Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Herramienta de modelado de carreteras para simuladores de tráfico

Road modeling tool for traffic simulators

Jorge González Cabrera

La Laguna, 1 de julio de 2020

D. **Jesús Manuel Jorge Santiso**, con N.I.F. 42.097.398-S profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Iván Castilla Rodríguez**, con N.I.F. 78.565.451-G profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Herramienta de modelado de carreteras para simuladores de tráfico*”

ha sido realizada bajo su dirección por D. **Jorge González Cabrera**, con N.I.F. 43.382.942-C.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 1 de julio de 2020

Agradecimientos

A mi pareja, por oírme repetidamente hablar de las dificultades de este proyecto y las funcionalidades que tenía pensado añadir.

A Jesús Manuel Jorge Santiso por confiar en mí para la realización de este proyecto y proponerme la beca de colaboración.

A Iván Castilla Rodríguez por esforzarse en hacer un proyecto que me interesara y del que se pudieran conseguir resultados beneficiosos, así como de mantenerme en calma en momentos de estrés.

A mis compañeros por ayudarme durante estos cuatro años de carrera en aquellas tareas que más difícil me resultaban.

A mi hermano por interesarse y procurar que consiga convertirme en un buen profesional.

Licencia

Resumen

El estudio del tráfico se ha convertido en un aspecto muy importante en las sociedades actuales con la creciente cantidad de vehículos en circulación y los atascos y accidentes que esto genera. Para llevar a cabo estos estudios es común el uso de simuladores de tráfico.

El primer paso de las simulaciones se basa en modelar carreteras, fase en la que se enfoca este proyecto. Debido a que normalmente las simulaciones son carreteras existentes o una modificación de estas, serán modeladas a partir de un mapa.

Haciendo uso de una API (Overpass) que utiliza un lenguaje personalizado (Overpass QL) para seleccionar carreteras de OpenStreetMap se ha creado una interfaz de usuario con tres objetivos: reducir el tiempo que requiere esta primera fase, facilitar el periodo de aprendizaje de las herramientas propias del modelado y combinar distintas tecnologías para aportar toda la información necesaria en el proceso.

Palabras clave: simulador, tráfico, modelado, Overpass, carretera, mapa

Abstract

The study of traffic is turning into an important topic in current societies because of the increasing number of vehicles in circulation and the subsequent traffic accidents and traffic jams. In order to carry out the studies it is common to use traffic simulators.

The first step when simulating is modeling the roads that are going to be studied. This is the phase where this project is focused. Keep in mind that the simulated roads are usually existing roads with some (or none) modifications, so they are going to be modelled from a map.

This project comprises a user interface that will model the roads by using Overpass API and its customized language (Overpass QL) that select roads from OpenStreetMap. The objectives are reducing the time that the modeling takes, easing the learning process of the neccessary knowledge when modeling and combining the tools that give information about the roads and the way they are stored.

Keywords: simulator, traffic, modeling, Overpass, road, map

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

El estudio del tráfico se ha convertido en un problema común de las sociedades actuales. Cada vez hay más vehículos en circulación (34.434.791 millones en España en 2019 [1]) y esto genera multitud de problemas. Uno de ellos es la creación de atascos, que a su vez provoca mayor contaminación, por mantener a los coches más tiempo arrancados; pérdida de dinero, al necesitar el suministro de combustible más a menudo y generar retrasos en el camino al trabajo, y una gran probabilidad de generar accidentes, teniendo en cuenta que es un cambio drástico y repentino de velocidad.

Uno de los métodos de análisis y búsqueda de soluciones para los problemas del tráfico se basa en el uso de simuladores. El proceso consistiría en definir adecuadamente el problema (como reducir el tráfico o comprobar que una nueva carretera va a cumplir su propósito) y establecer cuales son los elementos importantes que serán necesarios en la simulación. Es decir, todas las señales de tráfico, semáforos, número de carriles y demás información necesaria para realizar una simulación precisa. Una vez especificado se realiza el modelado de las carreteras.

Posteriormente es necesario establecer el flujo de tráfico, es decir, la cantidad de coches que aparecen en cada entrada en un tiempo determinado y normalmente una matriz origen-destino para determinar las rutas que seguirán. Para conseguir una simulación realista los datos utilizados son el resultado de estaciones de aforo.

Finalmente es posible realizar la simulación y utilizar los datos de salida para analizar dónde se encuentra el problema o si la solución a dado resultado. Es importante hacer simulaciones con distintos parámetros para comprobar cómo se comportaría en casos menos comunes o en distintas franjas horarias.

De esta forma, con el uso de simuladores, podemos justificar el motivo de realizar una obra y los gastos que conlleva de tiempo y dinero.

1.1. Motivación

En años anteriores otros TFG de la Universidad de La Laguna han mostrado la cantidad de tiempo que les ha ocupado el modelado de carreteras en sus estudios de tráfico. En algunos casos muchas semanas. Normalmente las simulaciones se realizan de carreteras existentes o modificaciones de estas por eso lo ideal es modelar las carreteras a partir de un mapa. De hecho uno de esos proyecto trataba sobre la autopista del norte con sus entradas y salidas, añadiendo únicamente unos semáforos [2]. En las fases preliminares de este TFG fue posible realizar la misma selección a partir de un mapa con una herramienta específica en pocos minutos (sin contar con el periodo de aprendizaje que requirió dos

días). Sin embargo, hay que tener en cuenta que, después de esta selección, algunos datos como la temporización de los semáforos pueden requerir un tratamiento manual. Aún así es una ganancia de tiempo significativa. Ese tiempo perdido impidió realizar un estudio más exhaustivo de la simulación. Por lo tanto este proyecto busca evitar que este consumo de tiempo ralentice futuros estudios de tráfico.

1.2. Antecedentes

Existen numerosas herramientas enfocadas a ayudar al procesamiento de carreteras para su uso en simulación u otros análisis.

Overpass API

Overpass API [3] nos permite hacer selecciones personalizadas desde un mapa. Para ello cuenta con dos lenguajes: Overpass QL (Query Language) [4] y Overpass XML. En este proyecto trabajaremos con esta API. Nos centraremos en el primer lenguaje, que está basado en C y nos permite realizar peticiones de forma más clara y reducida. A pesar de la potencia que tiene, sigue siendo un lenguaje que hay que aprender para sacarle beneficio y muchos de los usuarios potenciales de estas herramientas no están acostumbrados a este tipo de trabajo.

Overpass Turbo

Web diseñada para utilizar *Overpass API* y mostrar los elementos seleccionados, así como los datos en diferentes formatos [5]. Ofrece algunas ayudas como el dibujo de rectángulos para seleccionar los elementos en su interior o la utilización de un lenguaje más natural para escribir partes de la petición. A pesar de reducir la carga que podría suponer un lenguaje como *Overpass* sigue siendo necesario tener muchos conocimientos de este.

osmWebWizard

Web para seleccionar carreteras en el interior de un rectángulo [6]. Es bastante limitado pero cuenta con la ventaja de ofrecer también el flujo de tráfico de los distintos tipos de vehículos y de abrir un simulador (concretamente *SUMO*) directamente con los datos recogidos.

JOSM

Java Open Street Map[7] es una aplicación que permite modificar las carreteras en la base de datos de *OpenStreetMap*[8], proyecto desde donde consigue la información de las carreteras y de la que hablaremos en siguientes apartados. Sin embargo, para modificarlas también es necesario seleccionar carreteras primero y esta puede ser exportada para trabajar en el simulador. Trabaja con *Overpass API*, permite utilizar un rectángulo para seleccionar carreteras y utilizar un buscador basado en expresiones que se traducen a *Overpass QL*. Cuenta con la ventaja de poder modificar las carreteras pero también con un periodo de aprendizaje largo. A pesar de eso suele ser una gran alternativa para los expertos.

1.3. Objetivos

El objetivo de este proyecto es agilizar el primer proceso mencionado relacionado con la simulación: el modelado de las carreteras. Se realizará a partir de la selección de carreteras en un mapa. Durante la descripción de los antecedentes pudimos observar algunos márgenes de mejora. En primer lugar, el proceso puede ser muy largo. Si no utilizáramos ninguna herramienta específica tendríamos que crear la carretera en el simulador a mano. Hay que recordar que no sólo es la forma de la carretera, sino las señales, los tramos en los que se puede adelantar, velocidad máxima, etc. Otras herramientas que mencionaremos sólo permiten seleccionar con un rectángulo en un mapa. Si quisiéramos seleccionar la autopista del norte, el rectángulo recogería muchas carreteras innecesarias y sería necesario eliminarlas posteriormente. Por estos motivos el primer objetivo de todos es reducir el tiempo de modelado de carreteras para poder centrar los esfuerzos en encontrar la solución al verdadero problema.

En segundo lugar, el resto de herramientas (incluida *Overpass API*, en la que se basa este proyecto) reducen el tiempo de selección de mapas a cambio de una gran carga de aprendizaje. En concreto *Overpass* utiliza *Overpass QL*, un lenguaje muy similar a los de programación. Quizás para un informático sería la opción más cómoda si es un problema que tendrá que afrontar a menudo porque permite tener más control del proceso. Sin embargo, está tarea no es específica de informáticos y otro tipo de usuarios encontrarían muy complejo aprender a usarlo. Por otro lado, otras utilizan una interfaz para reducir la complejidad pero siguen siendo conocidas por su largo periodo de aprendizaje. Por eso, el segundo objetivo es reducir ese periodo.

Por último, durante las pruebas de ejecución del proyecto y de las herramientas ya existentes fue necesario utilizar muchas herramientas diferentes. Para seleccionar carreteras en el interior de un polígono hay que saber las coordenadas de los vértices y para ello accedemos a *Google Maps* [9]; para saber si las etiquetas que estamos utilizando son las correctas accedemos a *TagInfo* [10] y para saber con que etiquetas han sido almacenadas las carreteras que queremos seleccionar accedemos a *OpenStreetMap* [11]. Esto genera desconfianza en la aplicación o sensación de no tener tanta importancia así que como último objetivo se propuso procurar que no fuera necesario salir de la aplicación para obtener más información.

Capítulo 2

Diseño

2.1. Descripción general de la aplicación

El proyecto se basa en crear una aplicación de escritorio programada en *Python* para seleccionar carreteras de un mapa y modificar los datos para utilizarlos en una simulación de tráfico. Como es normal necesitar varios intentos para conseguir el resultado esperado se mostrará un mapa con las carreteras seleccionadas dibujadas en él de forma que sepamos como actuar. De igual forma se proporcionará un método para ver los datos recogidos en forma de tabla en función de sus propiedades e incluso una alternativa en la que cada fila mostrará un grupo de carreteras conectadas entre sí pero separadas del resto. En esa tabla se podrá seleccionar una fila para reconfigurar la selección y acotarla a lo que representa dicha fila.

Se dispondrá de una consola para informar al usuario de los posibles errores o inconvenientes que pudieran surgir durante la ejecución de la aplicación junto a una forma de solucionarlos si fuera posible. Además, también se informará de cuando empiezan y/o acaban los pasos de una ejecución para valorar el tiempo que podría tardar en terminarse.

Esencialmente la aplicación utilizará *Overpass API* para seleccionar las carreteras con su lenguaje *Overpass QL*. A bajo nivel cada configuración utilizada se traduce en una instrucción de dicho lenguaje. Para evitar complejidad innecesaria sólo se explicarán en esta memoria aquellas más básicas. Sin embargo, el usuario dispondrá de la posibilidad de escribir la petición manualmente aunque desaproveche algunas funcionalidades de la aplicación.

La unidad básica de selección de carreteras se llaman *requests*, que seleccionan a partir de una serie de características que deben cumplirse simultáneamente. Entre esas características se incluyen el dibujo de un polígono o la especificación de una localización para seleccionar los elementos en su interior, la descripción de las propiedades que deben tener o la inclusión de las carreteras cercanas a las previamente seleccionadas. Se proporcionarán plantillas que generen *requests* comúnmente utilizadas que ahorren al usuario el tiempo que conlleva la edición e investigación necesaria para configurarlas.

Con las *requests* se pueden hacer operaciones básicas entre conjuntos (unión, intersección y diferencia), que resulta en un nuevo conjunto que puede reutilizarse en otra operación. De esta forma pueden realizarse configuraciones más complejas.

También hay unas configuraciones generales que se aplicarán a la petición globalmente, es decir, influyen en todos los conjuntos creados. En este momento sólo se dispone de la fecha de las carreteras que se van a seleccionar, ya que es normal que estas cambien con el paso del tiempo.

La aplicación contiene funcionalidades básicas de un editor como la posibilidad de guardar y abrir las configuraciones que se realicen. Además, se pueden volver a estados anteriores de la configuración actual a modo de historial.

2.2. Tecnologías

En este apartado introduciremos las herramientas utilizadas en el proyecto y aquellas que fueron desechadas en el proceso de selección. Todas son de código abierto atendiendo a las especificaciones iniciales del proyecto.

2.2.1. Simuladores

La elección del simulador es importante ya que los datos recogidos en la aplicación serán guardados en el formato que el simulador entienda.

TRANSIMS (TRansportation ANalysis SIMulation System)

Simulador diseñado para planificación de transporte regional siguiendo un modelo microscópico, es decir, implica un gran nivel de detalle de los comportamientos de los vehículos. El objetivo de este simulador era poder analizar problemas importantes en el transporte, como el desarrollo sostenible, los impactos ambientales y el despliegue de Sistemas Inteligentes de Transporte (ITS). Fue finalmente desechado por la falta de información disponible [12].

MATSim

Simulador basado en agentes especialmente diseñado para simulaciones a gran escala [13]. El *framework* consiste en una serie de módulos que pueden ser combinados e incluso incluir alguno realizado por nosotros mismos. Sin embargo, a pesar de ser de código abierto y tener su propio visualizador (para ver las carreteras y los vehículos en movimiento), recomiendan utilizar uno externo que no es de código abierto porque no mantienen dicho visualizador. Este fue el motivo que impidió su uso. No obstante, aún es posible ver los datos resultado de la simulación.

SUMO

Simulador de tráfico multimodal continuo diseñado para manejar grandes redes de carreteras [14]. Sigue un modelo microscópico. Cuenta con una considerable fuente de documentación y con varias herramientas relacionadas a la selección de carreteras. Permite calcular las emisiones de ruido y contaminación, realizar el análisis de la seguridad de una carretera y la inclusión de vehículos autónomos entre sus múltiples funcionalidades. Finalmente fue utilizado en el proyecto.

2.2.2. Gestión de mapas

OpenStreetMap

Una vez decidido que se trabajaría con *Overpass API* la decisión de como gestionar los mapas ya estaba hecha. *OpenStreetMap (OSM)* es un proyecto colaborativo para

crear mapas a nivel global y acceder a dicha información. Al ser colaborativo es posible encontrarse con carreteras a las que les falta información básica (como la velocidad máxima permitida), etiquetas en las que no se siguen las recomendaciones establecidas u otro tipo de inconsistencias. Es usada por todas las herramientas de selección de carreteras mencionadas en los apartados anteriores y *SUMO* dispone de métodos para modificar el formato de los datos de *OSM* al suyo propio.

2.2.3. Visualización de mapas

Para que el usuario pueda comprobar si las carreteras que está seleccionando son las que le interesan es necesario mostrarlas dibujadas en un mapa. En el proceso de selección muchas herramientas fueron estudiadas pero en general proporcionaban una visualización poco profesional o no estaban preparadas para dibujar carreteras fácilmente.

Mapnik

Es una herramienta mencionada en la wiki de *OpenStreetMap* como una alternativa a utilizar con Python para visualizar mapas [15]. Sin embargo, también mencionan que es difícil configurarla y ese fue precisamente el problema que impidió utilizarla. Además, no ofrecía facilidades para dibujar carreteras con datos de *OSM* como es el caso de la siguiente alternativa.

OSMnx

OSMnx [16] es la opción utilizada en este proyecto para mostrar la selección de carreteras. Permite mostrar carreteras a partir de datos en el formato de *OSM*. Principalmente se utilizó con la librería *matplotlib* [17] pero esta opción no mostraba un mapa como fondo, por eso finalmente se utilizó la opción que trabaja con *Folium* [18]. *OSMnx* manipula los datos de *OSM* para que *Folium* los comprenda. Este a su vez generará un *html* que utiliza la librería de javascript *Leaflet* [19] para mostrar un mapa interactivo y dibujar sobre él los elementos seleccionados.

2.2.4. Otros

Python

Lenguaje de programación utilizado debido al interés personal de aprender dicho lenguaje y a que muchos scripts de *SUMO* están programados en *Python* [20].

Git

Sistema de control de versiones capaz de trabajar con grandes proyectos a gran velocidad [21]. Es especialmente conocido por su modelo basado en ramas que permite un desarrollo no lineal. Tiene una gestión distribuida, es decir, que cada desarrollador tiene una copia local en su ordenador y los cambios se comparten en uno o más servidores remotos.

Pycharm

Entorno de desarrollo interactivo (*IDE*, por sus siglas en inglés [22]) creado por *JetBrains* [23]. Proporciona un depurador, auto-completado y análisis de código y control de versiones integrado entre otras muchas funcionalidades que permite realizar el desarrollo más rápido y de formas más limpia [24].

TagInfo

TagInfo es un sistema para encontrar información acerca de las etiquetas utilizadas en los elementos de *OSM* a nivel global. Dispone de todas las etiquetas usadas al menos una vez (algunas con una breve descripción) y los valores que pueden tener. También cuenta con la cantidad de veces que se utiliza por cada tipo de elemento y la cantidad que se usa cada valor en comparación a las veces que se utiliza la etiqueta.

Networkx

Librería utilizada por *OSMnx* para tratar las carreteras como grafos. Está pensada para trabajar con grandes cantidades de datos (más 10 millones de nodos y 100 millones de aristas) [25]. Será utilizada en algunas funcionalidades para el tratamiento de los datos, entre las que se incluyen la búsqueda de subgrafos que no estén conectados entre sí.

NETEDIT

Aplicación proporcionada por *SUMO* utilizada para editar redes de carreteras y el flujo de vehículos [26]. Será utilizada por el usuario posteriormente al uso de este proyecto para manipular las carreteras que ya existen y añadir el comportamiento de los vehículos. *OSM* proporciona mucha información que es utilizada por *SUMO* para construir el modelo pero admiten que no siempre se encuentra solución a los conflictos y puede que a los elementos seleccionados les falte información. Además, aspectos como la temporización de los semáforos es añadida arbitrariamente porque no hay forma de extraer esa información.

Nominatim

API para localizar elementos de *OpenStreetMap* por su nombre y dirección [27]. Es utilizada por otras herramientas como *Overpass Turbo* de igual forma que se ha utilizado en este proyecto.

PyQT5

Binding de la biblioteca gráfica *Qt* [28] para el lenguaje de programación Python [29]. Qt es un framework multiplataforma orientado a objetos ampliamente utilizado para el desarrollo de interfaces gráficas y gracias a sus *bindings* es la biblioteca utilizada en este proyecto.

qtmodern

Paquete de *Python* utilizado para mejorar la apariencia de los componentes de *PyQT5* y añadir el modo oscuro [30]. En gran parte es el responsable de la elección de colores, la forma en la que se sombrean los botones y otros aspectos de la interfaz, pero no interviene en la organización de los componentes ni su comportamiento.

2.3. Conceptos básicos de OSM

Actualmente no es necesario saber como funciona el simulador (hasta que sea el momento de simular) ni cualquier otra herramienta utilizada en el proyecto a excepción de *OSM*. En está sección se comentarán los aspectos básicos que debería conocer un usuario antes de empezar a usar la aplicación.

2.3.1. Tipos de elementos

OpenStreetMap trabaja con tres tipos de elementos [31]. Nodos o *nodes* (puntos localizados con coordenadas), caminos o *ways* (unión de puntos ordenados que aportan un significado juntos) y relaciones o *relations* (agrupación de elementos, como carreteras que forman una línea de autobús). Sin embargo, también existe el concepto de *area* que hace referencia a un camino o relación cuyo primer y último elemento es el mismo.

Los nodos son los únicos elementos que tienen coordenadas, los demás elementos son capaces de ubicarse gracias a los nodos de los que están compuestos y su orden. Es decir, que las carreteras (*way*) pueden tener más de un nodo, normalmente para cambiar el ángulo. A pesar de esta separación no es posible seleccionar sólo un fragmento de la carretera. Todos los segmentos forman parte del mismo elemento.

OSM no está únicamente diseñado para trabajar con carreteras. Por lo tanto es posible que en algún momento seleccionemos los bordes de un parque o de un edificio. En el caso de que ocurra *SUMO* será capaz de distinguirlos de las carreteras así que no habrá ningún problema.

2.3.2. Etiquetas de los elementos

En OpenStreetMap cada elemento es almacenado con una serie de etiquetas del modo clave-valor que la describen. De esta forma podemos saber si un elemento de tipo way es una carretera o un edificio, o diferenciar la velocidad máxima de dos carreteras. Cada elemento puede tener un número arbitrario de etiquetas e incluso la presencia o no de una etiqueta puede aportar información.

2.4. Interfaz gráfica

En mi opinión las aplicaciones de código abierto tienen tendencia a minusvalorar la estética y la usabilidad en pos de concentrarse en el funcionamiento. El funcionamiento es indudablemente importante pero un comienzo confuso puede ser la diferencia entre usar una aplicación u otra. Por eso en este proyecto se ha intentado presentar una aplicación lo más limpia e intuitiva posible.

Al entrar en la aplicación pueden distinguirse cuatro partes como puede verse en la figura 2.1. Un mapa, para ver los resultados de nuestras búsquedas y comprobar si realmente es correcto; una consola, para mostrar información de los procesos que se ejecuten y los posibles errores; un cuadro de texto en modo sólo lectura, donde se escriben las peticiones en el lenguaje *Overpass QL* y, por último, una caja de herramientas para definir las peticiones de las carreteras. La caja de herramientas también se divide a su vez en varias partes: *Requests*, *Operations*, *General* y *Disambiguation*.

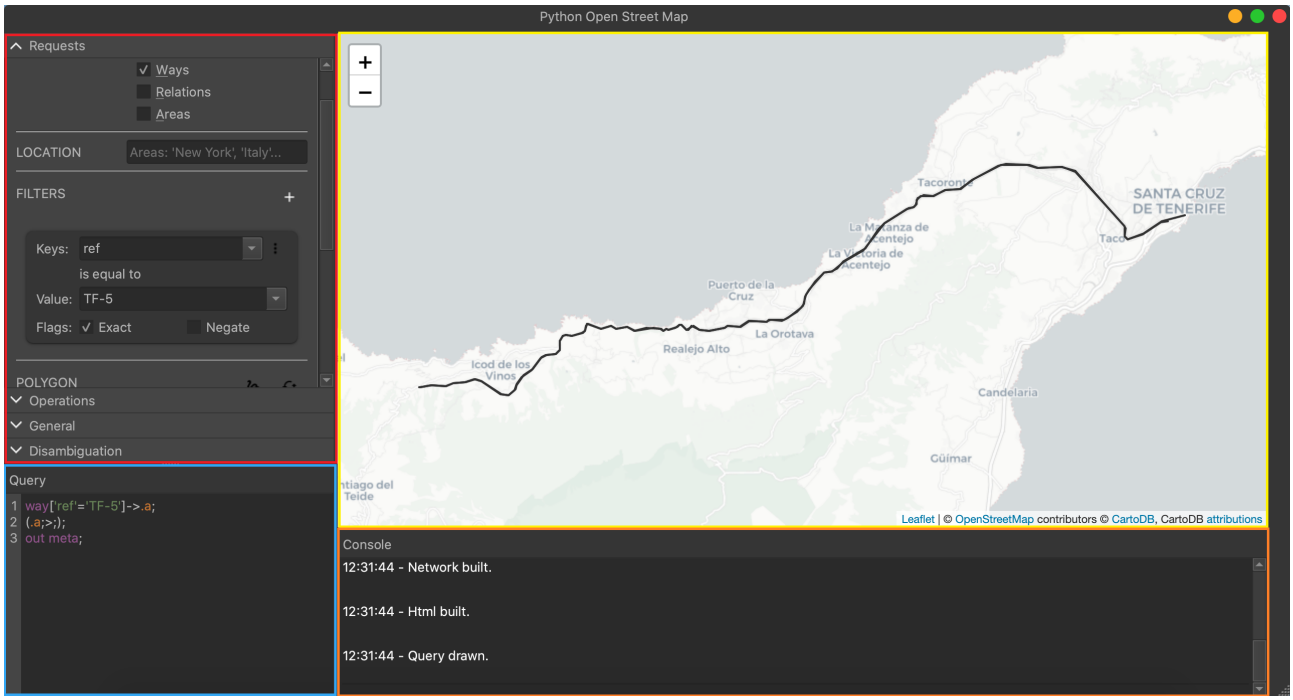

Figura 2.1: Muestra de la aplicación en ejecución y sus partes diferenciadas: un mapa (amarillo), una consola (naranja), un cuadro de texto (azul) y una caja de herramientas (rojo).

La caja de herramientas o *Modo interactivo* se ha dispuesto de forma que ocupe el menor espacio posible sin reducir su usabilidad. De esta forma el usuario puede centrar su atención en el mapa. De todas formas, el usuario puede ajustar el tamaño a sus necesidades.

Se ha aprovechado el menú superior y atajos de teclado para reducir la aglomeración de botones. Todas las configuraciones siguen una estructura de formulario y se han creado componentes personalizados para dar sensación de simetría en la aplicación.

El código de *Overpass QL* ha sido debidamente coloreado para mejorar la lectura de aquellas personas que deseen utilizar el modo manual.

Capítulo 3

Descripción funcional

En este capítulo se describen las diferentes funcionalidades de la herramienta desarrollada.

3.1. Requests

Las *Requests* seleccionan elementos de un mapa con una serie de características que tienen que cumplirse a la vez. *Overpass* permite almacenar elementos en conjuntos de datos de la misma forma que lo haríamos en un lenguaje de programación. El nombre de las variables corresponden con el nombre del *tab* que identifica cada *request*. Entre las características disponibles se incluyen:

- **Tipo de elemento**. Es necesario elegir el tipo de elemento que se quiere seleccionar (incluyendo las *areas*). Es posible seleccionar varios tipos de elementos a la vez a pesar de que no suelen tener las mismas etiquetas, excepto las áreas. Si el usuario seleccionase la casilla de las áreas el resto se deseleccionarían y viceversa.

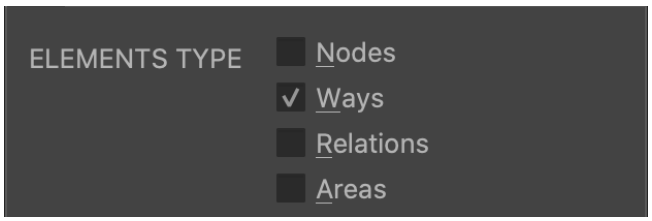

Figura 3.1: Selección del tipo de elemento en una *request*

.

A pesar de que se elige un tipo concreto, como los nodos son necesarios para ubicar el resto de elementos en un mapa la búsqueda final también selecciona los nodos de los que están compuestos los elementos seleccionados. De otra forma, no sería posible mostrar dicha carretera y concretamente *osmnx* no permite ver ningún otro elemento si no están todos los nodos necesarios.

A pesar de poder seleccionar todo tipo de elementos la aplicación está pensada para trabajar con carreteras (*ways*), por lo que no hay utilidades que puedan sacar partido del resto de tipos. Sin embargo, en el futuro podría ser de utilidad.

Este campo, es el único que debe ser estrictamente rellenado además del campo llamado *Alrededores*.

- **Localización**. Entendemos como localización un lugar que pueda interpretarse como un área. Serían válidos valores como ciudades (New York, Madrid), islas (Tenerife, Bali) o países (España, Polonia). Sin embargo, en valores no válidos se incluyen calles, parques o edificios. El nombre aportado sirve para seleccionar todos los elementos en su interior (véase la figura 3.2). Además, tiene la particularidad de no necesitar un nombre exacto. Es decir, los nombres “La Orotava”, “Orotava” y “Orotava, La” ofrecen el mismo resultado.

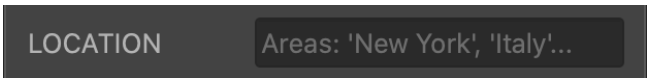

Figura 3.2: Selección de una localización en una *request*

.

Gracias a la API de *Nominatim* es posible localizar un lugar a partir del texto introducido en el *endpoint /search*. En la respuesta se ofrecen varias soluciones pero accedemos al primero de todos y extraemos el identificador del elemento de *OSM*. Cuando comentamos las áreas vimos que realmente eran *ways* o *relations* que forman una zona cerrada. Por eso *Overpass* identifica las áreas en base a estos elementos, concretamente el identificador de un área corresponde a la suma del identificador de un *way* y 2400000000 o a la suma del identificador de un *relation* y 3600000000 en caso de que formen un área. Con el identificador obtenido *Overpass* permite seleccionar las carreteras en el interior de esa área.

- **Filtros**. Las etiquetas de los elementos de *OSM* pueden utilizarse para seleccionar las carreteras adecuadas. Principalmente las comparaciones permitidas por *Overpass* son la igualdad (o desigualdad) de un valor para una clave concreta y la existencia (o no existencia) de una etiqueta. Sin embargo, también permiten el uso de expresiones regulares para hacer peticiones más complejas. Como es improbable que un usuario medio sepa utilizarlas se han realizado componentes específicos para distintas comparaciones que podrían ser de utilidad.

    - **Igualdad**. En este caso es necesario introducir una clave y su valor. Selecciona aquellos elementos en los que la clave dada tiene el valor especificado. Aparecen dos *flags* para determinar el comportamiento: *Exact*, para utilizar expresiones regulares si no está seleccionada y *negate* para diferenciar entre igualdad y desigualdad. Hay que tener que en cuenta que negar esta comparación implica aceptar que también se seleccionen elementos en los que la etiqueta no aparezca. Con este componente se podría imitar el comportamiento que tienen muchos de los componentes que requieren valor mencionados a continuación. Por lo tanto, se usará de referencia para su explicación.

      Se puede apreciar en la figura 3.3 que al lado de la clave o *key* no nos encontramos sólo con una entrada de texto, sino también una lista desplegable. Si la abriéramos veríamos cien claves oficiales (aparecen en la wiki de *OSM*) en orden descendente por número de elementos que la utilizan en la base de datos de *OpenStreetMap*. De esta forma no es necesario saberse todas las etiquetas y podrá aportar ideas de cuál podría ser la más adecuada. Las etiquetas están disponibles gracias a la API *TagInfo*, a la que se llama en cada ejecución de la aplicación para asegurar tener la

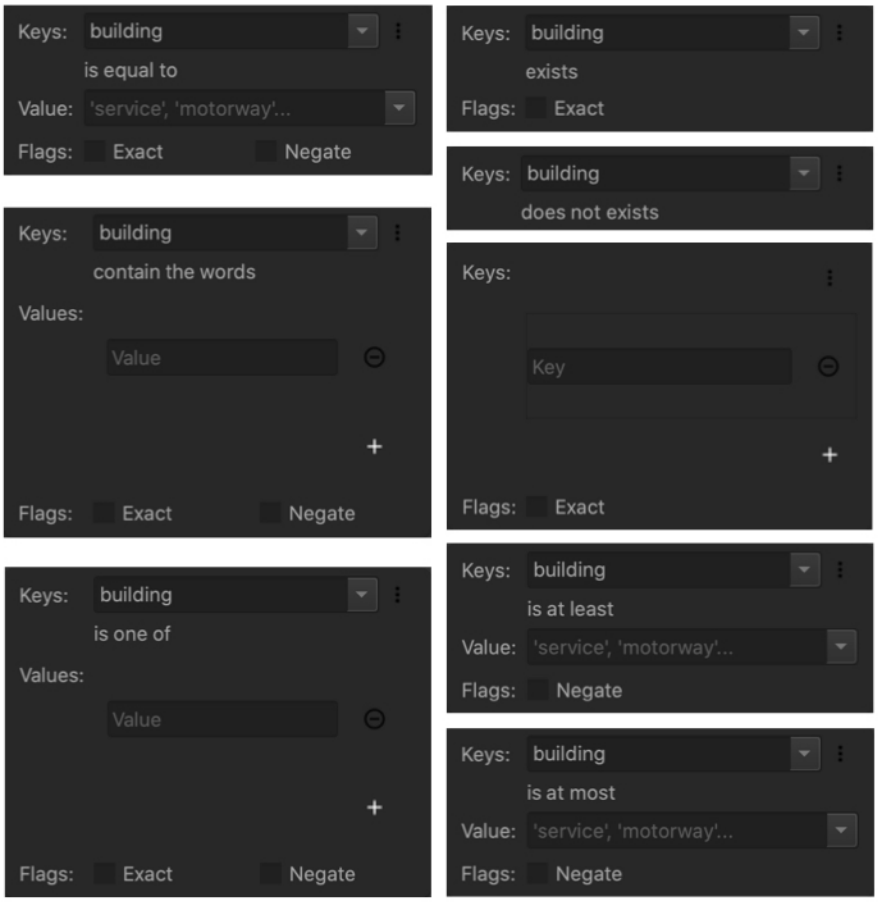

Figura 3.3: Componentes para comparar la clave y el valor de las etiquetas.

información actualizada. Es posible acceder a esta información con la url: **https://taginfo.openstreetmap.org/api/4/keys/all?filter=in_wiki& sortname=count_all&sortorder=desc**.

A la derecha de la lista desplegable se puede apreciar un botón con tres puntos alineados en vertical. Al pulsarlo aparece un menú con una opción para eliminar el filtro y otra que pone *help*. Este segundo botón proporciona en la consola una descripción de la clave actualmente escrita, sea o no parte de la lista desplegable. Hay que tener en cuenta que no todas las etiquetas tienen un descripción, pero en cualquier caso no tenerla indica que es poco usada y podría no ser la mejor opción.

Por último, el valor también está acompañado de una lista desplegable. Debido a la gran cantidad de valores existente no están incluidos por defecto. Pero al pulsar el botón *help* se incluirán. Es posible que no se muestren los valores disponibles aún cuando la etiqueta existe. Esto se debe a que algunas son sumamente específicas a una carretera, como el nombre. Lo que se traduce en una lista desplegable muy larga y poco usable que puede tardar muchos minutos en cargar. Por ello, si la petición para conseguir dichos valores tarda más de un tiempo determinado no serán añadidos. Estos valores también son conseguidos gracias a la API de *TagInfo*. El enlace para conseguir los datos es: **https://taginfo.openstreetmap.org/api/4/key/values?key=** seguido del nombre de la clave.

- **El valor contiene todas las palabras**. Aunque existen ciertas pautas para

almacenar valores de *OpenStreetMap* es un hecho que no siempre se siguen. Además, hay muchos detalles que tampoco se mencionan. Es posible tener una carretera localizada en “La Laguna” y otra localizada en “Laguna, La” o una carretera cuyo primer tramo se llame “Autopista del Norte” y otro se llame “Autopista Norte”. Para seleccionar estos casos podríamos usar una comparación que comprobará que algunas palabras están incluidas independientemente del orden y del resto del valor.

A bajo nivel esta comparación es equivalente a utilizar cada palabra en una comparación de igualdad con la *flag Exact* desactivada. De esta forma, la expresión regular comprobaría que la palabra aparece en el valor y no que es exactamente la misma. Este es un claro ejemplo de que es posible utilizar una misma clave en varios filtros de una misma *request* y que pueden ayudar a aportar más información. Debido a esta implementación la *flag Exact* se traduce en escapar todos los caracteres especiales si es marcada gracias a la función *re.escape* de Python. La negación de está comparación comprobaría que ninguna de esas palabras aparece, pero también aceptará los elementos en los que no exista la etiqueta.

Por ahora, aquellas comparaciones que tengan varios valores posibles no disponen de la lista desplegable que permite ver las distintas opciones. Sin embargo, sigue siendo posible acceder a la descripción de la clave.

- **Igual a uno de los posibles valores**. Un caso también muy común es que no nos interese sólo un posible valor de una etiqueta. Podemos necesitar las carreteras con dos y tres carriles. En ese caso el componente es exactamente igual al caso anterior, incluido el comportamiento de las *flags*. Este caso se puede realizar utilizando la comparación de igualdad con expresiones regulares: `^(value1|value2|value3)$`. Sin embargo, usando este componente disponemos de otras ventajas como escapar los caracteres especiales de la misma forma que lo hace el filtro anterior.
- **Existencia de una etiqueta**. Ya hemos comentado que las etiquetas pueden aportar información simplemente con su existencia en un elemento. Un ejemplo de ello es la etiqueta *highway* que indica el tipo de carretera y por lo tanto solicitando que exista podríamos asegurarnos de que es una carretera y no un edificio u otra alternativa. En este caso no necesitamos un valor. La *flag Exact* permite que utilicemos expresiones regulares para la clave pero en este caso no podemos negar la existencia de la clave. Para ello disponemos de una comparación diferente. Se ha implementado de esta forma porque *Overpass* no permite utilizar expresiones regulares cuando negamos la existencia. Aunque *Overpass QL* nos permite realizar esta comprobación de forma personalizada, también podríamos realizarla con la comparación de igualdad y un valor que utilice expresiones regulares: `.*`. Aunque de esta forma la interfaz no permite utilizar expresiones regulares en la clave.
- **Inexistencia de una etiqueta**. Como acabamos de comentar en este caso necesitamos un componente distinto para negar una comparación porque en este caso no podemos usar expresiones regulares. El resto de funcionamiento es igual a la comparación anterior.
- **Existencia de una de varias etiquetas**. De igual forma de la que seleccionamos una etiqueta que tenga uno de varios valores establecidos, podemos

seleccionar elementos que tengan al menos una de varias claves. Como son varios elementos es posible usar expresiones regulares pero en caso de que quieran utilizarse valores exactos se escaparán los caracteres especiales. En este caso tampoco es posible utilizar la negación y si se accede al botón *help* se mostrarán las descripciones de cada una de las claves.

- **Máximo**. Algunas etiquetas muy comunes son numéricas, como la velocidad máxima de una carretera o el número de carriles. Es por lo tanto muy útil poder hacer las comparaciones básicas como, en este caso, “menor o igual”.

  Requiere una clave y un valor numérico, pero no tiene sentido el uso de expresiones regulares. Sin embargo, podemos negar la expresión para conseguir la comparación “mayor que”. Al contrario que el resto de filtros esta comparación requiere del uso de condicionales en vez de las comparaciones básicas de *Overpass QL*.

  En etiquetas como *maxspeed* no todos los valores son numéricos, algunos hacen referencia a tipos de vías. En esos casos son excluidos directamente de la selección.

- **Mínimo**. Esta comparación funciona igual que la etiqueta anterior. Indica “mayor o igual”, y en su versión negada indica “menor que”.

- **Polígono**. Es posible seleccionar los elementos en el interior de un polígono dibujado por el usuario. El proceso es simple porque sólo hay dos botones como se muestra en la figura 3.4. El primero es un botón seleccionable que permite al usuario dibujar el polígono haciendo *click* en el mapa. Los puntos se guardan en el orden que han sido introducidos, por eso es recomendable dibujarlo como la silueta de la zona que se quiere seleccionar. El segundo botón borra todos los puntos seleccionados. Sin embargo, también es posible borrar únicamente el último punto con el atajo de teclado *Ctrl+Z* si el foco de la aplicación se encuentra en el mapa.

Figura 3.4: Selección de un polígono en una *request*

.

Las otras herramientas que utilizan *Overpass* sólo trabajan con rectángulos. Esta opción es definitivamente más limitada pero también mucho más eficiente. Por eso se recomienda no seleccionar demasiados puntos o el tiempo de espera puede ser muy grande.

También hay que tener en cuenta que las carreteras y relaciones de *OSM* no se pueden cortar. Por lo tanto, como el polígono selecciona aquellos elementos con alguna parte en su interior puede darse el caso de visualizar carreteras fuera de él.

El mapa que se visualiza es el resultado de un *html* que genera *osmnx* con *Folium*. *Folium* permite recibir las coordenadas del mapa al hacer *click* con el ratón. Esas coordenadas son posteriormente transferidas desde el *html* a la aplicación. De igual forma, cada vez que se actualiza el *html* con una nueva petición se le transfieren las coordenadas del polígono para que se vuelva a dibujar.

- **Alrededores**. Es habitual que en la simulación de tráfico se incluyan las carreteras contiguas a las que se quieren estudiar para añadir contexto o porque son aquellas de las que se tienen datos de flujo. Hay dos formas de abordar este paso (figura 3.5): seleccionando las carreteras inmediatamente unidas a las que ya están seleccionadas o seleccionar aquellas en el interior de un radio establecido de cada uno de los elementos ya seleccionados.

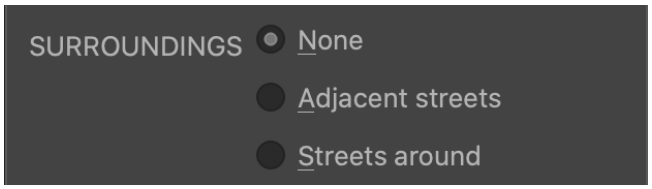

Figura 3.5: Selección de carreteras contiguas en una *request*

.

La primera forma es especialmente útil para casos como una autopista de la que se quieren seleccionar sus entradas y salidas. Para conseguirlo se siguen una serie de operaciones disponibles con *Overpass QL*. En primer lugar se seleccionan los nodos de los que están formados las *ways* y *relations* seleccionadas por el resto de la *request*. Luego se buscan las *ways* que incluyen al menos uno de dichos nodos, lo que resultará en la repetición de las carreteras que ya estaban seleccionadas y la inclusión de otras nuevas. Finalmente se añaden los nodos de las últimas carreteras incluidas. Este último punto es importante porque, como ya hemos dicho, no es posible mostrar una carretera sin sus nodos.

La segunda forma es una funcionalidad directamente ofrecida por *Overpass*. Es bastante más lenta y suele seleccionar carreteras sin ningún tipo de relación con el resto. Sin embargo, la tabla de desambiguación permitiría seleccionar la carretera principal fácilmente. Al seleccionar esta opción aparece un cuadro de texto para añadir el radio adecuado en metros.

- **Identificadores**. Como los elementos de *OSM* están almacenados en una base de datos, tienen un identificador único que podemos utilizar para seleccionarlos.

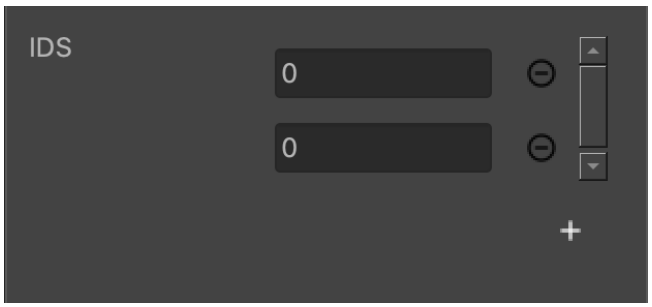

Figura 3.6: Selección de identificadores en una *request*

.

Esta opción no está pensada para que lo utilicen los usuarios. La aplicación tiene otras utilidades más potentes. Sin embargo, la tabla de desambiguación será capaz de utilizarlo para realizar algunas operaciones. Añadiéndolo a la interfaz como se muestra en la figura 3.6 se ofrece un mayor control al usuario en caso de que sea necesario y una mayor transparencia.

3.2. Operations

Como ya hemos visto, cada *Request* selecciona realmente un **conjunto** de carreteras y por ello es posible hacer operaciones básicas entre conjuntos: unión, intersección y diferencia. Estas operaciones permiten al usuario realizar peticiones más complejas e incluso utilizar los conjuntos resultantes para otras operaciones.

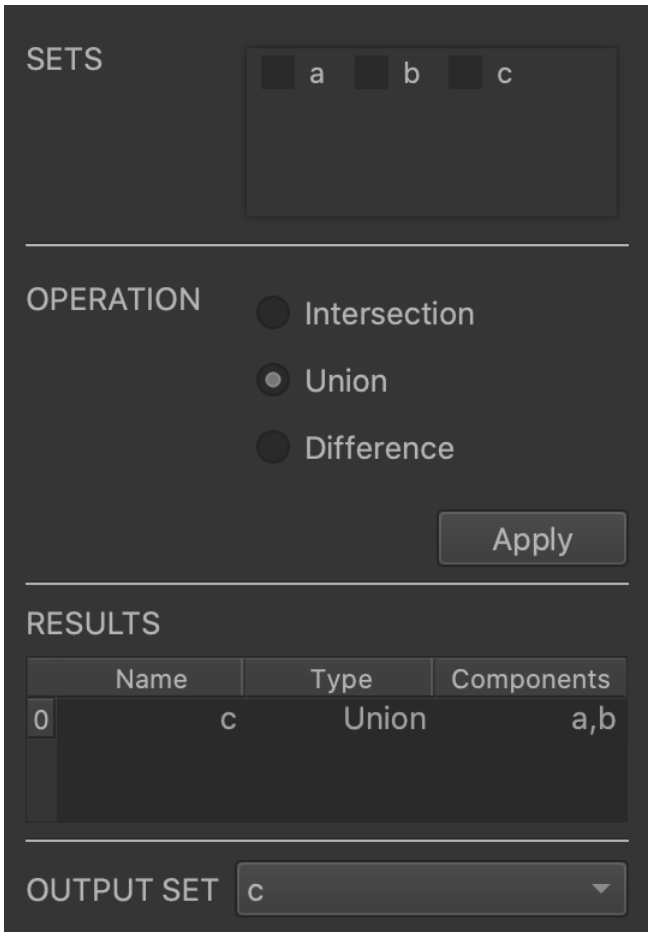

Figura 3.7: Componente para hacer operaciones entre conjuntos

.

En la figura 3.7 también podemos ver que aparece la selección de un *output set*. La verdad es que normalmente no es necesario seleccionar las carreteras de todas las *requests* o los resultados de las operaciones. Si lo que quiero es la intersección de dos conjuntos, ¿por qué también recibo las carreteras seleccionadas por esos dos conjuntos individualmente? El resultado no sería el esperado. Por ello, las carreteras seleccionadas serán el resultado de un único conjunto. En el caso de querer el resultado de varios lo único que se debería realizar es la unión de dichos conjuntos.

3.3. Selección de la fecha de las carreteras

El apartado *General* son una serie de configuraciones que afectan a toda la petición. Sin embargo, actualmente sólo dispone de la selección de la fecha del mapa del cual se seleccionarán las carreteras. Obviamente con el paso del tiempo se construyen más

carreteras, se realizan reformas o se cortan caminos. *OpenStreetMap* tiene el registro de los estados pasados de sus carreteras desde el 12 de septiembre de 2012 y permite el acceso a dichos datos. Como se puede ver en la figura 3.8, las fechas anteriores a la fecha mencionada o posteriores al día actual están inhabilitadas.

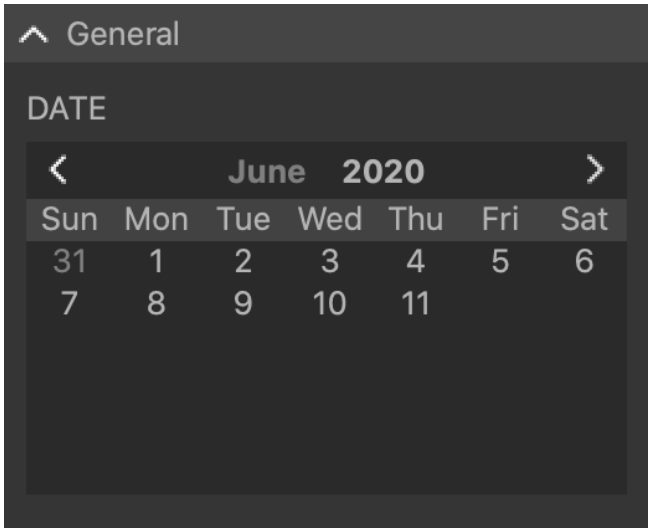

Figura 3.8: Apartado *General* de configuración de la petición.

Gracias a este campo es posible realizar estudios sobre el comportamiento de redes de carreteras pasadas con las condiciones de tráfico actuales o analizar el impacto de nuevas modificaciones viarias.

3.4. Interacción con SUMO

El objetivo último de este proyecto es modelar carreteras para utilizarlas en un simulador. Concretamente trabajamos con SUMO, manipulando los datos para que este pueda trabajar con ellos con un script que se descarga al instalar el simulador. De hecho, también es posible directamente abrir su editor del modelo, llamado *NETEDIT*, con la intención de completar algunos datos. La opción de abrir el simulador no ha sido implementada porque es necesario crear un archivo con el flujo de tráfico para poder realizar la simulación.

Para la conversión de datos se utiliza un *script* de *SUMO* llamado *osmBuild*[32]. En el existe un método *build* que recibe la configuración con la que se generarán los datos. Es necesario especificar el archivo con los datos en formato de *OSM* (opción **-f**) y un archivo de salida donde se guardarán los datos en el formato propio del simulador (opción **-p**). El archivo de salida lo selecciona el usuario, pero el de entrada se crea en cada ejecución de forma temporal hasta que se cierra la aplicación, momento en el que elimina.

Con la opción **–netconvert-typemap** se añaden los *typemaps*, que son ficheros que determinan el comportamiento de la conversión cuando faltan datos. Por ejemplo, el límite de velocidad se puede inferir del tipo de carretera atendiendo a las normas de circulación. Concretamente se usan **osmNetconvert.typ.xml**, que es el utilizado por defecto; **osmNetconvertUrbanDe.typ.xml**, para utilizar la velocidad típica urbana en vez de una por defecto cuando no se encuentra de forma explícita en los datos y **osmNetconvertPedestrians.typ.xml**, que añade aceras junto a algunos tipos de carreteras y algunas configuraciones relacionadas.

El resto de opciones deben ser precedidas por **–netconvert-options**. Son la unión de las que aparecen por defecto y las recomendadas por *SUMO*:

- **–geometry.remove**. Simplifica la red para ahorrar espacio pero no cambia la topología.
- **–roundabouts.guess**. Asegura la selección correcta del derecho de paso de los vehículos en una rotonda.
- **–ramps.guess**. Identifica los carriles de aceleración y deceleración porque en *OpenStreetMap* no suelen estar debidamente etiquetados.
- **–v**. Muestra una salida verbosa. Sin embargo, actualmente no se ve en la consola de información.
- **–junctions.join**. Une dos o más cruces para formar uno sólo ya que a veces la estructura de datos de *OpenStreetMap* separa un cruce (un nodo) en varios. Por ejemplo, cuando dos carreteras paralelas pero separadas por césped llegan a una carretera perpendicular a ellas.
- **–tls.guess-signals**. Asocia una intersección con el semáforo que lo controla cuando la información no esta en el mismo nodo. A veces el semáforo se separa en otro nodo para especificar la posición en la que se encuentra.
- **–tls.join**. Como varias intersecciones pueden depender del mismo semáforo pero *OSM* no puede asociarlas, esta opción intentará unirlas.
- **–output.original-names**. Utiliza los nombres de los elementos de entrada como atributo en los elementos de salida.
- **–junctions.corner-detail**. Genera nodos intermedios en las intersecciones para hacer las curvas más suaves.
- **–output.street-names**. Incluye, si están disponibles, los nombres de las calles en la salida.
- **–tls.default-type,actuated**. Establece que los semáforos cambien en función del tráfico en un momento determinado, en vez de funcionar con periodos estáticos.

Algunas ejecuciones de la aplicación pueden ser una carga muy grande para *osmnx*. En ese caso, aparecerá una notificación en la consola para advertir de que puede ir un poco lenta la interacción con el mapa. Ese mensaje también ofrece la opción de abrir el mapa en *NETEDIT*, que es capaz de gestionar las carreteras de forma más eficiente pero sin mostrar el mapa tras ellas. De igual forma, si no se ha hecho la petición correctamente en modo manual es posible que no puedan mostrarse los resultados. Por ejemplo, en el caso de que falte algún nodo de una carretera no podrá mostrarse ya que los nodos son los que almacenan las coordenadas y *osmnx* lo maneja no mostrando ningún elemento. *NETEDIT* es capaz de ignorar esas carreteras para mostrar el resto, por lo tanto también aparece el mensaje para ofrecer esta posibilidad.

3.5. Tabla de desambiguación

El propósito de la tabla de desambiguación es ayudar a diferenciar grupos de carreteras de una selección previa. Por ejemplo, en el caso de tener seleccionados ambos carriles de una autopista pero sólo querer uno de ellos. Por ahora es posible utilizar la tabla individualmente con cada *request* y existen dos formas diferentes de utilizarla.

3.5.1. Tabla basada en combinación de atributos

El usuario será capaz de seleccionar un conjunto de atributos entre aquellos que aparecen al menos una vez en la selección previa. En ese momento la tabla comprobará las diferentes combinaciones existentes en dicha selección y las mostrará de forma que no haya dos filas iguales. Ahora el usuario podrá hacer doble *click* en una fila o una celda para configurar su petición con los valores seleccionados. Sin embargo, lo más probable es que el usuario quiera ver primero a que hace referencia una fila. Por ello es posible visualizar de antemano lo que el usuario estaría seleccionando con cada una. Esta tabla nos permite ver todos los datos recogidos de forma ordenada y es una opción más rápida de ver las alternativas. El orden de las filas es en base a la cantidad de carreteras que tienen esos atributos, siendo la primera fila la que más carreteras tiene.

Las combinaciones se consiguen recorriendo todos los elementos preseleccionados y comprobando si su combinación de claves-valores ya ha aparecido o no. De esta forma también podemos contar la cantidad de veces que aparece cada una para ordenar las filas. En la figura 3.9 puede verse un ejemplo de los resultados que ofrece esta tabla.

highway	maxspeed	lanes
motorway	120	3
motorway_link	60	1
motorway	120	3
motorway	100	3
motorway_link	60	2

⇒

highway	maxspeed
motorway	120
motorway_link	60
motorway	100

Figura 3.9: Transición de datos preseleccionados a tabla de desambiguación (por combinación) con las etiquetas *highway* y *maxspeed*.

Selección de celdas

Para seleccionar una celda sólo hay que comprobar si la etiqueta existe en esa fila o no (caso en el que veríamos una casilla vacía) para elegir entre un filtro de *no existencia* o un filtro de *igualdad* (véase la figura 3.10, rama de carreteras desconectadas negativo). De esta forma es probable que se seleccionen elementos de otras filas también.

Selección de filas

La selección de una fila simplemente ejecuta una selección de celda para aquellas visibles. De esta forma sí se consigue elegir exclusivamente los elementos que representa

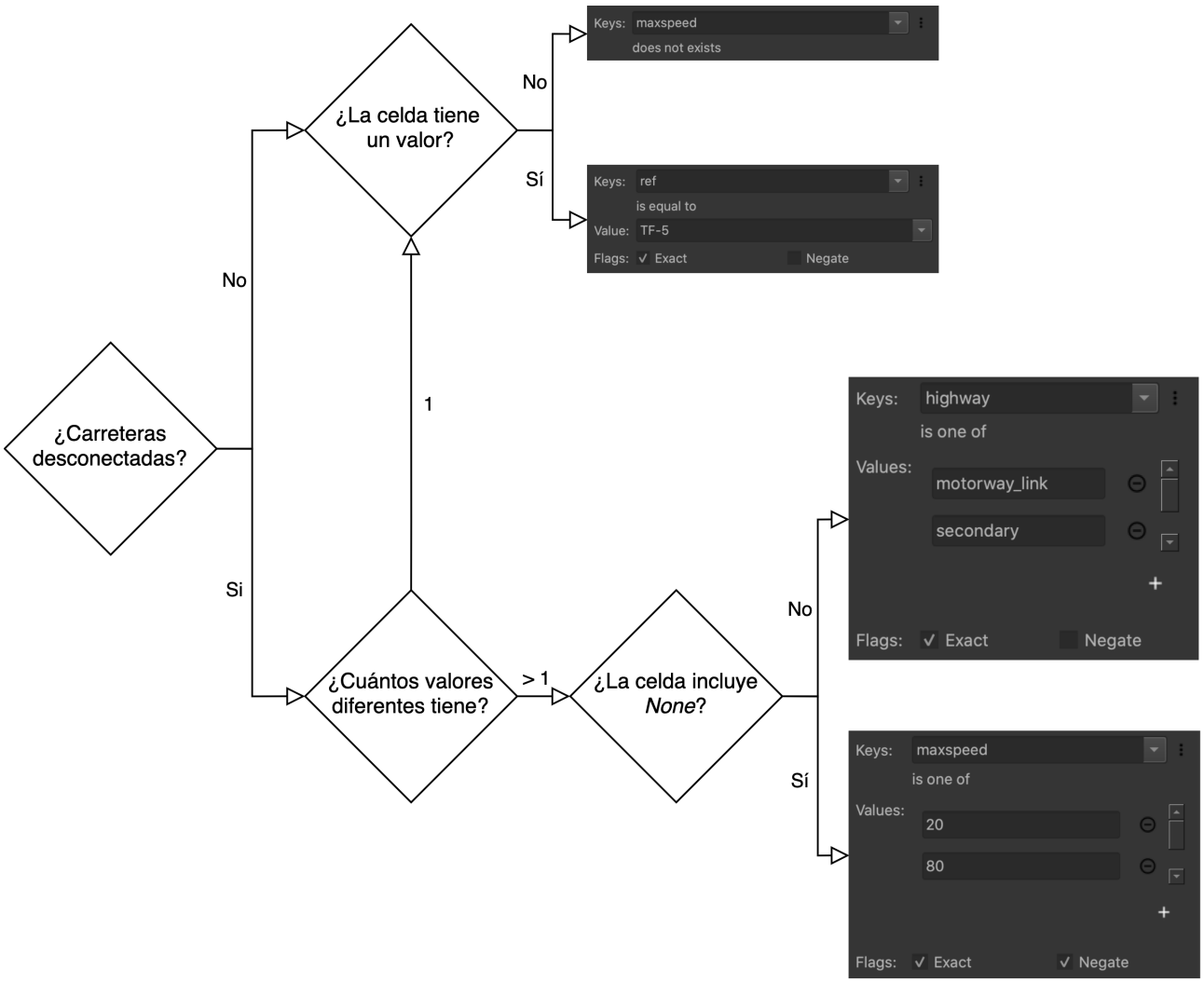

Figura 3.10: Configuración de una *request* a partir de una celda.

esa fila. Sin embargo, hay que tener en cuenta que, cada vez que añadimos un atributo más a la tabla, la cantidad de filas y los elementos a los que hacen referencia cambian.

3.5.2. Tabla basada en conjunto de carreteras desconectadas

La segunda alternativa es sumamente más compleja. En este caso no interesa la interpretación de las carreteras de *OpenStreetMap* sino lo que un usuario podría diferenciar a simple vista. Es decir, cada fila de la tabla hará referencia a un conjunto de carreteras conectadas entre sí pero desconectadas del resto. En este caso las etiquetas podrían no ser suficientes ya que podríamos referirnos a carreteras tan parecidas como dos sentidos de una misma carretera.

Selección de celdas

En cuanto a la selección de valores en una celda ocurrirá exactamente igual que en el primer tipo de tabla en aquellos casos que sólo exista un valor. Pero como ahora las filas no se forman en base a los atributos pueden haber filas con más de un valor separados por comas. Si en ese conjunto de elementos hubiera al menos uno sin esa etiqueta aparecería como *None* entre los valores.

En el caso de que haya varios valores hay que diferenciar si en esa fila hay elementos sin esa etiqueta o no (véase la figura 3.10). Si todos la incluyen se añade un filtro para asegurarnos de que es cualquiera de los valores. Sin embargo, en caso contrario no es posible decir con una única *request* que tenga uno de los valores mencionados o no exista, porque todas las condiciones de una *request* deben cumplirse simultáneamente. Para evitar crear una nueva *request* se accede a todos los valores que tienen el resto de filas para la etiqueta que estamos manipulando y creamos un filtro que niegue su aparición. En el caso de que uno de esos valores también los incluyera la fila seleccionada no serían añadidos porque son valores que si nos interesa que aparezcan. De esta forma no incluimos ninguna *request* adicional y obtenemos el mismo resultado, ya que, como hemos mencionado, las negaciones en los filtros aceptan que no exista la etiqueta (aceptan *None*).

Selección de filas

Al seleccionar una fila será necesario buscar la combinación adecuada de atributos para poder diferenciarla del resto. En primer lugar, hay que diferenciar las carreteras en conjuntos de elementos conectados entre sí y separados del resto. *osmnx* trata las carreteras de *OSM* con la librería *networkx* como un grafo. Esa librería tiene un método, *networkx.weakly_connected_components*, que a partir de un grafo devuelve los *subgrafos* de los que está compuesto. A partir de esa información creamos un diccionario o *map* por cada subgrafo en los que las claves serán los atributos, acompañados por todos los valores que aparecen en ese subgrafo. En el caso de que algún elemento no incluya esa etiqueta se añade en el diccionario como *None*. Por otro lado, también es necesario tener la lista de elementos excluyendo la fila seleccionada. La forma de seleccionar la fila es tratando de eliminar todos los demás elementos. Por eso esta lista sirve de referencia para comprobar los avances de la ejecución.

A partir de este punto llamaremos VNS o *valores no seleccionados* a la unión de los valores que tiene una etiqueta determinada en aquellas filas no seleccionadas por el usuario. Por otro lado, VS o *valores seleccionados* hará referencia a los valores que tienen los elementos de la fila seleccionada para una etiqueta en concreto. Estos valores se construyen a partir del diccionario creado para cada subgrafo.

Una vez hayamos inicializado las variables necesarias se itera en un bucle con cada clave (atributo) de todos los que aparecen al menos una vez en la selección previa (véase el Algoritmo 1). En cada iteración se seleccionan aquellos valores resultado de la diferencia de VNS y VS. Esos valores (a los que llamaremos DV o *diferencia de valores*) corresponden con aquellos que se pueden eliminar sin que haya riesgo de eliminar elementos de la fila seleccionada. Si DV no es vacío significa que ese atributo podría ayudar a diferenciar algunas carreteras si no lo ha hecho otro atributo antes. En la figura 3.11 se puede ver un ejemplo de este proceso, los elementos que quedan seleccionados después de cada iteración y los valores de VNS, VS y DV.

Si DV es exactamente igual a VNS habremos encontrado un atributo que nos permite distinguir completamente las carreteras deseadas ya que significa que no hay valores en común entre VNS y VS, es decir, para esa etiqueta no hay ningún valor en común entre la fila seleccionada y el resto. Después de comprobar en cual de los dos casos nos encontramos se elige el filtro más adecuado para configurar la petición.

En el caso de que los valores seleccionados sean distintos a VNS hay que comprobar previamente si realmente ese filtro ha deseleccionado carreteras, ya que podrían haber

Algoritmo 1: Selección de filas en tabla basada en conjunto de carreteras desconectadas

```
Input: elementos_sin_eliminar = lista de elementos de OSM de las filas no
       seleccionadas que no han sido eliminadas aún
Output: dupla cuyo primer elemento es una lista de filtros y cuyo segundo es una
        lista de identificadores

filtros ← []
ids ← []

for key in todas_las_keys do
    cantidad_de_elementos_sin_eliminar = length(elementos_sin_eliminar)
    DV_key = VNS_key \ VS_key
    if DV_key == VNS_key then
        return ([elegirFiltroAdecuado(DV_key)], [])
    else
        if length(DV_key) > 0 then
            elementos_sin_eliminar=eliminarPorEtiqueta(elementos_sin_eliminar,
            DV_key)
            if length(elementos_sin_eliminar) < cantidad_de_elementos_sin_eliminar
            then
                filtros.add(elegirFiltroAdecuado(DV_key))
                if length(elementos_sin_eliminar) == 0 then
                    return (filtros, ids)
for elemento in elementos_sin_eliminar do
    ids.add(elemento.id)
return (filtros, ids)
```

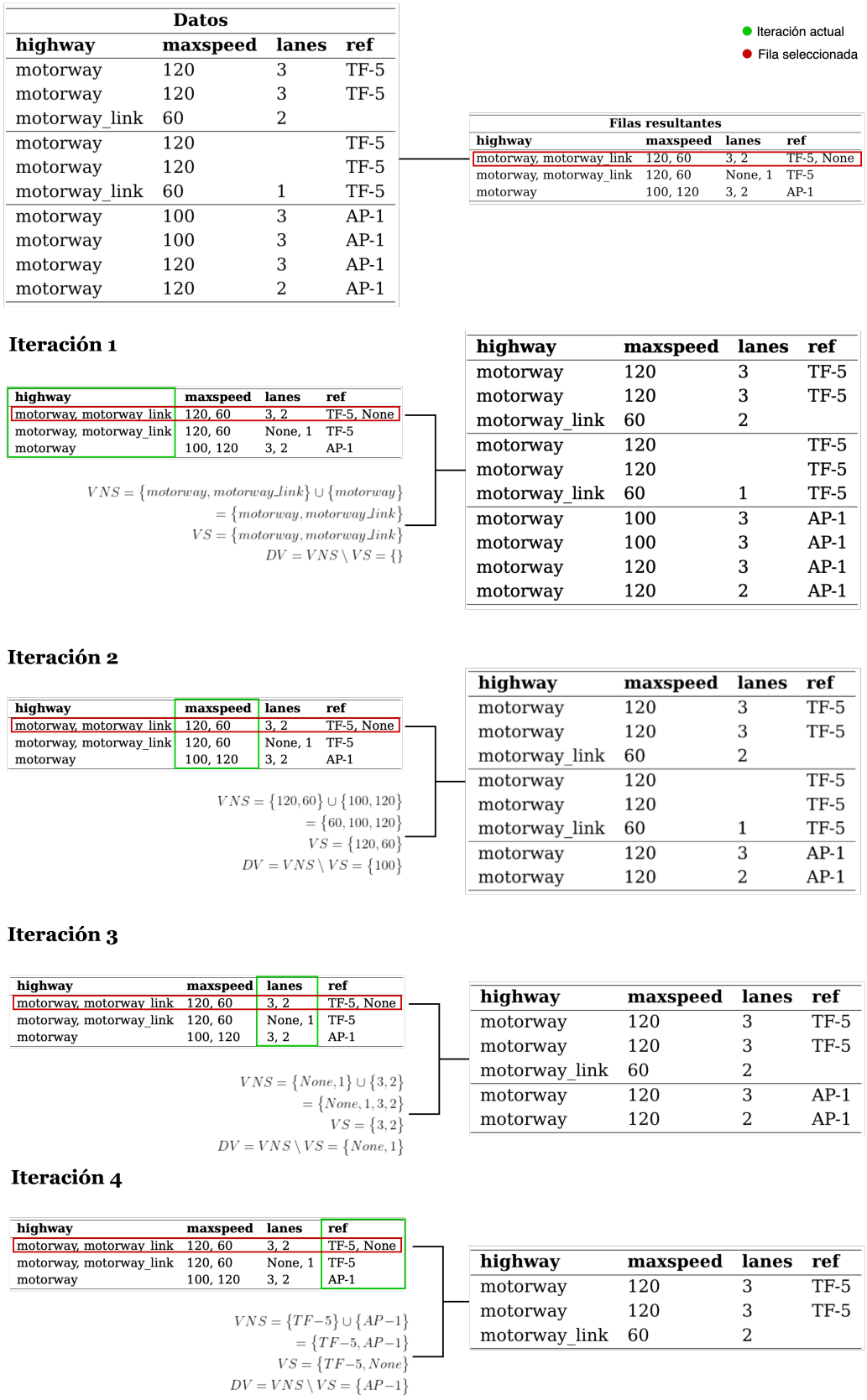

Figura 3.11: Proceso de selección de fila para la tabla de carreteras desconectadas.

sido deseleccionadas ya por filtros anteriores y por lo tanto no sería necesario. Para comprobarlo accedemos a la lista que habíamos creado con los elementos que no queríamos seleccionar y eliminamos aquellos que tengan los valores que aparecen en DV. Si alguno ha sido eliminado entonces nos interesa añadir este nuevo filtro.

La forma de elegir el filtro adecuado teniendo DV es igual para los dos casos mencionados. Sin embargo, el filtro elegido cuando DV es igual a VNS excluirá el resto de filtros establecidos previamente y se convertirá en el resultado del algoritmo. En el otro caso el filtro conseguido se añadirá al resto de filtros.

Si en DV sólo hubiera un valor habría que diferenciar si ese valor es *None* o realmente la etiqueta existe. En el primer caso el filtro confirmaría la existencia de la etiqueta y en el segundo caso confirmaría que esa etiqueta no tiene ese valor. Recordemos que lo que representa DV son los valores que nos ayudarán a **eliminar** carreteras que no nos interesan, por eso negamos que tengan esos valores. En el caso de que haya varios valores se niega que la clave en cuestión pueda tener uno de esos valores y en el caso de que *None* esté entre esos valores será necesario añadir otro filtro para confirmar que esa clave exista. En la figura 3.12 se muestra esquemáticamente esta decisión junto a imágenes de filtros de ejemplo.

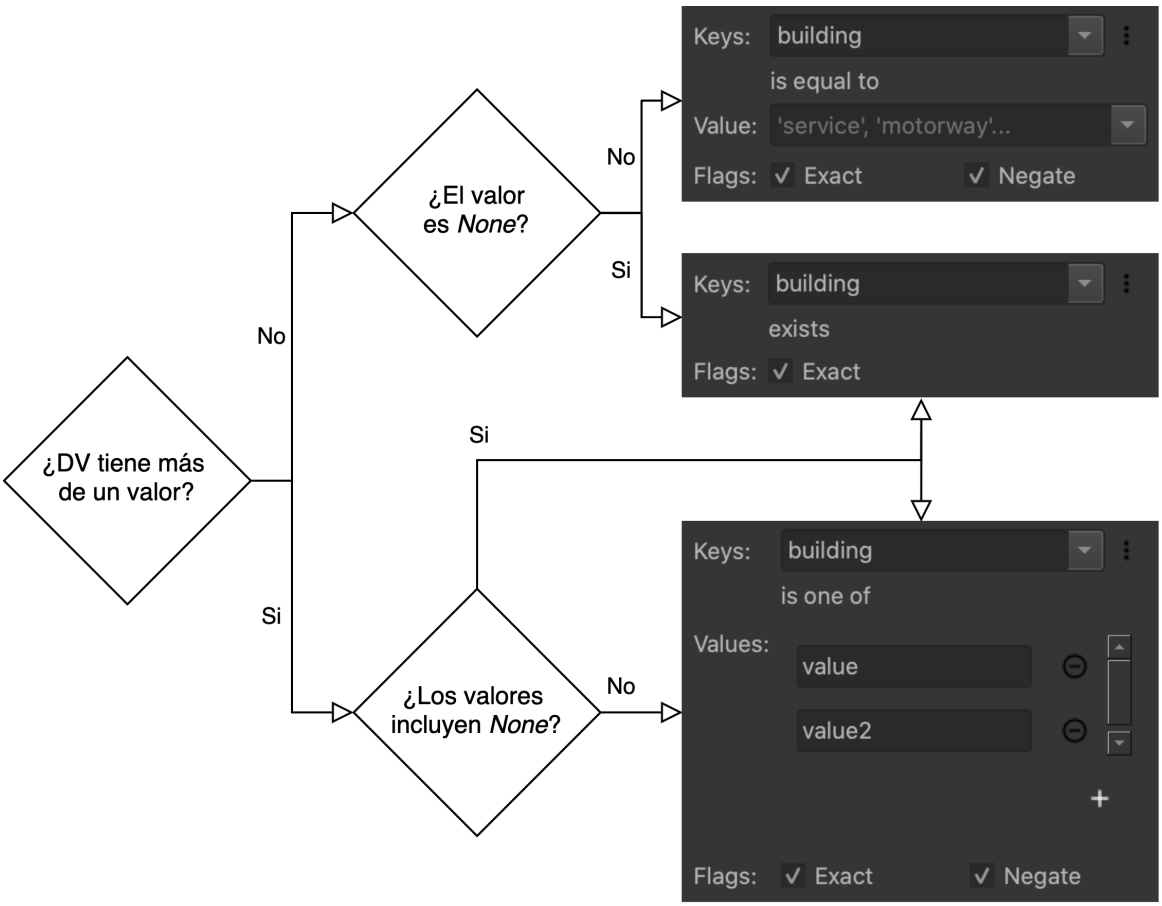

Figura 3.12: Proceso de selección de filtro a partir de DV.

Si al terminar el algoritmo no ha sido posible eliminar todas las carreteras que no nos interesan será necesario eliminarlas con su identificador. Es decir, se creará una nueva *request* en el que se incluirán los identificadores de los elementos que no se han podido eliminar y se creará un conjunto nuevo formado por la diferencia de la *request* original y la nueva. De esta forma nos aseguramos de que se consigue el resultado deseado.

Limitaciones

A pesar de que hay casos en los que es imposible no utilizar los identificadores, también hay casos en los que es posible pero el algoritmo seleccionado no es capaz de encontrarlo.

highway	maxspeed	lanes
motorway	120	3
motorway_link	60	1
motorway	120	3
motorway	120	
motorway	120	
motorway_link	60	2
motorway	100	3
motorway	100	3
motorway	120	2
motorway	120	2

Tabla 3.1: Datos de los que es posible diferenciar dos conjuntos de carreteras pero el algoritmo no encuentra la solución.

Sin embargo, la complejidad necesaria para resolverlo no se vio compensada con los resultados conseguidos. Veamos la tabla 3.1. Teniendo en cuenta el algoritmo, si quisiéramos seleccionar la primera fila la etiqueta *highway* no aportaría ningún resultado porque no hay ningún valor que tenga la segunda fila y la primera no. En la segunda columna podemos eliminar todas carreteras que tengan el valor *100* y finalmente la tercera columna tampoco aporta ninguna información. Por lo tanto, el algoritmo nos proporciona una etiqueta para quitar la mitad de los elementos que no queremos seleccionar y los otros dos tendrían que ser eliminados con identificadores.

Sin embargo, en este mismo ejemplo podríamos evitar utilizar identificadores haciendo combinaciones de valores. No ha sido posible utilizar el número de carriles hasta ahora porque las carreteras que faltan por eliminar tienen dos carriles o *lanes*, al igual que algunas carreteras que deben permanecer seleccionadas. La diferencia es que en la primera fila esos elementos tienen una velocidad máxima (*maxspeed*) de 60, mientras que en la segunda fila tienen una velocidad máxima de 120.

Esta diferencia puede utilizarse para distinguirlas, pero la idea fue desechada para evitar confundir al usuario ya que podría ser necesario crear muchas *requests*. Volvamos al ejemplo anterior. Teníamos un filtro que implicaba que *maxspeed* debía ser 100 y otro que implicaba que *maxspeed* debía ser 120 a la vez que *lanes* debía ser dos. Estos dos filtros no pueden aparecer en la misma *request*. No nos interesan las carreteras con dos carriles con un máximo de velocidad de 100 ni cualquier otra combinación. Esto implica que cada conflicto requiere de una nueva *request* prácticamente igual a la anterior con la diferencia de los nuevos filtros, en vez de un único filtro con los identificadores. Por eso ha sido preferible mantener el uso de identificadores.

3.6. Consola de información

Durante el uso de la aplicación pueden haber ejecuciones que no destaquen y el usuario puede haber pensado que no ha pasado nada aún. En otros casos suceden errores y no es agradable ver como continuamente aparecen ventanas de error. Por eso la consola de información es una parte muy importante de la aplicación. Sirve para notificar al usuario los avances de un proceso y los posibles errores que surjan. Todos los mensajes disponen

de la hora en la que fueron escritos, lo que podría servir para comprobar cuánto tarda una ejecución y concretamente cuál es el apartado más costoso.

La consola es una clase personalizada que hereda de un cuadro de texto normal configurado en modo lectura. Se le añaden los métodos *write* y *flush* para poder ser utilizado como un *stream* en la configuración del paquete *logging* [33] de *Python*. Gracias a este paquete y esta configuración en cualquier archivo del proyecto puede importar este mismo paquete y enviar mensajes de forma que siempre llegue a la consola. En la configuración también se incluyen tres argumentos más, aparte de *stream*.

- El nivel mínimo que recibirá. Los mensajes que se pueden enviar con este paquete tienen un nivel, entre los que aparecen *CRITICAL, ERROR, WARNING, INFO, DEBUG* y *NOTSET*. El orden es importante, ya que si se configura un nivel, todos los mensajes que se encuentre en ese nivel o superior (los que aparecen a la izquierda en la lista) se enviarán a la consola. En este caso el nivel seleccionado es *DEBUG*.

  Los mensajes del nivel *INFO* se muestran en blanco. Se usan para informar del comienzo y final de tareas. Por ejemplo, cuando se ejecuta una petición para seleccionar las carreteras aparecen los mensajes de la figura 3.13.

  Los mensajes del nivel *ERROR* se muestran en rojo. Sirven para avisar de que algo ha impedido la correcta ejecución de la aplicación y, si es posible, la forma de arreglarlo. Por ejemplo, cuando en modo manual se ha cometido un error de sintaxis.

  Por último, los *WARNING* se muestran en amarillo. Hace referencia a los mensajes que informan de que algo no se ha ejecutado como se esperaba, pero aún así ha podido terminar la ejecución. Un ejemplo de este tipo de mensajes es cuando no hay conexión al arrancar la aplicación y no se puede acceder a las claves más usadas con la API de *TagInfo*. La aplicación se puede seguir utilizando pero no dispondremos de esa información.

  El nivel *CRITICAL* realmente no se usa pero sería interpretado como un *ERROR* si se utilizara y el nivel *DEBUG* actualmente sólo se utiliza como señal con el mensaje "LINE" para dibujar una línea horizontal en señal de final de una ejecución.

- El formato del mensaje. Cuando se envía un mensaje desde otro punto del código el paquete formatea ese mensaje con información adicional según la configuración establecida, en este caso: *%(levelname)s %(asctime)s - %(message)s*. Lo que significa que en primer lugar se especifica el nivel del mensaje, seguido de la fecha en la que se envía y terminando con el mensaje en cuestión. Al recibir el mensaje se identifica el nivel para elegir el color y se elimina del texto para mostrarlo adecuadamente.

- El formato de la fecha. Como hemos visto en el mensaje aparece una fecha, y definir el formato requiere un argumento adicional. Sólo nos interesa la hora del mensaje y no el día o el mes, por lo tanto la fecha se muestra de la forma: *%H:%M:%S*. Es decir, hora, minuto y segundos separados por dos puntos.

3.7. Modo manual

La intención de la aplicación es no tener que utilizar nunca *Overpass QL* pero los que conozcan el lenguaje podrían necesitar alguna funcionalidad que no está incluida en la

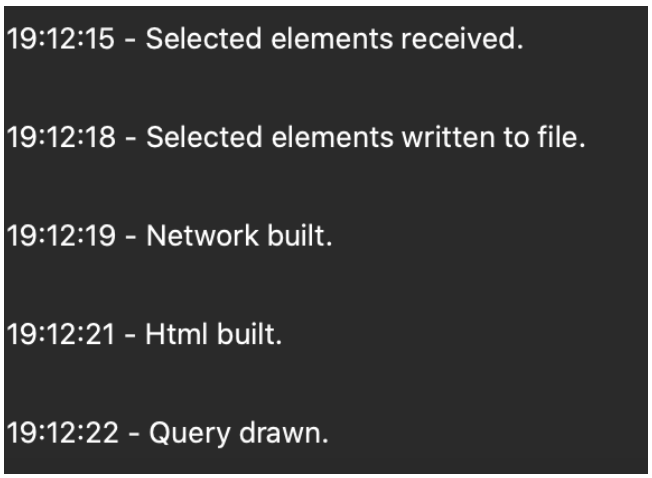

Figura 3.13: Mensajes de la consola cuando se ejecuta correctamente una petición.

interfaz. En este modo desaparecen todas las ventanas y opciones del menú propias del *modo interactivo*, pero se mantienen la opción de dibujar un polígono.

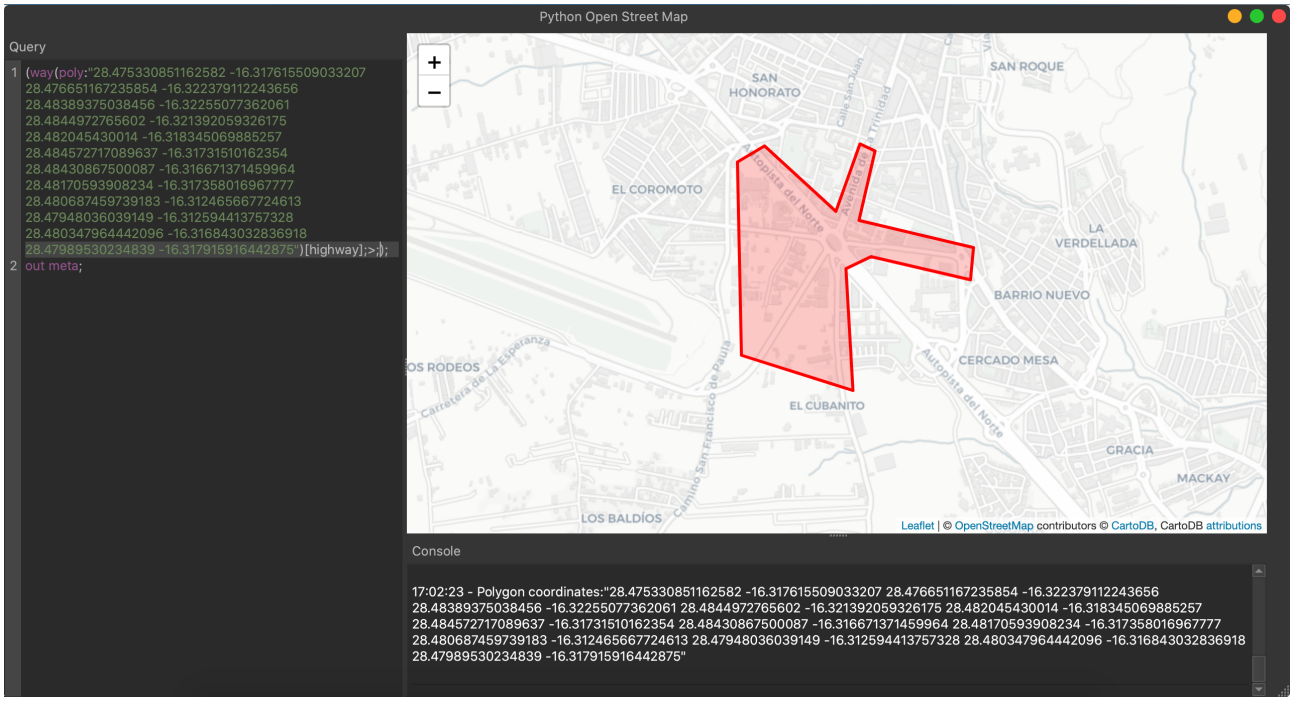

Figura 3.14: Manejo de los polígonos en modo manual.

En este caso será posible dibujar el polígono en todo momento sin necesidad de activarlo y sigue siendo posible usar el atajo de teclado *Ctrl+Z* para eliminar el último punto. En el menú superior hay una pestaña llamada *Manual mode* en la que aparece la opción para eliminar por completo el polígono y para recibir las coordenadas seleccionadas. Esas coordenadas aparecerán en la consola y se copiarán al portapapeles en el formato que utiliza *Overpass QL*. De esta forma el usuario tendría que escribir: **(poly:"pegar aquí las coordenadas")**. El formato de las coordenadas es una lista de al menos tres pares de números separados por espacios compuestos por latitud y longitud (en ese orden y también separados por espacios entre sí). En la figura 3.14 se muestra un ejemplo de este comportamiento.

Ahora la aplicación no gestiona la sintaxis de la petición así que los errores que puedan

aparecer se mostrarán en la consola después de ejecutarse. Dichos errores son recibidos de la API de *Overpass*.

3.8. Apertura y guardado de querys

Uno de los objetivos es reducir el tiempo de modelado de carreteras y en muchos casos será posible realizarlo en un par de horas. Sin embargo, es necesario un periodo de adaptación a la aplicación y siempre puede haber casos más complejos. Por ello, es posible guardar la configuración de la petición (tanto en *Overpass QL* como con la configuración de la interfaz) para seguir trabajando en otro momento.

La interfaz concretamente se guarda con una configuración en JSON similar a la configuración de la caja de herramientas. Se realiza manualmente recorriendo los atributos necesarios para adaptar los datos, como los enumeradores que deben ser asignados acorde a un número (por ejemplo, *comparison*). Una alternativa a este método que fue estudiada era utilizando *pickle* [34], que es capaz de serializar y deserializar objetos grandes rápidamente. Sin embargo, en la documentación advierten que no es seguro y que abrir un archivo malicioso podría ejecutar código desconocido durante el proceso. Por eso este método fue finalmente desechado.

```
{
  "requests":[
    {
      "name":"a",
      "type":"way",
      "filters":[
        {
          "key":"name",
          "comparison":1,
          "value":"Autopista del
             Norte",
          "negated":false,
          "exactValue":true
        }
      ],
      "surrounding":2,
      "aroundRadius":100,
      "polygon":[
        [
          28.222130007158565,
          -16.940917968750004
        ],
        [
          28.22697003891834,
          -16.078491210937504
        ],
        [
          28.565225490654658,
          -16.051025390625004
        ],
        [
          28.507315578441784,
          -16.935424804687504
        ]
      ],
      "ids":[],
      "location":""
    }, {
      "name":"b",
      "type":"way",
      "filters":[
        {
          "key":"ref",
          "comparison":5,
          "value":[
            "GC-1",
            "GC-2"
          ],
          "negated":false,
          "exactValue":true
        }
      ],
      "surrounding":3,
      "aroundRadius":100,
      "polygon":[

      ],
      "ids":[],
      "location":"Gran Canaria"
    }
  ],
  "outputSet":"c",
  "operations":[
    {
      "type":"Union",
      "sets":[
        "a",
        "b"
      ],
      "name":"c"
    }
  ],
  "configuration":{
    "date":"2019-06-11T00:00:00Z"
  }
}
```

El formato JSON ofrece una visión bastante clara de la configuración de la petición. De tal forma que incluso algunos atributos podrían ser manipulados manualmente.

3.9. Plantillas

Hay ciertas búsquedas que son frecuentes en el modelado de carreteras. Por lo tanto, es útil para un usuario no tener que repetir constantemente alguna combinación de atributos o tener que investigar la combinación adecuada para conseguir su propósito.

Las alternativas actuales son:

- **Carreteras**. Hay que tener en cuenta que *OpenStreetMap* trabaja también con edificios, parques y demás estructuras irrelevantes para el modelado de carreteras. Para evitarlas sólo es necesario confirmar que la etiqueta *highway* (tipo de carretera) existe. Como es una condición muy simple esta plantilla no ofrece valores para sus atributos, sino que muestra al usuario las principales etiquetas a tener en cuenta además de *highway*.

  Las etiquetas elegidas son el resultado de aquellas más utilizadas según *TagInfo* para *ways* que no hagan referencia a edificios u otro tipo de elementos que no nos interesan. Se incluyen *name*, el nombre de la carretera; *ref*, el código de referencia (como *TF-5*); *maxspeed*, la velocidad máxima de la carretera; *lanes*, la cantidad de carriles y *oneway* para distinguir las carreteras de doble sentido y sentido único.

- **Carreteras principales**. Selecciona las carreteras con mayor flujo de tráfico, que no estén en construcción, cuyo acceso sea público y que no sean carreteras sin salida.

  Después de algunas pruebas las carreteras con mayor flujo se seleccionaron teniendo en cuenta el tipo de carretera con una igualdad utilizando la expresión regular: `^(motorway|trunk|primary|secondary|residential)(_link)?$`.

  Las carreteras en construcción se eliminaron evitando la existencia de la etiqueta *construction*. Para las carreteras sin salida se comprueba que la etiqueta *noexit* es diferente de *yes*. Por último, para evitar carreteras de acceso restringido, condicionado por el tipo de vehículo o cualquier carretera que no sea de acceso público usamos la etiqueta *access* negada con la expresión regular: `^(y(e([^s]|$|s.)|[^e]|$)|[^y]|$).*`. La expresión regular acepta cualquier cadena de texto diferente a *yes*. Como está negada el resultado es la selección de las carreteras que tienen el valor *yes*. El motivo de hacerlo de esta manera es aceptar también los elementos sin dicha etiqueta, que es el comportamiento cuando negamos un filtro. Actualmente si quisiéramos solicitar que una etiqueta tenga un valor o no exista tendríamos que utilizar dos *requests* y hacer la unión. Con la expresión regular mantenemos la idea de que una plantilla sólo manipule una.

- **Aparcamientos**. Muestra grandes zonas de aparcamientos. En primer lugar comprobamos que la etiqueta *highway* existe. De esta forma nos aseguramos de seleccionar carreteras y no *ways* cerradas que formen el borde de los aparcamientos. En segundo lugar, comprobamos que la etiqueta *service* tenga un valor que contenga *parking* con una igualdad y la *flag Exact* sin seleccionar. De esta forma accedemos a varios valores disponibles como *parking_lane*, *parking_lot*, *parking* o *parking_space*.

- **Peatones**. Muestra calles peatonales. En este caso únicamente hace falta añadir un filtro con la clave *highway* que puede tener los valores *pedestrian*, *footway*, *path*, *cycleway*, *bridleway*, *steps* o *crossing*.

Las plantillas son completamente modificables una vez seleccionadas. Es decir, que el usuario podría decidir tener las carreteras principales pero aceptar aquellas que estén en construcción.

3.10. Manejo del servidor

Los servidores de *Overpass* tienen una capacidad limitada. Si se hicieran demasiadas consultas se podría banear nuestra IP para dar opción a otros usuarios de utilizarlo. Sin embargo, no hay un único servidor disponible. Por lo tanto, en el caso de que uno dé algún tipo de problema como el baneo se informaría debidamente al usuario y se intentaría utilizar uno de los otros servidores hasta encontrar uno que funcione o comprobar que todos han fallado.

Si la respuesta tiene un código de estado **200** quiere decir que la petición ha podido resolverse adecuadamente. Sin embargo, puede que la respuesta implique un error si aparece la etiqueta *remark*. En ese caso simplemente se muestra el mensaje que aparece porque no se ha encontrado especificación de los tipos de problemas que podrían aparecer en este caso.

Si el código de error es **400** significa que la petición tiene un error de sintaxis. Para comprobar de que errores se trata hay que recorrer el *html* que se devuelve y mostrar los que se encuentren en la consola. Este tipo de errores son propios del modo manual. El modo interactivo está pensado para que estos errores no aparezcan, pero es posible que algún caso no se haya tenido en cuenta. En cambio, si el formulario se ha rellenado incorrectamente se mostrarán errores propios de la aplicación más claros para el usuario.

El código de error **429** indica que se han hecho demasiadas peticiones desde una IP recientemente y está bloqueada durante un tiempo para dar oportunidad a otros usuarios de usar el servidor. En este caso antes de mostrar un error por consola se prueban tres servidores diferentes para intentar solucionar el problema.

El código de error **504** implica que el servidor tiene demasiada carga de trabajo y no puede resolver la petición. Lo normal es esperar un tiempo hasta que pueda volver a utilizarse. Pero según la documentación de *Overpass* valores más pequeños para *timeout* y *maxsize* podrían conseguir que se ejecutara la petición. Estas variables indican el tiempo máximo en segundos y la memoria RAM máxima permitidos en el servidor. Valores muy altos conlleva que a veces el servidor las deseche para ejecutar una mayor cantidad de peticiones. En este momento la interfaz no permite cambiarlas pero se proporciona la recomendación por consola en caso de que se quiera utilizar el modo manual para modificarlas.

Cualquier otro código de error será mostrado por consola sin ningún tipo de formateo.

3.11. Manejo de los nombre únicos

Cada **conjunto** (*request* u *operation*) debe tener un nombre único, porque si se crearan dos con el mismo nombre los valores del segundo sobrescribirían los del primero. Para evitar complejidad los nombres se escriben en orden alfabético automáticamente, tanto los de las *requests* como los resultados de las operaciones. Sin embargo, al abrir una petición desde un fichero se mantendrán los nombres guardados en el fichero, para que el usuario vea la interfaz exactamente como era al guardarla. Después de volver a abrirlo si se añade un nuevo conjunto se asignará el siguiente nombre en orden alfabético que no esté ya en uso, porque es posible que el siguiente conjunto deba ser llamado *e* pero este haya sido ya añadido con el archivo.

La gestión de los nombres se realiza en una clase que no debe ser instanciada porque todos sus métodos son de clase. En ella se almacenan los nombres en uso y el nombre

del próximo conjunto añadido. La clase permite acceder al próximo nombre disponible y actualizarlo para la próxima vez que se reclame, confirmar si un nombre está disponible, reservar un nuevo nombre y liberarlo para que pueda ser usado de nuevo en el futuro.

3.12. Intersecciones o cruces

Esta funcionalidad está implementada pero no está incluida en la interfaz debido a que no tiene utilidad aparente para simulaciones de tráfico. Sin embargo, puede ser útil en otras situaciones. *OpenStreetMap* no tiene implementada una forma clara de determinar si un nodo es una intersección o no. Como no hay curvas un nodo no tiene porque ser una intersección. Podría ser un punto en el que la carretera cambia de ángulo. Además no se ha encontrado una etiqueta que muestre esta diferencia. Por ello se ha planteado una solución.

El criterio seguido para diferenciar las carreteras es el número de *ways* que incluyen el nodo en cuestión y en que posición se encuentra ese nodo en el interior de las carreteras. En primer lugar, si un nodo forma parte de tres carreteras podemos decir indudablemente que es una intersección. Si fueran dos podría ocurrir el ejemplo mencionado al principio de este apartado, pero si se incluye un tercer elemento estaríamos ante una bifurcación. En el caso de que dos carreteras contengan un nodo es necesario tener en cuenta la posición de ese nodo en la carretera. Si en ambas carreteras ese nodo es el primero o el último obviamente una carretera es la continuación de la otra. Sin embargo, si en al menos una de las dos carreteras ese nodo no se encuentra en ninguno de los extremos podemos concluir que en ese nodo hay un cruce.

Al comienzo del desarrollo de este proyecto *Overpass* nos permitía acceder a los nodos de una carretera pero no en el orden correcto, por eso este proceso se realizó después de hacer la petición a la API. Recientemente se ha añadido la opción de acceder a los nodos determinando su posición por lo que sería posible realizarlo en la propia petición.

3.13. Historial de ejecuciones

Durante el uso de la aplicación lo más probable es que un usuario tenga que probar unas cuantas ejecuciones antes de conseguir el resultado correcto. También es muy probable que después de hacer algunos cambios se dé cuenta de que no ha seguido el camino correcto. En ese caso podría acceder a su historial de ejecuciones. En ese historial podrá ver una lista ordenada de horas en las que ha ejecutado una petición. Al acceder a una de ellas el mapa se ajustará al resultado de esa petición y la caja de herramientas y el código de *Overpass QL* a la configuración adecuada. Además, es posible guardar ejecuciones en modo manual.

Capítulo 4

Ejemplo de uso

En este apartado vamos a realizar el proceso completo de selección de mapas e introducción de los datos al simulador (*SUMO*). Contamos con que el usuario conoce las carreteras que quiere seleccionar pero no la forma en la que *OSM* las tiene almacenadas.

4.1. Autopista del Norte con entradas y salidas dirección Santa Cruz

4.1.1. Comprobar formato de los datos

El primer paso cuando queremos seleccionar las carreteras es averiguar como podemos hacer referencia a ellas. La mejor forma es seleccionar un pequeño fragmento con un polígono y acceder a la tabla de desambiguación. De esta forma podemos comprobar los valores para cada etiqueta que tenga y elegir aquella que se ajusta más a nuestro objetivo. En la figura 4.1 vemos que una de las etiquetas que se muestra por defecto se llama *name* y su valor es “Autopista del Norte” así que será el que seleccionemos. También podemos ver una etiqueta *ref* cuyo valor es “TF-5” pero al ejecutarlo nos damos cuenta de que alcanza mucho más recorrido que la primera etiqueta y eso no nos interesa.

4.1.2. Probar la configuración con el nombre

Hemos elegido una configuración con el nombre, pero tenemos que comprobar que nos aporta el resultado que nos interesa. Para ello quitamos el polígono que habíamos seleccionado al principio y hacemos doble *click* en la celda con el nombre “Autopista del Norte” para asignarlo automáticamente a nuestra *request*. Al ejecutarlo nos damos cuenta de que hay una autopista en la península con el mismo nombre. En este momento hay dos opciones muy sencillas. En primer lugar, podríamos volver a la tabla de desambiguación y ver las carreteras desconectadas. Ejecutando las distintas filas podríamos darnos cuenta de cual es la que nos interesa. E incluso podríamos averiguarlo simplemente viendo los valores de las columnas. Sin embargo, como son carreteras tan separadas, un polígono podría aportar una solución más directa. En este ejemplo usaremos el polígono para facilitarnos uno de los siguientes pasos.

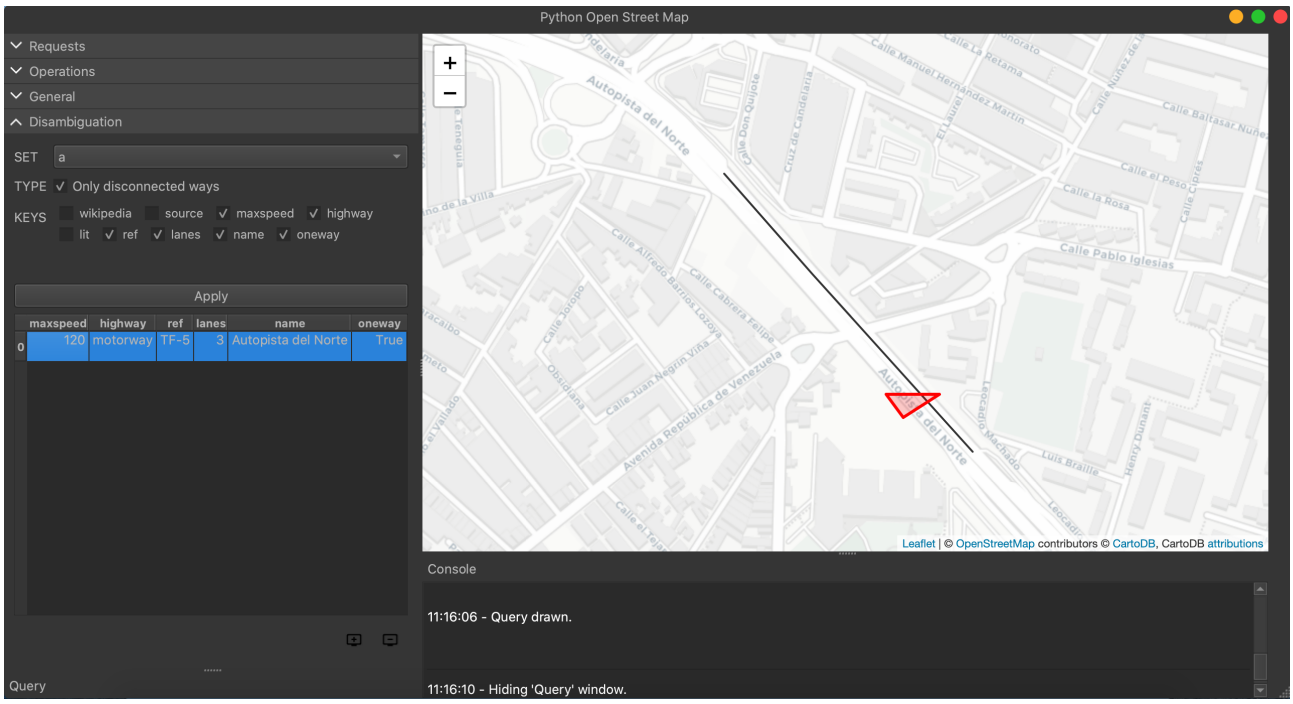

Figura 4.1: Ejemplo de comprobación del formato de los datos.

4.1.3. Diferenciar los dos sentidos de la autopista

Ahora que tenemos seleccionada correctamente la autopista del norte tenemos que seleccionar sólo el sentido en dirección Santa Cruz. En este momento podemos acceder a la tabla de desambiguación, ya que normalmente los distintos sentidos no están conectados entre sí. Sin embargo, en este caso si lo están porque en la tabla sólo aparece una fila con la opción de carreteras desconectadas. Al recorrer la selección actual nos damos cuenta que se unen justo al final de la autopista en el extremo izquierdo. Para el desarrollo de este ejemplo podemos suprimir ese extremo de la autopista, de forma que finalmente aparezcan dos carreteras separadas. Ese es el motivo por el que en el paso anterior elegimos usar el polígono, porque en este caso también lo utilizaremos para suprimir ese fragmento. Utilizar la otra alternativa no hubiera supuesto ningún problema adicional, simplemente una configuración que no aportaría nada.

Ahora finalmente aparecen dos filas en la tabla de desambiguación y mostrando la selección de cada una descubriremos cual es la que nos interesa. Hacemos doble *click* en el número de la fila y tendremos la configuración para seleccionar un sentido de la autopista del norte.

4.1.4. Seleccionar las entradas y salidas de la autopista

Este paso suele ser extremadamente simple pero en este caso tenemos un problema adicional. Las entradas y salidas de una autopista no son más que carreteras contiguas a otra. Esas carreteras se consiguen marcando la opción *Adjacent streets* en el apartado *surrounding* de una *request*. Sin embargo, al seleccionar un sentido de la autopista la tabla de desambiguación no ha sido incapaz de encontrar una combinación de etiquetas para diferenciarlas así que ha tenido que recurrir a los identificadores y una *request* adicional. Esto es natural en distinción de dos sentidos de una carretera ya que comparten la mayoría de etiquetas.

Debido a esto si seleccionáramos la opción *Adjacent streets* en nuestra *request* nos daríamos cuenta de que aparecen algunas entradas y salidas del sentido contrario. Esas entradas y salidas corresponden a las carreteras contiguas que fueron eliminadas en la diferencia de *requests*. Por lo tanto la solución es marcar la misma opción en la segunda *request* para que también sean eliminadas.

Este último paso puede ser algo confuso. Por eso una de las posibles mejoras mencionadas en el apartado sobre líneas futuras es la manipulación de las operaciones. Es decir, en este caso sería posible hallar las carreteras adyacentes a la diferencia de las *requests*. De esta forma no sería necesario tener que manipular todas las *request* implicadas y sería más sencillo entender el resultado obtenido.

4.1.5. Abrir el simulador

Una vez encontrado el resultado podemos abrir el simulador con las carreteras seleccionadas. En el menú superior podemos acceder a *File->Open netedit* y abriremos directamente una aplicación para editar manualmente las carreteras en caso de que fuera necesario algún ajuste más específico. Sin embargo, aunque es posible abrir directamente el simulador, no se ha permitido la opción porque no es útil sin un archivo con los datos del flujo de tráfico.

Si quisiéramos utilizar el simulador accederíamos a la opción *Save->output* para guardar un archivo con el que el simulador pueda modelar las carreteras. Posteriormente sería necesario crear un archivo con los tipos de vehículos y las rutas que deben seguir. En este ejemplo podemos aplicar un flujo aleatorio con un script [35] que ofrece *SUMO*:

```
$ randomTrips.py -n <archivo-de-carreteras> --route-file
<archivo-de-flujo-de-vehiculos>
```

Finalmente creamos un archivo de configuración (*.sumocfg* [36]) en el que se recojan los archivos necesarios para la ejecución. En este ejemplo sólo son necesarios el de modelado de las carreteras y el de flujo de tráfico. Ahora podremos abrir *SUMO* y comenzar la simulación (figura 4.2).

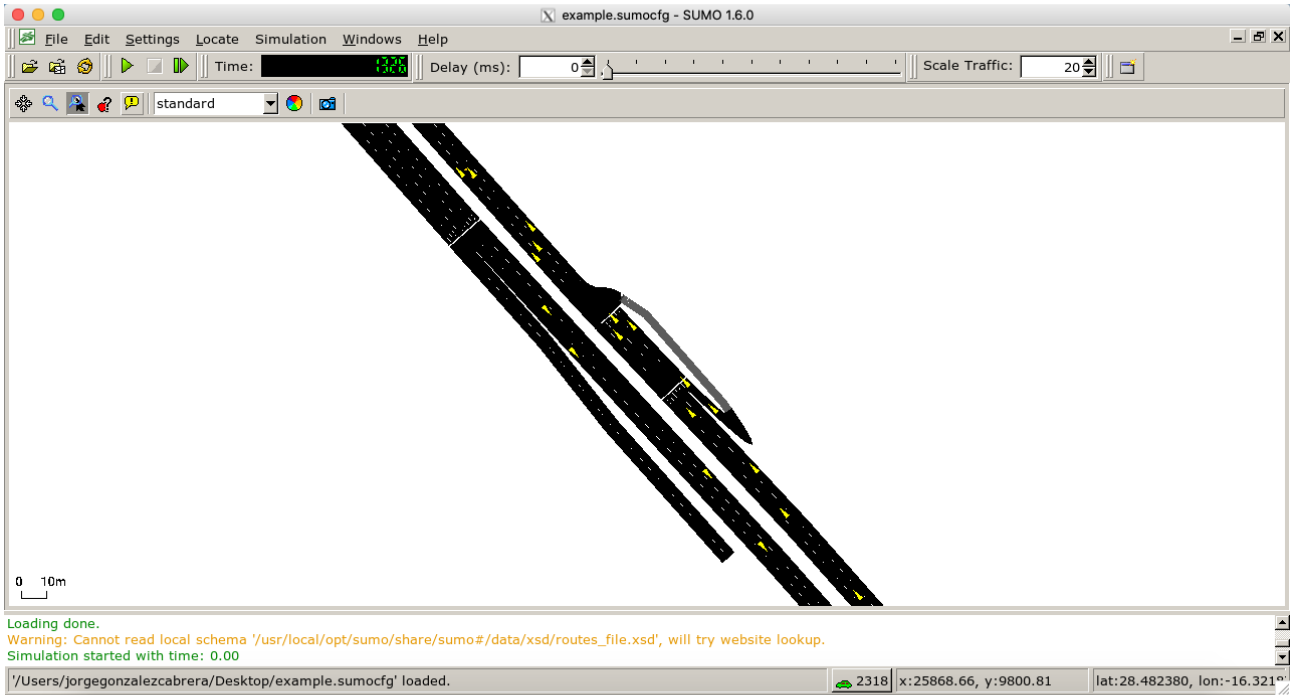

Figura 4.2: Ejemplo de simulación con el modelado de este proyecto.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

Gracias a las funcionalidades realizadas ha sido posible mejorar los tres objetivos propuestos para este proyecto. En primer lugar, el programa facilita el modelado de carreteras en comparación a aquellos que requieren el "dibujo" manual de las carreteras o sólo recurren a un rectángulo para seleccionar carreteras. Como mínimo es igual de potente a *Overpass* porque con el modo manual podemos escribir las peticiones en *Overpass QL* sin restricciones. En el caso de utilizar el modo interactivo será posible utilizar las funcionalidades más útiles y esenciales de *Overpass*. Además, se incluyen funcionalidades adicionales, como el dibujo del polígono en vez de utilizar coordenadas o la búsqueda de un área a partir de un nombre gracias a *Nominatim*.

El segundo objetivo consistía en facilitar el proceso de aprendizaje que conlleva utilizar un lenguaje como *Overpass QL* y por lo que son conocidas otras herramientas como *JOSM*. Para eso tenemos una interfaz que esta pensada para ser lo más reducida posible sin reducir claridad, evitando cualquier habilidad propia de informáticos como las expresiones regulares, que se solucionan gracias a los distintos tipos de filtros. Como el proceso de aprendizaje no puede reducirse por completo también se ha elaborado un manual con un caso de uso como en esta memoria y la explicación de cómo y para qué se utiliza cada componente.

Por último, se han implementado algunas funcionalidades para evitar utilizar herramientas externas. Al principio de este proyecto se accedía a aplicaciones como *Google Maps* para acceder a las coordenadas de los vértices de un polígono y ahora es posible dibujarlo. Para entender para que se usaba cada etiqueta y los valores disponibles se accedía a la página de *TagInfo*, pero ahora se hace uso de su *API* para mostrar dicha información al usuario. También es necesario saber como se almacenan los datos para poder crear los filtros adecuadamente. Antes se accedía a la página de *OpenStreetMap* que permite acceder a la información de un elemento haciendo *click* en el mapa cerca de él. Ahora la tabla de desambiguación nos permite hacer eso como pudo verse en el apartado anterior y además permite ver los datos de forma organizada y distinguir conjuntos de carreteras de forma automática, lo que hemos visto que puede ser complejo.

Por lo tanto, ha sido posible disminuir las dificultades existentes en otras herramientas. Sin embargo, siempre será necesario un periodo de aprendizaje tanto de la aplicación como del funcionamiento de los mapas con *OpenStreetMap*.

5.2. Líneas futuras

La aplicación planteada ha logrado mejorar muchas de las limitaciones que presentan otras herramientas. Sin embargo, la herramienta tiene potencial para incorporar muchas nuevas funcionalidades. En este apartado se mencionan las más importantes:

- Modificar las operaciones. En este momento los conjuntos resultado de las operaciones sólo puede utilizarse en otras operaciones. Pero, como vimos en el ejemplo de uso, a veces es necesario (o al menos más sencillo) hacer las modificaciones propias de una *request*. Una forma de solucionar este problema es eliminar la ventana de operaciones y que las *request* tengan la opción de seleccionar elementos a partir de una operación. Si sólo se quisiera realizar una operación y no modificarla bastaría con no rellenar ningún otro campo. Por último, el *set* de salida se situará en la ventana *General*.
- Cancelación de ejecución. Hay algunas peticiones que pueden durar mucho tiempo. Ahora mismo el usuario no es capaz de hacer nada si la aplicación está en ejecución y si cometiera un error como solicitar todas las carreteras del mundo tendría que reiniciar la aplicación. Con el uso de hilos podríamos solucionar ambos problemas.
- Usar una *cache* para no repetir ejecuciones que ya han sido hechas.
- Mostrar los datos recogidos. Ahora mismo la tabla nos da una aproximación que en la mayoría de casos será suficiente. Pero poder ver los datos individualmente, tanto en una tabla como en el mapa, evitaría tener que usar *NETEDIT* para terminar de eliminar/añadir carreteras. Si los filtros de las *request* o la tabla de desambiguación no fueran suficiente se podría acceder a los datos para eliminar o añadir una carretera por su identificador.
- Conectar conjuntos de carreteras por el camino más corto. Incluso se podría tener en cuenta la velocidad máxima de las carreteras para incluir el camino más rápido.
- Ampliar el espectro de las funcionalidades de *Overpass* que se están utilizando. Hay funcionalidades como bucles, condicionales, máximos y mínimos que no se están utilizando actualmente. Podrían ser útiles para crear nuevas funcionalidades o hacer un mapeo directo en la interfaz.
- Modificación de *osmnx*. Como hemos visto una carretera no puede mostrarse sin sus nodos porque estos son los que tienen las coordenadas. En general si esto sucede en otras herramientas se muestran las carreteras de las que si se tienen todos los nodos, pero *osmnx* lanza una excepción. Además, al hacer *click* en una carretera se puede configurar para que muestre un atributo, pero sería más útil si los mostrara todos.
- Añadir una opción en la selección de los polígonos para evitar aquellas carreteras con alguna parte en el exterior de los límites. Actualmente se seleccionan las carreteras con al menos una parte en el interior por lo que a veces se pueden ver carreteras fuera de la zona deseada.
- Recordar que la petición actual es el resultado de un archivo abierto o de uno que acaba de ser guardado, para que al guardarlo de nuevo no requiera la introducción del nombre del archivo sino que sobrescriba el anterior.

- Seleccionar las carreteras en el interior de un rectángulo. Aunque esta opción también puede ejecutarse con un polígono la opción de un rectángulo puede ser suficiente y resultará en un tiempo de ejecución más corto.
- Añadir un filtro que compruebe que una etiqueta no existe o tiene un valor específico. Durante esta memoria se han alcanzado casos en las que esta comparación hubiera aportado soluciones más claras. Sin embargo, actualmente sólo se puede alcanzar este resultado con una operación entre *requests*.
- Personalizar la transformación de datos a *SUMO*. Aunque ya existe una configuración por defecto es posible que algunos usuarios quieran incluir tranvías u otro tipo de vehículos. Puede que otros estén interesados en usar semáforos con periodos fijos. Son opciones disponibles con las herramientas de *SUMO* que podrían especificarse durante la ejecución de la aplicación.
- Crear un ejecutable para poder utilizar la aplicación en entornos sin los paquetes necesarios.

Capítulo 6

Summary and Conclusions

The implemented functionalities have contributed to the achievement of the objectives of this project. In the first place, this program eases the road modeling in comparison with others that make it by drawing or selecting the roads inside a rectangle. The developed application is at least as powerful as *Overpass* because the manual mode lets the user writing querys with *Overpass QL*. If the user chooses the interactive mode, he/she will be able to access the most useful and necessary functionalities of *Overpass* in a simpler way. Moreover, the user interface add several actions, like drawing a polygon instead of writing the coordinates or looking for an area by its name using *Nominatim*.

The second objective consists in reducing the learning process which entails using a language like *Overpass QL* or other tools known for their complexity like *JOSM*. The user interface is made as simple as possible but keeping all the useful functions. The tools only known by computer engineers are avoided. For example, despite letting the user work with regular expressions, there are several types of filters to make it easier. As the learning process cannot be completely reduced, there is a manual in the wiki of the repository with an example like the one in this document and the explanation of the reason and the behavior of each component.

Finally there are some functionalities implemented to avoid using external tools. At the beginning of this project it was necessary to use applications like *Google Maps* in order to get the coordinates of the vertices of a polygon, but now it can be drawn. If a user wanted to understand the meaning of a tag and the values it could have, he/she had to go to *TagInfo* website, but now this application gets this information from the *TagInfo* API and shows it to the user. Moreover, it is important to know the tags a *way* have to set the filters up correctly. The *OpenStreetMap* website let the user know these tags by clicking on the point of the map where the element is. In this project the disambiguation table let us make something similar as we saw in the last chapter. Additionally this table shows the data in an organized way and it is able to differentiate several set of roads, which can be complicated, as we have seen.

In conclusion, this application has been able to reduce the difficulties that other tools impose. However, it will always be necessary a learning process to understand how the application works and *OSM*'s management of roads.

Capítulo 7

Presupuesto

Debido a la utilización exclusiva de herramientas de código abierto el único gasto ha sido las horas de trabajo, que han sido valoradas en 30 €/h, incluyendo la amortización de los equipos informáticos utilizados así como los gastos por suministros.

Tarea	**Horas**	**Precio (€)**
Estudio de las herramientas	100	3.000
Creación de interfaz	200	6.000
Aplicación de la funcionalidad a la interfaz	350	10.500
Manual de uso	50	1.500
Total	**700**	**21.000**

Apéndice A

A.1. Repositorio

Repositorio en GitHub con el código fuente del proyecto : `https://github.com/alu0101038490/TrafficSimulator/` .

A.2. Manual de uso en inglés

Manual de uso realizada en la wiki del repositorio :`https://github.com/alu0101038490/TrafficSimulator/wiki` .

Bibliografía

[1] DGT. Estadísticas de vehículos en circulación. http://www.dgt.es/es/seguridad-vial/estadisticas-e-indicadores/parque-vehiculos/tablas-estadisticas/, 2019. (accedido el 25/06/2020).

[2] Alejandro Lorenzo Dávila. Simulación y gestión del tráfico en una autopista en situaciones de congestión. pages 6–9, 09 2019.

[3] Página oficial de Overpass. http://overpass-api.de/. (accedido el 25/06/2020).

[4] Documentación de Overpass QL. https://wiki.openstreetmap.org/wiki/ES:API_de_Overpass/Overpass_QL. (accedido el 25/06/2020).

[5] Página web de Overpass Turbo. https://overpass-turbo.eu/. (accedido el 25/06/2020).

[6] Documentación de osmWebWizard. https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html. (accedido el 25/06/2020).

[7] Página oficial de Java OpenStreetMap. https://josm.openstreetmap.de/. (accedido el 25/06/2020).

[8] Wiki de OpenStreetMap. https://wiki.openstreetmap.org/wiki/Main_Page. (accedido el 25/06/2020).

[9] Página oficial de Google Maps. https://www.google.es/maps/preview. (accedido el 25/06/2020).

[10] Página oficial de TagInfo y documentación de la API. https://taginfo.openstreetmap.org/. (accedido el 25/06/2020).

[11] Página de OpenStreetMap para acceder a la información de las carreteras. https://www.openstreetmap.org/#map=6/40.007/-2.488. (accedido el 25/06/2020).

[12] Laurence R. Rilett and Josias Zietsman. An overview of the transims micro-simulation model: application possiblities for south africa. In *20th South African Transport Conference*, pages 16–20, 08 2001.

[13] Simulador MATSim. https://www.matsim.org/. (accedido el 25/06/2020).

[14] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[15] Visualizador de mapas Mapnik. https://mapnik.org/. (accedido el 25/06/2020).

[16] Paquete OSMnx. https://osmnx.readthedocs.io/en/stable/. (accedido el 25/06/2020).

[17] Paquete matplotlib.

[18] Paquete Folium. https://python-visualization.github.io/folium/. (accedido el 25/06/2020).

[19] Librería de javascript Leaflet. https://leafletjs.com/. (accedido el 25/06/2020).

[20] Lenguaje de programación Python. https://www.python.org/. (accedido el 25/06/2020).

[21] Sistema de control de versiones Git. https://git-scm.com/. (accedido el 25/06/2020).

[22] Explicación del concepto IDE. https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado. (accedido el 25/06/2020).

[23] Página oficial de JetBrains. https://www.jetbrains.com/. (accedido el 25/06/2020).

[24] Acceso a la información y descarga de PyCharm. https://www.jetbrains.com/es-es/pycharm/. (accedido el 25/06/2020).

[25] Paquete NetworkX. https://networkx.github.io/. (accedido el 25/06/2020).

[26] Documentación de NETEDIT. https://sumo.dlr.de/docs/NETEDIT.html. (accedido el 25/06/2020).

[27] Página oficial de Nominatim. https://nominatim.org/. (accedido el 25/06/2020).

[28] Página oficial de QT. https://www.qt.io/. (accedido el 25/06/2020).

[29] Documentación de PyQT. https://doc.qt.io/qtforpython/. (accedido el 25/06/2020).

[30] Repositorio de qtmodern. https://github.com/gmarull/qtmodern. (accedido el 25/06/2020).

[31] Documentación sobre los elementos en OSM. https://wiki.openstreetmap.org/wiki/Elements. (accedido el 25/06/2020).

[32] Documentación del script osmBuild.py. https://sumo.dlr.de/docs/Networks/Import/OpenStreetMap.html. (accedido el 25/06/2020).

[33] Paquete logging. https://docs.python.org/3/library/logging.html. (accedido el 25/06/2020).

[34] Paquete pickle. https://docs.python.org/3/library/pickle.html. (accedido el 25/06/2020).

[35] Documentación del script randomTrips.py. https://sumo.dlr.de/docs/Tools/Trip.html#randomtripspy. (accedido el 25/06/2020).

[36] Ejemplo de un archivo .sumocfg. https://sumo.dlr.de/docs/Tutorials/Hello_Sumo.html. (accedido el 25/06/2020).