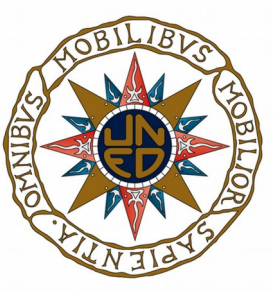


UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería en Tecnologías de la Información y la Comunicación

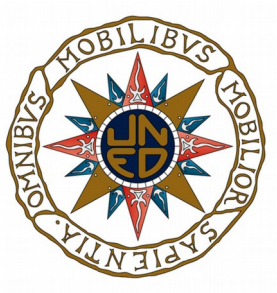
HERRAMIENTA DE SOPORTE A LA DECISIÓN PARA TRÁFICO POR CARRETERA

JOSÉ ISRAEL MARTÍNEZ YÁÑEZ

Dirigido por: ALFONSO URQUÍA MORALEDA

Codirigido por: CARLA MARTÍN VILLALBA

Curso: JULIO 2020



HERRAMIENTA DE SOPORTE A LA DECISIÓN PARA TRÁFICO POR CARRETERA

Proyecto de Fin de Grado en Ingeniería en Tecnologías de la Información y la Comunicación de modalidad específica

Realizado por: JOSÉ ISRAEL MARTÍNEZ YÁÑEZ

Dirigido por: ALFONSO URQUÍA MORALEDA

Codirigido por: CARLA MARTÍN VILLALBA

Fecha de lectura y defensa: ……

Resumen

Este documento describe el proceso de desarrollo de una herramienta software para apoyo a la decisión en lo que respecta al tránsito a través de redes de carretera.

Esta herramienta, la cual está diseñada para su uso en un centro de control de tráfico, aúna el cálculo de rutas entre poblaciones a través de la red de carreteras de una región concreta y la gestión de incidencias en las mismas. Se apoya en una base de datos conteniendo la cartografía esquemática (consistente en datos de tramos de carretera uniendo poblaciones con sus tiempos y distancias de recorrido).

El cálculo de rutas (usando el Algoritmo de Dijkstra) se hará de manera que tenga en cuenta las posibles incidencias que haya en las vías, y dando la posibilidad de elegir entre tres criterios distintos:

- Recorrido en distancia más corta.
- Recorrido menos tiempo.
- Recorrido optimizado, dando preferencia a las vías de mejor categoría.

Este cálculo de ruta ofrece al usuario instrucciones paso a paso al aproximarse a una población o a un cruce de carreteras.

La gestión de incidencias se realiza ofreciendo la posibilidad de cortar o reabrir al tráfico un tramo concreto de carretera. Estas incidencias generan:

- Cambios en la topología que son tenidos en cuenta para posteriores cálculos de ruta.
- La señalización provisional pertinente usando paneles luminosos para informar a los usuarios de la red de carreteras.

El sistema se apoya tanto en aplicaciones móviles para recibir información de los usuarios, como en los citados paneles luminosos para aportar información a los mismos.

El resultado será una sencilla herramienta software lista para ejecutarse en cualquier entorno, y que podrá ser fácilmente actualizable en lo que respecta a áreas a cubrir, y adaptable en lo que respecta a adición de funcionalidades.

Abstract

DECISION SUPPORT TOOL FOR ROAD TRAFFIC

This document describes the process of developing a software tool to support decision regarding traffic through road networks.

This tool, which is designed to be used in a traffic control centers, combines the calculation of routes between towns through the road network of a specific region and incident management. It is supported by a schematic cartography database (consisting of data from road sections linking towns with its proper journey times and distances).

The route calculation (using Dijkstra's algorithm) will be done in such a way that it takes into account possible incidents on the roads, and gives the possibility of choosing between three different criteria:

- Shortest distance.
- Quickest journey.
- Optimized route, giving preference to best quality roads.

This route calculation provides the user with step-by-step instructions when approaching a town or a road junction.

Incident management is done by offering the possibility of cutting or reopening traffic to a specific section of road. These incidents generate:

- Changes in the topology that are taken into account for subsequent route calculations.
- Pertinent provisional signage using illuminated panels to inform users of the road network.

The system relies both on mobile applications to receive information from users, and on the aforementioned illuminated panels to provide back information.

The result will be a simple software tool ready to run in any environment, and which can be easily updated in terms of areas to be covered, and adaptable in terms of adding features.

Palabras clave

Algoritmo voraz

Carretera

Red

Grafo

Ruta

Keywords

Greedy Algorithm

Road

Network

Graph

Route

Convenciones

En este documento se siguen las siguientes convenciones tipológicas:

Los textos en cursiva son citas literales. Al final de la cita se indicará una abreviatura de la referencia bibliográfica.

Lorem ipsum dolor sit amet, consectetur adipiscing elit (Cicerón, 45aD)

Las líneas en fuente courier representan comandos de consola.

```
sudo bash
```

Los fragmentos en fuente monospace representan fragmentos de código fuente, pseudocódigo, o resultados de consola.

```
public class Foo()
```

Índice de contenidos

Resumen.....I

Abstract.....II

Palabras clave.....III

Keywords.....IV

Convenciones.....V

Índice de contenidos.....VI

Índice de figuras.....IX

Índice de tablas.....XII

Índice de fragmentos de código.....XIII

Índice de resultados de consola.....XIV

Índice de algoritmos.....XV

Capítulo 1: Introducción, objetivos y estructura.....1

1.1 Introducción.....2

1.1.1 El Proyecto RutaSierra.....2

1.1.2 Ventajas frente a otros sistemas.....2

1.1.3 El centro de control.....3

1.1.4 Funcionamiento.....4

1.2 Objetivos.....4

1.3 Estructura.....6

Capítulo 2: Grafos: aplicación al modelado de redes de carreteras.....9

2.1 Introducción.....10

2.2 Grafos en programación orientada a objetos.....10

2.3 Algoritmos de búsqueda de camino más corto.....14

2.3.1 Algoritmos voraces.....14

2.3.2 El Algoritmo de Dijkstra.....15

2.4 Estándar de cartografiado.....25

2.4.1 Soporte.....25

2.4.2 Esquema.....27

2.5 Conclusiones.....28

Capítulo 3: Análisis.....29

3.1 Introducción.....30

3.2 Toma de requisitos.....30

3.2.1 Requisitos funcionales.....*31*
3.2.2 Requisitos prácticos.....*32*
3.2.3 Exclusiones.....*33*
3.3 Definiciones.....33
3.4 Funcionamiento.....34
3.4.1 Cálculo de itinerarios.....*34*
3.4.2 Gestión de incidencias.....*35*
3.4.3 Casos de uso.....*36*
3.5 Conclusiones.....37
Capítulo 4: Diseño.....39
4.1 Introducción.....40
4.2 Estructura de la aplicación.....40
4.2.1 Creación del grafo.....*41*
4.2.2 Formulario.....*44*
4.2.3 Recorrido del grafo.....*44*
4.3 Región a cartografiar.....46
4.4 Persistencia de datos.....47
4.4.1 Creación de la base de datos.....*47*
4.4.2 Esquema de la base de datos.....*48*
4.5 El ciclo de vida de la aplicación.....48
4.5.1 Desarrollo.....*49*
4.5.2 Integración continua.....*49*
4.5.3 Despliegue.....*50*
4.5.4 Operación.....*50*
4.5.5 Monitorización.....*50*
4.6 Conclusiones.....51
Capítulo 5: Implementación y pruebas.....53
5.1 Introducción.....54
5.2 Entorno de desarrollo utilizado.....54
5.2.1 Herramientas software.....*54*
5.2.2 Hardware.....*56*
5.3 Cartografiado.....56
5.4 Creación y conexión a la base de datos.....63
5.5 Desarrollo del algoritmo.....65
5.6 Recorrido del grafo.....72
5.7 Criterios de búsqueda.....79
5.8 Gestión de incidencias.....81
5.9 Empaquetado.....86
5.10 Problemas encontrados.....87

5.10.1 Versión de Java.....*87*
5.10.2 Limpieza de código.....*88*
5.10.3 Codificación regional.....*88*
5.11 Modificaciones sufridas.....90
5.11.1 Gestión de incidencias.....*90*
5.11.2 Tipología de la base de datos.....*90*
5.11.3 Categorización de rutas.....*90*
5.11.4 Recálculo del algoritmo.....*91*
5.12 Pruebas.....91
5.13 Conclusiones.....98
Capítulo 6: Planificación y costes del proyecto.....99
6.1 Introducción.....100
6.2 Planificación.....100
6.2.1 Etapas.....*100*
6.2.2 Diagramas.....*112*
6.3 Costes del proyecto.....113
6.4 Conclusiones.....113
Capítulo 7: Conclusiones y trabajos futuros.....115
7.1 Introducción.....116
7.2 Conclusiones.....116
7.3 Trabajos futuros.....117
Bibliografía.....120
Listado de abreviaturas.....122
Anexos.....125
Anexo A: Tabla de rutas.....126
Anexo B: Manual de despliegue.....129
B.1 Primeros pasos.....*129*
B.2 Actualización de la topología.....*129*
Anexo C: Manual de operación.....130
C.1 Cálculo de rutas.....*131*
C.2 Generación de incidencias.....*135*
C.3 Eliminación de incidencias.....*139*

Índice de figuras

Figura 2.1: ejemplo de grafo cíclico.....11
Figura 2.2: ejemplos de herencia de clases en el actual proyecto.....11
Figura 2.3: grafo acíclico a partir de la Figura 2.1.....14
Figura 2.4: ejemplo de esquema de red informática.....19
Figura 2.5: topología de red desde la perspectiva de R1.....20
Figura 2.6: señal de "todas direcciones" sobre la carretera M-501 (GSV).....21
Figura 2.7: grafo formado por ciudades y carreteras.....21
Figura 2.8: mapa de recubrimiento mínimo desde la perspectiva de uno de los nodos de la red.....24
Figura 3.1: diagrama de caso de uso de usuario.....36
Figura 3.2: diagrama de caso de uso de la gestión de incidencias.....37
Figura 4.1: esquema del patrón MVC.....40
Figura 4.2: diagrama de relaciones entre paquetes.....41
Figura 4.3: diagrama de clases del paquete 'db_management'.....42
Figura 4.4: ejemplo de grafo dirigido.....42
Figura 4.5: diagrama de clases del paquete 'node_management'.....43
Figura 4.6: diagrama de clases del paquete 'gui'.....44
Figura 4.7: diagrama de clases del paquete 'rutasierra'.....45
Figura 4.8: diagrama general de clases.....46
Figura 4.9: esquema de la base de datos.....48
Figura 4.10: modelo Entidad-Relación de la base de datos.....48
Figura 5.1: localización de la comarca en la Comunidad de Madrid (SO).....56
Figura 5.2: división municipal de la comarca (SO).....56
Figura 5.3: ejemplo de encrucijada entre la M-512 y la M-531 (OSM).....57
Figura 5.4: grafo uniendo poblaciones a través de encrucijadas.....58
Figura 5.5: grafo uniendo únicamente poblaciones.....58
Figura 5.6: señalización de poblaciones (GSV).....58
Figura 5.7: trazado de la carretera M-512 (OSM).....59
Figura 5.8: tramo de autovía de la M-501 (GSV).....60
Figura 5.9: carretera N-403 (GSV).....60
Figura 5.10: M-501 en su tramo como carretera (GSV).....60
Figura 5.11: carretera M-507 (GSV).....61

Figura 5.12: carretera M-855, carretera de un solo sentido (GSV).....62
Figura 5.13: carreteras cartografiadas (OSM).....63
Figura 5.14: carreteras que atraviesan Cadalso de los Vidrios (OSM).....65
Figura 5.15: diagrama del Algoritmo de Dijkstra.....71
Figura 5.16: ejemplo de ruta atravesando encrucijadas.....72
Figura 5.17: ejemplo de recorrido atravesando encrucijadas (GSV).....77
Figura 5.18: cruce entre la M-501 y la M-512 (GSV).....78
Figura 5.19: cruce entre la M-512 y la M-531 (GSV).....78
Figura 5.20: traza por criterio óptimo.....80
Figura 5.21: traza por criterio rápido.....80
Figura 5.22: traza por criterio corto.....80
Figura 5.23: resultado de las tres trazas superpuestas (OSM).....80
Figura 5.24: teleindicador previo a un cruce con la M-501 (GSV).....81
Figura 5.25: ejecución normal del cálculo de itinerario.....82
Figura 5.26: cálculo de itinerario sin incidencia (GSV).....83
Figura 5.27: cálculo de itinerario con incidencia (GSV).....83
Figura 5.28: ejecución del cálculo de itinerario teniendo en cuenta una incidencia. .84
Figura 5.29: conexión única de Valdemaqueda con la red de carreteras (GSV).....85
Figura 5.30: ejemplo de incidencia que aísla un nodo.....86
Figura 5.31: error de inicialización de JavaDB.....87
Figura 5.32: aplicación corriendo en Windows.....89
Figura 5.33: traza entre destinos en extremos de la región.....92
Figura 5.34: traza cambiando de criterio.....93
Figura 5.35: comparación de las dos rutas en un mapa (OSM).....93
Figura 5.36: traza de ruta afectada por incidencia.....94
Figura 5.37: alternativas propuestas en caso de incidencia (OSM).....95
Figura 5.38: puntos incomunicados por acumulación de incidencias.....96
Figura 5.39: traza tras restablecimiento de incidencia.....97
Figura 5.40: análisis de rendimiento.....98
Figura 6.1: vista previa de primer entorno gráfico creado.....101
Figura 6.2: plano-grafo.....102
Figura 6.3: vista de la base de datos de rutas.....103
Figura 6.4: resultado de consulta usando código en Java.....104
Figura 6.5: primera versión del entorno gráfico.....106

Figura 6.6: resultado en la guía de ruta.....107
Figura 6.7: aspecto de la aplicación con la gestión de incidencias.....108
Figura 6.8: interfaz gráfico de la versión 0.1.....109
Figura 6.9: resultado de la herramienta SonarLint.....110
Figura 6.10: portal Javadoc.....111
Figura 6.11: diagrama de Gantt de planificación del proyecto antes de iniciarlo.....112
Figura 6.12: diagrama de Gantt a la finalización del proyecto.....112
Figura C.1: ventana de la aplicación.....130
Figura C.2: despliegue del menú origen.....131
Figura C.3: despliegue del menú destino.....132
Figura C.4: selección de criterio de cálculo de ruta.....133
Figura C.5: ruta trazada.....134
Figura C.6: selección de carretera afectada por incidencia.....135
Figura C.7: lista de tramos de una carretera.....136
Figura C.8: incidencia generada.....137
Figura C.9: incidencia generada sin alternativas encontradas.....138
Figura C.10: menú de incidencias.....139
Figura C.11: incidencia eliminada.....140

Índice de tablas

Tabla 2.1: tablas de enrutamiento iniciales de las poblaciones de la Figura 2.6.....22
Tabla 2.2: tablas de enrutamiento tras primer intercambio de rutas.....22
Tabla 2.3: tablas de enrutamiento tras segundo intercambio de rutas.....23
Tabla 2.4: tablas de enrutamiento tras segundo intercambio de rutas.....24
Tabla 2.5: ejemplo de lista de carreteras que atraviesan una población.....25
Tabla 2.6: ejemplo de lista de carreteras que atraviesan otra población.....26
Tabla 2.7: ejemplo de lista de carreteras.....26
Tabla 2.8: tabla unificada de destinos.....27
Tabla 2.9: esquema final de la tabla de arcos.....28
Tabla 5.1: tabla de rutas directamente conectadas de Cadalso de los Vidrios.....65
Tabla 5.2: tabla de enrutamiento completa de Cadalso de los Vidrios, con el criterio optimizado.....67
Tabla 5.3: tabla de enrutamiento completa de Cadalso de los Vidrios, con el criterio por distancia.....68
Tabla 5.4: tabla de enrutamiento completa de Cadalso de los Vidrios, con el criterio por tiempo.....69
Tabla 5.5: categorización de las rutas.....79
Tabla 6.1: tiempo empleado en el proyecto.....113
Tabla A.1: tabla de rutas de la cartografía.....128

Índice de fragmentos de código

Código 2.1: instancia de población a partir de una clase nodo.....12
Código 2.2: creación estática y dinámica de nodos.....12
Código 2.3: inserción de arcos en un nodo.....13
Código 5.1: creación de la base de datos de rutas.....64
Código 5.2: ingesta de datos desde el archivo de cartografía a la base de datos.....64
Código 5.3: conexión a la table de datos.....64
Código 5.4: consulta lanzada a la base de datos.....65
Código 5.5: código Java del algoritmo.....70
Código 5.6: función de recorrido del grafo.....74
Código 5.7: cadena de conexión a la base de datos embebida.....87
Código 5.8 solución de problema de permisos en Java.....88

Índice de resultados de consola

Output 5.1: cabecera de información de ruta.....74
Output 5.2: primera iteración del recorrido del grafo.....75
Output 5.3: inicio de la segunda iteración.....75
Output 5.4: final de la segunda iteración.....75
Output 5.5: inicio de la tercera iteración.....76
Output 5.6: final de la tercera iteración.....76
Output 5.7: inicio de la quinta iteración y final de la función.....77

Índice de algoritmos

Algoritmo 2.1: Algoritmo de Dijkstra (Araujo et al., 2011).....16

Capítulo 1: Introducción, objetivos y estructura

1.1 Introducción

1.1.1 El Proyecto RutaSierra

RutaSierra es un proyecto de implementación de Red Inteligente de Carreteras. Consiste en un conjunto de aplicaciones, sensores y dispositivos que de por sí solos sean capaces de regular el tráfico de una determinada red de carreteras, con o sin intervención de operadores. Su nombre viene del proyecto piloto, que se pretende implementar en la comarca Sierra Oeste de Madrid.

La finalidad es que mediante la detección automática de flujos de tráfico y de posibles incidentes se pueda influir sobre el mismo mediante el uso de señalización luminosa y de semáforos capaz de:

- Redirigir tráfico por rutas alternativas en caso de congestión o incidente.
- Alterar límites de velocidad en función del flujo de tráfico detectado y de predicciones estadísticas para prevenir congestiones.
- Interactuar con los usuarios de la vía mediante aplicación móvil, y mostrando por los paneles luminosos guía de itinerario.
- Optimizar los tiempos de espera en semáforos.

Dependiendo del éxito o fracaso del proyecto piloto, la intención final es trasladarlo a redes de alta capacidad de cara a optimizar los flujos de tráfico y minimizar, en la medida de lo posible, las retenciones, los accidentes, e incluso la contaminación.

1.1.2 Ventajas frente a otros sistemas

De cara a los usuarios, este sistema sirve para, dada una red de carreteras, elegir el mejor itinerario para llegar de una población a otra, según los criterios del usuario. Aunque en la actualidad cualquier dispositivo con conexión a internet es capaz de trazar una ruta entre poblaciones teniendo en cuenta incidencias de tráfico, la gran ventaja de este sistema reside en evitar distracciones al usuario teniendo que mirar su dispositivo móvil o el sistema de navegación de su vehículo, sino que será la propia señalización luminosa de la carretera la que se adaptará a su paso para mostrarle las indicaciones que necesite en ese momento.

Además, frente a otras aplicaciones más populares que tienen en cuenta congestiones e incidentes en la ruta como pueden ser Google Maps o Waze, ésta, al estar integrada en el propio sistema de gestión de la red de carreteras, ofrece una respuesta más rápida, tanto para la generación como para

la eliminación de incidencias, ya que no depende de que sean los propios usuarios o corporaciones externas las que envíen los avisos, sino que es el mismo sistema el que genera o elimina las incidencias.

De cara a la gestión de tráfico, RutaSierra está orientada a la gestión autónoma (mínima intervención humana), intentando reducir al máximo el tiempo de respuesta frente a un incidente, logrando con esto, en la manera de lo posible, que dicho incidente tenga el menor impacto posible en el resto del tráfico. Además, su concepción proactiva, frente a la reactiva de los centros de control de tráfico actuales, busca anticiparse a una gran mayoría de incidentes que, con anticipación, son evitables.

1.1.3 El centro de control

Una pieza importante de este proyecto, que es de la que trata este documento, consiste en el sistema de gestión de incidencias y guía de ruta. Consiste en un sistema de apoyo a la gestión de un centro de control de carreteras. Este sistema se concibe para:

- Apoyar la labor de los operadores de dicho centro de control, ofreciendo información puntual y actualizada. Los propios operadores son los que administran las incidencias que gestiona este sistema. Sin embargo, el sistema de detección de flujos e incidentes también puede introducir o eliminar incidencias en el sistema. Las incidencias generan mensajes que se muestran automáticamente en los paneles luminosos, y también habilitan la señalización para cortar el tráfico en el tramo afectado y ofrecer alternativas.
- Ofrecer guía paso a paso a conductores mediante el uso de aplicación móvil para detectar su posición y de los paneles luminosos para mostrar mensajes con la guía de ruta. Los operadores también utilizar el sistema para realizar cálculos manuales de itinerario. A su vez, mostrar información genérica sobre incidentes a todos los usuarios, aunque no usen la aplicación móvil.

Está concebido como un sistema monolítico (pensado para que el mismo equipo sirva de cliente y servidor a la vez) en esta fase del proyecto, que serviría como prueba de concepto para una ampliación basándose en un sistema distribuido cliente-servidor en el que varios clientes pudiesen interactuar simultáneamente sin interferirse con el mismo servidor.

1.1.4 Funcionamiento

El núcleo del sistema se basa en el algoritmo que sirve de ayuda a elegir el mejor camino entre poblaciones basándose en tres criterios predeterminados: itinerario más corto, más rápido, u optimizado basándose en la categoría de las vías por las que transita. A su vez, se demuestra que en el caso de una modificación de la topología (incidencia) el sistema es capaz de recalcular basándose en la nueva topología propuesta, siempre que sea posible. También ofrece al operador un sistema de control sobre las incidencias que puedan surgir.

La solución obtenida es un sistema estructurado en varios bloques:

- Motor de decisión, *backend* desarrollado en Java, y núcleo del sistema en general, alrededor del cual se han ido desarrollando el resto de subsistemas.
- Interfaz gráfica, *frontend* desarrollado en Java con la ayuda de NetBeans, que sirve para la interacción con el operador y/o usuario del sistema.
- Base de datos, en DerbyDB, embebida, portable y ligera, que ofrece una gran versatilidad de cara a actualizaciones y ampliaciones de la topología.

1.2 Objetivos

Este proyecto tiene como objetivo diseñar y desarrollar el sistema descrito en el apartado 1.1.3, que sirva de apoyo para la gestión del centro de control de carreteras a la hora de gestionar incidencias y mensajes en paneles luminosos. Concretamente, es la pieza encargada de la gestión de los cambios de topología.

Disponiendo de la topología del área a cubrir en forma de grafo dirigido cíclico, dicho apoyo consiste en:

- Poder indicar a los usuarios de la aplicación móvil y a los operadores en cada momento cuál es la mejor manera de llegar de un destino a otro dentro del área mencionada. Se usará un algoritmo voraz para calcular el mejor camino (según el criterio elegido) y mostrarlo en forma de indicaciones paso a paso. Además, servirá para avisar de las incidencias a todos los conductores en general, e indicarles un itinerario alternativo.
- Gestión de incidencias que agilice los procedimientos a ejecutar, entre ellos, realizar cambios en el grafo y ejecutar el algoritmo sobre el mismo.

Respecto al criterio de selección de ruta, se dará a elegir al usuario entre ruta más corta o más rápida, basándose en las métricas recolectadas sobre distancia y tiempos de recorrido, y además se sugerirá un criterio optimizado, basándose en el criterio de distancia, pero dando preferencia al uso de vías de mayor categoría.

Es importante que la toma de decisiones se apoye en un sistema que proporcione información realmente fiable, más que basarse en tomar decisiones aleatorias. A su vez, es importante que la computación de las posibilidades no suponga un consumo exhaustivo de recursos. De ahí la necesidad del uso de un algoritmo voraz.

El proceso de cartografiado consistirá en plasmar en un soporte un conjunto de datos sobre tiempos y distancias de recorrido. El sistema usará estos datos para construir una base de datos de topografía, y ésta se utilizará para la creación del grafo que representará la red de carreteras. Se debe desacoplar la lógica del sistema de los datos de cartografía. Lo que se busca con esto es que el mantenimiento de la topología, en lo que se refiere a ampliaciones, actualizaciones, o incluso el uso del sistema en distintos territorios, sea un proceso sencillo y sin necesidad de refactorizar el código. De ahí que se busque que sean archivos separados del código fuente. El hecho de usar una base de datos como soporte intermedio responde un ahorro de recursos evitando cargar todos los datos en memoria, y a que se optimicen las consultas por parte de la aplicación evitando que se realicen las consultas sobre el soporte externo.

El desarrollo del proyecto se estructura en las siguientes fases:

- Toma de requisitos del sistema, en las que se basará el diseño de alto nivel.
- Elección de la tecnología a usar para su desarrollo y adecuación del diseño de alto nivel a uno de bajo nivel, de cara a guiar la implementación.
- Instalación del software necesario para el desarrollo.
- Estudio, documentación y pruebas para familiarizarse y aprender a usar el IDE y la base de datos.
- Creación de un repositorio donde almacenar las versiones del código desarrollado.
- Definición de un estándar para el cartografiado y cartografiado de la región piloto. Este estándar servirá para estructurar la arquitectura de datos que se manejará en el software.
- Creación de una base de datos en base al estándar creado para el cartografiado.

- Ingesta de los datos de cartografía en la base de datos.
- Elaboración del producto software usando un diseño modular que interactúe con la base de datos.
- Elaboración de un interfaz que ofrezca al usuario la posibilidad de operar con facilidad el sistema.
- Realización de una batería de pruebas que sirva para detectar y depurar errores, comprobar el funcionamiento, y realizar ajustes.
- Empaquetado del producto.
- Despliegue del producto final.
- Documentación del proceso completo.

1.3 Estructura

Esta memoria consta de siete capítulos y tres anexos:

Los capítulos 1 y 2 muestran la teoría previa al comienzo de la ejecución del proyecto, sus expectativas, objetivos y los elementos a usar para llevarlo a cabo. Los capítulos 3 y 4 muestran los aspectos relacionados con el análisis y diseño, desde la toma de requisitos a la definición de estructura y elementos que compondrán el producto. Los capítulos 5 a 7 muestran el resultado, el proceso atravesado hasta obtener el entregable final, y las conclusiones alcanzadas durante todo el proceso.

El Capítulo 1, “Introducción, objetivos y estructura”, muestra la idea del software que se pretende construir, cómo encaja dentro del proyecto en el que está enmarcado, qué elementos contendrá, qué pasos se van a seguir para obtenerlo, y la estructura de este documento.

El Capítulo 2, “Grafos, aplicación al modelado de redes de carreteras”, describe el uso de grafos en este proyecto, cómo crearlos como estructura de datos usando Java, los algoritmos usados para recorrerlos, incluyendo una aplicación práctica en redes de computadoras usando el protocolo OSPF y el paralelismo con redes de carreteras; y su representación y almacenaje en una base de datos para su posterior uso por parte de la aplicación.

El Capítulo 3, “Análisis”, realiza una descripción del diseño de alto nivel de la aplicación, teniendo en cuenta sus requisitos, características, partes, actores y posibles usos. A partir de este análisis se

realizará el diseño de bajo nivel que será el que será plasmado en código por parte del equipo de desarrollo.

El Capítulo 4, “Diseño”, especifica el diseño a bajo nivel, realizado a partir de la toma de requisitos inicial. Mostrará como desarrollar la base de datos y la estructura de clases deseada para el software. Indicará la manera de realizar el desarrollo de la misma, la metodología a usar y los objetivos a cumplir.

El Capítulo 5, “Implementación y pruebas”, habla del despliegue en sí del proyecto. Se enumeran las herramientas software y hardware que forman parte del entorno de desarrollo. Se describen los pasos seguidos, los problemas encontrados, los cambios sufridos por el proyecto desde su concepción inicial, y las pruebas realizadas para comprobar su correcto funcionamiento.

El Capítulo 6, “Planificación y costes del proyecto”, describe paso a paso cómo se ha ido desarrollando el proyecto, se comparan las fases con la planificación inicial, y se especifica el coste del proyecto.

El Capítulo 7, “Conclusiones y trabajos futuros”, habla de las conclusiones al finalizar el proyecto, y de las posibles ampliaciones que puede sufrir el producto entregado.

El Anexo A, “Tabla de rutas”, contiene la cartografía utilizada para este despliegue.

El Anexo B, “Manual de despliegue”, describe los procedimientos necesarios para la puesta en marcha del sistema.

El Anexo C, “Manual de usuario”, contiene las instrucciones para el uso de la aplicación.

El CD adjunto contiene:

- Una copia de este documento.
- El archivo ejecutable “RutaSierra.jar”, que es el programa del que se habla en este documento.
- El directorio “lib”, con las librerías de DerbyDB para que funcione la base de datos.
- El directorio “doc”, con el portal Javadoc del programa.
- El directorio “src”, con el código fuente de las clases que componen el programa.
- Los archivos “categorias.csv” y “rutas.csv”, conteniendo los datos de cartografía.

Capítulo 2: Grafos: aplicación al modelado de redes de carreteras

2.1 Introducción

Los grafos son un elemento bastante común en aplicaciones informáticas. Son estructuras de datos consistentes en un conjunto de nodos, y otro conjunto de arcos entre parejas de nodos. Los arcos representan caminos o conexiones entre los nodos. Se usan para almacenar datos que están relacionados, como en bases de datos, o representar estados de un autómata. También se usan para representar elementos físicos como circuitos electrónicos o redes de computadores (Koffman y Wolfgang, 2010).

Otro ejemplo de uso de grafos es una red de poblaciones unidas por carreteras, que es de lo que trata precísamente este proyecto.

En este capítulo se describe el uso que se hará de grafos para representar dicha red y, además, aportar una solución al desafío que supone encontrar el mejor camino para llegar de un origen a un destino atravesando una malla de nodos conectados por arcos, los cuales tienen un conjunto determinado de métricas. En concreto veremos:

- Cómo representar grafos con el paradigma orientado a objetos.
- Qué algoritmos existen para recorrerlos, y de sus usos en otros ámbitos.
- Qué soporte y formato tendrán a la hora de plasmar la topología real en archivos de cartografía.

2.2 Grafos en programación orientada a objetos

El paradigma orientado a objetos define la abstracción como un modelo de la realidad. Basándose en dicha abstracción el modelo propone maneras de representar dicha realidad, usando las clases como objetos abstractos (fruta, por ejemplo) y las instancias como objetos concretos (manzana, como instancia de la clase “fruta”). La instancia ofrece muchos más detalles (tiene piel, pepitas) de los que es capaz de ofrecer la clase, al ser más específica. La clase, al ser más genérica, sirve para definir a todo el conjunto de frutas (sandías, fresas, naranjas o plátanos) (Barnes y Kölling, 2007).

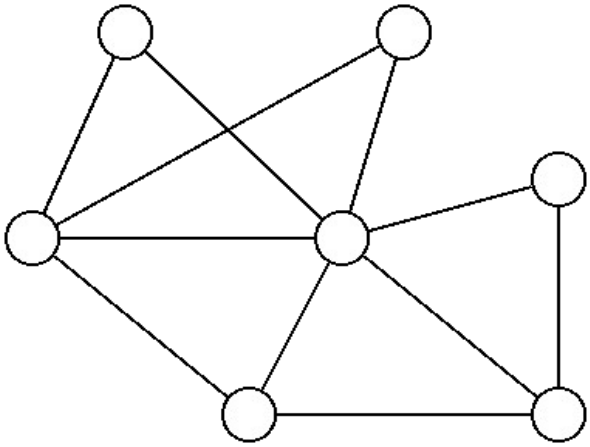


Figura 2.1: ejemplo de grafo cíclico

Dada la naturaleza del proyecto, que se trata básicamente de unir poblaciones mediante carreteras, mediante un grafo cíclico como el de la Figura 2.1, se ve un claro ejemplo de aplicación del paradigma orientado a objetos, frente a otros paradigmas que quizás no son tan apropiados para este proyecto, como el funcional, el lógico, o el imperativo (Cerrada y Collado, 2010).

En la aplicación que se daría a este paradigma de convertir una red de carreteras en un grafo, en los grafos se pueden distinguir dos clases: nodos y arcos. Como se verá más adelante en su aplicación al mundo real, el nodo de un grafo puede representar bien una población o bien una encrucijada, con lo que tenemos ejemplo similar al visto antes con la fruta de herencia con respecto a clase/objeto (véase la Figura 2.2) (Barnes y Kölling, 2007). El código 2.1 muestra un ejemplo en Java de instanciación a partir de una clase abstracta.

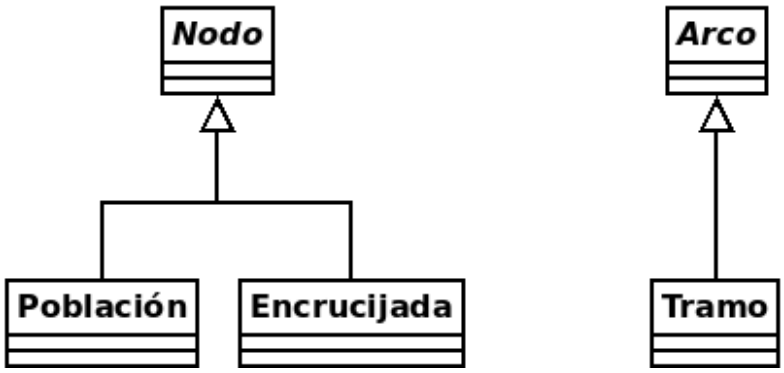


Figura 2.2: ejemplos de herencia de clases en el actual proyecto

A su vez, un arco representa un tramo de carretera que une dos nodos, sean del tipo que sean. Un tramo de carretera puede a su vez tener muchas categorizaciones: Si es de peaje o no, si es unidireccional o bidireccional, etc. Aunque para el caso que vamos a tratar usaremos únicamente un tipo de arco para construir el grafo, para construirlos se usará el mismo esquema de herencia de cara a dejar abierta la posibilidad de usar diferentes tipos de vías en el futuro.

```
public abstract class Nodo {
    String nombre;
    public Nodo(String nombre){
        this.nombre = nombre;
    }
}
```

```
public class Poblacion extends Nodo{
    public Poblacion(String nombre) {
       super(nombre);
    }
}
```

Código 2.1: instancia de población a partir de una clase nodo

Hay elementos de los que se sabe el número exacto de instancias que se van a crear (tres botones en el interfaz, una conexión a base de datos, un grafo...), y crearlas de manera estática no supone ningún problema, y de hecho facilita mucho el acceso a las mismas. Sin embargo, la manera de crear los elementos del grafo, lógicamente, no puede ser estática. No se sabe de entrada el número de elementos que va a tener (aunque se podría saber), pero puesto que por un lado se busca la escalabilidad (el número de nodos podría variar), y por otro, evitar la repetición de código, se debe buscar una manera de crear dinámicamente los nodos a partir de una lista (Louden, 2004).

```
Nodo Springfield = new Poblacion("Springfield");
Nodo Shelbyville = new Poblacion("Shelbyville");
Nodo Brockway = new Poblacion("Brockway");
Nodo CapitalCity = new Poblacion("Capital City");
Nodo BernacleBay = new Poblacion("Bernacle Bay");
Nodo CypressCreek = new Poblacion("Cypress Creek");
Nodo Ogdenville = new Poblacion("Ogdenville");
Nodo NorthHavenbrook = new Poblacion("NorthHavenbrook");
```

```
List<Nodo> todosNodos = new ArrayList<>();
ResultSet nodosDB;
nodosDB.first();
do {
  todosNodos.add(new Poblacion(nodosDB.getString("Nombre")));
} while (nodosDB.next());
```

Código 2.2: creación estática y dinámica de nodos

El Código 2.2 muestra un ejemplo de creación estática de nodos, y su equivalente dinámica usando el resultado de una consulta a una base de datos. El ahorro en líneas de código es evidente. Luego, el modo de acceder a los elementos del nodo se realiza iterando los elementos del *ArrayList*. Usando un patrón similar a *Abstract Factory*, se pueden crear dinámicamente los elementos que forman el grafo a partir de una lista (Gamma et al., 2003).

A la hora de crear la estructura de datos, se creará una lista de adyacencia mediante una estructura de datos basándose en nodos que contienen un número determinado de arcos (1...n), y a su vez, estos arcos conectan dos nodos (2...2) (Drozdek, 2007). Nuevamente, puesto que no se sabe el número de arcos que va a contener un nodo, se debe usar una manera dinámica de crearlos dentro del nodo.

```
public abstract class Nodo {
    String nombre;
    List<Arco> tramos new ArrayList<>();
    public Nodo(String nombre){
        this.nombre = nombre;
        ResultSet rutasDB;
        rutasDB.first();
        do {
            tramos.add(new Tramo(destino, rutasDB.getFloat("longitud"),
                                 rutasDB.getInt("tiempo"));
        } while (rutasDB.next());
    }

}
```

Código 2.3: inserción de arcos en un nodo

El Código 2.3 muestra cómo quedaría la clase *Nodo* con sus correspondientes arcos ya creados. Respecto al grafo inicial, esta aproximación realmente creará el doble de objetos *Arco* de los que existen realmente en el grafo, ya que el mismo arco se creará en los nodos de origen y de destino (a diferencia de un árbol, en el grafo los nodos a ambos lados de un arco tienen la misma jerarquía, no hay “hijos” ni “padres”). Esto viene dado por el hecho de que existe la posibilidad de que haya carreteras de un único sentido. Cada objeto *Arco* realmente representará un sentido de un tramo, y en el supuesto mencionado de carreteras unidireccionales sólo se crearía un arco en el nodo de origen.

La manera de iterar esta estructura, entonces, sería accediendo al nodo, obtener su lista de arcos, en el arco indicado obtener los datos interesantes (longitud, tiempo…), y el nombre del siguiente nodo, que es en el que habrá que buscar. La estructura sería similar a un árbol de dos niveles con tantos “padres” como nodos haya, y tantos “hijos” como arcos tenga cada padre (Gortázar et al., 2012).

2.3 Algoritmos de búsqueda de camino más corto

Los grafos en sí presentan un desafío a la hora de recorrerlos. Cuantos más nodos y arcos tiene el grafo, más posibilidades ofrece para encontrar un camino entre dos de sus nodos, con lo que calcular cuál de ellos es el mejor se complica según crece el grafo. Tomando como ejemplo el grafo de la Figura 2.1, si se pretende ir del nodo más cercano a la esquina inferior derecha al que está más próximo a la esquina superior izquierda, a simple vista parece que el mejor camino se puede recorrer en dos saltos. Sin embargo, sin un algoritmo que optimice las posibilidades, simplemente por mero azar (desde el nodo origen elegir aleatoriamente uno de sus arcos, ir al destino de dicho arco y si no es el nodo de destino elegir aleatoriamente otro arco excluyendo el arco previamente recorrido, y repetir la operación hasta que se alcance el destino) hay un riesgo bastante alto de caer en un bucle, generando un costo computacional que puede ser inasumible cuando el grafo alcance cierto tamaño. La aproximación más común para resolver este problema es crear un árbol (que viene a ser un grafo acíclico), como el de la Figura 2.3, para evitar dichos bucles.

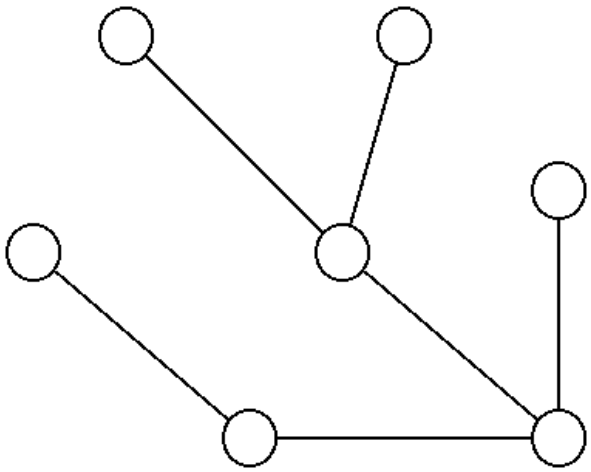


Figura 2.3: grafo acíclico a partir de la Figura 2.1

Aún así, la creación de dicho árbol no garantiza que se esté eligiendo el mejor camino para llegar de un nodo a otro. Puesto que existen varios posibles árboles para buscar una solución al problema planteado, hay que buscar cuál de ellos es el que representa el mejor camino para llegar a cada uno de los nodos. Además, hay que tener en cuenta que cada nodo tendrá su propia versión de dicho árbol, que no tiene por qué coincidir con el del resto de nodos. Para ello, nuevamente, en vez de calcular todos los árboles posibles para todos los nodos, se debe optimizar dicha búsqueda.

Para resolver este problema existe una amplia variedad de algoritmos para trabajar con árboles y grafos, aunque dependiendo del tipo de problema unos son más idóneos que otros.

2.3.1 Algoritmos voraces

Los algoritmos voraces se usan en situaciones en las que se, habiendo varias soluciones, se debe buscar la óptima. Se basan en evitar evaluar situaciones por las que ya se haya pasado y concentrar y reducir progresivamente las posibles soluciones. En base a eso, cada candidato es evaluado. Si se rechaza como posible se descarta, y si se declara como posible ya no se vuelve atrás. La principal

característica de los algoritmos voraces es que nunca deshace una decisión ya tomada. Esta característica los hace altamente eficientes (Araujo et al., 2011).

Los algoritmos voraces más conocidos son el de Prim, el de Kruskal, y el de Dijkstra. Estos algoritmos se usan como solución a dos problemas concretos:

- Encontrar un árbol de recubrimiento mínimo. Es aquél en el cual la suma de las métricas de sus aristas es la mínima posible, y a su vez, todos sus nodos quedan conectados. En un grafo de *n* nodos este árbol debe tener *n*-1 aristas.
- Calcular el camino de coste mínimo entre un nodo y los demás. Dado un grafo en el que todos los nodos están conectados a un nodo origen, determinar la longitud del camino mínimo desde dicho nodo origen hacia el resto de nodos (Araujo et al., 2011).

Los algoritmos voraces suelen utilizarse para resolver problemas de optimización. Ante un desafío que incluya varias soluciones (solución factible), se tiende a buscar la óptima. Algunos ejemplos son:

- Buscar la secuencia óptima para ejecutar una lista de tareas.
- Buscar el mejor camino entre poblaciones por carretera, o entre estaciones de metro.
- Juegos de estrategia, como resolución de laberintos, o el ajedrez.
- Cambio de divisa.
- Calcular recorridos a través de varias localizaciones.
- Redes informáticas.

2.3.2 El Algoritmo de Dijkstra

En el caso de los algoritmos de Prim y Kruskal, el resultado es un único árbol mínimo de recubrimiento. Son muy eficientes en el cómputo, ya que alcanzan la solución enseguida, pero en la práctica serían eficaces si, por ejemplo, lo que buscásemos fuese unir todas las poblaciones de una región usando el mínimo posible, como, por ejemplo, para hacer un despliegue de fibra óptica. En el ejemplo de la Figura 2.3, el nodo de la izquierda y el del centro en el grafo original cuentan con un arco que los une. Sin embargo, una vez trazado el árbol de recubrimiento mínimo resulta que para ir de uno a otro hay que atravesar dos arcos más. Es decir, que el árbol de la figura 2.3 sólo es realmente práctico para el nodo de la esquina inferior izquierda. Usando el Algoritmo de Prim se

puede escoger un nodo concreto como referencia para empezar a trazar el árbol, con lo que podríamos ejecutar el algoritmo individualmente para cada nodo. Sin embargo, el resultado siempre buscaría la optimización a nivel del total del grafo, no de cada nodo, habiendo pequeñas diferencias entre los árboles creados para cada nodo. Por no hablar de que, además, se repetirían de manera inútil muchas operaciones.

Edsger W. Dijkstra (Róterdam, 1930-2002) describió por primera vez en 1959 el algoritmo de caminos mínimos. Este algoritmo explora todos los caminos desde un vértice hasta el resto de vértices en un grafo con pesos en los arcos, para obtener el camino más corto hacia cada uno de ellos.

El Algoritmo 2.1 contiene la matriz *VectorNat*, que contiene las métricas, y el array *C*, que contiene los nodos. Se asignan las métricas a los nodos y se comparan todos con todos, sumando a la métrica lo anunciado por el nodo comparado. En el momento que se recibe una métrica inferior se sustituye la métrica del nodo por la nueva métrica, y se van sacando nodos del array. El algoritmo finaliza cuando ya no hay más nodos que comparar.

```
tipo VectorNat = matriz[0..n] de natural

fun Dijkstra (G = <N,A>: grafo): VectorNat
  var
    especial: VectorNat
    C: conjunto de nodos
  fvar
  C={2,3,...,n}
  para i <- 2 hasta n hacer
    especial [i] <- Distancia(1,i)
  fpara
  mientras C contenga más de 1 nodo hacer
    v <- nodo ∈ C que minimiza especial[v]
    C <- C\{v}
    para cada w ∈ C hacer
      especial[w] <- min(especial[w], especial[v] + Distancia(v,w))
    fpara
  fmientras
  dev especial[]
ffun
```

Algoritmo 2.1: Algoritmo de Dijkstra (Araujo et al., 2011)

Con una sola ejecución del algoritmo se crea un árbol de recubrimiento mínimo “personalizado” para cada uno de los nodos. Computacionalmente es más pesado que cualquiera de los otros dos algoritmos, pero resulta más efectivo para encontrar la solución al problema planteado. En este

caso, que desde cada población se pueda encontrar el mejor camino para llegar al resto a través de una red de carreteras.

Coste

Mientras los algoritmos de Prim y Kruskal tienen menor coste computacional que el de Dijkstra, la gran ventaja de este último reside en que es capaz de encontrar el camino más corto para cada caso en vez de construir un único árbol mínimo de expansión, como los otros dos.

Respecto a su coste, en la inicialización tiene un consumo de O(n). El bucle **mientras** y el bucle **para** se ejecutarán n-2 veces. El tiempo requerido por el algoritmo es O(n^2).

Se puede usar un montículo de mínimos para almacenar los nodos para los que no se ha determinado el camino de coste mínimo. Contendría los nodos de *C*, y en su raíz estaría el que hace mínimo el valor de *especial*[]. El uso de este montículo depende de si el grafo es denso o disperso. Si es denso, su coste estará en O($n^2 \log n$), mientras que si es disperso su coste estará en O($a \log n$), siendo a el número máximo de veces que será necesario usar la función *flotar* en el montículo. Evidentemente, merece la pena usar el montículo si se sabe que el grafo es disperso (Araujo et al., 2011).

En comparación con otros algoritmos, el de Kruskal tiene una complejidad de O($n \log n$), mientras que el de Prim está en O(n^2), y si se usa un montículo de Fibonacci, en O((v+n) log n), siendo v el número de arcos.

Uso del Algoritmo de Dijkstra en OSPF

En redes informáticas de tamaño medio-grande, en las que se requiere un cierto grado de redundancia y flexibilidad para cambios de topología, es normal el uso de protocolos de enrutamiento dinámico.

Estos protocolos se basan en la premisa de que para ir desde un nodo a otro de la red es necesario atravesar una malla de enrutadores conectados. Esta malla representa un grafo cíclico, con la particularidad de que en redes informáticas los bucles, sencillamente, no pueden suceder. Para evitar estos bucles se construyen árboles de expansión, o grafos acíclicos, para conectar todos los nodos. Otra particularidad de estos protocolos es la respuesta a cambios de topología. Ante un cambio de topología, ya sea por avería, o por ampliaciones o reducciones de la propia red, los enrutadores involucrados propagan el cambio detectado al resto hasta que todos han sido informados del

cambio. Esta respuesta ante cambios de topología es aprovechada en túneles de proveedores como MPLS, donde se delega en un IGP para que el cliente pueda propagar sus prefijos a lo largo del túnel. Lógicamente, también estos protocolos se utilizan para asegurar un camino determinista entre dos puntos de la red, a ser posible, el más eficiente (aunque esta característica no se da en todos los protocolos) (Forouzan, 2007).

Los protocolos más populares son: STP, utilizado en la capa 2 de las redes; RIP, uno de los protocolos más sencillos; EIGRP, bastante más complicado ya que usa distintas métricas, propietario de Cisco Systems; BGP, utilizado en redes interempresariales, y, básicamente, el protocolo que mueve Internet; y OSPF, quizá el más usado en redes intraempresariales, que usa el protocolo de Dijkstra como motor principal.

El cálculo delega el conjunto de redes de áreas internas asociadas a un área. Un enrutador calcula el árbol de camino más corto tomándose a sí mismo como raíz. La formación del árbol de camino más corto se hace en dos etapas. En la primera, sólo se consideran los enlaces entre enrutadores y redes de tránsito. Usando el Algoritmo de Dijkstra se forma un árbol a partir de este grupo de la base de datos de estados de enlace. En la segunda etapa se van añadiendo hojas al árbol considerando los enlaces a redes finales.

[…] La base de datos de estado de enlace del área se representa como un grafo dirigido. Los vértices del grafo son enrutadores, redes de tránsito y áreas finales. La primera etapa del procedimiento sólo afecta a los vértices de tránsito (enrutadores y redes de tránsito) y sus enlaces conectados. […]

A cada iteración del algoritmo hay una lista de vértices candidatos. Aparecen caminos desde la raíz a esos vértices, que no son necesariamente los más cortos. Sin embargo, los caminos al vértice candidato más próximo a la raíz sí que garantizan ser los más cortos. Este vértice se añade al árbol de camino más corto, eliminado de la lista de candidatos, y sus vértices adyacentes se examinan para posibles añadiduras o modificaciones de la lista de candidatos. El algoritmo itera otra vez, y termina cuando la lista de candidatos está vacía (OSPF).

Similitudes entre redes informáticas y redes viarias

El enfoque que aporta la implementación del Algoritmo de Dijkstra para OSPF resulta muy similar al que se busca para implementar en una red de carreteras, especialmente en la respuesta ante

cambios de topología. De ahí que se vaya a tratar de imitar el comportamiento de los enrutadores en una red OSPF para los nodos que componen el grafo.

En ambas situaciones se puede dibujar un grafo que represente de manera esquemática la red. Existe en redes el concepto de “métrica”, que es la categorización de un enlace, de cara a que sea más o menos elegible para albergar un flujo de datos. Igualmente, las carreteras tienen sus propias métricas (como su distancia o el tiempo que se tarda en recorrerlas), o categorizaciones (por ejemplo, el código de colores empleado en sus nombres de ruta).

En la Figura 2.4 se ve un ejemplo de esquema de red de capa 3. Se trata de una malla de ocho enrutadores interconectados entre ellos mediante enlaces punto a punto. En redes informáticas existe el concepto de redes punto a punto y redes multipunto, que de hecho, son las más extendidas (Kurose y Ross, 2010). En este caso, el grafo queda dibujado siendo los enrutadores los nodos del grafo, y sus enlaces los arcos del mismo.

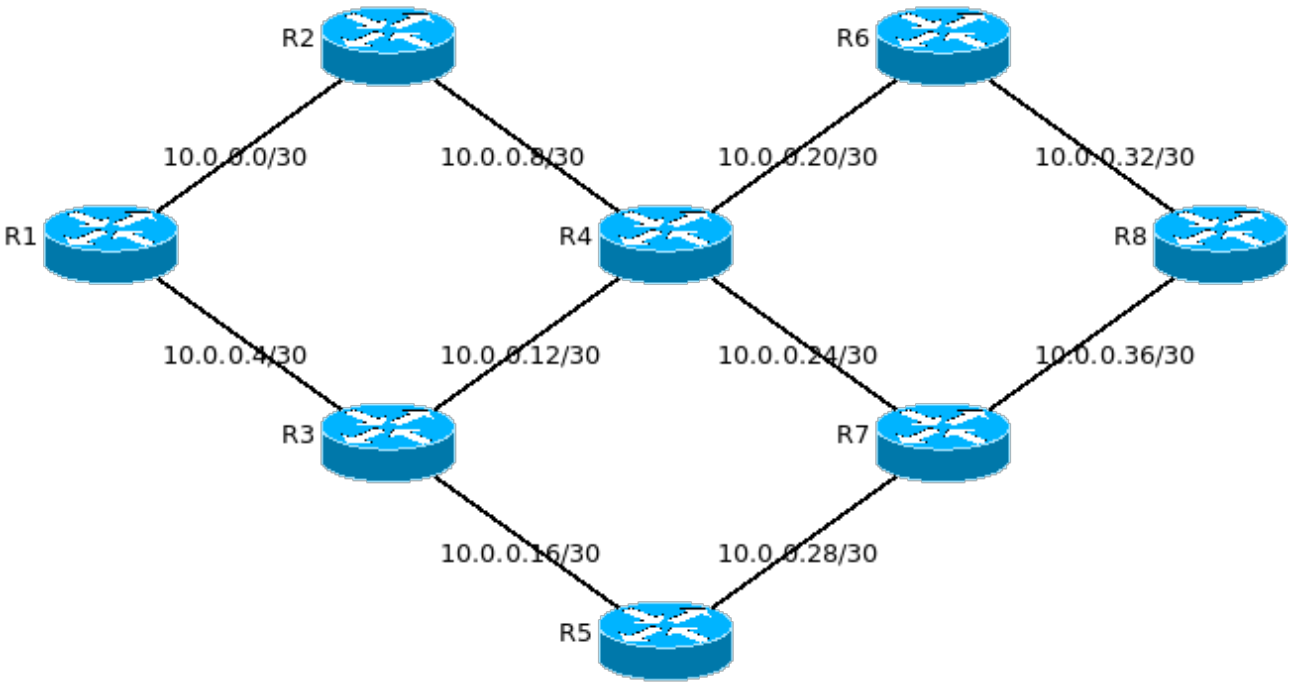


Figura 2.4: ejemplo de esquema de red informática

Si estos enrutadores ejecutasen OSPF (o cualquier otro protocolo), todos formarían su propia versión del grafo eliminando cualquier posible bucle, a base de sumar las métricas para llegar al resto de los nodos y elegir el camino con mejor métrica, como se ve en la Figura 2.5.

Hay protocolos (como EIGRP) que ofrecen balanceo de carga a igual coste de rutas por defecto, e incluso se puede configurar cierta tolerancia para realizar dicho balanceo si la diferencia de métricas no supera un cierto umbral. Sin embargo, éste no es el caso de OSPF.

En realidad los enrutadores no tienen consciencia de la topología más allá de los que tiene directamente conectados. Sin embargo, R1 sabe que, salvo para llegar a R2, tiene que usar el enlace con R3 para llegar al resto de nodos, puesto que éste le ha anunciado que tiene una mejor ruta para todos ellos. Como resultado, delega en R3 para llegar a esos nodos, y R3 a su vez consultará su tabla de rutas para saber dónde tiene que reenviar el paquete que le ha enviado R1.

Según el ejemplo, un paquete enviado desde R1 a R8 usaría el camino R3 – R4 – R7 y R8.

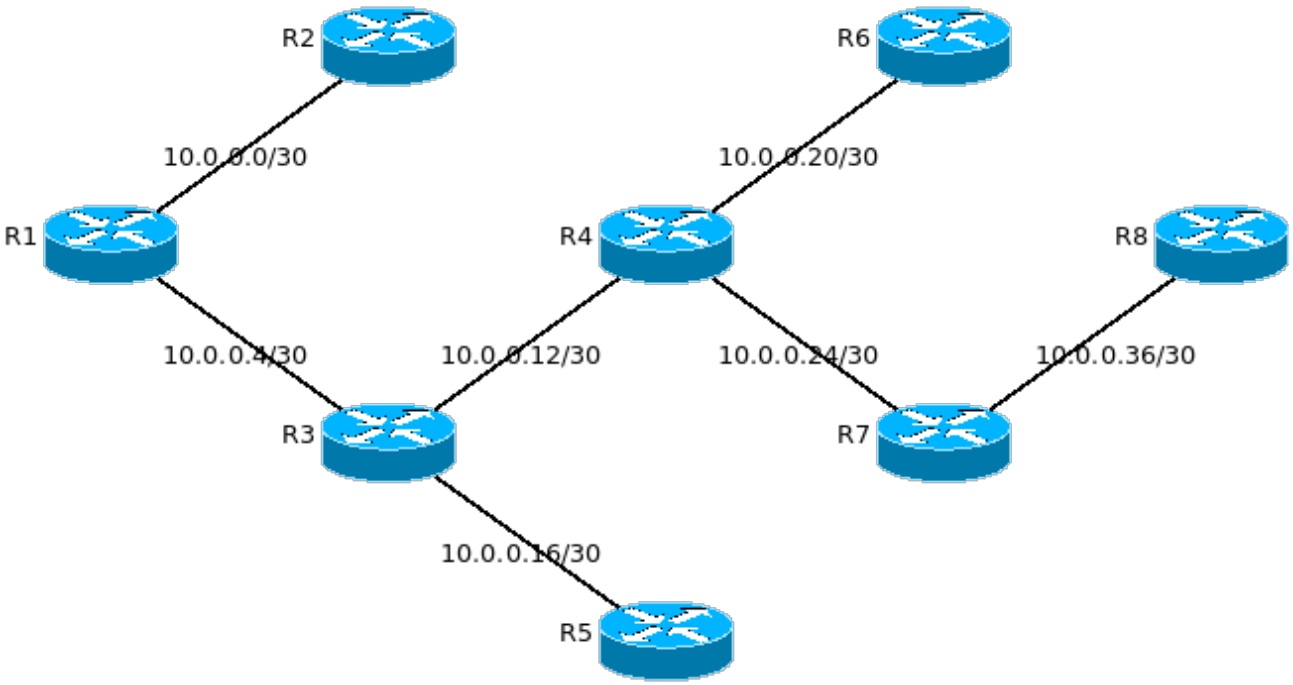


Figura 2.5: topología de red desde la perspectiva de R1

Si trasladamos el mismo dibujo a un mapa (esquemático) de carreteras el resultado es similar (Figura 2.7): un grafo en el que los nodos son ciudades, y los arcos son las carreteras que los unen. De manera similar al ejemplo de los enrutadores, el conductor de un vehículo que quiera ir de Springfield a North Havenbrook, por ejemplo, consultaría el equivalente a la “tabla de rutas” en una red viaria, que son las señales e indicadores de carretera. Aunque hubiese distintas maneras de alcanzar un destino encontraría señalización indicando los destinos directos (Shelbyville vía S1 y Brockway vía S2), y además, señalización hacia poblaciones importantes (Capital City vía S2), o

Figura 2.6: señal de "todas direcciones" sobre la carretera M-501 (GSV)

incluso una señalización hacia “todas direcciones” como la de la Figura 2.6, que podría ser el equivalente a una ruta por defecto (0.0.0.0/0) en la tabla de rutas de un enrutador.

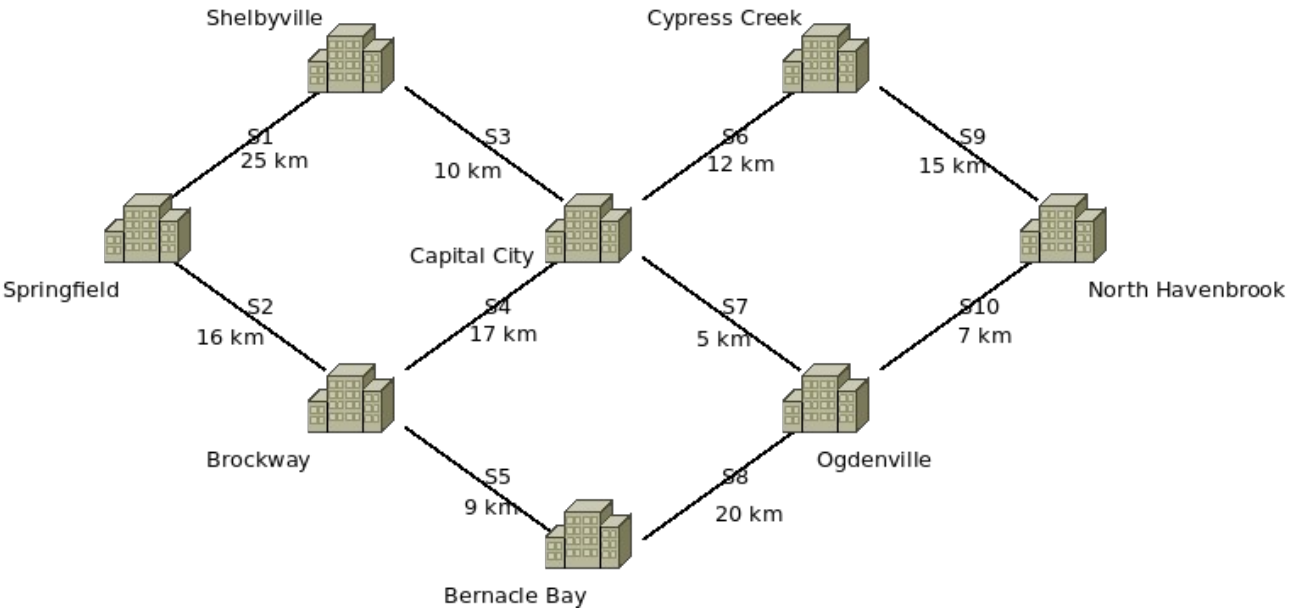


Figura 2.7: grafo formado por ciudades y carreteras

El algoritmo en acción

Cogiendo como ejemplo la red de carreteras de la Figura 2.7, el algoritmo empieza a ejecutarse en cada uno de sus nodos. En una primera iteración añadirán a su “tabla de rutas” las carreteras y destinos directamente conectados, indicando la métrica (en este caso la distancia), y el identificador de carretera, como se ve en la Tabla 2.1:

Springfield Shelbyville 25 km vía S1 Brockway 16 km vía S2	**Shelbyville** Springfield 25 km vía S1 C. City 10 km vía S3	**Brockway** Springfield 16 km vía S2 C. City 17 km vía S4 B. Bay 9 km vía S5	**Capital City** Shelbyville 10 km vía S3 Brockway 17 km vía S4 C. Creek 12 km vía S6 Ogdenville 5 km vía S7
Bernacle Bay Brockway 9 km vía S5 Ogdenville 20 km vía S8	**Cypress Creek** C. City 12 km vía S6 North H. 15 km vía S9	**Ogdenville** C. City 5 km vía S7 B. Bay 20 km vía S8 North H, 7 km vía S10	**North Havenbrook** C. Creek 15 km vía S9 Ogdenville 7 km vía S10

Tabla 2.1: tablas de enrutamiento iniciales de las poblaciones de la Figura 2.6

Una vez que todos los nodos tienen conocimiento de quiénes son sus vecinos, éstos empiezan a intercambiar información entre sí. Así pues, cada uno añade a su tabla de rutas las rutas del vecino, sumando a la métrica anunciada el coste de llegar a dicho vecino. Queda ilustrado en la Tabla 2.2:

Springfield Shelbyville 25 km vía S1 Brockway 16 km vía S2 C. City 33 km vía S2 B. Bay 25 km vía S2	**Shelbyville** Springfield 25 km vía S1 C. City 10 km vía S3 Brockway 27 km vía S3 C. Creek 22 km vía S3 Ogdenville 15 km vía S3	**Brockway** Springfield 16 km vía S2 C. City 17 km vía S4 B. Bay 9 km vía S5 Shelbyville 27 km vía S4 C. Creek 29 km vía S4 Ogdenville 22 km vía S4	**Capital City** Shelbyville 10 km vía S3 Brockway 17 km vía S4 C. Creek 12 km vía S6 Ogdenville 5 km vía S7 Springfield 33 km vía S4 B. Bay 25 km vía S7 North H, 12 km vía S7
Bernacle Bay Brockway 9 km vía S5 Ogdenville 20 km vía S8 Springfield 25 km vía S5 C. City 25 km vía S8 North H, 27 km vía S8	**Cypress Creek** C. City 12 km vía S6 North H. 15 km vía S9 Shelbyville 22 km vía S6 Brockway 29 km vía S6 Ogdenville 17 km vía S6	**Ogdenville** C. City 5 km vía S7 B. Bay 20 km vía S8 North H, 7 km vía S10 Shelbyville 15 km vía S7 Brockway 22 km vía S7 C. Creek 17 km vía S7	**North Havenbrook** C. Creek 15 km vía S9 Ogdenville 7 km vía S10 C. City 12 km vía S10 B. Bay 27 km vía S10

Tabla 2.2: tablas de enrutamiento tras primer intercambio de rutas

En esta iteración ya empiezan a pasar cosas interesantes: Por un lado, un nodo ya sabe cómo llegar a poblaciones más allá de aquéllas con las que tiene una carretera en común. Por otro lado, algunos nodos empiezan a recibir información sobre cómo llegar a un destino por distintas vías. En este caso, el nodo debe elegir la mejor vía. Es el caso de Springfield, a la que por un lado Shelbyville le anuncia que tiene Capital City a 10 kilómetros. Más los 25 que separan Springfield de Shelbyville, en total son 35 kilómetros hasta Capital City si se va vía Shelbyville. Por otro lado, Brockway le está anunciando que tiene Capital City a 17 kilómetros. Sumados a los 16 que separan Springfield

de Brockway, hacen un total de 33 kilómetros. Puesto que 33 es mejor que 35, Springfield instalará en su tabla de rutas Capital City a 33 kilómetros vía Brockway.

Una nueva iteración del algoritmo hace que los nodos vecinos intercambien sus rutas nuevamente. En este punto puede darse la posibilidad de que a un nodo se le anuncie una ruta que ya tenía instalada en su tabla de rutas, pero con mejor métrica. En este caso el nodo sustituiría la entrada en la tabla de rutas por una nueva entrada al mismo destino con la métrica actualizada. Las tablas de enrutamiento quedarían como se ve en la Tabla 2.3:

Springfield	**Shelbyville**	**Brockway**	**Capital City**
Shelbyville 25 km vía S1	Springfield 25 km vía S1	Springfield 16 km vía S2	Shelbyville 10 km vía S3
Brockway 16 km vía S2	C. City 10 km vía S3	C. City 17 km vía S4	Brockway 17 km vía S4
C. City 33 km vía S2	Brockway 27 km vía S3	B. Bay 9 km vía S5	C. Creek 12 km vía S6
B. Bay 25 km vía S2	C. Creek 22 km vía S3	Shelbyville 27 km vía S4	Ogdenville 5 km vía S7
C. Creek 45 km vía S2	Ogdenville 15 km vía S3	C. Creek 29 km vía S4	Springfield 33 km vía S4
Ogdenville 38 km vía S2	B. Bay 35 km vía S3	Ogdenville 22 km vía S4	B. Bay 25 km vía S7
	North H, 32 km vía S3	North H, 29 km vía S4	North H, 12 km vía S7
Bernacle Bay	**Cypress Creek**	**Ogdenville**	**North Havenbrook**
Brockway 9 km vía S5	C. City 12 km vía S6	C. City 5 km vía S7	C. Creek 15 km vía S9
Ogdenville 20 km vía S8	North H. 15 km vía S9	B. Bay 20 km vía S8	Ogdenville 7 km vía S10
Springfield 25 km vía S5	Shelbyville 22 km vía S6	North H, 7 km vía S10	C. City 12 km vía S10
C. City 25 km vía S8	Brockway 29 km vía S6	Shelbyville 15 km vía S7	B. Bay 27 km vía S10
North H, 27 km vía S8	Ogdenville 17 km vía S6	Brockway 22 km vía S7	Shelbyville 32 km vía S10
Shelbyville 35 km vía S8	Springfield 45 km vía S6	C. Creek 17 km vía S7	Brockway 29 km vía 10
C. Creek 37 km vía S8	B. Bay 37 km vía S6	Springfield 38 km vía S7	

Tabla 2.3: tablas de enrutamiento tras segundo intercambio de rutas

Finalmente, una última iteración permite que todos los nodos tengan sus tablas de rutas completas (Tabla 2.4). El algoritmo finaliza.

Springfield	**Shelbyville**	**Brockway**	**Capital City**
Shelbyville 25 km vía S1	Springfield 25 km vía S1	Springfield 16 km vía S2	Shelbyville 10 km vía S3
Brockway 16 km vía S2	C. City 10 km vía S3	C. City 17 km vía S4	Brockway 17 km vía S4
C. City 33 km vía S2	Brockway 27 km vía S3	B. Bay 9 km vía S5	C. Creek 12 km vía S6
B. Bay 25 km vía S2	C. Creek 22 km vía S3	Shelbyville 27 km vía S4	Ogdenville 5 km vía S7
C. Creek 45 km vía S2	Ogdenville 15 km vía S3	C. Creek 29 km vía S4	Springfield 33 km vía S4
Ogdenville 38 km vía S2	B. Bay 35 km vía S3	Ogdenville 22 km vía S4	B. Bay 25 km vía S7
North H, 45 km vía S2	North H, 32 km vía S3	North H, 29 km vía S4	North H, 12 km vía S7
Bernacle Bay	**Cypress Creek**	**Ogdenville**	**North Havenbrook**
Brockway 9 km vía S5	C. City 12 km vía S6	C. City 5 km vía S7	C. Creek 15 km vía S9
Ogdenville 20 km vía S8	North H. 15 km vía S9	B. Bay 20 km vía S8	Ogdenville 7 km vía S10
Springfield 25 km vía S5	Shelbyville 22 km vía S6	North H, 7 km vía S10	C. City 12 km vía S10
C. City 25 km vía S8	Brockway 29 km vía S6	Shelbyville 15 km vía S7	B. Bay 27 km vía S10
North H, 27 km vía S8	Ogdenville 17 km vía S6	Brockway 22 km vía S7	Shelbyville 32 km vía S10
Shelbyville 35 km vía S8	Springfield 45 km vía S6	C. Creek 17 km vía S7	Brockway 29 km vía 10
C. Creek 37 km vía S8	B. Bay 37 km vía S6	Springfield 38 km vía S7	Springfield 45 km vía S10

Tabla 2.4: tablas de enrutamiento tras segundo intercambio de rutas

Una vez completas las tablas de rutas se puede proceder al cálculo de rutas. En este caso, para ir de Springfield a North Havenbrook, se siguen los siguientes pasos:

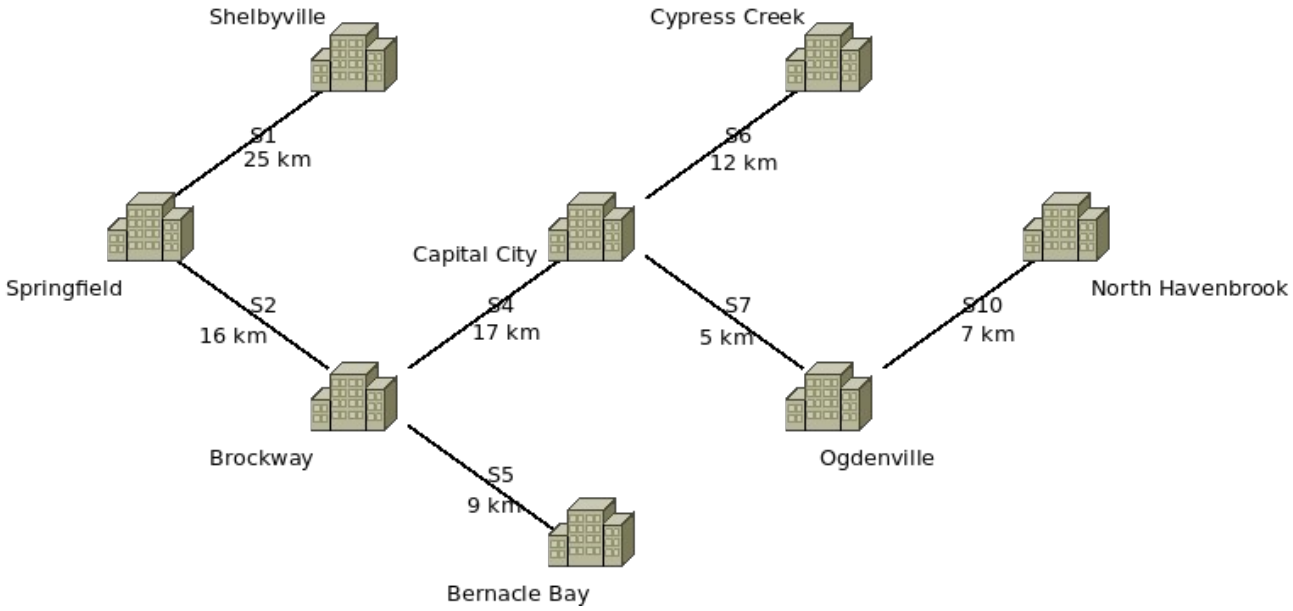


Figura 2.8: mapa de recubrimiento mínimo desde la perspectiva de uno de los nodos de la red

- Se consulta la tabla de rutas de Springfield. Hay que ir por la S2 hasta Brockway.
- Se consulta la tabla de rutas de Brockway. Hay que ir por la S4 hasta Capital City.
- Se consulta la tabla de rutas de Capital City. Hay que ir por la S7 hasta Ogdenville.

- Se consulta la tabla de rutas de Ogdenville. North Havenbrook aparece a 7 kilómetros por la S10.

Como ya se ha dicho, las tablas de rutas generan un árbol de recubrimiento mínimo diferente para cada nodo. El de Springfield quedaría como muestra la Figura 2.8. Nótese que, comparados con la figura 2.7, vienen a ser igual que la topología que se propone en las Figuras 2.4 y 2.5:

2.4 Estándar de cartografiado

De cara a proyectar la realidad (una red de carreteras uniendo poblaciones) a un elemento manejable por una aplicación informática, como es el grafo, es necesario, por un lado, establecer un soporte que dote de agilidad al intercambio de datos y, sobre todo, que permita que se realicen alteraciones relacionadas con las redes viarias como pueden ser actualizaciones, incidencias, ampliaciones, o interoperabilidad con otras regiones; y por otro, establecer un criterio que permita representar de una manera práctica y esquemática dicha realidad. Dicho de manera resumida, manipular los datos para convertirlos en un formato apropiado o útil (Ruiz Larrocha y Ruiz Virumbrales, 2012).

2.4.1 Soporte

Ya se ha hablado del paradigma orientado a objetos, y de cómo se va a representar el grafo mediante una estructura de datos que contendrá nodos y arcos. Resulta evidente que no tiene sentido crear estos elementos de manera estática (es decir, fijar en el código el número de poblaciones con sus nombres y enlaces de manera inalterable), con lo que hay que utilizar un soporte externo con los datos necesarios para crearlos de manera dinámica.

Dicho soporte puede residir en un simple archivo de texto anexado a la aplicación. Pero esto también resulta una limitación si se usan datos extendidos, o repetidos. Por ejemplo: supongamos dos listas de carreteras asociadas a una población, como las Tablas 2.5 y 2.6:

identificador	destino	distancia	categoría
S-3	Shelbyville	10 km.	local
S-4	Brockway	36 km.	comarcal
S-6	Cypress Creek	12 km.	local
S-7	Ogdenville	5 km.	camino

Tabla 2.5: ejemplo de lista de carreteras que atraviesan una población

identificador	destino	distancia	categoría
S-2	Springfield	120 km.	autopista
S-4	Capital City	36 km.	comarcal
S-5	Bernacle Bay	87 km.	nacional

Tabla 2.6: ejemplo de lista de carreteras que atraviesan otra población

En el caso de la carretera S-4 encontramos datos duplicados, ya que hay varios campos que aparecen en las dos tablas. Esta redundancia, más allá de desperdiciar espacio, que puede ser hasta despreciable, hace poco escalable la aplicación. Si hubiera que ampliar la cartografía hay que realizar un trabajo duplicado de introducir los datos de una carretera en concreto en todas las tablas de los nodos por donde ésta pasa, con el consiguiente riesgo de cometer errores, y por consiguiente, generar inconsistencias.

Si elaboramos una lista general como la Tabla 2.7, donde se incluyan los datos duplicados, y eliminamos los campos “distancia” y “categoría” de las Tablas 2.5 y 2.6, haciendo que el nodo en cuestión tenga que consultar la Tabla 2.7 basándose en el campo “identificador”, eliminamos las duplicidades, y por consiguiente, conseguimos que sea más escalable y consistente.

identificador	longitud	categoría
S-1	64 km.	nacional
S-2	120 km.	autopista
S-3	10 km.	local
S-4	36 km.	comarcal
S-5	87 km.	nacional
S-6	12 km.	local
S-7	5 km.	camino
S-8	91 km.	autopista
S-9	97 km.	nacional
S-10	46 km.	comarcal

Tabla 2.7: ejemplo de lista de carreteras

Esto sería, a grandes rasgos, la justificación para usar una base de datos relacional en vez de un simple archivo de texto. A partir de archivos de texto donde se hubiese plasmado la cartografía se puede poblar una base de datos, y ésta puede estar, o bien en un servidor aparte donde corra la aplicación, en el mismo sistema, o incluso en memoria (Silberschatz et al., 2002).

2.4.2 Esquema

Una vez decidido que se va a usar una base de datos relacional, hay que decidir el esquema de la misma. Es decir, hay que definir qué criterios se van a usar para el cartografiado.

El ejemplo anterior resulta bastante poco práctico por razones obvias. Crear una tabla por cada nodo atenta contra la escalabilidad, ya que un cambio de topología afectaría al esquema, cuando de lo que se trata es de poder reutilizar el mismo esquema para cualquier región cartografiada. Sin embargo, sí que se podría unificar toda esa información en una tabla única conteniendo los enlaces de todas las poblaciones, como en la Tabla 2.8. Aún así, siguen existiendo redundancias. La misma carretera S-4 que va de Capital City a Brockway aparece dos veces en la tabla (esto es debido a que se supone que todas las carreteras son bidireccionales). Se podría eliminar cualquiera de las dos filas perfectamente. De esta manera podríamos tener en esta tabla una lista con todos los arcos del grafo.

origen	destino	identificador
Capital City	Shelbyville	S-3
Capital City	Brockway	S-4
Capital City	Cypress Creek	S-6
Capital City	Ogdenville	S-7
Brockway	Springfield	S-2
Brockway	Capital City	S-4
Brockway	Bernacle Bay	S-5

Tabla 2.8: tabla unificada de destinos

Aunque ahora podría parecer que nuevamente tenemos una clave primaria de la tabla en el campo “identificador”, y por tanto la información de la Tabla 2.7 podría ir en esta tabla, el esquema que propone la Figura 2.7 es muy poco realista. Normalmente una carretera no empieza en una población y termina en la siguiente, sino que atraviesa varias poblaciones a lo largo de su recorrido, con lo que es posible que el campo “identificador” sí se acabe repitiendo en la tabla de arcos. En esta tabla representaremos tramos de carretera. La clave primaria estará representada por los

campos “origen” y “destino”, y en este caso sí que es interesante que aparezcan de nuevo los campos relativos a las métricas específicas del tramo (que no de la carretera entera) en la tabla de arcos. Finalmente quedaría como la Tabla 2.9.

origen	destino	identificador	longitud tramo
Capital City	Shelbyville	S-3	10 km.
Capital City	Brockway	S-4	17 km.
Capital City	Cypress Creek	S-6	12 km.
Capital City	Ogdenville	S-7	5 km.
Brockway	Springfield	S-2	16 km.
Brockway	Bernacle Bay	S-5	9 km.

Tabla 2.9: esquema final de la tabla de arcos

De esta manera, para obtener información completa de un arco en concreto se hace una consulta a las dos tablas, similar a:

```
SELECT * FROM arcos,carreteras WHERE
arcos.identificador==carreteras.identificador;
```

El hecho de que una población sea origen o destino no es relevante. Se puede obtener una lista de todos los nodos con una consulta como la siguiente:

```
SELECT DISTINCT origen,destino FROM arcos;
```

2.5 Conclusiones

En este capítulo se ha hablado del uso que se va a dar a los grafos en el proyecto. Cómo se van a implementar en lenguaje de programación, cómo se va a optimizar su análisis con un algoritmo de recubrimiento, y cómo se van a representar en una base de datos. En capítulos posteriores se profundizará más en la implementación y en aspectos más concretos de su diseño.

Capítulo 3: Análisis

3.1 Introducción

En este capítulo se describe el análisis de las tecnologías especificadas en el capítulo anterior con el fin de elaborar una línea base a la hora de empezar a desarrollar el producto propuesto. A partir de este análisis se pretenden establecer los requisitos y características que debe cumplir el producto. Este capítulo viene a ser el equivalente al documento de diseño de alto nivel en ingeniería del software, donde, tras hablar con el cliente, se establecen qué requisitos debe cumplir el sistema, y de qué manera se van a cumplir dichos requisitos de una manera que el personal no técnico involucrado en el proyecto sea capaz de comprenderlo.

El proceso de diseño se inicia con un análisis de mercado y con la definición de los requisitos del producto. Tras varias refinaciones a distintos niveles por parte del equipo de diseño. Si durante esta fase se detecta que no se ha tenido en cuenta alguna restricción o que alguna medida no ha sido satisfactoria se deben volver a especificar los requisitos, las especificaciones, o incluso la arquitectura (Gajski, 1997).

La importancia de esta fase es crítica, ya que éste es el punto donde el cliente y el proveedor se tienen que poner de acuerdo, y fijar las líneas base. Un error en cualquier definición en esta fase puede provocar que al final se entregue un producto que no cumpla con las expectativas del cliente, lo cual puede acarrear consecuencias negativas para ambas partes (por un lado, el cliente no recibe el producto por el que ha pagado, y por otro, la pérdida de prestigio del proveedor y confianza en el mismo, más allá de incluso posibles consecuencias legales). De hecho, no siempre se consigue plasmar todo lo esperado del producto en esta fase. Los problemas que genera esto, aunque se pueden solventar de manera más o menos amistosa por ambas partes una vez entregado, suponen una pérdida de tiempo y recursos empleados en solventar las carencias, por no hablar de que muchas veces las soluciones suponen poner “parches” que son más sencillos de implementar que la solución óptima, que pasaría por la refactorización de todo el conjunto del software.

3.2 Toma de requisitos

La toma de requisitos plasma la necesidad que tiene el cliente y que se trata de solucionar por medio del producto a desarrollar. Esto especificará el comportamiento del sistema.

Para empezar, es necesario especificar quién será el interlocutor por parte del cliente. Por parte del cliente debe ser el conjunto de personas expertas en el área donde va a implementarse el sistema. Si

no hubiese nadie apropiado para exponer las necesidades y requisitos, es el analista el que debe documentarse para poder elaborar la lista de requisitos del sistema (Cerrada, 2000).

3.2.1 Requisitos funcionales

Son los que describen qué debe hacer el sistema. Deben estar relacionados con el concepto propuesto por el cliente (Cerrada, 2000).

Basándose en la observación de sistemas similares de trazado y planificación de rutas, se establece que el sistema propuesto será un sistema monolítico (aunque en un futuro puede ser distribuido) (Liu, 2004), y debe cumplir los siguientes requisitos:

Respecto al trazado de rutas:

> RF1: Dada una cartografía de una región concreta, se debe poder elegir cualquier pareja de poblaciones de dicha región para trazar el mejor camino entre ellas. Se debe poder elegir entre la totalidad de las poblaciones de la región.
>
> RF2: Se deben poder ofrecer alternativas de criterio a la hora de elegir el mejor camino. Como mínimo, poder elegir entre un itinerario o más rápido o más corto. Adicionalmente se debería poder elegir además una ruta optimizada basándose en la categoría de las vías, dando más prioridad al hecho de recorrer el mayor número de kilómetros posible sobre las vías de mayor rango.
>
> RF3: Con estos datos el sistema tiene que devolver al usuario un itinerario con instrucciones sencillas y claras, paso a paso, para poder alcanzar el destino. Estas instrucciones deben incluir nombres de vías y de poblaciones.
>
> RF4: El sistema tiene que tener en cuenta a la hora de realizar la traza los posibles cambios en la topografía (por ejemplo, en caso de incidencia).

Respecto a la gestión de incidencias:

> RF5: Se debe poder elegir cualquier tramo de carretera de la región para declarar una incidencia en el mismo. Por incidencia se entiende corte total del tramo. Por tramo se entiende una sección ininterrumpida de carretera que empieza y termina en una población y/o en una intersección.

RF6: Una vez declarada la incidencia, se debe indicar el texto a mostrar en los teleindicadores próximos al tramo afectado. El texto debe mostrar claramente el identificador de carretera, el principio y el fin del tramo afectado.

RF7: Además de la incidencia, los teleindicadores deben, en la medida de lo posible, ofrecer alternativas viables para evitar el tramo afectado.

RF8: Cualquier incidencia debe poder ser eliminada.

RF9: Al eliminar la incidencia debe indicarse el texto a mostrar en los teleindicadores que reflejaron la incidencia. Nuevamente, debe indicar el identificador de carretera, el principio y el final del tramo afectado.

RF10: Cualquier incidencia, una vez eliminada, debe desaparecer de la lista de incidencias activas.

RF11: No se debe poder añadir una incidencia sobre un tramo ya afectado por una incidencia existente.

3.2.2 Requisitos prácticos

Son los que describen cómo debe funcionar el sistema, y cómo se debe usar. Suelen especificar características del interfaz de usuario, su inicialización y configuración (Cerrada, 2000).

El sistema debe cumplir los siguientes requisitos prácticos:

RP1: El sistema debe poder ejecutarse en cualquier plataforma, con independencia del sistema operativo, que cumpla los requisitos mínimos de hardware especificados por el proveedor. Al ser un producto Java, deberá poder ejecutarse mínimo con la versión 11 de Java.

RP2: El interfaz de usuario debe ser sencillo, fácil de usar e intuitivo. El usuario debe saber qué información se le requiere introducir. Debe estar íntegramente en idioma español.

RP3: Los resultados obtenidos deben ser fácilmente identificables, interpretables y sin ambigüedades.

RP3: El sistema debe poder actualizar, ampliar o intercambiar su topología de manera independiente por parte del cliente sin necesidad de intervención de un desarrollador de la aplicación.

RP4: El cartografiado debe fijar un estándar en el que se definan los registros a introducir en la base de datos de la topografía. Debe hacer sencillo realizar una ampliación manual. Estos registros deben incluir:

- Origen del tramo.
- Final del tramo.
- Identificador de carretera.
- Métricas asociadas al tramo (tiempo y distancia).
- Aspectos relevantes relativos a la señalización.

RP5: El sistema debe economizar recursos en la medida de lo posible.

RP6: El sistema debe funcionar sin errores, y debe ser lo más resiliente posible en caso de desastre.

3.2.3 Exclusiones

Quedan fuera de la entrega de este producto, pero no del proyecto RutaSierra, los siguientes elementos:

- La gestión e interconexión con los teleindicadores.
- El desarrollo de la aplicación móvil, así como el interfaz de la aplicación con el sistema.
- Los sistemas de activación de señalización provisional.
- Los sistemas de avisos a personal de mantenimiento, Protección Civil, ambulancias, bomberos, Guardia Civil o cuerpos de Policía Local que puedan estar involucrados en la gestión *in situ* de las incidencias sobre las carreteras.

3.3 Definiciones

Operador: Persona del centro de control que va a interactuar con el sistema.

Usuario: Conductor (o copiloto) que se encuentra en un vehículo transitando la red de carreteras. El usuario puede o no hacer uso de la aplicación móvil.

Cliente: Entidad que adquiere el software, y que pone a disposición del sistema los soportes necesarios y adecuados para que éste corra sin errores. Además debe disponer de personal

cualificado para realizar tareas sencillas como la instalación del software o la actualización del archivo de cartografía. Este personal puede ser el mismo operario.

Sistema: El componente software, incluyendo archivos ejecutables y de cartografía.

Interfaz: Elemento gráfico del sistema. Es con el que el operador interactuará, tanto para introducir datos, como para recibir una respuesta e interpretarla.

Cartografía o topografía: Se compone de archivos CSV conteniendo la base de datos que usará el sistema. Representa todas las poblaciones y carreteras de la región, y cómo están interconectados entre sí.

Aplicación móvil: Elemento que se ejecuta en el dispositivo móvil o en la consola del vehículo del usuario. Se usará para recibir las peticiones de traza de ruta, geolocalizar al usuario y mostrar las indicaciones de ruta.

Teleindicadores: Paneles luminosos informativos situados en las carreteras y poblaciones, con objeto de dar información en tiempo real a los usuarios de las mismas.

3.4 Funcionamiento

El funcionamiento esperado del sistema es bastante sencillo, lo cual favorecerá su uso por parte de un usuario inexperto. Teniendo en cuenta que existen bastantes aplicaciones similares que realizan una función parecida, no debería ser un problema para cualquier operador poder usarlo sin apenas recibir instrucciones sobre su uso.

3.4.1 Cálculo de itinerarios

Recepción de datos

El usuario debe poder tener la oportunidad de elegir entre una lista de orígenes y destinos. Estos orígenes y destinos se podrán elegir en un mapa o a partir de una lista. Adicionalmente, debe poder elegir un criterio de selección de itinerario (por ejemplo, basándose en ruta más rápida, o más corta). Mediante aplicación móvil, o mediante interacción del operador, se introducirán esos datos en el sistema

Cálculo

El sistema deberá crear una base de datos embebida, si no existe ya, usando los archivos de cartografía adjuntados.

Usando esta base de datos recopilará todos los nodos existentes en la topografía y sus conexiones para trazar el grafo de conectividades.

A partir de este grafo, y teniendo en cuenta el criterio elegido, deberá trazar un árbol de expansión de cada nodo, de modo que a partir de ahí se pueda trazar el itinerario a seguir.

Partiendo desde el origen irá indicando todos los nodos hasta llegar al destino.

Muestra de resultados

El resultado a mostrar al usuario será mediante la aplicación móvil y los teleindicadores. Deberá ser un conjunto de datos mostrando información del itinerario (duración, distancia), e indicaciones paso a paso sencillas y entendibles para llegar al destino (aplicación móvil). En los teleindicadores sólo se mostrará la indicación que sea pertinente en cada caso..

Estas indicaciones deben incluir elementos reconocibles por el usuario, como nombres de poblaciones y de carreteras.

Estas indicaciones se mostrarán al usuario en los teleindicadores según la aplicación detecte mediante geolocalización que se está aproximando a uno de ellos.

3.4.2 Gestión de incidencias

Recepción de datos

Para crear una incidencia, el operador debe poder tener la oportunidad de elegir entre una lista de tramos dentro de la región. Estos tramos se podrán elegir en un mapa o a partir de una lista.

Para eliminar una incidencia, el operador debe tener la oportunidad de elegir entre la lista de incidencias activas.

Tratamiento

El sistema debe mantener una lista de incidencias activas. Al añadir una nueva debe comprobar antes que no existe previamente, y si es el caso, añadirla a la lista. Al eliminarla debe desaparecer de dicha lista. Los cálculos de itinerario realizados mientras existan incidencias deben tener en cuenta esta lista.

Muestra de resultados

El resultado a mostrar al operador debe mostrar exactamente el texto a introducir en los teleindicadores referente a las incidencias. Debe indicar claramente el nombre de la vía, el inicio y el final del tramo afectado, y si la incidencia se ha generado o se ha eliminado.

En el caso de una nueva incidencia debe, además, proponer alternativas para poder evitar el tramo afectado.

3.4.3 Casos de uso

En el caso de los usuarios de la vía, éstos realizan mediante aplicación móvil una consulta de itinerario (véase la Figura 3.1). El sistema devuelve los resultados a la propia aplicación, que servirá de guía previa al inicio del viaje. A su vez, la aplicación enviará datos de geolocalización al sistema, y éste detectará qué teleindicador es el más próximo al usuario para indicar al usuario cuál es la siguiente instrucción a seguir (idealmente debería haber un teleindicador próximo a cada encrucijada y en la entrada a cada población).

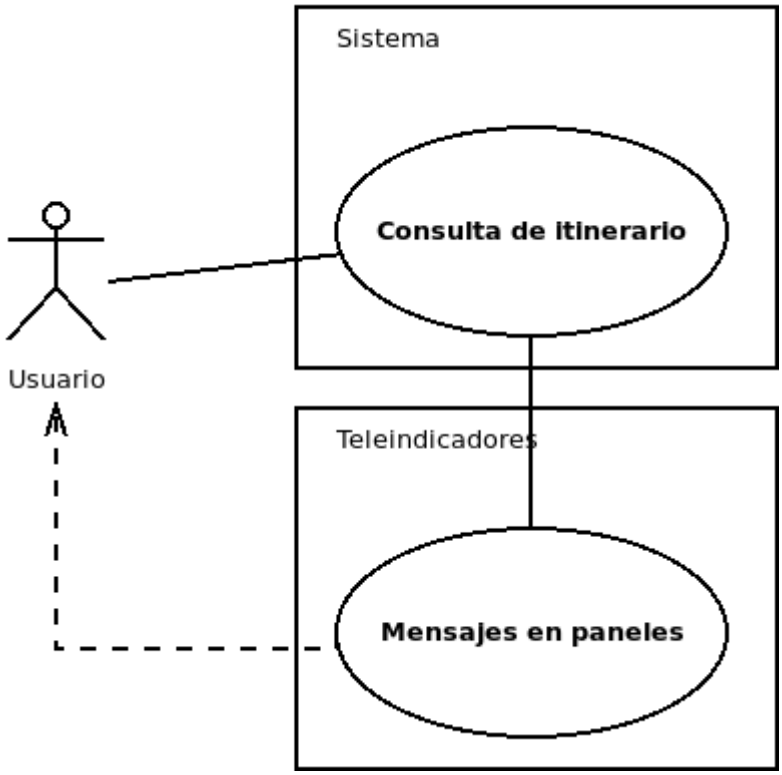


Figura 3.1: diagrama de caso de uso de usuario

En le caso de los operadores encargados de la gestión de incidencias (véase la figura 3.2), éstos, al recibir un aviso por parte de Guardia Civil, Protección Civil o cualquier canal de comunicación de

incidencias, introducen los datos en el sistema para crear o eliminar una incidencia. Ésta queda registrada en el sistema, y a su vez, muestra los mensajes pertinentes en los teleindicadores próximos al tramo afectado por la incidencia.

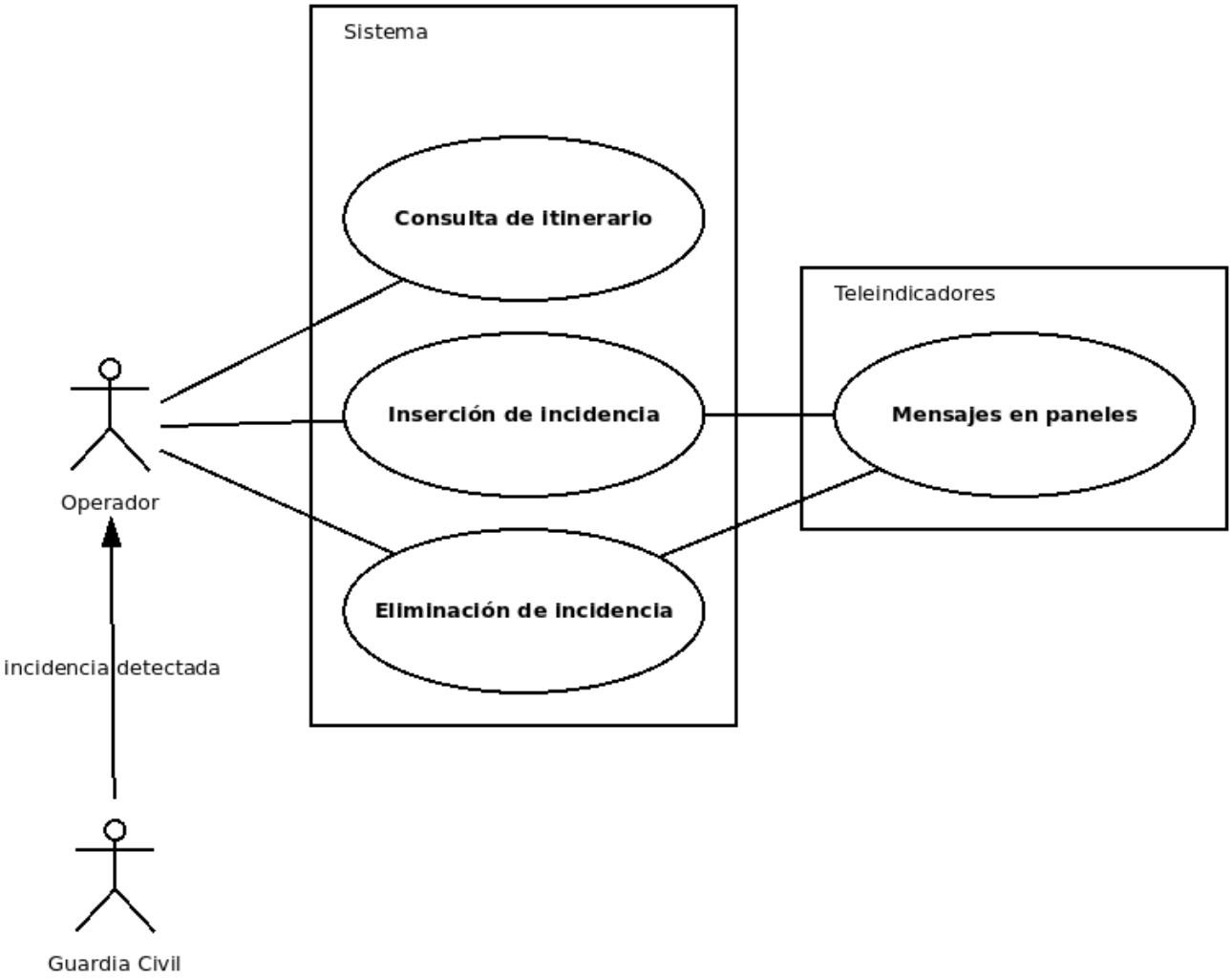


Figura 3.2: diagrama de caso de uso de la gestión de incidencias

3.5 Conclusiones

En este capítulo se han mostrado las características del software a desarrollar, estableciendo requisitos, modo de uso y modo de funcionamiento a alto nivel. Este capítulo sirve como prefacio al capítulo de diseño, en el que se elabora un documento de diseño de bajo nivel. Con este documento las dos partes del contrato conocen lo que se espera del producto a entregar, y es a lo que hay que ceñirse en las fases sucesivas.

El punto crítico de esta fase, la toma de requisitos, ha especificado qué se espera y cómo debe funcionar el producto entregado.

Capítulo 4: Diseño

4.1 Introducción

Este capítulo describe el diseño a bajo nivel del sistema. Tomando como base el capítulo anterior, especialmente en la toma de requisitos, se establecen en este capítulo las metodologías y procedimientos para el desarrollo del sistema.

El documento de diseño a bajo nivel suele ser un documento elaborado en una fase de la ingeniería del software en el que se dan especificaciones técnicas para que el personal encargado del desarrollo del producto sea capaz de llevar a buen término las especificaciones recogidas en el documento de diseño de alto nivel.

En esta fase se medirá la calidad del producto, la cual vendrá determinada en gran medida por las buenas prácticas y la experiencia del ingeniero que realice el diseño, más allá incluso de la destreza del equipo de desarrolladores (Piattini et al., 2015).

4.2 Estructura de la aplicación

El servicio debe definir sus prestaciones a través de un interfaz, y no debe estar ligado a plataformas concretas, de modo que sus distintos componentes puedan interactuar libremente con independencia del lenguaje de programación, el sistema operativo, o la red. Debe estar poco acoplado de cara a poder añadir nuevos requisitos, tanto funcionales como no funcionales (Arias Calleja, 2013).

Se usará una estructura siguiendo el patrón de diseño MVC. Este diseño es muy utilizado en sistemas que incluyen una GUI (Shalloway y Trott, 2001). En la Figura 4.1 se muestra el esquema de este patrón. En la figura 4.2 se muestra cómo quedará el esquema de paquetes de la aplicación.

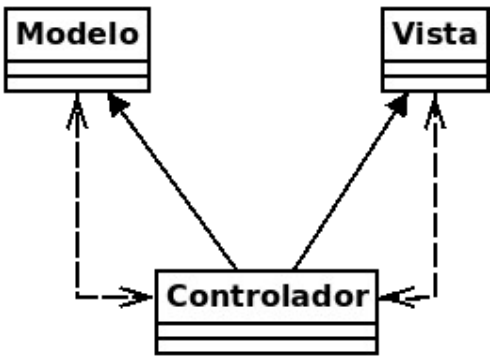


Figura 4.1: esquema del patrón MVC

MVC consta de tres subsistemas:

- El Modelo es la aplicación en sí.
- La Vista es la representación en pantalla.
- El Controlador es la reacción del sistema la entrada de datos de usuarios.

Antes de la aparición de MVC se tendía a agrupar los tres elementos anteriores en uno solo. La gran ventaja que ofrece MVC es la flexibilidad y la reutilización (Gamma et al., 2003).

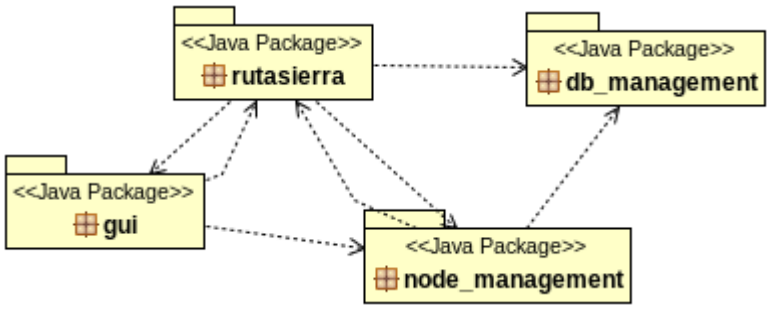


Figura 4.2: diagrama de relaciones entre paquetes

Para el producto que se pretende desarrollar, el código fuente del proyecto debe estar dividido en al menos tres paquetes principales perfectamente diferenciados:

- El paquete correspondiente al Modelo contendrá toda la lógica para crear el grafo a partir de la topología que encontrará en la base de datos. Se crearán dinámicamente los nodos y los arcos que componen el grafo. Cada nodo deberá tener su tabla de rutas propia, y ejecutará el algoritmo para ir completándola (Weiss, 2000).
- El paquete correspondiente a la Vista contendrá todo lo correspondiente a la interfaz visual del sistema. Será el encargado de mostrar el panel con el que interactuará el operario, y de mostrarle los resultados una vez hechos los cálculos y operaciones pertinentes. Se elaborará un formulario usando la biblioteca Swing, un conjunto de componentes JavaBeans que simplifica bastante el proceso de creación de una GUI (Eckel, 2007).
- El paquete correspondiente al Controlador se encargará de inicializar el resto de componentes y de orquestar la interacción entre ellos. Concretamente, a cada interacción del usuario accionará los mecanismos necesarios en el Modelo e interpretará los resultados que obtenga de éste para presentárselos al operario (Parsons, 2008).

4.2.1 Creación del grafo

La base de datos de cartografía contiene una lista con todos los arcos del grafo a crear. A raíz de los campos origen y destino de cada registro se crea una lista con todos los nodos que compondrán el grafo. El diagrama de la Figura 4.3 muestra las clases que tendrá el paquete encargado de gestionar la base de Datos.

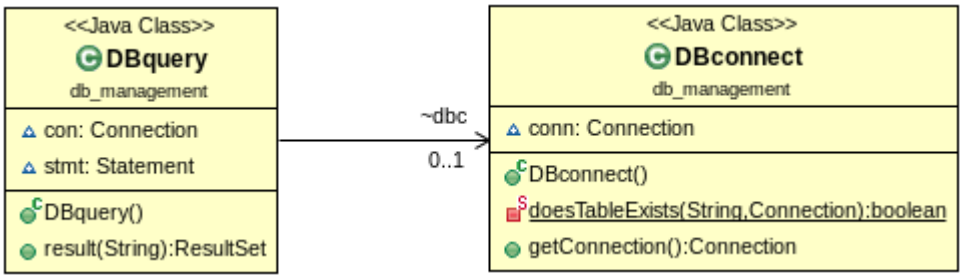


Figura 4.3: diagrama de clases del paquete 'db_management'

Por cada nodo se creará un objeto “población” que buscará todos los registros de la base de datos en los que figura como origen o destino. Por cada registro creará un objeto “tramo”. Este objeto contendrá información del destino, su distancia y el tiempo que tarda en recorrerse.

Las poblaciones, a partir de todos los tramos creados, crean la tabla de rutas con las poblaciones “vecinas”, es decir, con las que hay conexión directa. Más tarde se usarán estas poblaciones para ejecutar el algoritmo y completar la tabla de rutas.

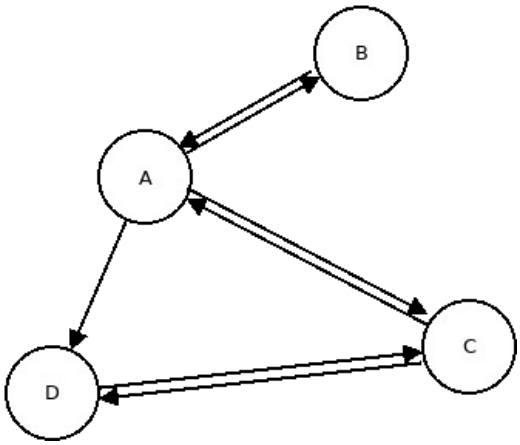


Figura 4.4: ejemplo de grafo dirigido

Nótese que los arcos que compondrán el grafo no serán exactamente todos los registros de la base de datos. Si el tramo en cuestión es bidireccional se crearán dos: uno en el nodo origen y otro en el nodo destino, pero sólo se creará uno en el caso de los tramos de una sola dirección. El grafo que se creará realmente será un grafo dirigido. En la Figura 4.4 la tabla de rutas de A estará compuesta de entradas hacia B,C y D. Sin embargo, D sólo tendrá una entrada hacia C.

Sea cual sea el criterio de enrutamiento, la tabla de rutas de nodos directamente conectados permanecerá inalterable. La Figura 4.5 muestra las relaciones entre clases en el paquete encargado de la gestión de objetos. En este paquete residen las clases abstractas, como son los nodos y los arcos, y los objetos que heredan de estas clases. Además se declara el interfaz de restricciones y el objeto que lo implementa, que es un incidente. Además habrá dos clases específicas para iterar los nodos y los arcos.

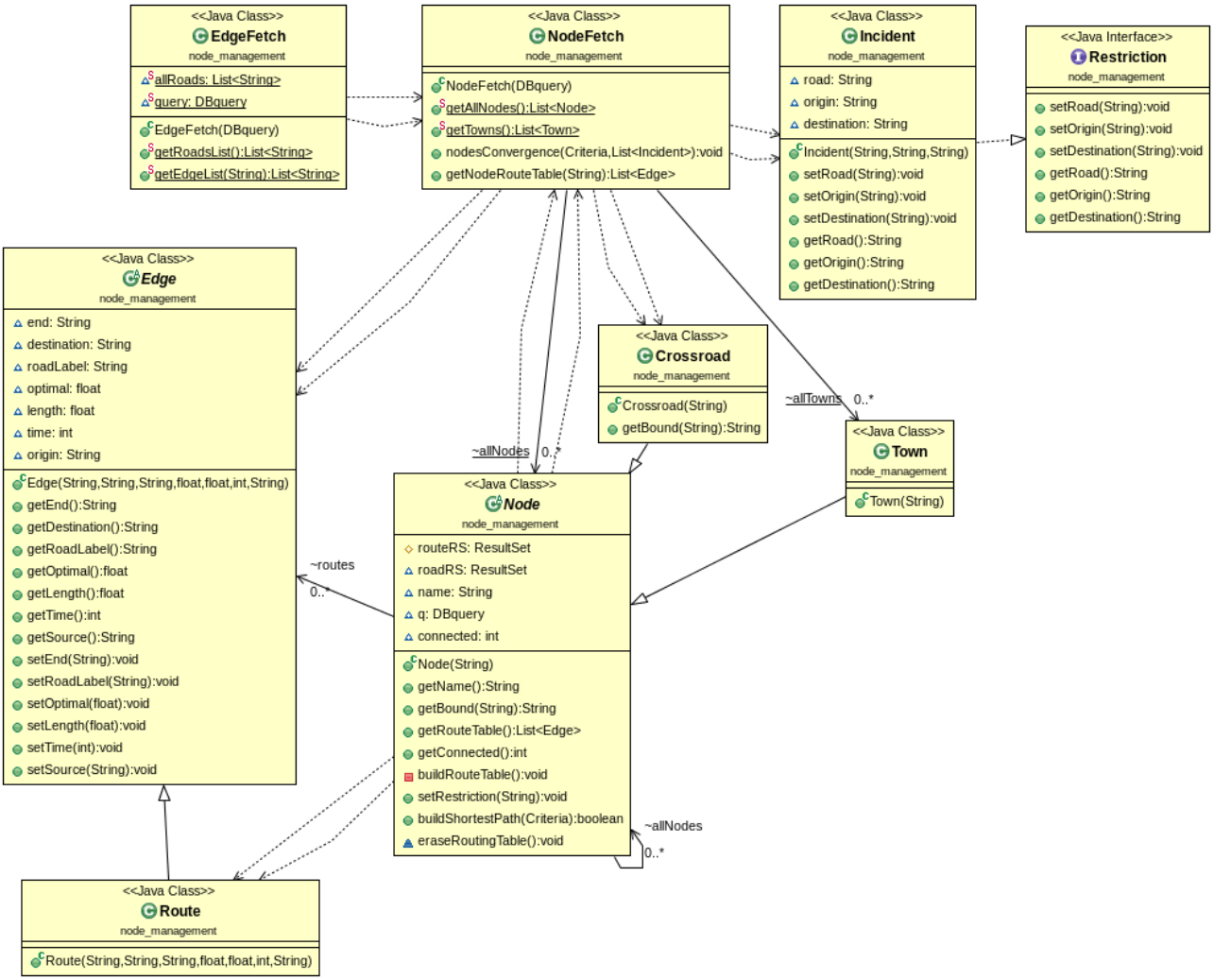


Figura 4.5: diagrama de clases del paquete 'node_management'

4.2.2 Formulario

El formulario será punto de recogida de datos del operador y a su vez será el que muestre los resultados. Sin embargo, no realizará ningún cálculo ni almacenará ningún dato. Los datos recogidos los enviará al Controlador y éste le devolverá el mensaje que deberá mostrar en pantalla. La Figura 4.6 muestra el diagrama de la única clase que contiene el paquete encargado de la Vista.

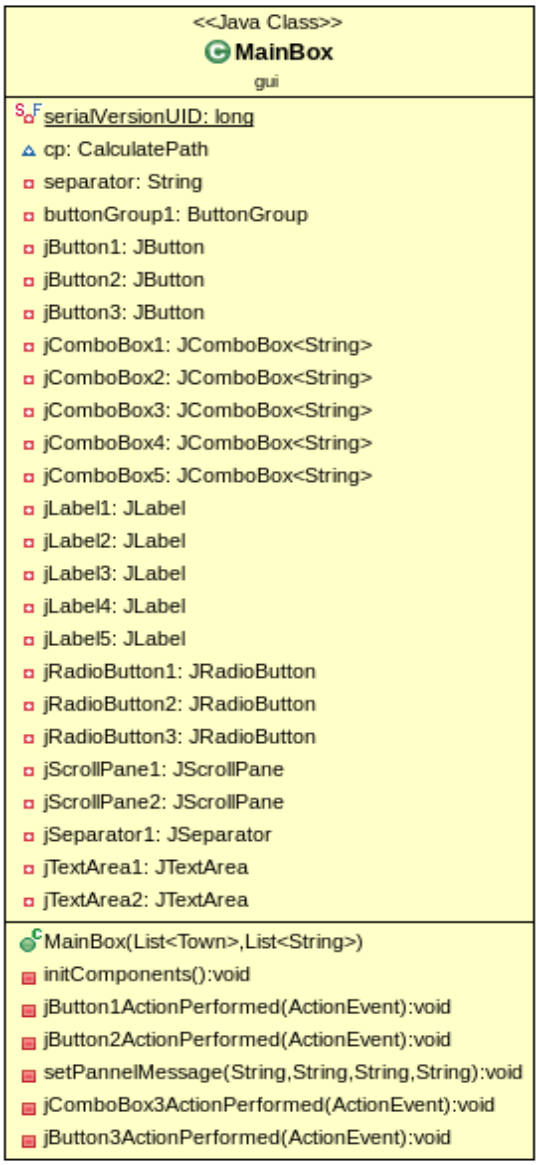


Figura 4.6: diagrama de clases del paquete 'gui'

4.2.3 Recorrido del grafo

Una clase dentro del paquete Controlador (ver Figura 4.7) deberá, en base a la información recibida por el formulario, inspeccionar los nodos del grafo para trazar el camino entre origen y destino. Suponiendo que el nodo D de la Figura 4.4 quisiera llegar al nodo B, el modo de proceder será el siguiente:

- Se analiza la tabla de rutas de D. C (que es su único vecino) le anuncia que sabe cómo llegar a B. Se añade C a la traza de la ruta.
- Se analiza la tabla de rutas de C. A le anuncia que sabe cómo llegar a B. Se añade A a la traza de la ruta.
- Se analiza la tabla de rutas de A. A tiene B directamente conectado. Se añade B a la traza de la ruta y ya se puede mostrar el resultado al operador: D-C-A-B.

Además, habrá que añadir tiempos y distancias totales del recorrido, y la distancia de cada tramo.

También se almacenará en el paquete Controlador la lista de incidencias. Esta lista será actualizada desde el formulario, y será consultada antes de construir las tablas de rutas de cada nodo.

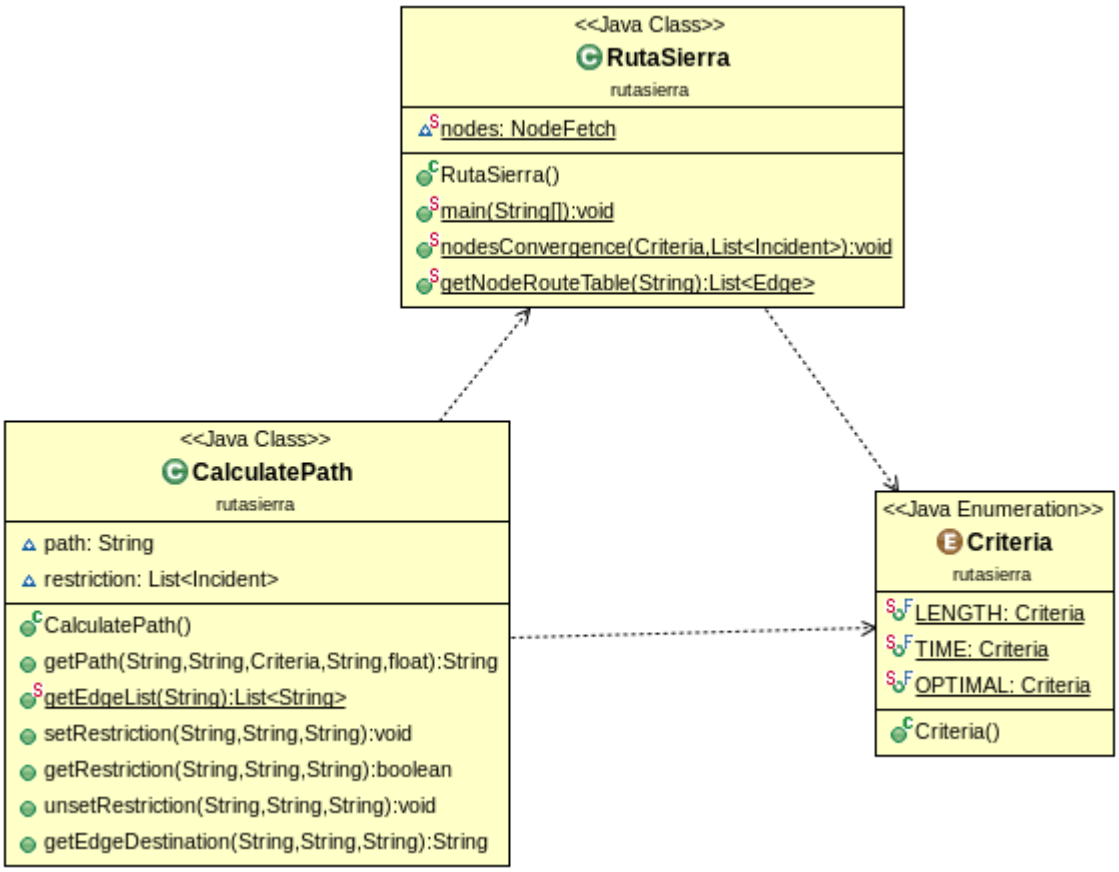


Figura 4.7: diagrama de clases del paquete 'rutasierra'

La Figura 4.8 muestra el diagrama general de clases.

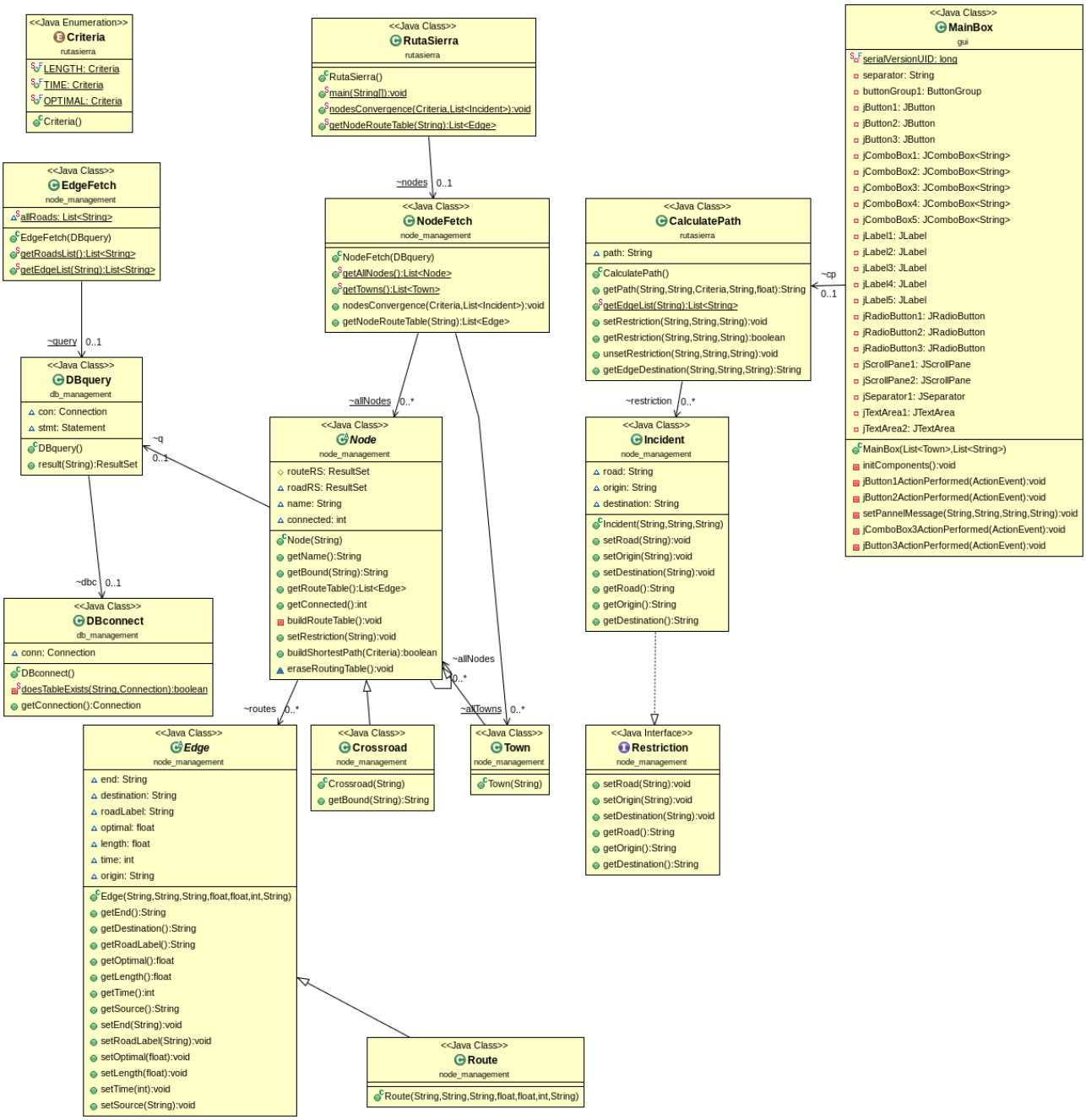


Figura 4.8: diagrama general de clases

4.3 Región a cartografiar

Se elegirá una región concreta que servirá como prueba de concepto para este proyecto. De cara a la sencillez y uniformidad, se procurará que la región se componga de un conjunto de alrededor de 20 poblaciones de similar tamaño, y que estén interconectadas de tal manera que, en la manera de lo posible, se pueda recorrer la región de extremo a extremo por más de un itinerario distinto sin tener

ningún tramo de carretera en común. El hecho de cartografiar una región pequeña atiende a la razón de que no tiene demasiado sentido realizar una cartografía de una provincia entera, por ejemplo, dado que supondría emplear una cantidad demasiado elevada de tiempo. Se considera más importante centrar el esfuerzo en el desarrollo de la aplicación en sí.

Esta región se representará en forma de grafo, donde las poblaciones y las encrucijadas serán los nodos, y los tramos de carretera serán los arcos. Se presentará un conjunto de CSV que será cargado a una base de datos embebida.

Las filas de la tabla principal de la base de datos representarán la lista de arcos. Cada entrada deberá indicar sus nodos de inicio y fin, la duración en minutos del recorrido y su distancia en kilómetros. En el caso de los tramos que se dirijan a una encrucijada deberán indicar la población más cercana más allá de dicha encrucijada. También deberán contener el nombre de la carretera a la que pertenecen.

Las tablas adicionales que contengan datos de categorización o señalización irán en archivos CSV aparte.

4.4 Persistencia de datos

En la aplicación propuesta habrá dos tipos de datos: Los persistentes (datos que deberán permanecer a pesar de reinicios o migraciones), y los efímeros (los relativos a la sesión en la que se está trabajando, y que en ningún caso tienen relevancia entre una sesión y otra).

Los efímeros serán almacenados en memoria, en estructuras de datos, como son por ejemplo la lista de incidencias, mientras que los persistentes serán almacenados en el soporte físico del sistema que esté ejecutando el programa (Ruiz Larrocha y Ruiz Virumbrales, 2012).

Un ejemplo de dato persistente es el archivo de cartografía, que será un conjunto de archivos CSV que se adjuntarán al programa ejecutable.

4.4.1 Creación de la base de datos

La base de datos se almacenará en el soporte físico donde se esté ejecutando la aplicación. También será persistente. Ésta se creará a partir del esquema que se indique desde la aplicación y se poblará con los datos existentes en los archivos de cartografía.

El proceso de poblado de la base de datos será necesario sólo la primera vez que se ejecute el programa en el dispositivo, quedando persistida para posteriores reinicios de la aplicación.

Sólo se modificará dicha base de datos en el caso de que se decida modificar la cartografía, ya sea por actualización, ampliación o cartografiado de una región distinta.

4.4.2 Esquema de la base de datos

El esquema consistirá en una tabla que contendrá la lista de tramos de la región cartografiada, y otra tabla con las categorizaciones de las carreteras, como se muestra en la Figura 4.9. El identificador de carretera será el punto en común entre ambas tablas, ya que un tramo pertenece a una carretera (ver Figura 4.10).

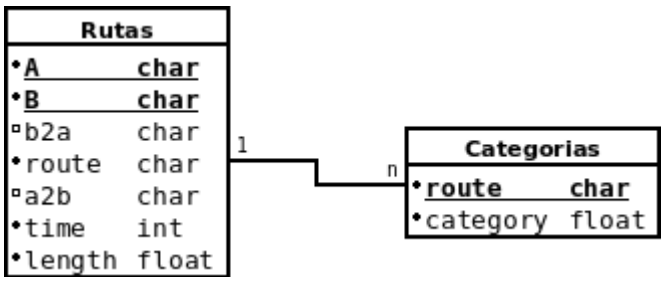


Figura 4.9: esquema de la base de datos

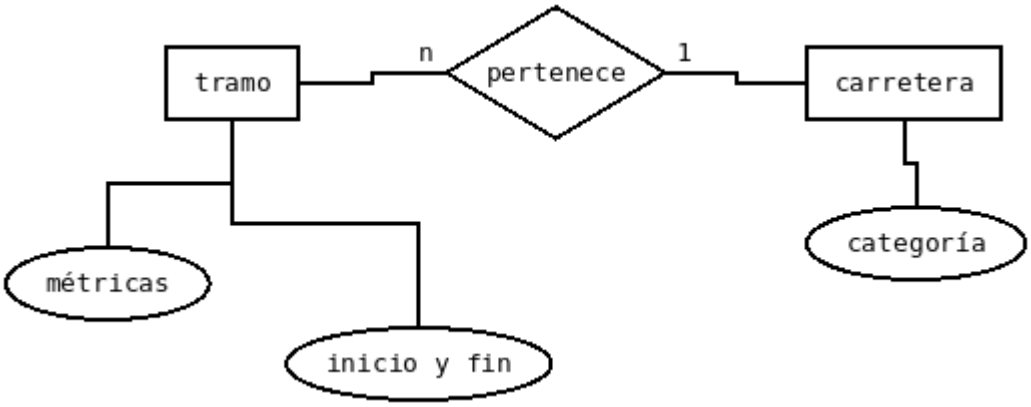


Figura 4.10: modelo Entidad-Relación de la base de datos

4.5 El ciclo de vida de la aplicación

La ingeniería del software ha servido para identificar la forma que adopta el proceso de desarrollo del software. Este proceso se llama *ciclo de vida del software*.

El modelo clásico plantea el desarrollo y explotación como una secuencia de actividades (Cerrada, 2000).

4.5.1 Desarrollo

Se entiende esta fase como la de propiamente elaborar el software. Tras la toma de requisitos y la elaboración de los correspondientes documentos de diseño de alto nivel y de bajo nivel, se procede a plasmar estos documentos en forma de código.

Se usarán metodologías ágiles, ya que según avance el proyecto es posible que el proyecto original sufra modificaciones en alguno de sus aspectos, y se pretende poder adaptar esas modificaciones de manera rápida y eficaz.

Se pretende plasmar un patrón MVC en el que cada paquete controle un aspecto distinto de la aplicación. Se pretende con esto aumentar la cohesión y reducir el acoplamiento en la mayor medida posible (Parsons, 2008):

- El Modelo deberá orquestar la interacción de los paquetes y clases.
- La Vista será la encargada de presentar la parte visual (GUI) y controlará la interacción con el usuario.
- El Controlador definirá las interacciones con la base de datos, el comportamiento de las clases y las administrará.

Las estructuras de datos y algoritmos elaborados deberán tener en cuenta criterios de eficiencia. A su vez, se deberá vigilar que el código presentado sea correcto, cumpla estándares, haga un uso correcto de los recursos, y no provoque problemas al dispositivo donde esté corriendo.

El código deberá estar debidamente comentado. Idealmente los comentarios deberán estar en inglés, así como los nombres de variables, clases y paquetes. La idea es que cualquier ingeniero de cualquier procedencia que tenga que realizar modificaciones al código sea capaz de entenderlo.

4.5.2 Integración continua

Se establecerán unas etapas (*sprints*) en las que se fijará un objetivo. Al alcanzar dicho objetivo se dejará constancia de la fecha en la que se alcanzó el objetivo, la manera de alcanzarlo, y en el caso de que hayan surgido complicaciones o se haya decidido alterar la hoja de ruta, explicarlo y/o justificarlo.

El código será repositado periódicamente en una plataforma de versionado de tipo Git. Esto servirá para poder portar el proyecto de una máquina a otra, si fuese necesario, y a su vez, poder recuperar versiones anteriores en caso de que apareciesen fallos en determinados subsistemas consecuencia de implementaciones posteriores.

Al final de cada *sprint* se comprobará que todas las implementaciones realizadas en *sprints* anteriores siguen cumpliendo su función. Se dedicará un *sprint* para realización de pruebas a la finalización de la fase de codificado.

Se valorarán posibles mejoras y añadidos para posibles revisiones y versiones del producto.

4.5.3 Despliegue

A la finalización del proceso de codificado se procederá a empaquetar el producto de manera que pueda ser ejecutado en cualquier dispositivo que cumpla con las especificaciones mínimas requeridas. Se pretende que el producto sea multiplataforma, con lo que debería ser lo más autónomo posible en lo que se refiere a dependencias.

Debido a que se entregará un producto autoinstalable, se debe aportar un archivo ejecutable acompañado de los archivos de cartografía pertinentes. Debe ser suficientemente sencillo de instalar para que cualquier usuario con los permisos pertinentes en el sistema pueda ejecutarlo sin necesidad de instrucciones, o por lo menos, con las menos posibles.

4.5.4 Operación

La operación del producto debe ser intuitiva y sencilla. No debería de necesitar explicaciones, o al menos, no demasiado exhaustivas.

4.5.5 Monitorización

Al ser un producto embebido no hay posibilidad de realizar una monitorización remota. El personal encargado de operar el producto deberá informar de posibles fallos o deficiencias encontradas. Si la solución a dichos problemas es poco costosa en tiempo se puede considerar una actualización menor del software. Si la solución requiere una factorización profunda se debería esperar al lanzamiento de la siguiente versión del producto.

4.6 Conclusiones

En este capítulo se ha abordado desde el punto de vista del diseño de bajo nivel la manera de ejecutar el proyecto. Se describe de manera esquemática el diseño, la estructura que debe tener y la metodología a utilizar.

Se recalca el hecho de que, al ser metodologías ágiles, el proyecto está sujeto a modificaciones según se crean convenientes, o se suponga que puedan aportar cierto valor al producto final.

Con este capítulo finaliza la fase de diseño del proyecto. Los siguientes capítulos describirán el proceso de desarrollo.

Capítulo 5: Implementación y pruebas

5.1 Introducción

Hasta el momento se ha estado hablando del proyecto en términos teóricos, de su diseño, y de las intenciones y pretensiones que se quieren lograr con él. A partir de este punto se empieza a hablar del proyecto de manera práctica, mostrando problemas y situaciones reales que se han afrontado durante su desempeño, y cómo ha ido evolucionando según el desarrollo avanza.

En este capítulo se describe el proceso de ejecución del proyecto. El entorno, la evolución del concepto original hasta el final, los problemas encontrados y la batería de pruebas realizadas.

5.2 Entorno de desarrollo utilizado

El proyecto comienza con la selección del entorno a utilizar. Ya se cuenta con el entorno hardware, ya que es algo que viene autoimpuesto desde incluso antes del comienzo del proyecto, pero aún así hay un trabajo por delante en lo que respecta a elegir lenguaje, IDE, repositorio, base de datos y esquema de despliegue. En la mayoría de los casos se ha optado por la herramienta con la que el autor ha trabajado más y se encuentra más cómodo.

5.2.1 Herramientas software

Lenguaje

La primera decisión es la relativa a la elección del lenguaje. El autor, en su entorno laboral, ha trabajado con Python y ha hecho pequeños desarrollos en este lenguaje. También, en el pasado ha tenido la oportunidad de trabajar con C, y sobre todo con Java. Se analizan también otros lenguajes muy usados en la actualidad como Perl o Scala.

Finalmente, se elige Java, usando OpenJDK 11.0.7. La razón es la amplia documentación que hay sobre este lenguaje, el hecho de haber tenido la oportunidad de trabajar con él en el pasado, y sobre todo, las librerías existentes que facilitan la creación de un entorno gráfico usable.

IDE

Existe una gran cantidad de entornos de desarrollo disponibles donde elegir. Se busca sobre todo que traiga herramientas específicas para trabajar con el lenguaje elegido, incluyendo precompilador. Tratándose de Java se encuentran en situación muy ventajosa IntelliJ y Eclipse.

Sin embargo, el hecho de que NetBeans integre la facilidad para crear un GUI, facilite poder crear y trabajar con bases de datos, y además favorezca el uso del patrón MVC, definitivamente sirve para tomar una decisión final al respecto.

Sin embargo, en determinados aspectos se ha utilizado Eclipse (corrección de errores y análisis de código).

IntelliJ, que es un excelente IDE, resultó dar problemas con el hardware, con lo que finalmente se acabó descartando.

Base de Datos

Se barajan opciones como MySQL, PostgreSQL o MongoDB. Puesto que será una base de datos sencilla, donde habrá a lo sumo dos o tres tablas, se opta por DerbyDB (JavaDB), que viene integrada con NetBeans. Inicialmente la idea es instalar un servidor de bases de datos en el sistema donde vaya a ejecutarse la aplicación. Se contemplan opciones como adjuntar un script en Bash con comandos que instalen el servidor, creen la base de datos y la pueblen. Puesto que este enfoque puede resultar demasiado agresivo, dando lugar a posibles conflictos, o aumentar los requerimientos del sistema, se estudian otras opciones.

Se considera crear un contenedor con Docker que contenga el servidor de bases de datos, y que la aplicación se conecte de manera estática al contenedor. Sin embargo esta opción sigue creando dependencias, como por ejemplo, instalar Docker.

Finalmente se opta por la versión embebida de DerbyDB. Tan solo necesita almacenamiento en el soporte físico donde se vaya a ejecutar el sistema, y la interacción con la misma a nivel de modificación de la cartografía resulta relativamente sencillo y no requeriría de uso de comandos.

Repositorio

Se busca un repositorio de tipo Git gratuito donde poder ir guardando las versiones del código. Entre Github y Bitbucket, siendo ambos bastante válidos, se opta por este último, debido a la mayor flexibilidad que permite.

Otras herramientas

Este documento ha sido creado con LibreOffice writer 6.4.3.2

Los diagramas han sido creados con GIMP 0.97, salvo los de Gantt, creados con la versión trial de Gantter (gantter.com).

5.2.2 Hardware

El proyecto se inició con un equipo HP EliteBook 8460p con Ubuntu 18.04.

- Procesador: Intel i5-2520M 2.50GHz. dual core.
- Memoria: 6 Gb. DDR3.

Por causas de fuerza mayor el proyecto se tuvo que migrar y finalizar en un HP Pavilion Power Laptop 15-cb0xx que en principio tenía Ubuntu 19.10 y fue migrado a Ubuntu 20.04.

- Procesador: Intel i7-7700HQ 2.80GHz. quad core.
- Memoria: 16 Gb. DDR4.

5.3 Cartografiado

Se establece como zona a cartografiar la comprendida por los municipios de la comarca Sierra Oeste de Madrid (Figuras 5.1 y 5.2). Los motivos son los siguientes:

Figura 5.1: localización de la comarca en la Comunidad de Madrid (SO)

Figura 5.2: división municipal de la comarca (SO)

- Es la zona en la que vive el autor de este proyecto y por tanto posee un conocimiento relativamente amplio de dicha comarca.
- La zona comprende un total de 19 municipios interconectados entre sí por una red de carreteras, según la web http://www.sierraoeste.org/. Esto supone elaborar un grafo bastante amplio, con un máximo de 6 enlaces por nodo, lo cual resulta ideal como prueba piloto del proyecto. A su vez, tiene un tamaño tal que su cartografiado no absorberá demasiado tiempo.
- Las poblaciones son pequeñas, lo cual supone que en condiciones normales se tienen que atravesar para ir de un municipio a otro, salvo algunas excepciones que existe alguna circunvalación o variante. Incorporar ciudades más grandes con circunvalaciones, múltiples destinos e incluso autovías urbanas habría supuesto una complicación innecesaria en la fase de cartografiado. De este modo se obtiene una malla más o menos homogénea, con distintos tipos de carreteras, e incluso accidentes geográficos que hacen que una distancia más corta no suponga menor tiempo de trayecto, lo cual será ideal para demostrar la eficacia del algoritmo.

Los municipios que comprenden la comarca son los siguientes: Aldea del Fresno, Cadalso de los Vidrios, Cenicientos, Chapinería, Colmenar del Arroyo, Fresnedillas de la Oliva, Navalagamella, Navas del Rey, Pelayos de la Presa, Robledo de Chavela, Rozas de Puerto Real, San Martín de Valdeiglesias, Santa María de la Alameda, Valdemaqueda, Villa del Prado, Villamanta, Villamantilla, Villanueva de Perales y Zarzalejo. Para este proyecto se excluyen las pedanías y urbanizaciones, puesto que no están catalogados, y en la mayoría de los casos supondría simplemente añadir un nodo “sin hijos”, ya que suelen tener una única carretera de acceso, y aportan muy poco valor al desarrollo del proyecto.

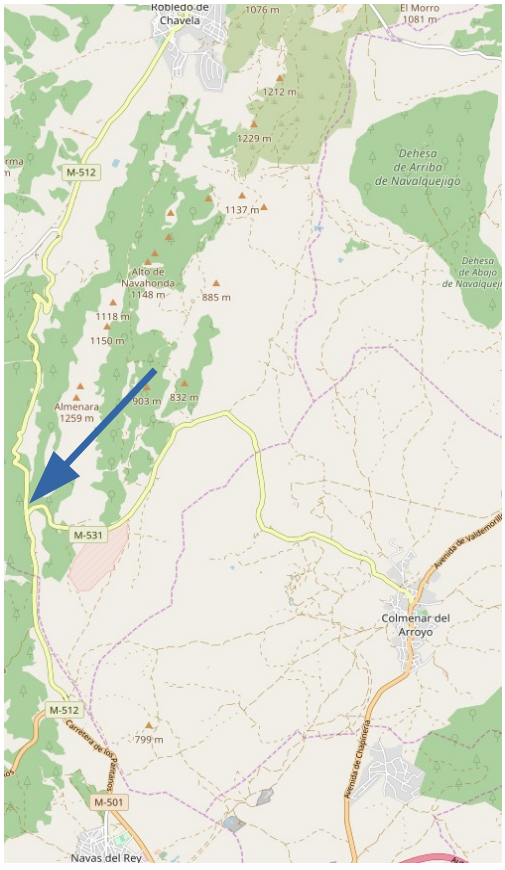


Figura 5.3: ejemplo de encrucijada entre la M-512 y la M-531 (OSM)

En un principio se calcularían las distancias y tiempos de recorrido exclusivamente entre núcleos de población. Sin embargo, se dan muchos casos en los

que, desde una población, y usando una misma carretera, puede alcanzar dos o más poblaciones directamente sin atravesar ninguna otra intermedia, con lo que se cree conveniente incluir como nodos también las encrucijadas (Figura 5.3).

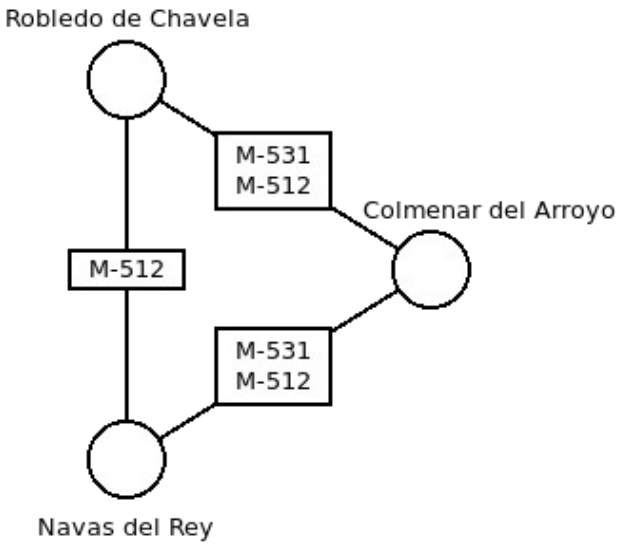


Figura 5.5: grafo uniendo únicamente poblaciones

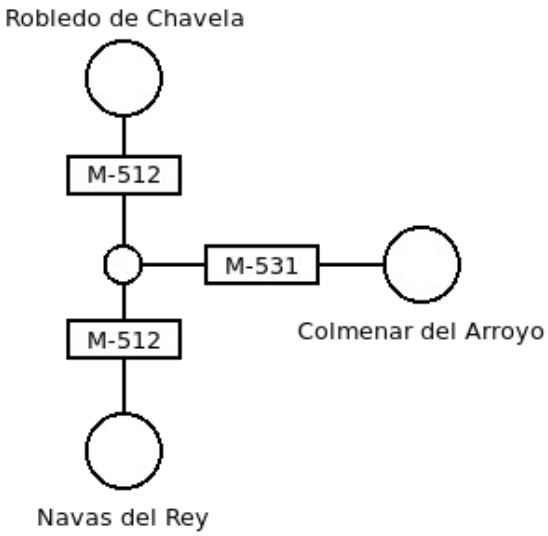


Figura 5.4: grafo uniendo poblaciones a través de encrucijadas.

El uso de las encrucijadas, de cara a dar indicaciones al usuario, resulta más aclaratorio. Si un usuario se encuentra, por ejemplo, en Robledo de Chavela y quiere dirigirse a Colmenar del Arroyo, entenderá con más claridad las indicaciones “Diríjase por la M-512 en dirección a Navas del Rey y luego en el cruce diríjase por la M-531 en dirección a Colmenar del Arroyo” (Figura 5.5) que “Diríjase por la M-512/M-531 a Colmenar del Arroyo” (Figura 5.4).

Usar las encrucijadas en el mapa, sin embargo, supone un problema: La mayoría no tienen nombre y no están señalizadas como tales, con lo que aunque se les ponga un nombre para identificarlas (se ha usado el de cualquier accidente geográfico o sitio cercano), ese nombre no se puede usar para las indicaciones ya que el usuario no tiene por qué conocer el nombre asignado. Esto fuerza a que cuando se muestren indicaciones a un usuario hacia una

Figura 5.6: señalización de poblaciones (GSV)

encrucijada hay que indicarle el de la población a la que lleva la carretera, aunque no tenga que llegar a dicha población.

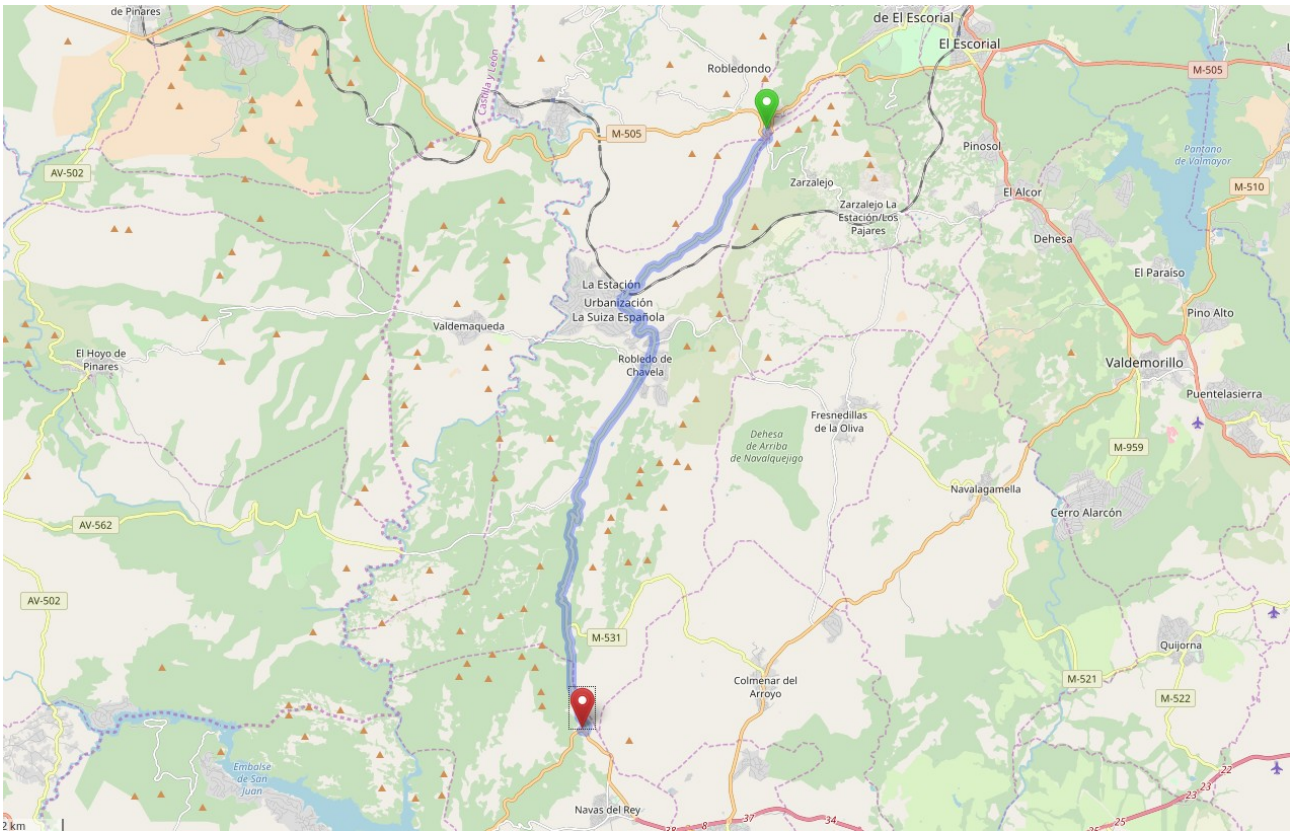


Figura 5.7: trazado de la carretera M-512 (OSM)

Un buen ejemplo es la carretera M-512 (Figura 5.7), que comienza en el Puerto de la Cruz Verde, donde se cruza con las carreteras M-505 y M-533; y termina en un cruce con la carretera M-501, antes de llegar a Navas del Rey. Si se observa la señalización de la zona, como en la Figura 5.6, se ve que se indican las direcciones más allá de dichos cruces, como se puede observar en el mapa de la Figura 5.7.

Para solucionar este problema hay que indicar en las rutas con origen o destino de una encrucijada cuál es la población que se encuentra más allá de la misma, y esto debe quedar reflejado tanto en la cartografía como en la base de datos.

Finalmente, se añaden al cartografiado dos tipos de métricas para cada ruta: Tiempo de recorrido en minutos, y distancia. De esta manera el usuario podrá elegir qué criterio elegir para calcular el itinerario y apreciar el funcionamiento del algoritmo.

Además, se añadirá un factor de categorización de la ruta. En la cartografía realizada existen varios tipos de carretera, a saber:

- Una autovía, la M-501 (Figura 5.8), desde su origen hasta Navas del Rey (a partir de Navas del Rey transcurre como carretera autonómica de primer orden).

Figura 5.8: tramo de autovía de la M-501 (GSV)

- Una carretera nacional, la N-403 (Figura 5.9).

Figura 5.9: carretera N-403 (GSV)

- Dos carreteras de la red principal autonómica, la M-501 (Figura 5.10) y la M-505.

Figura 5.10: M-501 en su tramo como carretera (GSV)

- Tres carreteras de la red secundaria autonómica, la M-507 (Figura 5.11), la M-510 y la M-512.

Figura 5.11: carretera M-507 (GSV)

- Diecisiete carreteras locales:
  - M-521
  - M-523
  - M-524
  - M-530
  - M-531
  - M-532
  - M-533
  - M-535
  - M-537
  - M-538
  - M-541
  - M-542
  - M-544
  - M-545
  - M-546
  - M-854

- M-855 (Figura 5.12), la cual tiene la particularidad de ser de sentido único en la mayor parte de su trazado.

Figura 5.12: carretera M-855, carretera de un solo sentido (GSV)

Estos distintos tipos de carretera se usarán para la categorización, de cara a la posible introducción de una tercera métrica distinta de las mencionadas basadas en tiempo y distancia, dando un factor de preferencia a las carreteras de mayor categoría sobre rutas equivalentes usando carreteras de menor categoría.

La Figura 5.13 muestra el área cartografiada.

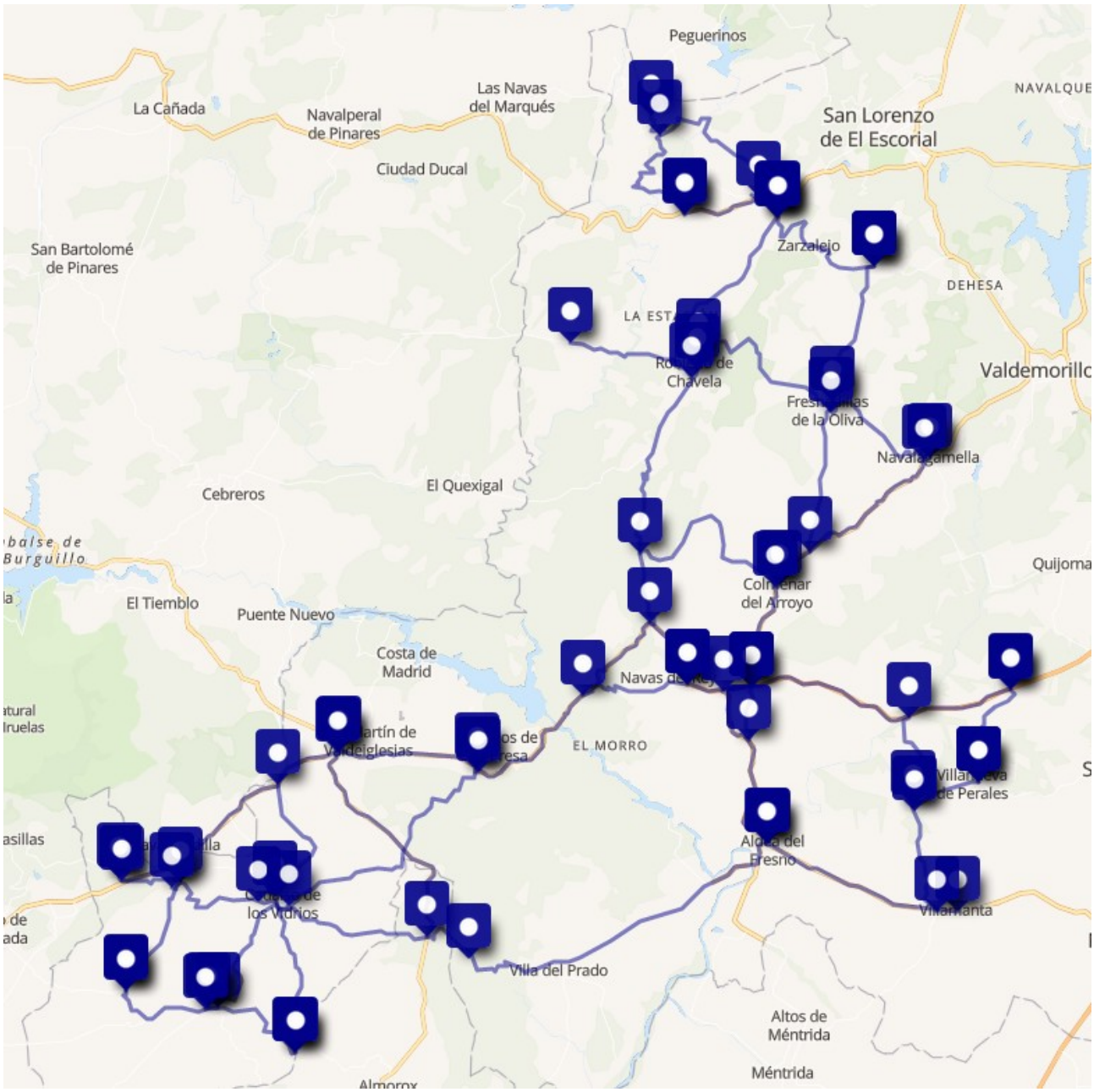


Figura 5.13: carreteras cartografiadas (OSM)

5.4 Creación y conexión a la base de datos

Usando las instrucciones de Apache Derby (DRM) se crea una base de datos usando el Código 5.1.

Cada fila de la tabla de datos representa un tramo simple de una carretera entre dos nodos contiguos (A y B), cuyos nombres representan la clave primaria de la tabla (se supone que no debe haber más de un tramo de carretera que una dos nodos adyacentes). El campo “route” contiene el identificador de carretera. Los campos “time” y “length” contienen las métricas en tiempo y longitud del tramo.

```
CREATE TABLE Routes (
  A         VARCHAR(50) not null,
  B         VARCHAR(50) not null,
  b2a       VARCHAR(50),
  route     VARCHAR(4) not null ,
  a2b       VARCHAR(50),
  time      NUMERIC not null ,
  length    DECIMAL(3,1) not null ,
  PRIMARY KEY (a,b)
);
CREATE TABLE Categorization (
  route     VARCHAR(4) not null ,
  length    DECIMAL(2,1) not null ,
  PRIMARY KEY (route)
);
```

Código 5.1: creación de la base de datos de rutas

Los campos “a2b” y “b2a” se usan para indicar destinos relevantes más allá del nodo de principio o final en el caso de que éste sea una encrucijada y no una población.

Se añaden datos a la base de datos usando el archivo de cartografía, usando el Código 5.2.

```
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE
   (null,'ROUTES','rutas.csv',',',null,null,0);
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE
   (null,'CATEGORIES','categorias.csv',',',null,null,0);
```

Código 5.2: ingesta de datos desde el archivo de cartografía a la base de datos

Para la conexión, se crea una clase que crea una conexión a la base de datos (Código 5.3), y otra que ejecutará las queries usando la conexión previamente creada (Código 5.4).

```
            this.con = DriverManager.getConnection(host, uName, uPass);
```

Código 5.3: conexión a la table de datos

```
        Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
            String SQL = Query;
            ResultSet rs = stmt.executeQuery( SQL );

            int i=0;
            while (rs.next()){
                i++;
            }
            String ret[] = new String[i];
            i=0;
            rs.first();
            do {
                ret[i] = rs.getString(cols);
                i++;
            } while (rs.next());
```

Código 5.4: consulta lanzada a la base de datos

5.5 Desarrollo del algoritmo

Figura 5.14: carreteras que atraviesan Cadalso de los Vidrios (OSM)

Según lo comentado en la sección 2.3.2, se pretende que cada nodo construya su tabla de rutas como si fuese un enrutador de red. Lo primero que hace es preguntar a la base de datos por sus vecinos directamente conectados, y en base a la información recibida comienza a completar la tabla. Tomando como ejemplo Cadalso de los Vidrios (Figura 5.14), su tabla de rutas quedaría como se ve en la Tabla 5.1. Nótese que el aspecto es muy similar al de una tabla de rutas de un enrutador de red. Cada entrada representa un destino, carretera, métrica optimizada, distancia en kilómetros y tiempo en minutos.

```
Cadalso de los Vidrios
 Partida Navarredonda M542 12.59 7.0 7
 Arroyo del Linar M541 12.42 6.9 9
 Almorox M507 10.88 6.8 7
 Enebrales M542 11.87 6.6 12
 Cenicientos M541 11.7 6.5 9
 Rozas de Puerto Real M507 9.12 5.7 7
```

Tabla 5.1: tabla de rutas directamente conectadas de Cadalso de los Vidrios

A continuación, pregunta a cada uno se sus vecinos por su tabla de rutas. Estos le responden y se analizan las rutas una a una. Con cada ruta analizada hay tres posibles escenarios:

- Que la ruta sea hacia el propio nodo que pregunta. Esta ruta se descarta.
- Que la ruta sea hacia un nodo que ya existe en la tabla del nodo que pregunta. En este caso, tomando en cuenta el criterio de selección de ruta elegido (distancia, tiempo u optimizado), pueden pasar dos cosas:
    - La ruta ofrecida tiene una métrica mejor que la ya existente. En este caso se reemplaza una por otra.
    - La ruta ofrecida tiene una métrica peor. En este caso se descarta.
- Que la ruta no exista en la tabla de rutas. En este caso se inserta actualizando la métrica.

Existe el período de convergencia, y es en el que todos los nodos están continuamente consultándose y actualizando sus tablas de rutas. Se considera que el grafo entero ha convergido cuando, tras una iteración completa, ninguno de los nodos ha realizado ninguna modificación en su tabla de rutas.

La distinta elección de criterio muestra diferencias en las tablas de enrutamiento para un mismo destino. Las Tablas 5.2, 5.3 y 5.4 representan las tablas de enrutamiento del mismo nodo usando un criterio distinto de búsqueda. Se añade el campo “dirección” al final, que es la dirección que hay que seguir en la señalización. Este campo, en el caso de tramos que terminan en una población, coincide con el nombre de la misma; y en el caso de tramos que terminan en una encrucijada, contiene el nombre de la población más allá del cruce. Se muestran en rojo las diferencias para el mismo destino (Villamantilla) dependiendo del criterio elegido.

```
Cadalso de los Vidrios
 Partida Navarredonda M542 12.59 7.0 7 San Martín de Valdeiglesias
 Arroyo del Linar M541 12.42 6.9 9 Pelayos de la Presa
 Almorox M507 10.88 6.8 7 Villa del Prado
 Enebrales M542 11.87 6.6 12 Almorox
 Cenicientos M541 11.7 6.5 9 Cenicientos
 Rozas de Puerto Real M507 9.12 5.7 7 Rozas de Puerto Real
 San Martín de Valdeiglesias M542 17.5 10.5 10 San Martín de Valdeiglesias
 Arroyo Avellaneda M507 12.96 8.1 10 Rozas de Puerto Real
 Pelayos de la Presa M541 22.68 12.6 21 Pelayos de la Presa
 Villa del Prado M507 23.36 14.6 16 Villa del Prado
 Diseminado Diseminados M507 18.66 11.0 15 Rozas de Puerto Real
 Arroyo Vallefrías M541 31.48 18.9 26 Pelayos de la Presa
 Arroyo Valdezate M541 38.22 23.69 30 Pelayos de la Presa
 Navas del Rey M541 41.22 24.31 33 Pelayos de la Presa
 Aldea del Fresno M507 40.83 25.5 28 Villa del Prado
 Arroyo del Santo M541 47.14 28.6 38 Pelayos de la Presa
 Villamanta M507 53.44 33.38 36 Villa del Prado
 Chapinería M541 44.32 27.4 37 Pelayos de la Presa
 Chapinería oeste M541 42.82 25.9 35 Pelayos de la Presa
 Villamantilla M541 59.26 38.9 47 Pelayos de la Presa
 NASA M541 43.18 26.81 33 Pelayos de la Presa
 Colmenar del Arroyo M541 51.839996 32.12 42 Pelayos de la Presa
 Robledo de Chavela M541 59.02 36.7 45 Pelayos de la Presa
 Río Perales M541 51.519997 34.62 42 Pelayos de la Presa
 Valle del Sol M541 55.17 34.2 45 Pelayos de la Presa
 Encrucijada M541 56.119995 39.2 45 Pelayos de la Presa
 Villanueva de Perales M541 64.66 41.9 52 Pelayos de la Presa
 Fresnedillas de la Oliva M541 66.72 40.62 54 Pelayos de la Presa
 Navalagamella M541 65.92 40.9 51 Pelayos de la Presa
 Puerto de la Cruz Verde M541 73.58 45.83 57 Pelayos de la Presa
 Valdemaqueda M541 70.72 43.2 57 Pelayos de la Presa
 Zarzalejo M541 78.96 47.4 61 Pelayos de la Presa
 Robledondo M541 75.68 47.33 59 Pelayos de la Presa
 Santa María de la Alameda M541 90.75 55.7 71 Pelayos de la Presa
 La Paradilla M541 80.25 50.62 62 Pelayos de la Presa
```

Tabla 5.2: tabla de enrutamiento completa de Cadalso de los Vidrios, con el criterio optimizado.

```
Cadalso de los Vidrios
 Partida Navarredonda M542 12.59 7.0 7 San Martín de Valdeiglesias
 Arroyo del Linar M541 12.42 6.9 9 Pelayos de la Presa
 Almorox M507 10.88 6.8 7 Villa del Prado
 Enebrales M542 11.87 6.6 12 Almorox
 Cenicientos M541 11.7 6.5 9 Cenicientos
 Rozas de Puerto Real M507 9.12 5.7 7 Rozas de Puerto Real
 San Martín de Valdeiglesias M542 17.5 10.5 10 San Martín de Valdeiglesias
 Arroyo Avellaneda M507 12.96 8.1 10 Rozas de Puerto Real
 Pelayos de la Presa M541 22.68 12.6 21 Pelayos de la Presa
 Villa del Prado M507 23.36 14.6 16 Villa del Prado
 Diseminado Diseminados M507 18.66 11.0 15 Rozas de Puerto Real
 Arroyo Vallefrías M541 31.48 18.9 26 Pelayos de la Presa
 Arroyo Valdezate M541 38.22 23.69 30 Pelayos de la Presa
 Navas del Rey M541 41.22 24.31 33 Pelayos de la Presa
 Aldea del Fresno M507 40.83 25.5 28 Villa del Prado
 Arroyo del Santo M541 47.14 28.6 38 Pelayos de la Presa
 Villamanta M507 53.44 33.38 36 Villa del Prado
 Chapinería M541 44.32 27.4 37 Pelayos de la Presa
 Chapinería oeste M541 42.82 25.9 35 Pelayos de la Presa
 Villamantilla M507 62.44 38.38 42 Villa del Prado
 NASA M541 43.18 26.81 33 Pelayos de la Presa
 Colmenar del Arroyo M541 51.84 32.12 42 Pelayos de la Presa
 Robledo de Chavela M541 59.02 36.7 45 Pelayos de la Presa
 Río Perales M541 51.52 34.62 42 Pelayos de la Presa
 Valle del Sol M541 55.17 34.2 45 Pelayos de la Presa
 Encrucijada M541 56.12 39.2 45 Pelayos de la Presa
 Villanueva de Perales M507 67.84 41.38 47 Villa del Prado
 Fresnedillas de la Oliva M541 66.72 40.62 54 Pelayos de la Presa
 Navalagamella M541 65.92 40.9 51 Pelayos de la Presa
 Puerto de la Cruz Verde M541 73.58 45.83 57 Pelayos de la Presa
 Valdemaqueda M541 70.72 43.2 57 Pelayos de la Presa
 Zarzalejo M541 78.96 47.4 61 Pelayos de la Presa
 Robledondo M541 75.68 47.33 59 Pelayos de la Presa
 Santa María de la Alameda M541 90.75 55.7 71 Pelayos de la Presa
 La Paradilla M541 80.25 50.62 62 Pelayos de la Presa
```

Tabla 5.3: tabla de enrutamiento completa de Cadalso de los Vidrios, con el criterio por distancia.

```
Cadalso de los Vidrios
 Partida Navarredonda M542 12.59 7.0 7 San Martín de Valdeiglesias
 Arroyo del Linar M541 12.42 6.9 9 Pelayos de la Presa
 Almorox M507 10.88 6.8 7 Villa del Prado
 Enebrales M542 11.87 6.6 12 Almorox
 Cenicientos M541 11.7 6.5 9 Cenicientos
 Rozas de Puerto Real M507 9.12 5.7 7 Rozas de Puerto Real
 San Martín de Valdeiglesias M542 17.5 10.5 10 San Martín de Valdeiglesias
 Arroyo Avellaneda M507 12.96 8.1 10 Rozas de Puerto Real
 Pelayos de la Presa M542 25.76 16.4 15 San Martín de Valdeiglesias
 Villa del Prado M507 23.36 14.6 16 Villa del Prado
 Diseminado Diseminados M541 20.16 11.2 14 Cenicientos
 Arroyo Vallefrías M542 34.56 22.7 20 San Martín de Valdeiglesias
 Arroyo Valdezate M542 41.3 27.5 24 San Martín de Valdeiglesias
 Navas del Rey M542 44.3 28.1 27 San Martín de Valdeiglesias
 Aldea del Fresno M507 40.83 25.5 28 Villa del Prado
 Arroyo del Santo M542 51.42 34.83 32 San Martín de Valdeiglesias
 Villamanta M507 53.44 33.38 36 Villa del Prado
 Chapinería M542 48.6 33.6 31 San Martín de Valdeiglesias
 Chapinería oeste M542 47.1 32.1 29 San Martín de Valdeiglesias
 Villamantilla M542 63.54 45.12 41 San Martín de Valdeiglesias
 NASA M542 46.26 30.6 27 San Martín de Valdeiglesias
 Colmenar del Arroyo M542 56.12 38.3 36 San Martín de Valdeiglesias
 Robledo de Chavela M542 62.1 40.5 39 San Martín de Valdeiglesias
 Río Perales M542 55.83 40.83 36 San Martín de Valdeiglesias
 Valle del Sol M542 59.48 40.4 39 San Martín de Valdeiglesias
 Encrucijada M542 60.4 45.4 39 San Martín de Valdeiglesias
 Villanueva de Perales M542 69.22 50.33 43 San Martín de Valdeiglesias
 Fresnedillas de la Oliva M542 76.5 48.5 48 San Martín de Valdeiglesias
 Navalagamella M542 70.25 47.1 45 San Martín de Valdeiglesias
 Puerto de la Cruz Verde M542 76.66 49.62 51 San Martín de Valdeiglesias
 Valdemaqueda M542 73.8 47.0 51 San Martín de Valdeiglesias
 Zarzalejo M542 88.74 55.33 55 San Martín de Valdeiglesias
 Robledondo M542 78.76 51.12 53 San Martín de Valdeiglesias
 Santa María de la Alameda M542 93.88 59.5 65 San Martín de Valdeiglesias
 La Paradilla M542 83.38 54.4 56 San Martín de Valdeiglesias
```

Tabla 5.4: tabla de enrutamiento completa de Cadalso de los Vidrios, con el criterio por tiempo.

En el Código 5.5 se muestra la implementación del algoritmo en Java. El diagrama de flujo aparece en la Figura 5.15.

```
public boolean buildShortestPath(String Criteria){

    boolean finished = true;
    for (int i=0; i<connected; i++) {
        Edge route = routes.get(i);
        for (int j=0; j<allNodes.size(); j++){
            if (route.getDestination().equals(allNodes.get(j).getName())){
                List<Edge> nrt = allNodes.get(j).getRouteTable();
                for (int k=0; k<nrt.size(); k++) {
                    Edge rt = nrt.get(k);
                    if (!rt.getDestination().equals(name)) {
                        boolean existing = false;
                        int m = routes.size();
                        for (int n=0; n<m; n++) {
                            if
         (routes.get(n).getDestination().equals(nrt.get(k).getDestination())) {
                                Boolean update = false;
                                if (Criteria.equals("Length")){
                                    if ((nrt.get(k).getLength()
                                  +route.getLength()) < routes.get(n).getLength()){
                                        update = true;
                                    }
                                } else if (Criteria.equals("Time")){
                                    if ((nrt.get(k).getTime()+route.getTime())
                                         < routes.get(n).getTime()){
                                        update = true;
                                    }
                                } else {
                                    if ((nrt.get(k).getOptimal()
                                +route.getOptimal()) < routes.get(n).getOptimal()){
                                        update = true;
                                    }
                                }
                                if (update) {
          routes.get(n).setRoadLabel(route.getRoadLabel());
          routes.get(n).setOptimal(nrt.get(k).getOptimal()+route.getOptimal());
          routes.get(n).setLength(nrt.get(k).getLength()+route.getLength());
          routes.get(n).setTime(nrt.get(k).getTime()+route.getTime());
          routes.get(n).setOrigin(allNodes.get(j).getBound(name));
                                }
                                existing = true;
                            }
                        }
                        if (!existing) {
                            routes.add(new Route(rt.getDestination(),
                                route.getRoadLabel(),
                                rt.getOptimal()+route.getOptimal(),
                                rt.getLength()+route.getLength(),
                                rt.getTime()+route.getTime(),
                                allNodes.get(j).getBound(name)));
                            finished = false;
                        }
                    }
                }
            }
        }
    }
    return finished;
}
```

Código 5.5: código Java del algoritmo

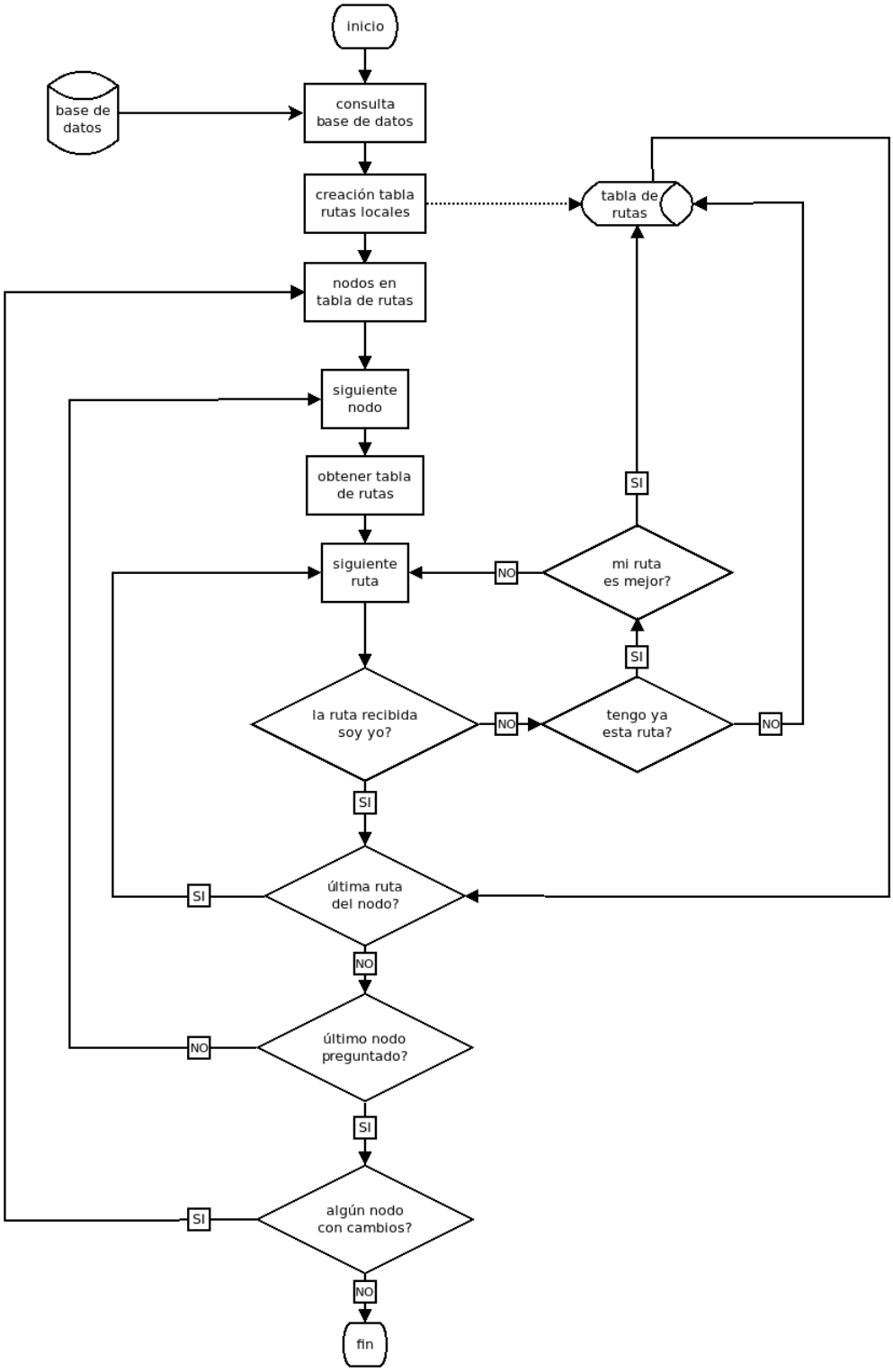


Figura 5.15: diagrama del Algoritmo de Dijkstra

5.6 Recorrido del grafo.

Una vez que la red ha convergido, ir de un punto A a un punto B consistirá en seguir las tablas de rutas de cada nodo por el que atraviese el grafo creado de A a B. Hasta aquí resulta relativamente sencillo. Sin embargo, de cara a mostrar el camino a seguir al usuario, hay ciertas consideraciones que complican el algoritmo.

- Las encrucijadas, como ya se ha mencionado, forman parte del recorrido. Sin embargo deben ser transparentes para el usuario. Se debe enrutar en dirección a la población que haya detrás de la encrucijada.
- Además, si un recorrido por una misma carretera atraviesa una encrucijada, a la hora de presentar los tramos, se deben unificar los tramos anterior y posterior y calcular la suma de sus distancias. Esto evitará ver en el cuadro de texto de guía de ruta entradas que parecen repetidas.

Normalmente, en la traza se nombrarían todos los nodos por los que habría que pasar.

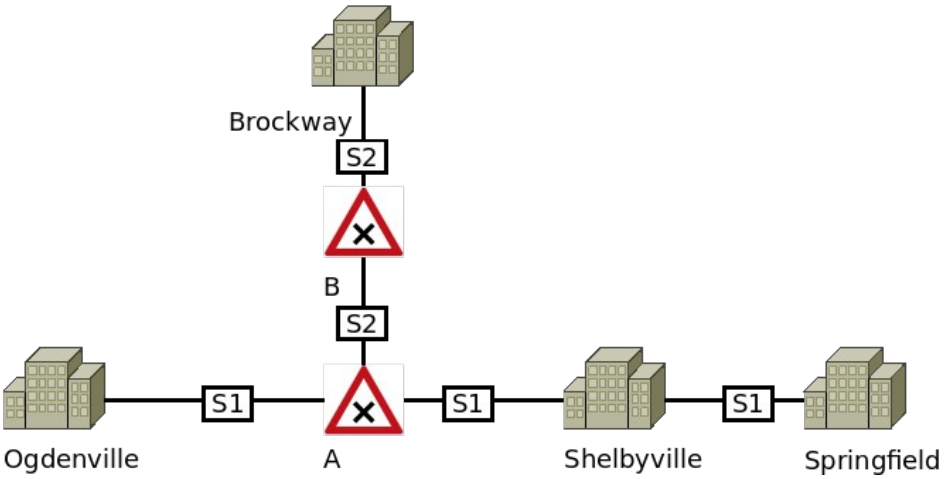


Figura 5.16: ejemplo de ruta atravesando encrucijadas

En el ejemplo de la Figura 5.16, para ir de Springfield a Brockway la traza, normalmente, sería la siguiente:

- Dirigirse a Shelbyville.
- Dirigirse a A.

- Dirigirse a B.
- Dirigirse a Brockway.

Sin embargo, A y B no son necesariamente puntos conocidos por el usuario ni tienen por qué aparecer en la señalización, con lo que habría que usar un trazado por referencias que sí sean reconocibles. Suponiendo que cada tramo de carretera mide *n* kilómetros, una traza lógica sería como sigue:

- Dirigirse en dirección a Shelbyville durante *n* kilómetros.
- Dirigirse en dirección a Ogdenville durante *n* kilómetros.
- Dirigirse en dirección a Brockway durante *n* kilómetros.
- Dirigirse en dirección a Brockway durante *n* kilómetros.

Y, puesto que una vez eliminada la referencia a las encrucijadas se pueden producir redundancias innecesarias, una traza debería quedar como sigue:

- Dirigirse en dirección a Shelbyville durante *n* kilómetros.
- Dirigirse en dirección a Ogdenville durante *n* kilómetros.
- Dirigirse en dirección a Brockway durante *n*+*n* kilómetros.

Con esto se logra una guía mucho más fácil de comprender para el usuario. A la hora de plasmarlo en el código, sin embargo, ha supuesto un verdadero reto. Se ha optado por usar un autómata con pila para guardar el estado y pasarle a la iteración siguiente de información del estado actual, e ir reteniendo la información de ruta respecto a la iteración anterior, y si el tramo actual es repetido, no mostrar nada y simplemente sumar la longitud del tramo (Hopcroft et al., 2008).

La función que realiza el recorrido del grafo es recursiva, y está recogida en el Código 5.6.

A continuación se muestra cómo se comporta ante la ruta:

- La función recibe como argumentos el inicio de la ruta (Springfield), el final (Brockway), el criterio (en este caso es irrelevante), el tramo precedente (ninguno, puesto que es el primer tramo), y la longitud acumulada del tramo anterior (0, puesto que es la primera iteración).
- Se obtiene la tabla de rutas de Springfield y se busca Brockway.

- La tabla de rutas dice que para ir a Brockway hay que ir en dirección a Shelbyville. Se busca la entrada hacia Shelbyville. Se obtiene la distancia del tramo (*n*) y se deposita en la variable *next*.

```
public String getPath(String origin, String destination, Criteria criteria, String previous,
                      float carry){
    float next = 0;
    String head = "";
    if (carry != 0)
        head =" durante " + String.format("%.1f", carry) +" km.\n";
    path = head;
    if (origin != destination) {
        List<Edge> routeTable = RutaSierra.getNodeRouteTable(origin);
        for (int i=0; i<routeTable.size(); i++){
            if (routeTable.get(i).getDestination().equals(destination)){
                for (int j=0; j<routeTable.size(); j++){
                    if (routeTable.get(i).getEnd().equals(routeTable.get(j).getDestination())){
                        next = routeTable.get(j).getLength();
                    }
                }
                if (previous.equals(routeTable.get(i).getSource() +
                                    routeTable.get(i).getRoadLabel()))
                    path = getPath(routeTable.get(i).getEnd(), destination, criteria,
                           routeTable.get(i).getSource() + routeTable.get(i).getRoadLabel(),
                           (carry+next));
                else {
                    if (carry==0)
                        head = "Ruta: "+ String.format("%.1f", routeTable.get(i).getLength()) +
                                " km, "+ routeTable.get(i).getTime() + " minutos.\n\n";
                    path = head +
                           "vaya en dirección a "+ routeTable.get(i).getSource() + " via " +
                           routeTable.get(i).getRoadLabel() +
                           getPath(routeTable.get(i).getEnd(), destination, criteria,
                                   routeTable.get(i).getSource() +
                                   routeTable.get(i).getRoadLabel(), next);

                }
            }
        }
    }
    return path;
}
```

Código 5.6: función de recorrido del grafo

- Se analiza el tramo anterior por si coincide con el actual. Como no es el caso, y la variable *carry* equivale a 0, se muestra el Output 5.1.

```
Ruta: 4n km, 4t minutos.
```

Output 5.1: cabecera de información de ruta

- Se extraen los datos de la entrada de Shelbyville de la tabla de rutas. Se añade esta información a la primera instrucción y como resultado se muestra el Output 5.2.

```
Ruta: 4n km, 4t minutos.

Vaya en dirección a Shelbyville vía S1
```

Output 5.2: primera iteración del recorrido del grafo

- Se llama a la misma función con los argumentos “Shelbyville”, “Brockway”, “criterio”, “ShelbyvilleS1”,”*n*”.
- Puesto que *carry* ya no vale 0, se añade la distancia del tramo anterior a las instrucciones, y el resultado es el Output 5.3.

```
Ruta: 4n km, 4t minutos.

Vaya en dirección a Shelbyville vía S1 durante n km.
```

Output 5.3: inicio de la segunda iteración

- Se obtiene la tabla de rutas de Shelbyville y se busca Brockway en ella.
- La tabla de rutas dice que para ir a Brockway hay que ir en dirección a la encrucijada A. Se busca la entrada hacia dicha encrucijada. Se obtiene la distancia del tramo (*n*) y se deposita en la variable *next*.
- Se extraen los datos de la entrada de la encrucijada A de la tabla de rutas. Puesto que A es una encrucijada se muestra el destino más allá de dicha encrucijada. Se añade esta información a la información existente (Output 5.4).

```
Ruta: 4n km, 4t minutos.

Vaya en dirección a Shelbyville vía S1 durante n km.
Vaya en dirección a Ogdenville vía S1
```

Output 5.4: final de la segunda iteración

- Se llama a la misma función con los argumentos “A”, “Brockway”, “criterio”, “AS1”,”*n*”.
- Se añade la distancia del tramo anterior a las instrucciones (Output 5.5).

```
Ruta: 4n km, 4t minutos.

Vaya en dirección a Shelbyville vía S1 durante n km.
Vaya en dirección a Ogdenville vía S1 durante n km.
```

Output 5.5: inicio de la tercera iteración

- Se obtiene la tabla de rutas de A y se busca Brockway.
- La tabla de rutas dice que para ir a Brockway hay que ir en dirección a la encrucijada B. Se busca la entrada hacia dicha encrucijada. Se obtiene la distancia del tramo (*n*) y se deposita en la variable *next*.
- Se extraen los datos de la entrada de la encrucijada B de la tabla de rutas. Puesto que B es una encrucijada se muestra el destino más allá de dicha encrucijada. Se añade esta información a la información existente (Output 5.6).

```
Ruta: 4n km, 4t minutos.

Vaya en dirección a Shelbyville vía S1 durante n km.
Vaya en dirección a Ogdenville vía S1 durante n km.
Vaya en dirección a Brockway vía S2
```

Output 5.6: final de la tercera iteración

- Se llama a la misma función con los argumentos “B”, “Brockway”, “criterio”, “BS1”,”*n*”.
- Se añade la distancia del tramo anterior a las instrucciones. Sin embargo, esta información no se usará esta vez.
- Se obtiene la tabla de rutas de B y se busca Brockway.
- La tabla de rutas dice que para ir a Brockway hay que ir en dirección a Brockway. Se busca la entrada hacia Brockway, que al ser un nodo conectado directamente, resulta ser la misma entrada. Se obtiene la distancia del tramo (*n*) y se deposita en la variable *next*.

- Se compara el argumento *previous* con el tramo actual. Puesto que coinciden, para evitar introducir información redundante, se llama a la misma función sumando el acumulado de distancia del tramo anterior. Los argumentos son “Brockway”, “Brockway”, “criterio”, “BS1”, “*n*+*n*”.
- Se añade la distancia del tramo anterior a las instrucciones. La traza queda como muestra el Output 5.7.

```
Ruta: 4n km, 4t minutos.

Vaya en dirección a Shelbyville vía S1 durante n km.
Vaya en dirección a Ogdenville vía S1 durante n km.
Vaya en dirección a Brockway vía S2 durante n+n km.
```

Output 5.7: inicio de la quinta iteración y final de la función

- Se comparan origen y destino. Puesto que coinciden la función finaliza.

Llevándolo a un ejemplo práctico en la topología real:

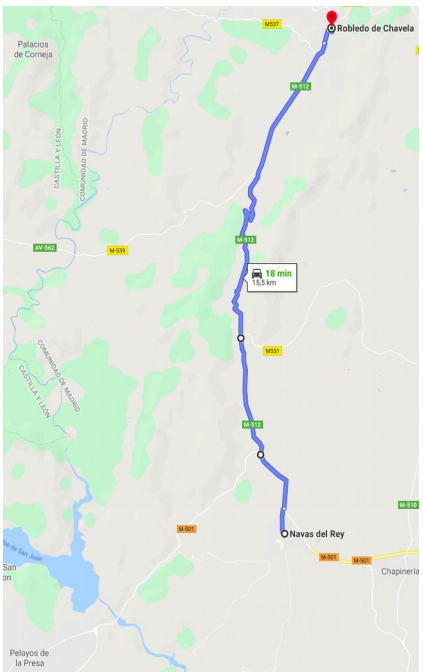


Figura 5.17: ejemplo de recorrido atravesando encrucijadas (GSV)

Para ir de Navas del Rey a Robledo de Chavela (Figura 5.17) hay que atravesar dos encrucijadas. En una de ellas, además, hay que cambiar de carretera. Lo primero es dirigirse por la carretera M-501 en dirección al cruce con la carretera M-512 (a esta encrucijada la hemos llamado Arroyo Valdezate, Figura 5.18). El conductor que se encuentre en Navas del Rey puede coger la M-501 o bien en dirección a Chapinería, o bien en dirección a Pelayos de la Presa. Puesto que Pelayos de la Presa es la siguiente población que hay tras el cruce con la M-512, hacemos al usuario coger la M-501 en dirección a Pelayos de la Presa durante 3 kilómetros.

Esto, en el algoritmo, significa saber qué población hay tras cada encrucijada (ya venía planteado cuando se diseñó la base de datos), y además, en la siguiente iteración del algoritmo,

tener en cuenta que hay que consultar la tabla de rutas de la encrucijada y no del destino (es decir, consultar la tabla de rutas de Arroyo Valdezate en vez de Pelayos de la Presa).

Figura 5.18: cruce entre la M-501 y la M-512 (GSV)

Una vez en la M-512 se atravesará la encrucijada con la M-531 (Figura 5.19). Sin embargo, el usuario debe ignorar este cruce, ya que debe seguir por la M-512 y no desviarse. Sobre el grafo está recorriendo dos arcos distintos de la M-512, pero en el resultado debe aparecer como un único tramo con la suma de las distancias de los dos tramos.

Figura 5.19: cruce entre la M-512 y la M-531 (GSV)

El hecho de que haya que elegir una ruta según el criterio motiva que la convergencia de tablas de rutas entre los nodos, que se realizaba en la inicialización del programa, se tenga que realizar cada vez que el usuario intenta hacer una consulta, ya que las tablas de rutas varían en función del criterio. Sin embargo, para la posterior gestión de incidencias esto será una ventaja.

5.7 Criterios de búsqueda

La búsqueda del mejor camino entre dos poblaciones responde a tres posibles criterios distintos:

- Itinerario más corto: Basado en el camino con la menor suma de distancias en los tramos que lo comprende.
- Itinerario más rápido: Basado en el camino con la menor suma de tiempos en los tramos que lo comprende.
- Itinerario óptimo: Basado en el camino con la menor suma de distancia por cuantificador de calidad en los tramos que lo comprende.

Es bastante obvio en qué se basan los dos primeros criterios. El tercer criterio está basado en un cálculo sencillo que consiste en multiplicar cada tramo por un cuantificador de calidad basado en la categoría de la carretera a la que pertenece. La Tabla 5.5 muestra el valor asignado de dicho cuantificador para cada categoría.

Tipo de vía	Factor de calidad
Autovía	×1
Carretera nacional	×1.2
Carretera autonómica de primer orden	×1.4
Carretera autonómica de segundo orden	×1.6
Carretera autonómica de tercer orden	×1.8

Tabla 5.5: categorización de las rutas

La necesidad de incluir este criterio viene dada por la preferencia de determinados conductores por realizar el viaje usando carreteras principales, aunque supongan más kilómetros o más tiempo de viaje, debido a la mayor capacidad que éstas ofrecen, o simplemente porque el trazado o el firme de las mismas suele ser mejor comparado con el de las vías de peor categoría. En muchos casos coincide con cualquiera de los otros dos criterios, y no tiene por qué suponer una mejora de las métricas. Se podría decir que este criterio lo que aporta es ahorro de combustible, en términos de que busca una ruta corta pero evitando en la medida de lo posible las carreteras de peor categoría.

Para ver un ejemplo, realizaremos una traza usando los tres criterios entre dos poblaciones y compararemos los resultados.

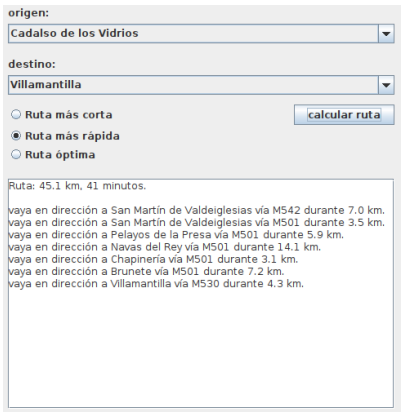


Figura 5.21: traza por criterio rápido

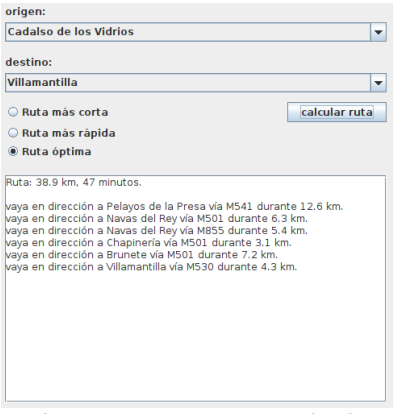


Figura 5.20: traza por criterio óptimo

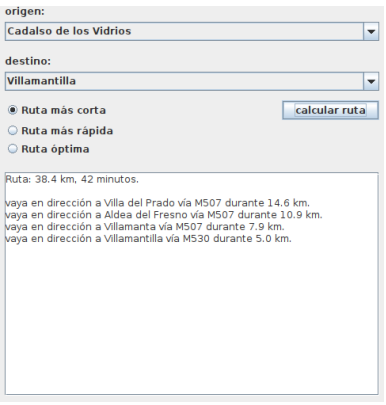


Figura 5.22: traza por criterio corto

En las Figuras 5.20, 5.21 y 5.22 se muestra el resultado de buscar el mismo camino con los tres distintos criterios. Si trasladamos esos resultados a un mapa y los superponemos el resultado es el de la Figura 5.23.

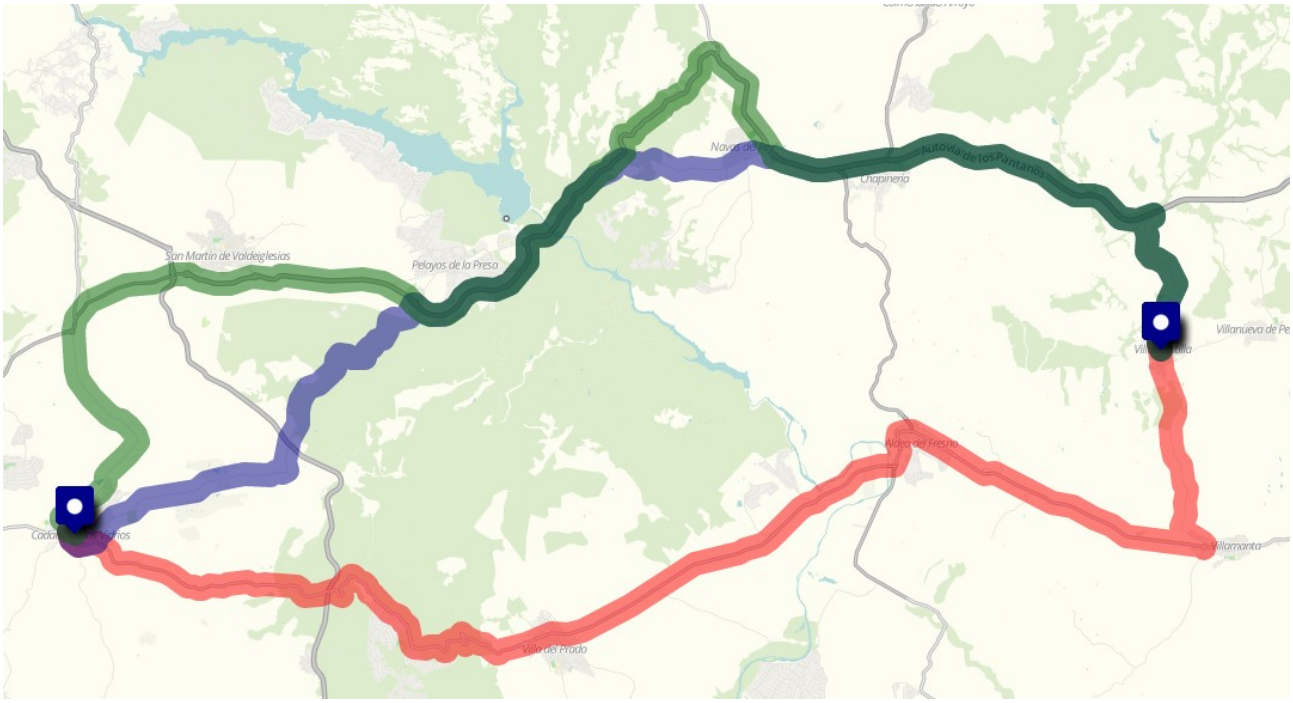


Figura 5.23: resultado de las tres trazas superpuestas (OSM)

En el mapa se muestra en rojo la traza más corta, la cual transcurre en su mayoría por una comarcal de segundo orden, la M-507. La traza en azul corresponde a la ruta óptima, que es ligeramente más larga que la más corta (menos de un kilómetro), pero da preferencia al hecho de utilizar la M-501, que es una carretera cuyo trazado es en parte comarcal de primer orden y en parte autovía. Finalmente, la traza en verde corresponde a la ruta más rápida, que es la más larga de las tres, la cual discurre aún durante más distancia por la M-501. Comparada con ésta, se ve cómo la ruta óptima realiza un trazado similar, pero realizando ciertos “atajos”.

Ajustando los valores del factor de calidad se puede lograr una mayor o menor tolerancia a discurrir por carreteras secundarias.

5.8 Gestión de incidencias

Una incidencia es algo que sucede de manera aleatoria, con lo que plasmarlo en este proyecto supondría algo tan sencillo como, al azar, eliminar un tramo (arco entre nodos) y dejar que el algoritmo recalcule.

Sin embargo, de cara al uso de la aplicación por un posible centro de control de carreteras, una incidencia es algo que detecta un sensor, un usuario, un operario o un agente de la ley en la red de carreteras, y en consecuencia avisa para que se tomen las medidas oportunas.

Figura 5.24: teleindicador previo a un cruce con la M-501 (GSV)

De cara al centro de control, esta incidencia no se genera aleatoriamente, sino que un operario, o el módulo de recepción de incidencias, debe recibir e interpretar los datos de que disponga (informes

de atestados, cámaras de vigilancia, etc) y a continuación tomar las medidas oportunas. Éstas incluyen reflejar dicha incidencia en la topología para que los cálculos posteriores tengan en cuenta dicho cambio, e informar a los usuarios mediante mensajes (teleindicadores luminosos de carretera, como el de la Figura 5.24).

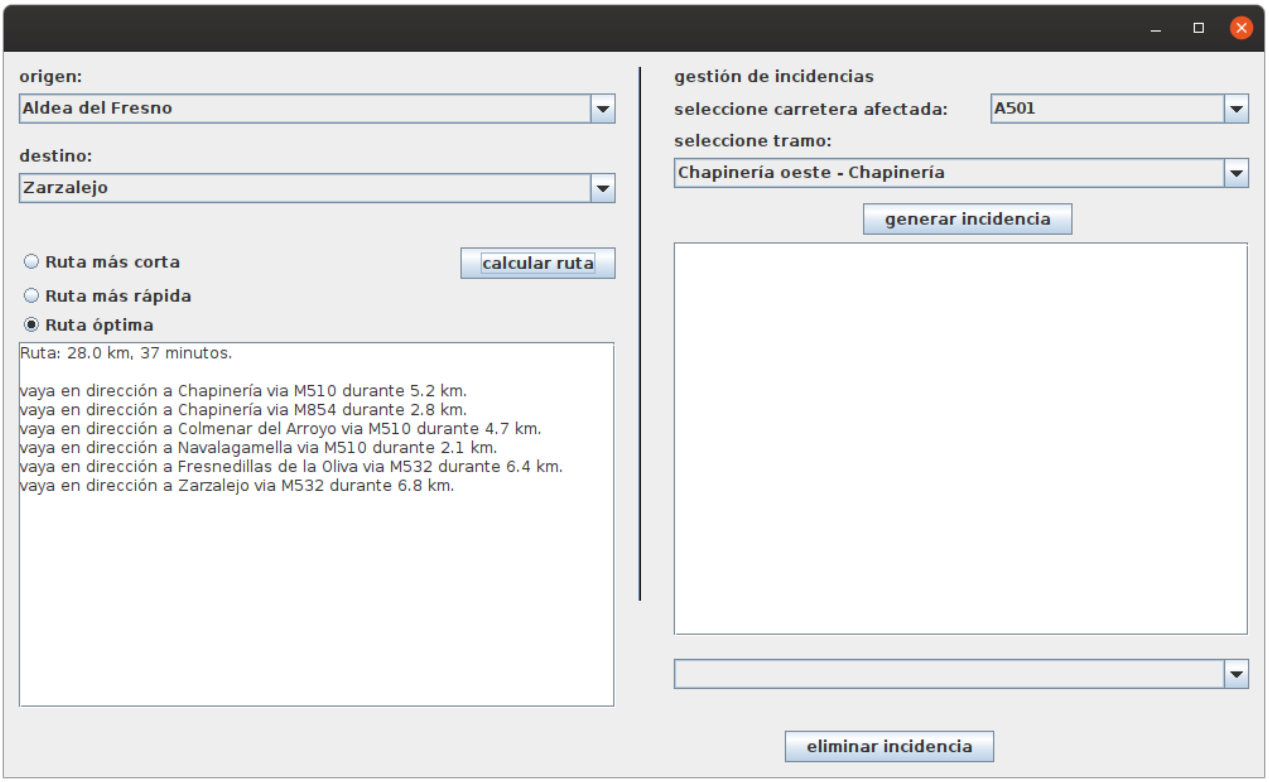


Figura 5.25: ejecución normal del cálculo de itinerario

Así pues, este proceso de inserción de incidencias debe ser algo “manual”. Se tiene que ofrecer al operario un listado de tramos donde pueda haber aparecido la incidencia y la posibilidad de “cortar” dicho tramo al tráfico. Mientras dure la incidencia las consultas a la base de datos deberán tener en cuenta este corte; y ademas, se debe mantener un control sobre las incidencias para poder eliminarlas y reabrir los tramos al tráfico cuando se considere que la incidencia ha sido solucionada.

Por el momento, sólo se plantea el tipo de incidencia que corta totalmente el tráfico de una vía en ambos sentidos. En un futuro se puede implementar la posibilidad de que la incidencia simplemente afecte a los tiempos de recorrido, o sólo a uno de los dos sentidos.

De entrada, se amplía el panel de la aplicación añadiendo un segundo campo a la derecha (Figura 5.25): En este campo aparecen un menú desplegable con todas las carreteras de la comarca. Según se elige una, en un segundo menú desplegable aparecen todos los tramos de dicha carretera.

Al clicar el botón “generar incidencia” suceden varias cosas:

- Se genera una entrada en una lista de incidencias (un ArrayList llamado *restriction* ubicado en *calculatePath*). Todos los cálculos de itinerarios se tienen que realizar tomando en cuenta esta lista.
- Se ejecuta una versión reducida del algoritmo de recorrido del grafo en la que simplemente se itera sobre el siguiente salto para los dos extremos del tramo afectado. Es decir, si se corta el tramo entre A y B, se busca la manera para ir de A a B y de B a A a través de itinerarios alternativos.
- En el panel de texto se muestra el texto que debe aparecer en los teleindicadores de los tramos anteriores a los dos puntos donde comienza y termina el nodo afectado para advertir a los usuarios de la incidencia. El mensaje, que debe ser breve, puesto que debe ser leído y asimilado por un conductor en apenas unos segundos, contendrá el tramo afectado por la incidencia y un itinerario alternativo (sólo la carretera que debe coger) para llegar al otro punto. En este caso, puesto que se debe ser lo más concreto posible, sí que se tendrán en cuenta las encrucijadas.
- Se añade la incidencia generada al menú desplegable localizado en la parte inferior.

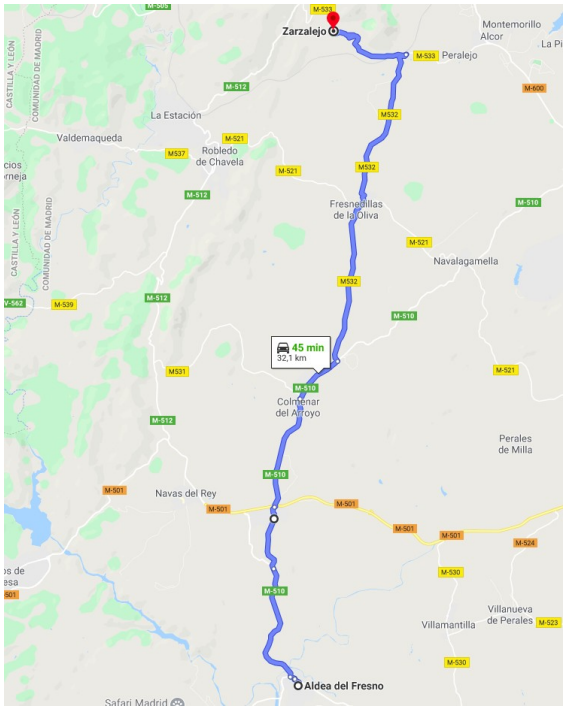


Figura 5.26: cálculo de itinerario sin incidencia (GSV)

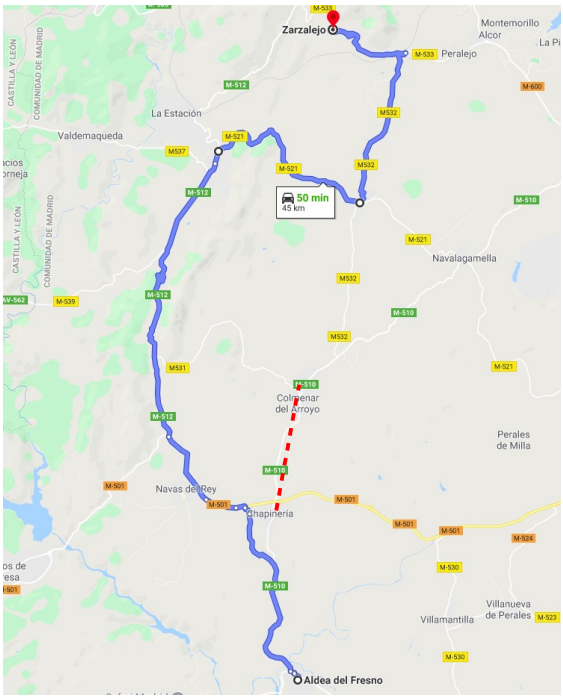


Figura 5.27: cálculo de itinerario con incidencia (GSV)

El menú inferior contiene un listado de las incidencias activas. Al elegir una y pulsar el botón “eliminar incidencia”, sucede lo siguiente:

- Se elimina la incidencia del arrayList “restriction”
- Se muestra en el panel de texto el mensaje que debe aparecer en los teleindicadores indicando que la incidencia ha sido resuelta.
- Se elimina la incidencia del desplegable.

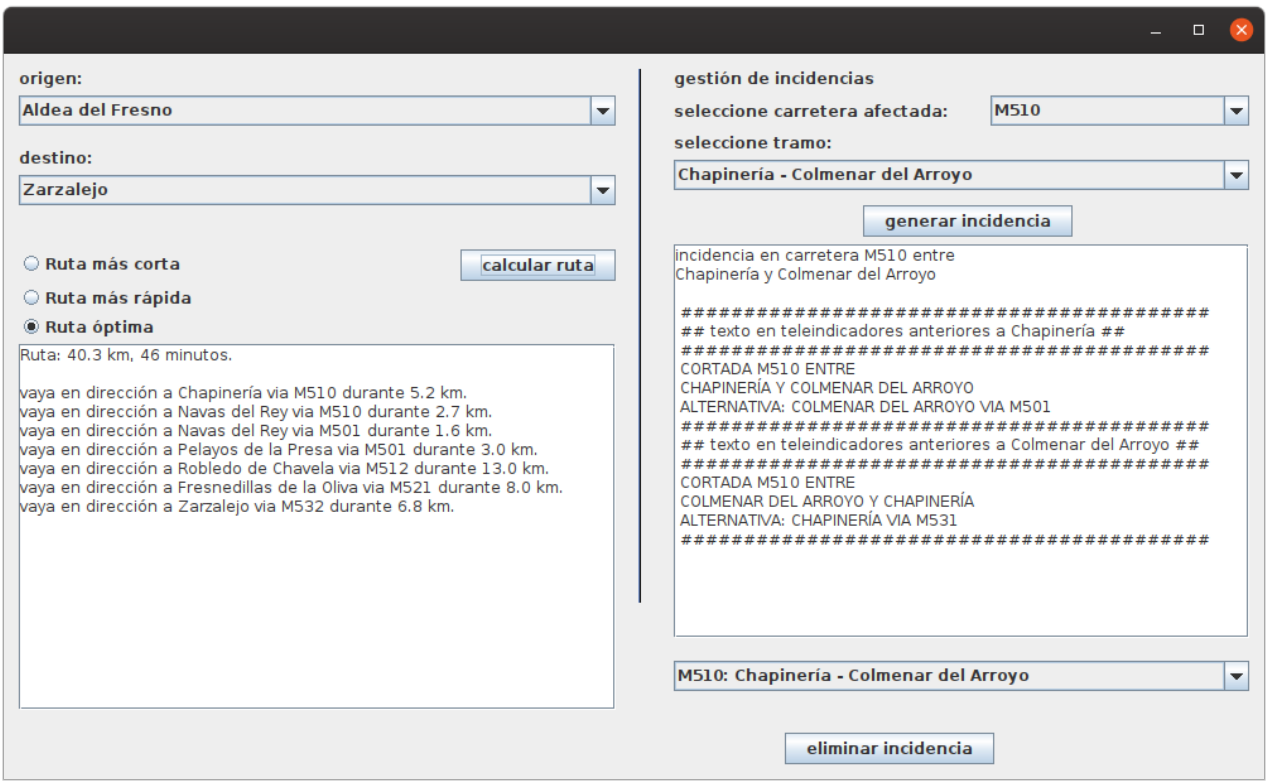


Figura 5.28: ejecución del cálculo de itinerario teniendo en cuenta una incidencia

Lógicamente, mientras una incidencia esté activa, cualquier cálculo de itinerario tiene que tener en cuenta dicha incidencia, con lo que antes de ejecutar la convergencia de nodos para realizar un cálculo hay que consultar la lista de restricciones y eliminar los tramos de los nodos directamente afectados.

En la Figura 5.25 se ve cómo sería, sin incidencias, viajar de Aldea del Fresno a Zarzalejo usando un criterio óptimo (el trazado se ve en la Figura 5.26). Al introducir una incidencia en uno de las

tramos del itinerario se ve que la traza varía (Figura 5.27), el nuevo itinerario queda reflejado en la Figura 5.28.

También se ha tenido que tener en cuenta, a la hora de presentar los mensajes en las áreas de texto, que en el caso de que una incidencia (o conjunto de ellas) pueda dejar inaccesible uno de los nodos, hay que comunicar que es imposible calcular el itinerario. En el caso de Valdemaqueda (Figura 5.29), basta con generar una incidencia en la carretera M-537, ya que es la única que la une al resto de la comarca. En la Figura 5.30 se ve cómo quedaría un intento de cálculo de itinerario entre Valdemaqueda y Robledo de Chavela habiendo una incidencia en dicha carretera.

Figura 5.29: conexión única de Valdemaqueda con la red de carreteras (GSV)

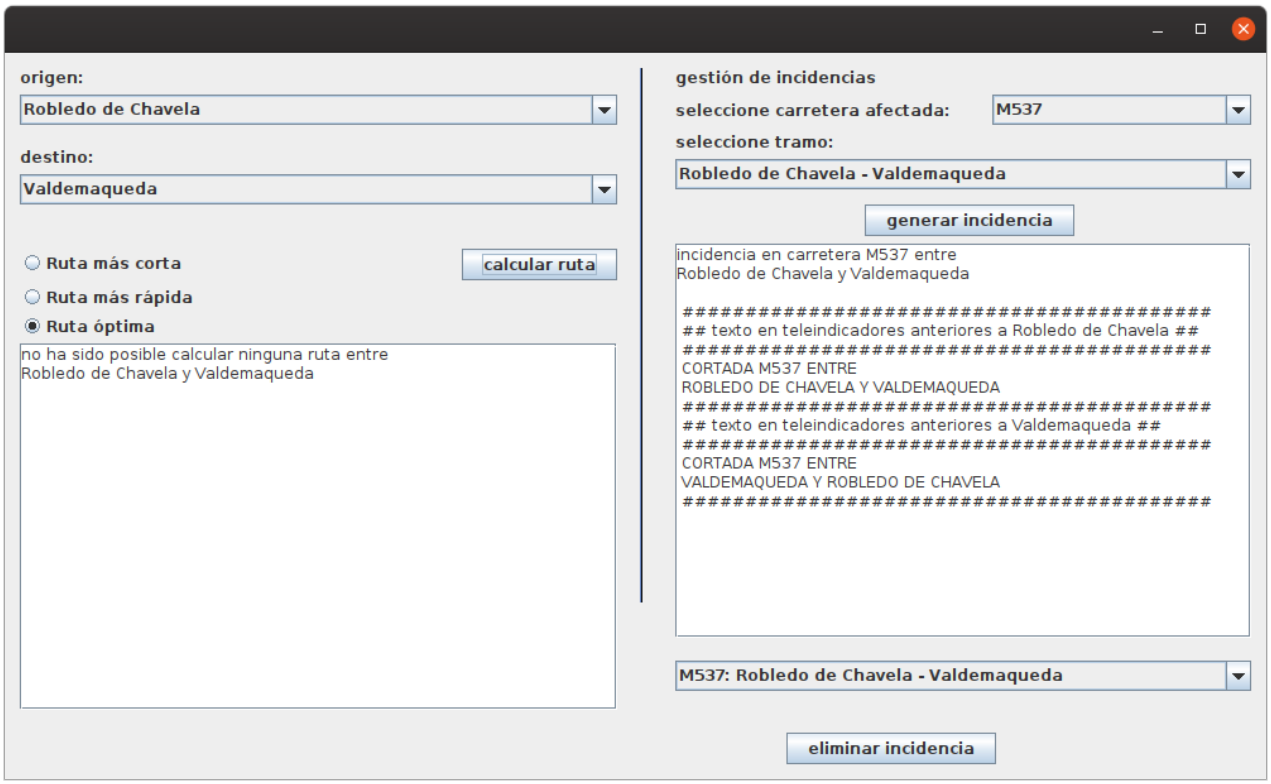


Figura 5.30: ejemplo de incidencia que aísla un nodo

5.9 Empaquetado

Crear un .jar con el software es tarea sencilla. El problema reside en la base de datos. El objetivo era hacer una aplicación lo más portable posible, con lo que se pretende evitar exigir al operador que se instale un servidor de bases de datos en su máquina. En su lugar, se opta por la versión embebida de DerbyDB de Apache, de modo que sea el propio programa el que levante el servidor, cree la estructura de la base de datos, y la pueble con los datos.

Así pues, se acompaña al .jar de un directorio *lib* con la versión embebida de DerbyDB embedded, y también se acompaña de los CSV correspondientes a las tablas de la base de datos. Estos son necesarios en la primera ejecución del programa, ya que creará las tablas sólo si éstas no existen. Si se quisiese usar un conjunto de datos distinto al ofrecido (por ejemplo, una ampliación de la cartografía existente, o una topología totalmente distinta), bastaría con eliminar el directorio de la base de datos y sustituir los CSV con los conjuntos de datos correspondientes. Esto hace que las ampliaciones sean realmente sencillas.

Para llevar esto a cabo, se sustituyeron los drivers de Glassfish por los de DerbyDB en el classpath, y fue necesaria la remodelación de la clase *DBconnect*, ubicada en el paquete *db_management*. El

cambio consiste en cambiar la cadena de conexión (Código 5.7), e incluir los comandos para crear las tablas y poblarlas a partir de los CSV (que habíamos hecho aparte cuando usábamos la versión en red)

```
String databaseURL = "jdbc:derby:routetracker;create=true";
```

Código 5.7: cadena de conexión a la base de datos embebida

5.10 Problemas encontrados

5.10.1 Versión de Java

Al principio del proyecto, cuando se comenzó a realizar el desarrollo, no había manera de inicializar la base de datos JavaDB desde NetBeans. Devolvía el error que se muestra en la Figura 5.31:

Se intentó dilucidar si el error era propio de la instalación, o si era algo inherente al propio NetBeans. Hay que destacar que en ese mismo equipo se realizó una práctica también en NetBeans con JavaDB con éxito en el pasado.

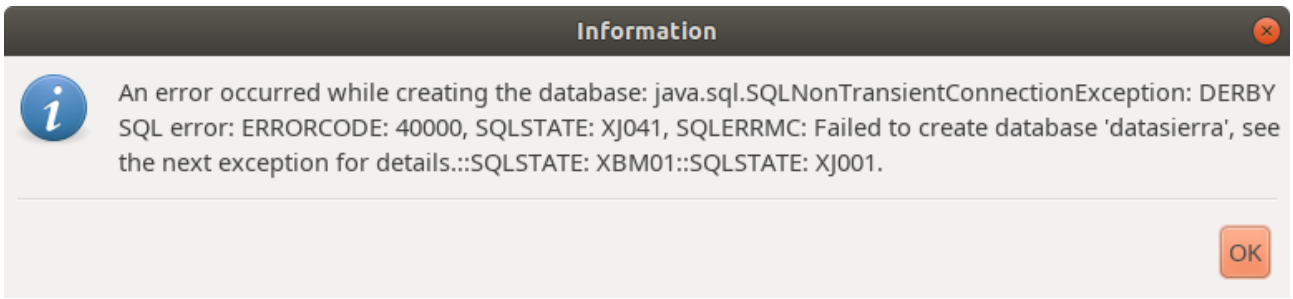


Figura 5.31: error de inicialización de JavaDB

Al realizar instalaciones limpias de NetBeans en sus versiones 11.2 y 11.3 en otros equipos observamos que el problema persiste. Se intenta buscar una solución buscando en foros en internet, sin éxito.

Finalmente, tras varias semanas buscando la solución al error, se encontró la solución en el siguiente enlace:

https://www.mail-archive.com/derby-user@db.apache.org/msg16348.html

El problema reside en la versión de Java. Concretamente, es un problema de permisos relativo a una implementación de seguridad de Java 11. Cuando se usó por última vez el equipo tenía instalada la versión 8 de Java. En el momento del error, el equipo corre la versión 11.0.6 de OpenJDK.

Concretamente, la solución consiste en añadir la línea contenida en el Código 5.8 al archivo */etc/java-11-openjdk/security/java.policy*.

```
permission java.security.AllPermission;
```

Código 5.8 solución de problema de permisos en Java.

Tras aplicar esta solución el servidor de base de datos arranca perfectamente.

5.10.2 Limpieza de código

Al final del proceso de limpieza y comentado de código, al ejecutar el programa y realizar todas las pruebas de la batería de pruebas, se detectó un comportamiento anormal en el sentido de al trazar una ruta no variaban las métricas de distancia y tiempo al cambiar el criterio. Además, se observó que, tras añadir incidencias para intentar incomunicar origen y destino, al eliminar una de las incidencias, el sistema seguía siendo incapaz de encontrar una ruta.

Tras comparar con la versión anterior al proceso guardada en Bitbucket se dio con el problema (un comando eliminado erróneamente).

Tras corregir el error se realizan de nuevo todas las pruebas de manera satisfactoria.

5.10.3 Codificación regional

Se detecta que, aunque en Linux el problema no reporta problemas de codificación regional, al ejecutarlo en Windows los acentos aparecen sustituidos por caracteres extraños. El problema se encuentra localizado en varios sitios.

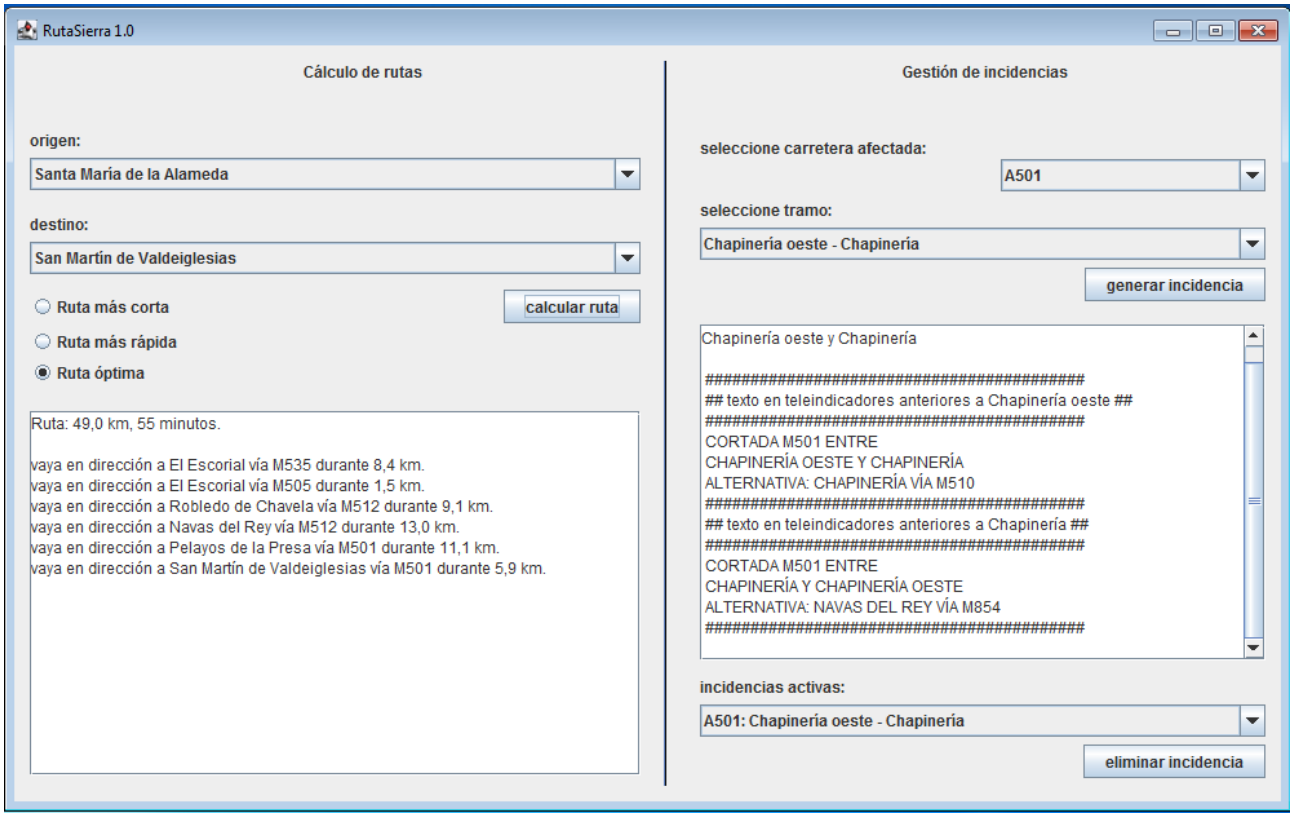


Figura 5.32: aplicación corriendo en Windows

Primero, se cambia la codificación por defecto (UTF8) a la soportada por Windows en Europa Occidental (ISO-8859-1) en las propiedades del proyecto en NetBeans. Esto elimina los problemas de la GUI. Sin embargo, siguen apareciendo los mencionados caracteres extraños. Concretamente, en los nombres de los nodos, lo cual hace pensar que el problema puede estar relacionado con la base de datos.

Tras probar infructuosamente a cambiar la codificación de DerbyDB, se prueba a revisar la codificación de los CSV desde los que se carga la base de datos.

Al cambiar la codificación de estos de UTF8 a ISO-8859-1 aparecen en el propio CSV los mencionados caracteres. Se sustituyen manualmente y se guarda el CSV con la nueva codificación. Tras esto, al probar de nuevo en una máquina Windows el problema ha desaparecido, como se comprueba en la Figura 5.32. Al volver a ejecutarlo en Linux los caracteres con tilde siguen apareciendo correctamente.

5.11 Modificaciones sufridas

Respecto al concepto inicial el proyecto sufre numerosas modificaciones a lo largo de su desarrollo:

5.11.1 Gestión de incidencias

Ésta era una implementación que dependía del tiempo del que se dispusiera para poder realizarla. El proyecto base comprendía el apartado de selección de mejor ruta. Se deseaba introducir algún tipo de generación aleatoria de incidencias para poder observar al algoritmo reaccionando a los cambios de topología.

Sin embargo, se pudo introducir como añadido un apartado donde generar y administrar incidencias, además de poder añadir instrucciones para mensajes en la señalización luminosa, dotando al sistema de una cierta utilidad para un organismo oficial, como los centros de pantallas de Tráfico, o los centros de conservación de carreteras.

Se descartó ademas el hecho de generar aleatoriamente las incidencias ya que, aunque éstas sí que aparecen aleatoriamente (no se puede predecir un accidente), el hecho de gestionarlas no es aleatorio, sino que requiere un tipo de interacción por parte de los operadores, que es lo que se ha pretendido emular.

5.11.2 Tipología de la base de datos

Como ya se ha mencionado anteriormente, la base de datos inicialmente estaba pensada para ser un servidor corriendo, o bien en el mismo equipo donde correría la aplicación, o bien en otro distinto y acceder a través de la red.

Finalmente, a la hora de realizar el empaquetado de la aplicación se opta por la versión más compacta (base de datos embebida), lo cual simplifica notablemente la portabilidad de la aplicación.

5.11.3 Categorización de rutas

Inicialmente los criterios de selección de rutas iban a ser, o bien por tiempo, o bien por distancia. Se decide un tercer criterio basado en distancia y categoría de la carretera. Esto motiva que haya que crear una segunda tabla en la base de datos con la categorización asignada a cada ruta.

5.11.4 Recálculo del algoritmo

La idea original era que los nodos consultasen a la base de datos sólo a la inicialización del programa, y en caso de que hubiese cambios de topología, modificasen la tabla existente en función de los cambios.

Sin embargo, el hecho de uso de criterios para elegir la ruta forzó que hubiese un recálculo completo de la topología cada vez que se invocase el algoritmo. ¿Por qué? Porque había que elegir entre mantener dos (o al final tres, con el criterio óptimo) tablas de rutas simultáneas por cada nodo, con lo que, en memoria supone triplicar la demanda de recursos, o lo que se decidió finalmente, a pesar del coste computacional.

También, al probar con una sola tabla, la modificación continua de la tabla de enrutamiento inicial provocaba ciertas inconsistencias cuando se eliminaban las incidencias. Puesto que incluso en OSPF cada cierto tiempo el algoritmo se “refresca” para evitar esas inconsistencias, y el hecho de que en este sistema los nodos se pueden permitir el “lujo” de perder completamente su tabla de rutas momentáneamente, cosa que no no puede ocurrir en una red de enrutadores, se opta por, ante cada petición de trazado de rutas, borrar completamente la tabla de rutas de cada nodo y ejecutar el algoritmo de nuevo teniendo en cuenta el criterio y las incidencias.

5.12 Pruebas

La batería de pruebas que se realizan al final del desarrollo queda como sigue:

- P1: Intentar trazar una ruta con la misma población como origen y destino. No debe realizar ninguna acción.

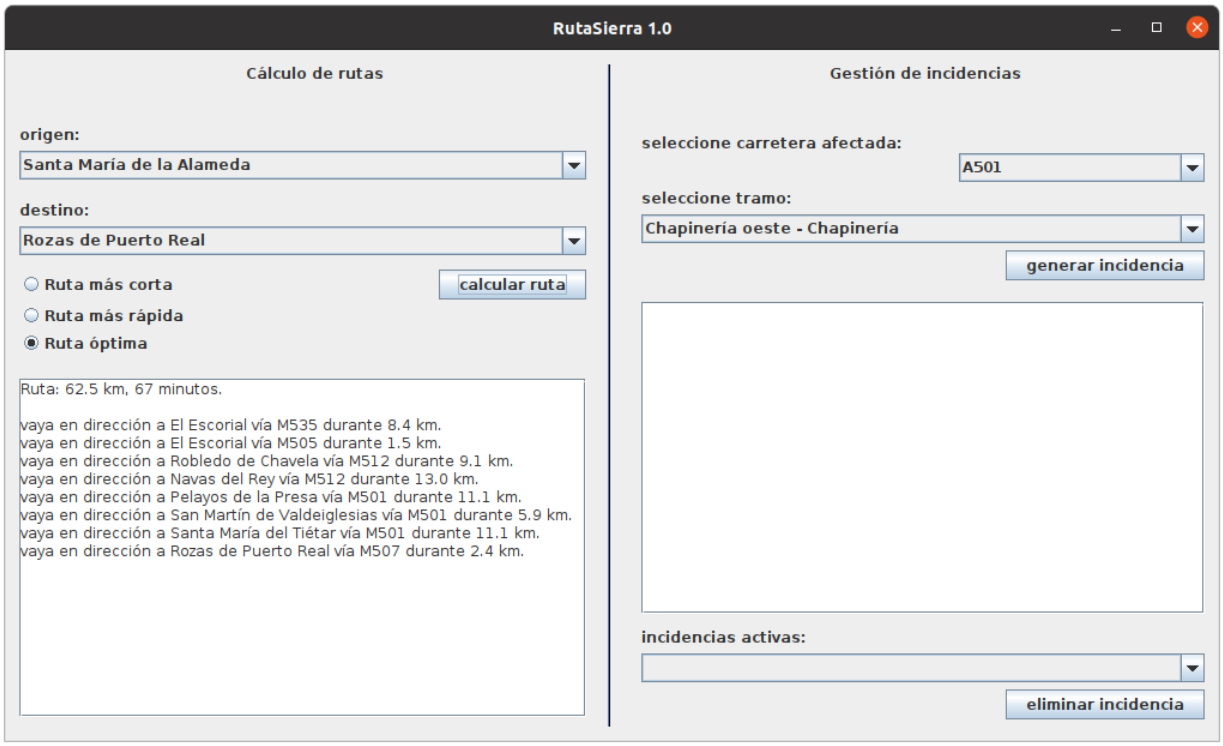


Figura 5.33: traza entre destinos en extremos de la región

- P2: Traza de rutas en destinos lo más alejados posible este-oeste, oeste-este, norte-sur, sur-norte (Figura 5.33).

- P3: Cambiar de criterio. Comprobar que la ruta elegida varía en función del criterio elegido (Figura 5.34).

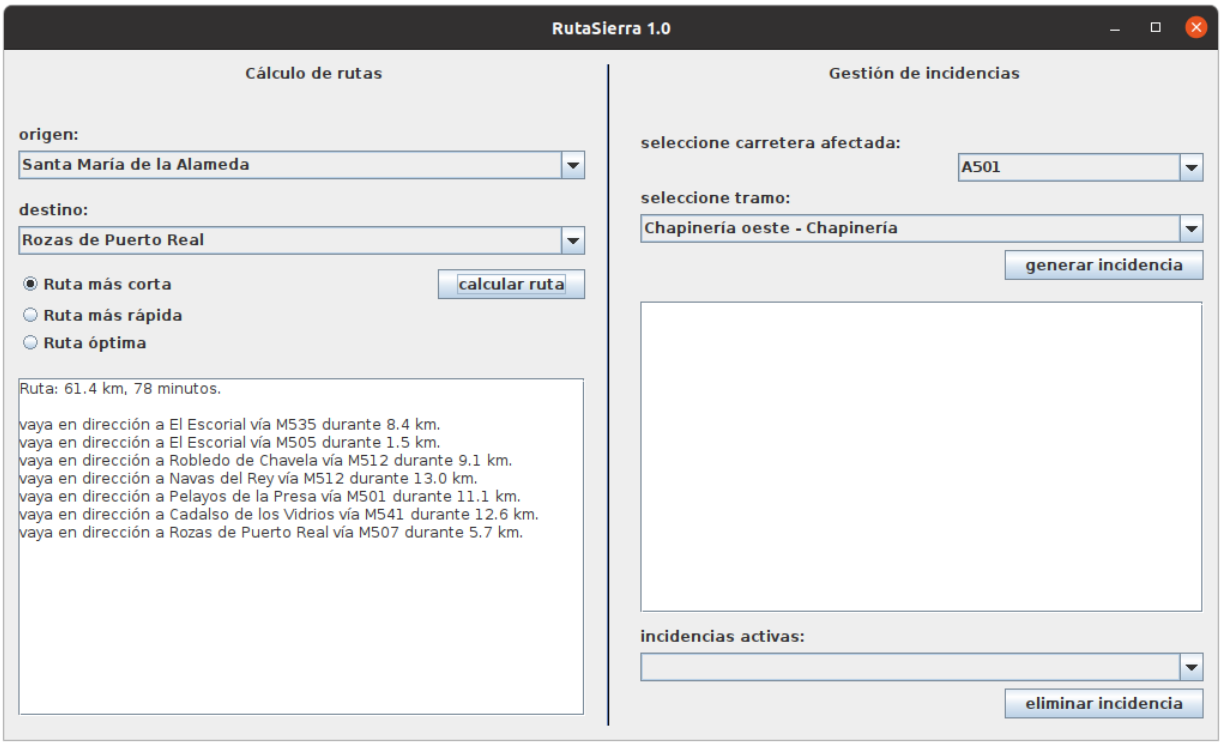


Figura 5.34: traza cambiando de criterio

- P4: Comprobar en un mapa que las rutas elegidas sean coherentes (Figura 5.35).

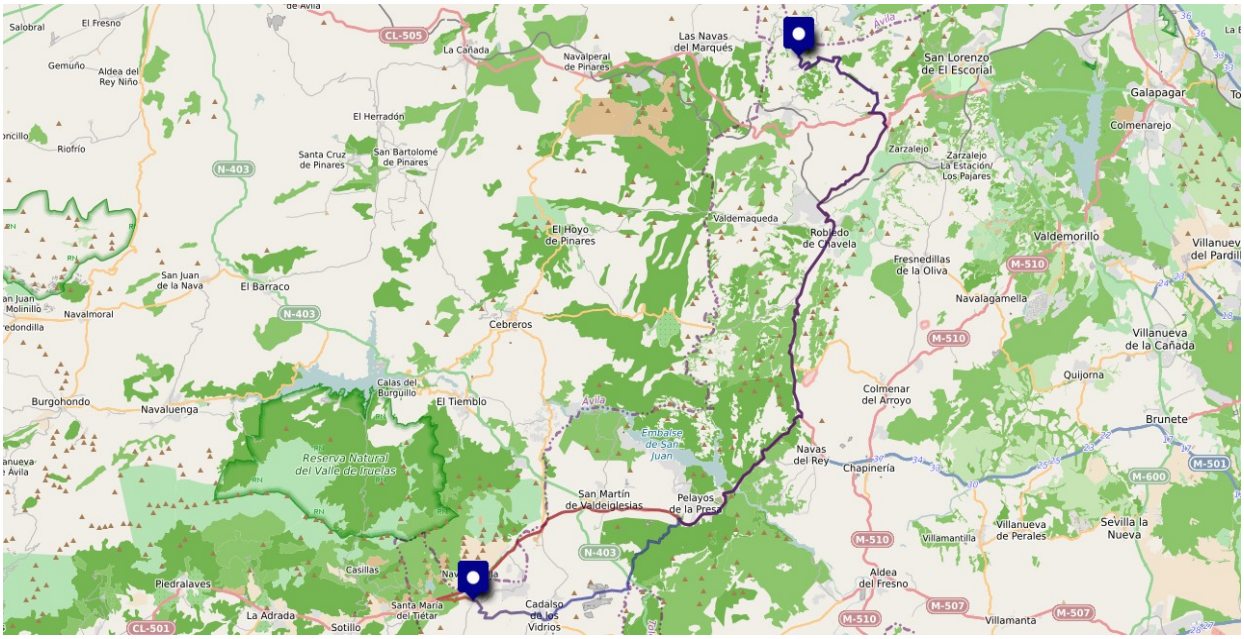


Figura 5.35: comparación de las dos rutas en un mapa (OSM)

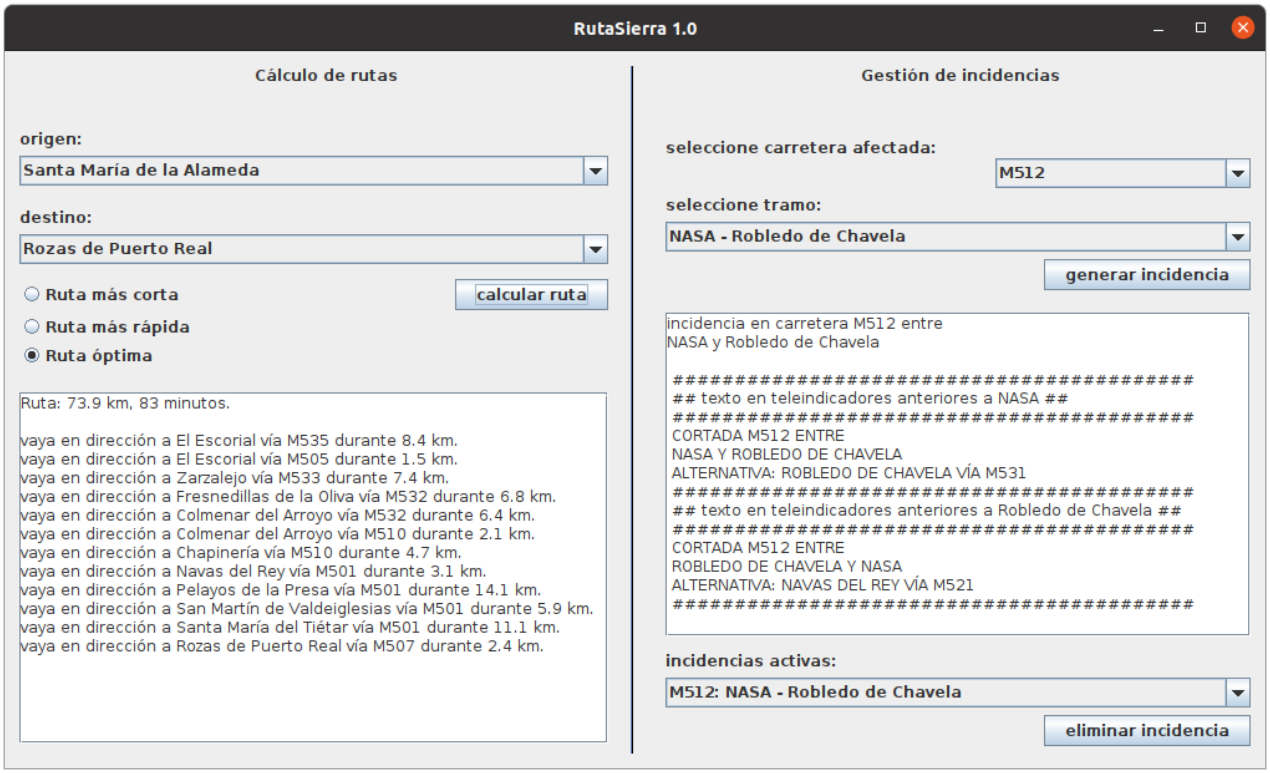


Figura 5.36: traza de ruta afectada por incidencia

- P5: Trazada una ruta, probar a generar una incidencia que afecte a un tramo por los que pasa la ruta. Comprobar que se recalcula usando otro camino (Figura 5.36).

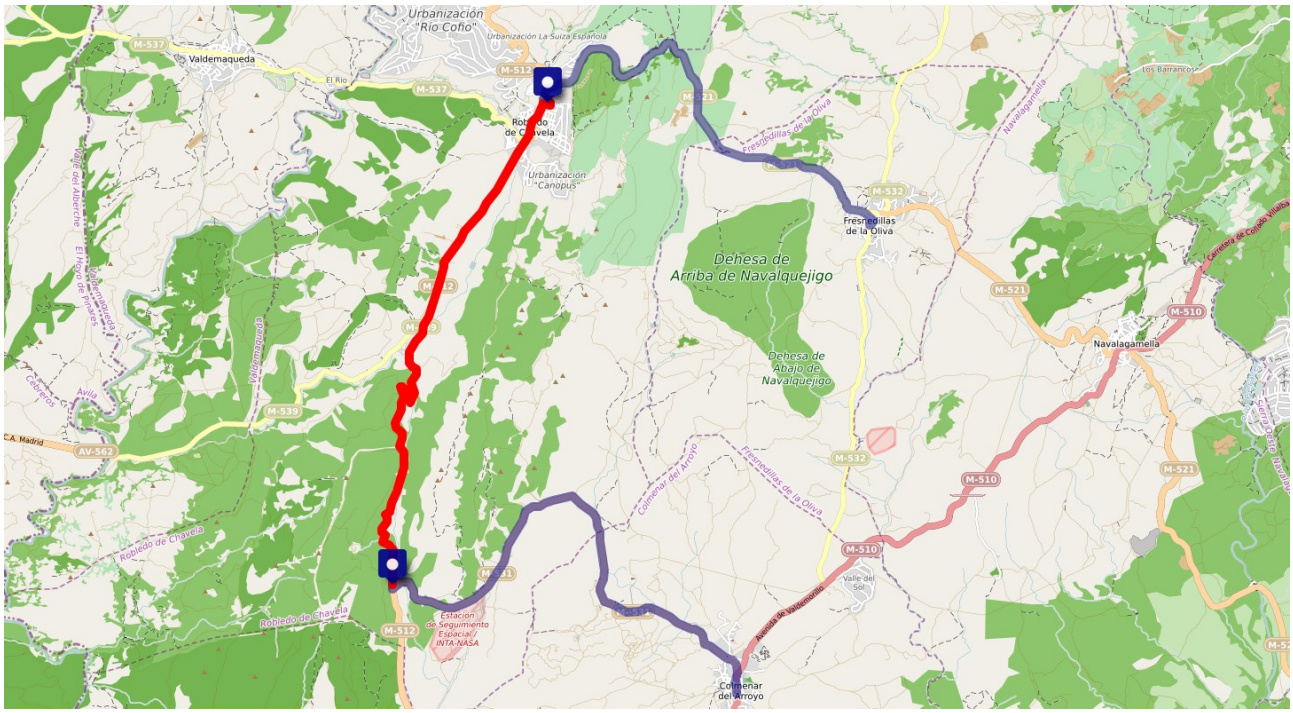


Figura 5.37: alternativas propuestas en caso de incidencia (OSM)

- P6: Generada una incidencia (P5), comprobar con un mapa que las alternativas propuestas en el texto para los teleindicadores sean coherentes (Figura 5.37).

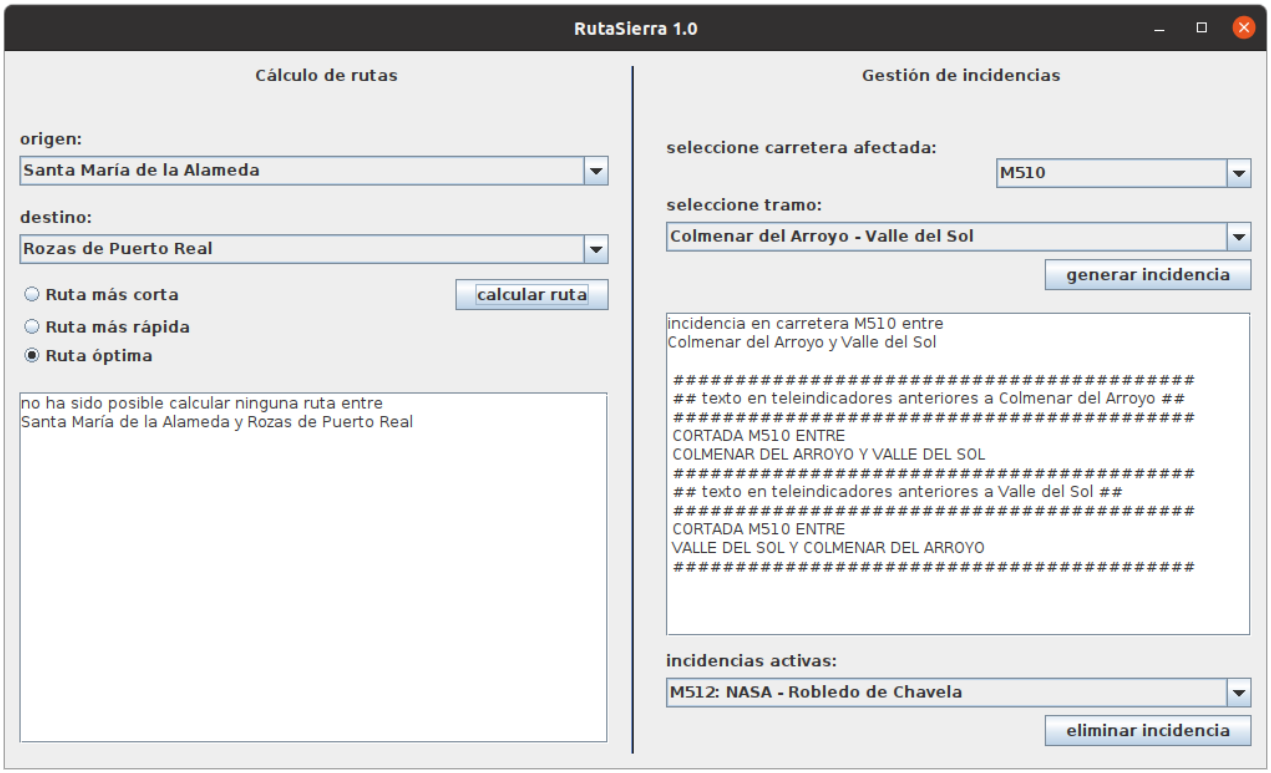


Figura 5.38: puntos incomunicados por acumulación de incidencias

- P7: Intentar generar tantas incidencias como sean necesarias como para incomunicar los dos puntos de la traza realizada en P5. Comprobar que efectivamente se falla en intentar encontrar una ruta entre ambos puntos (Figura 5.38).
- P8: Comprobar que todas las incidencias introducidas quedan almacenadas en la lista de incidencias.
- P9: Intentar introducir una incidencia repetida y observar que no se produce ningún cambio.
- P10: Tras incomunicar dos puntos eliminar una incidencia. Intentar generar otra traza (Figura 5.39). Se tiene que restablecer la comunicación entre ambos.

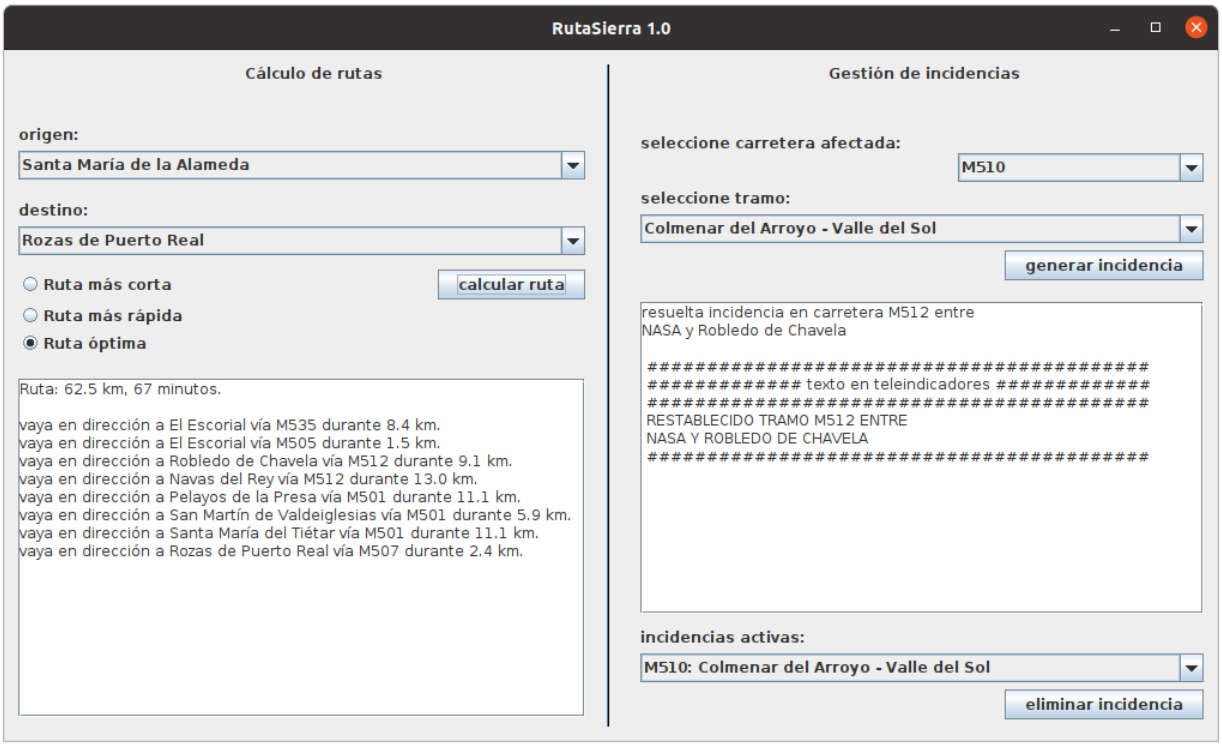


Figura 5.39: traza tras restablecimiento de incidencia

- P11: Intentar generar todas las incidencias posibles y observar cómo son almacenadas en la lista de incidencias.
- P12: Eliminar una incidencia e intentar realizar una traza entre sus dos puntos afectados. La traza debería ser exitosa.
- P13: Eliminar todas las incidencias, y realizar trazados durante el proceso. Observar que se realizan cambios en la traza de rutas.
- P14: Probar la aplicación en distintos entornos y sistemas operativos. No debería alterar el funcionamiento.

Finalizada la batería de pruebas, con ayuda de la aplicación VisualVM 2.0.2 se realiza un análisis de rendimiento de la aplicación para determinar cuántos recursos consume (ver Figura 5.40)

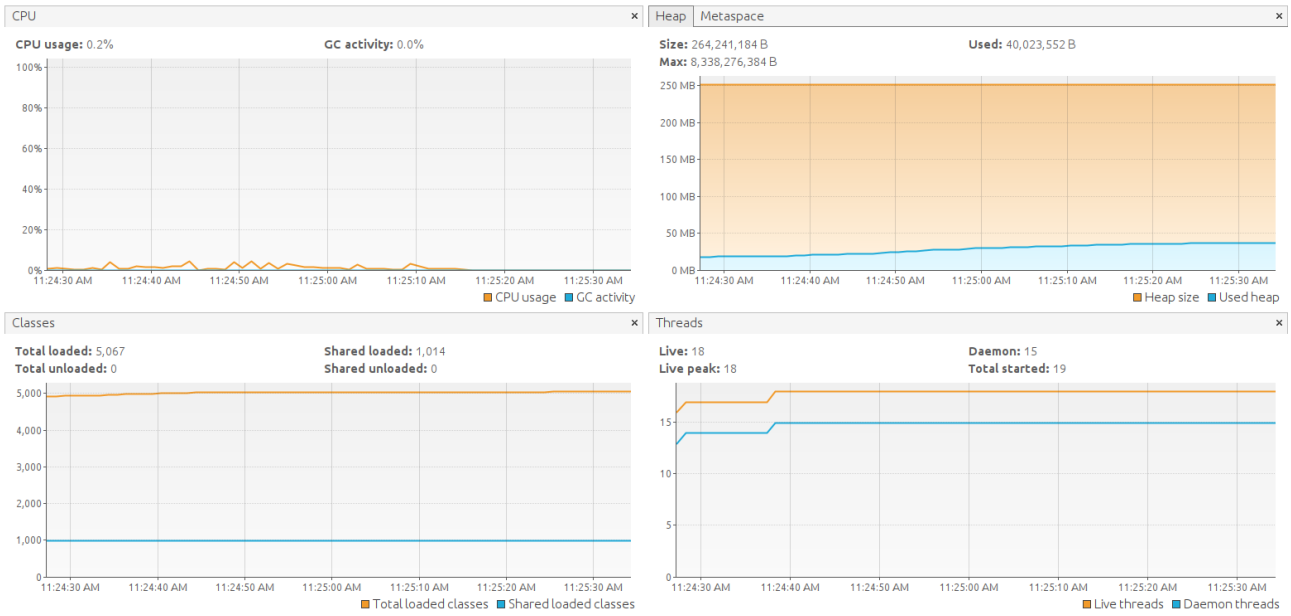


Figura 5.40: análisis de rendimiento

Se observa que en lo que respecta a CPU no llega al 5% en ninguno de los casos, y la memoria, aunque reserva un montículo de 250Mb, no llega a usar ni 50 Mb. Se puede decir que esta aplicación se puede ejecutar en sistemas con muy pocos recursos.

5.13 Conclusiones

En este capítulo se ha descrito el proceso de desarrollo del proyecto. Ha quedado plasmada la evolución del citado proyecto, mediante los errores, problemas, rectificaciones, y desarrollo del mismo.

Dado que el planteamiento del algoritmo de recorrido del grafo ha sido el que ha supuesto un mayor reto se se le ha dedicado en este capítulo una descripción bastante profunda, puesto que se ha tenido que desarrollar un autómata de pila que tuviese en cuenta el estado anterior y posterior del estado actual. Dentro del proceso de codificación, éste ha sido, sin duda, el que más tiempo ha consumido.

Se han descrito también las pruebas a realizar, las cuales se efectuaron a lo largo del desarrollo, y durante una etapa concreta del desarrollo. Etapas que se mostrarán en el siguiente capítulo.

Capítulo 6: Planificación y costes del proyecto

6.1 Introducción

En este capítulo se muestra el proceso de creación del proyecto paso a paso, con sus etapas, sus costes, y documentación del producto final.

En la planificación se especifican las etapas (o *sprints*) planificadas, en las que se establecen los hitos a alcanzar antes de pasar a la siguiente etapa. Al final de cada *sprint* es cuando se establece cuál es el siguiente objetivo a alcanzar.

6.2 Planificación

Como ya se ha dicho anteriormente, aunque hay unas líneas base y unas ideas sobre las que empezar a desarrollar, el concepto de tiempo y etapas al principio del proyecto quedan un poco ambiguos, teniendo en cuenta que, desde que se inició el proyecto en septiembre de 2019, cuando se presenta el anteproyecto, hasta que se finaliza, en julio de 2020, hay un espacio de tiempo inmenso en el cual se ha tenido que replanificar en función del avance del proyecto.

Ha habido etapas que han durado más o menos de lo previsto. También, a lo largo del desarrollo se han variado las prioridades y el orden de las etapas, en base a lo que se iba percibiendo como prioritario. De ahí que el diagrama de previsión y el diagrama final hayan diferido tanto.

6.2.1 Etapas

Sprint 1: Decisión de temario para la memoria (11 octubre)

Previo al inicio del desarrollo de la aplicación se intercambian mensajes con el director del proyecto sobre el temario de esta memoria. Aunque a priori puede que parezca que no sea importante en el desarrollo de la aplicación, sí que lo es a la hora de enfocarla a lo que debe importar y a la manera de desarrollarla. Asimismo, el hecho de crear el índice antes de comenzar con el despliegue ha dado una visión distinta sobre algunos aspectos relativos a la estructuración del trabajo en el proyecto.

Sprint 2: Lenguaje empleado y entorno de desarrollo (27 octubre)

Se prepara el IDE y se crea un prototipo de lo que será el producto software (Figura 6.1).

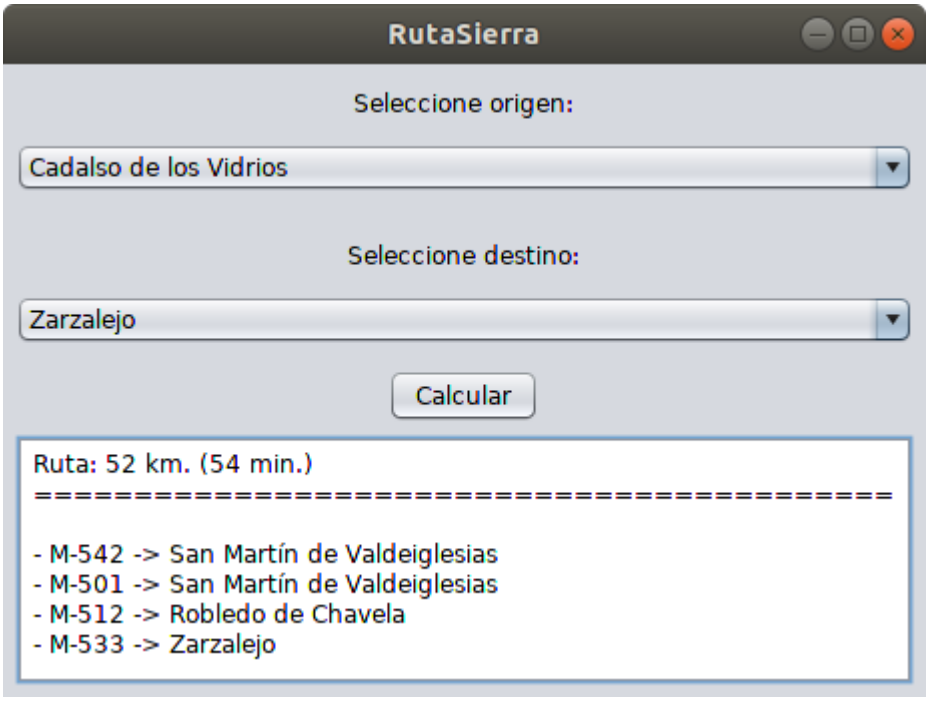


Figura 6.1: vista previa de primer entorno gráfico creado

Sprint 3: Cartografiado (9 noviembre)

Se realiza el cartografiado y con ello se produce el primer entregable del proyecto (ver Anexo A): La Tabla de Rutas.

De cara a que sirva de guía se elabora un plano-grafo del área cartografiada (Figura 6.2).

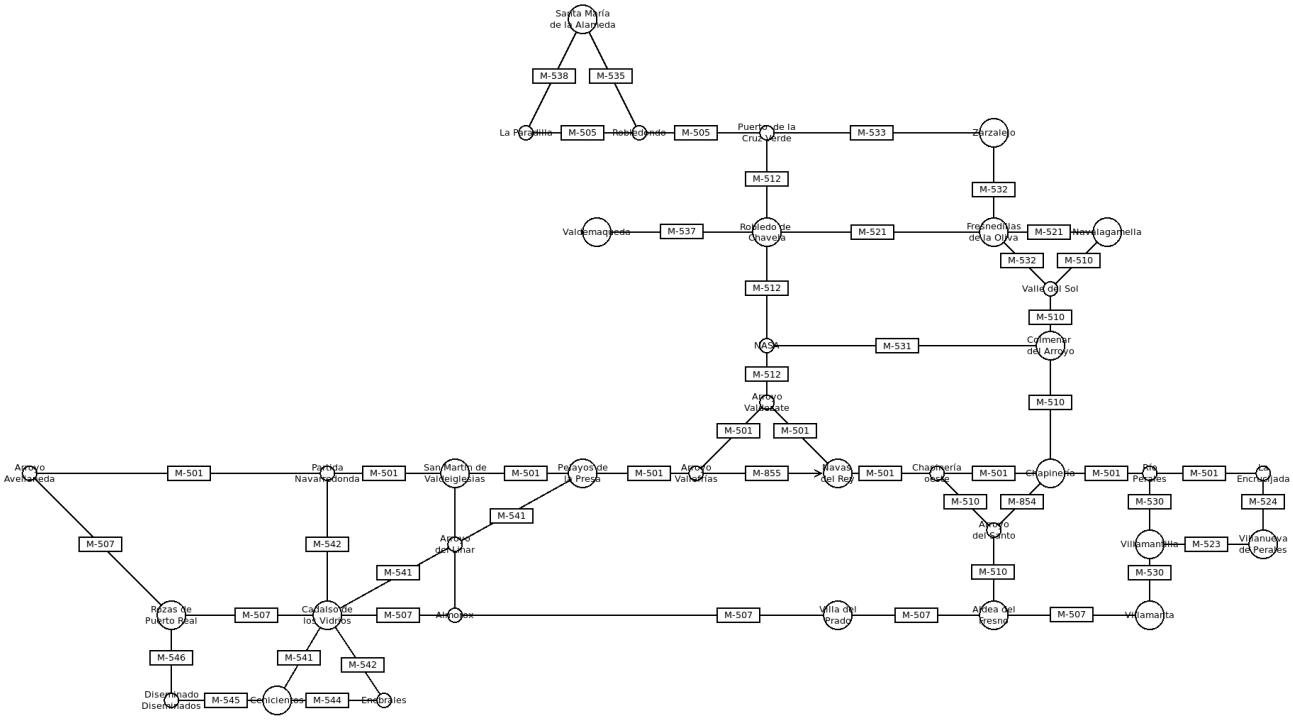
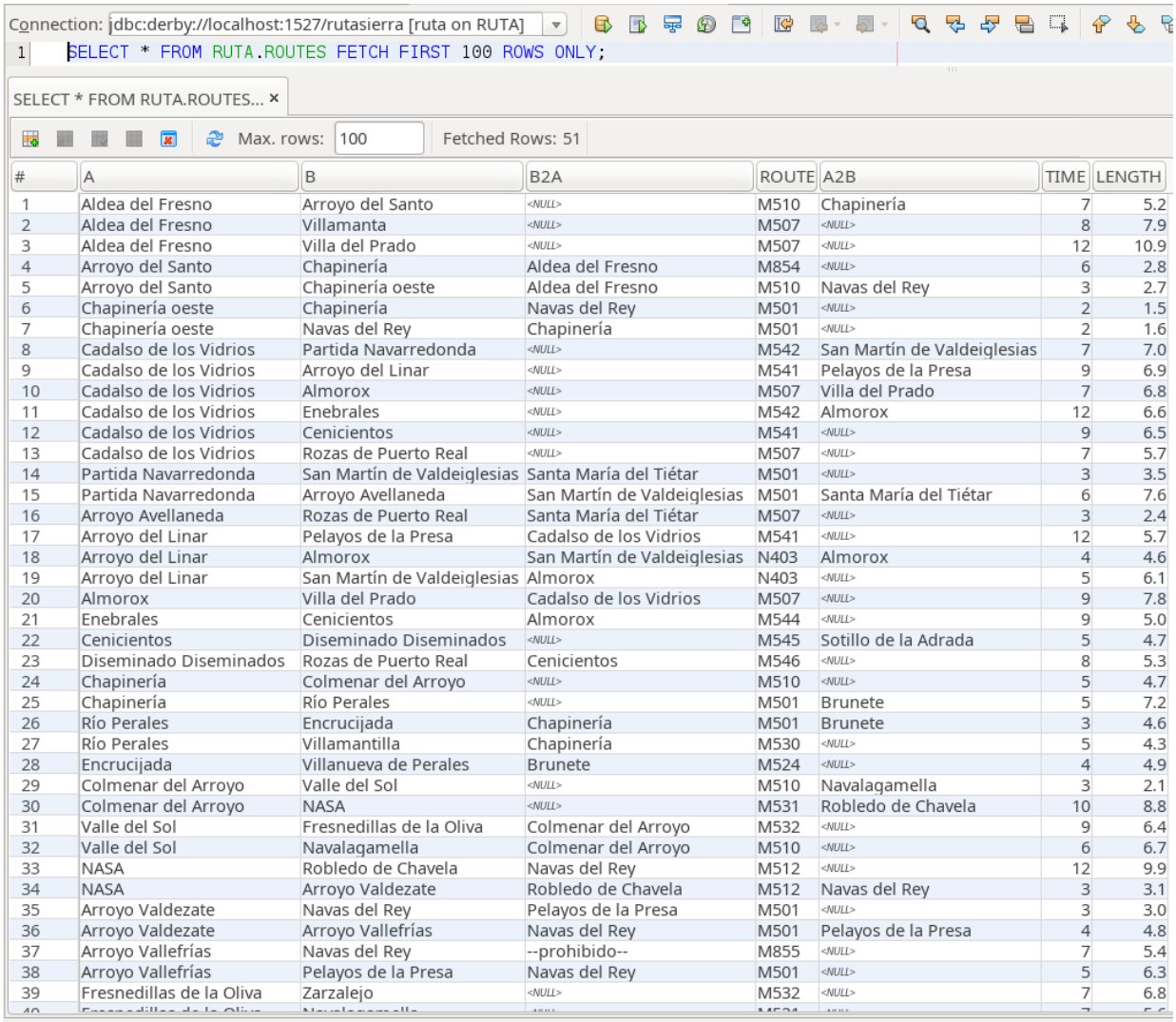


Figura 6.2: plano-grafo

Sprint 4: Creación de la base de datos (29 febrero)

Se crea la base de datos y se carga con los datos de cartografía. La Figura 6.3 muestra la base de datos una vez poblada.



Connection: jdbc:derby://localhost:1527/rutasierra [ruta on RUTA]

```
1  SELECT * FROM RUTA.ROUTES FETCH FIRST 100 ROWS ONLY;
```

SELECT * FROM RUTA.ROUTES... ×

Max. rows: 100 Fetched Rows: 51

#	A	B	B2A	ROUTE	A2B	TIME	LENGTH
1	Aldea del Fresno	Arroyo del Santo	<NULL>	M510	Chapinería	7	5.2
2	Aldea del Fresno	Villamanta	<NULL>	M507	<NULL>	8	7.9
3	Aldea del Fresno	Villa del Prado	<NULL>	M507	<NULL>	12	10.9
4	Arroyo del Santo	Chapinería	Aldea del Fresno	M854	<NULL>	6	2.8
5	Arroyo del Santo	Chapinería oeste	Aldea del Fresno	M510	Navas del Rey	3	2.7
6	Chapinería oeste	Chapinería	Navas del Rey	M501	<NULL>	2	1.5
7	Chapinería oeste	Navas del Rey	Chapinería	M501	<NULL>	2	1.6
8	Cadalso de los Vidrios	Partida Navarredonda	<NULL>	M542	San Martín de Valdeiglesias	7	7.0
9	Cadalso de los Vidrios	Arroyo del Linar	<NULL>	M541	Pelayos de la Presa	9	6.9
10	Cadalso de los Vidrios	Almorox	<NULL>	M507	Villa del Prado	7	6.8
11	Cadalso de los Vidrios	Enebrales	<NULL>	M542	Almorox	12	6.6
12	Cadalso de los Vidrios	Cenicientos	<NULL>	M541	<NULL>	9	6.5
13	Cadalso de los Vidrios	Rozas de Puerto Real	<NULL>	M507	<NULL>	7	5.7
14	Partida Navarredonda	San Martín de Valdeiglesias	Santa María del Tiétar	M501	<NULL>	3	3.5
15	Partida Navarredonda	Arroyo Avellaneda	San Martín de Valdeiglesias	M501	Santa María del Tiétar	6	7.6
16	Arroyo Avellaneda	Rozas de Puerto Real	Santa María del Tiétar	M507	<NULL>	3	2.4
17	Arroyo del Linar	Pelayos de la Presa	Cadalso de los Vidrios	M541	<NULL>	12	5.7
18	Arroyo del Linar	Almorox	San Martín de Valdeiglesias	N403	Almorox	4	4.6
19	Arroyo del Linar	San Martín de Valdeiglesias	Almorox	N403	<NULL>	5	6.1
20	Almorox	Villa del Prado	Cadalso de los Vidrios	M507	<NULL>	9	7.8
21	Enebrales	Cenicientos	Almorox	M544	<NULL>	9	5.0
22	Cenicientos	Diseminado Diseminados	<NULL>	M545	Sotillo de la Adrada	5	4.7
23	Diseminado Diseminados	Rozas de Puerto Real	Cenicientos	M546	<NULL>	8	5.3
24	Chapinería	Colmenar del Arroyo	<NULL>	M510	<NULL>	5	4.7
25	Chapinería	Río Perales	<NULL>	M501	Brunete	5	7.2
26	Río Perales	Encrucijada	Chapinería	M501	Brunete	3	4.6
27	Río Perales	Villamantilla	Chapinería	M530	<NULL>	5	4.3
28	Encrucijada	Villanueva de Perales	Brunete	M524	<NULL>	4	4.9
29	Colmenar del Arroyo	Valle del Sol	<NULL>	M510	Navalagamella	3	2.1
30	Colmenar del Arroyo	NASA	<NULL>	M531	Robledo de Chavela	10	8.8
31	Valle del Sol	Fresnedillas de la Oliva	Colmenar del Arroyo	M532	<NULL>	9	6.4
32	Valle del Sol	Navalagamella	Colmenar del Arroyo	M510	<NULL>	6	6.7
33	NASA	Robledo de Chavela	Navas del Rey	M512	<NULL>	12	9.9
34	NASA	Arroyo Valdezate	Robledo de Chavela	M512	Navas del Rey	3	3.1
35	Arroyo Valdezate	Navas del Rey	Pelayos de la Presa	M501	<NULL>	3	3.0
36	Arroyo Valdezate	Arroyo Vallefrías	Navas del Rey	M501	Pelayos de la Presa	4	4.8
37	Arroyo Vallefrías	Navas del Rey	--prohibido--	M855	<NULL>	7	5.4
38	Arroyo Vallefrías	Pelayos de la Presa	Navas del Rey	M501	<NULL>	5	6.3
39	Fresnedillas de la Oliva	Zarzalejo	<NULL>	M532	<NULL>	7	6.8

Figura 6.3: vista de la base de datos de rutas

Sprint 5: Conexión a la base de datos (20 marzo)

Se crea un paquete con una clase para conectarse a la base de datos y otra para ejecutar las consultas y recoger los resultados. Se realiza la primera consulta de prueba.

La consulta devuelve un array de cadenas de texto con los contenidos de las filas sobre la columna indicada.

El resultado de lanzar una consulta nos muestra los nombres de los pueblos de las columnas A y B, como se ve en la Figura 6.4.

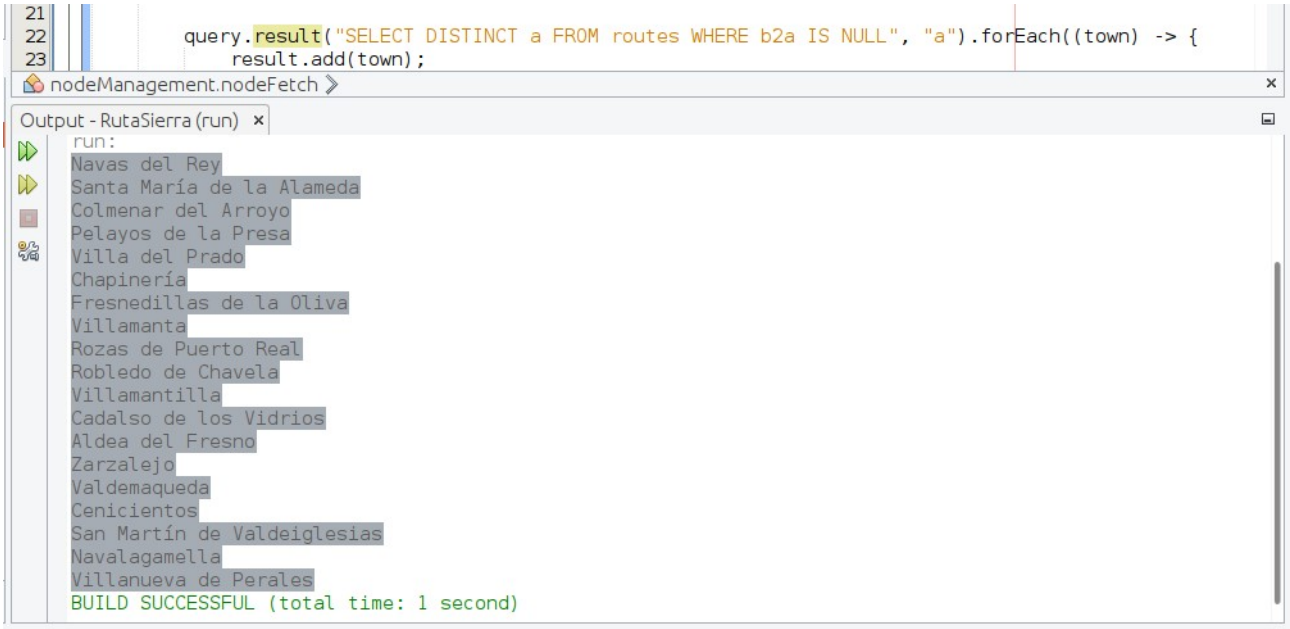


Figura 6.4: resultado de consulta usando código en Java

Sprint 6: Creación del algoritmo (24 marzo)

Se crea el paquete node_management, que contiene las clases:

- *NodeFetch*: El “orquestador” de los nodos. Consulta a la base de datos cuántos nodos hay para luego crearlos, y organiza la convergencia de la comunicación entre nodos.
- *Node*: Una clase abstracta que representa un nodo en el grafo. Realiza una conexión a la base de datos y pregunta por sus rutas directamente conectadas, con las que comienza a elaborar su tabla de rutas. A continuación “pregunta” a sus “vecinos” por sus rutas y en base a la información recibida completa su tabla de rutas.
- *Town*: Representa una población. Es una clase concreta que hereda de la clase *Node*.
- *Crossroad*: Representa una encrucijada. También hereda de la clase *Node*, pero con la diferencia de que ésta, al ser “preguntada” por sus “vecinos”, responde con el nombre de la población que se encuentra pasada ésta misma.
- *Edge*: Una clase abstracta que representa un arco en el grafo. Los almacenan los nodos como rutas individuales. Contiene datos de métricas para el desarrollo del algoritmo.
- *Route*: Representa un tramo de carretera. Hereda de la clase *Edge*.

La clase *Node* contiene en su método *buildShortestPath()* la aplicación del algoritmo para este caso concreto. Se comprueba que los nodos construyen sus tablas de rutas correctamente.

También se introduce el criterio de ruta óptima. Para el cálculo de la ruta óptima se tiene en cuenta la categoría de cada carretera y kilómetros recorridos por la misma. O sea, se toma la métrica de distancia y se multiplica por un factor de calidad asignado a cada tramo.

Sprint 7: Creación del entorno gráfico (29 marzo)

El entorno gráfico se tratará de algo bastante sencillo. Basándose en lo propuesto en la Figura 6.1, y con la ayuda de NetBeans, se crea un GUI bastante funcional (Figura 6.5). Con respecto al modelo inicial se incluye la posibilidad de elegir el criterio de ruta. Aún no contiene sección de gestión de incidencias.

Se compone de un panel (*JFrame*), dos menús desplegables (*JComboBox*) conteniendo la lista de pueblos cada uno, un grupo (*ButtonGroup*) de tres botones de radio (*JRadioButton*) para las opciones de ruta, un botón (*JButton*) para el envío del formulario, y una superficie de texto (*JTextArea*) donde aparecerán los resultados.

Se centraliza todo en una única clase (*GUI.java*) que recibe como argumento la lista de pueblos. Contiene un método (*jButton1ActionPerformed*) que se ejecuta al enviar el formulario.

Este método evalúa el estado de los botones de radio y menús desplegables, vacía de contenido el área de texto y llama al método *calculatePath()* contenido en el paquete principal.

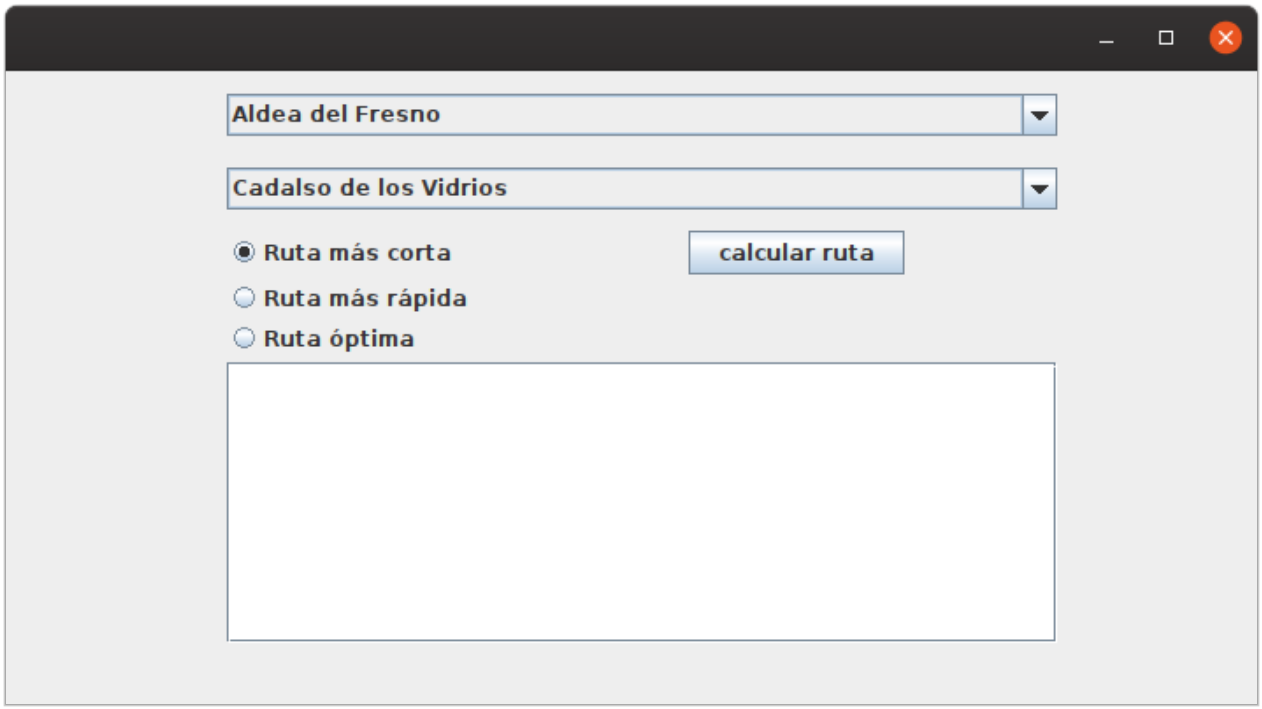


Figura 6.5: primera versión del entorno gráfico

Sprint 8: Algoritmo de recorrido del grafo (3 abril)

Se finaliza el algoritmo de recorrido del grafo, en la clase CalculatePath() y ya se pueden ver resultados en el entorno gráfico sobre una traza de ruta (Figura 6.6).

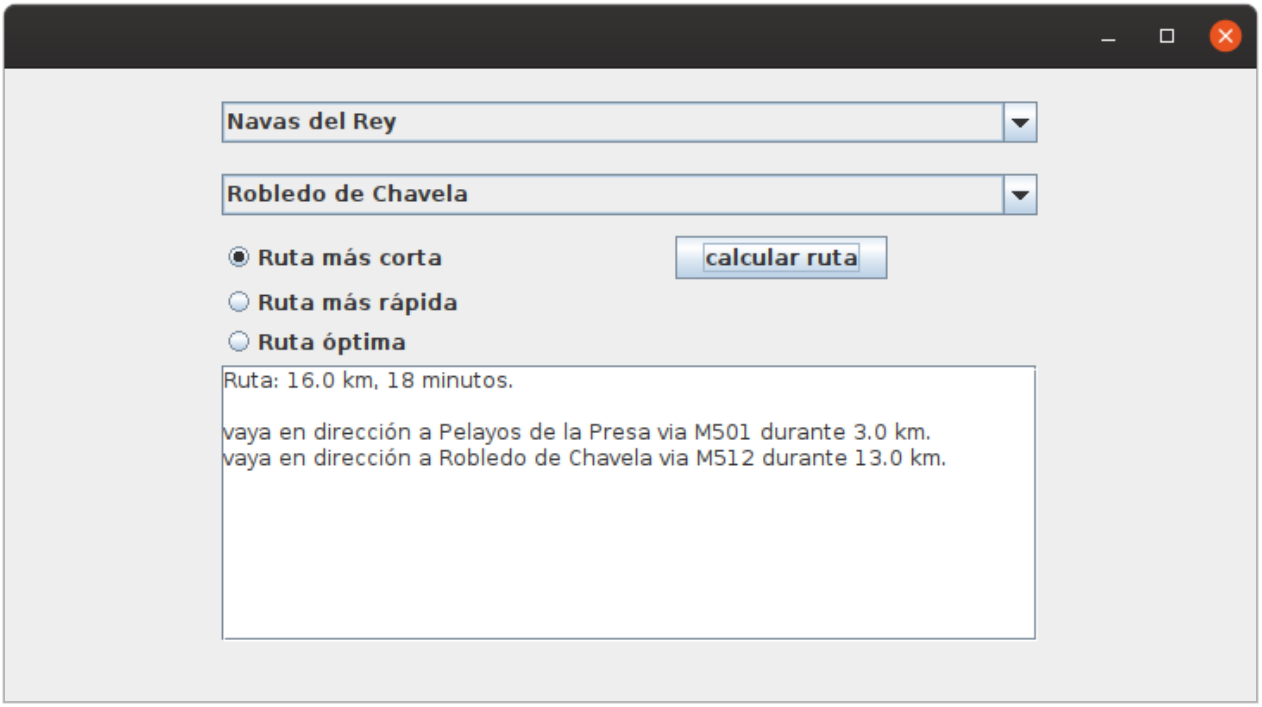


Figura 6.6: resultado en la guía de ruta

Sprint 9: Gestión de incidencias (13 abril)

Se añade la sección de gestión de incidencias a la GUI (Figura 6.7). Se compone de tres menús desplegables (*JComboBox*) conteniendo la lista de carreteras, la de tramos, y la de incidencias activas; dos botones (*JButton*) para el envío de los formularios, y una superficie de texto (*JTextArea*) donde aparecerán los resultados.

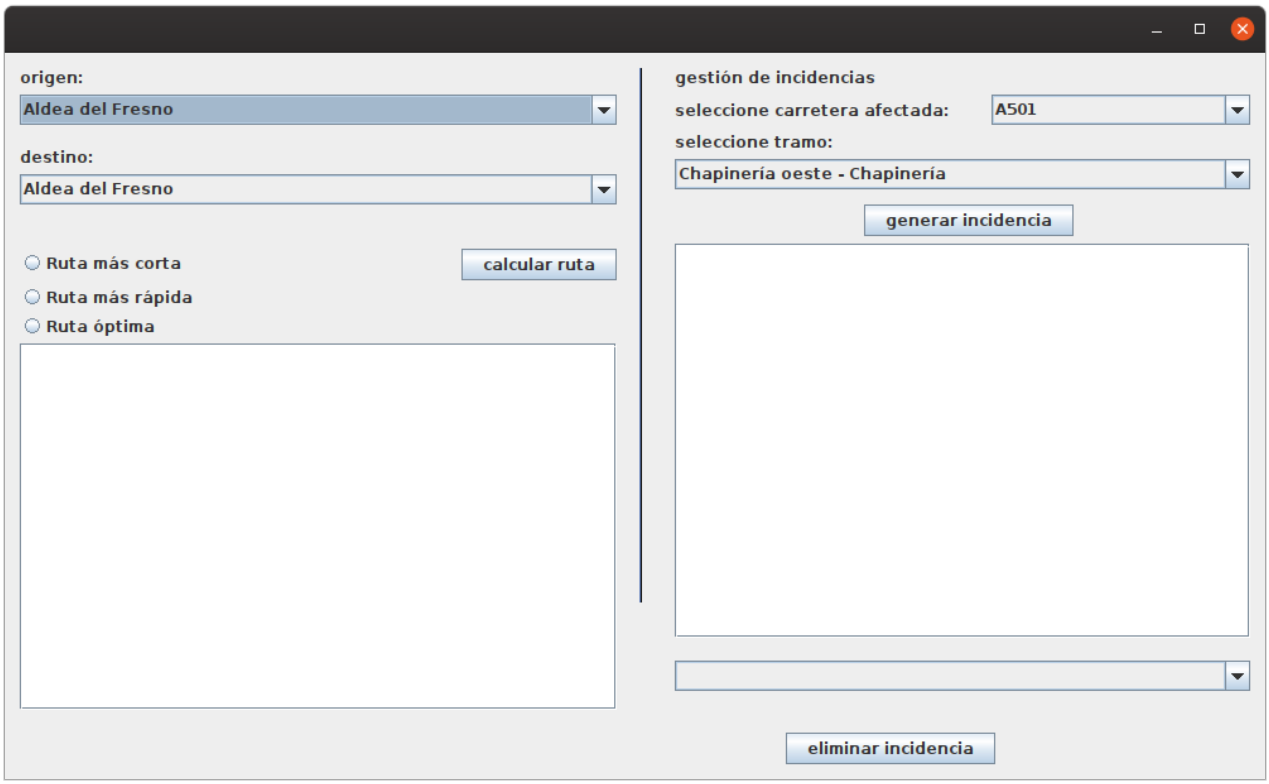


Figura 6.7: aspecto de la aplicación con la gestión de incidencias

Sprint 10: Batería de pruebas (20 abril)

Se realizan las pruebas especificadas en la sección 5.12. Las pruebas constan de trazado de rutas con distintos criterios, generación de incidencias, trazado de rutas con incidencias, eliminación de incidencias y trazado de rutas con incidencias eliminadas. Se encuentran pequeños errores que son subsanados, sin afectar a la estructura de la aplicación.

Sprint 11: Comentado y limpieza de código (7 mayo)

- Ligeros retoques al interfaz gráfico (Figura 6.8).
- Se propone a terceros el uso de la herramienta para conocer su usabilidad y cómo de intuitivo es su uso.
- Se añaden comentarios en el código donde todavía no había.
- Se revisa el código en busca de redundancias, partes inútiles e incongruencias.
- Se pasa la herramienta SonarLint para analizar el código (Figura 6.9). Se eliminan gran parte de los errores detectados. Quedan unos cuantos que no es posible eliminar (muchos son código invariable que produce NetBeans, y otros, al solucionarlos, provocan que la aplicación no funcione correctamente). La mayoría de los errores solucionados son sobre sintaxis de nombres de variables, clases y paquetes.

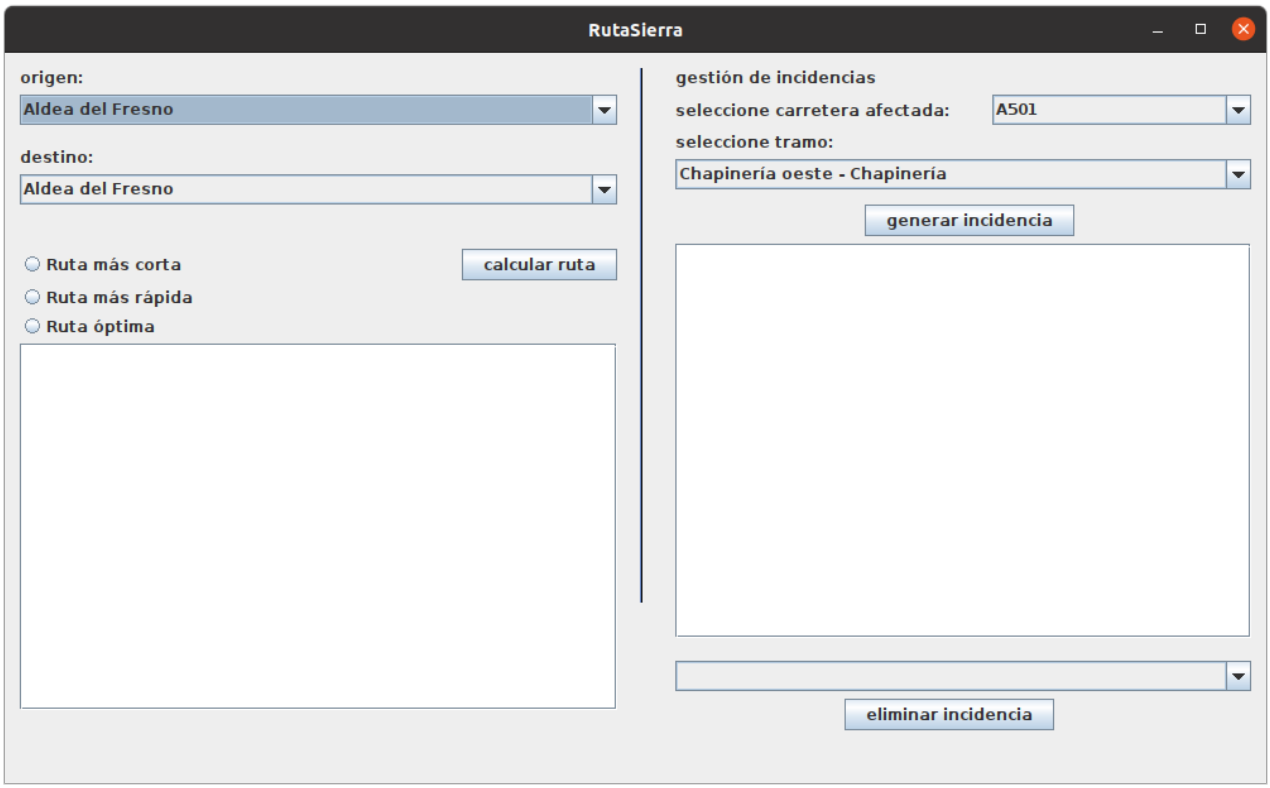


Figura 6.8: interfaz gráfico de la versión 0.1

- Se generan los correspondientes javadocs (Figura 6.10).

- Se realiza una versión reducida de la batería de pruebas del *Sprint* 10 para asegurar que, tras las modificaciones realizadas, la aplicación continúa funcionando perfectamente.

SonarLint Report

34 items

Resource	Date	Description
CalculatePath.java	5 days ag	Refactor this method to reduce its Cognitive Complexity from 26 to the 15 allowed.
EdgeFetch.java		Hide this public constructor.
EdgeFetch.java		Remove this assignment of "query".
MainBox.java		Remove the "buttonGroup1" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jButton1" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jButton2" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jButton3" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jLabel1" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jLabel2" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jLabel3" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jLabel4" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jLabel5" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jScrollPane1" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jScrollPane2" field and declare it as a local variable in the relevant methods.
MainBox.java		Remove the "jSeparator1" field and declare it as a local variable in the relevant methods.
MainBox.java		Make this anonymous inner class a lambda
MainBox.java		Make this anonymous inner class a lambda
MainBox.java		Make this anonymous inner class a lambda
MainBox.java		Make this anonymous inner class a lambda
MainBox.java		Remove this unused method parameter "evt".
MainBox.java		Remove this unused method parameter "evt".
MainBox.java		Remove this unused method parameter "evt".
MainBox.java		Remove this unused method parameter "evt".
MainBox.java		This class has 6 parents which is greater than 5 authorized.
MainBox.java		Make "cp" transient or serializable.
Node.java	5 days ag	Remove this unused method parameter "origin".
Node.java	5 days ag	This block of commented-out lines of code should be removed.
Node.java	5 days ag	This block of commented-out lines of code should be removed.
Node.java	5 days ag	Refactor this method to reduce its Cognitive Complexity from 47 to the 15 allowed.
NodeFetch.java	few seco	Refactor this method to reduce its Cognitive Complexity from 18 to the 15 allowed.
NodeFetch.java		This block of commented-out lines of code should be removed.
RutaSierra.java	few seco	Remove this unused "edgeFetch" local variable.
RutaSierra.java	few seco	Remove this instantiation of "EdgeFetch".
RutaSierra.java	few seco	Remove this useless assignment to local variable "edgeFetch".

120 files of project rutasierra (at 09/05/2020 09:49)

Figura 6.9: resultado de la herramienta SonarLint

Tras este *sprint* se produce el siguiente entregable: El código fuente, con sus correspondientes javadocs.

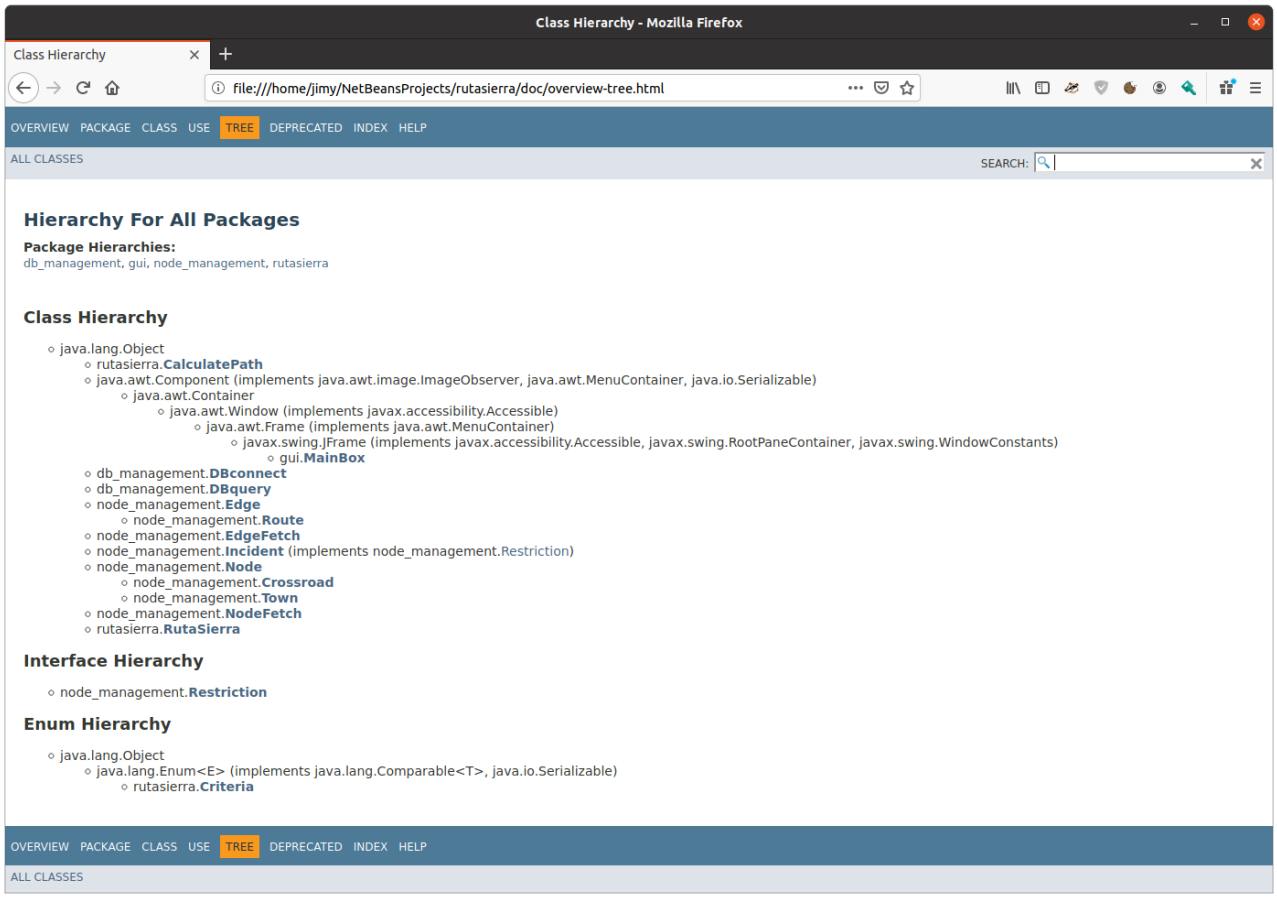


Figura 6.10: portal Javadoc

Sprint 12: Empaquetado (10 mayo)

Este *sprint* produce el entregable: producto final del software, versión 0.1.

Primera revisión (4 junio)

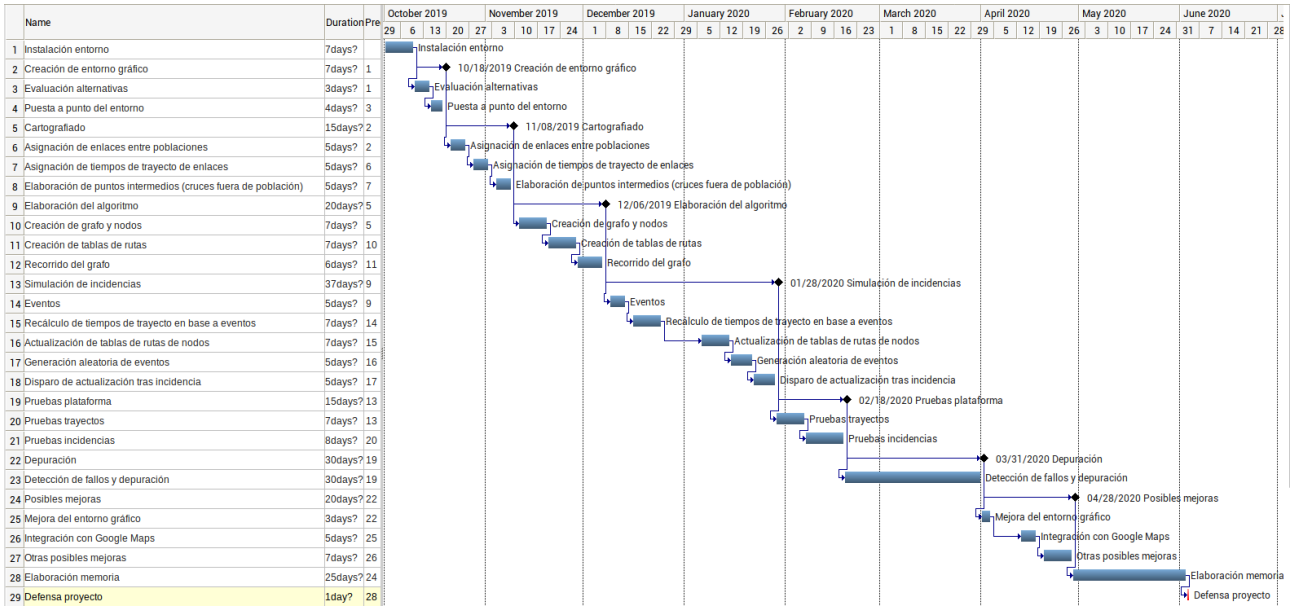
Se soluciona un fallo que arroja la consola cuando se pulsa el botón “eliminar incidencia” si no hay incidencias activas.

Se detecta que en equipos Windows no aparecen correctamente los caracteres especiales, como las vocales con tilde. Se soluciona cambiando la codificación del proyecto (de UTF-8 a ISO-8859-1).

También se realiza una revisión del entorno gráfico para hacerlo más intuitivo.

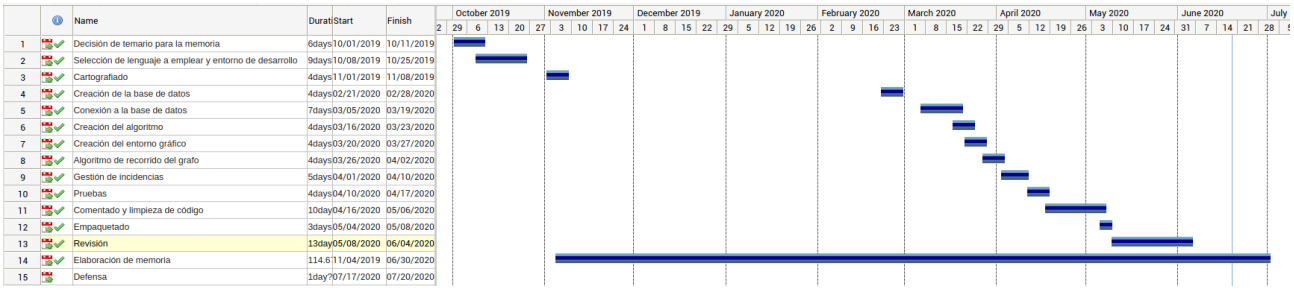
Tras esta revisión el software pasa a versión 1.0.

6.2.2 Diagramas



	Name	Duration	Pre
1	Instalación entorno	7days?	
2	Creación de entorno gráfico	7days?	1
3	Evaluación alternativas	3days?	1
4	Puesta a punto del entorno	4days?	3
5	Cartografiado	15days?	2
6	Asignación de enlaces entre poblaciones	5days?	2
7	Asignación de tiempos de trayecto de enlaces	5days?	6
8	Elaboración de puntos intermedios (cruces fuera de población)	5days?	7
9	Elaboración del algoritmo	20days?	5
10	Creación de grafo y nodos	7days?	5
11	Creación de tablas de rutas	7days?	10
12	Recorrido del grafo	6days?	11
13	Simulación de incidencias	37days?	9
14	Eventos	5days?	9
15	Recálculo de tiempos de trayecto en base a eventos	7days?	14
16	Actualización de tablas de rutas de nodos	7days?	15
17	Generación aleatoria de eventos	5days?	16
18	Disparo de actualización tras incidencia	5days?	17
19	Pruebas plataforma	15days?	13
20	Pruebas trayectos	7days?	13
21	Pruebas incidencias	8days?	20
22	Depuración	30days?	19
23	Detección de fallos y depuración	30days?	19
24	Posibles mejoras	20days?	22
25	Mejora del entorno gráfico	3days?	22
26	Integración con Google Maps	5days?	25
27	Otras posibles mejoras	7days?	26
28	Elaboración memoria	25days?	24
29	Defensa proyecto	1day?	28

Figura 6.11: diagrama de Gantt de planificación del proyecto antes de iniciarlo.



	Name	Durati	Start	Finish
1	Decisión de temario para la memoria	6days	10/01/2019	10/11/2019
2	Selección de lenguaje a emplear y entorno de desarrollo	9days	10/08/2019	10/25/2019
3	Cartografiado	4days	11/01/2019	11/08/2019
4	Creación de la base de datos	4days	02/21/2020	02/28/2020
5	Conexión a la base de datos	7days	03/05/2020	03/19/2020
6	Creación del algoritmo	4days	03/16/2020	03/23/2020
7	Creación del entorno gráfico	4days	03/20/2020	03/27/2020
8	Algoritmo de recorrido del grafo	4days	03/26/2020	04/02/2020
9	Gestión de incidencias	5days	04/01/2020	04/10/2020
10	Pruebas	4days	04/10/2020	04/17/2020
11	Comentado y limpieza de código	10day	04/16/2020	05/06/2020
12	Empaquetado	3days	05/04/2020	05/08/2020
13	Revisión	13day	05/08/2020	06/04/2020
14	Elaboración de memoria	114.6	11/04/2019	06/30/2020
15	Defensa	1day?	07/17/2020	07/20/2020

Figura 6.12: diagrama de Gantt a la finalización del proyecto.

6.3 Costes del proyecto

Dado que no se ha usado equipamiento dedicado, y las licencias de los productos de software son todas gratuitas, se podría decir que el único coste que ha tenido ha sido el tiempo empleado tanto en la planificación como para el desarrollo de la aplicación y la documentación. También ha habido que emplear tiempo en aprender a usar determinadas tecnologías. La Tabla 6.2 detalla las horas dedicadas a cada ámbito.

Planificación	15 horas
Desarrollo software	175 horas
Aprendizaje	10 horas
Documentación	250 horas
Total	**450 horas**

Tabla 6.1: tiempo empleado en el proyecto

Se estima que se ha dedicado un total de 450 horas en el proyecto. A 60 euros la hora el coste total del proyecto asciende a la cantidad de **27 000 €**.

6.4 Conclusiones

En este capítulo, a modo de resumen, se han explicado las fases por las que ha atravesado el desarrollo del proyecto. También se han presentado su representación temporal y su tabla de costes asociados.

Capítulo 7: Conclusiones y trabajos futuros

7.1 Introducción

En este capítulo, escrito una vez terminado el proyecto, se analiza y resume el trabajo realizado, y se proponen trabajos futuros de ampliación y el resto de la implementación del proyecto RutaSierra.

7.2 Conclusiones

Este proyecto ha estado dedicado al desarrollo de una herramienta de software pensada para la ayuda en toma de decisiones al transcurrir por una red de carreteras. Esta herramienta se apoya en el uso de paneles luminosos de señalización, aunque también podría usarse con aplicaciones móviles o mensajería instantánea.

La herramienta cuenta con dos secciones:

- Cálculo de rutas, la cual es usada para calcular la mejor ruta entre dos poblaciones de una región concreta.
- Gestión de incidencias, la cual es usada para crear y cerrar incidencias que puedan suceder en dicha región.

El cálculo de rutas tiene siempre en cuenta las incidencias en vigor. Además, al crear una incidencia inmediatamente se proponen rutas alternativas para poder evitar el tramo afectado.

El sistema se apoya en tres subsistemas, de acuerdo al patrón MVC:

- Un subsistema que gestiona el Modelo: un grafo dirigido formado a partir de la información que se encuentra en la base de datos de cartografía. Este grafo representa en sus nodos a las poblaciones e intersecciones que componen la región, y en sus arcos a las carreteras que los unen. Cada nodo del grafo corre un algoritmo que crea una tabla en cada uno con el camino más corto hacia el resto de nodos del grafo.
- Un subsistema que gestiona la Vista: un interfaz gráfico donde se recogen los datos del operador, y que se usa también para ofrecerle una respuesta en base a los datos obtenidos.
- Un subsistema que gestiona el Controlador. Inicializa el resto de subsistemas, interpreta los datos introducidos por el operador, administra el motor del Modelo para obtener el resultado deseado y lo presenta otra vez al operador utilizando la Vista.

Para el desarrollo ha sido necesaria una fase de planificación en la que se ponen en claro los modelos de datos, el esquema de la aplicación, el cartografiado de la región y el ensamblado. Durante esta fase se han analizado los requerimientos y se ha realizado el diseño por el que regirse durante el desarrollo.

Después, una fase de desarrollo que se ha dividido en subfases, o *sprints*. Estos *sprints* se han fijado de manera dinámica, según avanzaba el proyecto y se definían las necesidades más inmediatas para poder avanzar. Durante esta fase se han ido documentando y realizando pruebas tras cada avance.

Finalmente una fase final de documentación y pruebas, si bien una parte de las mismas se ha ido materializando a lo largo de las fases anteriores, es al final donde más tiempo se ha dedicado a documentar. También durante esta fase se han realizado pequeñas correcciones en el software.

El código fuente y la aplicación han sido sometidos a herramientas de detección de fallos y de rendimiento. Se ha procurado presentar un código bien estructurado y comentado. Se ha procurado reducir el acoplamiento y cuidar el consumo de recursos y, en definitiva, seguir la guía de buenas prácticas en ingeniería del software.

El resultado final ha sido una plataforma monolítica e independiente. Es a la vez cliente y servidor. Capaz de crear y almacenar en el mismo dispositivo donde está corriendo su propia base de datos a partir de archivos intercambiables de cartografía. No necesita conexión a internet. Capaz de ejecutarse en cualquier plataforma en la que se pueda instalar Java 11.

La región cartografiada (Sierra Oeste de Madrid) es un área pequeña con 19 poblaciones que ha servido como prueba de concepto para este software. Sin embargo, dado que se ha definido un estándar para el cartografiado, es posible ampliar la propia región o incluso cartografiar una región completamente distinta siempre que se respete el mencionado estándar. Con simplemente cambiar el archivo de cartografía se puede hacer que el producto funcione usando una región distinta sin tener que realizar cambios en el software.

El objetivo inicial se considera cumplido, siendo la aplicación susceptible de recibir ampliaciones o mejoras en un futuro, o de formar parte de un proyecto mucho más ambicioso.

7.3 Trabajos futuros

Como ya se ha comentado, el software desarrollado en este proyecto forma parte de un macroproyecto llamado RutaSierra, que trata de la conversión de las carreteras de una región en una

Red Inteligente de Carreteras con capacidad para autorregular sus flujos de tráfico, gestionar las incidencias, y mejorar la experiencia de los conductores usuarios de la red mediante evitar en la medida de lo posible congestiones, tramos cortados o información específica a a través de los paneles luminosos; y ofrecer ruta guiada hacia los destinos usando los paneles luminosos de cara a evitar distracciones.

Dentro de este ecosistema, este proyecto se ha centrado en la pieza dedicada al control de rutas, con un motor de cálculo de rutas y gestión de incidencias.

Respecto al cálculo de rutas, aunque tal como se entrega el software es plenamente usable por un operador, el objetivo final es que acoplándole un interfaz de tipo REST (Parsons, 2008) se ofrezca la posibilidad de atender peticiones desde sistemas externos, como los dispositivos de los propios usuarios de la red mediante una aplicación móvil. El sistema devolverá a la aplicación los resultados de la ruta, y adicionalmente, usando la geolocalización del usuario mediante su dispositivo móvil, ofrezca los mensajes correspondientes a través de los paneles luminosos. Para ello, habría que desarrollar el interfaz REST, la aplicación móvil, y el localizador que, triangulando la posición del usuario, pueda localizar el próximo panel luminoso por el que pasará y ofrecer el mensaje que corresponda.

De manera similar, el control de incidencias deberá poder soportar que sea una aplicación externa la que introduzca las incidencias en base a sensores, cámaras o interacción de operadores que no tengan por qué estar físicamente en el centro de control. El hecho de delegar en los sensores y cámaras la detección autónoma requerirá un gran trabajo de desarrollo de una IA que sea capaz de identificar qué es (y qué no) una incidencia. Esta misma IA también deberá detectar congestiones de tráfico y condiciones climatológicas adversas. Al igual que en el cálculo de rutas, se deberá decidir a qué paneles luminosos se envía la información a mostrar según su cercanía al tramo afectado. Además del uso de paneles luminosos se desarrollarán mecanismos automáticos de señalización provisional y balizado que sean capaces, si fuese preciso, de llegar a cortar completamente un tramo desviando de manera satisfactoria el tráfico por vías alternativas sin intervención por parte de operarios.

Haciendo uso de la misma base de datos de topografía se desarrollaría el sistema de control de flujo, consistente en la prevención de congestiones mediante análisis de los flujos estadísticos y actuales de tráfico de cara a regular dinámicamente los semáforos y los límites de velocidad u ofrecer rutas alternativas usando la señalización luminosa.

Respecto a la escalabilidad del producto existente, para el cartografiado, que se ha realizado de manera “artesanal”, deberá buscarse una solución basándose en bases de datos existentes, como las del Instituto Geográfico Nacional, realizando pequeñas modificaciones si fuese necesario.

Finalmente, en lo que respecta a las mejoras propias de la aplicación presentada, se debería implementar un sistema de *logging*, donde registrar las transacciones por operador o máquina en lo que respecta al control de incidencias, así como sistemas de identificación y autenticación de los mismos mediante un mecanismo similar a MTLS.

También sería interesante poder ampliar las categorías de incidencias de tal manera que haya incidencias “menores” que no supongan cortar completamente el tramo afectado, sino simplemente penalizar sus métricas de tal manera que para ciertos trayectos siga mereciendo la pena seguir usando ese tramo, pero para otros convenga más usar recorridos alternativos. A su vez, implementar el hecho de que una incidencia afecte sólo a uno de los dos sentidos del tramo.

Bibliografía

Araujo Serna, Lourdes; Rodríguez Artacho, Miguel; Martínez Unanue, Raquel (2011). «Programación y estructuras de datos avanzadas». *Ed Ramón Areces*

Arias Calleja, Manuel (2013). «Tecnologías y programación integrativas». *Ed. Ramón Areces*

Barnes, David J.; Kölling, Michael (2007). «Programación orientada a objetos con Java». *Pearson*

Cerrada Somolinos, José Antonio; Collado Machuca, Manuel E. (2010). «Fundamentos de programación». *Ed. Ramón Areces*

Cerrada, José A. (2000). «Introducción a la Ingeniería de Software». *Ed. Ramón Areces*

Drozdek, Adam (2007). «Estructuras de datos y algoritmos con Java». *Thomson*

Eckel, Bruce (2007). «Piensa en Java». *Pearson*

Forouzan, Behrouz (2007). «Transmisión de datos y redes de comunicaciones». *McGraw-Hill*

Gajski, Daniel D. (1997). «Principios de diseño digital». *Prentice Hall*

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (2003). «Patrones de diseño». *Addison Wesley*

Gortázar Bellas, Francisco; Martínez Unanue, Raquel; Fresno Fernández, Víctor (2012). «Lenguajes de Programación y Procesadores». *Ed. Ramón Areces*

Hopcroft, J. E.; Motwani, R.; Ullman, J. D. (2008). «Introducción a la teoría de autómatas, lenguajes y computación». *Pearson*

Koffman, Elliot B.; Wolfgang, Paul A. T. (2010). «Data Structures: Abstraction and Design Using Java». *John Wiley and Sons*

Kurose, James; Ross, Keith (2010). «Redes de computadores, un enfoque descendente». *Pearson*

Liu, M. L. (2004). «Computación distribuida, Fundamentos y aplicaciones». *Pearson*

Louden, Kenneth C. (2004). «Lenguajes de programación, principios y práctica». *Thomson*

Parsons, David (2008). «Desarrollo de aplicaciones Web dinmicas con XML y Java». *Anaya*

Piattini, M.; García, F.O.; García, I.; Pino, F. (2015). «Calidad de sistemas de información». *RA-MA*

Ruiz Larrocha, Elena; Ruiz Virumbrales, José Luis (2012). «Sistemas de información de las organizaciones». *Ed. Ramón Areces*

Shalloway, Alan; Trott, James R. (2001). «Design Patterns Explained». *Addison-Wesley*

Silberschatz, Abraham; Korth, Henry F. ; Sudarshan, S. (2002). «Fundamentos de Bases de Datos». *McGraw-Hill*

Weiss, Mark Allen (2000). «Estructuras de datos en Java». *Pearson*

Documentación en la Web:

DRM: Apache Derby 10.14 API Documentation, 2018, The Apache Software Foundation

GSV: Google Street View, 2020, maps.google.com

OSM: OpenStreetMap, 2020, umap.openstreetmap.fr

OSPF: The Internet Society, RFC OSPF v2, 1998, IETF

SO: Sierra Oeste, 2019, www.sierraoeste.org

Listado de abreviaturas

ARP – Protocolo de Resolución de Direcciones

https://es.wikipedia.org/wiki/Protocolo_de_resoluci%C3%B3n_de_direcciones

BGP – Protocolo de Pasarela de Frontera

https://es.wikipedia.org/wiki/Border_Gateway_Protocol

CPU – Unidad Central de Procesamiento

https://es.wikipedia.org/wiki/Unidad_central_de_procesamiento

CSV – Valores Separados por Comas

https://es.wikipedia.org/wiki/Valores_separados_por_comas

EIGRP – Protocolo de Pasarela Interna Mejorado

https://es.wikipedia.org/wiki/Enhanced_Interior_Gateway_Routing_Protocol

GPS – Sistema de Posicionamiento Global

https://es.wikipedia.org/wiki/GPS

GUI – Interfaz Gráfica de Usuario

https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario

IA – Inteligencia Artificial

https://es.wikipedia.org/wiki/Inteligencia_artificial

IDE – Entorno de Desarrollo Integrado

https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado

IGP – Protocolo de Pasarela Interior

https://es.wikipedia.org/wiki/Interior_Gateway_Protocol

ISO – Organización Internacional de Normalización

https://es.wikipedia.org/wiki/Organizaci%C3%B3n_Internacional_de_Normalizaci%C3%B3n

MPLS – Conmutación por Etiquetas Multi-Protocolo

https://es.wikipedia.org/wiki/Multiprotocol_Label_Switching

MTLS – Seguridad Mutua de la Capa de Transporte

https://en.wikipedia.org/wiki/Mutual_authentication

MVC – Modelo-Vista-Controlador

https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

OSPF – Primero el Camino más Corto Abierto

https://es.wikipedia.org/wiki/Open_Shortest_Path_First

REST – Transferencia de Estado Representacional

https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional

RIP – Protocolo de Información de Enrutamiento

https://es.wikipedia.org/wiki/Routing_Information_Protocol

SMS – Servicio de Mensajes Cortos

https://es.wikipedia.org/wiki/Servicio_de_mensajes_cortos

STP – Spanning Tree Protocol

https://es.wikipedia.org/wiki/Spanning_tree

UTF8 – Formato de Transformación Unicode de 8 bits

https://es.wikipedia.org/wiki/UTF-8

Anexos

Anexo A: Tabla de rutas

A	B	B → A	ruta	A → B	min	km
Aldea del Fresno	cruce Arroyo del Santo		M510	Chapinería	7	5.2
	Villamanta		M507		8	7.9
	Villa del Prado		M507		12	10.9
cruce Arroyo del Santo	Chapinería	Aldea del Fresno	M854		6	2.8
	cruce Chapinería oeste	Aldea del Fresno	M510	Navas del Rey	3	2.7
cruce Chapinería oeste	Chapinería	Navas del Rey	M501		2	1.5
	Navas del Rey	Chapinería	M501		2	1.6
Cadalso de los Vidrios	cruce Partida Navarredonda		M542	San Martín de Valdeiglesias	7	7
	cruce Arroyo del Linar		M541	Pelayos de la Presa	9	6.9
	cruce Almorox		M507	Villa del Prado	7	6.8
	cruce Enebrales		M542	Almorox	12	6.6
	Cenicientos		M541		9	6.5
	Rozas de Puerto Real		M507		7	5.7
cruce Partida Navarredonda	San Martín de Valdeiglesias	Santa María del Tiétar	M501		3	3.5
	cruce Arroyo Avellaneda	San Martín de Valdeiglesias	M501	Santa María del Tiétar	6	7.6
cruce Arroyo Avellaneda	Rozas de Puerto Real	Santa María del Tiétar	M507		3	2.4
cruce Arroyo del Linar	Pelayos de la Presa	Cadalso de los Vidrios	M541		12	5.7
	cruce Almorox	San Martín de Valdeiglesias	N403	Almorox	4	4.6
	San Martín de Valdeiglesias	Almorox	N403		5	6.1
cruce Almorox	Villa del Prado	Cadalso de los Vidrios	M507		9	7.8
cruce Enebrales	Cenicientos	Almorox	M544		9	5
Cenicientos	cruce Diseminado Diseminados		M545	Sotillo de la Adrada	5	4.7

A	B	B → A	ruta	A → B	min	km
cruce Diseminado Diseminados	Rozas de Puerto Real	Cenicientos	M546		8	5.3
Chapinería	Colmenar del Arroyo		M510		5	4.7
	cruce Río Perales		M501	Brunete	5	7.2
cruce Río Perales	cruce Encrucijada	Chapinería	M501	Brunete	3	4.6
	Villamantilla	Chapinería	M530		5	4.3
cruce Encrucijada	Villanueva de Perales	Brunete	M524		4	4.9
Colmenar del Arroyo	cruce Valle del Sol		M510	Navalagamella	3	2.1
	cruce NASA		M531	Robledo de Chavela	10	8.8
cruce Valle del Sol	Fresnedillas de la Oliva	Colmenar del Arroyo	M532		9	6.4
	Navalagamella	Colmenar del Arroyo	M510		6	6.7
cruce NASA	Robledo de Chavela	Navas del Rey	M512		12	9.9
	cruce Arroyo Valdezate	Robledo de Chavela	M512	Navas del Rey	3	3.1
cruce Arroyo Valdezate	Navas del Rey	Pelayos de la Presa	M501		3	3
	cruce Arroyo Vallefrías	Navas del Rey	M501	Pelayos de la Presa	4	4.8
cruce Arroyo Vallefrías	Navas del Rey	--prohibido--	M855		7	5.4
	Pelayos de la Presa	Navas del Rey	M501		5	6.3
Fresnedillas de la Oliva	Zarzalejo		M532		7	6.8
	Navalagamella		M521		7	5.6
	Robledo de Chavela		M521		9	8
Pelayos de la Presa	San Martín de Valdeiglesias		M501		5	5.9
Robledo de Chavela	cruce Puerto de la Cruz Verde		M512	El Escorial	12	9.1
	Valdemaqueda		M537		12	6.5

A	B	B → A	ruta	A → B	min	km
cruce Puerto de la Cruz Verde	Zarzalejo	Las Navas del Marqués	M533		12	7.4
	cruce Robledondo	El Escorial	M505	Las Navas del Marqués	2	1.5
cruce Robledondo	Santa María de la Alameda	El Escorial	M535		12	8.4
	cruce La Paradilla	El Escorial	M505	Las Navas del Marqués	3	3.3
cruce La Paradilla	Santa María de la Alameda	Las Navas del Marqués	M538		10	6.9
Villamanta	Villamantilla		M530		6	5
Villamantilla	Villanueva de Perales		M523		5	3

Tabla A.1: tabla de rutas de la cartografía

Anexo B: Manual de despliegue

B.1 Primeros pasos

Contenido del paquete:

- Archivo *RutaSierra.jar* ejecutable.
- Archivos de cartografía *rutas.csv* y *categorias.csv*.
- Directorio *lib*, conteniendo las librerías de DerbyDB.
- Archivo *README.TXT*, con instrucciones de despliegue.

Es importante que todos estos elementos estén en el mismo directorio.

Requerimientos:

- Windows o Linux (no probado en MAC).
- 20 Mb. libres de disco duro.
- 250 Mb. libres de memoria.
- Java 11 o superior.

Para la instalación, abrir una consola de interfaz de comandos, dirigirse al directorio donde se encuentre el archivo .jar y ejecutar el siguiente comando:

```
java –jar RutaSierra.jar
```

Se iniciará la aplicación y en el directorio donde está el paquete aparecerán los siguientes elementos:

- Archivo *Derby.log*, con el log de transacciones de la base de datos.
- Directorio *routetracker*, con los archivos de la base de datos.

B.2 Actualización de la topología

- Sustituir los archivos *rutas.csv* y *categorias.csv* por los archivos con la topología nueva. Deben tener los mismos nombres y la misma estructura.
- Eliminar el directorio *routetracker.*
- Ejecutar de nuevo la aplicación. La base de datos se volverá a crear.

Anexo C: Manual de operación

La aplicación se abre en una ventana que está dividida en dos mitades, como se muestra en la Figura C.1. La mitad izquierda está destinada al cálculo de rutas, y la mitad derecha está destinada a la gestión de incidencias.

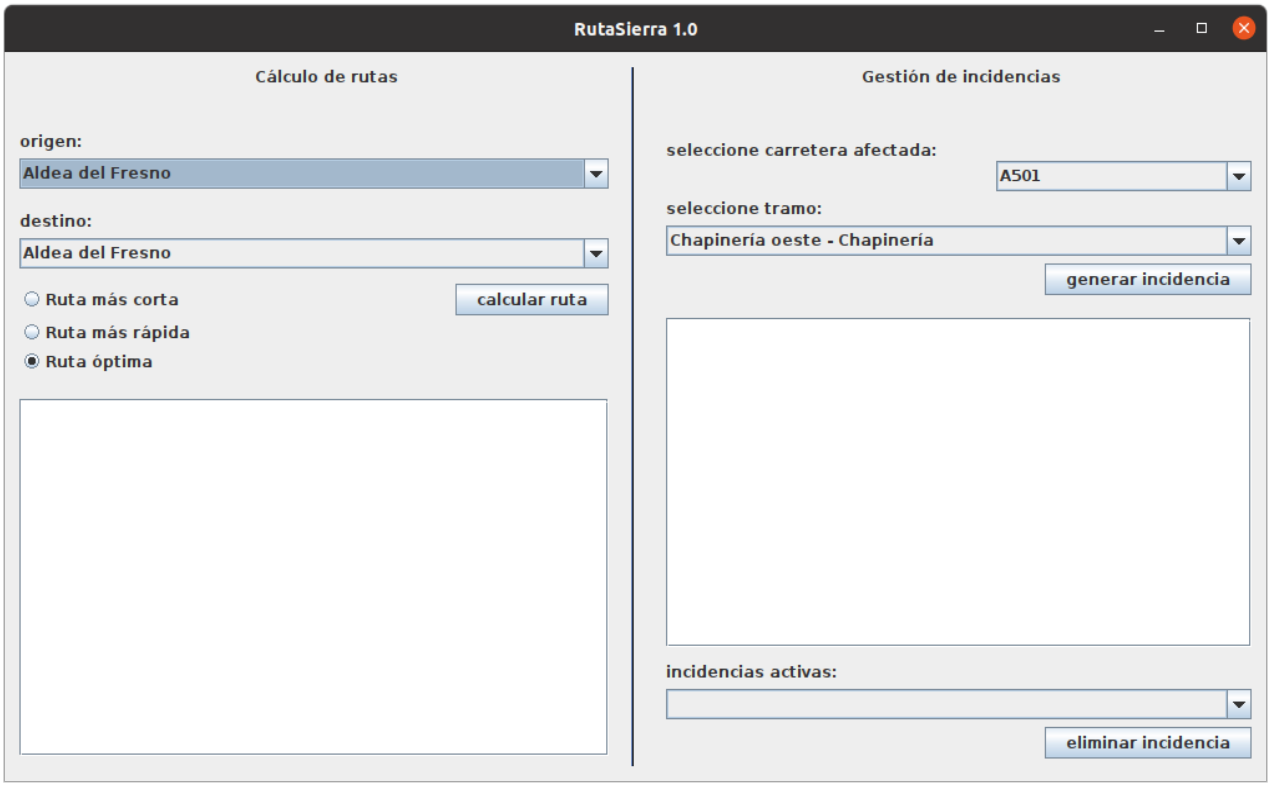


Figura C.1: ventana de la aplicación

C.1 Cálculo de rutas

La mitad izquierda de la ventana de la aplicación contiene un formulario para realizar el cálculo de ruta entre dos poblaciones.

Para realizar dicho cálculo, seleccione el origen entre las poblaciones del menú desplegable "origen" (ver Figura C.2).

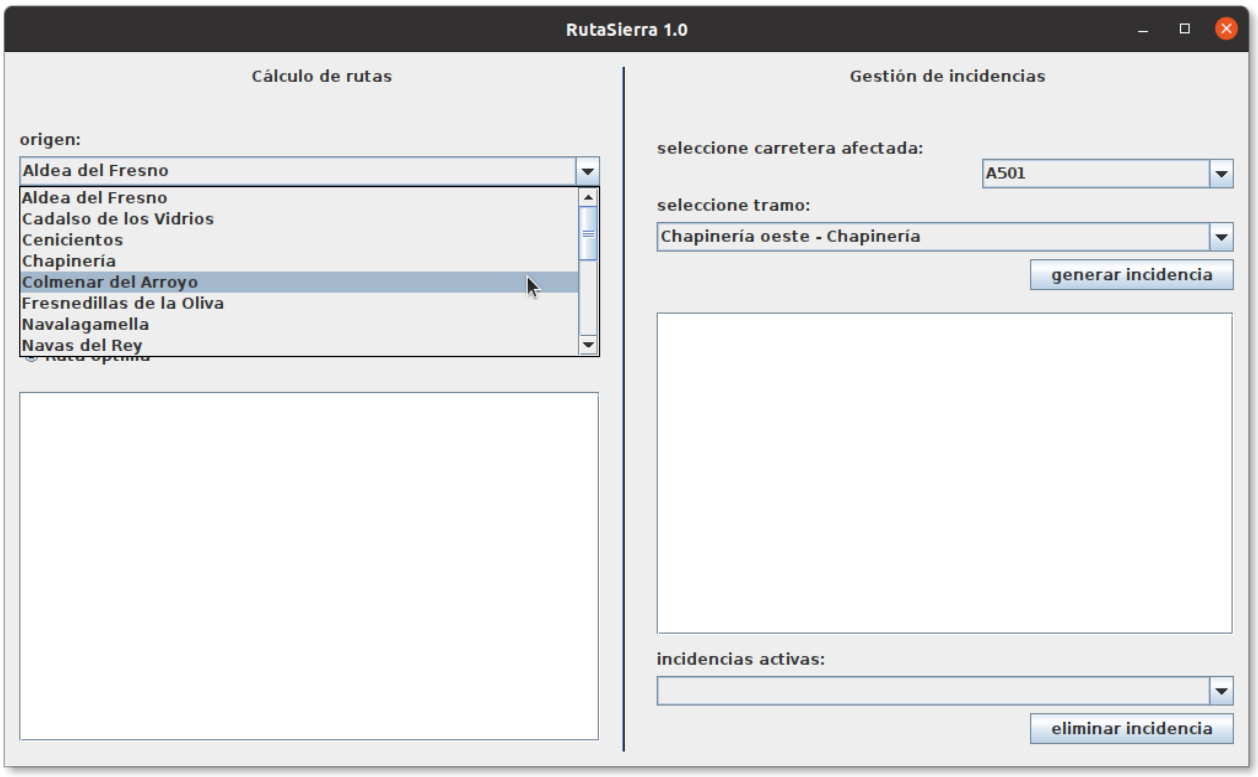


Figura C.2: despliegue del menú origen

A continuación, seleccione la localidad de destino entre las poblaciones del menú desplegable “destino” (ver Figura C.3).

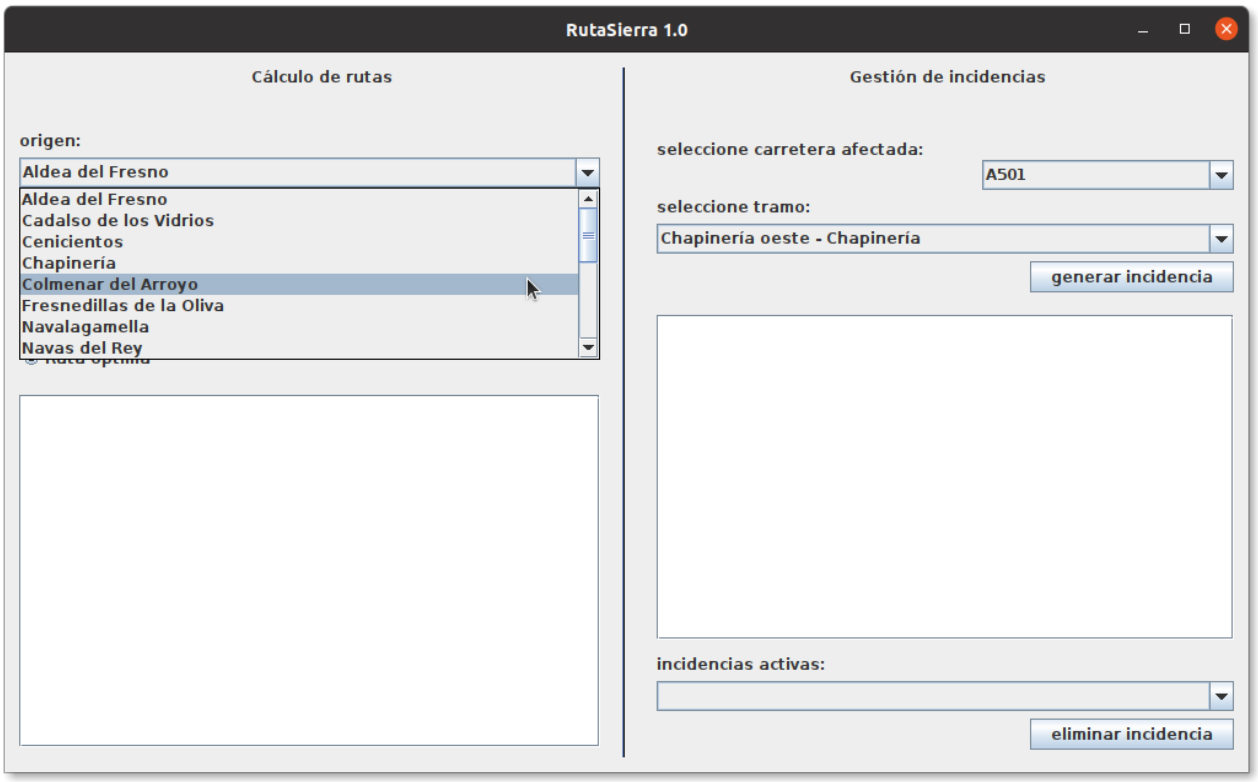


Figura C.3: despliegue del menú destino

Seleccione un criterio de cálculo de ruta, entre “Ruta más corta”, “Ruta más rápida”, o “Ruta óptima” (ver Figura C.4).

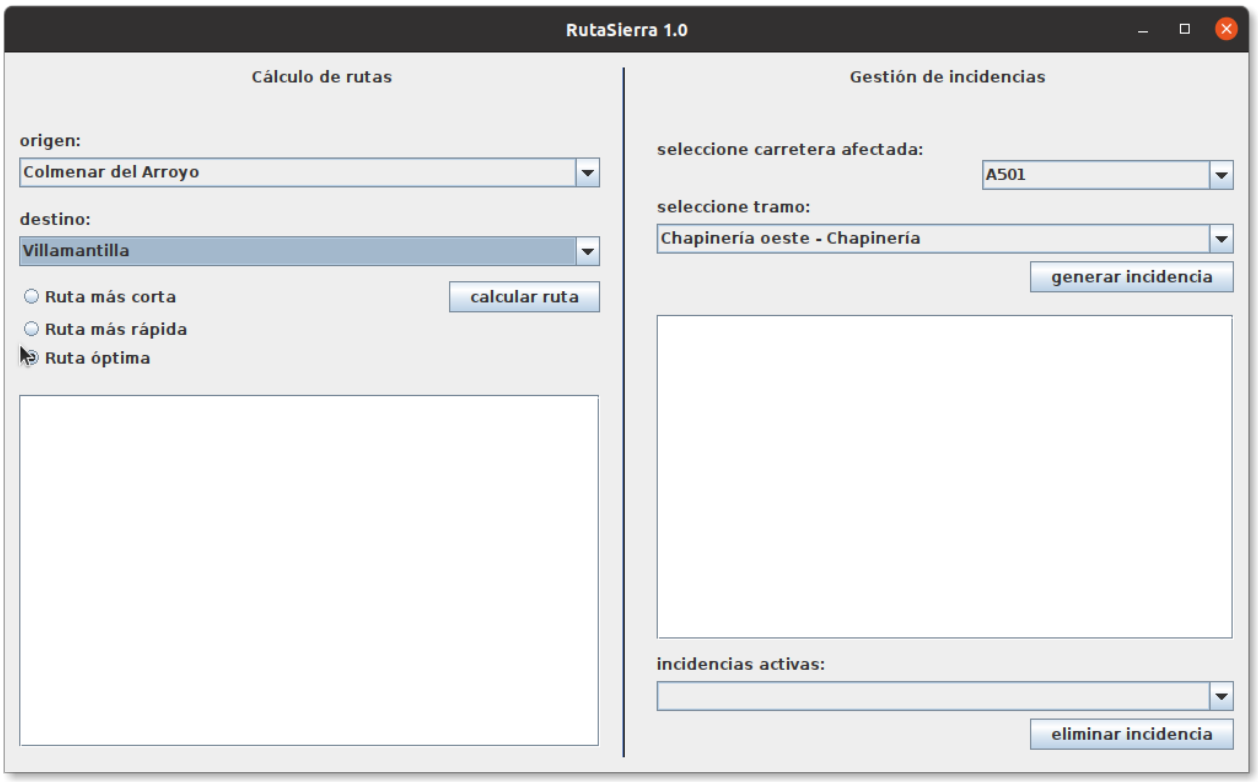


Figura C.4: selección de criterio de cálculo de ruta

Haga clic en el botón “calcular ruta”, y en la ventana inferior aparecerán las instrucciones para realizar el trayecto (ver Figura C.5).

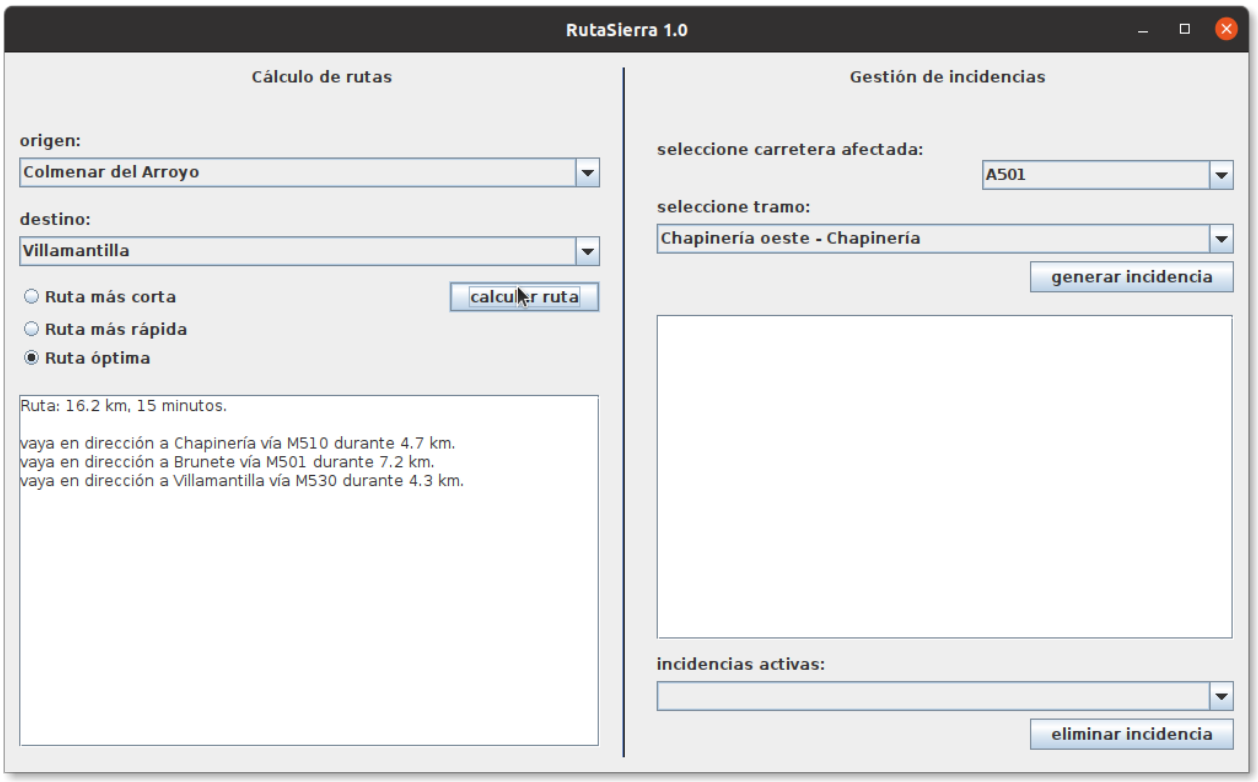


Figura C.5: ruta trazada

C.2 Generación de incidencias

En la mitad derecha de la ventana de la aplicación está la gestión de incidencias. Para generar una nueva incidencia, busque en el menú desplegable donde está la lista de carreteras la carretera donde ha ocurrido la incidencia (ver Figura C.6).

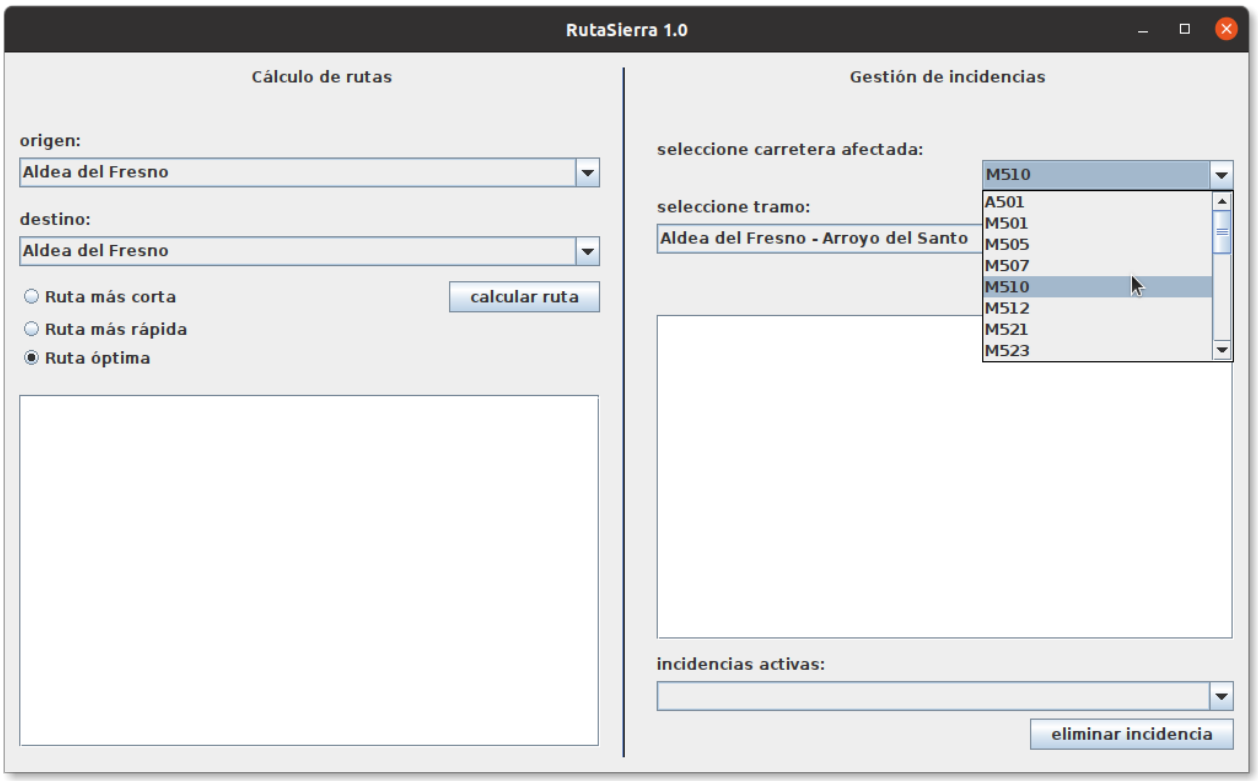


Figura C.6: selección de carretera afectada por incidencia

A continuación, seleccione el tramo afectado en el menú desplegable en el que aparecen los tramos (ver Figura C.7).

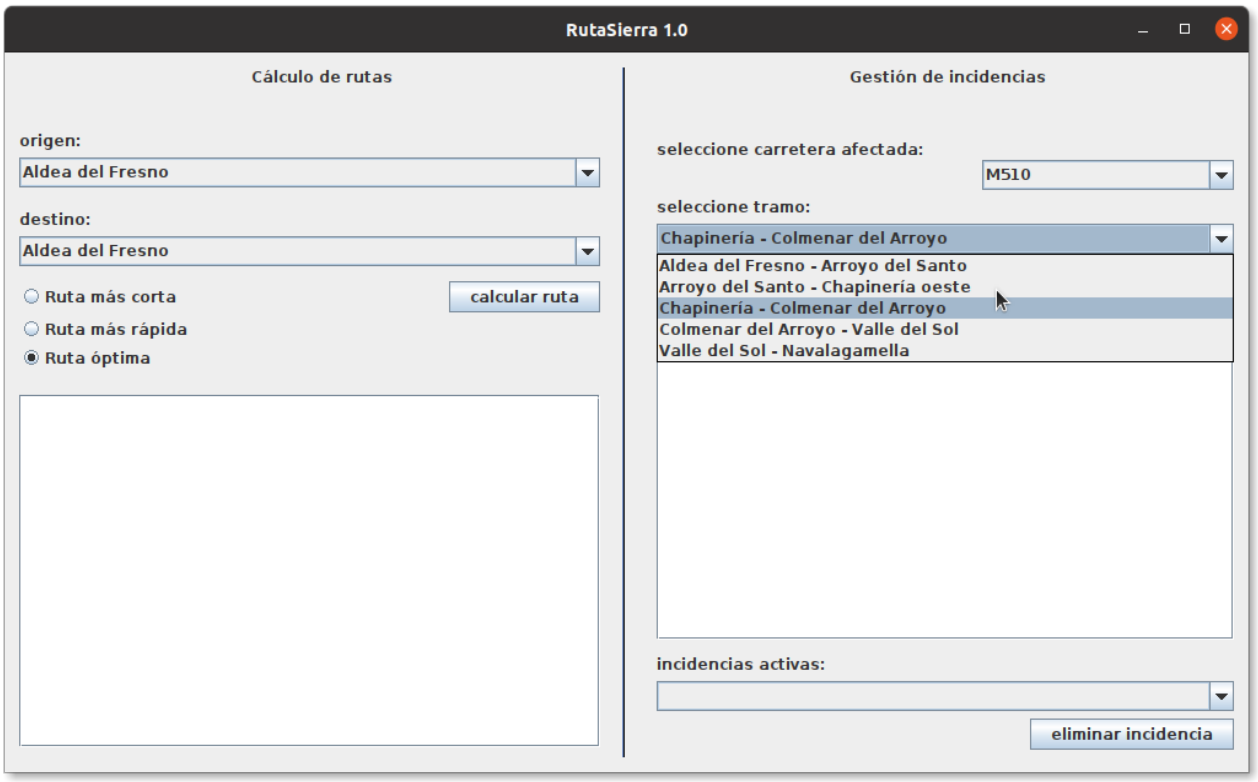


Figura C.7: lista de tramos de una carretera

Haga clic sobre el botón “generar incidencia”. La incidencia quedará registrada. En la ventana inferior aparecerán los mensajes que deberán aparecer en los teleindicadores (ver Figura C.8).

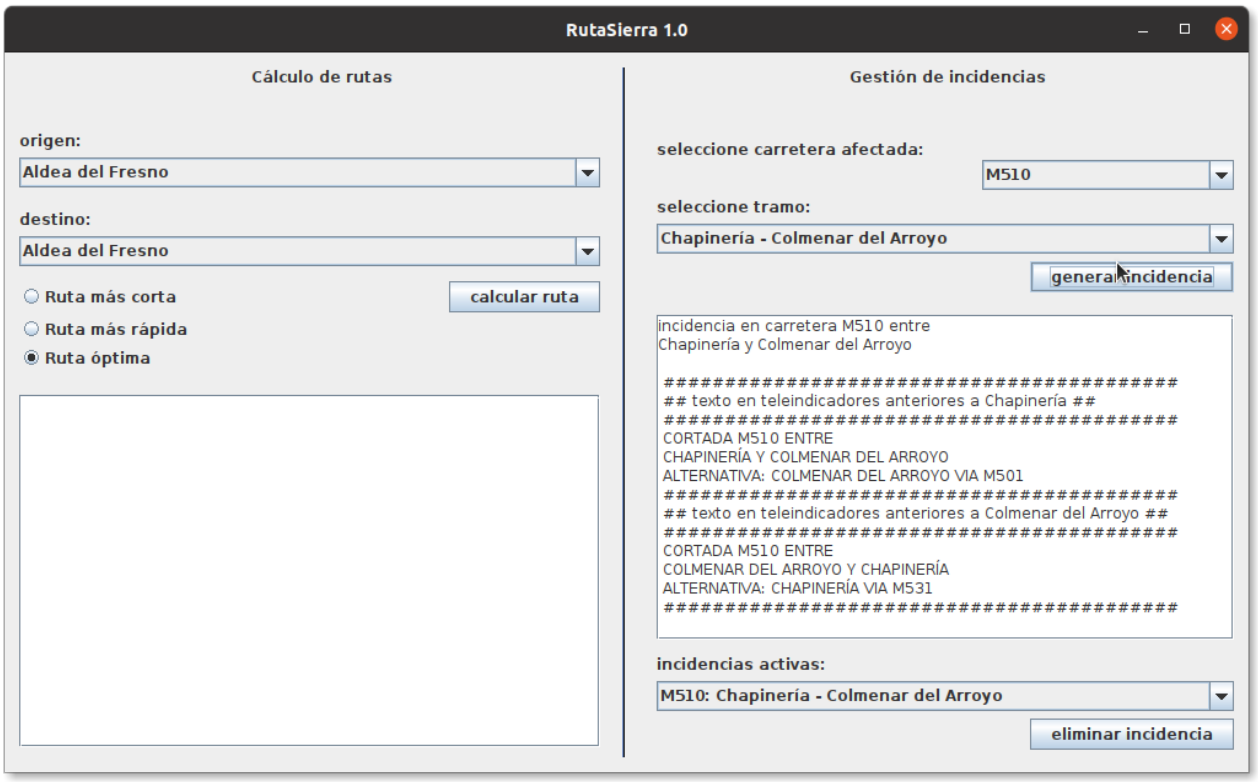


Figura C.8: incidencia generada

Es posible que en los mensajes de los teleindicadores no aparezca ninguna sugerencia de ruta alternativa. Esto significaría que, posiblemente debido a acumulación de inicidencias, no haya alternativas disponibles al tramo afectado (ver Figura C.9).

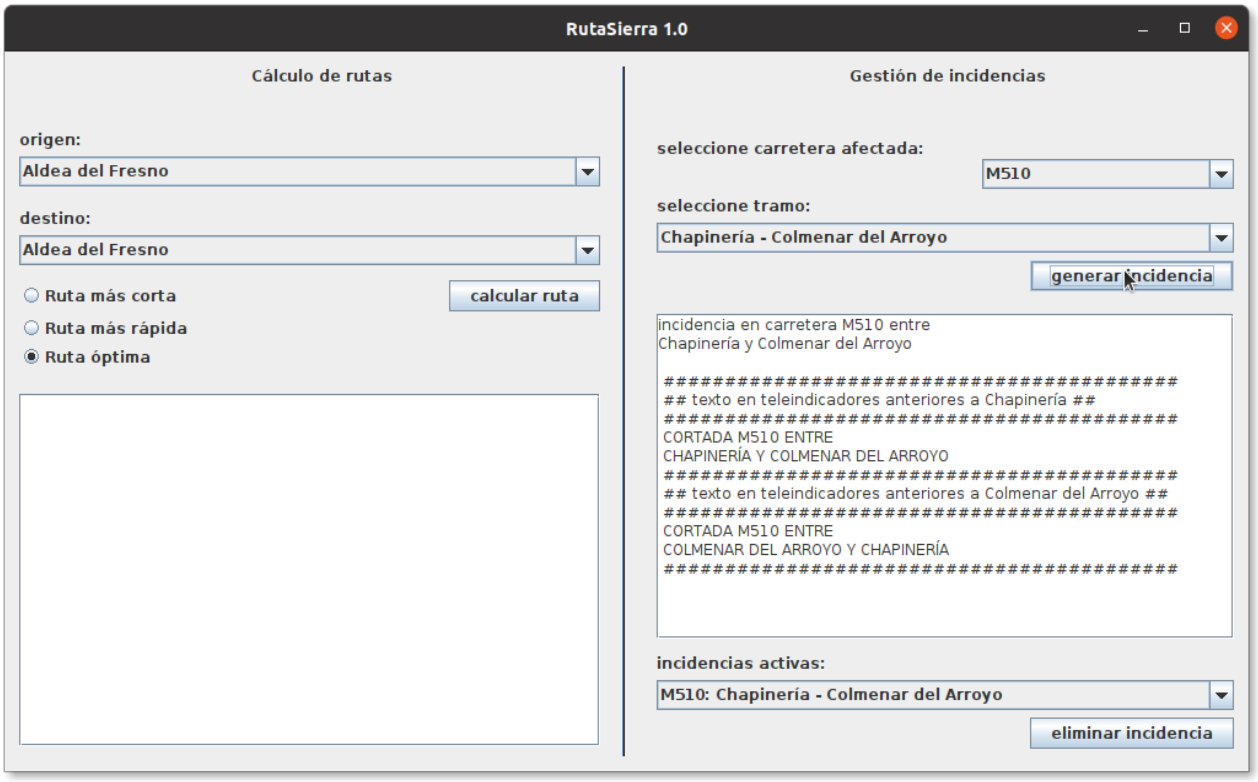


Figura C.9: incidencia generada sin alternativas encontradas

C.3 Eliminación de incidencias

Debajo de la ventana donde se registran las incidencias hay un menú desplegable con todas las incidencias generadas (ver Figura C.10).

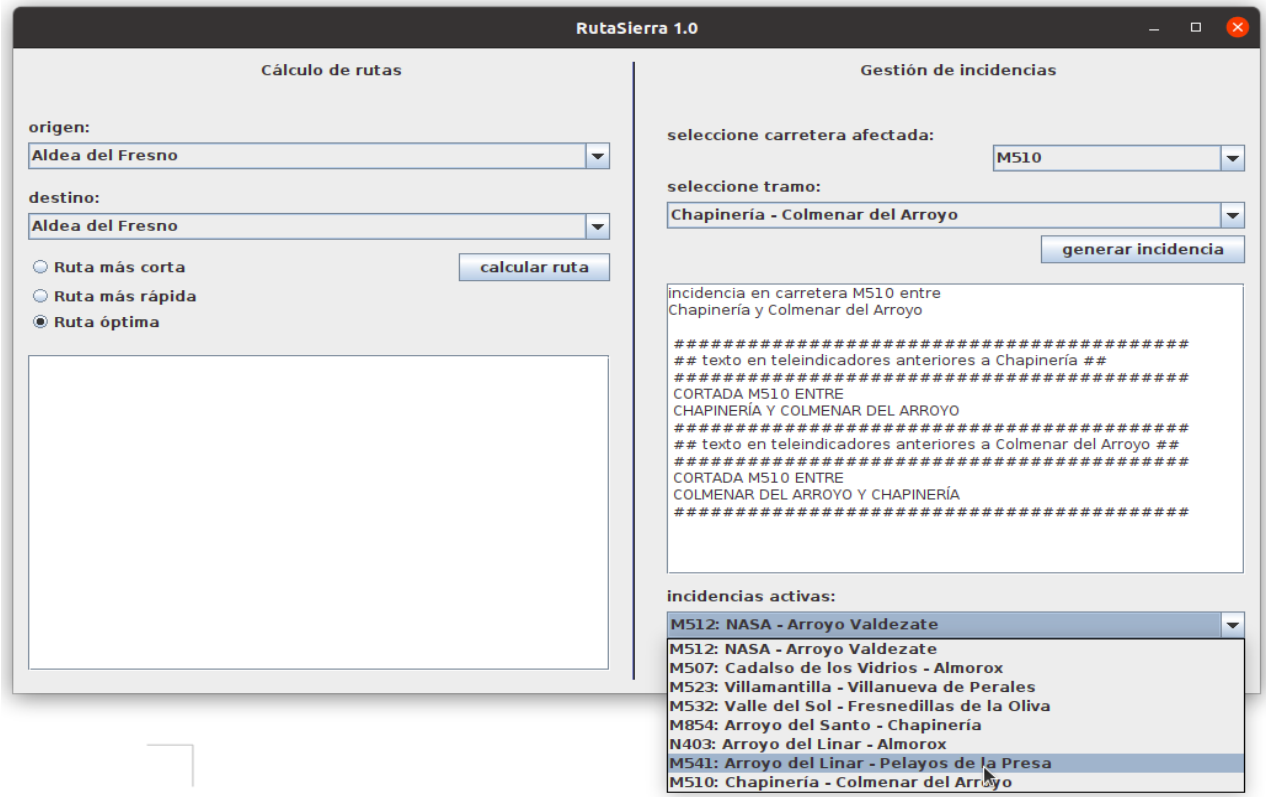


Figura C.10: menú de incidencias

Seleccione la incidencia a eliminar, haga clic en el botón “eliminar incidencia”, y la incidencia quedará eliminada. En la ventana aparecerá el mensaje que deberán aparecer en los teleindicadores (ver Figura C.11).

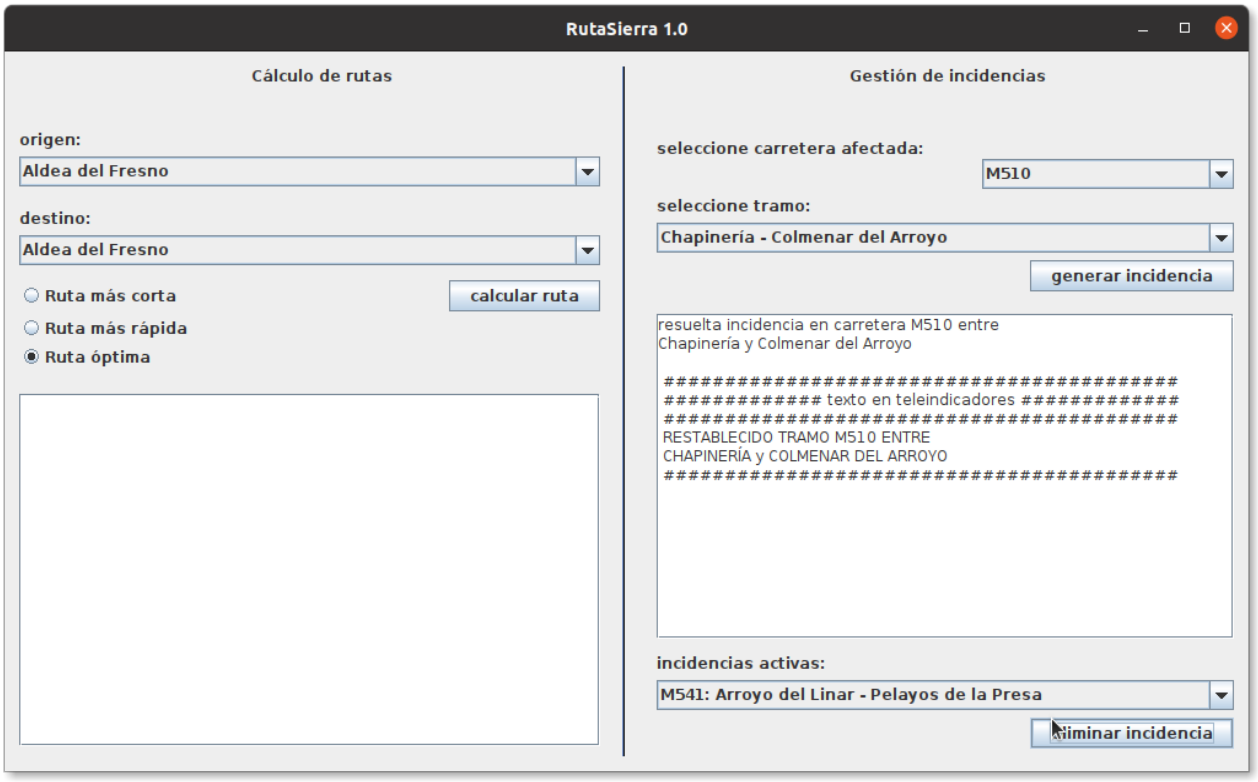


Figura C.11: incidencia eliminada.