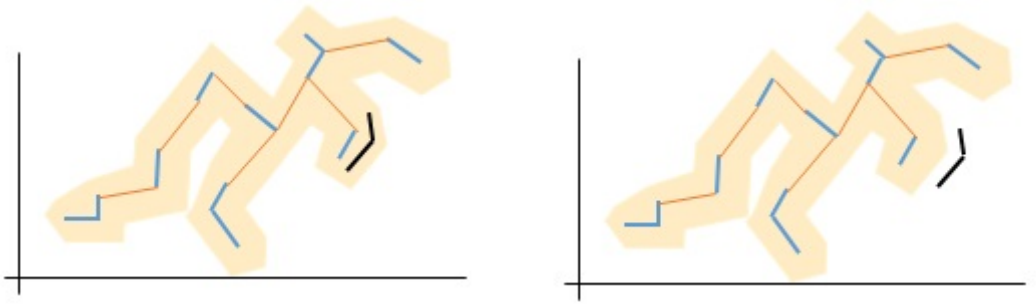
Figura 5.2: Ejemplo de la función ST_MakeLine.

- ST_Dwithin

  ```
  boolean ST_DWithin(geometry g1, geometry g2, double precision distance_of_srid);
  ```

Una operación de análisis de proximidad típica es la búsqueda o selección de entidades geométricas que se encuentran dentro de una determinada distancia de otras entidades geométricas. Esta función permite calcular si dos objetos se encuentran a una distancia dada uno del otro, en cuyo caso devuelve true.

Es importante resaltar la trascendencia del orden de los argumentos, dado que esta función comprueba si "g1" esta dentro del área de influencia especificado para "g2", como muestra la figura a continuación:



(a) Resultado de la función ST_Dwithin (True) (b) Resultado de la función ST_Dwithin (False)

Figura 5.3: Ejemplo de la función ST_Dwithin.

Podemos observar como "g2" se corresponde con el resultado de la función ST_MakeLine visto anteriormente, pero con un área de influencia. La geometría "g1" esta dentro del área de influencia especificado para q2 en la ilustración izquierda, cosa que no ocurre en la derecha.

Principalmente, ST_Dwithin trabaja con el tipo geometría especificando la distancia en grados, pero si se desea especificar la distancia en metros bastaría con trabajar con el tipo geografía.

```
boolean ST_DWithin(geography gg1, geography gg2,double precision distance_meters);
```

La siguiente tabla muestra una aproximación de la conversión entre la especificación de las distancias en grados o metros:

Grados	Distancia
1.0	111 km
0.1	11.1 km
0.01	1.11 km
0.001	111 m
0.0001	1.11 m
0.00001	0.111 m
0.000001	1.11 cm
0.0000001	1.11 mm

Tabla 5.3: Conversión de las distancias en grados o metros.

Volviendo a la consulta general, con la combinación de ambas funciones lo que se logra es: a partir de origen y destino (en el ejemplo: 8324560, 17383662) obtener todos los nodos que se encuentran entre ellos y unirlos mediante una línea con ST_MakeLine. Tras lo cual, se aplica la función ST_Dwithin sobre

dicha línea y toda la geometría europea, para así lograr limitar la búsqueda solo a un área, en concreto las vías que se encuentran dentro del buffer.

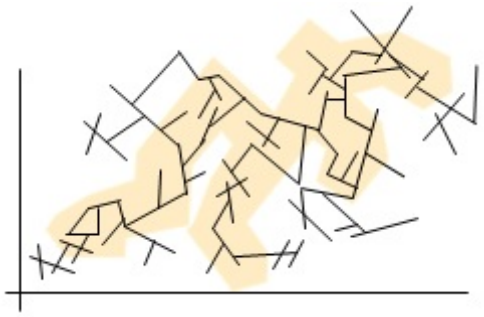


Figura 5.4: Área de influencia propuesta por los desarrolladores.

5.2.1. Resultados de la propuesta

La siguiente tabla recoge el tiempo de ejecución de cada uno de los casos de prueba de la cartografía europea. Aparecen las quince pruebas junto con sus resultados para las cuatro funciones pgRouting con las que se trabaja. Para cada una de estas funciones se midió el tiempo de tres ejecuciones y se obtuvo su media, todo ello en la unidad de milisegundos. Dicha media se convierte a segundos decimales para facilitar su observación y evaluación.

Comenzaremos analizando cada algoritmo. Lo primero que se puede apreciar es como, tanto el algoritmo A* como el algoritmo B.A*, esta vez si reflejan algún resultado. En concreto, se limitan a rutas basadas en el mismo nodo, nodos adyacentes y nodos pertenecientes a vías de la misma ciudad, no siendo posible obtener resultado para vías de ciudades distintas del mismo país o países diferentes. No obstante, es un resultado que confirma que el problema se debía al intento de ambas funciones por cargar todos los registros en memoria. Limitando dicha carga a los registros pertenecientes a un área de influencia, se solventa en parte dicho problema.

Se debe resaltar que, como se ha podido ver en el apartado anterior, para definir el área es necesario proporcionar la distancia o radio de influencia. Este es un aspecto que se debe tener en cuenta, puesto que cuanto mayor sea la distancia entre origen y destino, mayor debe ser el radio proporcionado en la consulta. En la realización de las pruebas fue necesario afinar dicho valor para cada caso, especialmente para las últimas pruebas. Este punto es relevante porque especificar un área pequeña puede dar lugar a que pgRouting no pueda calcular la ruta, mientras que especificar un área demasiado grande puede implicar cargar demasiados registros en memoria.

Esta parece ser la causa por la que A* y B.A* no funcionan en rutas grandes. El área de influencia que genera la propuesta de los desarrolladores, engloba demasiados registros, provocando que en grandes distancias se vuelva a obtener el famoso error:

```
ERROR: invalid memory alloc request size 1073792000
```

Algoritmo \ Prueba		01	02	03	11	12	13	21	22	23
*A**	1ª	64861,542	610908,030	633221,814	656996,390	709446,365	705267,986	662163,376	709232,973	625554,929
	2ª	556441,476	645459,224	612753,717	653348,463	697637,829	584398,144	707607,583	683327,221	678481,235
	3ª	594092,929	565412,227	528324,086	610547,337	577325,237	588869,01	596486,836	499741,134	561552,354
	Media:	598465,316	607259,827	591433,206	640297,397	661469,810	626178,380	655419,265	630767,109	621862,839
	Segundos:	598,47	607,26	591,43	640,30	661,47	626,18	655,42	630,77	621,86
*B.A.**	1ª	536492,703	567956,227	717519,320	703774,352	718935,133	691201,868	722240,970	634265,209	686270,373
	2ª	759039,841	725038,981	652797,484	575103,935	746033,798	542696,944	622676,393	494975,882	632930,802
	3ª	565352,222	566619,561	537357,396	597624,673	517642,712	612467,086	588528,397	505525,852	583984,314
	Media:	620294,922	619871,590	635891,400	625500,987	660870,548	615455,299	644481,920	544922,314	634395,163
	Segundos:	620,29	619,87	635,89	625,50	660,87	615,46	644,48	544,92	634,40
Dijkstra	1ª	127784,169	121618,092	126552,496	135307,707	146391,582	137764,565	113344,511	154860,579	160830,344
	2ª	115716,793	137690,611	151044,251	125737,298	123653,235	110865,689	122686,184	144338,539	155643,251
	3ª	117307,423	113040,821	132179,619	136768,787	159599,288	133956,166	122894,577	111149,362	105279,988
	Media:	120269,462	124116,508	136592,122	132604,597	143214,702	127528,807	119641,757	136782,827	140584,528
	Segundos:	120,27	124,12	136,59	132,60	143,21	127,53	119,64	136,78	140,58
B. Dijkstra	1ª	132574,908	115413,903	142434,003	134898,981	142310,033	152967,215	124779,611	155523,96	144655,514
	2ª	123123,596	126434,335	132403,743	146290,574	113230,485	161354,485	159178,478	139469,488	141958,777
	3ª	126980,943	144007,827	135303,014	117404,558	130019,628	144215,725	137496,556	124965,221	123015,337
	Media:	127559,816	128618,688	136713,587	132864,704	128520,049	152845,808	140484,882	139986,223	136543,209
	Segundos:	127,56	128,62	136,71	132,86	128,52	152,85	140,48	139,99	136,54

Tabla 5.4: Resultados de la propuesta (1/2)

Algoritmo \ *Prueba*		21	22	23	31	32	33	41	42	43
A*	1ª	662163,376	709232,973	625554,929	*	*	*	*	*	*
	2ª	707607,583	683327,221	678481,235	*	*	*	*	*	*
	3ª	596486,836	499741,134	561552,354	*	*	*	*	*	*
	Media:	655419,265	630767,109	621862,839						
	Segundos:	655,42	630,77	621,86						
B.A.*	1ª	722240,970	634265,209	686270,373	*	*	*	*	*	*
	2ª	622676,393	494975,882	632930,802	*	*	*	*	*	*
	3ª	588528,397	505525,852	583984,314	*	*	*	*	*	*
	Media:	644481,920	544922,314	634395,163						
	Segundos:	644,48	544,92	634,40						
Dijkstra	1ª	113344,511	154860,579	160830,344	117584,984	135591,106	117267,979	136602,942	135277,129	142260,315
	2ª	122686,184	144338,539	155643,251	126284,624	124673,124	156424,145	141254,522	156234,546	146051,586
	3ª	122894,577	111149,362	105279,988	116284,624	138442,264	144596,235	144564,456	136530,486	156951,184
	Media:	119641,757	136782,827	140584,528	120051,411	132902,165	139429,453	140807,307	142680,720	148421,028
	Segundos:	119,64	136,78	140,58	120,05	132,90	139,43	140,81	142,68	148,42
B. Dijkstra	1ª	124779,611	155523,96	144655,514	101149,549	134092,066	118840,729	154202,294	124910,161	139411,141
	2ª	159178,478	139469,488	141958,777	161257,563	128954,165	148889,462	126482,562	160168,622	140992,546
	3ª	137496,556	124965,221	123015,337	145128,452	144618,462	154862,004	135994,523	159852,264	139854,775
	Media:	140484,882	139986,223	136543,209	135845,188	135888,231	140864,065	138893,126	148415,802	140086,154
	Segundos:	140,48	139,99	136,54	135,85	135,89	140,86	138,89	148,42	140,09

Tabla 5.5: Resultados de la propuesta (2/2)

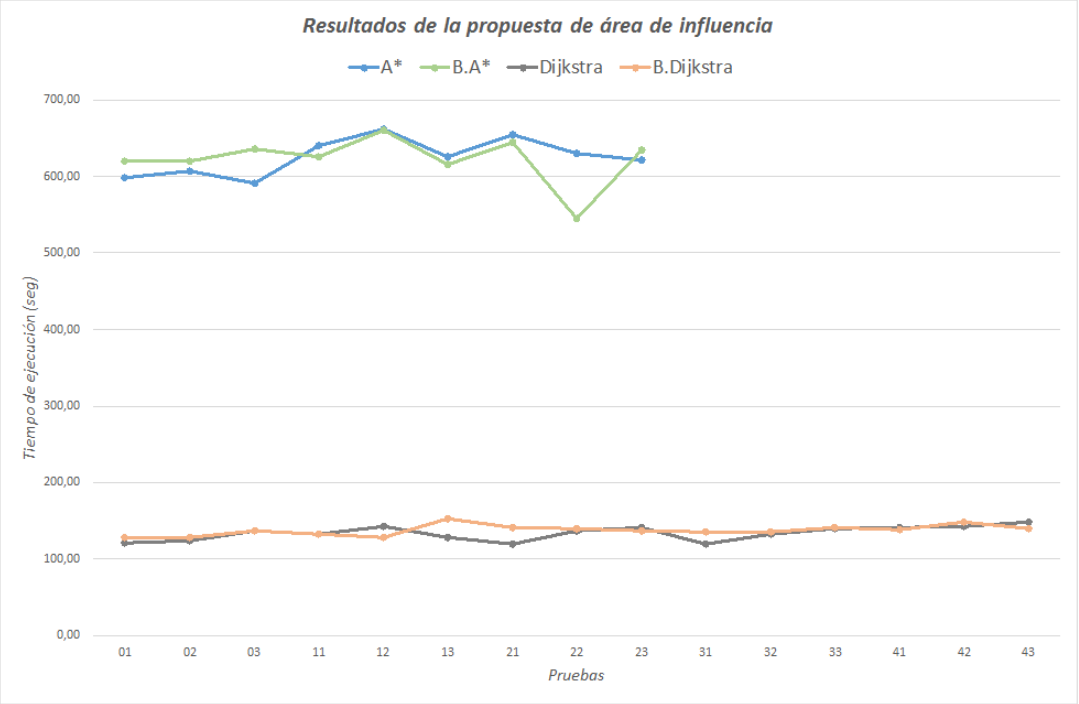


Figura 5.5: Resultados de la propuesta de área de influencia.

En cuanto a los tiempos de respuesta obtenidos, puede apreciarse una significativa diferencia entre los tiempos de las versiones A* y las versiones Dijkstra. Las versiones de Dijkstra se mantienen entre los 119,64 a los 152,85 segundos decimales, valores ligeramente más pequeños que los que se obtenían en las pruebas iniciales antes de aplicar el área de influencia. La ruta óptima sobre la cartografía europea es calculada en torno a los 2 m - 2 m 33 s, mientras que antes de aplicar el buffer el tiempo de respuesta rondaba entre los 2 min 23 s - 3 min 27 s.

Por su parte, las versiones A* requieren tiempos de respuesta desorbitados. Los tiempos de ejecución para estas funciones se encuentran en un rango de 544,92 a 661,47 segundos decimales. Es decir, estamos hablando de que calcular una ruta óptima sobre la cartografía europea puede tardar entre 9 min 5 s y 11 min 2 s. Esto implica que a pesar de que las versiones A* devuelvan la ruta calculada, el tiempo de ejecución es tan elevado que hace imposible su utilización.

El tiempo que pgRouting tarda en calcular la ruta tomando como origen y destino el mismo punto, y el tiempo que tarda cuando origen y destino pertenecen a dos ciudades distintas de distintos países, no muestran apenas diferencia en lo que respecta a versiones del mismo algoritmo. Pero estos tiempos de respuesta deben ser mejorados para garantizar la funcionalidad de la plataforma.

5.3. Definición de nuevas áreas de influencia

Ante los resultados obtenidos con la propuesta de los desarrolladores, se considero necesario que la siguiente tarea a desempeñar fuera la búsqueda y definición de nuevas áreas de influencia que permitieran mejorar los resultados obtenidos hasta el momento. Se obtuvieron así dos nuevas definiciones, basadas en la idea propuesta por los desarrolladores. Fueron nombradas como: BufferLineString y BufferMultiLineString, ambas serán definidas en los siguientes apartados.

5.3.1. BufferLineString

Como se menciono anteriormente, la forma más simple de área de influencia es la que consiste simplemente en crear un polígono "circular" en torno a un punto, de radio equivalente a la distancia del buffer. Pues bien, la definición del área de influencia BufferLineString se basa en esta idea.

Dada la vía de origen se le aplica la función ST_Buffer para obtener un área de influencia en torno a ella según la distancia especificada. Dicha área de influencia se combina con toda la geometría europea para obtener los nodos comprendidos en el área. Así se obtiene el buffer temporal que servirá para aplicar el algoritmo de pgRouting deseado.

```
with buffered as (select * from eu_2po_4pgr where geom_way && (select ST_Buffer(geom_way,
radius) from eu_2po_4pgr where id=10197465)) select id, source, target, cost from buffered
```

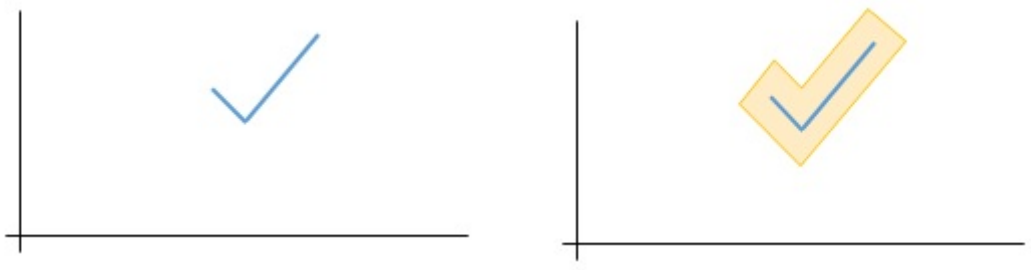
Las funciones PostGIS utilizadas en la definición de este buffer son:

- ST_Buffer

  ```
  geometry ST_Buffer(geometry g1, float radius_of_buffer);
  ```

  Toma un objeto de geometría y una distancia, y devuelve una geometría que representa todos los puntos cuya distancia de la geometría origen es menor o igual a la distancia. Al igual que ST_DWithin trabaja con el tipo geometría especificando la distancia en grados, pero si se desea especificar la distancia en metros bastaría con trabajar con el tipo geografía.

  ```
  geography ST_Buffer(geography g1, float radius_of_buffer_in_meters);
  ```



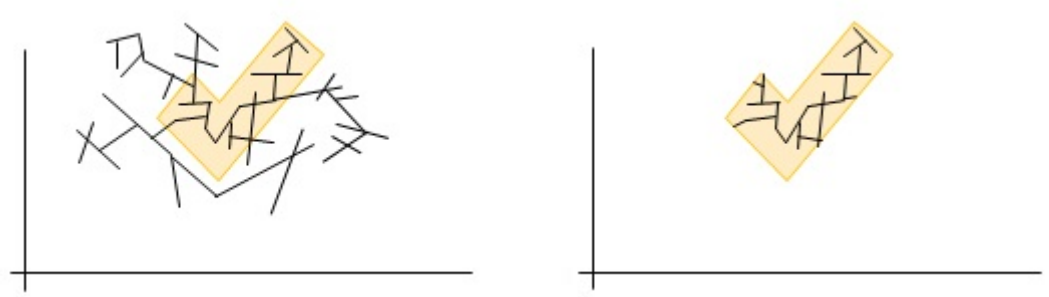
(a) Geometría LineString (b) Resultado de la función ST_Buffer

Figura 5.6: Ejemplo de la función ST_Buffer.

- &&

  ```
  boolean &&( geometry A , geometry B );
  ```

  Este operador de PostGIS realiza una operación espacial de superposición de capas (overlay), como una intersección de capas. Es decir, superpone ambos operandos y devuelve el área que tienen en común.



(a) Geometría LineString (b) Resultado de la función ST_Buffer

Figura 5.7: Ejemplo del operador &&.

El resultado final de la consulta a utilizar con pgRouting sería:

```
select seq, id1 AS node, id2 AS edge, cost from pgr_dijkstra('with buffered as (select *
from eu_2po_4pgr where geom_way && (select ST_Buffer(geom_way,radius) from eu_2po_4pgr
where id=10197465)) select id, source, target, cost from buffered', 8324560, 17383662,
false, false);
```

5.3.2. Resultados obtenidos para BufferLineString

La siguiente tabla recoge el tiempo de ejecución de cada uno de los casos de prueba de la cartografía europea. Aparecen las quince pruebas junto con sus resultados para las cuatro funciones pgRouting con las que se trabaja. Para cada una de estas funciones se midió el tiempo de tres ejecuciones y se obtuvo su media, todo ello en la unidad de milisegundos. Dicha media se convierte a segundos decimales para facilitar su observación y evaluación.

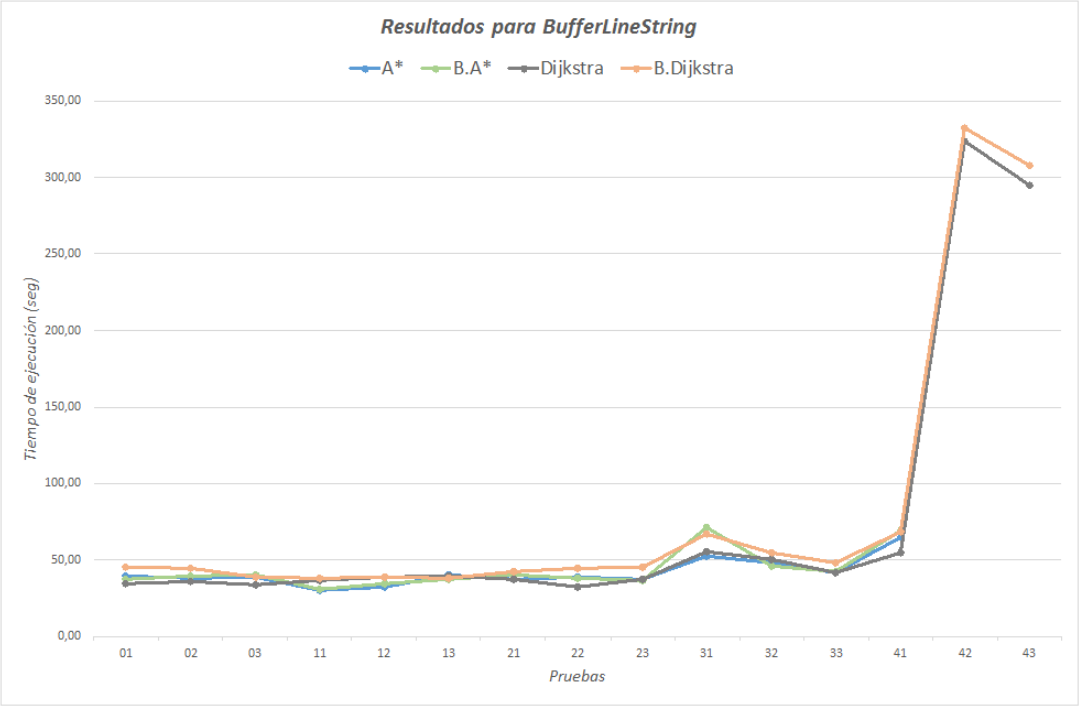


Figura 5.8: Resultados de BufferLineString.

Algoritmo \ *Prueba*		01	02	03	11	12	13	21	22	23
A*	1ª	38875,298	39304,641	36101,832	22492,024	39936,679	40926,785	39620,778	39864,067	39828,378
	2ª	39539,514	38858,047	40483,979	33466,357	23709,456	40964,776	39767,732	39982,626	40778,071
	3ª	39335,901	34959,711	40387,443	33499,054	33926,332	39247,131	32241,443	36632,033	31363,208
	Media:	39250,238	37707,466	38991,085	29819,145	32524,156	40379,564	37209,984	38826,242	37323,219
	Segundos:	39,25	37,71	38,99	29,82	32,52	40,38	37,21	38,83	37,32
B.A.*	1ª	32568,261	40088,635	40351,889	22208,894	40023,844	39608,613	40066,797	39951,122	39511,721
	2ª	39462,881	38361,116	39947,691	40182,036	35709,903	40371,187	39943,978	35636,419	40235,671
	3ª	40291,803	40057,046	40077,077	29461,769	28099,813	31529,419	39964,824	39211,322	30254,596
	Media:	37440,982	39502,266	40125,552	30617,566	34611,187	37169,740	39991,866	38266,288	36667,329
	Segundos:	37,44	39,50	40,13	30,62	34,61	37,17	39,99	38,27	36,67
Dijkstra	1ª	33819,877	29614,708	33694,233	36534,239	45209,673	39327,409	37292,013	30887,516	39196,617
	2ª	32240,302	38756,164	32237,744	34268,844	45120,301	39676,903	38986,666	33288,999	39287,383
	3ª	37237,321	38917,547	34732,706	38820,247	25463,324	39171,943	36399,005	32979,433	34002,332
	Media:	34432,500	35762,806	33554,894	36541,110	38597,766	39392,085	37559,228	32385,316	37495,444
	Segundos:	34,43	35,76	33,55	36,54	38,60	39,39	37,56	32,39	37,50
B. Dijkstra	1ª	50144,942	43189,684	33661,769	24429,584	38847,897	32323,635	37516,705	44642,519	44873,603
	2ª	44778,581	45341,214	38474,183	44696,446	38741,703	37950,053	44852,131	44854,064	44910,709
	3ª	41694,895	45206,247	44765,326	44481,669	38252,932	44915,958	44587,691	44468,964	45075,401
	Media:	45539,473	44579,048	38967,093	37869,233	38614,177	38396,549	42318,842	44655,182	44953,238
	Segundos:	45,54	44,58	38,97	37,87	38,61	38,40	42,32	44,66	44,95

Tabla 5.6: Resultados de BufferLineString (1/2)

Algoritmo	Prueba	21	22	23	31	32	33	41	42	43
*A**	1ª	39620,778	39864,067	39828,378	59962,207	40582,349	37986,881	62631,934	*	*
	2ª	39767,732	39982,626	40778,071	42396,704	51272,932	43797,358	66017,809	*	*
	3ª	32241,443	36632,033	31363,208	54641,802	52360,515	43938,942	65760,912	*	*
	Media:	37209,984	38826,242	37323,219	52333,571	48071,932	41907,727	64803,552		
	Segundos:	37,21	38,83	37,32	52,33	48,07	41,91	64,80		
*A**	1ª	40066,797	39951,122	39511,721	74343,689	38336,782	40335,722	64729,908	*	*
	2ª	39943,978	35636,419	40235,671	66284,482	52134,065	43034,895	74521,049	*	*
	3ª	39964,824	39211,322	30254,596	72653,84	46509,988	43688,908	69028,248	*	*
	Media:	39991,866	38266,288	36667,329	71094,004	45660,278	42353,175	69426,402		
	Segundos:	39,99	38,27	36,67	71,09	45,66	42,35	69,43		
Dijkstra	1ª	37292,013	30887,516	39196,617	38801,541	48386,013	40065,563	49374,879	307599,634	301645,162
	2ª	38986,666	33288,999	39287,383	66988,066	49192,011	42649,143	48435,161	333364,918	294561,568
	3ª	36399,005	32979,433	34002,332	60262,285	52838,443	41780,258	66022,086	329519,556	289476,158
	Media:	37559,228	32385,316	37495,444	55350,631	50138,822	41498,321	54610,709	323494,703	295227,629
	Segundos:	37,56	32,39	37,50	55,35	50,14	41,50	54,61	323,49	295,23
B. Dijkstra	1ª	37516,705	44642,519	44873,603	67505,208	59452,695	48504,891	62598,668	315944,265	298886,146
	2ª	44852,131	44854,064	44910,709	72677,951	45570,681	48236,813	72326,259	346984,499	306861,116
	3ª	44587,691	44468,964	45075,401	61573,206	59020,083	48626,779	69966,771	334498,452	318516,154
	Media:	42318,842	44655,182	44953,238	67252,122	54681,153	48456,161	68297,233	332475,739	308087,805
	Segundos:	42,32	44,66	44,95	67,25	54,68	48,46	68,30	332,48	308,09

Tabla 5.7: Resultados de BufferLineString (2/2)

A simple vista se puede apreciar como se logra obtener respuesta para algún caso de prueba más, tanto del algoritmo A* como del algoritmo A* bidireccional. En concreto, se llega a obtener resultado para vías de ciudades distintas del mismo país, así como para el caso de prueba de dos ciudades que se encuentran a ambos lados de una frontera nacional. Aunque aún no se consigue obtener respuesta para las rutas de mayor distancia, no deja de ser un resultado que indica que nos movemos por el buen camino.

Advertir que, para definir este área de influencia es necesario proporcionar la distancia o radio respecto del origen. Este es un aspecto que se debe tener en cuenta, puesto que el destino no es considerado a la hora de delimitar el buffer. Lo que puede dar lugar a que se proporcione un radio que no llegue a englobar el destino, provocando que pgRouting no pueda calcular la ruta y devuelva el siguiente mensaje de error que indica que el destino no fue encontrado:

```
ERROR: target not found
```

Por lo que respecta a los tiempos de respuesta obtenidos, puede apreciarse una importante mejora respecto a los tiempos obtenidos en la propuesta de los desarrolladores. Ahora, las versiones A* y las versiones Dijkstra se mueven en el mismo rango de valores. Los algoritmos se mantienen entre los 29,82 a los 71,09 segundos decimales, valores muy inferiores a los que se obtenían en la propuesta (de 119,64 a los 152,85). La ruta óptima sobre la cartografía europea es calculada en torno a los 30 s - 1 m 11 s, mientras que con la propuesta de los desarrolladores rondaba los 2 m - 2 m 33 s.

Eso respecto a las pruebas en las que todos los algoritmos retornan una ruta. Para los casos de prueba basados en grandes distancias, como es el caso de Valladolid-Varsovia (prueba 42), sólo las versiones Dijkstra emiten resultado. Suponiendo el tiempo de respuesta un máximo respecto al resto de casos, como puede observarse en la gráfica. El tiempo de ejecución para las últimas dos pruebas se corresponde con valores en torno a los 295,23 y los 332,48 segundos decimales, lo que es lo mismo, 4 m 55 s y 5 m 33 s. Por lo que es evidente que en estos dos casos ha empeorado el tiempo de respuesta respecto a la propuesta de los desarrolladores. Esto es debido a que el área de influencia creado desde el origen, debe ser tan grande para abordar el destino, que al final se cargan en memoria prácticamente todos los registros de la cartografía.

5.3.3. BufferMultiLineString

Analizados los resultados obtenidos para el área de influencia BufferLineString, surgió la idea para el área BufferMultiLineString.

```
with buffered as (select * from eu_2po_4pgr where geom_way && (select ST_Buffer(
ST_Collect(ARRAY[(select ST_AsText(geom_way) from eu_2po_4pgr where id=10197465),(select
ST_AsText(geom_way) from eu_2po_4pgr where id=10197466)]),radius))) select id, source,
target, cost from buffered
```

Su definición es algo más compleja que la de los anteriores casos, de hecho el número de funciones PostGIS que se utilizan en su definición es algo mayor. Comencemos por explicar la función individual de cada una de las funciones PostGIS empleadas, para posteriormente analizar el resultado de su combinación.

- ST_Buffer y &&

  Nuevamente recurrimos a su uso y, como ya se ha comentado, la función ST_Buffer devuelve una geometría que representa todos los puntos cuya distancia de la geometría dada es menor o igual a la distancia. Y el operador && devuelve la intersección de dos geometrías.

- ST_Collect

```
geometry ST_Collect(geometry[] g1_array);
```

Recibe una colección de geometrías, pero el valor de retorno es una única geometría. Se entiende fácilmente con un ejemplo, supongamos que recibe dos geometrías de tipo Point, pues bien, la salida es una única geometría de tipo MultiPoint. Fundamentalmente une geometrías.

Colección de geometrías	Geometría resultante
POINT(-2 3), POINT(-2 3)	MULTIPOINT(1 2,-2 3)

Tabla 5.8: Ejemplo de la función ST_Collect.

- ARRAY

```
geometry[]
```

El tipo de array que conocemos pero de tipo geometría. Este operador simplemente sirve para generar un array que recoja la colección de geometrías a pasar como argumento a la función ST_Collect o cualquier otra que reciba un array de geometrías.

- ST_AsText

```
text ST_AsText(geometry g1);
```

Devuelve la representación WKT (Well-Known Text) de la geometría.

```
select ST_AsText('0103000000010000000500000000000000000000000000000000000000000000000000
00000000000000000000F03F000000000000F03F000000000000F03F000000000000F03F0000000000
0000000000000000000000000000000000000000');

                    st_astext
--------------------------------
 POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)
```

Bien en cuanto a la consulta general, con la combinación de estas funciones lo que se logra es: dados origen y destino (en el ejemplo: 8324560, 17383662) generar un array que contenga su correspondiente representación WKT, proceso que se realiza combinando la función ST_AsText y ARRAY. A continuación, dicho array de geometrías se pasa a la función ST_Collect que devuelve una geometría de tipo MultiLineString (pues recibe dos geometrías de tipo LineString). De este modo hemos creado una geometría que contiene ambas vías.

Ahora sobre la misma podemos aplicar la función ST_Buffer para obtener un área de influencia según la distancia especificada. Tras lo cual dicha área se combina con toda la geometría europea para obtener los nodos comprendidos en el área. Así se obtiene el buffer temporal que servirá para aplicar el algoritmo de pgRouting deseado.

Esta ultima parte es idéntica a la definida en BufferLineString pero, como se habrá podido observar, en este caso se tiene en cuenta tanto la vía de origen como la de destino. Creando una nueva geometría sobre la que aplicar el buffer.

```
select seq, id1 AS node, id2 AS edge, cost from pgr_dijksta('with buffered as (select *
from eu_2po_4pgr where geom_way && (select ST_Buffer(ST_Collect(ARRAY[(select ST_AsText(
geom_way) from eu_2po_4pgr where id=10197465),(select ST_AsText(geom_way) from
eu_2po_4pgr where id=10197466)]),radius))) select id, source, target, cost from buffered',
8324560, 17383662, false, false);
```

5.3.4. Resultados obtenidos para BufferMultiLineString

La siguiente tabla recoge el tiempo de ejecución de cada uno de los casos de prueba de la cartografía europea. Aparecen las quince pruebas junto con sus resultados para las cuatro funciones pgRouting con las que se trabaja. Para cada una de estas funciones se midió el tiempo de tres ejecuciones y se obtuvo su media, todo ello en la unidad de milisegundos. Dicha media se convierte a segundos decimales para facilitar su observación y evaluación.

Como se puede apreciar, esta vez se logra obtener respuesta para todos los casos de prueba por parte de todos los algoritmos de pgRouting. Al fin se da con la solución del rompecabezas que había puesto en riesgo la funcionalidad del sistema y con ello, la viabilidad de este proyecto.

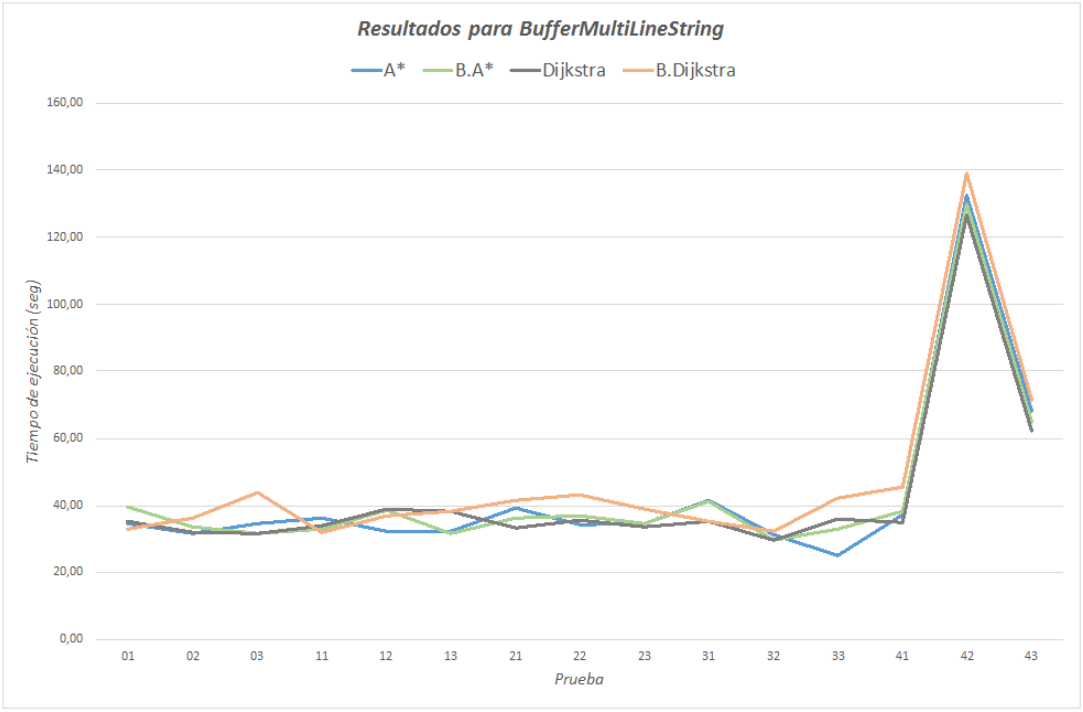


Figura 5.9: Resultados de BufferMultiLineString.

Algoritmo	Prueba	01	02	03	11	12	13	21	22	23
*A**	1ª	34238,055	33134,148	26856,791	36259,933	36575,381	32860,763	39656,469	40193,061	23312,53
	2ª	33560,122	30482,462	37725,528	33527,612	33712,007	40601,711	38259,324	39957,396	40648,653
	3ª	35506,648	31555,568	39168,881	38493,077	26820,913	23347,603	39619,484	22752,811	40007,352
	Media:	34434,942	31724,059	34583,733	36093,541	32369,434	32270,026	39178,426	34301,089	34656,178
	Segundos:	34,43	31,72	34,58	36,09	32,37	32,27	39,18	34,30	34,66
*B.A.**	1ª	39744,743	36892,664	35987,424	24776,356	40139,859	31905,906	39954,057	40024,172	39845,011
	2ª	39732,602	33654,892	23129,786	34804,974	35685,922	24022,848	33733,307	40107,476	23925,591
	3ª	39087,415	30586,573	35708,676	39612,985	39849,452	39299,136	34519,979	30642,469	40034,329
	Media:	39521,587	33711,376	31608,629	33064,772	38558,411	31742,630	36069,114	36924,706	34601,644
	Segundos:	39,52	33,71	31,61	33,06	38,56	31,74	36,07	36,92	34,60
Dijkstra	1ª	38119,937	32561,521	22060,406	37729,965	38925,445	39293,494	38769,476	29849,817	22277,918
	2ª	38113,055	29426,248	38762,385	38012,261	39079,432	39397,466	38702,381	38860,075	39681,185
	3ª	29646,062	33596,997	34163,923	25541,361	38606,612	35888,501	22127,811	38125,587	38665,832
	Media:	35293,018	31861,589	31662,238	33761,196	38870,496	38193,154	33199,889	35611,826	33541,645
	Segundos:	35,29	31,86	31,66	33,76	38,87	38,19	33,20	35,61	33,54
B. Dijkstra	1ª	33631,791	35002,188	43426,289	27106,391	40831,341	44338,808	44424,608	44101,513	29757,543
	2ª	40483,066	32226,096	44235,157	43863,862	44699,493	35276,339	44487,212	45073,066	41805,656
	3ª	24611,588	41645,985	44131,313	25068,636	25082,004	35331,995	35917,534	40160,742	44800,755
	Media:	32908,815	36291,423	43930,920	32012,963	36870,946	38315,714	41609,785	43111,774	38787,985
	Segundos:	32,91	36,29	43,93	32,01	36,87	38,32	41,61	43,11	38,79

Tabla 5.9: Resultados de BufferMultiLineString (1/2)

Algoritmo \ *Prueba*		21	22	23	31	32	33	41	42	43
A*	1ª	39656,469	40193,061	23312,53	41489,363	29416,512	25206,738	41777,196	119913,197	67807,669
	2ª	38259,324	39957,396	40648,653	41838,905	35154,425	25851,861	27710,168	129236,264	68544,672
	3ª	39619,484	22752,811	40007,352	41610,487	28886,048	23720,388	41790,223	148345,659	68574,928
	Media:	39178,426	34301,089	34656,178	41646,252	31152,328	24926,329	37092,529	132498,373	68309,090
	Segundos:	39,18	34,30	34,66	41,65	31,15	24,93	37,09	132,50	68,31
B.A.*	1ª	39954,057	40024,172	39845,011	41213,833	30566,991	40161,178	39684,795	1007389,651	68406,517
	2ª	33733,307	40107,476	23925,591	41247,359	29652,148	22820,668	33622,333	1105216,866	64214,584
	3ª	34519,979	30642,469	40034,329	41154,511	28606,117	35902,173	41072,721	1075749,371	61887,302
	Media:	36069,114	36924,706	34601,644	41205,234	29608,419	32961,340	38126,616	1062785,296	64836,134
	Segundos:	36,07	36,92	34,60	41,21	29,61	32,96	38,13	1062,79	64,84
Dijkstra	1ª	38769,476	29849,817	22277,918	25771,639	39716,444	39959,936	40321,972	117112,297	62752,273
	2ª	38702,381	38860,075	39681,185	39975,895	22627,857	29340,774	23776,926	130355,927	59273,486
	3ª	22127,811	38125,587	38665,832	40279,601	26264,038	38414,912	40382,367	131989,637	64849,778
	Media:	33199,889	35611,826	33541,645	35342,378	29536,113	35905,207	34827,088	126485,954	62291,846
	Segundos:	33,20	35,61	33,54	35,34	29,54	35,91	34,83	126,49	62,29
B. Dijkstra	1ª	44424,608	44101,513	29757,543	35726,359	38615,526	44005,711	44848,919	147167,383	72105,763
	2ª	44487,212	45073,066	41805,656	27069,215	33650,569	37597,971	45806,613	135749,982	72037,742
	3ª	35917,534	40160,742	44800,755	43322,413	24682,018	45254,265	46048,837	134275,987	69950,489
	Media:	41609,785	43111,774	38787,985	35372,662	32316,038	42285,982	45568,123	139064,451	71364,665
	Segundos:	41,61	43,11	38,79	35,37	32,32	42,29	45,57	139,06	71,36

Tabla 5.10: Resultados de BufferMultiLineString (2/2)

Con esta área de influencia no sólo se solventan los problemas encontrados para BufferLineString, además se obtienen los mejores resultados en cuanto a tiempo de ejecución de entre todas las áreas y pruebas realizadas. Lo que confirma que es BufferMultiLineString la definición de área de influencia que se estaba buscando.

Además, para definir este área de influencia es necesario proporcionar la distancia o radio respecto del origen y destino. Este es un aspecto que facilita enormemente el ajuste óptimo del área, dado que por una parte no se va a producir el problema comentado para BufferLineString por no encontrar el destino dentro del buffer, y por otra parte, se ha comprobado que un radio de 0.001 grados o lo que es lo mismo 111 metros, permite obtener respuesta para todas las pruebas, no siendo necesario perder el tiempo buscando el mejor ajuste para cada caso, como ocurría con la anterior definición.

Por lo que respecta a los tiempos de respuesta, puede apreciarse una importante mejora respecto a los tiempos obtenidos en el resto de definiciones. Nuevamente, las versiones A* y las versiones Dijkstra se mueven en el mismo rango de valores, como ya ocurría con BufferLineString. Los algoritmos se mantienen entre los 24,93 a los 139,06 segundos decimales, valores muy inferiores a los que se obtenían en BufferLineString (de 29,82 a los 332,48), al menos en lo que a la cota máxima respecta. La ruta óptima sobre la cartografía europea es calculada en torno a los 25 s - 2 m 19 s, mientras que con la definición BufferLineString rondaba los 30 s - 5 m 33 s.

Para los casos de prueba basados en grandes distancias, como es el caso de Valladolid-Varsovia (prueba 42), el tiempo de respuesta supone un máximo respecto al resto de valores, como puede observarse en la gráfica. Este patrón se repite para todos los algoritmos. Pero es sin duda en este caso donde se produce la mayor mejora de tiempos, con valores en torno a los 126,49 y los 139.06 segundos decimales, respecto a los 295,23 y los 332,48 de BufferLineString. Lo que es lo mismo, 2 m 7 s - 2 m 19 s frente a 4 m 55 s - 5 m 33 s. Pero salvo dicho caso, el resto de tiempos de ejecución se mantienen en valores óptimos que rondan los 35 segundos de respuesta.

Capítulo 6

Evaluación comparativa del rendimiento

Una vez encontrada la definición de área de influencia que solucionaba el problema del uso de memoria por parte de los algoritmos de pgRouting. Se procedió a evaluar con más detalle dicha definición, con el objetivo de comparar su rendimiento y analizar la posibilidad de mejora en los tiempos de respuesta.

Este capítulo recoge la comparativa de rendimiento realizada sobre el área de influencia diseñada BufferMultiLineString. Se compara su rendimiento en la máquina virtual frente a una máquina física. También se analiza la creación de un índice espacial para la mejora de los tiempos de respuesta. Y por último, se compara su rendimiento con la cartografía OpenStreetMap frente a la cartografía TeleAtlas.

Esta evaluación se lleva a cabo utilizando los casos de prueba empleados hasta el momento para la cartografía europea y que fueron recogidos en la sección "3.2. Batería de pruebas y resultados para la cartografía europea" apartado "3.2.2. Casos de prueba" del capítulo 3. Seleccionando el algoritmo de Dijkstra para llevar a cabo dicha evaluación, debido a que es el algoritmo que presenta mejores resultados.

6.1. Fragmentación de la consulta

Antes de abordar la comparativa, es necesario explicar como se estructura el proceso de medida, para facilitar la comprensión de las posteriores tablas e ilustraciones. Este proceso se realizo con el objetivo de obtener una medición lo más fiel y precisa posible, tomando la decisión de dividir la definición de la consulta BufferMultiLineString en tres apartados:

1. Tamaño del espacio de búsqueda / Data recovering
2. Tiempo necesario para recuperar el espacio de búsqueda / Data recovering runtime
3. Tiempo invertido en la aplicación del algoritmo / pgRouting runtime

Evidentemente, los tiempos obtenidos hasta el momento estan constituidos por el tiempo que el sistema toma en crear el área de influencia y el tiempo que pgRouting tarda en aplicar su función sobre dicho área. Estructurar la medición de acuerdo a estos tres parámetros, permite obtener una visión más clara de como evoluciona la consulta según el número de datos analizados.

El siguiente código refleja un ejemplo de la fragmentación de la consulta BufferMultiLineString. Puede observarse una primera medición del tiempo que tarda en generarse el buffer, tras lo cual se recoge una sentencia que crea una tabla temporal en base a dicho resultado. Pero esta última medida es despreciable

en el análisis, dado que en el uso normal del sistema, la creación de una tabla temporal no sera necesaria Sólo se requiere en lo que respecta a la toma de medidas, dado que es necesario aplicar la función de pgRouting sobre dicha tabla temporal para poder obtener el tiempo invertido en la aplicación del algoritmo sobre el buffer.

Finalmente, la penúltima sentencia se encarga de recoger el número de registros que forman el buffer y la última simplemente devuelve la ruta calculada, de modo que se verifique que no hubo problemas durante el procesamiento. Sin olvidar la eliminación de la tabla temporal.

```
\qecho '*************************************************************'
\qecho '| Data recovering runtime'
\qecho '*************************************************************'
\qecho ''

EXPLAIN ANALYZE select * from eu_2po_4pgr where geom_way && (select ST_Buffer(ST
_Collect(ARRAY[(select ST_AsText(geom_way) from eu_2po_4pgr where id=10197465),(
select ST_AsText(geom_way) from eu_2po_4pgr where id=10197466)]),0.001));

\qecho '*************************************************************'
\qecho '| Data recovering and table creation runtime'
\qecho '*************************************************************'
\qecho ''

EXPLAIN ANALYZE create table buffer_tmp as select * from eu_2po_4pgr where geom_
way && (select ST_Buffer(ST_Collect(ARRAY[(select ST_AsText(geom_way) from eu_2p
o_4pgr where id=10197465),(select ST_AsText(geom_way) from eu_2po_4pgr where id=
10197466)]),0.001));;

\qecho '*************************************************************'
\qecho '| pgRouting runtime'
\qecho '*************************************************************'
\qecho ''

EXPLAIN ANALYZE select seq, id1 AS node, id2 AS edge, cost from pgr_dijkstra('se
lect id, source, target, cost from buffer_tmp', 8324560, 17383662, false, false)
;

\qecho '*************************************************************'
\qecho '| Data recovering'
\qecho '*************************************************************'
\qecho ''

select count(*) from eu_2po_4pgr where geom_way && (select ST_Buffer(ST_Collect(
ARRAY[(select ST_AsText(geom_way) from eu_2po_4pgr where id=10197465),(select ST
_AsText(geom_way) from eu_2po_4pgr where id=10197466)]),0.001));
```

```
\qecho '*************************************************************'
\qecho '| Shortest Path'
\qecho '*************************************************************'
\qecho ''

select seq, id1 AS node, id2 AS edge, cost from pgr_dijkstra('select id, source,
 target, cost from buffer_tmp', 8324560, 17383662, false, false);

DROP TABLE buffer_tmp;
```

6.2. Resultados de la fragmentación de BufferMultiLineString

En la siguiente tabla se recogen los resultados obtenidos para el algoritmo de Dijkstra una vez fragmentada la consulta BufferMultiLineString. Aparecen cada uno de los quince casos de prueba de la cartografía europea. Para cada prueba se midió el tiempo de tres ejecuciones y se obtuvo su media, todo ello en la unidad de milisegundos. Dicha media se convierte a segundos decimales para facilitar su observación y evaluación.Hay que percatarse de que estos resultados pertenecen a la ejecución sobre la máquina virtual inicial sobre la que fue instalada la plataforma, lo cual permitirá realizar la comparativa en posteriores apartados.

En primer lugar se recoge el resultado para el tamaño del espacio de búsqueda. Como se puede observar, el número de registros que forman el área de influencia devuelto por BuffeMultiLineString, es reducido en la mayor parte de las pruebas. Siendo nuevamente en las rutas que abarcan el continente europeo donde el número de registros se dispara. En concreto, la prueba 42 (Valladolid-Varsovia) genera un buffer que contiene 16.414.121 registros de los 33.387.548 que recoge la cartografía europea de OpenStreetMap. Es decir, estamos hablando de que casi la mitad de la cartografía es cargada en memoria para poder procesar dicha consulta.

Este hecho tiene una repercusión no demasiado notable en el tiempo necesario para recuperar el espacio de búsqueda (Data recovering runtime). Puesto que salvo una ligera variación para las grandes rutas, por lo general los tiempos oscilan en un rango de valores concreto, como puede observarse en la posterior gráfica. Traza que no se mantiene en el caso del tiempo invertido en la aplicación del algoritmo (pgRouting runtime). Puede observarse claramente la relación que este tiempo guarda respecto al número de registros procesados. La función de pgRouting aumenta su tiempo de ejecución conforme aumenta el número de registros del área de influencia.

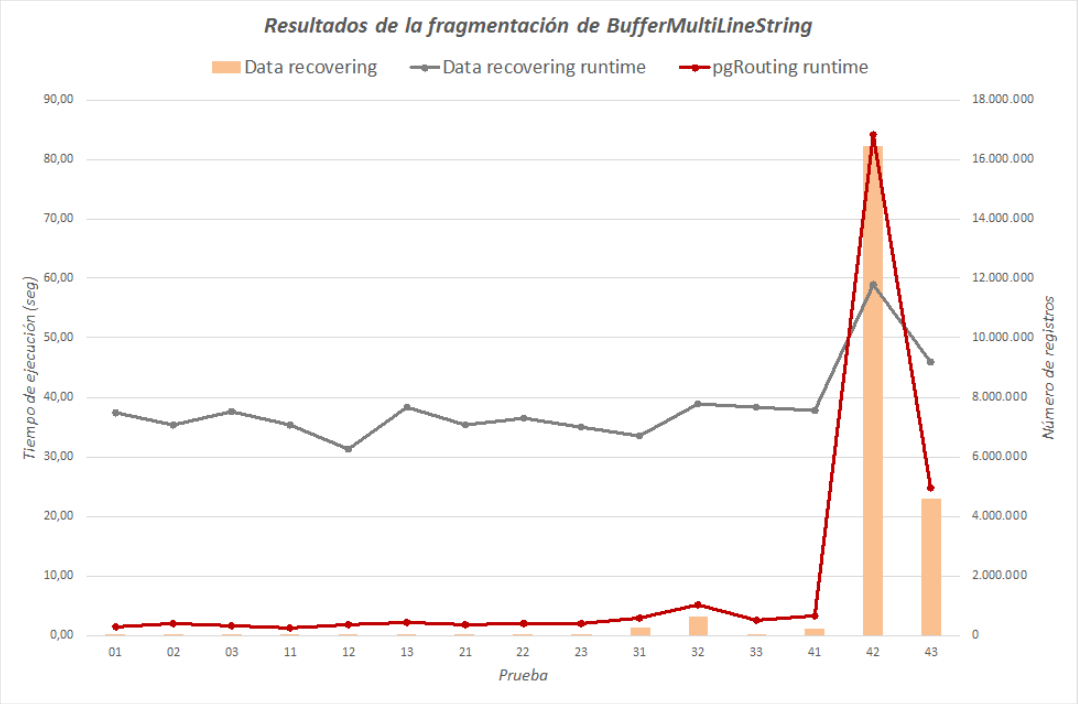


Figura 6.1: Resultados de la fragmentación de BufferMultiLineString.

El tiempo total de ejecución es el resultado de combinar ambas mediciones, la suma del tiempo necesario para recuperar el espacio de búsqueda y el tiempo invertido en aplicar el algoritmo. Lo que provoca que el tiempo de respuesta final este condicionado al tiempo que el sistema consume en crear el buffer. Porque como se ha podido ver, la función de pgRouting arroja teimpos de respuesta por debajo de los 3 segundos para la mayoria de las pruebas.

Algoritmo Dijkstra \ *Prueba*		01	02	03	11	12	13	21	22	23
Data recovering		36	32	7	61	34	17	920	148	184
Data recovering runtime	1ª	36696,022	34115,826	38600,977	33994,734	20946,457	38559,709	32126,707	38435,122	38514,752
	2ª	38334,804	39225,067	35440,467	37381,748	36412,846	38624,55	37567,655	35894,665	28924,998
	3ª	37345,622	32574,106	38875,014	34713,631	36371,406	38145,079	36689,486	35245,916	37396,614
	Media:	37458,816	35305,000	37638,819	35363,371	31243,570	38443,113	35461,283	36525,234	34945,455
	Segundos:	37,46	35,30	37,64	35,36	31,24	38,44	35,46	36,53	34,95
pgRouting runtime	1ª	1357,986	1938,364	1665,025	1177,939	1914,528	2114,192	1738,624	2287,501	2144,02
	2ª	1292,153	1958,936	1695,228	1328,806	1928,949	2106,781	1704,84	1783,072	2255,29
	3ª	1399,017	1880,924	1655,27	1167,333	1736,841	2099,921	1784,861	1661,469	1765,247
	Media:	1349,719	1926,075	1671,841	1224,693	1860,106	2106,965	1742,775	1910,681	2054,852
	Segundos:	1,35	1,93	1,67	1,22	1,86	2,11	1,74	1,91	2,05
	Total:	38,81	37,23	39,31	36,59	33,10	40,55	37,20	38,44	37,00

Tabla 6.1: Resultados de BufferMultiLineString en máquina virtual (1/2)

Algoritmo Dijkstra \ Prueba		21	22	23	31	32	33	41	42	43
Data recovering		920	148	184	261.889	637.229	28.613	224.048	16.414.121	4.587.321
Data recovering runtime	1ª	32126,707	38435,122	38514,752	24686,044	39646,083	38476,701	37053,392	55186,145	46384,088
	2ª	37567,655	35894,665	28924,998	38066,233	37303,035	38088,823	38199,097	61655,735	45798,202
	3ª	36689,486	35245,916	37396,614	37970,693	39842,712	38442,855	37950,312	60186,258	45809,191
	Media:	35461,283	36525,234	34945,455	33574,323	38930,610	38336,126	37734,267	59009,379	45997,160
	Segundos:	35,46	36,53	34,95	33,57	38,93	38,34	37,73	59,01	46,00
pgRouting runtime	1ª	1738,624	2287,501	2144,02	2436,771	5291,169	2734,088	3261,564	85672,579	25545,404
	2ª	1704,84	1783,072	2255,29	3189,734	5128,257	2809,214	3155,126	83581,977	23837,158
	3ª	1784,861	1661,469	1765,247	3175,981	5091,492	2352,753	3209,657	91508,064	25006,83
	Media:	1742,775	1910,681	2054,852	2934,162	5170,306	2632,018	3208,782	84278,844	24796,464
	Segundos:	1,74	1,91	2,05	2,93	5,17	2,63	3,21	84,28	24,80
	Total:	37,20	38,44	37,00	36,51	44,10	40,97	40,94	143,29	70,79

Tabla 6.2: Resultados de BufferMultiLineString en máquina virtual (2/2)

6.3. Comparativa entre máquina virtual y física

Una de las evaluaciones realizadas durante el análisis del rendimiento, fue comparar los resultados que BufferMultiLineString ofrecía en una máquina física respecto a los datos obtenidos hasta el momento en la máquina virtual.

Fue necesario repetir todo el proceso de instalación de la plataforma, junto con la carga cartográfica para poder llevar a cabo dicha tarea sobre la máquina física. Siendo necesario ajustar los parámetros a la configuración de la máquina física disponible para realizar las pruebas. Esta máquina pertenece al grupo de investigación Trasgo, denominada Frontend, en la cual los miembros del grupo trabajan a diario.

Las características de esta máquina física son:

- 12 cores Intel Xeon E5-2620 a 2.4GHz
- 8Gb de RAM
- Red Hat 4.2.2-16

La tabla de este apartado contiene los resultados obtenidos para el algoritmo de Dijkstra sobre la máquina física utilizando BufferMultiLineString. Aparecen cada uno de los quince casos de prueba de la cartografía europea. Para cada prueba se midió el tiempo de tres ejecuciones y se obtuvo su media, todo ello en la unidad de milisegundos. Dicha media se convierte a segundos decimales para facilitar su observación y evaluación.

En las gráficas posteriores se puede percibir las principales diferencias entre la ejecución sobre la máquina virtual y la física. El aspecto más destacable, que evidencia la primera de las gráficas, es la diferencia notable entre el tiempo necesario para recuperar el espacio de búsqueda (Data recovering runtime) entre máquina virtual y física. El tiempo que Frontend necesita para generar el área de influencia es mucho mayor que el requerido por la máquina física. Esto puede ser debido principalmente a que la máquina virtual cuenta con tamaños de memoria mayores y, sobretodo, a que es una máquina dedicada íntegramente a este proyecto, en la cual no se realizan más tareas. Mientras que Frontend, es una máquina compartida por varios usuarios, ejecutando múltiples tareas a la vez, con el consiguiente consumo de recursos. Hecho por el cual, el tiempo requerido para generar el buffer es mayor que en la máquina virtual.

En cuanto al tiempo invertido en la aplicación del algoritmo, recogido también en la primera gráfica. Tanto los resultados de la máquina virtual como los de la física, siguen el mismo patrón de tendencia. Muestran tiempos de respuesta mínimos para la mayor parte de las consultas, salvo en los casos de prueba que definen rutas que prácticamente atraviesan el continente.

Al unir ambos tiempos, el necesario para recuperar el espacio de búsqueda y el invertido en aplicar el algoritmo, se obtiene el tiempo de respuesta total de la consulta. Se observa que, una vez más, el tiempo total es condicionado por el tiempo de generación del buffer. Al presentar Frontend tiempos de respuesta mayores para esta medida, el resultado final es también un tiempo de respuesta total mayor que el obtenido para la máquina física.

Pero recalquemos una vez más, que Frontend es una máquina física con una memoria limitada donde varios usuarios trabajan simultáneamente. Por lo que no se puede considerar que la plataforma no ofrezca

un buen resultado sobre una máquina física, porque seguramente si la máquina esta destinada únicamente a procesar consultas de enrutado, estos resultados mejoren.

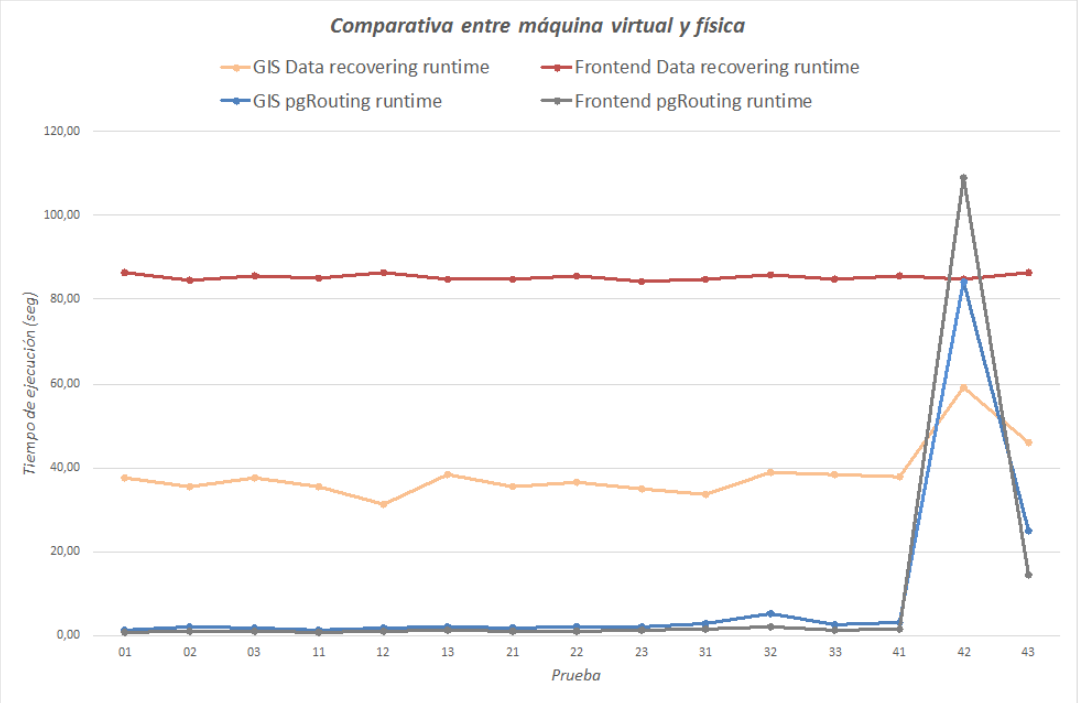


Figura 6.2: Comparativa entre máquina virtual y física.

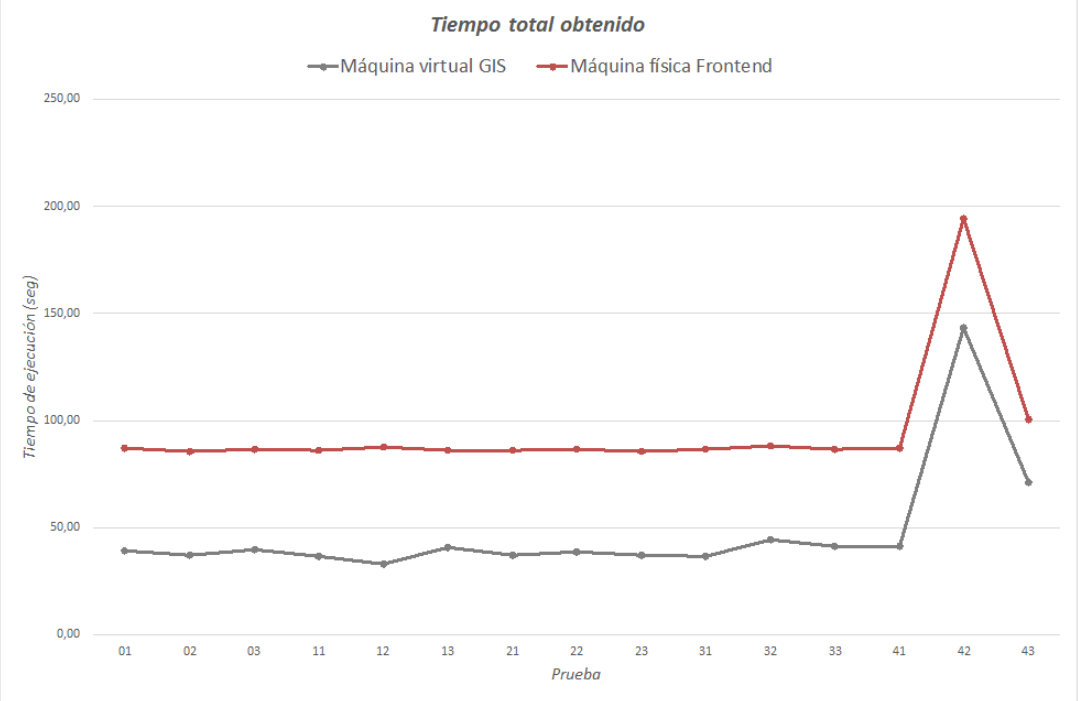


Figura 6.3: Comparativa del tiempo total obtenido.

Algoritmo Dijkstra \ Prueba		01	02	03	11	12	13	21	22	23
Data recovering		36	32	7	61	34	17	920	148	184
Data recovering runtime	1ª	89476,85	85059,326	86446,393	85120,664	85226,279	84298,781	84389,289	86440,49	85150,156
	2ª	84316,946	84712,678	85474,248	84425,855	84734,077	84676,99	85106,609	84448,213	83923,855
	3ª	85546,951	84301,054	84903,43	85683,607	89446,529	85319,966	85103,721	85722,342	83917,934
	Media:	86446,916	84691,019	85608,024	85076,709	86468,962	84765,246	84866,540	85537,015	84330,648
	Segundos:	86,45	84,69	85,61	85,08	86,47	84,77	84,87	85,54	84,33
pgRouting runtime	1ª	687,505	973,206	867,237	713,797	1049,161	1233,576	1078,799	1029,844	1276,688
	2ª	609,902	941,043	918,136	727,067	1061,777	1174,676	995,164	1053,972	1273,692
	3ª	642,042	925,741	843,622	619,916	1110,153	1156,298	947,747	1034,127	1315,851
	Media:	646,483	946,663	876,332	686,927	1073,697	1188,183	1007,237	1039,314	1288,744
	Segundos:	0,65	0,95	0,88	0,69	1,07	1,19	1,01	1,04	1,29
	Total:	87,09	85,64	86,48	85,76	87,54	85,95	85,87	86,58	85,62

Tabla 6.3: Resultados de BufferMultiLineString en máquina física (1/2)

Algoritmo Dijkstra \ *Prueba*		21	22	23	31	32	33	41	42	43
Data recovering		920	148	184	261.889	637.229	28.613	224.048	16.414.121	4.587.321
Data recovering runtime	1ª	84389,289	86440,49	85150,156	85356,455	86189,701	84916,052	86083,113	84726,531	85958,814
	2ª	85106,609	84448,213	83923,855	85009,19	86335,645	84571,079	84808,903	85015,177	85014,608
	3ª	85103,721	85722,342	83917,934	84001,342	85378,62	85421,021	85872,655	84715,384	88051,169
	Media:	84866,540	85537,015	84330,648	84788,996	85967,989	84969,384	85588,224	84819,031	86341,530
	Segundos:	84,87	85,54	84,33	84,79	85,97	84,97	85,59	84,82	86,34
pgRouting runtime	1ª	1078,799	1029,844	1276,688	1432,12	2199,852	1393,885	1540,381	108514,524	14065,953
	2ª	995,164	1053,972	1273,692	1512,895	2120,971	1314,05	1380,17	109440,427	14310,768
	3ª	947,747	1034,127	1315,851	1565,6	2047,018	1257,156	1506,398	107370,86	14533,839
	Media:	1007,237	1039,314	1288,744	1503,538	2122,614	1321,697	1475,650	109131,793	14303,520
	Segundos:	1,01	1,04	1,29	1,50	2,12	1,32	1,48	109,13	14,30
	Total:	85,87	86,58	85,62	86,29	88,09	86,29	87,06	193,95	100,65

Tabla 6.4: Resultados de BufferMultiLineString en máquina física (2/2)

6.4. Comparativa sobre indexación espacial

PostgreSQL implementa un algoritmo de indexación espacial denominado GiST (Generalized Search Tree). GiST en PostgreSQL se puede extender para ser utilizado con geometrías o tipos personalizados. De esta forma, PostGIS extiende los índices GiST de PostgreSQL para que funcionen de forma adecuada con el tipo geometry y con el tipo geography.

Si los índices no espaciales sobre una columna de una tabla son extremadamente importantes para acelerar las consultas alfanuméricas, los indices espaciales aún lo son más al indexar los datos en dos o tres dimensiones. Crear un índice espacial puede requerir bastante tiempo en tablas muy grandes, aunque solo será necesario realizarlo una vez, ya que PostgreSQL actualiza el índice cuando es necesario.

Resumiendo, se puede decir que cuando los datos cartográficos exceden de algunas miles de filas, se debe crear índices espaciales para incrementar las velocidades de búsqueda espacial. Así que se procedió a incluir un índice espacial sobre la columna de tipo geometry (geom_way o geom) de la cartografía OSM importada.

La tabla de este apartado contiene los resultados obtenidos para el algoritmo de Dijkstra utilizando BufferMultiLineString con indexación espacial. Aparecen cada uno de los quince casos de prueba de la cartografía europea. Para cada prueba se midió el tiempo de tres ejecuciones y se obtuvo su media, todo ello en la unidad de milisegundos. Dicha media se convierte a segundos decimales para facilitar su observación y evaluación.

Hasta el momento, en los anteriores apartados, se ha comentado que el tiempo de ejecución total obtenido se ve afectado principalmente por el tiempo invertido en la generación del buffer. Con la creación del índice espacial comentado, lo que se pretendía era reducir este tiempo, de modo que fuera posible apreciar la mejora en el tiempo total de ejecución. Y precisamente este fue el resultado que se consigo.

En la primera gráfica, puede visualizarse como el tiempo invertido en la aplicación del algoritmo sigue sin verse afectado por otra referencia que no sea el número de registros procesados, como ya se comento al analizar la fragmentación de BufferMultiLineString. Pero esta vez, el tiempo necesario para recuperar el espacio de búsqueda se ve notablemente reducido, tanto que se acerca a los valores obtenidos en la aplicación del algoritmo y seguir su tendencia.

Como resultado, la suma de ambas mediciones nos lleva a obtener la segunda gráfica. Es evidente la mejora de los tiempos de respuesta gracias a la indexación espacial. Ahora el global de la consulta ronda los 3 segundos, salvo en las rutas de mayor distancia. Cuando antes los tiempos de respuesta rodaban, por lo general, los 40 segundos. Una mejora que afianza sin duda la solución planteada como una alternativa viable en el cálculo de rutas ópticas.

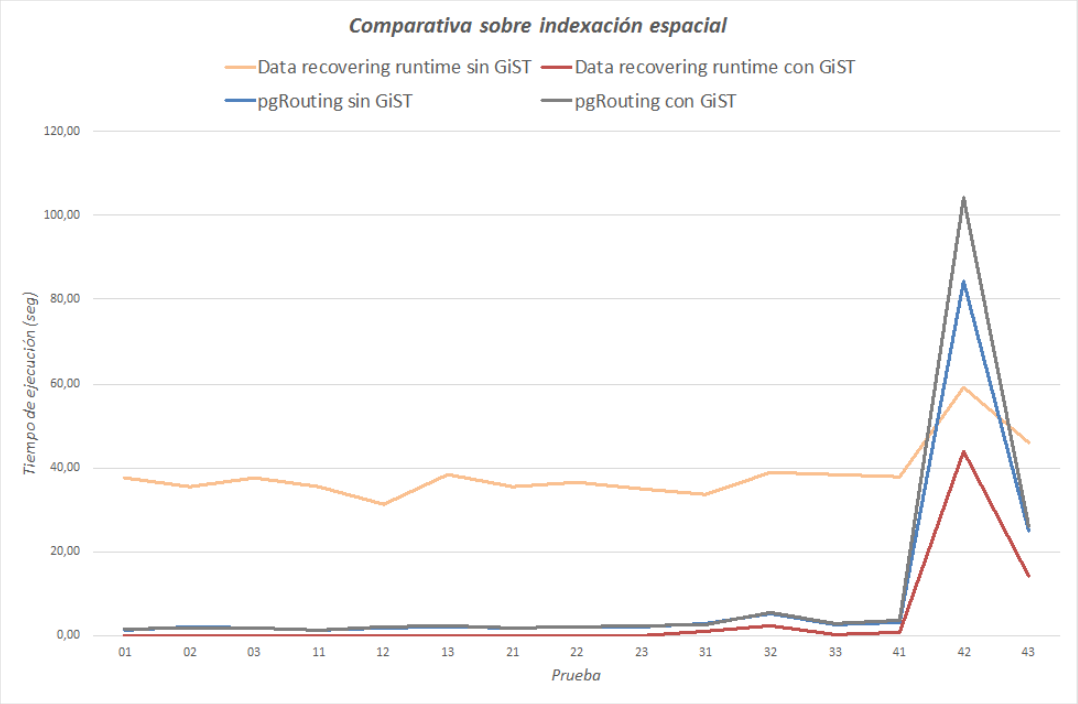


Figura 6.4: Comparativa sobre indexación espacial.

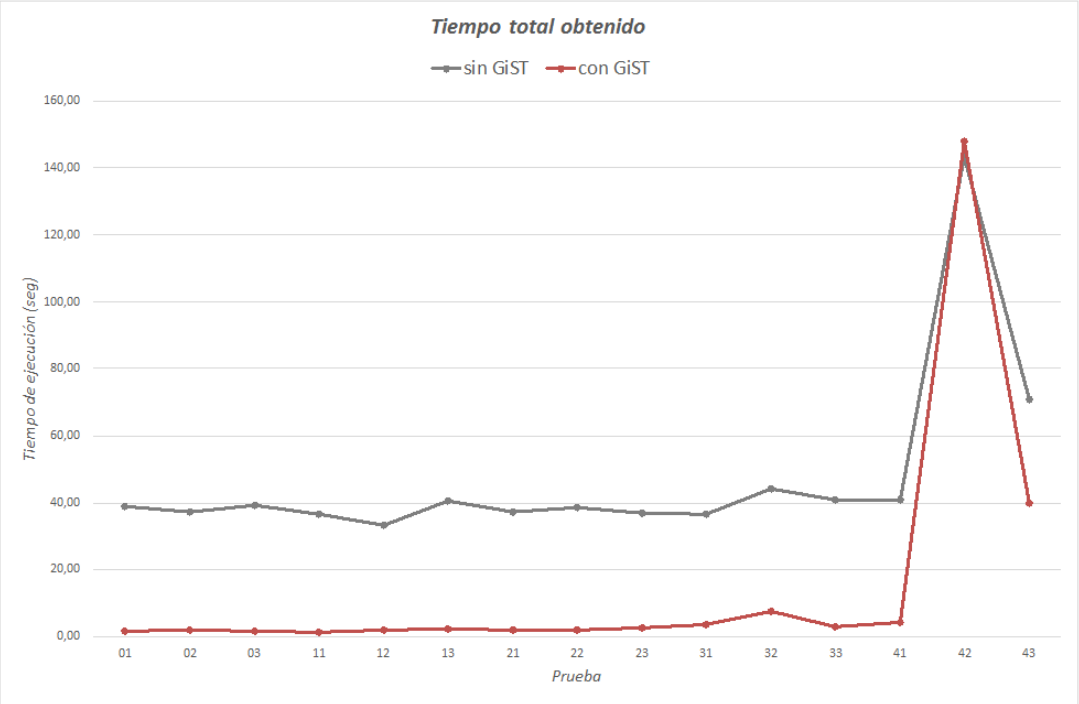


Figura 6.5: Tiempo total obtenido con indexación espacial.

Algoritmo Dijkstra \ *Prueba*		01	02	03	11	12	13	21	22	23
Data recovering		36	32	7	61	34	17	920	148	184
Data recovering runtime	1ª	4,475	3,71	4,466	3,607	2,448	3,629	7,688	3,947	5,016
	2ª	4,39	4,088	4,492	3,669	2,94	4,254	4,566	2,181	3,125
	3ª	5,328	4,879	4,445	3,345	2,81	4,243	7,431	3,783	6,342
	Media:	4,731	4,226	4,468	3,540	2,733	4,042	6,562	3,304	4,789
	Segundos:	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,00	0,00
pgRouting runtime	1ª	1496,331	1940,679	1374,97	1273,084	1956,805	2492,541	1968,912	1997,728	2579,56
	2ª	1410,352	1926,915	1834,045	1248,529	2024,334	1868,742	1438,723	1982,867	1924,952
	3ª	1445,285	1836,266	1828,084	1169,296	1902,056	2458,925	1937,303	1962,657	2697,097
	Media:	1450,656	1901,287	1679,033	1230,303	1961,065	2273,403	1781,646	1981,084	2400,536
	Segundos:	1,45	1,90	1,68	1,23	1,96	2,27	1,78	1,98	2,40
	Total:	1,46	1,91	1,68	1,23	1,96	2,28	1,79	1,98	2,41

Tabla 6.5: Resultados de utilizar indexación espacial con BufferMultiLineString (1/2)

Algoritmo Dijkstra \ *Prueba*		21	22	23	31	32	33	41	42	43
Data recovering		920	148	184	261.889	637.229	28.613	224.048	16.414.121	4.587.321
Data recovering runtime	1ª	7,688	3,947	5,016	856,881	1994,695	240,441	681,388	40028,736	15292,782
	2ª	4,566	2,181	3,125	1060,12	2317,97	180,482	684,371	42088,501	11521,975
	3ª	7,431	3,783	6,342	861,608	2275,019	209,618	755,389	49379,191	15373,451
	Media:	6,562	3,304	4,789	926,203	2195,895	210,180	707,049	43832,143	14062,736
	Segundos:	0,01	0,00	0,00	0,93	2,20	0,21	0,71	43,83	14,06
pgRouting runtime	1ª	1968,912	1997,728	2579,56	2215,078	5757,339	2780,836	3080,075	98232,133	25525,848
	2ª	1438,723	1982,867	1924,952	3100,245	4993,631	2803,26	4018,762	107162,271	27672,824
	3ª	1937,303	1962,657	2697,097	2614,836	5443,849	2909,305	3400,222	101779,697	24535,938
	Media:	1781,646	1981,084	2400,536	2643,386	5398,273	2831,134	3499,686	104185,558	25911,537
	Segundos:	1,78	1,98	2,40	2,64	5,40	2,83	3,50	104,19	25,91
	Total:	1,79	1,98	2,41	3,57	7,59	3,04	4,21	148,02	39,97

Tabla 6.6: Resultados de utilizar indexación espacial con BufferMultiLineString (2/2)

6.5. Comparativa entre cartografía OSM y TeleAtlas

Para realizar la siguiente comparativa, se utilizo la cartografía española tanto de OpenStreetMap como de TeleAtlas. No se utilizo para esta tarea la cartografía europea porque el proceso de importación de dicha cartografía para el caso de TeleAtlas, supone un trabajo pesado y demasiado minucioso que requiere de un intenso procesamiento previo, por no hablar del coste en espacio de almacenamiento que suponía y del que no se disponía.

Se selecciono una nueva batería de pruebas para realizar la comparativa, la cual se recoge al final de este apartado. Se seleccionaron de acuerdo al patrón seguido hasta el momento: mismo nodo, nodos adyacentes, nodos distantes de una misma ciudad y nodos de ciudades distintas del mismo país, es decir, España. La tabla que se recoge en este apartado muestra el tiempo de ejecución de cada uno de estos casos de prueba. Aparecen las seis pruebas junto con sus resultados para ambas cartografías. Para cada una se midió el tiempo de tres ejecuciones y se obtuvo su media, todo ello en la unidad de milisegundos. Dicha media se convierte a segundos decimales para facilitar su observación y evaluación.

En la realización de esta comparativa se volvió a tener en cuenta el tamaño del espacio de búsqueda. Ya que si anteriormente se había mantenido invariable al tratarse de la misma cartografía y las mismas pruebas, ahora se esperaba que el número de registros devueltos por TeleAtlas variase respecto al resultado de OSM. Y así fue. En el gráfico de barras incluido en la primera figura, el número de registros que OSM devuelve en la creación del buffer para cada prueba es mínimo, no superando los 285.816 registros para la ruta más larga. Por el contrario, en las rutas largas TeleAtlas se dispara. Devuelve un número de registros muy elevado respecto al resultado de OSM, estamos hablando de que para la misma ruta TeleAtlas retorna 5.445.405 registros, es decir, 5.159.589 de registros más que OSM. Esto se debe a que el nivel de detalle que ofrece la cartografía de TeleAtlas es mucho más preciso. No debemos obviar que se esta comparando una cartografía comercial con una cartográfica libre que se genera a partir de contribuciones de usuarios, por lo que hay vías que aún no se encuentran registradas. Luego el almacén de información de TeleAtlas, siendo una de las principales fuentes de cartografía del mundo, es superior al de OSM. Basta con observar el tamaño de la base de datos que almacena cada cartografía, así como el número de registros que contiene:

DB Spain	Size	Rows
spain (OSM)	1078 MB	2.719.088
spain (TeleAtlas)	15 GB	7.237.315

DB Europe	Size	Rows
osm2po (OSM)	11 GB	33.387.548

Se comprobó que TeleAtlas hace una división mayor que OSM para una misma vía y proporciona mayor información que OSM, por lo que sería necesario realizar un intenso procesamiento previo para eliminar la información inservible de TeleAtlas. Y como el tamaño del buffer condiciona el tiempo invertido en la aplicación del algoritmo de pgRouting, es evidente que los tiempos de ejecución de TeleAtlas aumentan en comparación a los de OSM. Al igual que el tiempo de creación del buffer al contar con una mayor número de registros.

Teniendo en cuenta todos estos detalles, es normal que los resultados obtenidos arrojen un peor resultado de la cartografía de TeleAtlas respecto de OpenStreetMap. En la segunda gráfica donde se recoge el tiempo de ejecución total de la consulta, queda claro que el tiempo de respuesta con TeleAtlas se dispara en consultas largas en comparación con OpenStreetMap. Habría que reducir la información ofrecida por TeleAtlas para su base de datos, pero con sumo cuidado de no afectar a parámetros que intervengan en la aplicación del algoritmo, de modo que se compruebe si estos tiempos pueden ser mejorados reduciendo el nivel de detalle de TeleAtlas.

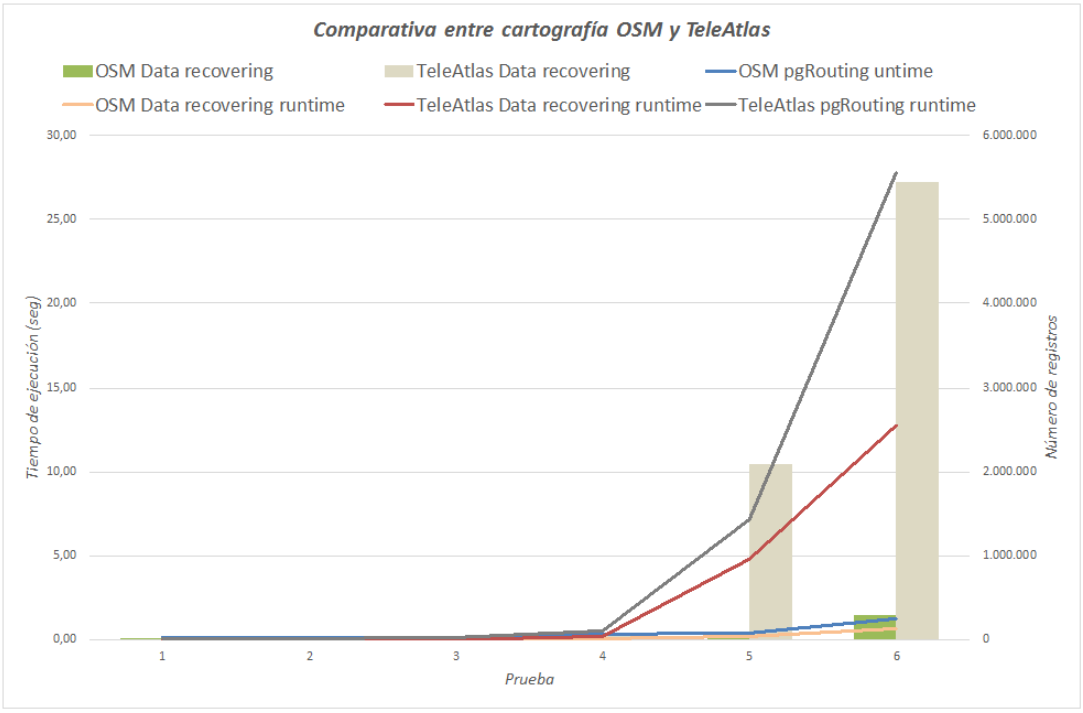


Figura 6.6: Comparativa entre cartografía OSM Y TeleAtlas.

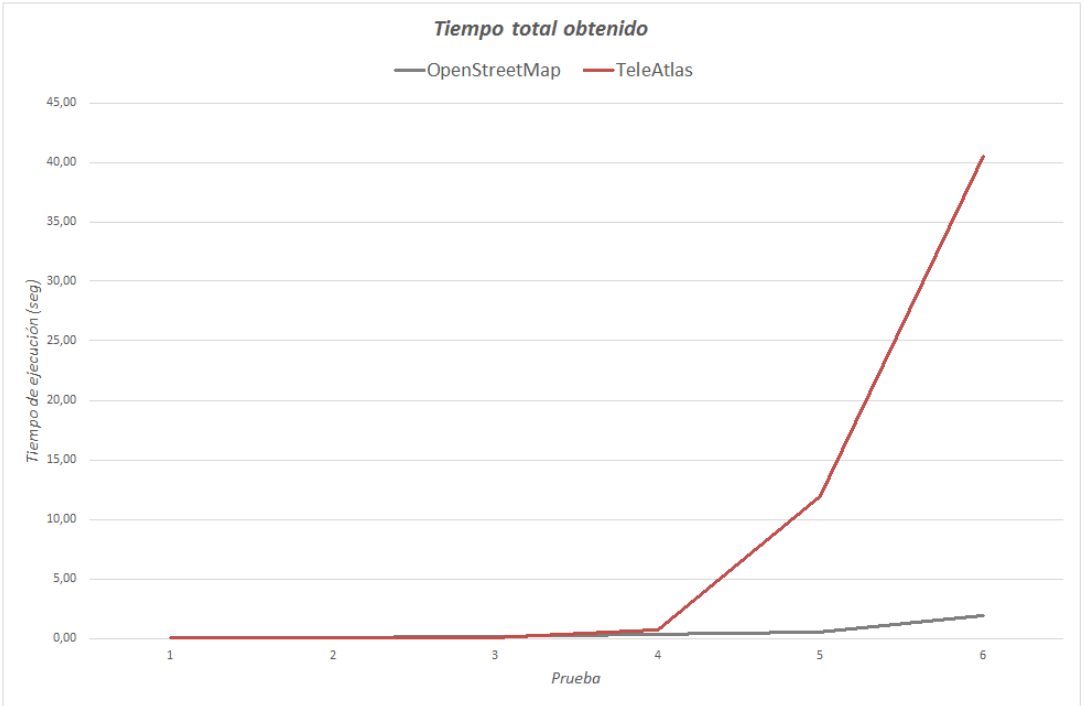


Figura 6.7: Comparativa del tiempo total obtenido con cada cartografía.

Algoritmo Dijkstra \ *Prueba*		1		2		3		4		5		6	
Cartografía		OSM	TeleAtlas	OSM	TeleAtlas	OSM	TeleAtlas	OSM	TeleAtlas	OSM	TeleAtlas	OSM	TeleAtlas
Data recovering		16	24	23	24	76	213	3.073	73.621	56.693	2.091.275	285.816	5.445.405
Data recovering *runtime*	1ª	4,329	2,495	3,627	3,056	3,336	3,942	16,282	142,035	153,275	5523,792	736,042	13350,367
	2ª	3,948	2,884	2,89	2,515	3,298	3,914	16,682	254,312	150,514	3507,707	726,919	13951,28
	3ª	4,007	2,628	2,927	2,588	3,336	3,901	16,286	146,043	150,381	5341,377	404,32	10830,057
	Media:	4,095	2,669	3,148	2,720	3,323	3,919	16,417	180,797	151,390	4790,959	622,427	12710,568
	Segundos:	0,000	0,00	0,00	0,00	0,00	0,00	0,02	0,18	0,15	4,79	0,62	12,71
pgRouting *runtime*	1ª	103,354	67,495	96,347	92,147	122,252	108,498	295,296	418,92	370,999	7149,066	1283,438	25745,602
	2ª	103,092	67,199	109,967	76,478	124,024	95,07	364,946	693,949	369,8	6762,377	1198,956	27537,074
	3ª	101,808	66,303	97,452	65,76	139,185	81,357	316,781	446,391	373,516	7482,977	1301,321	30122,888
	Media:	102,751	66,999	101,255	78,128	128,487	94,975	325,674	519,753	371,438	7131,473	1261,238	27801,855
	Segundos:	0,10	0,07	0,10	0,08	0,13	0,09	0,33	0,52	0,37	7,13	1,26	27,80
	Total:	0,11	0,07	0,10	0,08	0,13	0,10	0,34	0,70	0,52	11,92	1,88	40,51

6.5.1. Batería de pruebas para la comparativa OSM y TeleAtlas

Prueba 1: Usando como "sourcez "target.[el] mismo nodo.

OSM					
id	**osm_id**	**osm_source_id**	**osm_target_id**	**source**	**target**
1975135	192505339	864117527	864117754	1434779	1434780
España, Palencia, vía: 'Calle Prado de la Lana'					

Tabla 6.8: Prueba 1 - OSM.

TeleAtlas					
gid	**id**	**f_jnctid**	**t_jnctid**	**source**	**target**
3987557	17240026643026	17240203205304	17240203206510	3266762	3266763
España, Palencia, vía: 'Calle Prado de la Lana'					

Tabla 6.9: Prueba 1 - TeleAtlas.

Prueba 2: Usando como "sourcez "target"dos nodos consecutivos de una misma vía.

OSM					
id	**osm_id**	**osm_source_id**	**osm_target_id**	**source**	**target**
1975135	192505339	864117527	864117754	1434779	1434780
España, Palencia, vía: 'Calle Prado de la Lana'					
1975136	192505339	864117754	864117770	1434780	1434781
España, Palencia, vía: 'Calle Prado de la Lana'					

Tabla 6.10: Prueba 2 - OSM.

TeleAtlas					
gid	**id**	**f_jnctid**	**t_jnctid**	**source**	**target**
3987557	17240026643026	17240203205304	17240203206510	3266762	3266763
España, Palencia, vía: 'Calle Prado de la Lana'					
3987556	17240026643023	17240203205449	17240203205304	3266760	3266762
España, Palencia, vía: 'Calle Prado de la Lana'					

Tabla 6.11: Prueba 2 - TeleAtlas.

Prueba 3: Usando como "sourcez "target"dos nodos de dos vías dentro de una misma ciudad.

OSM					
id	**osm_id**	**osm_source_id**	**osm_target_id**	**source**	**target**
1975135	192505339	864117527	864117754	1434779	1434780
España, Palencia, vía: 'Calle Prado de la Lana'					
778645	73285730	864117701	985627854	546019	642722
España, Palencia, vía: 'Calle de San Juan de la Cruz'					

Tabla 6.12: Prueba 3 - OSM.

TeleAtlas					
gid	**id**	**f_jnctid**	**t_jnctid**	**source**	**target**
3987557	17240026643026	17240203205304	17240203206510	3266762	3266763
España, Palencia, vía: 'Calle Prado de la Lana'					
3459984	17240017333790	17240203205947	17240202077294	2805070	2805073
España, Palencia, vía: 'Calle de San Juan de la Cruz'					

Tabla 6.13: Prueba 3 - TeleAtlas.

Prueba 4, 5 y 6: Usando como "sourceż "target"dos nodos de dos vías de diferentes ciudades.

OSM					
id	**osm_id**	**osm_source_id**	**osm_target_id**	**source**	**target**
1975135	192505339	864117527	864117754	1434779	1434780
España, Palencia, vía: 'Calle Prado de la Lana'					
318288	33644285	384886002	765089642	221291	221289
España, Valladolid, vía: 'Calle del Doctor Bañuelos'					

Tabla 6.14: Prueba 4 - OSM.

TeleAtlas					
gid	**id**	**f_jnctid**	**t_jnctid**	**source**	**target**
3987557	17240026643026	17240203205304	17240203206510	3266762	3266763
España, Palencia, vía: 'Calle Prado de la Lana'					
3447463	17240001998222	17240202059262	17240203974549	2793684	2793685
España, Valladolid, vía: 'Calle del Doctor Bañuelos'					

Tabla 6.15: Prueba 4 - TeleAtlas.

OSM					
id	**osm_id**	**osm_source_id**	**osm_target_id**	**source**	**target**
1975135	192505339	864117527	864117754	1434779	1434780
España, Palencia, vía: 'Calle Prado de la Lana'					
1096188	101219220	1168505972	1168505914	774094	774100
España, Toledo, vía: 'Paseo del Circo Romano'					

Tabla 6.16: Prueba 5 - OSM.

TeleAtlas					
gid	**id**	**f_jnctid**	**t_jnctid**	**source**	**target**
3987557	17240026643026	17240203205304	17240203206510	3266762	3266763
España, Palencia, vía: 'Calle Prado de la Lana'					
5695966	17240062710887	17240204408692	17240204408845	4534111	4547072
España, Toledo, vía: 'Paseo del Circo Romano'					

Tabla 6.17: Prueba 5 - TeleAtlas.

OSM					
id	**osm_id**	**osm_source_id**	**osm_target_id**	**source**	**target**
1975135	192505339	864117527	864117754	1434779	1434780
España, Palencia, vía: 'Calle Prado de la Lana'					
1420406	146237326	61366599	185927162	1012571	1012572
España, Granada, vía: 'Calle Labella Dávalos'					

Tabla 6.18: Prueba 6 - OSM.

TeleAtlas					
gid	**id**	**f_jnctid**	**t_jnctid**	**source**	**target**
3987557	17240026643026	17240203205304	17240203206510	3266762	3266763
España, Palencia, vía: 'Calle Prado de la Lana'					
1591812	17240001215509	17240201590717	17240201577532	1272531	1272908
España, Granada, vía: 'Calle Labella Dávalos'					

Tabla 6.19: Prueba 6 - TeleAtlas.

Capítulo 7

Uso en entornos de producción

Por el momento se ha trabajado en la funcionalidad de esta plataforma obteniendo el requisito clave de la ruta óptima Valladolid-Varsovia, pero el modo en que se realizan las consultas no facilita la usabilidad del sistema. Con la información ofrecida hasta el momento por el esquema de las bases de datos espaciales, no es posible la identificación de una vía de forma sencilla. Por ahora, la única forma de identificar una vía de forma única es a través de su campo "osm_id", el identificador univoco que OpenStreetMap asigna a cada vía de su cartografía.

Esto supone una ardua tarea de identificación que se detalla en la siguiente sección de este capitulo, no siendo viable con que el sistema desarrollado pueda ser usado en entornos de producción. Lógicamente, llevar a cabo este proceso es incompatible con el uso de cualquier sistema, aún más cuando se trata de un sistema de enrutado donde indicar origen y destino tiene que ser un mero trámite. Ante este inconveniente surge la necesidad de poblar las bases de datos espaciales con un mayor detalle de información, los esquemas no proveen información útil como país, comunidad, provincia, municipio, código postal, etc, información de codificación geográfica que facilite la identificación de una vía.

En las siguientes secciones se comentan las tareas llevadas a cabo para poblar las bases de datos con dicha información. Inicialmente se opto por utilizar la cartografía OSM cargando la capa correspondiente, pero se observo que esta cartografía no respeta un estándar a la hora de registrar sus vías, por lo que hay multitud de inconsistencias que hacen que sea inviable su uso para localizar una vía. La única alternativa posible era emplear TeleAtlas, pero ya se había comprobado que era una mala opción a la hora de calcular rutas óptimas con pgRouting, dado que incrementaba los tiempos de respuesta considerablemente. Aunque, como se detalla en el correspondiente apartado, era posible combinar el uso de ambas cartografías, destinando TeleAtlas a la identificación de la vía y OpenStreetMap al cálculo de rutas.

Logrado el objetivo de facilitar la identificación de una vía de forma única en el sistema, se implementa una interfaz web basada en dicho método, de modo que se facilite la interacción con el sistema a la hora de realizar las consultas sobre la ruta óptima entre dos vías de la red lineal.

7.1. Identificación de una vía

Hasta el momento es necesario consultar la página oficial de OpenStreetMap (`https://www.openstreetmap.org`) para certificar que el valor del campo "osm_id" se corresponde con el de la vía que se quiere establecer como origen o destino. Veamos este proceso mediante un ejemplo.

Supongamos que estamos interesados en tomar como origen la "Calle Prado de la Lana" que se encuentra en la provincia de Palencia. Si realizamos una simple consulta como la que se muestra a continuación, obtenemos 9 resultados:

```
spain=# select id, osm_id, osm_name, source, target from sp_2po_4pgr where osm_name=
'Calle Prado de la Lana';

   id    |  osm_id   |        osm_name         | source  | target
---------+-----------+-------------------------+---------+---------
  782695 |  74202542 | Calle Prado de la Lana |  548906 |  544149
  782696 |  74202542 | Calle Prado de la Lana |  544149 |  544163
  782697 |  74202542 | Calle Prado de la Lana |  544163 |  546025
  782698 |  74202542 | Calle Prado de la Lana |  546025 |  548907
 1975135 | 192505339 | Calle Prado de la Lana | 1434779 | 1434780
 1975136 | 192505339 | Calle Prado de la Lana | 1434780 | 1434781
 1975137 | 192505339 | Calle Prado de la Lana | 1434781 | 1434782
 1975138 | 192505339 | Calle Prado de la Lana | 1434782 |  544153
 1975139 | 192505339 | Calle Prado de la Lana |  544153 | 1434783
(9 rows)
```

Si nos fijamos en el campo "osm_id" tenemos dos valores diferentes para todo el conjunto de resultados. Esto quiere decir que en la cartografía española hay dos vías registradas con el mismo nombre. Y ahora, ¿cómo sabemos que vía es la que estamos buscando?

Pues hasta el momento la única manera es dirigirse a la página oficial de OSM, buscar la vía y observar el identificador de la misma, tal y como muestra la siguiente imagen:

Figura 7.1: Ejemplo de búsqueda en OpenStreetMap.

7.2. Importar codificación geográfica desde la cartografía OSM

La cartografía de OSM, como ya se comento, se distribuye en un único fichero que contiene datos de todo tipo: desde calles y carreteras, edificios, ríos, hasta tendido eléctrico. Toda la información se estructura mediante etiquetas que clasifican los datos. Ver la sección "3.4.2. Importación de datos OSM en PostGIS" apartado "Estructura de la base de datos OSM".

Teniendo esto en cuenta se procedió a utilizar la herramienta osm2pgsql, la cual permite seleccionar que información se desea importar modificando su fichero de configuración. Las herramientas como osm2pgrouting y osm2po, destinadas a crear esquemas de enrutado, no permiten llevar a cabo esta tarea. Están limitadas a importar únicamente aquella información de la base de datos OSM cuyo tag se corresponde con "higways".

Así que la opción era cargar la capa de codificación geográfica mediante osm2pgsql y mediante el campo "osm_id", el identificador único de cada vía en la cartografía de OSM, consultar los esquemas de enrutado creados por las otras dos herramientas.

Para realizar la importación con osm2pgsql era necesario modificar su archivo de configuración "mapconfig.xml" para que importe la capa especificada. Se consulto información sobre el sistema de etiquetado de OSM para encontrar la etiqueta o etiquetas que identifican datos de codificación geográfica. Para ello se consulto principalmente la página que OpenStrreetMap ofrece con información sobre sus tags (`https://taginfo.openstreetmap.org/`). Así se encontró la etiqueta (tag) "street", que dispone de múltiples combinaciones las cuales proveen información como país, ciudad, código postal, etc. Es decir, lo que se estaba buscando. En la siguiente tabla se recogen algunas de las combinaciones disponibles:

Etiqueta	Descripción
addr:housenumber	Número de casa
addr:street	Nombre de la calle
addr:postcode	Código postal
addr:city	Municipio
addr:province	Provincia
addr:state	Comunidad
addr:country	País

Tabla 7.1: Etiqueta OSM street.

Una vez identificado el tag necesario para importar la capa de codificación geográfica, se procedió a modificar el archivo de configuración "default.style" de osm2pgsql. Basto con añadir las siguientes líneas en dicho archivo:

```
# OsmType  Tag                 DataType      Flags
node,way   addr:housenumber    text          linear
node,way   addr:street         text          linear
node,way   addr:postcode       text          linear
node,way   addr:city           text          linear
node,way   addr:province       text          linear
node,way   addr:state          text          linear
node,way   addr:country        text          linear
```

Tras realizar dicha modificación, el último paso necesario era ejecutar la herramienta osm2pgsql para realizar la importación de la capa. Una vez finalizada la importación, contamos con una serie de tablas generadas por osm2pgsql, siendo la tabla "planet_osm_line"la que recoge toda la información referente a la capa de codificación geográfica, además de otra serie de datos. Parte de la estructura de esta tabla se recoge a continuación:

Campo	Tipo	Descripción
osm_id	bigint	Identificador de OpenStreetMap
addr:housenumber	text	Número de casa
addr:street	text	Nombre de la calle
addr:postcode	text	Código postal
addr:city	text	Municipio
addr:province	text	Provincia
addr:state	text	Comunidad
addr:country	text	País

Tabla 7.2: Campos de la tabla de codificación geográfica OSM.

Pero la sorpresa fue comprobar como la cartografía libre de OpenStreetMap no sigue un estándar para almacenar la información de sus vías. Una vez cargada la codificación geográfica de todas las vías a nivel del país de España, se consulto las provincias registradas, obteniéndose el siguiente resultado:

```
spain_addr=# select distinct "addr:province" from  planet_osm_line;

 addr:province
---------------
 Cantabria
 Salamanca
 Illes Balears
 Jaén
(5 rows)
```

Como se puede apreciar, de las cincuenta provincias (además de las ciudades autónomas de Ceuta y Melilla) con las que cuenta España, solo cuatro aparecen referenciadas en la cartografía. De hecho se comprobó como no aparece la provincia de Valladolid, mientras que si aparece referenciada como municipio:

```
spain_addr=# select distinct "addr:city" from  planet_osm_line where "addr:city" like '%ad%';

        addr:city
--------------------------
 Bercio, Grado
 Fuejo-Grado
 Fuenlabrada
 Sada
 Valladolid
 Ribadeo
 Ponferrada
 Madrid
```

```
 Barriada de las Malvinas
 La Cavada
(10 rows)
```

Si se consulta una de las vías cuya ciudad sea Valladolid, se ve que la mayoría de los campos aparecen vacíos:

```
spain_addr=# select * from  planet_osm_line where "addr:city"='Valladolid';

  osm_id  | addr:housenumber | addr:street | addr:postcode | addr:city  | addr:province |
  addr:state | addr:country
----------+------------------+-------------+---------------+------------+---------------
 86715377 |                  |             |               | Valladolid |               |
            |
(1 row)
```

Así que llegados a este punto, se comprobó que con la cartografía OSM no era posible garantizar la usabilidad del proyecto. ¿La alternativa? En el siguiente apartado se detalla.

7.3. Combinar el uso de TeleAtlas y OpenStreetMap

Ya se había comprobado que utilizar la cartografía de TeleAtlas ralentizaba considerablemente el cálculo de rutas óptimas, debido a su mayor nivel de detalle. Por lo que utilizar esta cartografía para localizar origen/destino y luego calcular la ruta óptima no parecía una buena opción desde el principio.

Por lo que la solución planteada ante esta situación se basa en utilizar la cartografía de TeleAtlas para obtener una vía aportando su nombre, código postal, provincia, país, etc. De dicha vía obtener su geometría (coordenadas) y buscar la geometría (coordenadas) más cercana a ella en la cartografía de OSM.

7.3.1. Importar codificación geográfica desde la cartografía TeleAtlas

El primer paso fue identificar la capa de la cartografía de TeleAtlas que provee la codificación geográfica sobre las vías. Para ello se empleo los manuales que se distribuyen junto con los DVDs de la cartografía de TeleAtlas. Tarea nada sencilla y bastante intuitiva dado que la información contenida en estos manuales es bastante esquemática y carece de detalles.

Pero finalmente se dio con la capa en cuestión, la capa de geocodificación (GC). Esta capa esta diseñada exclusivamente para codificación geográfica. Contiene todos los atributos necesarios para las operaciones de geocodificación de direcciones. En la siguiente tabla aparecen sus principales atributos:

Campo	Tipo	Descripción
gid	integer	Clave primaria
id	double precison	Identificador de vía
feattyp	smallint	Tipo de elemento de transporte
fullname	character varying(150)	Nombre de la calle
namelc	character varying(3)	Código de idioma
nametyp	smallint	Tipo de vía
l_laxon	character varying(100)	Municipio
l_axon	character varying(100)	Provincia
l_order01	character varying(11)	Comunidad autónoma
l_pc	character varying(10)	Código postal
geom	geometry(MultiLineString,4326)	Geometría (coordenadas)

Tabla 7.3: GC Geocodificación, Geometría con atributos de geocodificación.

No se entrara en detalles de como se importo dicha capa, dado que se siguió el proceso indicado en la sección "3.4. Importación de cartografia" apartado "3.4.1. TeleAtlas: Importar cartografía a PostGIS". Solo resaltar que la capa de geocodificación (GC) únicamente se distribuye con extensión .dbf, por lo que es necesario utilizar la opción "-n" de la herramienta de importación shp2pgsql para indicar que solo se importa el archivo .dbf.

Este proceso de importación solo se llevo a cabo con la cartografía española, dado que realizar el mismo para la cartografía europea suponía un coste de espacio de almacenamiento y de tiempo de procesado del cual no disponíamos en dicho momento.

Una vez realizada la importación, contamos con una base de datos espacial "routing" la cual dispone de la tabla sp_2po_4pgr que contiene la información de enrutado OSM y la tabla esp_eur2014 que contiene la codificación geográfica de direcciones de TeleAtlas. Todo ello a nivel de país, en concreto, España.

Llegados a este punto se verifico que esta vez la información de geocodificación estuviera correctamente referenciada y no ocurriera lo mismo que con la cartografía OSM. De este modo se comprobó que la cartografía TeleAtlas contenía correctamente los valores de geocodificación para España, como puede verse en las siguientes consultas:

- El código de idioma, como se puede visualizar aparecen: inglés (ENG), gallego (GLG), español (SPA), valenciano (VAL), catalán (CAT), euskera (BAQ) e indeterminado (UND). Se recoge el inglés dentro de España debido a que la cartografía de TeleAtlas incorpora en el mismo área geográfica el territorio británico de Gibraltar.

```
routing=# select distinct namelc from esp_eur2014;

 namelc
--------
 ENG
 GLG
 SPA
 VAL
 CAT
```

```
 BAQ
 UND
(7 rows)
```

- El número de municipios. Actualmente existen en España un total de 8122 municipios, que el número de municipios que devuelve esta consulta sea mayor se debe básicamente a que TeleAtlas registra nombres de municipios en las variantes vinculadas al territorio al que pertenecen. Es decir, por ejemplo el municipio de Guecho de la provincia de Vizcaya, aparece registrado tanto con su topónimo en español (Guecho) como por su topónimo en Euskera (Getxo). Y así multitud de municipios pertenecientes a todo el país. De hecho lo mismo ocurre con el nombre de vías o provincias.

```
routing=# select count(distinct l_laxon) from esp_eur2014;

count
-------
  8362
(1 row)
```

- El número de provincias y de comunidades autónomas. España está organizada territorialmente en 17 comunidades autónomas (además de las ciudades autónomas de Ceuta y Melilla) y 50 provincias. Datos que se vinculan perfectamente a los devueltos en las siguientes consultas, ya que el número de comunidades autónomas concuerda perfectamente al incluir las ciudades autónomas (19). Y como se ha comentado para el caso de los municipios, los nombres de las provincias se registran según varios topónimos (ejemplo: Vizcaya – Bizkaia).

```
routing=# select count(distinct l_axon) from esp_eur2014;

 count
-------
    62
(1 row)

routing=# select count(distinct l_order01) from esp_eur2014;

 count
-------
    19
(1 row)
```

- Veamos un ejemplo de una vía que aparece registrada dos veces debido a que dispone de varios topónimos. Podemos ver como en este caso solo se ven alterados el campo "fullname" y "namelc", que se corresponden con el nombre de la vía y el código de idioma respectivamente.

```
routing=# select * from esp_eur2014 where gid=1867482 or gid=1867483;

      fullname       | namelc | l_laxon  | l_axon  | l_order01 | l_pc
```

```
--------------------+--------+----------+---------+-----------+-------
 Ibaizabal Kalea    | BAQ    | Galdakao | Bizkaia | PAV       | 48960
 Calle de Ibaizábal | SPA    | Galdakao | Bizkaia | PAV       | 48960
(2 rows)
```

De hecho se comprobó que las coordenadas geográficas contenidas en su geometría fueran las mismas:

```
routing=# select ST_AsText(geom) from esp_eur2014 where gid=1867482;

                          st_astext
-------------------------------------------------------------
 MULTILINESTRING((-2.8674241 43.2342775,-2.8668867 43.2339281))
(1 row)

routing=# select ST_AsText(geom) from esp_eur2014 where gid=1867483;

                          st_astext
-------------------------------------------------------------
 MULTILINESTRING((-2.8674241 43.2342775,-2.8668867 43.2339281))
(1 row)
```

Retomando la sección, queda claro que la codificación geográfica que nos proporciona la cartografía de TeleAtlas cumple los requisitos para ser la solución en la que apoyarnos a la hora de identificar una vía de forma univoca a través de su codificación geográfica.

7.3.2. Superponer coordenadas geográficas de TeleAtlas en OpenStreetMap

El problema ahora es que la cartografía de TeleAtlas no registra del mismo modo las vías que la cartografía de OpenStreetMap. Lógicamente es prácticamente imposible que una misma vía comparta las mismas coordenadas geográficas en ambas cartografías, dado que la producción cartográfica de ambos sistemas utiliza técnicas, así como fuentes, distintas para su recopilación.

Pero esto no supone un impedimento para relacionar una vía obtenida en TeleAtlas, con su correspondiente representación en OpenStreetMap. Dado que gracias a sus coordenadas geográficas almacenadas en el campo de geometría y a la multitud de funciones geográficas ofrecidas por PostGIS, es posible obtener un resultado altamente fiable. Para ello, el procedimiento se basa en obtener la geometría (coordenadas) que representa la vía en TeleAtlas y superponer esta en la cartografía de OSM, obteniendo la vía OSM cuya geometría sea la más cercana a la de TeleAtlas.

Eso si, se debe recordar que OSM es una cartografía libre que crece día a día pero que a pesar de la multitud de datos cartográficos que posee, se puede encontrar que alguna vía aún no este incluida. Aunque con el método planteado se obtendría siempre la vía alternativa más cercana en OSM.

Para llevar a cabo este proceso se comprobó inicialmente los tipos y proyecciones de la geometría de ambas cartografías, siendo el resultado obtenido:

Cartografía	Campo	Tipo
TeleAtlas	geom	geometry(MultiLineString,4326)
OSM	geom_way	geometry(LineString,4326)

Tabla 7.4: Tipo de geometría y proyección.

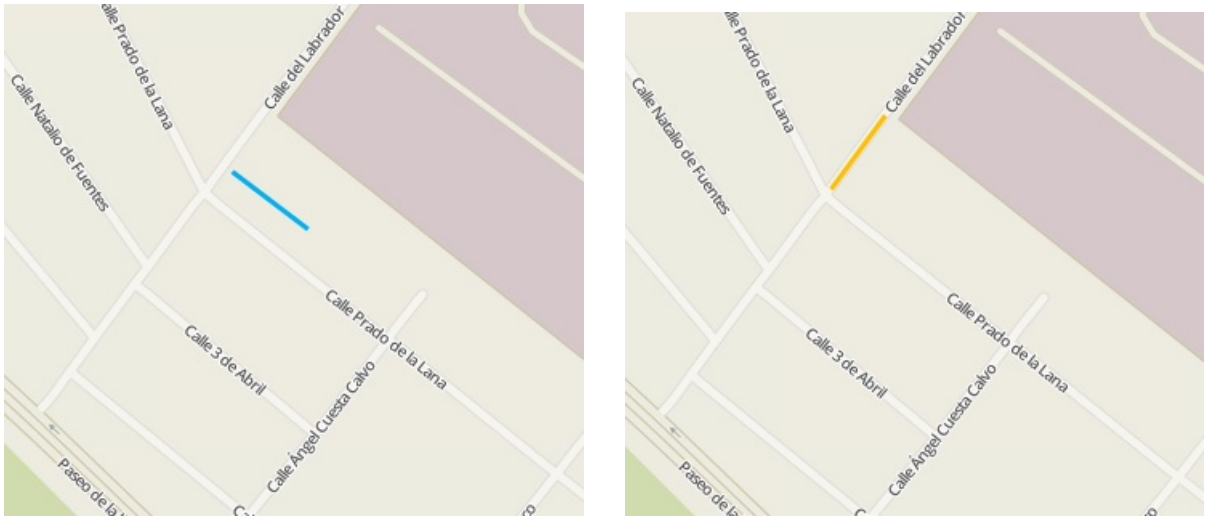
Las dos cartografías utilizan el mismo sistema de proyección (4326), lo cual facilita enormemente la tarea a la hora de trabajar con las coordenadas de una y otra cartografía, no siendo necesario realizar operaciones de conversión y ajuste de las mismas.

El tipo de geometría con que se representan las vías es distinto en cierta medida. La única diferencia entre el tipo LineString y MultiLineString radica en que este último puede estar formado por una o varias representaciones LineString. Pero se verifico que aunque el tipo de geometría empleada en TeleAtlas sea MultiLineString, este únicamente se compone de un LineString para todos los casos.

```
routing=# select ST_AsText(geom) from esp_eur2014 where id=17240026643026;

                              st_astext
-----------------------------------------------------------------
 MULTILINESTRING((-4.5275812 41.9947632,-4.5288098 41.9954846))
(1 row)
```

Comprobado que no había obstáculos a la hora de superponer una geometría TeleAtlas sobre la cartografía OpenStreetMap, se procedió con el diseño y desarrollo de la consulta que permitiera llevar a cabo dicha tarea. La siguiente imagen muestra la idea del proceso a llevar a cabo y el resultado esperado: obtener la vía OSM cuya geometría sea la más cercana a la de TeleAtlas.



(a) Geometría de Teleatlas sobre OSM

(b) Geometría OSM más cercana

Figura 7.2: Superponer geometría TeleAtlas en OpenStreetMap.

Tras varios intentos y diseños se dio con la consulta que permitía realizar este procedimiento, obteniendo el resultado deseado y de forma fiable. La sintaxis de la consulta desarrollada es:

```
select id, osm_id, osm_name, source, ST_Distance(ST_SetSRID(ST_LineMerge('GEOM'),4326),
geom_way) as distance from sp_2po_4pgr where ST_Dwithin(geom_way, ST_SetSRID(ST_LineMerge
('GEOM'),4326), 0.001) and osm_name!='' order by distance limit 1;
```

Analicemos inicialmente las funciones PostGIS utilizadas en esta consulta, para proceder después a comentar el resultado final que se consigue con su combinación.

- ST_LineMerge

  ```
  geometry ST_LineMerge(geometry amultilinestring);
  ```

  Esta función retorna una geometría de tipo LineString a partir de un tipo MultiLineString que recibe como argumento. Básicamente une los posibles LineString que forman el tipo MultiLineString en uno solo.

  ```
  select ST_AsText(ST_LineMerge(ST_GeomFromText(
  'MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45 -33,-46 -32))')));

                        st_astext
  ----------------------------------------------------
  LINESTRING(-29 -27,-30 -29.7,-36 -31,-45 -33,-46 -32)
  (1 row)
  ```

- ST_SetSRID

  ```
  geometry ST_SetSRID(geometry geom, integer srid);
  ```

  Establece el SRID ("identificador de referencia espacial") en una geometría a un valor entero en particular.

- ST_Distance

  ```
  float ST_Distance(geometry g1, geometry g2);
  ```

  Calcula la mínima distancia entre dos geometrías que recibe como argumento.

  ```
  select ST_Distance(ST_GeomFromText('POINT(-72.1235 42.3521)',4326),
  ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 42.1546)', 4326));

  st_distance
  -----------------
  0.00150567726382282
  ```

- ST_Dwithin

  ```
  boolean ST_DWithin(geometry g1, geometry g2, double precision distance_of_srid);
  ```

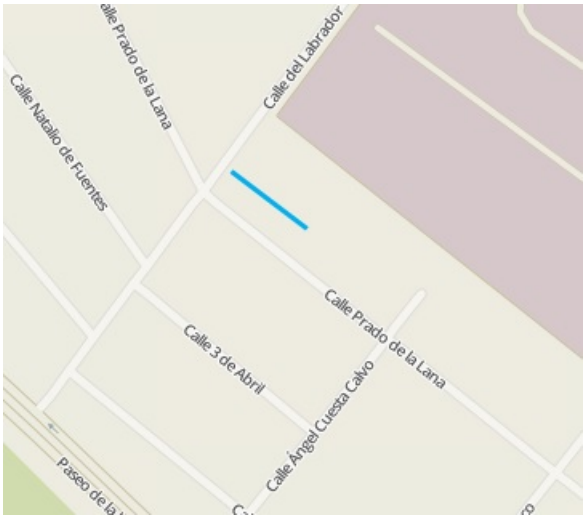
  Búsqueda o selección de entidades geométricas que se encuentran dentro de una determinada distancia de otras entidades geométricas. Esta función permite calcular si dos objetos se encuentran a una distancia dada uno del otro, en cuyo caso devuelve true. Ya fue analizada en anteriores apartados, en concreto la sección "5.2. Propuesta de los desarrolladores" del capítulo 5. Consultar en caso de dudas.

Presentada la competencia particular de cada una de estas funciones, analicemos el resultado global obtenido al combinarlas como aparecen en la sintaxis de la consulta.

```
select id, osm_id, osm_name, source, ST_Distance(ST_SetSRID(ST_LineMerge('GEOM'),4326),
geom_way) as distance from sp_2po_4pgr where ST_Dwithin(geom_way, ST_SetSRID(ST_LineMerge
('GEOM'),4326), 0.001) and osm_name!='' order by distance limit 1;
```

La combinación de ST_LineMerge y ST_SetSRID es necesaria simplemente por cuestiones de conversión de tipos (casting) que son requeridas a la hora de poder emplear el resto de funciones. GEOM se correspondería con el campo geometría de la vía seleccionada en TeleAtlas, es sobre dicho campo de tipo MultiLineString sobre el que se aplican estas conversiones.

Mediante la función ST_Distance se obtiene la vía OSM cuya geometría es la más cercana a la geometría de la vía TeleAtlas. Pero sera necesaria la función ST_Dwithin para acotar el área de búsqueda a unos 100 metros alrededor de la geometría TeleAtlas, ya que en caso de no utilizar esta acotación se aplicaría la función ST_Distance a todas las vías almacenadas en OSM, ralentizando enormemente la consulta por el cálculo de operaciones innecesarias.



(a) Geometría de Teleatlas sobre OSM

(b) Geometría OSM más cercana dentro del área

Figura 7.3: Búsqueda de la geometría OSM más cercana.

Un ejemplo de la salida retornada por la consulta hasta el momento sería:

```
routing=# select id, osm_id, osm_name, st_distance(ST_SetSRID(ST_LineMerge(
'MULTILINESTRING((-4.5275812 41.9947632,-4.5288098 41.9954846))'),4326), geom_way) as
distance from sp_2po_4pgr where ST_DWithin(geom_way, ST_SetSRID(ST_LineMerge(
'MULTILINESTRING((-4.5275812 41.9947632,-4.5288098 41.9954846))'),4326), 0.001) and
osm_name!='' order by distance;

   id    |  osm_id   |         osm_name          |       distance
---------+-----------+---------------------------+----------------------
  775637 |  72799684 | Calle del Labrador        | 1.35824364499831e-06
 1975135 | 192505339 | Calle Prado de la Lana    | 3.19186166701964e-06
  775636 |  72799684 | Calle del Labrador        | 3.75765884428332e-06
```

```
  775609 |  72799651 | Calle Ángel Cuesta Calvo  | 6.55487962750859e-06
  775610 |  72799651 | Calle Ángel Cuesta Calvo  | 1.24615408372189e-05
 1975136 | 192505339 | Calle Prado de la Lana    | 1.24615408372189e-05
  775656 |  72799722 | Calle 3 de Abril          | 0.000633024432091053
  775635 |  72799684 | Calle del Labrador        | 0.000655972758577476
  775611 |  72799651 | Calle Ángel Cuesta Calvo  | 0.000663400884994419
(9 rows)
```

En este ejemplo se ha especificado a mano un valor concreto para GEOM, en la practica este valor sera pasado como parámetro que tomara su valor cuando seleccionemos la vía TeleAtlas. Como vemos devuelve un conjunto de vías cuya distancia es mínima a la geometría TeleAtlas, los valores se muestran de modo creciente por lo que la vía OSM más cercana se corresponde con la primera fila. Ajustando nuestra consulta como se muestra a continuación, afinamos el resultado obtenido:

```
routing=# select id, osm_id, osm_name, st_distance(ST_SetSRID(ST_LineMerge(
'MULTILINESTRING((-4.5275812 41.9947632,-4.5288098 41.9954846))'),4326), geom_way) as
distance from sp_2po_4pgr where ST_DWithin(geom_way, ST_SetSRID(ST_LineMerge(
'MULTILINESTRING((-4.5275812 41.9947632,-4.5288098 41.9954846))'),4326), 0.001) and
osm_name!='' order by distance limit 1;

   id   |  osm_id  |      osm_name      |       distance
--------+----------+--------------------+----------------------
 775637 | 72799684 | Calle del Labrador | 1.35824364499831e-06
(1 row)
```

Este resultado, así como el resto de pruebas realizadas, garantizan la fiabilidad y funcionamiento de la sentencia desarrollada. Habiendo comprobado meticulosamente que la vía OSM que se obtiene de esta consulta, realmente se corresponde con la vía más cercana a la geometría de TeleAtlas.

7.4. Diseño e implementación de una interfaz web

Finalmente se abordo la tarea de desarrollar una interfaz que facilitase la interacción con el Sistema de Información Geográfica planteado. Esta tarea no entra dentro de los requisitos u objetivos definidos por la empresa GMV, ya que dicha empresa cuenta con sus propias interfaces adaptadas a los dispositivos desde los que se realizan las peticiones. De hecho, GMV únicamente solicito que el sistema fuera capaz de devolver, como información a la consulta del camino óptimo, las coordenadas geográficas que componían la ruta. Pero no obstante, se considero por parte del tutor del proyecto un aspecto interesante en vistas a la presentación final de este trabajo fin de grado y a facilitar el intercambio de información con la plataforma, el desarrollo de una interfaz web. Así que se procedió a diseñar una interfaz basada en el lenguaje de programación PHP para su implementación.

En cuanto al diseño, la solución más utilizada actualmente en programación web para organizar el código es el patrón de diseño MVC. Como ya se ha estudiado en alguna asignatura del grado, la idea básica de este patrón es separar nuestro sistema en tres capas: el Modelo, la Vista y el Controlador.

- El Modelo se encarga de todo lo que tiene que ver con la persistencia de datos. Guarda y recupera la información del medio persistente que utilicemos, en el caso de este proyecto, la base de datos espacial.

- La Vista presenta la información obtenida con el modelo de manera que el usuario la pueda visualizar.
- El Controlador, dependiendo de la acción solicitada por el usuario, es el que pide al modelo la información necesaria e invoca a la plantilla (de la vista) que corresponda para que la información sea presentada.

Al utilizar el patrón de diseño MVC, obtenemos numerosos beneficios tales como: aplicar la modularidad y la partición de aplicación, aumentar la capacidad de gestión de código y aumentar la extensibilidad del código (capacidad de adaptación a cambios). Pero el diseño no quedo aquí, se amplió con la utilización de Front Controller con Dispatchers. Front Controller es un patrón de diseño de software muy utilizado en aplicaciones web que consiste en definir un único punto de acceso para todas las peticiones, delegando en el Dispatcher correspondiente la resolución de datos y el manejo de la vista.

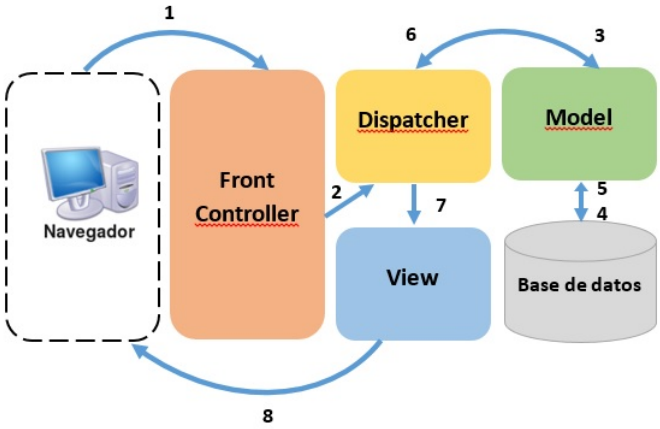


Figura 7.4: Diseño de la interfaz.

Una vez definido el diseño de nuestra aplicación web, se procedió a su implementación. Para ello fue necesario contar con el siguiente software:

- Apache 2
- PHP5 y el módulo de Apache PHP5
- Soporte pgsql en PHP5 y en Apache 2

Este software en conjunto proporciona un servidor web HTTP que permite alojar sitios web basados en PHP e interacturar con el sistema gestor de bases de datos PostgreSQL. El proceso de instalación de este software se recoge en el correspondiente manual de instalación adjuntado mediante soporte digital junto a esta memoria. Sólo resaltar que es necesario contar con este software para poder recibir y procesar las solicitudes, así como la importancia de configurar el dominio donde se encuentra alojado el servicio.

Dicho esto, en el siguiente diagrama de despliegue se provee la vista de implementación del sistema. A la hora de describir el sitio web, este diagrama de despliegue muestra los componentes hardware que lo componen, los componentes software que se ejecutan en cada nodo y la forma en la que las distintas partes están conectadas entre sí.

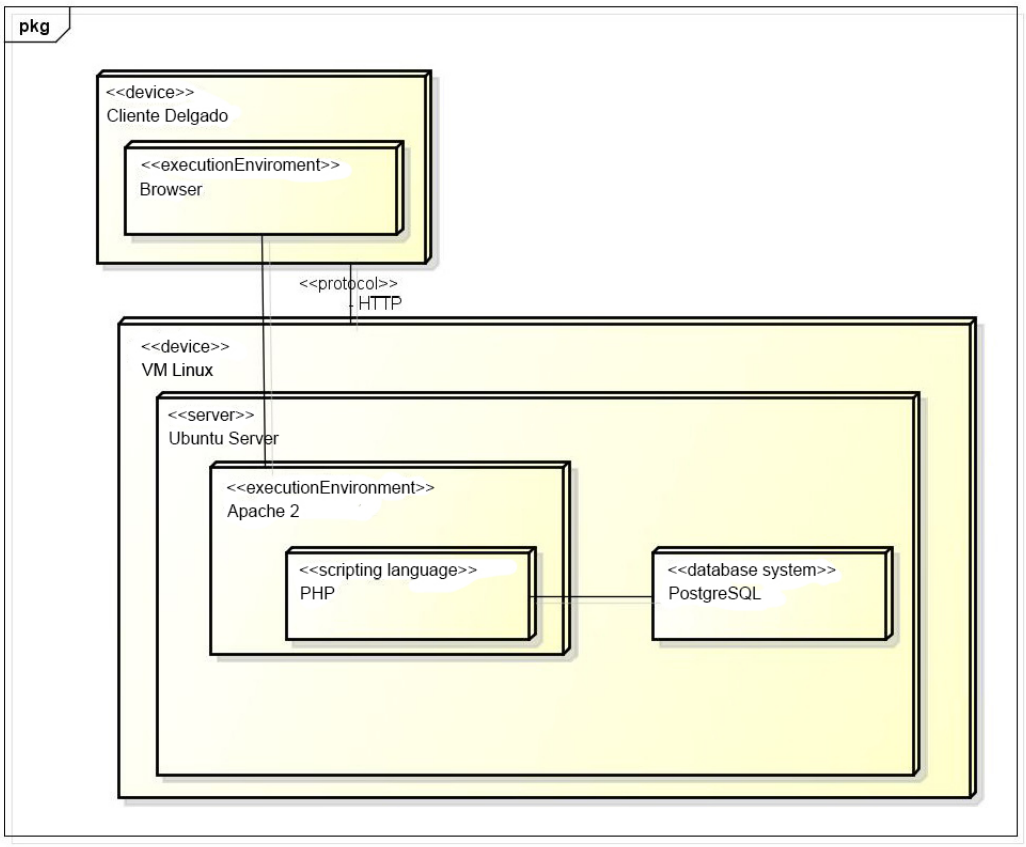


Figura 7.5: Despliegue del sistema final.

Sobre esta implementacion, la ubicación predeterminada donde crear nuestro árbol web es, por defecto, /var/www/html. Aquí es donde guardaremos los archivos de nuestra web que se adjuntan en el soporte digital (carpeta Aplicación Web/). Basta con copiar el contenido de esta carpeta en el directorio /var/www/html para poder trabajar con el código desarrollado. A continuación se muestra un esquema de la estructura de directorios que componen la aplicación:

Directorio GMV:

- config/config.php: Archivo de configuración de la aplicación.
- css/: Hoja de estilo CSS que define detalles de la presentación.
- dispatchers/: Ubicación de los controladores secundarios que se seccionan dependiendo de la acción solicitada por el usuario. Son los que piden al modelo la información necesaria e invocan a la vista correspondiente para que la información sea presentada.
- images/: Contenido multimedia de la aplicación.
- libs/: Contiene el controlador principal y una serie de librerías.
- modelos/: Contiene los modelos que se encargan de todo lo que tiene que ver con almacenar y recuperar la información de la base de datos.
- vistas/: Vistas que presentan la información obtenida con el modelo de manera que el usuario la pueda visualizar.

- index.php: Página principal de acceso a la aplicación.

- layout.php: Código HTML genérico a toda la aplicación.

El código es bastante simple de entender y viene acompañado de comentarios a lo largo del mismo. No se entrara en detalles en esta memoria, ya que como se ha indicado, dicho código esta disponible en el soporte digital que se adjunta junto a este documento. Si se desea analizar el mismo basta con acceder al directorio correspondiente.

Conclusiones y trabajo futuro

Conclusiones

- Se ha logrado desarrollar una plataforma que cumple los requisitos solicitados por la empresa GMV, aportando una batería de algoritmos que resuelven la obtención del camino más corto o Shortest Path entre dos nodos o ejes de la red lineal.
- Se han evaluado soluciones de cálculo de rutas basadas en el uso de bases de datos. Dicha evaluación permite concluir que las soluciones analizadas presentan las ventajas e inconvenientes típicos del software libre. Por ejemplo, es necesario un cierto esfuerzo para la carga total de registros por las funciones de pgRouting.
- Se ha mejorado el proceso de carga de de registros en la base de datos gracias a la adaptación del mecanismo de selección del área de influencia BufferMultiLineString. Esto acota el número de registros cargados en memoria, a la hora de realizar los cálculos por parte de las funciones pgRouting. De este modo se mejora la solución planteada por los propios desarrolladores de dicho software.
- Se han mejorado considerablemente los tiempos de respuesta en las consultas. Gracias al área de influencia diseñada en este proyecto, dichos tiempos se reducen notablemente respecto a los tiempos manejados por la empresa GMV hasta el momento con su propio sistema, lo que demuestra la competitividad de la solución desarrollada.
- Se ha confirmado una vez más la importancia de los índices en las bases de datos. Si los índices no espaciales sobre una columna de una tabla son extremadamente importantes para acelerar las consultas alfanuméricas, los espaciales aún lo son más al indexar los datos en dos o tres dimensiones. Nuestra experiencia muestra que cuando los datos cartográficos exceden de algunas miles de filas, se debe crear índices espaciales para incrementar las velocidades de búsqueda espacial.
- Se ha conseguido que el desarrollo no dependa de componentes comerciales, ni en lo que respecta a los algoritmos, ni en el software subyacente, ni tampoco en los mapas utilizados. De hecho, se confirma mediante los experimentos realizados que la solución planteada permite trabajar tanto con mapas propietarios como libres.
- Se han desarrollado soluciones que posibilitan el uso de esta plataforman en entornos de producción. Dado que el sistema sólo permitía realizar cálculos de rutas utilizando su codificación geográfica, se ha diseñado una solución que compagina la cartografica libre de OSM y la comercial de TeleAtlas, superponiendo las coordenadas de una en la otra.

En resumidas cuentas, se ha conseguido evaluar exhaustivamente los pros y contras de un Sistema de Información Geográfica basado en el uso de bases de datos, en lo que respecta al cálculo de rutas. El

estudio demuestra que esta solución es competitiva desde el punto de vista del rendimiento, siendo además más flexible y versátil que otras alternativas, como el desarrollo de software que manipule directamente la cartografía de TeleAtlas.

Trabajo futuro

El trabajo recogido en esta memoria aún puede ser mejorado en muchos aspectos, siendo importante identificar las líneas de trabajo que pueden dar continuidad al esfuerzo invertido. Por esto, esta sección pretende mostrar el trabajo futuro que es necesario realizar para seguir avanzando en el conocimiento. Estas líneas pueden resumirse a grandes rasgos en los siguientes puntos:

- Una posible propuesta para el trabajo futuro sería hacer la evaluación del algoritmo pgr_TSP que pgRouting implementa para dar solución al famoso problema del viajante o Traveling Salesperson Problem (TSP). Si imaginamos un comerciante que debe visitar una serie de ciudades distintas el problema a resolver cosiste en encontrar una ruta óptima que pase una única vez por cada una de las ciudades minimizando la distancia total recorrida por el comerciante. Habría que estudiar como pgRouting implementa la solución para este problema, analizando para ello la salida aportada por la función y el código que compone la misma. De modo que se pueda concretar si se trata de una solución eficiente para dicho problema.

- Otra línea de trabajo futuro sería producir una aplicación web que permita obtener rutas óptimas a nivel europeo. Como se ha reflejado en la memoria de este trabajo, la cartografía de TeleAtlas requería de un intenso procesamiento para poder cargar la cartografía europea y manejar tiempos de respuesta adecuados. Además, la interfaz desarrollada es un prototipo básico que enruta a través de España, estando adapta a la distribución regional del país. Por lo que habría que buscar un esquema que se adapte al resto de países.

- Siguiendo por este mismo enfoque, sería interesante tratar aspectos como la creación de una interfaz que mediante herramientas como OpenLayers y Geoserver, dibuje la ruta óptima calculada sobre un mapa. Al mismo tiempo, se podrían estudiar herramientas que favorezcan la predicción de resultados, como AJAX, de modo que se realizase una búsqueda instantánea de resultados coincidentes con el patrón que se va introduciendo, de forma similar a como funciona el motor de búsqueda de Google Maps.

Apéndice A

Contenido del soporte digital

Junto a la copia en papel de este documento, se entrega un soporte digital con la siguiente información:

Soporte digital:

- memoria.pdf: La versión en PDF del documento impreso completo en un archivo único.
- Aplicación Web/: Carpeta que contiene la interfaz web desarrollada en versión fuente.
- Manuales/: Contiene la versión electrónica de los manuales de instalación de cada uno de los módulos que integran la plataforma, además de los manuales de instalación y uso del resto de herramientas.
- Software libre/: Contiene la versión de instalación de los módulos de la plataforma y de la herramienta de importación cartográfica, junto con el archivo OpenStreetMap para la cartografía española.

Bibliografía

[1] Frederik Ramm, Jochen Topf y Steve Chilton. *OpenStreetMap: Using and Enhancing the Free Map of the World*. UIT Cambridge, Septiembre 2010.

[2] Gustavo Buzai. *Sistemas de Información Geográfica (SIG) y Cartografía*. Lugar Editorial, Enero 2008.

[3] Hans-Jürgen Schönig. *PostgreSQL Administration Essentials*, Primera edición. Packt Publishing, Octubre 2014.

[4] Jonathan Bennett. *OpenStreetMap: Be your own Cartographer*. Packt Publishing, Septiembre 2010.

[5] José C. Martínez Llario. *PostGIS 2 Analisis Espacial Avanzado*, Primera edición. CreateSpace Independent Publishing Platform, Abril 2013.

[6] José Miguel Santos Preciado. *Sistemas de Información Geográfica*, Primera edición. Editorial UNED, Agosto 2004.

[7] Juan Antonio Cebrián de Miguel. *Información geográfica y Sistemas de información geográfica (SIGs)*, Primera edición. Editorial Universidad de Cantábrica, Abril 1992.

[8] Juan Peña Llopis. *Sistemas de Información Geográfica Aplicados a la Gestión del Territorio: Entrada, manejo, análisis y salida de datos espaciales. Teoría general y práctica para ESRI ArcGIS 9*. Editorial Club Universitario, 2006.

[9] Larry Ullman. *PHP: paso a paso*. Anaya Multimedia, 2009.

[10] Neil Matthew y Richard Stones. *Beginning Databases with PostgreSQL: From Novice to Professional*, Segunda edición. Apress, Septiembre 2007.

[11] Paolo Corti, Thomas J.Kraft, Stephen Vincent Mather y Bborie Park. *PostGIS CookBook*. Packt Publishing, Enero 2014.

[12] Regina O. Obe y Leo S. Hsu. *PostGIS in Action*, Primera edición. Manning Publications, Mayo 2011.

[13] Regina O. Obe y Leo S. Hsu. *PostgreSQL: Up and Running*, Segunda edición. O'Reilly Media, Diciembre 2014.

[14] Regina O. Obe y Leo S. Hsu. *PostGIS in Action*, Segunda edición. Manning Publications, Abril 2015.

[15] Scott McCracken. *Curso de programación web con HTML5, CSS, JavaScript, PHP 5/6 y MySQL*. Inforbook's, Noviembre 2011.

[16] Stefano Iacovella. *Geoserver CookBook*. Packt Publishing, Noviembre 2014.

[17] Stefano Iacovella y Brian Youngblood. *GeoServer Beginner's Guide*. Packt Publishing, Febrero 2013.

[18] Thomas Gratier, Erik Hazzard y Paul Spencer. *OpenLayers 3 Beginner's Guide*, Segunda edición. Packt Publishing, Enero 2015.

[19] Timothy Boronczyk, Elizabeth Naramore, Jason Gerner, Yann Le Scouarnec, Jeremy Stolz y Michael K. Glass. *Desarrollo web con PHP 6, Apache y MySQL*. Anaya Multimedia, Octubre 2009.

[20] GIS Stack Exchange. [Internet] Disponible en: `http://gis.stackexchange.com/` [Fecha de última consulta: 25 de agosto de 2015]

[21] OpenStreetMap. [Internet] Disponible en: `https://www.openstreetmap.org` [Fecha de última consulta: 25 de agosto de 2015]

[22] OpenStreetMap España. [Internet] Disponible en: `http://www.openstreetmap.es/` [Fecha de última consulta: 25 de agosto de 2015]

[23] pgRouting Documentation. [Internet] Disponible en: `http://pgrouting.org/documentation.html` [Fecha de última consulta: 25 de agosto de 2015]

[24] pgRouting Manual(2.0.0). [Internet] Disponible en: `http://docs.pgrouting.org/2.0/en/doc/index.html` [Fecha de última consulta: 25 de agosto de 2015]

[25] pgRouting Official Website. [Internet] Disponible en: `http://pgrouting.org/` [Fecha de última consulta: 25 de agosto de 2015]

[26] pgRouting Support. [Internet] Disponible en: `http://pgrouting.org/support.html` [Fecha de última consulta: 25 de agosto de 2015]

[27] pgRouting Tools. [Internet] Disponible en: `http://pgrouting.org/docs/tools.html` [Fecha de última consulta: 25 de agosto de 2015]

[28] pgRouting Workshop Manual. [Internet] Disponible en: `http://workshop.pgrouting.org/` [Fecha de última consulta: 25 de agosto de 2015]

[29] PHP: PostgreSQL – Manual. [Internet] Disponible en: `http://php.net/manual/es/book.pgsql.php` [Fecha de última consulta: 25 de agosto de 2015]

[30] Planet PostGIS: `http://planet.postgis.net/` [Fecha de última consulta: 25 de agosto de 2015]

[31] PostGIS Documentation. [Internet] Disponible en: `http://postgis.net/documentation`[Fecha de última consulta: 25 de agosto de 2015]

[32] PostGIS Essential. [Internet] Disponible en: `https://dzone.com/refcardz/essential-postgis` [Fecha de última consulta: 25 de agosto de 2015]

[33] PostGIS Official Website. [Internet] Disponible en: `http://postgis.net/` [Fecha de última consulta: 25 de agosto de 2015]

[34] PostGIS Support. [Internet] Disponible en: `http://postgis.net/support` [Fecha de última consulta: 25 de agosto de 2015]

[35] PostGIS Tutorial: Introduction to PostGIS. [Internet] Disponible en: `http://workshops.boundlessgeo.com/postgis-intro/` [Fecha de última consulta: 25 de agosto de 2015]

[36] PostgreSQL-es. Portal en español sobre PostgreSQL. [Internet] Disponible en: `http://www.postgresql.org.es/` [Fecha de última consulta: 25 de agosto de 2015]

[37] PostgreSQL Documentation. [Internet] Disponible en: `http://www.postgresql.org/docs/` [Fecha de última consulta: 25 de agosto de 2015]

[38] PostgreSQL Manuals. [Internet] Disponible en: `http://www.postgresql.org/docs/manuals/` [Fecha de última consulta: 25 de agosto de 2015]

[39] PostgreSQL Official Website. [Internet] Disponible en: `http://www.postgresql.org/` [Fecha de última consulta: 25 de agosto de 2015]

[40] PostgreSQL Support. [Internet] Disponible en: `http://www.postgresql.org/support/` [Fecha de última consulta: 25 de agosto de 2015]