Universidad de Valladolid

E.T.S.I.T

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías Específicas de Telecomunicación

Simulación de tráfico en ciudades inteligentes con SUMO

Autor:

D. Tomás de francisco Aparicio

Tutor:

Dr. D. Juan Carlos Aguado Manzano

Valladolid, Agosto de 2018

Universidad de Valladolid

Trabajo Fin de Grado

Título:	Simulación de tráfico en ciudades inteligentes con SUMO
Autor:	D. Tomás De Francisco Aparicio
Tutor:	Dr. D. Juan Carlos Aguado Manzano
Departamento:	Teoría de la Señal y Comunicaciones e Ingeniería Telemática

Tribunal

Presidente:	Ignacio de Miguel Jiménez
Vocal:	Ramón Durán Barroso
Secretario:	Juan Carlos Aguado Manzano
Suplente 1:	Noemí Merayo Álvarez
Suplente 2:	Juan Blas Prieto

Fecha:	Septiembre 2018
Calificación:	

Universidad de Valladolid

Resumen

Las Smart Cities son el resultado de la necesidad cada vez más imperiosa de orientar nuestra vida hacia la sostenibilidad. Así, estas ciudades se sirven de infraestructuras, innovación y tecnología para disminuir el consumo energético y reducir las emisiones de CO_2. En este Trabajo Fin de Grado se demostrará el uso de la herramienta SUMO para simular el tráfico en ciudades, y se propondrán métodos para implementar simulaciones de políticas de control de semáforos mediante la información obtenida de aros inductivos.

Palabras clave:

SUMO, Smart City, simulación, semáforos, políticas de control de semáforos.

Abstract

Keywords:

SUMO, Smart City, Simulation, Traffic Lights, Traffic Lights control policies.

Universidad de Valladolid

Agradecimientos

Me gustaría agraceder a mi tutor del Trabajo de Fin de Grado Juan Carlos Aguado Manzano por toda la ayuda que me ha prestado para la realización y revisión de todo el proyecto así como el constante interés que ha mostrado durante la realización de este

Índice

Índice

Índice

1

Introducción

1.1 Motivación del proyecto final de carrera

Desde finales del siglo XIX, el uso de los semáforos se empezó a popularizar como mecanismo para regular el tráfico en las grandes ciudades. Dicho mecanismo de regulación del tráfico ha perdurado hasta nuestros días sin apenas modificaciones sustanciales de su comportamiento. Actualmente los semáforos utilizados, en su mayoría, tienen unos ciclos fijos de funcionamiento en que abren y cierran dicho semáforo en unos intervalos de tiempo preprogramados. Sin embargo todos los días se producen grandes atascos en muchas ciudades del mundo por diferentes motivos, como la inmensa afluencia de automóviles en hora punta, obras u otros tipos de eventos. Cuando se producen congestiones en una única dirección, este tipo de política de control de los semáforos mediante tiempos de ciclo estáticos no es eficiente ya que en muchos casos es un mecanismo que no solo no disminuye la congestión sino que ocasionalmente la agrava. Sin embargo en pleno siglo XXI y con la sensorización de las ciudades y el surgimiento de las Smart Cities, se pueden obtener datos en tiempo real con sensores implantados en este nuevo tipo de ciudades. Dicha información podría utilizarse para regular el tiempo de los ciclos de los semáforos de manera dinámica y, de esta manera, introducir un conjunto de mejoras para el total de los ciudadanos como son reducir el tiempo medio de viaje, aliviar la congestión que pueda causarse o que este causada en una determinada dirección, reducir la emisión de gases contamientes o el ruido[1], abriendo de esta forma un mundo de nuevas posibilidades al ya más que centenario sistema de control de tráfico en las ciudades.

1.2 Objetivos

El propósito de este proyecto es simular políticas de control de semáforos mediante la herramienta SUMO usando la información obtenida de aros inductivos en Smart Cities. Este objetivo requiere lograr una serie de objetivos secundarios:

- Generar mapas automáticos o cargar mapas de ciudades reales con todos los elementos correspondientes tales como semáforos, aros inductivos, etc.

- Elegir un documento o documentos de referencia cuyos resultados sirvan de comparación con nuestro modelo.

- Automatizar la extracción de datos de un mapa e individualizar el control de cada conjunto de semáforos en cada cruce del mapa para que en trabajos posteriores sólo sea necesario programar la política concreta de control de los semáforos que se desea probar.

- Programar un algoritmo de control de semáforos que demuestre que el sistema desarrollado se puede utilizar para la planificación de ciudades, comparar y estudiar resultados.

1.3 Fases y métodos

La realización del proyecto comenzó con una **primera fase** de toma de contacto con los materiales y con la documentación necesaria para su realización. Tras evaluar diferentes tipos de simuladores para llevar a cabo el proyecto, se eligió SUMO. Además, se exploró la bibliografía para conocer el estado del arte, y se seleccionó un escenario cuyos resultados se utilizarían de base para comparar con los nuestros.

La **segunda fase** consistió en la instalación del simulador y las librerías necesarias para continuar con el proyecto.

La **tercera fase** fue familiarizarse con el simulador, así como la estructura de los diferentes tipos de ficheros usada para llevar a cabo las simulaciones. También hubo una parte importante de documentación sobre las librerías necesarias para la realización del proyecto y una familiarización con el lenguaje de programación Python, necesario para interactuar con dichas librerías.

En la **cuarta fase** se empezó a hacer pruebas con las librerías mencionadas anteriormente, comprobar los datos obtenidos y compararlos con los resultados dados en la bibliografía seleccionada, y a esbozar una primera idea de la aproximación más adecuada para tratar dichos datos. Otro aspecto importante de esta fase fue establecer el formato en el que se almacenarían y se usarían los datos, buscando un enfoque eficiente.

Quinta fase, implementación del código que automatizará el tratamiento de los datos del mapa y que desacopla el tratamiento de dicho mapa de la política de control de los semáforos. Resolución de problemas derivados del tratamiento de los datos obtenidos durante las pruebas para la implementación de este código.

Sexta y última fase, simulación de varios escenarios con diferente grado de complejidad en el que se demuestra la utilidad del código desarrollado, y análisis y comparación de resultados obtenidos con el escenario inicialmente expuesto.

1.4 Material utilizado

En el desarrollo de este Trabajo Fin de Grado se ha utilizado el siguiente material:

- Bibliografia
- Ordenador
- Simulador de movilidad SUMO
- Herramienta Netedit provista por SUMO
- Herramienta Netconvert provista por SUMO
- Herramienta Traffic Modeller
- Entorno de programación Pycharm
- Editor de texto Notepadqq
- Librerias TraCI y Sumolib provistas por SUMO
- Librerias matplotlib, numpy y shapely
- Matlab
- Herramienta diagramas de flujo draw.io

2

Estado del Arte

2.1 Introducción

En las últimas décadas se ha producido una mayor concienciación sobre consumo de energía, las consecuencias de su producción en el medio ambiente y un incremento del coste de la misma, por tanto, la población se preocupa cada vez más de reducir el consumo de energía eléctrica o de combustibles fósiles. No es el único reto que afecta a las modernas urbes, aumentar la calidad de vida en ellas en general es una preocupación que ha ido calando en las instituciones y gobiernos locales y nacionales lo que promovió, en parte, el surgimiento de las Smart Cities con el fin de mejorar dicha calidad de vida de sus habitantes haciendo uso de tecnologías de la información ya sea por ejemplo reduciendo el ruido de las ciudades mediante el estudio de mapas de ruido, reduciendo el tiempo medio de viaje en los desplazamientos urbanos a través de información del tráfico, estudiando la composición de los gases en el entorno urbano para intentar mejorar la calidad del aire, hacer recomendaciones concretas sobre el uso de transporte privado y realización de actividades al aire libre, y por tanto en general mejorando la salud de sus habitantes.

2.2 Smart Cities

Como se ha mencionado anteriormente a principios de siglo empezó el surgimiento de las Smart Cities. El traslado de la población del ámbito rural al ámbito urbano a lo largo del todo el siglo XX y lo que llevamos del presente provocó que aproximadamente más la mitad de la población mundial está concentrada en las ciudades. Como causa de esto se generaron múltiples problemas logísticos en las ciudades: gestión de residuos, escasez de recursos, problemas en la calidad del aire que afecta a la salud de los habitantes, deterioro de las infraestructuras, problemas de tráfico de vehículos derivados en atascos, etc. Dado este crecimiento vertiginoso de la población concentrado en la ciudad, se requería hacer frente a los problemas surgidos mejorando sustancialmente la sostenibilidad ambiental y creando

ambientes sociales competitivos (en el sentido de que son más prósperos por su mayor calidad de vida). Cuando a estos dos factores de sostenibilidad se le añade una fuerte inversión en TIC (Tecnologías de la Información y la Comunicación) para la resolución de problemas es cuando se puede hablar de que una ciudad se ha convertido en Smart City.[2] Desde este punto de vista, la definición de Smart City abarca una serie de conceptos y situaciones muy amplios (ver por ejemplo la Ámbitos de desarrollo de las Smart Cities), ya que el punto de partida como hemos visto es poco restrictivo. Algunas definiciones de Smart City que encajan en la versión más amplia anterior y que han sido utilizadas son la siguientes [3]:

- Una ciudad creada pensando en el futuro de la economía, los habitantes, la movilidad, el medio ambiente y habitabilidad.

- Una ciudad que monitoriza el estado de sus infraestructuras críticas incluyendo carreteras, puentes, túneles, vías férreas, metro aeropuertos y puertos marítimos.

- Una ciudad que conecta la infraestructura física, de tecnologías de la información, social y empresarial para lograr una inteligencia colectiva de la ciudad.

- Una ciudad que se esfuerza por hacerse más inteligente, más eficiente, sostenible, equitativa y habitable.

- Una ciudad que combina tecnologías de la información y de comunicación, web, de diseño y aspectos organizativos para agilizar los trámites burocráticos y ayudar a identificar nuevas soluciones innovadoras para la compleja gestión de la ciudad para lograr una mayor sostenibilidad y habitabilidad.

- Una ciudad que usa tecnologías de computación inteligentes para hacer las infraestructuras críticas y servicios de la ciudad tales como administración de la ciudad, educación, salud, seguridad pública, transporte e instalaciones más inteligentes interconectadas y eficientes.

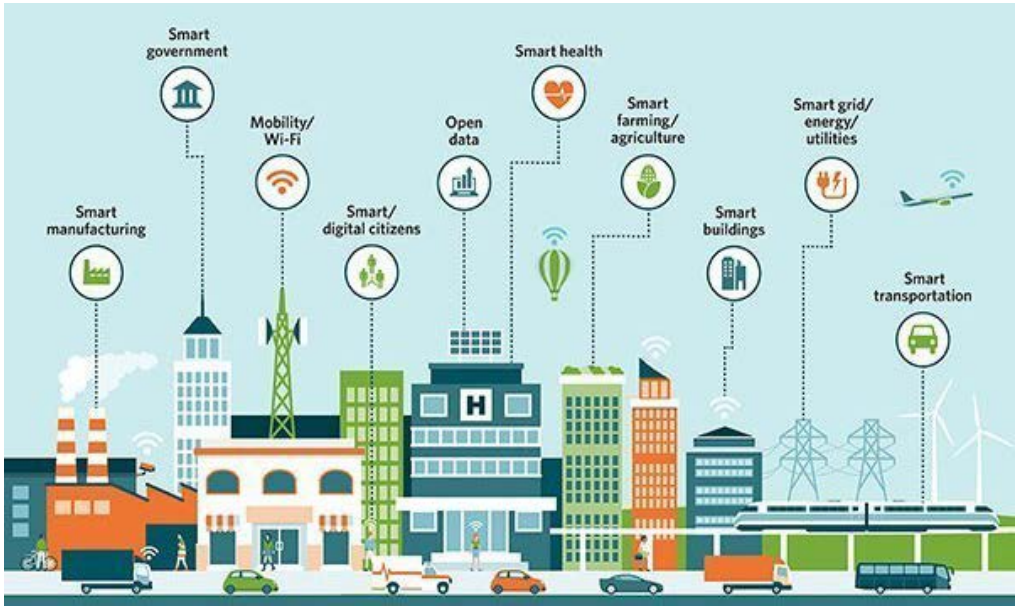

Ilustración 1. Ámbitos de desarrollo de las Smart Cities

Un ejemplo de Smart City en España es Santander, en ella se han desplegado miles de dispositivos que permiten conocer en tiempo real el estado de múltiples servicios tales como agua, residuos, movilidad, alumbrado, etc. Estos datos recogidos por los dispositivos son usados por el ayuntamiento de la ciudad para adecuar la gestión municipal a las necesidades de sus habitantes logrando una mayor eficiencia en el funcionamiento de los servicios públicos. Los ciudadanos pueden acceder a los datos recogidos por los dispositivos mediante aplicaciones móviles o bien mediante paneles informativos desplegados en la ciudad, lo que permite conocer en tiempo real, entre otras cosas, el estado del tráfico.[4]

Otro ejemplo de Smart City es La Coruña [5] en la que se ha desarrollado un proyecto centrado en cuatro áreas diferentes:

- El agua: se obtiene de manera remota los datos de los contadores de agua y gas de la ciudad, se monitoriza las redes de abastecimiento y saneamiento de agua, así como la calidad del agua y el sistema de riego está automatizado para parques y jardines.

- Energía: se monitoriza el consumo de agua y gas en los edificios públicos, sistema de automatización de los procesos de la estación de tratamientos de agua potable.

- Medio Ambiente: monitorización del aire y ruido.

- Movilidad: sistema de optimización del tráfico en tiempo real mediante la frecuencia del transporte urbano, control de aparcamiento en zonas especiales y una aplicación que permite consultar el estado de las plazas de aparcamiento.

Estos proyectos constituyen un ejemplo de cómo las Smart Cities se van conviertiendo en una realidad creciente, y el máximo interés que existe en desarrollar aplicaciones que realmente permitan desarrollar el máximo potencial que se espera de ellas.

2.3 Tipos de sensores

La congestión causada por el tráfico es uno de los grandes problemas de las ciudades en el siglo XXI, parte de estos problemas son causados por los intervalos de tiempo de las fases de los semáforos. Esto radica en que los semáforos no utilizan información en tiempo real, sino que estos tienen unos intervalos de tiempo fijos. Una automatización que regule dichos tiempos disminuiría la congestión y además eliminaría la necesidad de personal para regular el tráfico de forma manual y en caso de que se detecte un problema se ganaría más rapidez y efectividad a la hora de aplicar una solución [6]. Para solucionar esto se podría dotar las carreteras con diferentes tipos de sensores con los que obtener los datos de tráfico. Este es un enfoque que las Smart Cities han recogido y muchas de ellas cuentan con algún ejemplo de al menos medición de tráfico con sensores. Por ejemplo tanto Londres[7] como Santander [4] tienen este tipo de sensores y proyectos. En nuestro proyecto debemos suponer que nuestra ciudad cuenta con estos sensores en número y distribución adecuada, ahora bien hay multitud de sensores para monitorizar el tráfico, algunos de los cuales son los siguientes:

- Sensores fotográficos: mediante una instantánea o un video y tras un complejo procesado, se obtiene información del vehículo o flujo de vehículos. Sin embargo este tipo de detección está condicionada por la cantidad de luz en el ambiente y el fondo cambiante de la escena, además el tiempo atmosférico puede también introducir inexactitud en la lectura. Además, este sistema totalmente generalizado suscita una cuestión sobre cómo mantener de forma adecuada la privacidad. [6]

- Sensores RFIF(Radio Frecuency IDentification): este tipo de tecnología es un sistema de almacenamiento y recuperación de datos mediante ondas de radio

frecuencia en el que usa elementos llamados etiquetas o transpondedores para almacenar u obtener la información, estas etiquetas son actualmente utilizadas en trasporte, gestión de seguridad, productos de lujo, etc. Suscita la misma preocupación sobre la privacidad que el caso anterior.[8][9]

- Aros inductivos: El funcionamiento de un aro inductivo se basa en que al acercar el objeto metálico al aro inductivo se produce un cambio de inductancia por el campo magnético de dicho objeto. En el caso de los aros inductivos colocados en las calles (ver Ilustración 2), se interpreta la presencia de un vehículo mediante un circuito controlador de adquisición de datos [10]. Se pueden colocar uno o varios para obtener la información, sin embargo aunque se puede estimar datos como la velocidad del vehículo y la longitud del mismo con un único aro las mediciones no son muy precisas, por lo que para lograr mayor exactitud en los datos se necesitarían dos de ellos. Si bien, aun hay algunos problemas en el procesado de los datos para situaciones de gran congestión de tráfico [11].

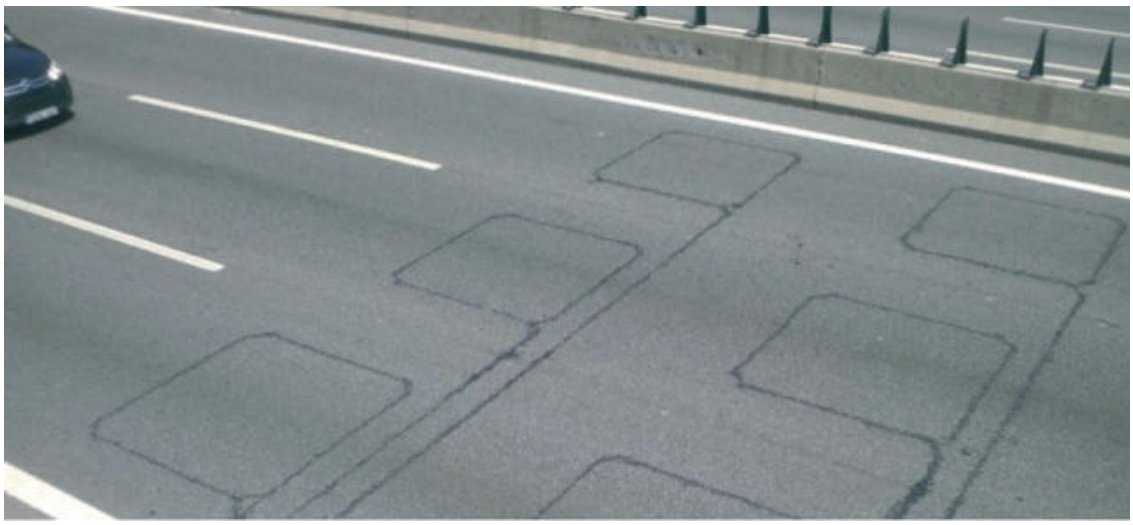

Ilustración 2. Ejemplo de aros inductivos integrados en la carretera

Para llevar a cabo el proyecto se optó por simular la presencia en las calles de la ciudad del último tipo de sensor mencionado. Se eligió así porque no plantea problemas de privacidad que no se abordan en este trabajo fin de grado, además de por la simplicidad. Supondremos también que este tipo de sensores se hayan de forma generalizada en toda la ciudad o en el mapa que esté bajo estudio, en la distribución que se explicará en la sección 3.5. Debe tenerse en cuenta que sustituir este sensor por otro, desde el punto de vista de una

simulación, no debería causar mayor problema, dado que la información básica que se necesita conocer, flujo de automóviles en un punto determinado por periodo de tiempo, seguirá siendo la misma.

2.4 Control de tráfico

El control de los semáforos se puede enfocar desde diferentes metodologías. Una de ellas consiste en establecer tiempos prefijados a partir de un estudio previo de los datos obtenidos mediante el análisis del tráfico. Otra metodología sería llevar a cabo el análisis en tiempo real de los datos y controlar el semáforo a partir de un análisis de dichos datos. Se pueden establecer diferentes objetivos que se pretenden perseguir con este control dinámico como por ejemplo minimizar el tiempo medio de espera de los vehículos que se aproximan a la intersección, reducir el tamaño medio de la cola de los vehículos que esperan en la intersección o bien reducir el consumo total de combustible y las emisiones producidas [12]. A continuación se explicarán algunas de las soluciones propuestas hasta ahora u otras planteadas para el futuro.

2.4.1 Comunicación unidireccional infraestructura – vehículo

Esta solución se basa en equipar cada semáforo con un transmisor de radio de forma que pueda comunicar información a los coches que se van aproximando a él. Una de las muchas posibilidades que ofrece este tipo de solución es que cada semáforo transmita su estado continuamente mediante broadcast a todos los vehículos dentro de su rango. La información del estado constaría de una descripción de la intersección, el color de cada una de sus luces en las múltiples direcciones e información del siguiente estado esperado. Posteriormente la unidad a bordo del vehículo destinada a procesar los datos provee información al conductor con sugerencias sobre la velocidad óptima para aproximarse a la intersección para alcanzarlo en verde. Dicha velocidad estará entre el rango mínimo y máximos impuestos por las normas de circulación pertinentes. Además si el vehículo está detenido por un semáforo en color rojo, se podría informar al conductor cuándo va a finalizar dicha fase. En caso de que el coche no pueda alcanzar la fase verde en la siguiente intersección se podría mostrar en el panel de aviso la máxima velocidad de la vía para alcanzar la intersección de la forma más rápida y eficiente. Dichos mecanismos podrían combinarse con un "Start-Stop" del motor permitiendo ahorrar combustible. Por tanto esta solución busca reducir los tiempos de espera en los semáforos para reducir el consumo de combustible.[13]

2.4.2 Semáforos adaptativos

En este segundo método, además de tener un canal descendente de comunicación entre el semáforo y el vehículo, se incluye un canal de subida. De esta manera hay una comunicación bidireccional entre los vehículos y el semáforo. Esto permite, por ejemplo, a los vehículos de emergencias modificar el estado del semáforo mediante unos mensajes de control específicos que permitiría abrir el semáforo para dicho tipo de vehículo y cerrarlo en el resto de direcciones. Otra utilidad es usar sensores para medir la densidad de tráfico y por tanto incorporar esta información al ciclo del semáforo junto con otras informaciones como por ejemplo si hay vehículos en espera, cuántos hay en un momento concreto en cada dirección, etc. Incluso se podría informar a varios semáforos con anterioridad del camino completo que va a seguir un vehículo o conjunto de vehículos para que los semáforos usen dicha información en su ciclo, si bien, esto podría plantear algunos problemas de privacidad.[13]

2.4.3 Semáforos virtuales

En esta última aproximación se reemplazan los semáforos físicos por semáforos virtuales gestionados y controlados por el conjunto de vehículos en los alrededores de una intersección. Todos los vehículos que se aproximan a la intersección son tratados como un clúster, este clúster está coordinado por un líder que se convierte en el semáforo virtual operando con el algoritmo usual de un semáforo. Sin embargo esto requiere una coordinación con el resto de vehículos que se aproximan a la intersección. Las ventajas de esta aproximación son la reducción de tiempo medio de viaje así como de las emisiones de carbono y una reducción de los costes de desplegar y mantener la infraestructura física de los semáforos.[13]

2.4.4 Control de tráfico mediante sensorización

Uno de los sistemas clásicos que se ha utilizado en la gestión de políticas de semáforos es SCOOT (Split, Cycle and Offset Optimising Technique), que es un sistema de control de tráfico adaptativo que usa los datos obtenidos en tiempo real mediante aros inductivos u otro tipo de sensores. Con estos datos un programa calcula la configuración óptima de la fase que minimiza el retraso. Para llevar esto a cabo, SCOOT tiene un modelo de tráfico en el que estima el tamaño de la cola y el número de vehículos que tienen que pararse en el semáforo, SCOOT además conoce los flujos de vehículos en cada carretera, los tiempos de los semáforos y cuánta capacidad restante hay en cada semáforo, lo que le

permite llevar a cabo la coordinación de los tiempos de los semáforos para minimizar el tiempo medio de espera, modificando el tiempo de los ciclos del semáforo o el tiempo de las fases en verde. Adicionalmente el programa hace varias pruebas de posibles modificaciones a los ciclos y valora si dicha modificación tuvo el impacto deseado en la optimización de la coordinación entre los semáforos para así poder ser utilizada en los ciclos venideros[14][1][15]. La primera versión de este sistema data del año 1981, se implantó en ciudades como Londres, Oxford y Glasgow. Posteriormente se implantó en más de doscientas instalaciones en diferentes países alrededor del mundo[16]. El objetivo de este Trabajo de Fin de Grado es crear la base de una herramienta similar a SCOOT como paso previo para posteriormente poder realizar simulaciones más complejas y poder implantar diferentes políticas de control de tráfico a través de los semáforos.

2.4.5 Semáforos de tráfico adaptativo usando comunicación vehículo a vehículo

Equipando a los vehículos y al semáforo con un equipo de comunicación inalámbrica que les permita intercambiarse información mediante broadcast el vehículo enviará información al semáforo acerca de: su identificador, velocidad y dirección permitirá al sistema de control del semáforo modificar la duración de los tiempos de ciclo de éste, ya bien eliminando algunas fases del ciclo, modificando la longitud del ciclo o modificando el número de rutas en verde para cada fase mediante métricas de control del retardo y tamaño de la cola.[12]

2.4.6 DIVERT

Dentro del repaso al estado del arte, es conveniente hablar con un poco de detalle sobre la propuesta DIVERT[17], cuyos escenarios y resultados han sido elegidos como base para comparar con los de este proyecto fin de carrera por la cantidad de detalles que ofrece de las siumulaciones. Divert es un sistema de reenrutado de vehículos para evitar y aliviar la congestion. Este sistema parte de dos premisas: el sistema debe preservar la privacidad en la localización de los vehículos y dicho sistema tiene que ser un sistema distribuido. En este caso con sistema distribuido se hace referencia a que el cálculo de la nueva ruta no lo hace un servidor central sino un cluster formado por los propios vehículos para así reducir la comunicación y computación del servidor central con lo que se mejora la escalabilidad del sistema. Respecto a la privacidad DIVERT se centra en proteger la localización de los vehículos en el servidor central, no de los otros coches que tiene en las proximidades, es

decir, se minimiza el número de datos sensibles al servidor en vez de proteger dichos datos una vez que están subidos al servidor.[17].

Cada vehículo detecta la densidad de tráfico de la carretera por la que circula Esto se lleva a cabo mediante los *beacons* que envían los vehículos periodicamente, de tal manera que contando el número de ellos recibidos en un vehículo en una ventana de tiempo se hace una estimación de la densidad de tráfico. Cada vehículo envía información con cierta probabilidad de manera anónima al servidor, a través de programas que protegen el anonimato como *Tor*. Cuando un vehículo detecta que la densidad de la carretera es mayor que un cierto umbral, el vehículo decide probabilísticamente cuando debe enviar dichos datos al servidor. Si el servidor detecta congestión envía información sobre el mapa de tráfico a los vehículos que le enviaron dicha información y posteriormente dichos vehículos compartirán la información recibida con sus vehículos próximos en forma de broadcast pero limitando el número de saltos para así no inundar la red.

La arquitectura del sistema está compuesta por un servidor central y un dispositivo a bordo de cada vehículo. DIVERT utiliza dos tipos de comunicaciones, una en que los vehículos se comunican con el servidor mediante una red celular en la que informan al servidor sobre los datos de densidad local en sentido ascendente, mientras que en sentido descendente los vehículos reciben información sobre la densidad global de la red que el servidor construye un mapa de densidades a partir de los datos enviados por los vehículos. Cada segmento de carretera tiene asociado un peso dinámico que representa en tiempo real la densidad de tráfico de dicho segmento de carretera. Cada vez que un segmento de carretera exhibe señales de congestión, es decir, el valor de la densidad supera un umbral, el servidor envía a los vehículos que enviaron información anteriormente nuevos datos sobre la densidad de tráfico de los segmentos de carretera no más lejanos a tres segmentos. El segundo tipo es una comunicación broadcast entre vehículos próximos que es para determinar la densidad de tráfico local y además distribuir los datos recibidos del servidor al resto de vehículos, indicando la ruta que va a seguir cada vehículo para así calcular de forma óptima la ruta a seguir.

Para estimar los tiempos de viaje de los vehículos DIVERT considera tres casos:

- Si no hay reportes de información para las carreteras, el servidor considera el tiempo de viaje igual a si hubiera densidad cero.

- Si la densidad no es cero, pero el tiempo desde el último reporte de información es superior a un umbral, el servidor no tiene en cuenta el reporte y resetea la densidad de tráfico a cero.

- Si la densidad no es cero y además el tiempo desde el último reporte no es superior al umbral, el servidor usa el modelo Greenshield para estimar el tiempo de viaje.

2.5 Los diferentes Simuladores

A la hora de escoger un simulador para llevar a cabo este trabajo de fin de grado, se valoraron diferentes simuladores y las características de cada uno de ellos, que se pueden ver en la Tabla 1. Estos simuladores son los que mayoritariamente se utilizan para realizar investigación en comunicación vehículo a vehículo, coche conectado y Smart Cities, según se deduce de una revisión pormenorizada de la bibliografía [18][17][19][20][21].

	Tipo de Simulador	Open Source	Estructura de fichero y programación	Dependencias
SUMO	Movilidad	Si	XML, Python	
Omnet++	red	Si	C++, XML, NED	
Veins	Red y movilidad	Si	Python, Java, C++, matlab	Uso de SUMO y Omnet++
iTetris	Red y movilidad	Si	XML, C++, Python	Uso de SUMO y Ns3
ns3	red	Si	C++	

Tabla 1 Tabla comparativa de los diferentes simuladores

Los simuladores se podrían clasificar en tres tipos: simulador de movilidad, de red o simulador VANET.

Los simuladores de movilidad permiten realizar simulaciones de flujos de vehículos, ya sea individualmente (microscópico) o en densidad de vehículos (macroscópico).

Los simuladores de red son normalmente usados para testear el rendimiento de los protocolos para la movilidad de los nodos, ya que estos están la mayor parte del tiempo en movimiento, y la topología de la red cambia continuamente.

Los simuladores VANET permiten una combinación de los dos anteriores, esto puede ser bien porque el simulador consta del software necesario para realizarlo, o que utiliza otros simuladores especializados en cada tipo de simulación[19][21].

Debido a la bibliografía estudiada previamente a la realización del proyecto [18][17] se constató la versatilidad que SUMO proporciona mediante las múltiples herramientas que incorpora de conversión entre formatos de mapas, conversión de rutas, ficheros de salida de análisis tras las simulación e integración junto con otras librerías para obtener gráficas de resultados respecto a la simulación. Por otro lado, al ser ns3 y Omnet++ simuladores de red no se adecuaban concretamente a lo que se pretendía simular. Itetris es un proyecto que terminó en 2010 y por tanto ya no recibe más actualizaciones, por lo que fue una razón de peso para descartar su uso. Se optó finalmente por SUMO y Veins, aunque tras realizar parte de este trabajo de fin de grado no se hizo uso de Veins en ninguna parte de las simulaciones ni del desarrollo del programa, por lo que finalmente, se optó por utilizar únicamente SUMO en el que hablaremos en más profundidad en el siguiente capítulo.

3

Simulador SUMO

3.1 Introducción

SUMO (**S**imulation of **U**rban **Mo**bility) es un simulador de código abierto de tráfico rodado microscópico, es decir, en las simulaciones es capaz de proporcionar parámetros específicos de cada vehículo incluyendo por lo tanto parámetros microscópicos además de parámetros macroscópicos de la simulación. Los ficheros necesarios para realizar la simulación están en formato XML, que se pueden modificar directamente o con el uso de diferentes herramientas proporcionadas por el simulador. A continuación se explica la estructura de los ficheros para la simulación de SUMO, cómo generar un fichero de red para la simulación, exportar mapas para crear ficheros de red, generar el fichero de rutas para la simulación, lanzar la simulación y obtener las estadísticas de la simulación y además se detallan las librerías de utilidad para gestionar los datos durante la simulación. Para más información se puede acudir al manual [22].

3.2 Estructura de ficheros de SUMO

Los ficheros para la simulación de SUMO se dividen en 4 partes:

- Un fichero de red *.net.xml*
- Un fichero de rutas *.rou.xml*
- Un fichero de configuración *.sumocfg*
- Ficheros adicionales

En el fichero de red *.net.xml* se define los nodos (intersecciones), líneas (carreteras), la topología de dicha red, los diferentes parámetros de los elementos de la red tales como las conexiones entre las diferentes líneas en los nodos, la velocidad de la línea, el número de

carriles por línea, etc. En la sección 9.1 se puede ver el fichero *.net.xml* en el que está definido el cruce expuesto en la Ilustración 3 Ejemplo Cruce Se puede ver que el fichero consta de varios atributos *edge* (carriles), en el que se indica su ID, qué tipo de carril, la velocidad del carril máxima en m/s y la forma de éste. También se especifica los semáforos de la red mediante el atributo *tlLogic* que indica el ID, el tipo, los programas del semáforo, el offset junto con las fases del semáforo y sus duraciones. Otro atributo asociado a este fichero es *junction* que indica los nodos intersección junto con su ID, tipo coordenadas y conexiones que tiene asociados. Por último aparece el atrubuto *connections* de la red que indican las rutas posibles de los vehículos en las diferentes intersecciones, esto viene dado por el atributo *connection* que indica el carril de origen y destino, a través de qué carriles se llevan a cabo y el ID de la conexión en la intersección.[23]

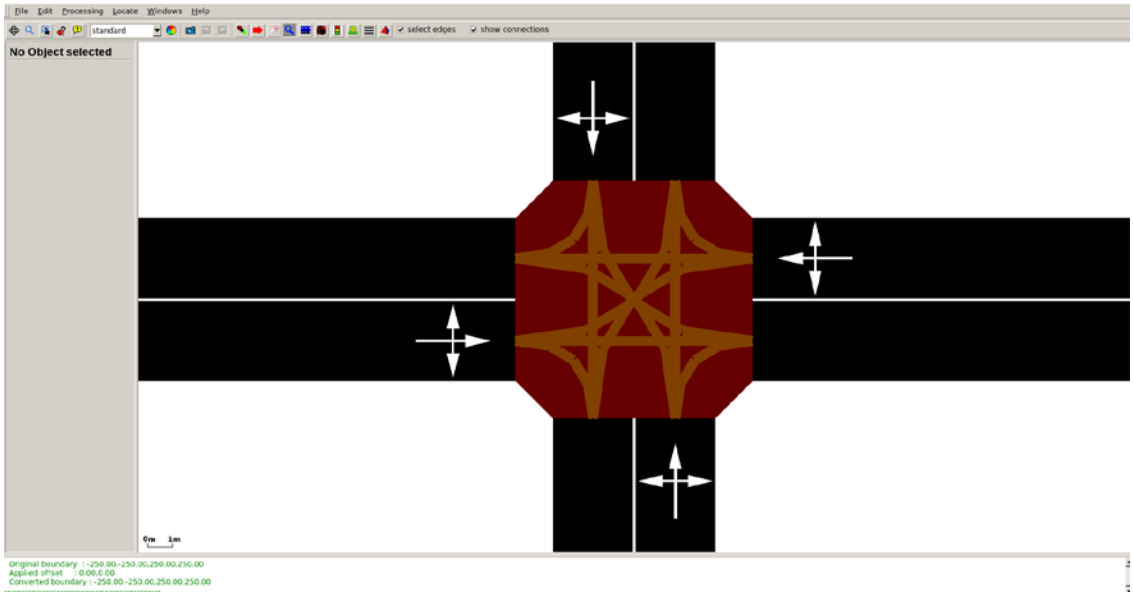

Ilustración 3 Ejemplo Cruce

En el fichero de rutas (*.rou*) se definen los tipos de vehículos y parámetros específicos de cada uno de ellos. También se definen las rutas que van a seguir cada vehículo o las copias de cada tipo de vehículos, y los instantes en que saldrán los vehículos del nodo origen hacia el nodo destino. En la sección 9.2 se puede ver el fichero *.rou.xml*. En él se aprecian varios atributos de tipo *vehicle* con cada uno indicando su ID, el instante en que el vehículo comienza su viaje, y los nodos por los que tiene que pasar durante su ruta. Ilustración 3 Ejemplo Cruce[24]

El fichero de configuración es el fichero básico para lanzar la simulación en el que se incluye una referencia por cada fichero que se quiere usar en la simulación, es decir, al menos

para llevar a cabo la simulación se necesitan los ficheros *.net* y *.rou*; también se podrían incluir ficheros adicionales. Por último habría que definir el tiempo de inicio y fin de la simulación. En la sección 9.3 se puede ver el fichero *.sumocfg* en el que esta definido el cruce expuesto en la Ilustración 10. Ejemplo cruce en fichero de red.

En los ficheros adicionales se definen los elementos adicionales de la red tales como aros inductivos, semáforos, paradas de autobús, etc.[25]. En la sección 9.4 se puede ver el fichero *.add.xml* en el que está definido el cruce expuesto en la Ilustración 10. Ejemplo cruce en fichero de red y en el se pueden ver varios atributos de tipo *e1Detector*, en el que se indica su ID, la línea en el que está el detector, la posición en la línea, su frecuencia de muestreo y fichero donde está definido el aro inductivo (todos pueden estar definidos en un mismo fichero).[26]

Para generar el fichero *.sumocfg* se ha creado un script llamado *gencfg* (ver sección 9.1). La sintaxis de uso del script es la siguiente: para que funcione correctamente el nombre de los ficheros (sin contar la extensión) *.net .rou* o *.add* tienen que ser idénticos y estar en la misma carpeta. Además, únicamente tiene que haber un fichero de cada tipo en la carpeta, si no, no funcionará correctamente. Por último para que funcione la simulación hay que tener en cuenta que se necesitan al menos tener creados previamente un fichero *.rou* y otro *.net*. Teniendo esto en consideración se ejecuta el script el cual te pide el nombre del fichero sin la extensión y éste genera el fichero *.sumocgf* relativo a los ficheros con mismo nombre mencionado anteriormente. Si se desea modificar la duración de la simulación se puede editar con un editor de texto el fichero *.sumocfg* generado. Para más información sobre el código ver el Anexo I.

3.3 Generación de una red

Para generar una nueva red (intersecciones de carretera + carreteras que las unen), o lo que es lo mismo el fichero *.net.xml*, hay varias alternativas. En la primera alternativa la red se puede generar mediante un script provisto por SUMO, el cual utilizando el comando *netgenerate* permite generar formas típicas de red tal como mallas, red en forma de tela de araña o bien redes aleatorias, especificando diferentes parámetros en función de la topología deseada (más información sobre este comando se puede encontrar en [27]). Otra opción para generar una red es hacerlo con el editor gráfico provisto por SUMO, llamado *netedit*, que se muestra en la Ilustración 4, en la que se permite dibujar la red de manera similar, a un programa de dibujo, y donde puedes ajustar todos los parámetros de la red. Este editor

también permite modificar redes ya existentes guardadas en formato *.net.xml*. Una tercera alternativa sería definir el fichero de red manualmente, escribiendo el código necesario para generar la red deseada.[27][28][29]

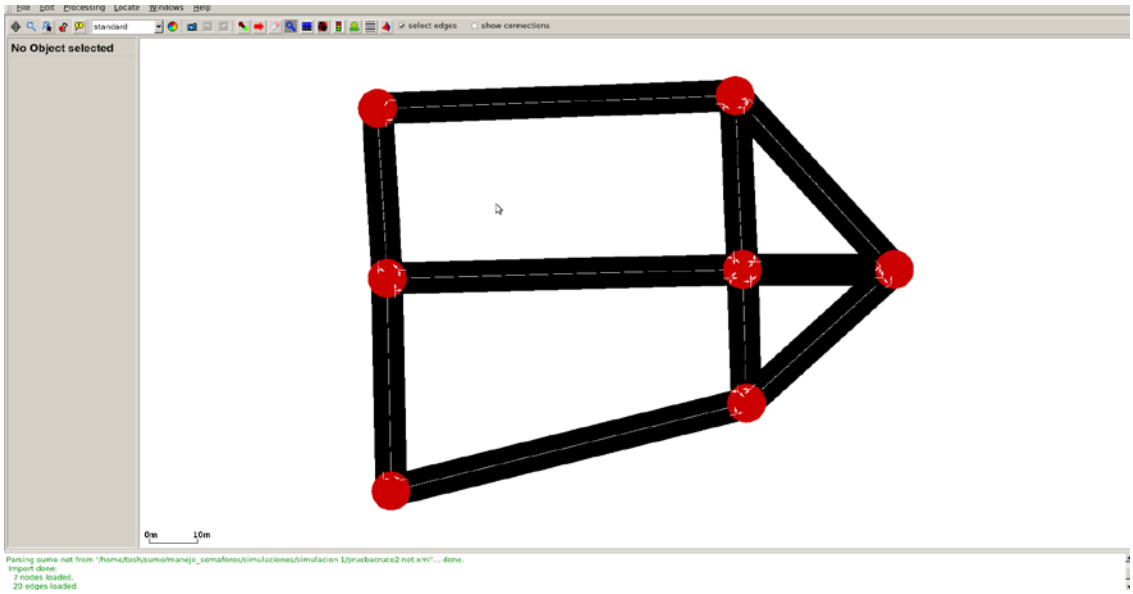

Ilustración 4: Imagen de la herramienta Netedit

3.3.1 Exportar mapas en otros formatos

SUMO nos permite importar mapas externos en otros formatos diferentes al formato nativo usado por el simulador. Esto se puede llevar a cabo mediante el script *netconvert* provisto por el simulador, que nos permitirá de manera sencilla importar mapas de código abierto en formato OSM obtenidos de Open Street Map [30][31]. En la Ilustración 5 se puede ver un ejemplo de dicho mapa en los alrededores del campus Miguel Delibes de la UVa.

Se ha creado un script con nombre *genmappol* (ver sección 9.6) que automatiza este proceso. La sintaxis de uso es la siguiente: se introduce el nombre del fichero sin la extensión del fichero que contiene el mapa OSM descargado de Open Street Map. El script no solo convierte el mapa, sino que además crea un fichero de rutas y un fichero de red, para posteriormente generar el fichero *.sumocfg,* como se mencionó en la sección anterior, e inicia una simulación gráfica en SUMO. El mapa resultado de puede ver en la Ilustración 6. Para más información ver el anexo I.

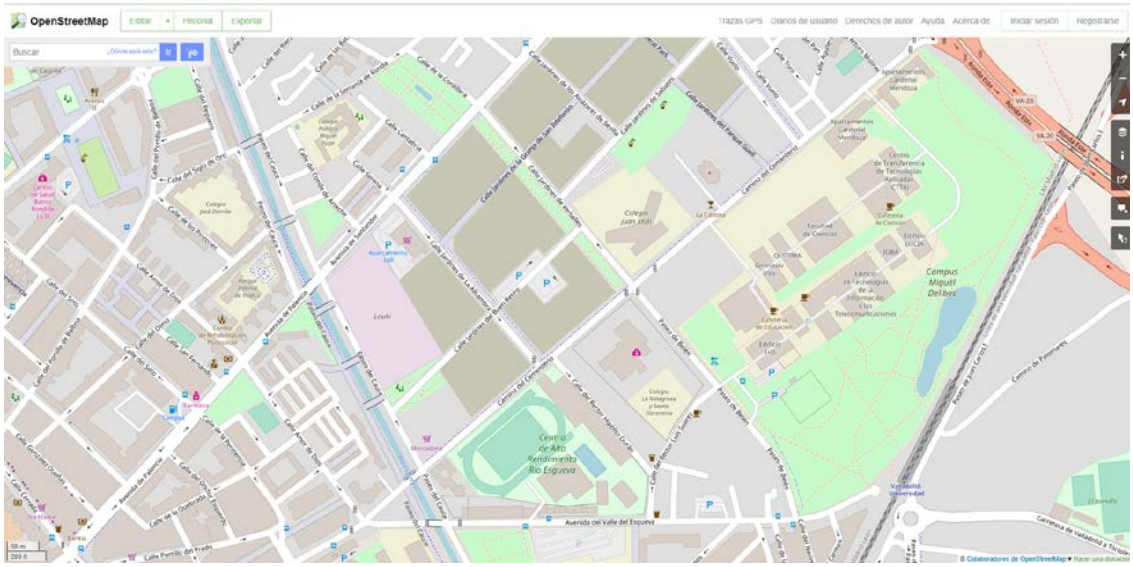

Ilustración 5. Mapa Campus Miguel Delibes obtenido de Open Street Map para exportar a formato SUMO

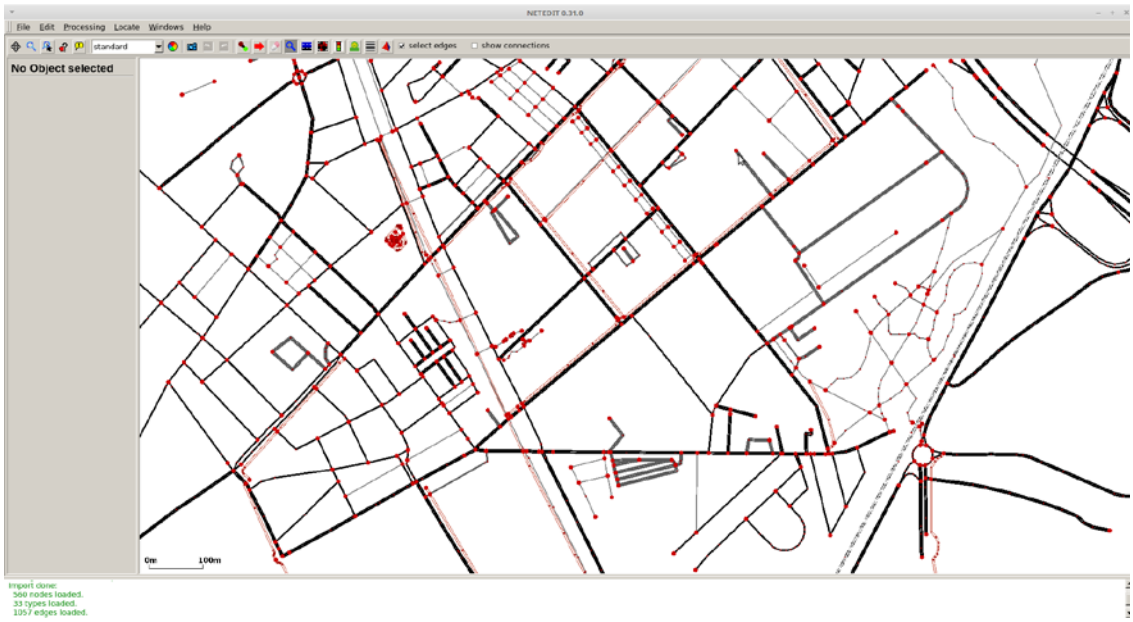

Ilustración 6. Mapa Campus Miguel Delibes obtenido de Open Street Map convertido a formato SUMO

3.4 Generación de rutas

Para generar las rutas que van a seguir los automóviles durante la simulación a partir de un mapa dado hay varias opciones. La primera opción es definir las rutas manualmente, especificando el tipo de vehículo, los nodos por los que se quiere que transcurra dicha ruta y el tiempo en que se quiere lanzar una copia de ese tipo de vehículo. Hay varios tipos de vehículos: privado, autobuses, taxi, etc. Para ello se utiliza el atributo *vClass* o *vType* del fichero de rutas, para mayor información consultar [24].

Por otro lado hay varias opciones interesantes que pueden generar las rutas automáticamente. La primera opción para generarlas es mediante el script *randomtrips.py* provisto por SUMO. Éste permite generar rutas aleatorias y especificar varios parámetros como la probabilidad de generación de tráfico en función de la longitud del carril o el número de carriles, modificar dicho peso de probabilidad de generación si los nodos de la red son nodos frontera de la red, generar el tráfico en instantes determinados mediante un periodo específico o bien en instantes determinados mediante una variable aleatoria binomial. Para la generación de la ruta el programa utiliza tres nodos aleatorios de la red que marca como nodo inicio, nodo fin y un nodo intermedio por el que tiene que pasar dicha ruta, después mediante el algoritmo de Dijkstra se busca la ruta más corta. Dicho script proporciona unos ficheros *.trips*, que se pueden convertir a formato *.rou*, necesarios para la simulación, mediante el comando *duarouter* proporcionado por SUMO especificando el fichero de red para el cual se generaron los ficheros *.trips*.

Se ha creado un script con nombre *gentrafico* (ver sección 9.7). La sintaxis de uso del script es la siguiente: se introduce el nombre del fichero de red sin la extensión *.net.xml* y a continuación esto genera un fichero *.rou* haciendo uso del script *randomtrips* provisto por SUMO. Para más información ver el Anexo I.

Otra alternativa es usar la herramienta Traffic Modeller creada por Leontios G. Papaleontiou y Marios D. Dikaiakos [32] que permite visualizar el mapa y definir rutas específicas de forma gráfica sobre el mapa que se visualiza. Además, permite definir gráficamente en el mapa áreas de generación y de recepción de tráfico para crear después múltiples rutas entre esas dos regiones tal y como se puede ver en la Ilustración 7. Se ha generado un script llamado *gentraftm* (ver sección 9.8) para usar esta herramienta, especificando simplemente el mapa OSM obtenido, que lo convierte al formato para poder trabajar con SUMO, y que en conjunto con Traffic Modeller y duarouter te permite crear los ficheros de rutas[33]. Cabe destacar que durante la ejecución del script es importante poner el mismo nombre del fichero a todos los ficheros que se vayan almacenando (es decir, mismo nombre aunque distinta extensión), si no el script no funcionará correctamente. Además cuando se abra la aplicación de Java Traffic Modeller inicialmente hay que crear el proyecto en el mismo directorio donde está el fichero OSM, con el mismo nombre que el mapa OSM. Posteriormente se puede generar tráfico en dos tipos de formatos: el primero es crear la ruta individualmente indicando el nodo origen y nodo destino de la ruta, el segundo tipo es crear una región de generación de tráfico y otra región receptora, como se puede ver

en la Ilustración 7, en el que se crearán múltiples rutas entre ambas regiones. Por último se exporta el fichero de tráfico generado utilizando la opción de menú *Simulation*➔*Export*. Dentro del menú que se abre seleccionamos la carpeta nombre_simulacion que se ha creado en el mismo directorio que el mapa OSM, en la casilla *simulation name* escribimos el nombre del fichero de simulación sin las extensiones, seleccionamos las opciones deseadas y clicamos en *export*, le damos a continuar, esperamos 5 segundos para que el programa lo almacene correctamente y cerramos la aplicación en Java (no cerrar el script) y finalmente se puede seleccionar qué tipo de simulación es deseada si con interfaz gráfica o no. El script además provee los resultados de simulación en un fichero de estadísticas en la carpeta nombre_simulacion y un gráfico que indica el tiempo medio de viaje durante la simulación. Cabe indicar un bug al generar las rutas mediante la herramienta Traffic Modeller, dicho bug afecta a las conexiones de las intersecciones, concretamente establece todas las conexiones posibles en un cruce aunque en el fichero *.net* original usado para generar las rutas las conexiones no existieran. Es decir si originalmente tenemos un carril que tiene tres posibles conexiones, el programa crearía además una nueva conexión de dicho carril al carril en sentido contrario, en el sentido de permitir un giro de 180 grados, aunque ésta no existiera previamente. Además, esta nueva conexión no puede ser eliminada del fichero ni siquera manualmente, dado que en ese caso nos da un error durante la simulación. Por lo tanto, por lo que se optó es que en el algoritmo de creación de fases este giro no se tiene en cuenta, dado que, por ejemplo en España, la mayoría de estos giros están prohibidos.

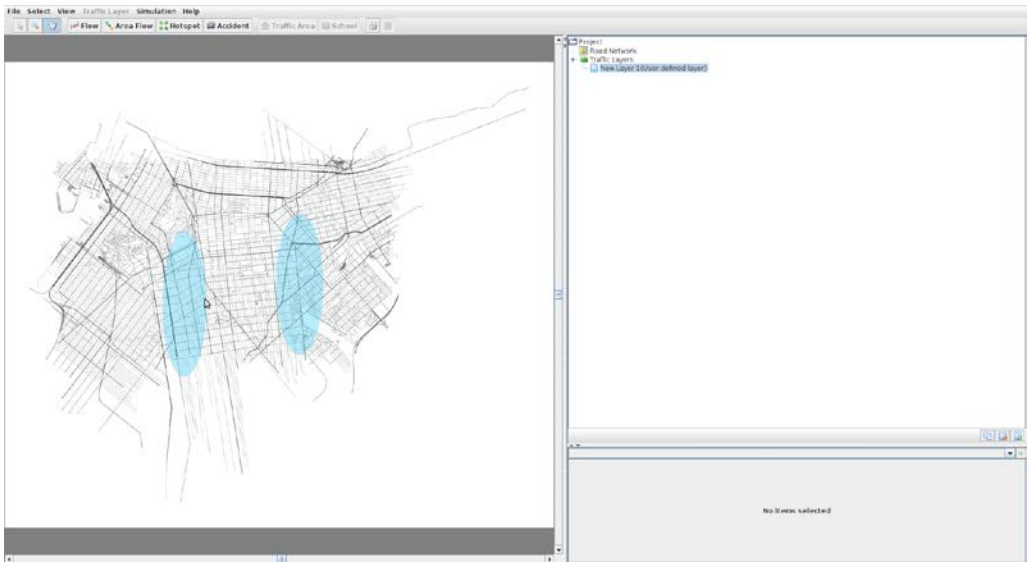

Ilustración 7. Herramienta Traffic Modeller

3.5 Añadir aros inductivos al fichero de red

Para que los semáforos puedan obtener la información necesaria requieren unos aros inductivos u otro tipo de sensores. Sin embargo en un mapa exportado no incluye un fichero adicional que indique las posiciones de los aros inductivos. Para ello se ha creado un script llamado *crearloopsennet* (ver sección 9.9) a partir del script de SUMO llamado *generateTLSE1Detectors.py* que crea dos loops por carril controlados por un semáforo, uno para un carril de entrada y otro para un carril de salida. Para utilizar el script simplemente editando en las primeras líneas el atributo ruta y nombre, que indican el directorio del fichero de red y el nombre de éste respectivamente. Al ejecutar el script se genera un fichero *nombre.add.xml* con los aros inductivos en la misma ruta indicada anteriormente.

3.6 Simulación

Una vez generados todos los ficheros necesarios para la simulación y creado el fichero de configuración de la simulación (*.sumocfg*), SUMO nos permite correr la simulación de dos maneras. La primera tiene interfaz gráfica, que permite visualizar el mapa utilizado en la simulación y los vehículos que están recorriendo dicho mapa; esta opción es útil para ver posibles causas de congestión tales como semáforos estáticos, o intersecciones complejas. Para realizar esto tenemos que lanzarlo bien desde línea de comandos o desde un script de un lenguaje de programación con el comando *sumo-gui –c fichero.sumocfg*, donde *–c* sirve para especificar el fichero de configuración. Se pueden añadir diversas opciones como no

especificar fichero de configuración para posteriormente añadir mediante opciones los ficheros de rutas, red y adicionales. También se pueden especificar diferentes ficheros de salida que se desean obtener tras la simulación. Para mayor información consultar[34].

La otra opción consiste en lanzar la simulación mediante el terminal de comandos, sin el interfaz gráfico, con un funcionamiento similar a lo explicado anteriormente.[25]

Cabe destacar que la simulación de SUMO se produce en pasos (steps) de tiempo definidos en el lanzamiento de la simulación, es decir no es un simulador de eventos puro, por lo tanto en cada paso de tiempo se pueden producir varios eventos. Esto podría tener consecuencias en la adquisición de los datos, ya que en ese paso se pueden haber producido varias veces la adquision del mismo dato en los aros inductivos al tener éste una frecuencia de muestreo con un periodo de muestreo menor al paso de la simulación, por lo que habrá que eliminar los datos duplicados obtenidos.

3.7 Estadísticas de simulación

Para poder ver las estadísticas básicas de la simulación se requiere el uso de la opción –*summary* <*fichero*> en la ejecución de la simulación. En dicho fichero se mostrarán el tiempo medio de viaje, el tráfico insertado en la red, la velocidad media relativa, el tiempo medio de espera, cuántos vehículos hay esperando, cuántos vehículos han terminado su ruta y el instante de dichos parámetros en los pasos de la simulación[35][36]. Posteriormente con los datos obtenidos se pueden realizar gráficas con las evoluciones de los diferentes parámetros; para mayor información consultar [37]. Un ejemplo de fichero de salida puede verse en Ilustración 8.

Ilustración 8. Ejemplo de fichero de estadísticas de simulación

3.8 Librerías para obtener información del mapa durante la simulación

Sumolib es un conjunto de módulos escritos en Python, que permiten obtener características del mapa durante la simulación. Por ejemplo, nos permite obtener las coordenadas de diferentes elementos en la red, las intersecciones que tienen semáforos, las rutas que siguen los vehículos en un nodo dependiendo de la dirección en la que quieran ir, etc. Estas librerías serán ampliamente usadas en este Trabajo Fin de Grado para localizar la posición de los carriles a partir de las coordenadas de cada carril en las intersecciones. Con esto obtendremos qué rutas son compatibles entre sí para definir las fases de cada semáforo[38]. Una explicación más pormenorizada del uso de estas librerías en el Trabajo Fin de Grado se encuentra en el siguiente capítulo, en la sección 4.2.

4

Planteamiento del problema y proceso para solucionarlo

4.1 Planteamiento del problema

Como ya se explicó en el primer capítulo de la memoria, el objetivo a resolver con este proyecto es crear una herramienta que permita analizar diferentes situaciones de tráfico a partir de los datos obtenidos mediante aros inductivos y establecer unos tiempos de duración de las fases así como las fases mínimas necesarias para el funcionamiento de dicho semáforo. Este procedimiento puede ayudar a resolver cuestiones más complejas, como minimizar el tiempo de viaje medio, reducir las emisiones contaminantes, etc., utilizando para ello algoritmos de distinto tipo. Una posible aproximación para resolver los diferentes problemas planteados y que, como hemos visto en la revisión del estado del arte, es ampliamente utilizada, es variar los tiempos que duran las fases en los semáforos en función de información que es recogida en tiempo real. Para obtener dicha información en tiempo real, en nuestro caso de uso se utilizarán aros inductivos como detector de tráfico con la particularidad de que desde el punto de vista de la programación identificaremos los vehículos con un identificador. Sin embargo esto no afecta a la aplicción de los resultados de la simulación al mundo real, tal y como se explica en la sección 4.5.

En el resto de secciones de este capítulo se explica primero cómo se construye el programa para un tratamiento lo más automático posible de los mapas. Se introducen primero las librerías TraCI y Sumolib, explicando cómo se utilizan y los componentes esenciales para el tratamiento automático de la información en el programa. Una vez se conocen estos componentes se pasa a explicar detalladamente los pasos para generar toda la información que nos va a permitir tratar de forma individualizada cada cruce de semáforo, concretamente los aros inductivos que hay en el cruce, las rutas y cómo se relacionan entre ellas. A partir de este punto se describe una heurística para generar las fases que van a seguir los semáforos de cada cruce. Estas fases serán a las que posteriormente se les asigne un

tiempo variable en función de la política de control que se desee probar. Finalmente se describe como extraer los datos de la simulación en tiempo real.

4.2 Librerías para control de tráfico durante la simulación

TraCI (**Tra**ffic **C**ontrol **I**nterface) es una librería incluida en SUMO que permite obtener datos de los diferentes objetos de la simulación durante su ejecución, además de permitir modificar su comportamiento durante ésta. Es decir, permite modificar el estado de los semáforos, las rutas de los vehículos, y obtener diferentes datos tales como número de vehículos en un carril, número de vehículos que han pasado por un aro inductivo, etc. Esta librería, la cual está escrita en Python, será de utilidad para asignar aros inductivos a un semáforo, y a partir de los datos obtenidos de los aros inductivos de un semáforo modificar los tiempos de las fases de dicho semáforo.[39]. Para hacer esto utilizamos la función *get_semloops* (ver sección 9.10). Esta función inicialmente lee fichero *.net.xml,* usando la función *sumolib.net.readnet()* de la librería sumolib previamente mencionada con lo que obtendremos unos datos que serán llamados mapa para la simplicidad de la explicación. A continuación se obtienen los semáforos asociados al fichero de red mediante la función *mapa.getTrafficLights()* la cual nos devuelve una lista con todos los objetos semáforos de la red. Cabe recalcar que la salida son objetos, no un string, que es lo que nos interesa en nuestro caso. Por lo tanto para obtener su string correspondiente habrá que utilizar la función *objeto.getID()* (esta función no viene definida en la documentación de python provista por SUMO, para mayor información sobre estas funciones u otras funciones consultar las librerías correspondientes a los objetos deseados a procesar). Además también obtenemos una lista de todos los aros inductivos que hay en el mapa mediante la función *traci.inductionloop.getIDList()*. Posteriormente se comprueba si hay aros inductivos y semáforos en el mapa de la red, en caso negativo el programa lanzará un warning y terminará la simulación. Una vez obtenida la lista de semáforos se recorre la lista de objetos de semáforo mediante un bucle for y con la función *traci.trafficlight.getControlledLinks()* se obtienen los carriles que controla un semáforo para cada objeto de tipo semáforo. A continuación se procede a comprobar si hay más de una ruta por semáforo. Esta comprobación se lleva a cabo para evitar errores durante la preparación de los datos para la simulación ya que si hay una única ruta es porque el semáforo se importó incorrectamente al convertir el mapa de formato OSM a formato *.net.xml*. Una vez obtenidos los carriles se procede a comprobar la correspondencia entre los carriles del semáforo y el carril en el que está el aro inductivo con la función *traci.inductionloop.getLaneID()*. Tras recorrer todos los semáforos finalmente se

resuelven las correspondencias entre aros inductivos y semáforos. En la Ilustración 9 se puede ver el diagrama de flujo correspondiente a la explicación.

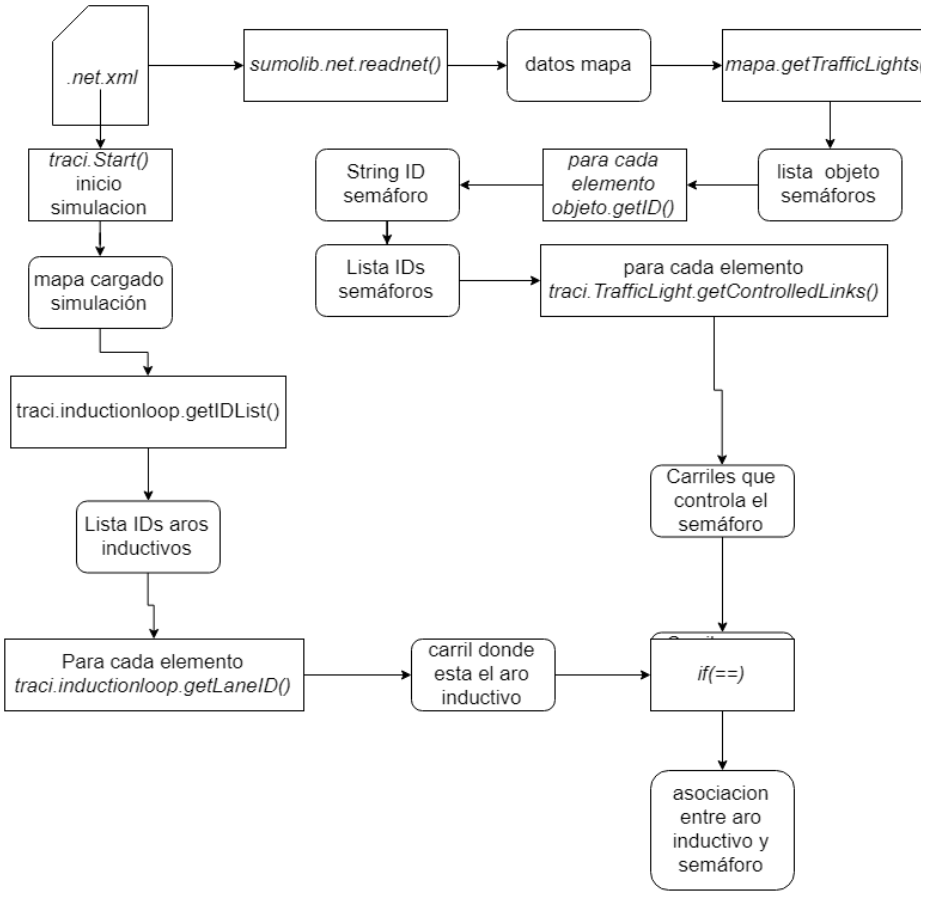

Ilustración 9. Diagrama de flujo para obtener correspondencia entre aros inductivos y semáforos

Para obtener los datos de los aros inductivos durante la simulación se usa la función *obtenervehiculospaso*. Su funcionamiento se hace con hilos (threads) para mejorar el rendimiento de la función y además simular el sistema independiente que corresponde a cada semáforo. En ella se crea un hilo por semáforo, cada hilo obtiene los IDs de los aros inductivos a partir de la lista calculada anteriormente y usando la función *traci.inductionloop.getLastStepID()* obtenemos los IDs de los vehículos que pasaron por el aro inductivo en el último paso de la simulación. Hay que tener en cuenta que un mismo ID puede estar repetido, ya que el aro inductivo tiene un atributo frecuencia de muestreo. Por lo tanto, hay que eliminar los duplicados, para ello primero se convierte la lista de vehículos en una tupla de Python y posteriormente mediante la función *difference()* de las tuplas se eliminan los duplicados presentes de la tupla. Por último hay que eliminar de la lista los IDs de los vehículos que han salido de la intersección, tanto de la lista de salida como de la lista de entrada. El ID del

vehículo es aprovechado para calcular específicamente la ruta que ha seguido el vehículo en el cruce, dicha información se aplicará de manera directa para influir sobre el tiempo de apertura de la fase correspondiente a esa ruta (ver sección 4.3).

4.3 Lectura de la red y preparación para la simulación

Se parte de un fichero *.net* y uno o varios ficheros adicionales que componen el mapa de carreteras de una región. Este mapa indica otras características de la red además de la topología de la red de carreteras como son los semáforos, aros inductivos, etc. Para crear una herramienta que trabaje de la forma más automática posible y que nos permita hacer simulaciones con diferentes heurísticas y métodos de control de trabajo, a partir de este mapa hay que identificar todos los semáforos de la red. Para ello se hará uso de la librería sumolib tal y como se explicó en la sección anterior, que permite identificar todos los semáforos incluidos en el fichero de red. Una vez que la librería ha ayudado a identificar todos los semáforos, hay que generar de forma automática las fases de dichos semáforos. SUMO es capaz de hacer una generación básica de dichas fases, pero es muy simple y no es capaz de interactuar con un algoritmo que cambie dinámicamente su duración o su orden. Por ello, una parte importante de este trabajo final de grado consistió en crear un algoritmo que fuera capaz de hacer este trabajo. Dicho algoritmo se divide en varias fases que serán explicadas con detalle en esta sección, pero que enumeradas serían:

1. Examinar todas las rutas posibles que pueden seguir los coches dentro del cruce y determinar cuáles de ellas son compatibles entre sí (la trayectoria de los automóviles no se cruza)

2. Determinar cuántas fases de semáforos se necesitan para que todos los trayectos estén abiertos por lo menos una vez en el ciclo. Se impondrá además la condición de que en una fase nunca existan trayectorias incompatibles.

3. Ordenar las fases de los semáforos para minimizar el número de cambios.

Es importante resaltar que una vez se ha construido este algoritmo que se puede aplicar de forma automática a cualquier mapa, nos permitiría aplicar cualquier heurística o algoritmo de inteligencia artificial o aprendizaje automático tan complejo como queramos, como puede ser aplicado a cada cruce por separado, al mapa al completo coordinando las fases de los cruces y buscando diferentes objetivos, minimizar el tiempo medio, reducir las

emisiones, etc. Por supuesto las simulaciones quedarán limitadas a variar los tiempos de semáforos, sin establecer aun posibilidad de comunicaciones entre los diferentes elementos de la infraestructura y los vehículos.

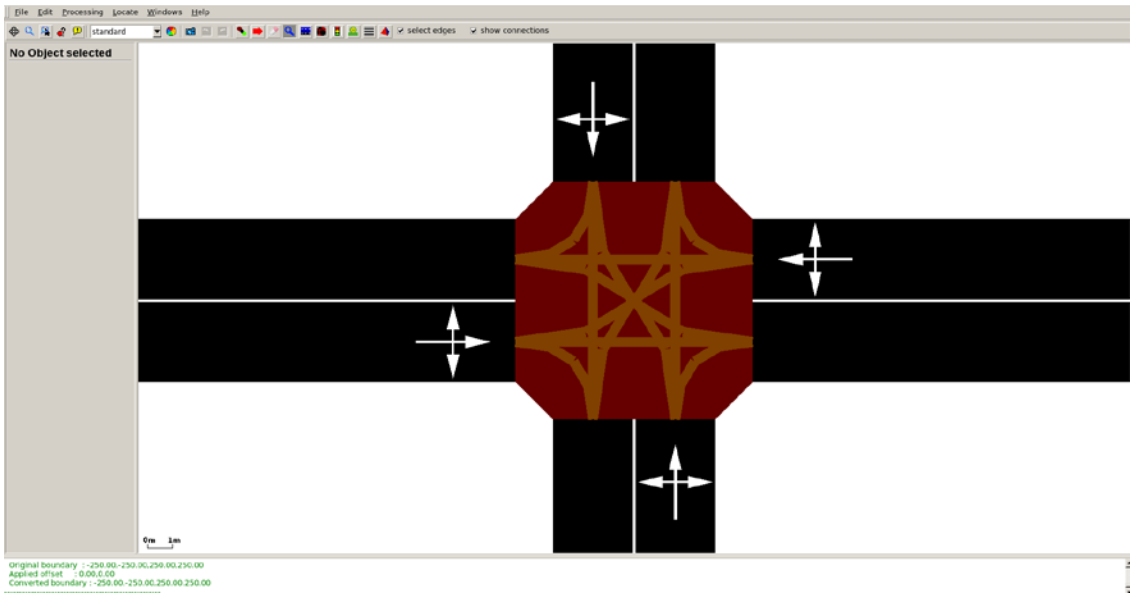

Ilustración 10. Ejemplo cruce en fichero de red

Para resolver el primero de los pasos, examinar todas las rutas posibles en un cruce, el criterio que se ha seguido es el siguiente: en una intersección, normalmente una ruta no es compatible con otra ruta si se produce una intersección entre ellas (véase por ejemplo la Ilustración 10), por tanto, un criterio para resolver dicho problema podría ser aproximar las rutas como rectas entre un origen y un destino, y si dos rutas se cortan dentro de la intersección no serán compatibles. El algoritmo que se ha seguido es el siguiente:

1. Por cada semáforo hay que leer todas las rutas posibles. Para ello recorremos el atributo *connections* del semáforo. El objeto que devuelve el atributo es una lista de la forma:

   `[[objetonodoorigen1,objetonododestino1,IDconexion1]….]`

2. A partir de esta lista se crean dos listas, una contiene datos del formato:

   `[[IDsemaforo1,IDnodoorigen1,Coordenadanodoorigen1,in,IDnododestino1,Coordenadanododestino1,out,IDnodoorigen2,Coordenadanodoorigen2,in,IDnododestino2,Coordenadanododestino2,…..]……[IDsemaforN1,IDnodoorigen1,Coordenadanodoorigen1,in,IDnododestino1,Coordenadanodod`

```
estino1,out,IDnodoorigen2,Coordenadanodoorigen2,in,IDnododestino2
,Coordenadanododestino2,…]]
```

3. Para obtener las coordenadas se usa la función *conexión[índice1].getShape(IncludeJunctions=false[índice2])*, donde *índice1* puede ser 1 o 0 dependiendo de si es nodo origen o nodo destino del carril respectivamente e índice2 puede ser 1 o 0 si es coordenada del punto donde empieza o acaba el carril. Para clarificarlo se puede ver en la Ilustración 11, donde A y D son las coordenadas donde empieza el carril y B y C son las coordenadas donde finaliza el carril.

4. La segunda lista creada tiene el formato:

```
[[IDsemaforo1,IDnodoorigen1,IDnododestino1,IDnodoorigen2,IDnododestino2,…..]……[IDsemaforN1,IDnodoorigen1,IDnododestino1,IDnodoorigen2,IDnododestino2,…]]
```

5. Con dichas coodenadas de cada ruta en el área de la intersección se traza una recta equivalente al recorrido de la ruta, con el que se podrá comprobar la compatibilidad de las rutas calculando si se cortan dichas rectas previamente generadas.

6. A continuación se calcula el punto de corte de las rectas generadas por el punto de entrada y salida de cada ruta; para lograrlo tengo que distinguir dos casos: que las rectas se corten dentro del área del cruce o fuera de este. En caso de que se crucen fuera, es que ambas rutas son compatibles entre sí, ya que no hay problemas de compatibilidad dentro del la intersección, en caso opuesto, que se crucen dentro de la intersección ambas rutas no son compatibles entre sí. Puede ocurrir que también se corten en el borde del área, en cuyo caso hay que distinguir dos casos, si la convergencia de las rectas se produce en carril de salida de la intersección o en un carril de entrada, si es una entrada ambas rutas serán compatibles, ya que partirían del mismo carril, y no habría problema de compatibilidad, en cambio si las dos rectas convergen en una salida, no serán compatibles. En la Ilustración 12 se muestra un ejemplo de cruce en el que se puede apreciar gráficamente qué rutas son compatibles y cómo se parametriza dicho cruce para su procesamiento.

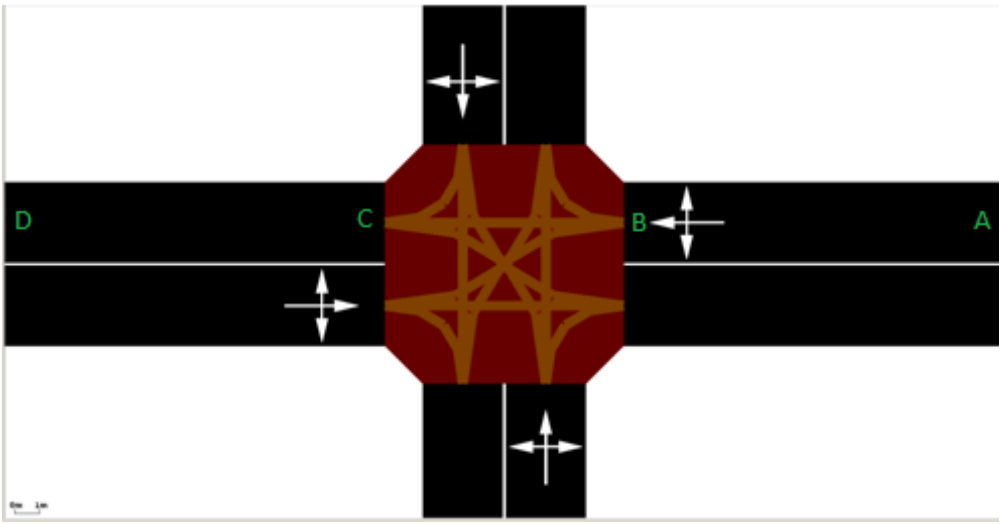

Ilustración 11 Indicación coordenadas del carril

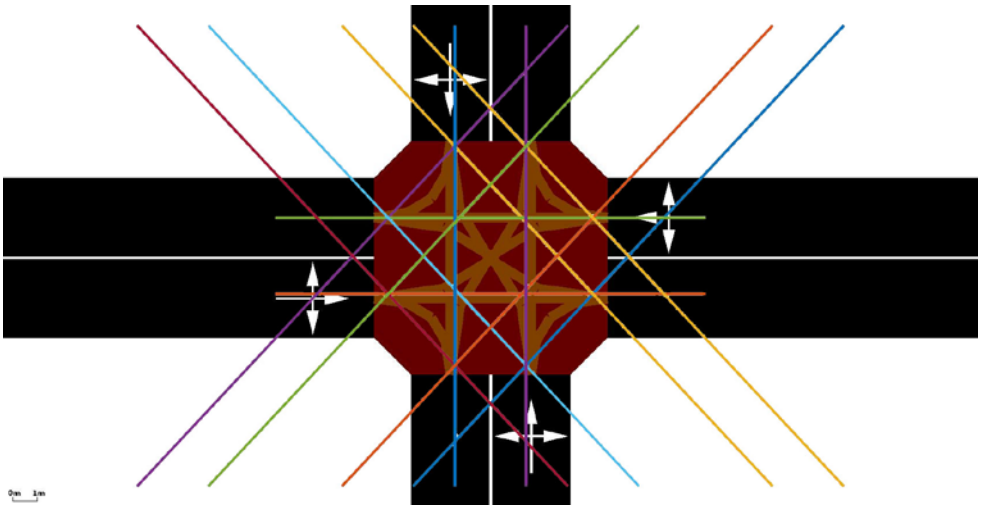

Ilustración 12. Cruce con rectas calculadas

El algoritmo usado para calcular los puntos de corte de las rutas tiene que evitar la comparación de una ruta consigo misma, además de que como en la comparación el orden de las diferentes comparaciones (ruta uno con la dos o ruta dos con la uno) no es relevante, hay evitar hacer dos veces la misma comparación. Con todo esto se conseguiría un algoritmo que necesita la mitad de iteraciones para resolver esta cuestión. Finalmente el algoritmo utilizado es el siguiente:

```
minimo=1
maximo=7
k=minimo
l=k+1
i=minimo
```

```
j=i
while k<maximo:
    j=k+1
    l=k+1
    while l<maximo:
        j+=1#esto es para ir recorriendo los elementos
        l+=1#esta es la condición para salir del segundo while
    i+=1#esto es para ir recorriendo los elementos
    k += 1#este elemento es la condición para que termine el primer while
```

Como se puede ver en el código, dicho algoritmo de comparación cumple las condiciones necesarias expuestas anteriormente. Una vez procesados todos los datos necesarios, obtendremos una matriz simétrica que ordena la información de la comparación de la siguiente manera: cada fila y cada columna corresponde a la ruta de igual número, así la fila 1 corresponde a la ruta 1, la fila 2 a la dos y así sucesivamente. Por lo tanto la fila 1 con la columna 2 o la fila 2 con la columna 1 nos dirían si la ruta 1 y 2 son compatibles entre sí. Por ello la matriz es simétrica. Se indica con un uno si las rutas son compatibles, y con un cero si no lo son. Una vez obtenidas estas rutas, esto nos servirá como base para crear las fases necesarias en un semáforo.

1	1	0	1	1
1	1	0	1	0
0	0	1	0	1
1	1	0	1	0
1	0	1	0	1

Ilustración 13. Matriz de rutas compatibles

Se obtendrá una matriz similar a la que se puede ver en la Ilustración 13, en la que se indica, por ejemplo, en la primera fila que la ruta 1 es compatible con la 2 la 4 y la 5. Es importante resaltar que no todas las rutas compatibles con la ruta 1 son compatibles entre sí, como por ejemplo la ruta 2 con la ruta 5. Esto deberá ser tenido en cuenta en la siguiente fase de solución del problema.

4.4 Creación de las fases de los semáforos y fases mínimas

Para cada cruce se van a crear las fases mínimas necesarias para que se abran al menos una vez todas las rutas. Para esto, una vez obtenida la matriz de rutas compatibles, procesando dicha matriz se comprueba qué rutas son compatibles totalmente entre sí, ya que la matriz anterior indica únicamente una relación entre solo dos rutas, es decir, no garantiza

que todas las rutas que hay en una fila sean compatibles entre sí. Por lo tanto, una vez procesada la matriz original se obtiene una nueva matriz cada una de cuyas filas indica todas las rutas que son compatibles entre sí, formando así una posible fase o estado de los semáforos que se podría utilizar dentro del ciclo. Posteriormente se procede a buscar el número de fases mínimas para formar un ciclo completo. Este problema no se puede resolver por fuerza bruta, dado que requeriría todas las combinaciones posibles de las distintas fases del ciclo calculado, que como depende de una combinación más una permutación hace que rápidamente tienda a infinito. Por ello acudimos a una heurística en la que se coge primero la fase que más rutas abre a la vez. Si hay dos fases que abren el mismo número de rutas, entonces la selección de una de ellas se puede considerar aleatoria, aunque en realidad lo que se hace es coger la primera que está en la matriz ya calculada. A partir de este punto el algoritmo itera, lo siguiente que hace calcular las filas que más rutas abre de las que todavía no han sido abiertas al menos una vez en el ciclo, añadiéndose una nueva fase hasta completar todas las rutas. Una vez obtenidas las fases mínimas se comprueba que orden de las fases minimiza el número de cambios de rutas abiertas entre fases, esto es, dado que hemos ido escogiendo filas que maximizaban el número de rutas que faltaban por abrir, dentro de una fila escogida a partir de la segunda iteración puede reaparecer como abierta una ruta que ya lo estuvo anteriormente. Lo ideal es que una ruta que aparezca dos o más veces en un ciclo como abierta se mantenga en este estado de forma constante. Para conseguir esto se hacen todas las permutaciones de la matriz de fases mínimas y tomamos aquella permutación con menos cambios de rutas abiertas o cerradas entre fases.

4.5 Extracción de los datos durante la simulación y procesado de estos en tiempo real

Inicialmente, se identifican todos los semáforos que hay en la red (como se vio en la sección 3.8 y cuyo código puede verse en la sección 9.10). Seguidamente se descartan los semáforos que solo abren una única ruta. Esto se hace para solucionar un problema que es causado al exportar un mapa real, ya que en la exportación se eliminan ciertas calles y deja únicamente un semáforo en una calle cortada. Los semáforos obtenidos son almacenados en una lista, y dentro de esta misma lista a cada semáforo se le asocian unos aros inductivos teniendo en cuenta cada carril que servirán para contar el número de coches que entran y salen de la intersección controlada por el semáforo tal como se explicó en la sección 3.8 y 9.10. Dicha lista tiene el siguiente formato:

```
[[IDsemaforo1, IDloopA, IDloopB ,…..], …….,[IDsemaforoN , IDloopJ , IDloopK]]
```

Como se puede ver, en la lista se almacenan varias listas, una por cada semáforo de la red. Cada sublista contendrá el ID del semáforo, junto con los ID's de los loops que tiene asociados.

En caso de que haya más de un aro inductivo por carril (tal y como se explicó en la sección 3.6) el criterio usado para seleccionar uno de ellos es: si son aros inductivos de entrada a la intersección, permanecerá el más lejano a dicha intersección para tener una mayor exactitud de los datos del número de vehículos que van a entrar en el cruce; en cambio si son aros inductivos de salida permanecerá el más cercano a la intersección para tener la información de salida de los vehículos lo más actualizada posible. Sin embargo se plantea un problema, dado un vehículo que ha entrado por un carril concreto en el cruce, y teniendo varias rutas disponibles desde ese cruce ¿cómo se puede distinguir un vehículo por dónde ha salido? En realidad este no es un problema por la forma en la que se construyó la matriz de fases mínima, dado que no hay rutas incompatibles en las llegadas, cada coche de salida se sabe de forma unívoca desde qué carril ha salido, de esta forma la información obtenida es fidedigna para poder calcular la densidad de tráfico de cada ruta concreta en el cruce. Por lo tanto las estadísticas de flujo de vehículos en realidad no dependen del identificador de vehículo que hemos utilizado en el programa por simplicidad, sino que se podría calcular en el mundo real directamente a través de los datos en bruto de los aros inductivos sin dicho identificador.

Posteriormente tras la obtención de todos estos datos ya se puede proceder a la modificación de los de las fases de los semáforos.

5

Análisis de resultados

5.1 Introducción

El objetivo de este capítulo es demostrar que con el programa desarrollado es posible generar automáticamente a partir de un escenario base distintas políticas de control de semáforos. Por ello, en este capítulo se va a crear una política de control de semáforos en función de la densidad de tráfico, dicha política tomará decisiones de concesiones de tiempo a cada fase del semáforo. Cabe indicar que es posible implementar otro tipo de políticas haciendo uso de la interfaz existente. En la política implementada inicialmente a las fases se les asigna un tiempo fijo al que a posteriori se le añadirá un tiempo extra en función de la densidad de tráfico, es decir, se hace una estimación de la densidad de tráfico del ciclo siguiente a partir de los datos de una o varias densidades de ciclos anteriores. Dicho tiempo extra se obtiene de restar al tiempo de la fase los tiempos de las fases en ambar y de los tiempos fijos asignados. Para llevar esto a cabo se almacenan los datos obtenidos de los aros inductivos y se almacenan tantos ciclos como se deseen, a continuación se convierten dichos datos a densidad ya que se comparan con los tiempos de una o varias fases de los datos almacenados como se explicó anteriormente, después se normalizan los datos del número de vehículos obtenidos para cada ruta, posteriormente se obtiene el porcentaje sobre el total a partir de dichos valores normalizados para ya asignar mediante este porcentaje el tiempo restante disponible extra que hay en el semáforo.

Para dicha política se evaluarán diferentes escenarios en los no habrá ningún control de fase, control de fase mediante densidad en que se variará el porcentaje de ciclo fijo, el tiempo de ciclo, el número de ciclos de almacenamiento de datos para la previsión de la densidad, además de plantearan diferentes flujos de tráfico para una misma red de carreteras y diferentes tipos de carreteras

5.2 Estrategia de establecimiento de tiempos de cada fase

Inicialmente se tiene la información del número de vehículos que han pasado por cada ruta de cada semáforo según se vió en la sección 4.5. Además como se explicó anteriormente en la sección 3.6, se conocen los tiempos relativos a la simulación en que se obtuvieron dichos datos. Con todo esto se puede calcular la densidad de tráfico, esto es, el número de vehículos entre el tiempo total de adquisisión de los datos de los vehículos. El objetivo de esta política de semáforos es igualar las densidades de tráfico de todas las rutas, suponiendo que esto permitirá que el tiempo medio de viaje baje. Hay que fijarse que para minimizar el tiempo de viaje global se necesitaría coordinación entre los diferentes semáforos y utilizar la información concreta de cada vehículo a escala global, pero esto implicaría una política mucho más compleja y con problemas para garantizar la privacidad de los usuarios. Políticas que no necesitaran información concreta de los vehículos y que de alguna forma son equivalentes a la que proponemos aquí pueden ser como las aplicadas en SCOOT, en las que se minimiza el tamaño medio de las colas.

Para igualar la densidad de tráfico entre rutas se aumentará el tiempo de las fases con una mayor densidad de tráfico. La otra opción para lograrlo es disminuir el número de vehículos, la cual en principio no es viable. Cabe indicar que la lista de vehículos por ruta para el cálculo de la densidad actúa como una ventana deslizante quedándose con los últimos ciclos indicados por el parámetro *periodociclos* para que así solo haya influencia de la información obtenido en los esos últimos ciclos. Para contar los ciclos que han pasado entre la adquisición de los vehículos por ruta se ha introducido el carácter # en la lista de vehículos por ruta con el formato:

```
[IDsemáforo,[IDvehiculo1,IDvehiculo2,…,#,IDvehiculoN,#]…[IDvehiculoK,#,IDvehiculoK+1,…]].
```

En la lista anterior el IDvehiculo indica qué vehículos pasaron a lo largo de diversos pasos del simulador. Varios de estos de estos componen un clico completo de semáforo. Nuestra ventana deslizante permite calcular la densidad de tráfico agregando los vehículos que pasaron en un número concreto de ciclos de semáforo, por lo tanto es importante distinguir qué vehículos pasaron en un determinado ciclo. Para ello, se ha introducido el carácter # en la lista que cada vez que aparece indica que empieza un ciclo de semáforo nuevo, permitiendo implementar de esta manera la ventana deslizante.

5.3 Cálculo de tiempos según la densidad

Para calcular los tiempo asignados a cada fase de semáforo según la densidad de tráfico se utiliza la función *calculartiempospordensidad()*. Dicha función crea un hilo por cada semáforo y calcula para cada semáforo los tiempos de las fases mínimas creadas como se explicó en la sección 4.4, para ello la función necesita que se establezcan unos tiempos de las fases ámbar (fases intermedias), el tiempo fijo total del ciclo y el porcentaje de ciclo fijo, que es un parámetro de la simulación y se debe especificar de forma oportuna. Una vez establecidos todos los parámetros el programa comprueba que el tiempo de las fases intermedias no es mayor que el tiempo de la fase total. Si esto se da, el script lanza un error diciendo que los tiempos de las fases ámbar no permite el ciclo. Posteriormente se recorre la lista de vehículos por ruta para cada semáforo, en ella se cuenta el número de vehículos que han pasado y se obtiene un vector con estos. Una vez obtenido este vector se normaliza y al multiplicarlo por cada fase (dichas fases tienen un valor 0 o 1 para las rutas cerradas o abiertas respectivamente) obtenemos un vector de pesos de tiempo de cada fase, estos pesos los pongo en base a un porcentaje del peso total, y así se sabría qué porcentaje del tiempo de ciclo disponible tengo que asignar a cada fase.

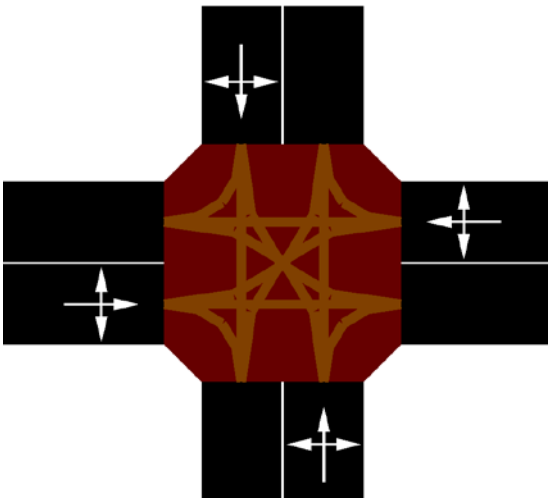

Ilustración 14. Único cruce para varios sentidos por carril

5.4 Escenario 1: Un único cruce con varios sentidos por carril para diferentes flujos de tráfico

En este escenario se utiliza el mapa de red de la Ilustración 14. Como se puede observar en este mapa cada carril tiene varios sentidos. Es importante señalar aquí, que las

reglas de tráfico implementadas tendrán en este caso específico un impacto muy relevante. Habrá ciertos giros que estén prohibidos hasta que la fase concreta de semáforo lo permita, entonces, aunque por ejemplo el semáforo esté en verde para seguir hacia delante, si un vehículo en primera posición quiere girar hacia la izquierda, no lo podrá hacer, retrasando a todos los que sí que podrían circular de frente. Esto se puede considerar como un fallo de las reglas de tráfico que se han implementado, pero demuestra que en el método de resolución del problema elegido se ha desacoplado totalmente las reglas de tráfico de la política de semáforo.

Para este caso se evaluarán los siguientes escenarios:

1. Vehículos en una única dirección y sentido.
2. Vehículos en una dirección y diferentes sentidos.
3. Vehículos en diferentes direcciones y único sentido por cada dirección.
4. Vehículos con tráfico aleatorio generado mediante el script *randomtrips.py* (más información sobre el script en la sección 3.4.)

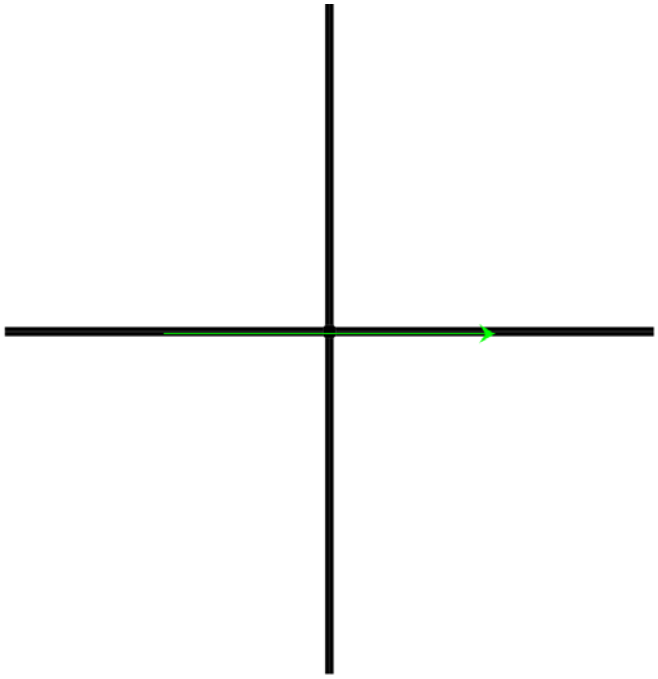

Ilustración 15. Vehículos en una única dirección y sentido

5.4.1 Vehículos en una única dirección y sentido

Para este caso se puede ver que los vehículos tienen un único nodo origen y destino como se ve en la Ilustración 15. En este caso se ha utilizado un único flujo de 6000 vehículos con tasa de generación constante con inicio y fin de generación 0 y 60 segundos. Los resultados obtenidos relativos al tiempo medio de viaje están en Ilustración 16. En dicha imagen podemos ver en el eje de abscisas el instante de simulación en segundos y el tiempo medio de viaje en el eje de ordenadas en segundos. La información que proporciona la gráfica se corresponde con la evolución del tiempo medio de viaje de todos los vehículos que han finalizado su trayecto hasta el instante del eje de abscisas de la gráfica. Se han realizado simulaciones para los casos de tiempo de la fase ámbar de 3 segundos para las fases estáticas y dinámicas, duración de la fase de 90 segundos para también ambos tipos de fases, periodo de almacenamiento de la información de 2 y 10 ciclos de semáforo y porcentaje del ciclo fijo del 30 , 50 y 100 por ciento. En el caso en que no se utiliza ninguna política de semáforos (defecto) se puede apreciar que la curva de tiempo medio crece hasta estabilizarse alrededor de 140 segundos de tiempo medio de viaje. En cambio para los otros dos casos se consigue una mejora en el tiempo medio de viaje una vez estabilizado, con un resultado de 100 segundos para el 50% del tiempo de ciclo fijo y de 70 segundos para el caso de tiempo de ciclo fijo del 30%. Cabe destacar que el pico inicial que experimenta el tiempo medio para los casos en que se utiliza la política de semáforos dinámica es debido a que el programa no empieza a actuar hasta que finaliza la primera fase que tiene duración de 90 segundos. Para mayor completitud del análisis se ha incluido una simulación con las fases creadas automáticamente pero en la que no se utilizan tiempos dinámicos (100% del tiempo de ciclo fijo), sino que distribuye el tiempo de ciclo disponible de manera equitativa entre todas las fases.

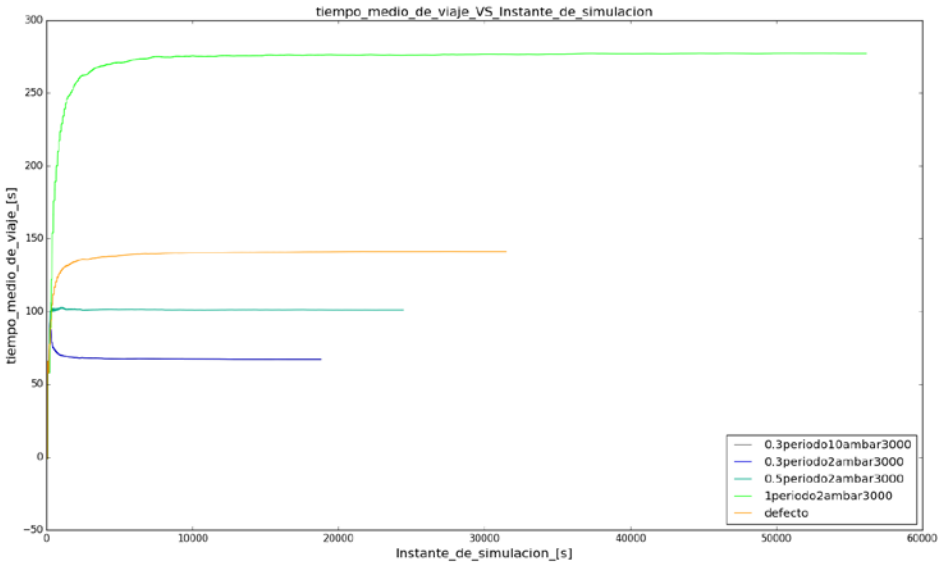

Ilustración 16. Tiempo medio de viaje VS instante de simulación para vehículos de una dirección y sentido

5.4.2 Vehículos en una dirección y diferentes sentidos

En este caso los vehículos parten de un mismo nodo origen pero tienen diferentes nodos destino como se puede ver en la Ilustración 17. Para la simulación se utilizan los mismos parámetros que en la sección 5.4.1 Aquí se han establecido dos flujos, ambos de 3000 vehículos con una tasa de generación constante con inicio y fin de generación del flujo en 0 y 60 segundos. Los resultados obtenidos se pueden ver en la Ilustración 18, como se puede apreciar en este caso el tiempo medio de viaje por defecto se mantiene respecto a la sección anterior, esto es debido a que los ciclos de los semáforos son iguales en ambos casos y que el número de vehículos es el mismo que en el caso anterior, en cambio cuando se hace uso de la política de semáforos el tiempo medio de viaje crece, esto es causado porqué al haber un único carril por sentido para una misma dirección los vehículos que están a continuación del primer vehículo tienen que esperar a que se abra la fase del primer vehículo más la suya propia ya que estas pueden estar diferentes fases aunque partan del mismo nodo origen y por tanto aumenta el tiempo medio de viaje.

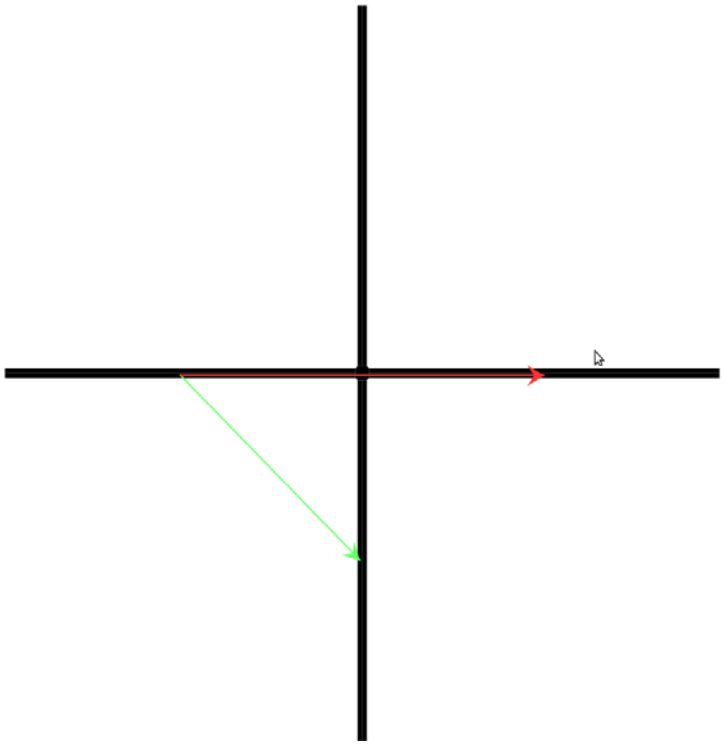

Ilustración 17. Vehículos en una única dirección pero diferentes sentidos

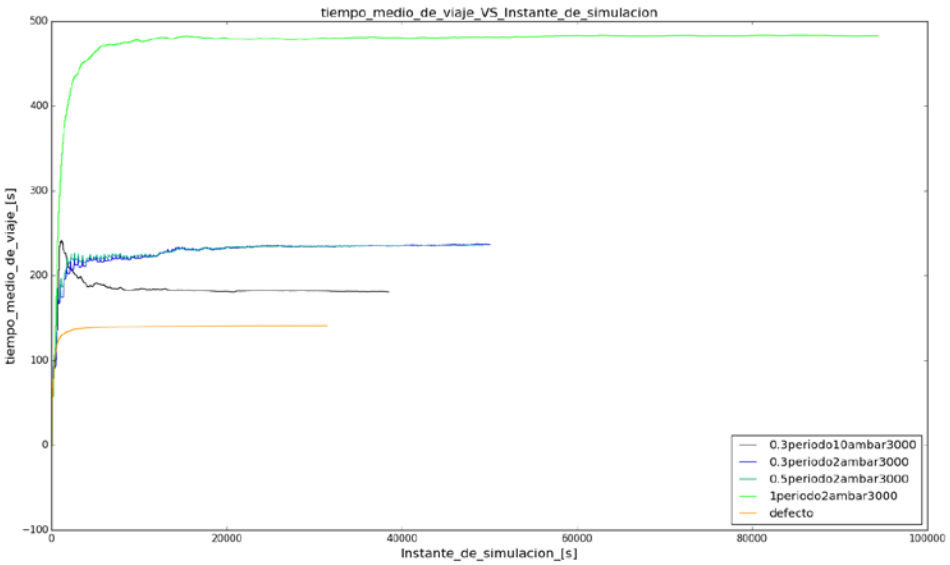

Ilustración 18. Tiempo medio de viaje VS instante de simulación para misma dirección pero diferentes sentidos

5.4.3 Vehículos en diferentes direcciones y único sentido por cada dirección

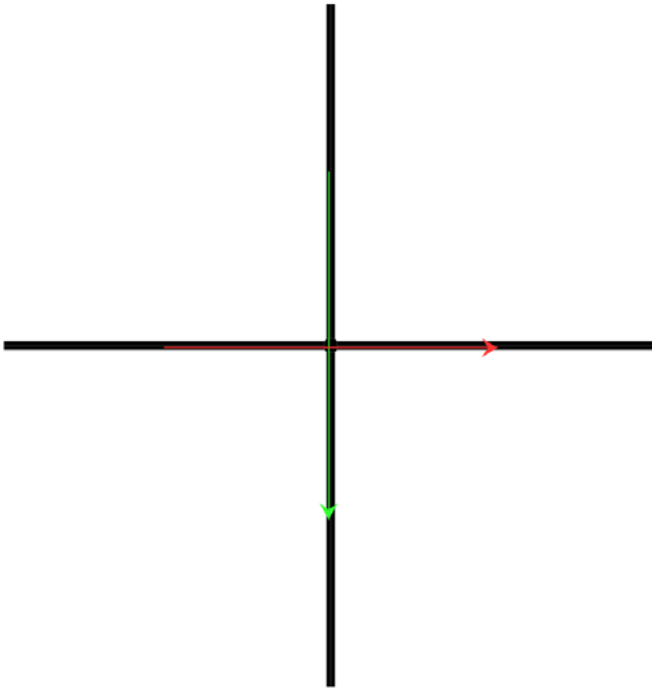

Ilustración 19. Vehiculos con dos direcciones y un único sentido por dirección

Para el caso de dos flujos de vehículos que se puede ver en la Ilustración 19, en el flujo verde se insertan 6000 vehículos con tasa constante entre el instante 0 y 60 de la simulación, el flujo rojo es un flujo de 600 vehículos con inicio en el instante 9000 y final en el instante 9060. Para la simulación se utilizan los mismos parámetros que en la sección 5.4.1. En la Ilustración 20 se puede ver que inicialmente progresa como se ha explicado en el apartado 5.4.1, sin embargo para el caso de la política de semáforos con tiempo dinámico se produce una subida del tiempo medio de viaje en el instante 9000. Esto es debido a la inserción de ese segundo flujo verde, una vez que termina el flujo verde el tiempo medio de viaje empieza a decrecer, esto se manifiesta en la gráfica en forma de un pico, si se comparan todos los casos de la Ilustración 20, el mecanismo más efectivo entre los evaluados para reducir el tiempo medio es el de 30% de periodo fijo y 2 ciclos de almacenamiento de información. Esto es debido a que al haber más tiempo dinámico se permite gestionar mejor los tiempos de las fases para reducir el tiempo medio, además se puede ver la influencia en el almacenamiento de los datos para calcular los tiempos de las fases ya que a menor número de ciclos de almacenamiento el mecanismo de tiempos responde de manera más rápida. Sin embargo esto no es adecuado si se producen pequeños flujos de vehículos esporádicos en comparación con el flujo de mayor densidad de tráfico ya que puede perjudicar al flujo

principal. Para el mismo caso pero en el que el flujo de vehículos verde es de 3000 vehículos en vez de 600, su gráfica es la Ilustración 21, aquí se puede comprobar la influencia del periodo en función del tamaño de flujo, ya que el flujo verde ya es considerable en comparación con el flujo rojo.

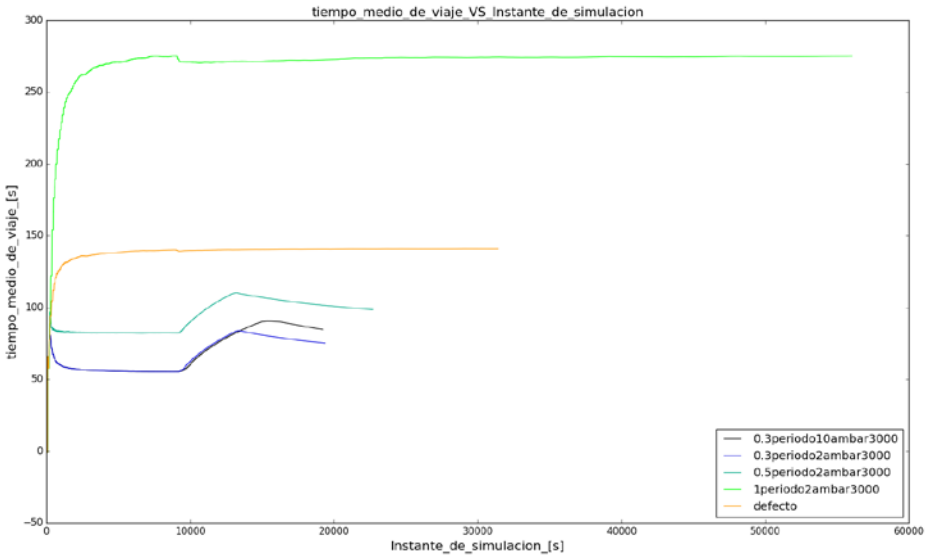

Ilustración 20. Tiempo medio de viaje VS instante de simulación para dos direcciones y un único sentido por dirección

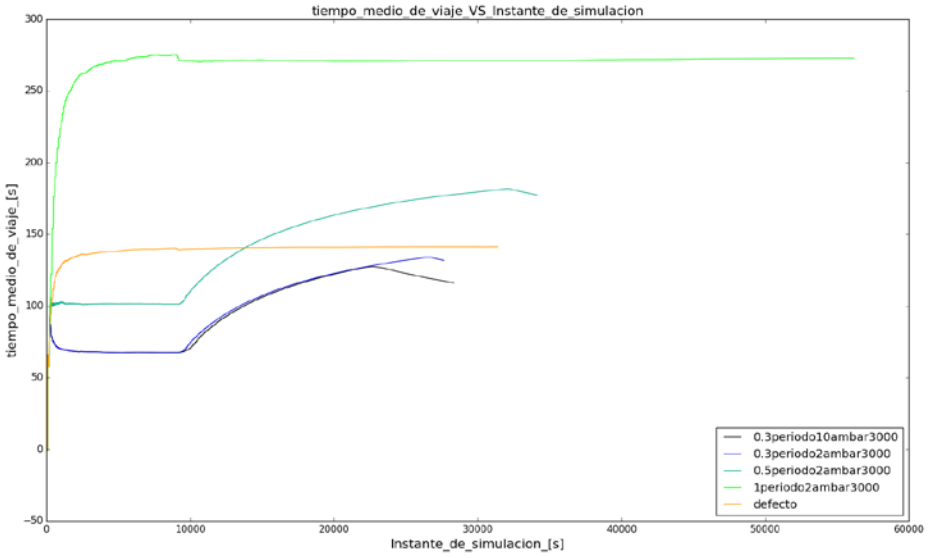

Ilustración 21. Tiempo medio de viaje VS instante de simulación para dos direcciones y un único sentido por dirección para mayor flujo

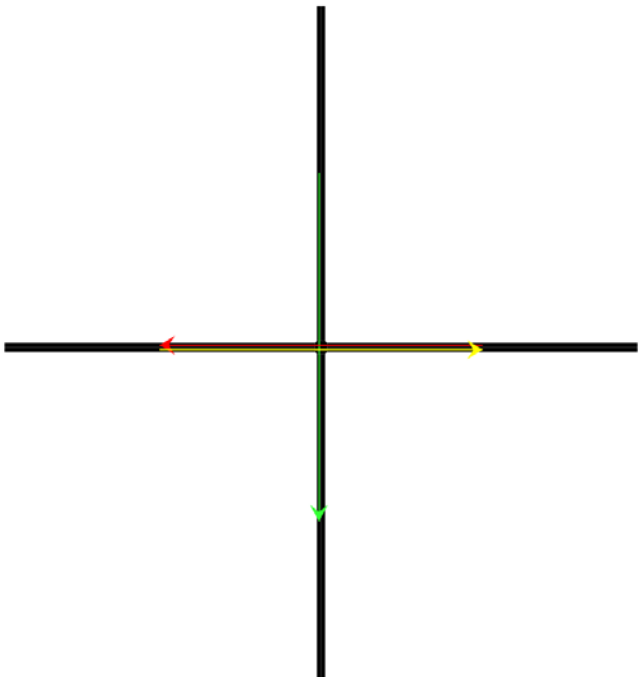

Ilustración 22. Vehiculos en tres direcciones y un único sentido por dirección

El caso de tres direcciones y un único sentido por dirección es similar al caso de dos direcciones. Para la simulación se utilizan los mismos parámetros que en la sección 5.4.1. En este caso se ha propuesto dos flujos de vehículos: uno en que los tiempos de dos de los flujos minoritarios se solapan en tiempo. El flujo amarillo empieza en 0 y acaba en 60, el flujo rojo empieza en 4000 y acaba en 4020 y el flujo verde empieza en 5000 y acaba en 5040 para 6000, 400 y 600 vehículos respectivamente, todos los flujos tienen una tasa de generación constante de vehículos. En la Ilustración 24 se pueden ver los resultados para este escenario. En cambio en la Ilustración 23 dichos flujos minoritarios no se solapan. El flujo amarillo empieza en 0 y acaba en 60, el rojo empieza en 5000 y se acaba en 5040 y el verde empieza en 9000 y acaba en 9060 con 6000, 400 y 600 vehículos respectivamente. En el caso solapado se puede apreciar como el tiempo medio crece cuando se inserta el primer flujo minoritario e incrementa aún más cuando se introduce el segundo flujo de este tipo, para el caso de mayor periodo de almacenamiento de datos se puede ver una evolución más lenta de tiempo medio. En cambio para el caso no solapado se aprecia una transición menos abrupta del tiempo medio al introducir menos tráfico simultáneo en la intersección.

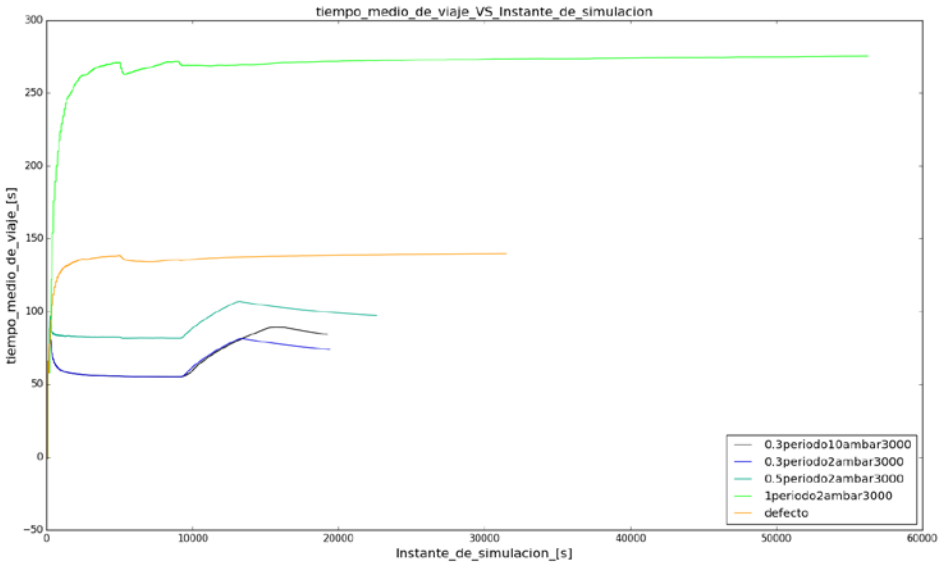

Ilustración 23. Tiempo medio de viaje VS instante de simulación para tres direcciones y un único sentido por dirección con vehículos en instantes no solapados

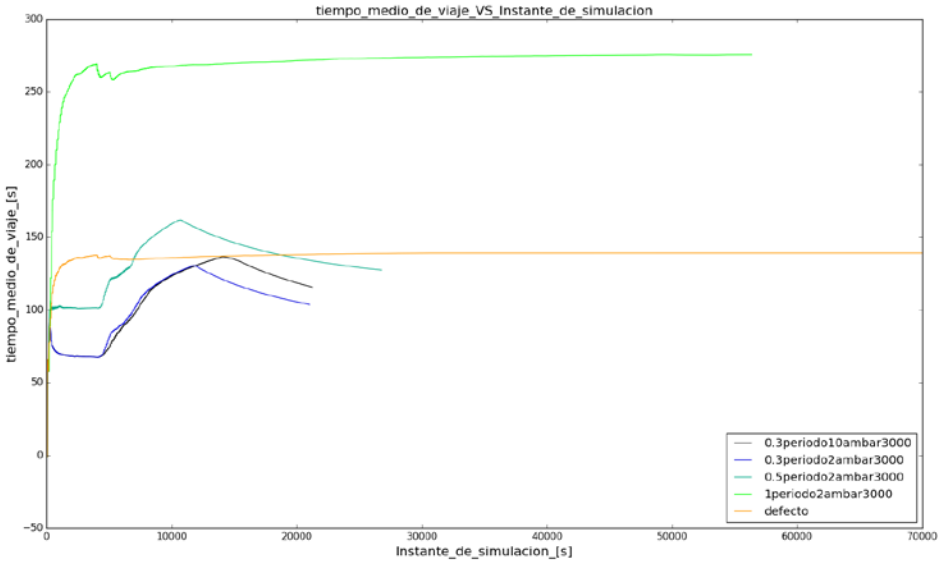

Ilustración 24. Tiempo medio de viaje VS instante de simulación para tres direcciones y un único sentido por dirección con vehículos en instantes solapados

5.4.4 Vehículos con tráfico aleatorio generado mediante *randomtrips.py*

Para este caso el flujo de vehículos es generado mediante el script *randomtrips.py* y las opciones *–l –e 3600* (ver sección 3.4 para más información sobre este script). Los resultados obtenidos se pueden ver en la Ilustración 25, dicha gráfica parece sugerir que para el caso dinámico la política utilizada no afecta al tiempo medio de viaje ya que todas las curvas se comportan de manera similar, en cambio el caso estático es el primero en finalizar la simulación.

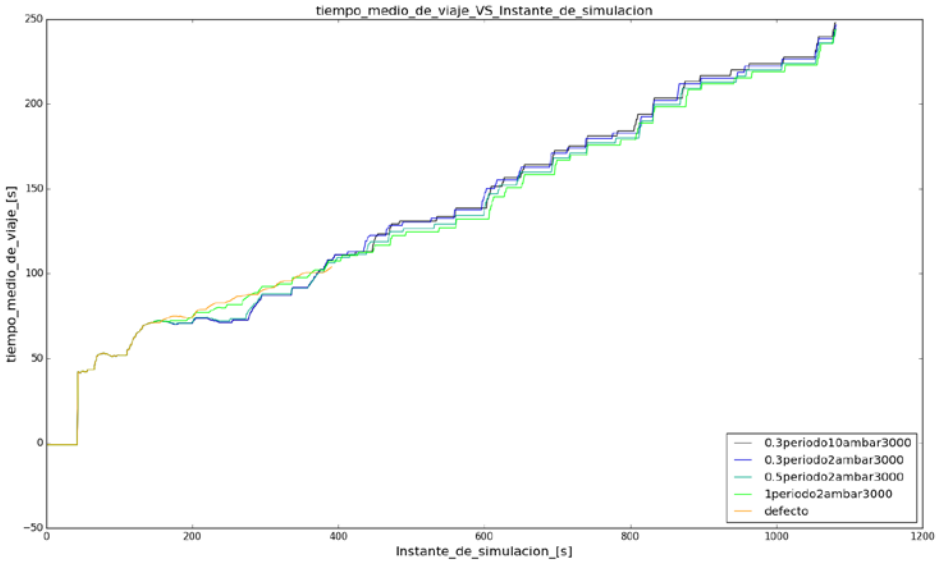

Ilustración 25. Tiempo medio de viaje VS instante de simulación para rutas generadas mediante randomtrips.py

5.5 Un único cruce con un sentido por carril para diferentes flujos de tráfico

En este caso se usa el cruce que se puede ver en la Ilustración 26. Se evaluarán situaciones similares a las del apartado anterior con los parámetros mencionados en la sección 5.4.1.

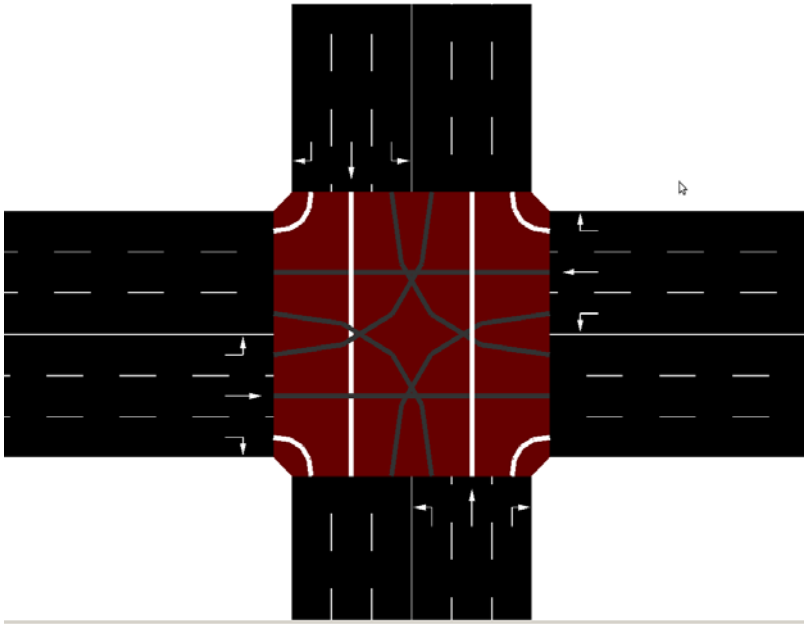

Ilustración 26. Cruce con un sentido por carril

5.5.1 Vehículos en una dirección y diferentes sentidos

Para el caso de tres flujos de vehículos en el que todos los flujos que comparten un mismo origen pero tres destinos diferentes tal y como se ve en la Ilustración 27, cada flujo de vehículos genera 1000 vehículos entre instantes 0 y 60 con tasa de generación constante y con características de las fases mencionadas en la sección 5.4.1. Los resultados obtenidos se reflejan en la Ilustración 28. Estos resultados a primera vista pueden parecer incoherentes con los resultados obtenidos en la sección 5.4.3 ya que el tiempo medio de viaje con las políticas dinámicas crece respecto a la estática, sin embargo cabe destacar dos causas de ello:

- El flujo de vehículos generado se acumula antes de la intersección en varios carriles, por tanto los vehículos tienen que cambiar a un único carril para dirigirse en un sentido, esto se ve reflejado en un incremento relevante del tiempo medio de viaje. Se puede ver un ejemplo de esta situación en la Ilustración 29, en este caso todos los vehículos rojos quieren girar a la izquierda, por lo que todos tienen que colocarse al carril izquierdo para realizar dicho giro. Este hecho también perjudica a la predicción del tráfico ya el vehículo se puede incorporar al carril apropiado casi al final de la longitud de este y no pasar por el detector correcto lo que inserta información errónea al sistema.

- Si la longitud del carril es pequeña un menor número de vehículos puede ser medido con los aros inductivos (para mayor información sobre la posición de los aros inductivos elegida consultar la sección 4.5). Este problema afecta a la preción de los datos obtenidos y por tanto a la decisión de las fases del semáforo. Se puede ver un ejemplo en la Ilustración 29 en que el aro inductivo más lejano solo permite medir a lo sumo 6 vehículos.

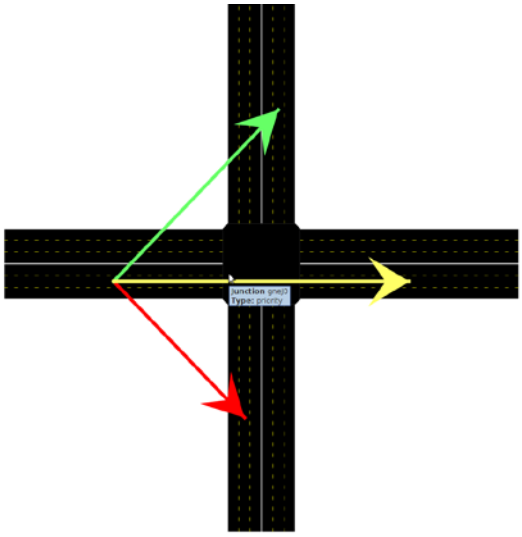

Ilustración 27. Flujos de vehículos generados para el cruce

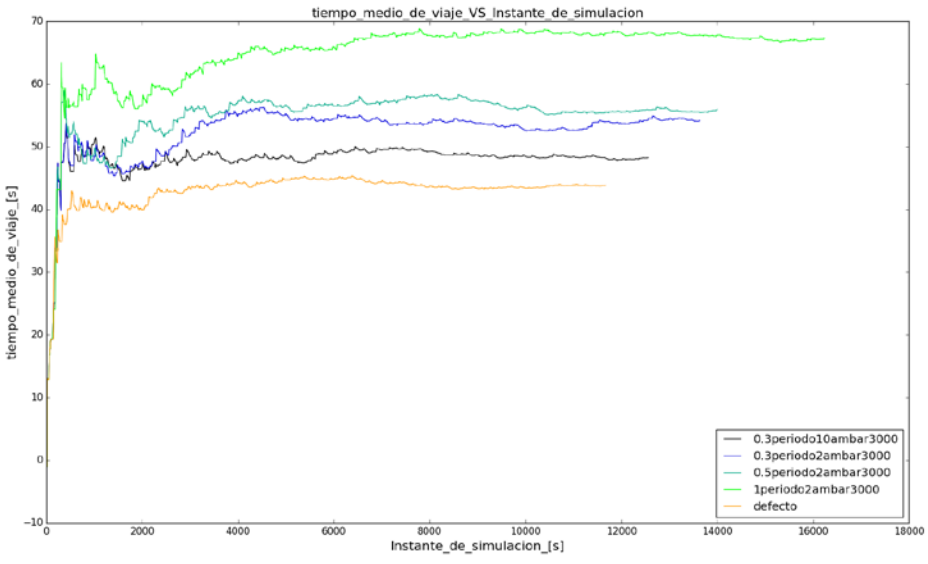

Ilustración 28. Tiempo medio de viaje VS instante de simulación para vehículos en una dirección y diferentes sentidos del cruce

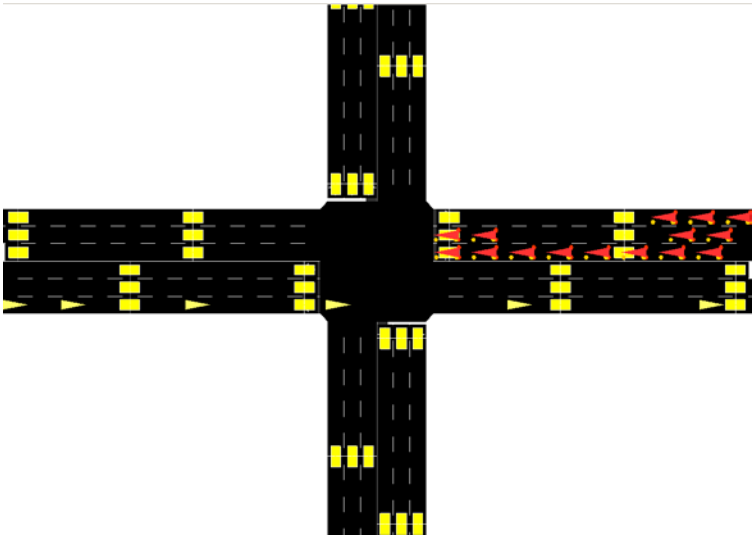

Ilustración 29. Ejemplo de congestión para girar a la izquierda

5.6 Flujos de vehículos para una red con estructura regular

En este caso se ha generado un fichero *.net* mediante el comando *netgenerate –g --grid.x-number 3 --grid.y-number 3 --grid.x-length 150 --grid.y-length 150*; con esto se genera una red en malla como se puede ver en la Ilustración 30. Dicha red tiene un conjunto de cinco semáforos. Para este caso se han simulado dos situaciones:

- Dos flujos iguales de vehículos con una misma dirección
- Dos flujos iguales de vehículos con una misma dirección y un flujo de mayor densidad perpendicular a ambos.

5.6.1 Dos flujos iguales de vehículos con una misma dirección

En este caso se ha generado dos flujos como se puede ver en la Ilustración 31. Cada flujo consta de 150 vehículos generados entre 0 y 300 segundos con una tasa de generación constante. Se ha optado por generar el tráfico de esta manera para intentar obtener los datos lo menos desviados posibles debido al error de la herramienta Traffic Modeller mencionado al final de la sección 3.4. Para la política dinámica se ha optado por un tiempo ámbar de 3 segundos, una duración de la fase de 90 segundos y un porcentaje del ciclo fijo del 30 y 50 % junto con un periodo de almacenamiento de 2 y de 10 ciclos. El resultado de la simulación se puede ver en la Ilustración 32. Se puede observar que el tiempo medio de viaje para las

políticas dinámicas decrece respecto a las estáticas ya que hace un mejor uso del tiempo de las fases respecto a la política estática. Como ya se explicó anteriormente en la sección 5.4.1, todas las gráficas permanecen iguales hasta que entra en funcionamiento el sistema dinámico en este caso en 90 segundos.

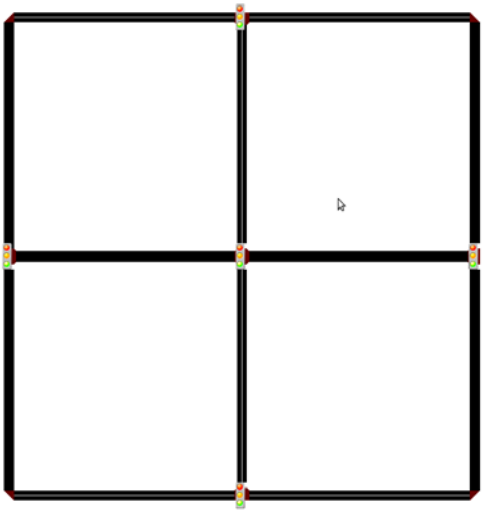

Ilustración 30. Red grid generada con el comando netgenerate

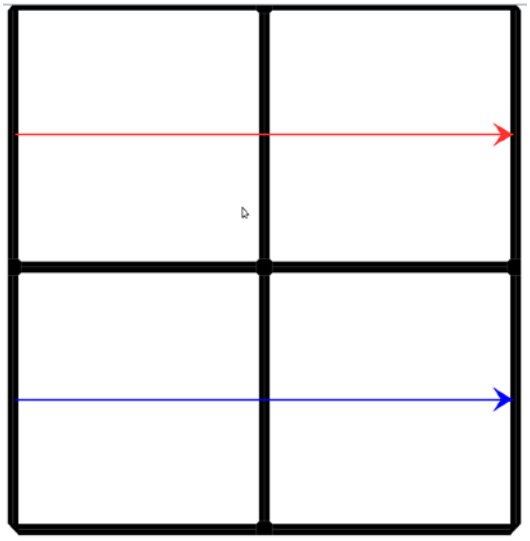

Ilustración 31 Flujos de vehículos iguales con misma dirección

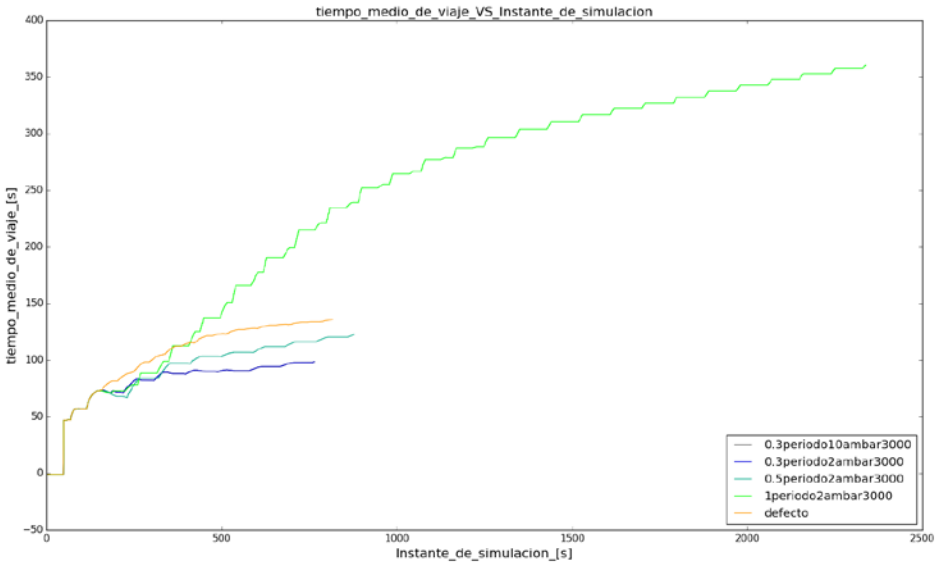

Ilustración 32. Tiempo medio de viaje VS instante de simulación para dos flujos idénticos

5.6.2 Dos flujos iguales de vehículos con una misma dirección y un flujo de mayor densidad perpendicular a ambos con aceleración y deceleración por defecto

En este caso se han generado tres flujos de vehículos como se puede ver en la Ilustración 33, los flujos rojo y azul son flujos de 150 vehículos con inicios en 0 y 200 segundos y finalizan en 300 y 500 segundos respectivamente, en cambio el flujo amarillo es un flujo de 1000 vehículos con inicio en 0 y fin en 1000 segundos, todos los flujos tienen una tasa de generación constante. La aceleración por defecto de SUMO es de 2,6 m/s^2 y una deceleración de 4,5 m/s^2. Además se utilizan los mismos parámetros de la política de semáforos que en la sección 5.6.1. Los resultados de la simulación se pueden ver en la Ilustración 34. En este caso se puede ver que el caso del 50 % varía entorno al tiempo medio de las fases estáticas, este vuelve a crecer cuando finalizan los dos flujos de tráfico menores y solo queda el flujo principal, se puede visualizar una pequeña tendencia de bajada al final de la simulación para el tiempo medio en este caso del 50% del tiempo fijo.

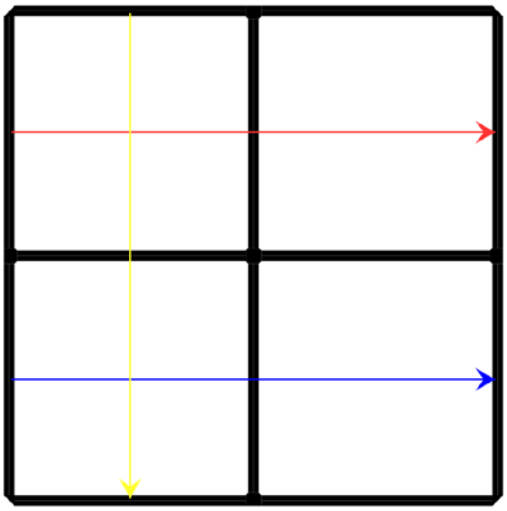

Ilustración 33. Flujos de vehículos iguales en una misma dirección y un flujo de mayor densidad perpendicular a ambos

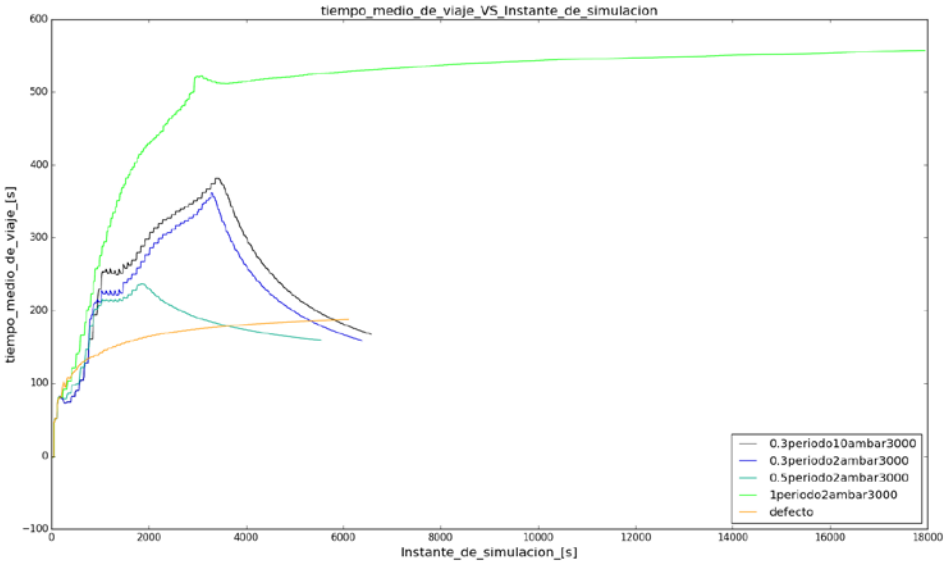

Ilustración 34. Tiempo medio de viaje VS instante de simulación para dos flujos idénticos y uno perpendicular

5.6.3 Dos flujos iguales de vehículos con una misma dirección y un flujo de mayor densidad perpendicular a ambos con aceleración y deceleración menor a la de por defecto

Este caso se realiza bajo las mismas condiciones que la sección 5.6.2, sin embargo ya que la aceleración y deceleración por defecto de SUMO pueden ser un poco elevadas se utilizará para esta simulación una aceleración de 2.3 m/s^2 y una deceleración de 4.2 m/s^2 para adecuar

más los resultados a la realidad [40]. Los resultados de la simulación se pueden ver en la Ilustración 35, en este caso se puede apreciar que las políticas dinámicas sí que reducen el tiempo medio de viaje, además se puede ver claramente la influencia de la longitud del periodo de almacenamiento de datos entre la gráfica negra y azul.

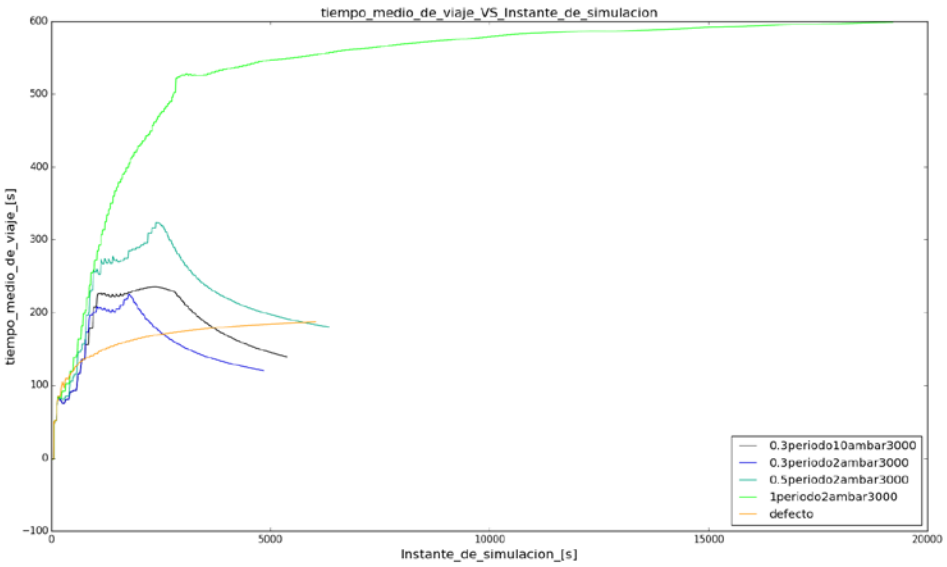

Ilustración 35. Tiempo medio de viaje VS instante de simulación para dos flujos idénticos y uno perpendicular con aceleración y deceleración no por defecto

5.7 Escenario de DIVERT

En este apartado inicialmente se va a reproducir la simulación del artículo [17] con el que contrastaremos los resultados obtenidos. Posteriormente se eligirá un escenario más simple a partir de este en el que se utilizará la política dinámica de semáforos. En el artículo de DIVERT se utiliza un mapa de Brooklyn descargado en formato *OSM*, dicho mapa a continuación se convierte a un fichero *.net* como se explicó en la sección 3.3.1. El artículo crea el tráfico a partir de la herramienta Traffic Modeller (para más información ver sección 3.4) que genera un flujo de tráfico de derecha a izquierda como puede verse en la Ilustración 36. Dicho flujo es de 1000 vehículos, generando el tráfico con una tasa constante de un vehículo por segundo, la velocidad de todas las carreteras es de 13,9 m/s, ello se puede conseguir con el comando *netconvert* usando las opciones *--speed.factor 0 --speed.minimum 13.9*.[31]. Los resultados obtenidos tras la simulación se pueden ver en la Ilustración 37. El tiempo medio de viaje obtenido está en torno a los 4000 segundos, esto es similar a los 3570

segundos obtenidos en el artículo ya que en el artículo no se especifican más parámetros ni área concreta de simulación.

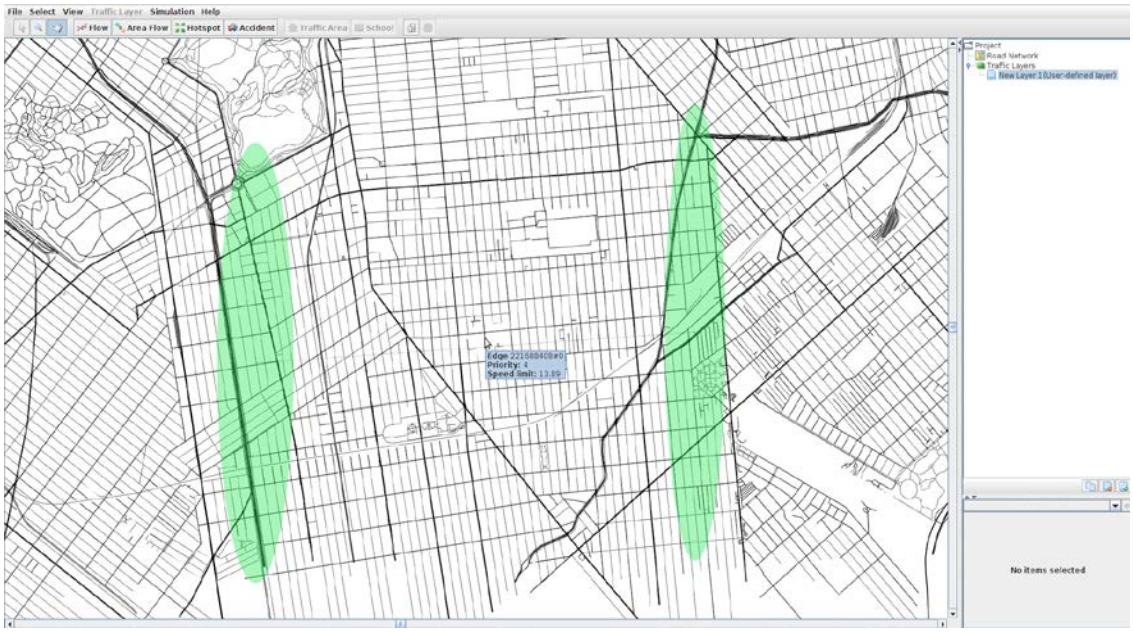

Ilustración 36. Áreas de generación de tráfico en el mapa de Brooklyn

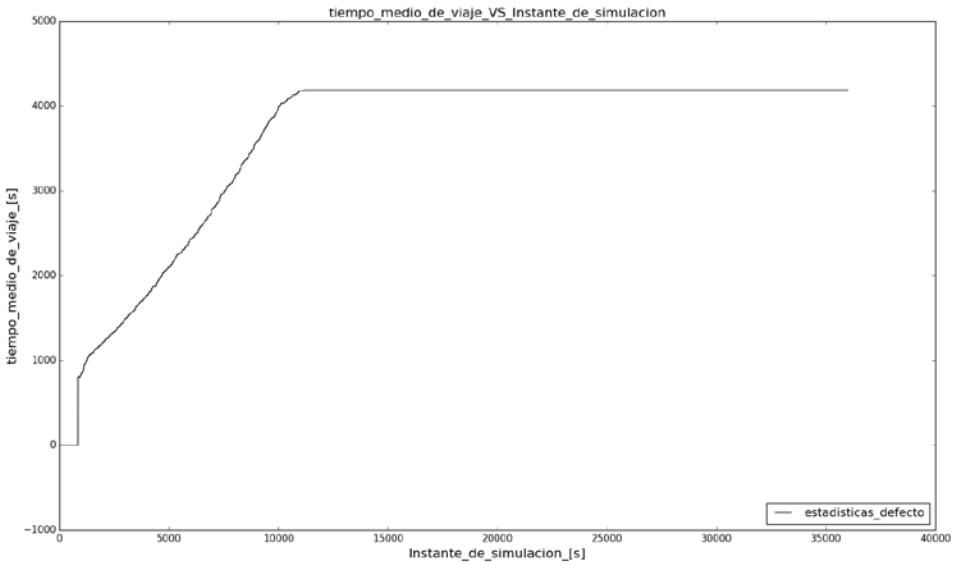

Ilustración 37. Tiempo medio de viaje VS instante de simulación obtenido de la simulación de DIVERT

Debido a las limitaciones en potencia de cálculo para realizar las simulaciones con el mapa de Brooklyn se ha reducido sus dimensiones. En este caso se ha optado por una red de 96 semáforos. La representación de la red se puede ver en la Ilustración 38, y aquí también podemos ver los flujos de tráfico generados, 5 flujos de idénticas características de 50

vehículos cada uno generados con una tasa constante de un vehículo por segundo. Se ha optado por generar el tráfico de esta manera para intentar obtener los resultados lo menos desviados posibles debido al error de la herramienta Traffic Modeller mencionado al final de la sección 3.4. En la Ilustración 39 se muestra el resultado de la simulación para los parámetros tiempo ámbar 3 segundos, tiempo de ciclo 90 segundos, periodo de almacenamiento de datos de 2 y 10 ciclos, porcentaje de ciclo fijo del 30 y 50% y aceleración y deceleración por defecto de 2,6 m/s^2 y 4,5 m/s^2 respectivamente; en cambio en la Ilustración 40 se muestran los resultados para los mismos parámetros pero con aceleración y deceleración de 2,3 m/s^2 y 4,2 m/s^2 respectivamente. Es importante indicar que los resultados vienen afectados por el error mencionado al final de la sección 3.4. Esto se ve reflejado en un crecimiento pronunciado de los tiempos medios de viaje en las gráficas ya que el simulador descarta el vehículo si permanece más de un umbral de tiempo sin moverse, sin embargo en ambos casos antes de que se produzcan el primer crecimiento abrupto en la Ilustración 39 en el instante 1050 para el trazo azul y en la Ilustración 40 en el instante 1300 para el trazo azul los tiempos medios de viaje son menores en los casos dinámicos que en los estáticos (defecto).

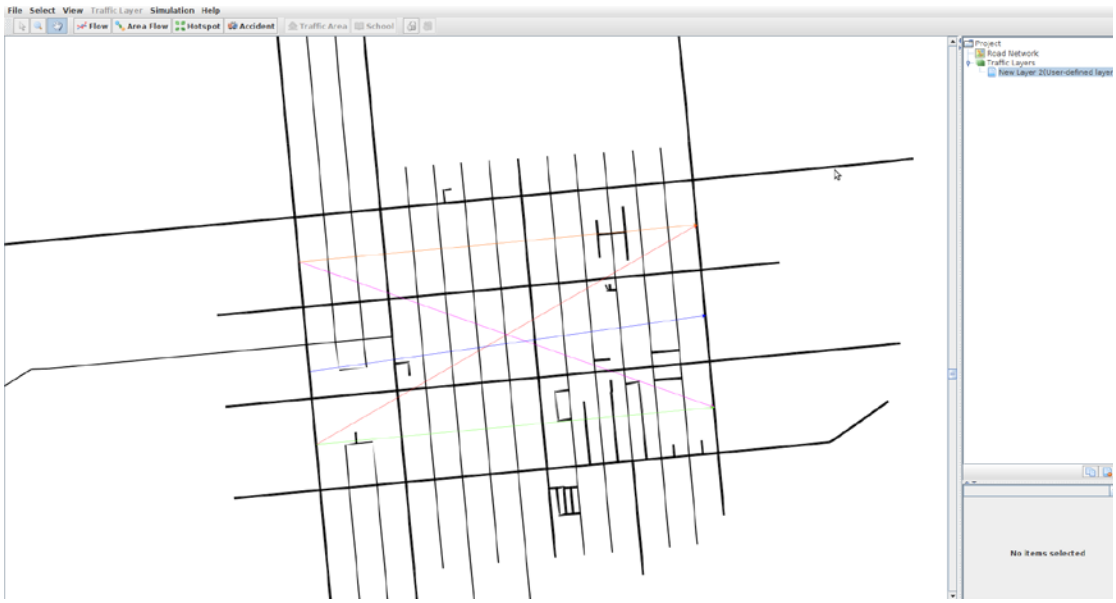

Ilustración 38. Mapa de Brooklyn reducido con las representaciones de flujos generados

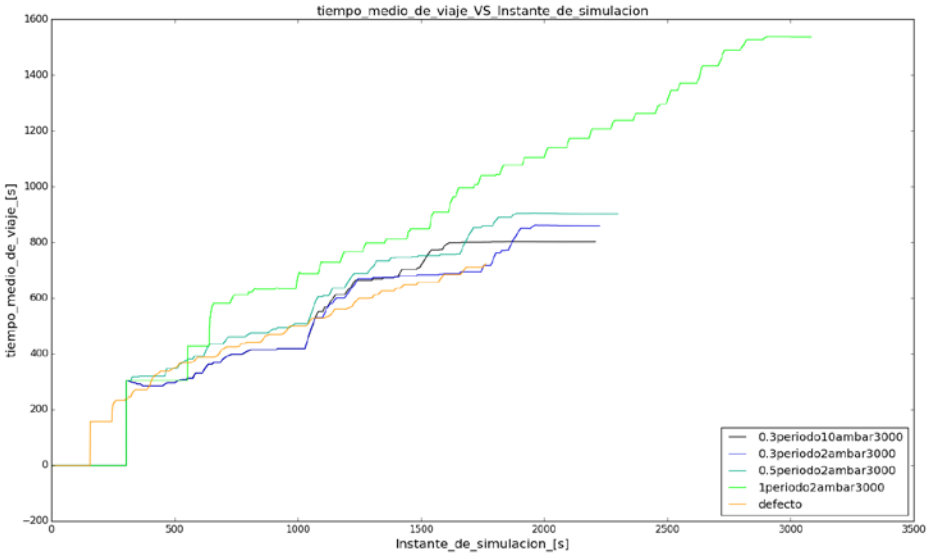

Ilustración 39. Tiempo medio de viaje VS instante de simulación para el mapa de Brooklyn reducido con aceleraciones y deceleraciones por defecto

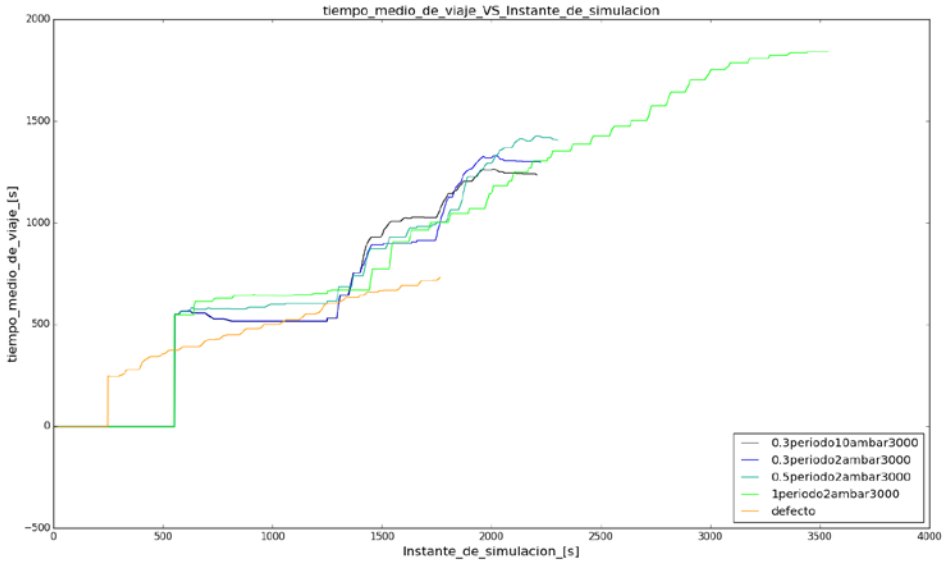

Ilustración 40. Tiempo medio de viaje VS instante de simulación Tiempo medio de viaje VS instante de simulación para el mapa de Brooklyn reducido con aceleraciones y deceleraciones no por defecto

6

Conclusiones y líneas futuras

6.1 Conclusiones

Tras la realización de este Trabajo de Fin de Grado se consiguió:

- Realizar una herramienta para implantar políticas de automáticamente semáforos a partir de la información obtenida de la red y de los los vehículos durante la simulación
- Crear una política de semáforos a partir de la densidad de vehículos que permite regular los tiempos de las fases de los semáforos
- Crear scrips para facilitar la preparación de las simulaciones
- Exportar mapas de fuentes externas para usarlos en simulaciones
- Establecer un criterio para asociar aros inductivos a un semáforo
- Establecer un criterio de compatibilidad de fases en una intersección
- Crear las fases minimas de un cruce con semáforo de forma automática para su funcionamiento
- Hacer uso de la herramienta Traffic Modeller para crear flujos de tráfico de manera gráfica
- Separar la políticas de tráfico realizada de las reglas de tráfico para generar las fases de manera automática
- Logró comprender las diferentes aproximaciones utilizadas por diferentes autores para la regulación del tráfico

Tras un análisis de los resultados obtenidos se puede concluir que la política implementada permite reducir el tiempo medio de viaje en múltiples casos, sin embargo dicha política podría optimizarse más agrupando de una manera más eficiente las fases para así contribuir a disminuir el tiempo medio de viaje en el caso de un mismo nodo origen pero diferentes nodos destinos.También es importante indicar que debido a la generalización de muchas de las funciones creadas para la implementación de la política creada, su utilización puede ser de gran utilidad para crear y analizar otro tipo de políticas que se deseen probar o implementar

6.2 Lineas futuras

Como líneas futuras cabría implementar:

- La coordinación de tiempos de los diferentes semáforos
- El cálculo automático de los parámetros necesarios para crear las fases como tiempos de ciclo y tiempos de las fases intermedias
- Inclusión de eventos para vehículos de emergencias que permitieran facilitarles el paso en las intersecciones mediante un cambio de fase favoreciendo su prioridad
- Inclusión de la posibilidad de realizar un giro de 180 grados en la creación de las fases de semáforos para evitar problemas en los mapas importados que hagan uso de la herramienta Traffic Modeller para generar el tráfico y asi obtener resultados más precisos
- Creación de fases mediante un mayor agrupamiento de las fases compatibles para intentar reducir el tiempo medio en las carreteras con un único carril
- Implementar un mecanismo en la política de semáforos que permita saltarse fases ante la ausencia de vehículos durante un largo periodo de tiempo
- Implementar una política de asignación de tiempos de las fases en función de la demanda en vez de una política mediante estimación de la demanda

7 Bibliografía

1] R. D. Bretherton, "SCOOT INFORMATION," Traffic Management Division of the Transport and Road Research Laboratory, Crowthorn, 1988.

2] "Smart City," Wikipedia, 9 7 2018. [Online]. Available: https://es.wikipedia.org/wiki/Ciudad_inteligente. [Accessed 28 7 2018].

3] H. Chourabi, T. Nam, S. Walker, J. R. Gil-Garcia, S. Mellouli, K. Nahon, T. A. Pardo and H. J. Scholl, "Understanding Smart Cities: An Integrative Framework," in *45th Hawaii International Conference on System Sciences*, Maui, HI, USA, 2012.

4] "Blogthinkbig.com," Telefónica, 10 7 2018. [Online]. Available: https://blogthinkbig.com/onlife/santander-smart-city. [Accessed 18 7 2018].

5] "Coruña Ciudad Smart," Ayuntamiento de A coruña, [Online]. Available: http://www.coruna.gal/smart/es/ciudad-smart. [Accessed 19 7 2018].

6] S. G.C, S. Shirabadagi and R. S. Hegadi, "High Density Traffic Management using Image background subtraction Algorithm," *International Journal of Computer Applications,* 2014.

7] "Using 120-billion IoT sensors to improve London traffic.," Datatonic, [Online]. Available: https://datatonic.com/case-studies/using-120-billion-iot-sensors-to-improve-london-traffic/. [Accessed 28 7 2018].

8] J.-D. Tseng, W.-D. Wang and R.-J. Ko, "An UHF Band RFID Vehicle Management System," in *International Workshop on Anti-Counterfeiting, Security and Identification (ASID)*, Xiamen, Fujian, China, 2007.

J. Landt, "The history of RFID," *IEEE Potentials,* vol. 24, 2005.

9]

N. K. Singh, L. Vanajakashi and A. K. Tangirala, "Segmentation of vehicle
10] signatures from inductive loop detector (ILD) data for real-time traffic monitoring," in *10th International Conference on Communication Systems & Networks (COMSNETS)*, Bengaluru, India, 2018.

W. Heng, Q. Ai, D. Eustace and P. Yi, "Optimal Loop Placement and Models
11] for Length-based," Ohio Transportation Consortium, Akron OH, 2011.

V. Gradinescu, C. Gorgorin, R. Diaconescu, V. Cristea and L. Iftode, "Adaptive
12] Traffic Lights Using Car-to-Car Communication," in *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*, Dublin, 2007.

C. Sommet and F. Dressler, Vehicular Networking, Paderborn, Germany:
13] Cambndge University Press, 2015.

P. B. Hunt, D. I. Robertson, R. D. Bretherton and R. I. Winton, SCOOT - A
14] TRAFFIC RESPONSIVE METHOD OF COORDINATING SIGNALS, Crowthorne, Berkshire: Urban Networks Division Traffic Engineering Department Transport and Road Research Laboratory, 1981.

J. Perrin, P. Martin and B. Hansen, "Connecting SCOOT to CORSIM: real-time
15] signal optimization simulation," in *IECON'01. 27th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.37243)*, Denver, CO, USA, USA, 2001.

R. Jayakrishnan, S. P. Mattingly and M. G. McNally, "Anaheim, Performance
16] Study of SCOOT Traffic Control System with Non-Ideal Detectorization: Field Operational Test in the City of," 2000. [Online]. Available: https://stat.hevra.haifa.ac.il/~gweiss/courses/NIDAWorkshop/Transport/Scoot-performance.pdf. [Accessed 19 7 2018].

J. (. Pan, I. S. Popa and C. Borcea, "DIVERT: A Distributed Vehicular Traffic
17] Re-Routing System for Congestion Avoidance," *IEEE Transactions on Mobile Computing,* vol. 16, 2017.

L. Codeca, R. Frank and T. Engel, "Luxembourg SUMO Traffic (LuST)
18] Scenario: 24 Hours of Mobility for Vehicular Networking Research," in *2015 IEEE Vehicular Networking Conference*, Kyoto, Japan, 2015.

D. Krajzewicz, J. Erdmann, M. Behrisch and L. Bieker, "Recent Development
19] and Applications of SUMO – Simulation of Urban MObility," in *International Journal on Advances in Systems and Measurements, vol 5 no 3 & 4, year 2012*, 2012.

S. Joerer, C. Sommer and F. Dressler, "Toward reproducibility and comparability
20] of IVC simulation studies: a literature survey," *IEEE Communications Magazine ,* 2012.

H. Noori, "Realistic urban traffic simulation as vehicular Ad-hoc network
21] (VANET) via Veins framework," in *2012 12th Conference of Open Innovations Association (FRUCT)*, Oulu, Finland, 2012.

"SUMO dlr wiki tutorial Hello SUMO," 5 11 2017. [Online]. Available:
22] http://sumo.dlr.de/wiki/Tutorials/Hello_Sumo. [Accessed 2018 7 10].

"SUMO dlr Network," 8 7 2018. [Online]. Available:
23] http://sumo.dlr.de/wiki/Networks/PlainXML. [Accessed 28 7 2018].

"SUMO dlr Definition of vehicles," 18 6 2018. [Online]. Available:
24] http://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes. [Accessed 28 7 2018].

"SUMO dlr wiki tutorial," 8 11 2017. [Online]. Available:
25] http://sumo.dlr.de/wiki/Tutorials/Driving_in_Circles. [Accessed 29 6 2018].

"SUMO dlr formar of additional files," 18 12 2017. [Online]. Available:
26] http://sumo.dlr.de/wiki/SUMO#Format_of_additional_files. [Accessed 28 7 2018].

"SUMO dlr wiki NETGENERATE," 13 6 2018. [Online]. Available:
27] http://sumo.dlr.de/wiki/NETGENERATE. [Accessed 29 6 2017].

"SUMO dlr wiki NETEDIT," 18 6 2018. [Online]. Available:
28] http://sumo.dlr.de/wiki/NETEDIT. [Accessed 29 6 2018].

"SUMO dlr wiki Abstract Network Generation," 23 8 2017. [Online]. Available:
29] http://sumo.dlr.de/wiki/Networks/Abstract_Network_Generation. [Accessed 29 6 2018].

"Open Street Map," [Online]. Available: https://www.openstreetmap.org/.
30] [Accessed 29 6 2018].

"SUMO dlr wiki NETCONVERT," 18 12 2017. [Online]. Available:
31] http://www.sumo.dlr.de/wiki/NETCONVERT. [Accessed 29 6 2018].

L. G. Papaleontiou and M. D. Dikaiakos, "SUMO Traffic Modeler," 7 8 2015.
32] [Online]. Available: https://sourceforge.net/projects/trafficmodeler/. [Accessed 21 7 2018].

L. G. Papaleondiou and M. D. Dikaiakos, "TrafficModeler: A Graphical Tool
33] for Programming Microscopic Traffic Simulators through High-Level Abstractions," in *VTC Spring 2009 - IEEE 69th Vehicular Technology Conference*, Barcelona, Spain, 2009.

"SUMO dlr SUMO," 18 12 2017. [Online]. Available:
34] http://sumo.dlr.de/wiki/SUMO. [Accessed 2018 7 11].

"SUMO dlr userdoc plot summary," 1 6 2014. [Online]. Available:
35] http://www.sumo.dlr.de/userdoc/Tools/Visualization.html#plot_summary.py. [Accessed 30 6 2018].

"SUMO dlr userdoc Summary," 31 6 2017. [Online]. Available:
36] http://www.sumo.dlr.de/userdoc/Simulation/Output/Summary.html. [Accessed 30 6 2018].

"SUMO dlr Visualization tools," 1 7 2014. [Online]. Available:
37] http://www.sumo.dlr.de/userdoc/Tools/Visualization.html. [Accessed 11 7 2018].

"SUMO dlr Sumolib," 6 11 2018. [Online]. Available:
38] http://sumo.dlr.de/wiki/Tools/Sumolib. [Accessed 30 6 2018].

"SUMO dlr TraCI," 19 12 2017. [Online]. Available:

39] http://sumo.dlr.de/wiki/TraCI. [Accessed 30 6 2018].

40] P.S.Bokare and A.K.Maury, "Acceleration-Deceleration Behaviour of Various Vehicle Types," in *World Conference on Transport Research*, Shanghai, 2016.

41] J. P. D. Prochazkova, "Smart Cities and Critical Infrastructure," in *Smart Cities Symposium Prague 2018*, Praga, 2018.

42] D. Lucic, M. Boban and D. Mileta, "An Impact of General Data Protection Regulation on a Smart City Concept," in *MIPRO 2018*, Opatija, Croacia, 2018.

43] K. Al-Khateeb and J. A. Y. Johari, "Intelligent dynamic traffic light sequence using RFID," in *International Conference on Computer and Communication Engineering*, Kuala Lumpur, Malaysia, 2008.

44] L. Bhaskar, A. Sahai, D. Sinha, G. Varshney and T. Jain, "Intelligent Traffic Light Controller Using Inductive Loops for Vehicle Detection," in *International Conference on Next Generation Computing Technologies (NGCT-2015)* , Dehradun, India, 2015.

8 Anexo I

8.1 Ejemplo fichero *.net.xml*

<net version="0.27" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/net_file.xsd">

<location netOffset="0.00,0.00" convBoundary="-250.00,-250.00,250.00,250.00" origBoundary="-10000000000.00,-10000000000.00,10000000000.00,10000000000.00" projParameter="!"/>

<edge id=":2_0" function="internal">

<lane id=":2_0_0" index="0" speed="13.89" length="5.00" shape="-1.65,4.75 -1.84,3.39 -2.42,2.42 -3.39,1.84 -4.75,1.65"/>

</edge>

<edge id=":2_1" function="internal">

<lane id=":2_1_0" index="0" speed="13.89" length="9.50" shape="-1.65,4.75 -1.65,-4.75"/>

</edge>

<edge id=":2_2" function="internal">

<lane id=":2_2_0" index="0" speed="13.89" length="5.28" shape="-1.65,4.75 -1.25,1.95 -0.05,-0.05 0.05,-0.11"/>

</edge>

<edge id=":2_12" function="internal">

<lane id=":2_12_0" index="0" speed="13.89" length="5.04" shape="0.05,-0.11 1.95,-1.25 4.75,-1.65"/>

</edge>

<edge id=":2_3" function="internal">

<lane id=":2_3_0" index="0" speed="13.89" length="5.00" shape="4.75,1.65 3.39,1.84 2.42,2.42 1.84,3.39 1.65,4.75"/>

</edge>

<edge id=":2_4" function="internal">

<lane id=":2_4_0" index="0" speed="13.89" length="9.50" shape="4.75,1.65 -4.75,1.65"/>

</edge>

<edge id=":2_5" function="internal">

<lane id=":2_5_0" index="0" speed="13.89" length="5.28" shape="4.75,1.65 1.95,1.25 -0.05,0.05 -0.11,-0.05"/>

</edge>

<edge id=":2_13" function="internal">

<lane id=":2_13_0" index="0" speed="13.89" length="5.04" shape="-0.11,-0.05 -1.25,-1.95 -1.65,-4.75"/>

</edge>

<edge id=":2_6" function="internal">

<lane id=":2_6_0" index="0" speed="13.89" length="5.00" shape="1.65,-4.75 1.84,-3.39 2.42,-2.42 3.39,-1.84 4.75,-1.65"/>

</edge>

<edge id=":2_7" function="internal">

<lane id=":2_7_0" index="0" speed="13.89" length="9.50" shape="1.65,-4.75 1.65,4.75"/>

</edge>

<edge id=":2_8" function="internal">

<lane id=":2_8_0" index="0" speed="13.89" length="5.28" shape="1.65,-4.75 1.25,-1.95 0.05,0.05 -0.05,0.11"/>

</edge>

<edge id=":2_14" function="internal">

<lane id=":2_14_0" index="0" speed="13.89" length="5.04" shape="-0.05,0.11 -1.95,1.25 -4.75,1.65"/>

</edge>

<edge id=":2_9" function="internal">

<lane id=":2_9_0" index="0" speed="13.89" length="5.00" shape="-4.75,-1.65 -3.39,-1.84 -2.42,-2.42 -1.84,-3.39 -1.65,-4.75"/>

</edge>

<edge id=":2_10" function="internal">

<lane id=":2_10_0" index="0" speed="13.89" length="9.50" shape="-4.75,-1.65 4.75,-1.65"/>

</edge>

<edge id=":2_11" function="internal">

<lane id=":2_11_0" index="0" speed="13.89" length="5.28" shape="-4.75,-1.65 -1.95,-1.25 0.05,-0.05 0.11,0.05"/>

</edge>

<edge id=":2_15" function="internal">

<lane id=":2_15_0" index="0" speed="13.89" length="5.04" shape="0.11,0.05 1.25,1.95 1.65,4.75"/>

</edge>

<edge id="cardo1i" from="4" to="2" priority="1">

<lane id="cardo1i_0" index="0" speed="13.89" length="245.25" shape="-1.65,250.00 -1.65,4.75"/>

</edge>

<edge id="cardo1v" from="2" to="4" priority="1">

<lane id="cardo1v_0" index="0" speed="13.89" length="245.25" shape="1.65,4.75 1.65,250.00"/>

</edge>

<edge id="cardo2i" from="5" to="2" priority="1">

<lane id="cardo2i_0" index="0" speed="13.89" length="245.25" shape="1.65,-250.00 1.65,-4.75"/>

</edge>

<edge id="cardo2v" from="2" to="5" priority="1">

<lane id="cardo2v_0" index="0" speed="13.89" length="245.25" shape="-1.65,-4.75 -1.65,-250.00"/>

</edge>

<edge id="decumano1i" from="1" to="2" priority="1">

<lane id="decumano1i_0" index="0" speed="13.89" length="245.25" shape="-250.00,-1.65 -4.75,-1.65"/>

</edge>

```
<edge id="decumano1v" from="2" to="1" priority="1">
    <lane id="decumano1v_0" index="0" speed="13.89" length="245.25" shape="-4.75,1.65 -250.00,1.65"/>
</edge>
<edge id="decumano2i" from="3" to="2" priority="1">
    <lane id="decumano2i_0" index="0" speed="13.89" length="245.25" shape="250.00,1.65 4.75,1.65"/>
</edge>
<edge id="decumano2v" from="2" to="3" priority="1">
    <lane id="decumano2v_0" index="0" speed="13.89" length="245.25" shape="4.75,-1.65 250.00,-1.65"/>
</edge>

<tlLogic id="2" type="static" programID="0" offset="0">
    <phase duration="33" state="GGgrrrGGgrrr"/>
    <phase duration="3" state="yygrrryygrrr"/>
    <phase duration="6" state="rrGrrrrGrrr"/>
    <phase duration="3" state="rryrrrryrrr"/>
    <phase duration="33" state="rrrGGgrrrGGg"/>
    <phase duration="3" state="rrryygrrryyg"/>
    <phase duration="6" state="rrrrrGrrrrrG"/>
    <phase duration="3" state="rrrrryrrrrry"/>
</tlLogic>
```

<junction id="1" type="dead_end" x="-250.00" y="0.00" incLanes="decumano1v_0" intLanes="" shape="-250.00,-0.05 -250.00,3.25 -250.00,0.05"/>

<junction id="2" type="traffic_light" x="0.00" y="0.00" incLanes="cardo1i_0 decumano2i_0 cardo2i_0 decumano1i_0" intLanes=":2_0_0 :2_1_0 :2_12_0 :2_3_0 :2_4_0 :2_13_0 :2_6_0 :2_7_0 :2_14_0 :2_9_0 :2_10_0 :2_15_0" shape="-3.25,4.75 3.25,4.75 4.75,3.25 4.75,-3.25 3.25,-4.75 -3.25,-4.75 -4.75,-3.25 -4.75,3.25">

<request index="0" response="000000000000" foes="000100010000" cont="0"/>

<request index="1" response="000000000000" foes="111100110000" cont="0"/>

<request index="2" response="000011000000" foes="110011110000" cont="1"/>

<request index="3" response="000010000000" foes="100010000000" cont="0"/>

<request index="4" response="000110000111" foes="100110000111" cont="0"/>

<request index="5" response="011110000110" foes="011110000110" cont="1"/>

<request index="6" response="000000000000" foes="010000000100" cont="0"/>

<request index="7" response="000000000000" foes="110000111100" cont="0"/>

<request index="8" response="000000000011" foes="110000110011" cont="1"/>

<request index="9" response="000000000010" foes="000000100010" cont="0"/>

<request index="10" response="000111000110" foes="000111100110" cont="0"/>

<request index="11" response="000110011110" foes="000110011110" cont="1"/>

</junction>

<junction id="3" type="dead_end" x="250.00" y="0.00" incLanes="decumano2v_0" intLanes="" shape="250.00,0.05 250.00,-3.25 250.00,-0.05"/>

<junction id="4" type="dead_end" x="0.00" y="250.00" incLanes="cardo1v_0" intLanes="" shape="-0.05,250.00 3.25,250.00 0.05,250.00"/>

<junction id="5" type="dead_end" x="0.00" y="-250.00" incLanes="cardo2v_0" intLanes="" shape="0.05,-250.00 -3.25,-250.00 -0.05,-250.00"/>

<junction id=":2_12_0" type="internal" x="0.05" y="-0.11" incLanes=":2_2_0 cardo2i_0" intLanes=":2_4_0 :2_5_0 :2_6_0 :2_7_0 :2_10_0 :2_11_0"/>

<junction id=":2_13_0" type="internal" x="-0.11" y="-0.05" incLanes=":2_5_0 decumano1i_0" intLanes=":2_1_0 :2_2_0 :2_7_0 :2_8_0 :2_9_0 :2_10_0"/>

<junction id=":2_14_0" type="internal" x="-0.05" y="0.11" incLanes=":2_8_0 cardo1i_0" intLanes=":2_0_0 :2_1_0 :2_4_0 :2_5_0 :2_10_0 :2_11_0"/>

<junction id=":2_15_0" type="internal" x="0.11" y="0.05" incLanes=":2_11_0 decumano2i_0" intLanes=":2_1_0 :2_2_0 :2_3_0 :2_4_0 :2_7_0 :2_8_0"/>

<connection from="cardo1i" to="decumano1v" fromLane="0" toLane="0" via=":2_0_0" tl="2" linkIndex="0" dir="r" state="o"/>

<connection from="cardo1i" to="cardo2v" fromLane="0" toLane="0" via=":2_1_0" tl="2" linkIndex="1" dir="s" state="o"/>

<connection from="cardo1i" to="decumano2v" fromLane="0" toLane="0" via=":2_2_0" tl="2" linkIndex="2" dir="l" state="o"/>

<connection from="cardo2i" to="decumano2v" fromLane="0" toLane="0" via=":2_6_0" tl="2" linkIndex="6" dir="r" state="o"/>

<connection from="cardo2i" to="cardo1v" fromLane="0" toLane="0" via=":2_7_0" tl="2" linkIndex="7" dir="s" state="o"/>

<connection from="cardo2i" to="decumano1v" fromLane="0" toLane="0" via=":2_8_0" tl="2" linkIndex="8" dir="l" state="o"/>

<connection from="decumano1i" to="cardo2v" fromLane="0" toLane="0" via=":2_9_0" tl="2" linkIndex="9" dir="r" state="o"/>

<connection from="decumano1i" to="decumano2v" fromLane="0" toLane="0" via=":2_10_0" tl="2" linkIndex="10" dir="s" state="o"/>

<connection from="decumano1i" to="cardo1v" fromLane="0" toLane="0" via=":2_11_0" tl="2" linkIndex="11" dir="l" state="o"/>

<connection from="decumano2i" to="cardo1v" fromLane="0" toLane="0" via=":2_3_0" tl="2" linkIndex="3" dir="r" state="o"/>

<connection from="decumano2i" to="decumano1v" fromLane="0" toLane="0" via=":2_4_0" tl="2" linkIndex="4" dir="s" state="o"/>

<connection from="decumano2i" to="cardo2v" fromLane="0" toLane="0" via=":2_5_0" tl="2" linkIndex="5" dir="l" state="o"/>

<connection from=":2_0" to="decumano1v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_1" to="cardo2v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_2" to="decumano2v" fromLane="0" toLane="0" via=":2_12_0" dir="s" state="m"/>

<connection from=":2_12" to="decumano2v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_3" to="cardo1v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_4" to="decumano1v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_5" to="cardo2v" fromLane="0" toLane="0" via=":2_13_0" dir="s" state="m"/>

<connection from=":2_13" to="cardo2v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_6" to="decumano2v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_7" to="cardo1v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_8" to="decumano1v" fromLane="0" toLane="0" via=":2_14_0" dir="s" state="m"/>

<connection from=":2_14" to="decumano1v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_9" to="cardo2v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_10" to="decumano2v" fromLane="0" toLane="0" dir="s" state="M"/>

<connection from=":2_11" to="cardo1v" fromLane="0" toLane="0" via=":2_15_0" dir="s" state="m"/>

<connection from=":2_15" to="cardo1v" fromLane="0" toLane="0" dir="s" state="M"/>

```
</net>
```

8.2 Ejemplo fichero *.rou.xml*

```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">
    <vehicle id="1" depart="1.00">
        <route edges="cardo2i decumano1v"/>
    </vehicle>
    <vehicle id="2" depart="2.00">
        <route edges="cardo1i decumano1v"/>
    </vehicle>
    <vehicle id="3" depart="3.00">
        <route edges="decumano2i cardo1v"/>
    </vehicle>
    <vehicle id="4" depart="4.00">
        <route edges="cardo2i decumano1v"/>
    </vehicle>
    <vehicle id="5" depart="5.00">
        <route edges="decumano2i decumano1v"/>
    </vehicle>
    <vehicle id="6" depart="6.00">
        <route edges="decumano2i cardo1v"/>
```

```
</vehicle>
<vehicle id="7" depart="7.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="8" depart="8.00">
   <route edges="decumano1i cardo2v"/>
</vehicle>
<vehicle id="9" depart="9.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="10" depart="10.00">
   <route edges="decumano1i cardo1v"/>
</vehicle>
<vehicle id="11" depart="11.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="12" depart="12.00">
   <route edges="decumano2i cardo1v"/>
</vehicle>
<vehicle id="13" depart="13.00">
   <route edges="decumano1i cardo2v"/>
</vehicle>
```

```
<vehicle id="14" depart="14.00">
   <route edges="cardo1i decumano2v"/>
</vehicle>
<vehicle id="15" depart="15.00">
   <route edges="decumano2i decumano1v"/>
</vehicle>
<vehicle id="16" depart="16.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="21" depart="21.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="22" depart="22.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="23" depart="23.00">
   <route edges="decumano2i cardo2v"/>
</vehicle>
<vehicle id="25" depart="25.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="26" depart="26.00">
```

```
    <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="27" depart="27.00">
    <route edges="cardo1i decumano2v"/>
</vehicle>
<vehicle id="28" depart="28.00">
    <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="29" depart="29.00">
    <route edges="decumano2i decumano1v"/>
</vehicle>
<vehicle id="30" depart="30.00">
    <route edges="decumano2i decumano1v"/>
</vehicle>
<vehicle id="31" depart="31.00">
    <route edges="decumano1i cardo1v"/>
</vehicle>
<vehicle id="32" depart="32.00">
    <route edges="decumano2i decumano1v"/>
</vehicle>
<vehicle id="33" depart="33.00">
    <route edges="decumano1i decumano2v"/>
```

```
</vehicle>
<vehicle id="34" depart="34.00">
   <route edges="decumano1i cardo2v"/>
</vehicle>
<vehicle id="35" depart="35.00">
   <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="36" depart="36.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="37" depart="37.00">
   <route edges="decumano1i cardo1v"/>
</vehicle>
<vehicle id="39" depart="39.00">
   <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="41" depart="41.00">
   <route edges="cardo2i decumano1v"/>
</vehicle>
<vehicle id="42" depart="42.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
```

```
<vehicle id="43" depart="43.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="44" depart="44.00">
   <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="45" depart="45.00">
   <route edges="decumano2i cardo1v"/>
</vehicle>
<vehicle id="47" depart="47.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="48" depart="48.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="49" depart="49.00">
   <route edges="decumano2i cardo1v"/>
</vehicle>
<vehicle id="50" depart="50.00">
   <route edges="cardo1i decumano2v"/>
</vehicle>
<vehicle id="51" depart="51.00">
```

```
    <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="52" depart="52.00">
    <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="53" depart="53.00">
    <route edges="decumano1i cardo1v"/>
</vehicle>
<vehicle id="54" depart="54.00">
    <route edges="decumano1i cardo2v"/>
</vehicle>
<vehicle id="55" depart="55.00">
    <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="56" depart="56.00">
    <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="58" depart="58.00">
    <route edges="decumano1i cardo2v"/>
</vehicle>
<vehicle id="59" depart="59.00">
    <route edges="cardo2i cardo1v"/>
```

```
</vehicle>
<vehicle id="60" depart="60.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="61" depart="61.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="65" depart="65.00">
   <route edges="cardo2i decumano1v"/>
</vehicle>
<vehicle id="66" depart="66.00">
   <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="67" depart="67.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="68" depart="68.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="69" depart="69.00">
   <route edges="decumano2i cardo1v"/>
</vehicle>
```

```
<vehicle id="70" depart="70.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="71" depart="71.00">
   <route edges="decumano1i cardo2v"/>
</vehicle>
<vehicle id="73" depart="73.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="74" depart="74.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="75" depart="75.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="76" depart="76.00">
   <route edges="decumano2i cardo2v"/>
</vehicle>
<vehicle id="77" depart="77.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="79" depart="79.00">
```

```
    <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="80" depart="80.00">
    <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="83" depart="83.00">
    <route edges="cardo1i decumano2v"/>
</vehicle>
<vehicle id="84" depart="84.00">
    <route edges="decumano2i decumano1v"/>
</vehicle>
<vehicle id="85" depart="85.00">
    <route edges="decumano1i cardo1v"/>
</vehicle>
<vehicle id="86" depart="86.00">
    <route edges="cardo1i decumano2v"/>
</vehicle>
<vehicle id="87" depart="87.00">
    <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="88" depart="88.00">
    <route edges="decumano1i decumano2v"/>
```

```
</vehicle>
<vehicle id="90" depart="90.00">
   <route edges="cardo1i decumano2v"/>
</vehicle>
<vehicle id="93" depart="93.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="94" depart="94.00">
   <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="95" depart="95.00">
   <route edges="decumano2i cardo2v"/>
</vehicle>
<vehicle id="96" depart="96.00">
   <route edges="decumano2i decumano1v"/>
</vehicle>
<vehicle id="97" depart="97.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="99" depart="99.00">
   <route edges="decumano1i cardo2v"/>
</vehicle>
```

```
<vehicle id="100" depart="100.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="101" depart="101.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="102" depart="102.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="103" depart="103.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="105" depart="105.00">
   <route edges="decumano1i cardo1v"/>
</vehicle>
<vehicle id="106" depart="106.00">
   <route edges="decumano2i decumano1v"/>
</vehicle>
<vehicle id="107" depart="107.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="108" depart="108.00">
```

<route edges="cardo1i decumano2v"/>

</vehicle>

<vehicle id="109" depart="109.00">

<route edges="decumano1i cardo1v"/>

</vehicle>

<vehicle id="110" depart="110.00">

<route edges="decumano2i decumano1v"/>

</vehicle>

<vehicle id="111" depart="111.00">

<route edges="decumano2i cardo2v"/>

</vehicle>

<vehicle id="112" depart="112.00">

<route edges="decumano2i cardo1v"/>

</vehicle>

<vehicle id="113" depart="113.00">

<route edges="decumano1i cardo1v"/>

</vehicle>

<vehicle id="115" depart="115.00">

<route edges="cardo2i decumano1v"/>

</vehicle>

<vehicle id="116" depart="116.00">

<route edges="cardo1i decumano1v"/>

```
</vehicle>
<vehicle id="117" depart="117.00">
   <route edges="decumano1i cardo2v"/>
</vehicle>
<vehicle id="118" depart="118.00">
   <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="122" depart="122.00">
   <route edges="decumano2i cardo2v"/>
</vehicle>
<vehicle id="123" depart="123.00">
   <route edges="decumano1i cardo2v"/>
</vehicle>
<vehicle id="125" depart="125.00">
   <route edges="cardo2i decumano1v"/>
</vehicle>
<vehicle id="126" depart="126.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="127" depart="127.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
```

```
<vehicle id="129" depart="129.00">
   <route edges="decumano2i decumano1v"/>
</vehicle>
<vehicle id="130" depart="130.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="131" depart="131.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="132" depart="132.00">
   <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="133" depart="133.00">
   <route edges="cardo2i decumano1v"/>
</vehicle>
<vehicle id="134" depart="134.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="135" depart="135.00">
   <route edges="decumano2i cardo2v"/>
</vehicle>
<vehicle id="136" depart="136.00">
```

```
    <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="137" depart="137.00">
    <route edges="decumano1i cardo2v"/>
</vehicle>
<vehicle id="140" depart="140.00">
    <route edges="decumano2i cardo2v"/>
</vehicle>
<vehicle id="141" depart="141.00">
    <route edges="decumano2i cardo2v"/>
</vehicle>
<vehicle id="142" depart="142.00">
    <route edges="cardo1i decumano2v"/>
</vehicle>
<vehicle id="144" depart="144.00">
    <route edges="decumano2i cardo1v"/>
</vehicle>
<vehicle id="145" depart="145.00">
    <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="146" depart="146.00">
    <route edges="decumano2i cardo2v"/>
```

```
</vehicle>
<vehicle id="147" depart="147.00">
   <route edges="cardo2i decumano1v"/>
</vehicle>
<vehicle id="148" depart="148.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="149" depart="149.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="150" depart="150.00">
   <route edges="decumano2i cardo2v"/>
</vehicle>
<vehicle id="151" depart="151.00">
   <route edges="cardo2i decumano1v"/>
</vehicle>
<vehicle id="152" depart="152.00">
   <route edges="decumano2i cardo2v"/>
</vehicle>
<vehicle id="153" depart="153.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
```

```
<vehicle id="156" depart="156.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="157" depart="157.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="158" depart="158.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="159" depart="159.00">
   <route edges="decumano2i cardo1v"/>
</vehicle>
<vehicle id="160" depart="160.00">
   <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="161" depart="161.00">
   <route edges="cardo2i cardo1v"/>
</vehicle>
<vehicle id="165" depart="165.00">
   <route edges="cardo2i decumano2v"/>
</vehicle>
<vehicle id="166" depart="166.00">
```

<route edges="cardo1i decumano2v"/>

</vehicle>

<vehicle id="167" depart="167.00">

<route edges="cardo2i decumano2v"/>

</vehicle>

<vehicle id="168" depart="168.00">

<route edges="decumano1i decumano2v"/>

</vehicle>

<vehicle id="169" depart="169.00">

<route edges="cardo2i decumano1v"/>

</vehicle>

<vehicle id="170" depart="170.00">

<route edges="decumano2i cardo2v"/>

</vehicle>

<vehicle id="171" depart="171.00">

<route edges="decumano2i cardo1v"/>

</vehicle>

<vehicle id="172" depart="172.00">

<route edges="cardo1i decumano1v"/>

</vehicle>

<vehicle id="174" depart="174.00">

<route edges="cardo1i decumano2v"/>

```
</vehicle>
<vehicle id="175" depart="175.00">
   <route edges="decumano1i cardo1v"/>
</vehicle>
<vehicle id="176" depart="176.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="177" depart="177.00">
   <route edges="decumano1i cardo2v"/>
</vehicle>
<vehicle id="178" depart="178.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="179" depart="179.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="180" depart="180.00">
   <route edges="decumano2i cardo1v"/>
</vehicle>
<vehicle id="181" depart="181.00">
   <route edges="cardo2i decumano1v"/>
</vehicle>
```

```
<vehicle id="182" depart="182.00">
   <route edges="cardo1i decumano2v"/>
</vehicle>
<vehicle id="183" depart="183.00">
   <route edges="cardo1i cardo2v"/>
</vehicle>
<vehicle id="186" depart="186.00">
   <route edges="decumano2i decumano1v"/>
</vehicle>
<vehicle id="187" depart="187.00">
   <route edges="cardo2i decumano1v"/>
</vehicle>
<vehicle id="188" depart="188.00">
   <route edges="cardo1i decumano1v"/>
</vehicle>
<vehicle id="191" depart="191.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="192" depart="192.00">
   <route edges="decumano1i decumano2v"/>
</vehicle>
<vehicle id="193" depart="193.00">
```

```
    <route edges="cardo2i cardo1v"/>
  </vehicle>
  <vehicle id="194" depart="194.00">
    <route edges="cardo1i decumano2v"/>
  </vehicle>
  <vehicle id="195" depart="195.00">
    <route edges="cardo2i decumano2v"/>
  </vehicle>
  <vehicle id="197" depart="197.00">
    <route edges="decumano2i cardo2v"/>
  </vehicle>
  <vehicle id="199" depart="199.00">
    <route edges="cardo2i decumano1v"/>
  </vehicle>
</routes>
```

8.3 Ejemplo fichero *.sumocfg*

```
<configuration>
<input>
<net-file value="cruce.net.xml"/>
<route-files value="cruce.rou.xml"/>
<additional-files value="cruce.add.xml"/>
</input>
```

<time>

<begin value="0"/>

<end value="86400"/>

</time>

</configuration>

8.4 Ejemplo fichero *.add.xml*

<additionals>

<e1Detector id="e1Detector_cardo1i_0_7" lane="cardo1i_0" pos="196.20" freq="100.00" file="e1Detector_cardo1i_0_7.txt" friendlyPos="0"/>

<e1Detector id="e1Detector_cardo1v_0_1" lane="cardo1v_0" pos="4.91" freq="100.00" file="e1Detector_cardo1v_0_1.txt" friendlyPos="0"/>

<e1Detector id="e1Detector_cardo2i_0_5" lane="cardo2i_0" pos="196.20" freq="100.00" file="e1Detector_cardo2i_0_5.txt" friendlyPos="0"/>

<e1Detector id="e1Detector_cardo2v_0_2" lane="cardo2v_0" pos="4.91" freq="100.00" file="e1Detector_cardo2v_0_2.txt" friendlyPos="0"/>

<e1Detector id="e1Detector_decumano1i_0_6" lane="decumano1i_0" pos="196.20" freq="100.00" file="e1Detector_decumano1i_0_6.txt" friendlyPos="0"/>

<e1Detector id="e1Detector_decumano1v_0_0" lane="decumano1v_0" pos="4.91" freq="100.00" file="e1Detector_decumano1v_0_0.txt" friendlyPos="0"/>

<e1Detector id="e1Detector_decumano2i_0_4" lane="decumano2i_0" pos="196.20" freq="100.00" file="e1Detector_decumano2i_0_4.txt" friendlyPos="0"/>

<e1Detector id="e1Detector_decumano2v_0_3" lane="decumano2v_0" pos="4.91" freq="100.00" file="e1Detector_decumano2v_0_3.txt" friendlyPos="0"/>

</additionals>

8.5 Código gencfg

```
#!/bin/bash

#script creado por Tomás De Francisco Aparicio

#este script lo que hace es autogenerar el sumo.cfg a partir de los ficheros que hay en la carpeta

echo "Stript para generar .sumocfg"

#lo que busca primero es un fichero .net.xml en el directorio actual

#ojo solo funciona si hay un .net.xml en el directorio

nombre="$(ls |grep ".net.xml")"

#con esto he obtenido el nombre del fichero entero que tiene eso, luego ahora tengo que obtener el resto de ficheros con ese nombre para añadirlos a las lista

echo $nombre

extension=".net.xml"

#ahora lo que hago es obtener la primera parte del nombre del fichero es decir parseo el fichero

inicio="${nombre%$extension}"

#con la linea anterior lo que consigo es eliminar la extension de nombre

#ahora lo que tengo que hacer es obtener los nombres de los ficheros completos que contienen ese prefijo obtenido del nombre y elimo el net y el rou ya que estan predefinidos al autocrear el script y los trips que no me interesan, y el propio .sumocfg

ficheros="$(ls |grep  "$inicio" | grep -v "$inicio.net.xml" | grep -v "$inicio.rou"| grep -v "trips" | grep -v "$inicio.sumocfg")"

#ahora lo que hago es que lo redirijo al fichero inicialmente sobrescribo el fichero que hay
```

```
echo '<configuration>' > "$inicio.sumocfg"

#con la doble >> lo que hago es escribir al final del fichero

echo '<input>' >> "$inicio.sumocfg"

echo '<net-file value="""$inicio"'.net.xml"/>' >> "$inicio.sumocfg"

echo '<route-files value="""$inicio"'.rou.xml"/>' >> "$inicio.sumocfg"

#hasta ahora he puesto los fichero basicos, hay que buscar a parte los ficheros extra

if [ -n "$ficheros" ]

then

#primero tengo que comprobar cuantos ficheros adicionales hay en numero

num="$(ls |grep  "$inicio" | grep -v "$inicio.net.xml" | grep -v "$inicio.rou"| grep -v "trips" | grep -v "$inicio.sumocfg" | grep -c "$inicio")"

echo el numero de ficheros adicionales es $num

#con esto tengo que repetir este comando tantas veces cambiando el prefijo del lo de incluir, es decir un bucle for

parcial=$ficheros

for ((i=0; i<"$num";i++))

do

#aqui tengo que obtener el nombre del fichero a partir de la variable fichero similar a lo que haciamos antes para extraer el sufijo y quedarnos con el prefijo, es decir me quedo con lo primero que cuentro del grep y luego lo elimino

parcial="$(echo $parcial |cut -d " " -f1 )"

echo "el nombre del fichero en la iteracion es $parcial"

echo '<additional-files value="""$parcial"""/>' >> "$inicio.sumocfg"
```

parcial=${ficheros#$parcial}

#ahora tengo que eliminar la primare coincidencia como hago con lo del sufijo y el prefijo de antes

done

else

echo "no hay ficheros adicionales"

fi

echo '</input>' >> "$inicio.sumocfg"

echo '<time>' >> "$inicio.sumocfg"

echo '<begin value="""0"""/>' >> "$inicio.sumocfg"

echo '<end value="""86400"""/>' >> "$inicio.sumocfg"

echo '</time>' >> "$inicio.sumocfg"

echo '</configuration>' >> "$inicio.sumocfg"

#para imprimir las comillas dobles las meto dentro de simples, sino el bash las interpreta

#por ultimo abro en modo grafico el fichero sumogui creado

sumo-gui -c "$inicio.sumocfg"

8.6 Código genmappol

#!/bin/bash

#script creado por Tomás De Francisco Aparicio

#este script lo que hace es ejjecutar el comando para generar trafico y generarte el rou.xml directamente sin tener que ejecutar los comandos por consola cada vez

echo Script para generar los mapas en el formato y despues generar el sumocfg

```
echo introduzca el nombre del fichero de red

read nombre

#con esto compruebo si el fichero existe

if [ -f "$nombre" ]

then

#ahora renombro el fichero

mv "$nombre" "$nombre.osm.xml"

netconvert --osm-files $nombre.osm.xml -o $nombre.net.xml --output.street-names true --output.original-names true

if [ -f "typemap.xml" ]

then

#creo los poligonos

polyconvert --net-file $nombre.net.xml --osm-files $nombre.osm.xml --type-file typemap.xml -o $nombre.poly.xml

echo "se crearon los poligonos"

else

"el fichero typemap.xml no existe, se copiara al directorio"

#aqui tambien se puede ir al directorio donde deberia estar y copialo al directorio actual con el nombre de typemap

cp $SUMO_HOME/data/typemap/"osmPolyconvert.typ" ./"typemap.xml"

polyconvert --net-file $nombre.net.xml --osm-files $nombre.osm.xml --type-file typemap.xml -o $nombre.poly.xml

echo "se crearon los poligonos"
```

```
fi

else

if [ -f "$nombre.osm.xml" ]

then

netconvert --osm-files $nombre.osm.xml -o $nombre.net.xml --output.street-names true --output.original-names true

if [ -f "typemap.xml" ]

then

#creo los poligonos

polyconvert --net-file $nombre.net.xml --osm-files $nombre.osm.xml --type-file typemap.xml -o $nombre.poly.xml

echo "se crearon los poligonos"

else

"el fichero typemap.xml no existe, se copiara al directorio"

#aqui tambien se puede ir al directorio donde deberia estar y copialo al directorio actual con el nombre de typemap

cp $SUMO_HOME/data/typemap/"osmPolyconvert.typ" ./"typemap.xml"

polyconvert --net-file $nombre.net.xml --osm-files $nombre.osm.xml --type-file typemap.xml -o $nombre.poly.xml

echo "se crearon los poligonos"

fi

else

echo "el fichero no existe"
```

```
exit

fi

#si no existe el fichero finalizo el prgrama

fi
```

```
#ahora llamo a los 2 scripts de generar trafico y de generar el cfg

#!/bin/bash

#este script lo que hace es ejjecutar el comando para generar trafico y generarte el rou.xml directamente sin tener que ejecutar los comandos por consola cada vez

echo genero trafico

#con esto compruebo si el fichero existe

if [ -f "$nombre.net.xml" ]

then

$SUMO_HOME/tools/randomTrips.py -n $nombre.net.xml -l -e 600 -o $nombre.trips.xml

duarouter -n $nombre.net.xml -t $nombre.trips.xml -o $nombre.rou.xml

echo se creo correctamente

else

echo "el fichero no existe"
```

```
fi



#!/bin/bash

#este script lo que hace es autogenerar el sumo.cfg a partir de los ficheros que hay en la carpeta

echo "Stript para generar .sumocfg"

#lo que busca primero es un fichero .net.xml en el directorio actual

#ojo solo funciona si hay un .net.xml en el directorio

#con esto he obtenido el nombre del fichero entero que tiene eso, luego ahora tengo que obtener el resto de ficheros con ese nombre para añadirlos a las lista

extension=".net.xml"

#ahora lo que hago es obtener la primera parte del nombre del fichero es decir parseo el fichero

inicio="${nombre%$extension}"

#con la linea anterior lo que consigo es eliminar la extension de nombre

#ahora lo que tengo que hacer es obtener los nombres de los ficheros completos que contienen ese prefijo obtenido del nombre y elimo el net y el rou ya que estan predefinidos al autocrear el script y los trips que no me interesan, y el propio .sumocfg

ficheros="$(ls |grep  "$inicio" | grep -v "$inicio.net.xml" | grep -v "$inicio.rou"| grep -v "trips" | grep -v "$inicio.sumocfg"  | grep -v "$inicio.osm.xml")"

#ahora lo que hago es que lo redirijo al fichero inicialmente sobrescribo el fichero que hay

echo '<configuration>' > "$inicio.sumocfg"

#con la doble >> lo que hago es escribir al final del fichero
```

```
echo '<input>' >> "$inicio.sumocfg"

echo '<net-file value="""$inicio"'.net.xml"/>' >> "$inicio.sumocfg"

echo '<route-files value="""$inicio"'.rou.xml"/>' >> "$inicio.sumocfg"

#hasta ahora he puesto los fichero basicos, hay que buscar a parte los ficheros extra

if [ -n "$ficheros" ]

then

#primero tengo que comprobar cuantos ficheros adicionales hay en numero

num="$(ls |grep  "$inicio" | grep -v "$inicio.net.xml" | grep -v "$inicio.rou"| grep -v "trips" | grep -v "$inicio.sumocfg" | grep -v "$inicio.osm.xml"| grep -c "$inicio")"

echo el numero de ficheros adicionales es $num

#con esto tengo que repetir este comando tantas veces cambiando el prefijo del lo de incluir, es decir un bucle for

parcial=$ficheros

for ((i=0; i<"$num";i++))

do

#aqui tengo que obtener el nombre del fichero a partir de la variable fichero similar a lo que haciamos antes para extraer el sufijo y quedarnos con el prefijo, es decir me quedo con lo primero que cuentro del grep y luego lo elimino

parcial="$(echo $parcial |cut -d " " -f1 )"

echo "el nombre del fichero en la iteracion es $parcial"

echo '<additional-files value="""$parcial"""/>' >> "$inicio.sumocfg"

parcial=${ficheros#$parcial}

#ahora tengo que eliminar la primare coincidencia como hago con lo del sufijo y el prefijo de antes
```

done

else

echo "no hay ficheros adicionales"

fi

echo '</input>' >> "$inicio.sumocfg"

echo '<time>' >> "$inicio.sumocfg"

echo '<begin value="""0"""/>' >> "$inicio.sumocfg"

echo '<end value="""86400"""/>' >> "$inicio.sumocfg"

echo '</time>' >> "$inicio.sumocfg"

echo '</configuration>' >> "$inicio.sumocfg"

#para imprimir las comillas dobles las meto dentro de simples, sino el bash las interpreta

#por ultimo abro en modo grafico el fichero y le pongo un fichero de las estadisticas de salida

sumo-gui -c "$inicio.sumocfg" --summary "$nombre.estadisticas"

8.7 Código gentrafico

#!/bin/bash

#script creado por Tomás De Francisco Aparicio

#este script lo que hace es ejjecutar el comando para generar trafico y generarte el rou.xml directamente sin tener que ejecutar los comandos por consola cada vez

echo Script para generar trafico introduciendo solo el nombre del fichero

echo introduzca el nombre del fichero de red

read nombre

```
#con esto compruebo si el fichero existe

if [ -f "$nombre.net.xml" ]

then

$SUMO_HOME/tools/randomTrips.py -n $nombre.net.xml -l -e 600 -o $nombre.trips.xml

duarouter -n $nombre.net.xml -t $nombre.trips.xml -o $nombre.rou.xml

echo se creo correctamente

else

echo "el fichero no existe"

fi
```

8.8 Código gentraftm

```
#!/bin/bash

#script creado por Tomás De Francisco Aparicio

#este script lo que hace es ejjecutar el comando para generar trafico y generarte el rou.xml directamente sin tener que ejecutar los comandos por consola cada vez

echo Script para generar generar trafico mediante la herramienta Traffic Modeler

echo introduzca el nombre del fichero del mapa osm

read nombre

#con esto compruebo si el fichero existe

if [ -f "$nombre" ]

then

echo "se generaran los ficheros necesarios para la herramienta traffic modeler"
```

```
#con esto obtengo los ficheros nod,edg y net necesarios para la herramienta

netconvert --osm-files $nombre -o $nombre.net.xml --remove-edges.isolated TRUE
--geometry.remove --roundabouts.guess TRUE --ramps.guess TRUE --junctions.join TRUE

netconvert --osm-files $nombre --plain-output-prefix $nombre --remove-
edges.isolated TRUE --geometry.remove --roundabouts.guess TRUE --ramps.guess TRUE --
junctions.join TRUE

#netconvert --osm-files $nombre -o $nombre.net.xml --remove-edges.isolated
TRUE --geometry.remove --roundabouts.guess TRUE --ramps.guess TRUE --junctions.join
TRUE --tls.guess-signals TRUE --tls.discard-simple TRUE --tls.join TRUE

#netconvert --osm-files $nombre --plain-output-prefix $nombre --remove-
edges.isolated TRUE --geometry.remove --roundabouts.guess TRUE --ramps.guess TRUE --
junctions.join TRUE --tls.guess-signals TRUE --tls.discard-simple TRUE --tls.join TRUE

#netconvert --osm-files $nombre -o $nombre.net.xml --output.street-names true --
output.original-names true

#netconvert --osm-files $nombre --plain-output-prefix $nombre

else

echo "el fichero mapa no existe"

exit 1

fi

if [ -d "$nombre"_simulacion ]

then

echo "existe el directorio"

else

echo "el directorio no existe, se creara"

mkdir ./"$nombre"_simulacion
```

```
fi

#ahora creo un nuevo directorio donde se meteran los archivos de la simulacion es decir un subdirectorio con el nombre simulcion donde esta el mapa

printf "\n\nIMPORTANTE\n\n"

echo "si es la primera vez que ejecuta esto, en settings al abrirse el .jar la ruta de duarouter y de netconvert utilize algo similar a /usr/local/bin/duarouter y /usr/local/bin/netconvert"

echo "la ruta la puede obtener con whereis duarouter , whereis netconvert"

echo "GUARDA EL PROYECTO CON EL MISMO NOMBRE DEL FICHERO DE RED"

echo "al exportar el trafico usar la carpeta $nombre_simulacion creada en el directorio como export path y usar el mismo nombre de simulation name que el del fichero correspondiente al mapa original"

read -p "pulse cualquier tecla para continuar"

echo "se abrira la aplicacion de java"

#ahora abro la aplicacion desde la ruta predefinida(se necesita java ya que es un .jar),

java -jar "$SUMO_HOME/bin/TrafficModeller-1.1.jar"

#cuando finalice la ejecucion del jar se procedera a copiar el fichero de la red a la carpeta especificada ya que se habra generado el fichero de trafico

cp "$nombre".net.xml ./"$nombre"_simulacion/

echo "se copio el fichero de red"

#abro el fichero

#entro el en directorio

cd ./"$nombre"_simulacion
```

#esto genera un fichero de rutas .trips hay que convertirlo a un fichero .rou con el comando duarouter

duarouter -n "$nombre".net.xml -t "$nombre".generated-trips.xml -o "$nombre".rou.xml --repair TRUE --ignore-errors TRUE --keep-all-routes --repair.to TRUE --repair.from TRUE

#como dentro de la carpeta deberia estar creado el fichero de configuracion ejecuto sumo con interfaz grafica desde dentro

#por ultimo lanzo sumo con su interfaz grafica con la simulacion y para que salga un fichero de salida con los parametros de la simulacion

echo "¿modo grafico? s/n"

read graph

if ["$graph"="s"]

then

sumo-gui -c "$nombre".sumocfg --summary "estadisticas_$nombre" --time-to-teleport 3000

else

sumo -c "$nombre".sumocfg --summary "estadisticas_$nombre" --time-to-teleport 3000

fi

echo "se creo el fichero con las estadisticas de la simulacion con nombre estadisticas_$nombre en el directorio ./"$nombre"_simulacion"

#ahora lo que hago es generar una grafico el el tiempo medio de viaje

python "$SUMO_HOME/tools/visualization/plot_summary.py" -i estadisticas_"$nombre" -o ./tiempomedio.png -m meanTravelTime

8.9 Código crearloopsnet

```
#!/usr/bin/env python
#este script lo que hace es generar 2 loops por cada carril que tenga
semanforo
# coding=utf-8
# Eclipse SUMO, Simulation of Urban MObility; see
https://eclipse.org/sumo
# Copyright (C) 2009-2017 German Aerospace Center (DLR) and others.
# This program and the accompanying materials
# are made available under the terms of the Eclipse Public License v2.0
# which accompanies this distribution, and is available at
# http://www.eclipse.org/legal/epl-v20.html

# @file    generateTLSE1Detectors.py
# @author  Daniel Krajzewicz
# @author  Karol Stosiek
# @author  Michael Behrisch
# @date    2011-10-07
# @version $Id$
#editado por Tomás De Francisco, para generar dos loops por carretera

from __future__ import absolute_import
from __future__ import print_function

import logging
import optparse
import os
import sys

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__
))))
import  sumolib  # noqa


def open_detector_file(destination_dir, detector_file_name):#esto es
para crear un nuevo fichero de detectores y escribilrlo ahi con permisos
de escritura
    """ Opens a new detector file in given directory. """

    return open(os.path.join(destination_dir, detector_file_name), "w")

def get_net_file_directory(net_file):#esto es para definir que fichero
de red quieres leer
    """ Returns the directory containing the net file given. """

    dirname = os.path.split(net_file)[0]
    return dirname

if __name__ == "__main__":#esta es la funcion en si que se encarga de
poner los detectores
    # pylint: disable-msg=C0103

    #el nombre del fichero es

    logging.basicConfig(level="INFO")

    option_parser = optparse.OptionParser()

    logging.basicConfig(level="INFO")
```

```
    option_parser = optparse.OptionParser()
    option_parser.add_option("-n", "--net-file",
                             dest="net_file",
                             help="Network file to work with.
Mandatory.",
                             type="string")
    option_parser.add_option("-l", "--detector-length",
                             dest="requested_detector_length",
                             help="Length of the detector in meters "
                             "(-1 for maximal length).",
                             type="int",
                             default=250)
    option_parser.add_option("-d", "--distance-to-TLS",
                             dest="requested_distance_to_tls",
                             help="Distance of the detector to the
traffic "
                             "light in meters. Defaults to 0.1m.",
                             type="float",
                             default=.1)
    option_parser.add_option("-f", "--frequency",
                             dest="frequency",
                             help="Detector's frequency. Defaults to
60.",
                             type="int",
                             default=60)
    option_parser.add_option("-o", "--output",
                             dest="output",
                             help="The name of the file to write the
detector "
                             "definitions into. Defaults to
e1.add.xml.",
                             type="string",
                             default="e1.add.xml")
    option_parser.add_option("-r", "--results-file",
                             dest="results",
                             help="The name of the file the detectors
write "
                             "their output into. Defaults to
e1output.xml.",
                             type="string",
                             default="e1output.xml")
    option_parser.set_usage("generateTLSE1Detectors.py -n
example.net.xml "
                            "-l 250 -d .1 -f 60")

    (options, args) = option_parser.parse_args()

    #aqui es donde se asigna el nombre de los dicheros
    nombre="brooklyn10"#nombre del fichero .net sin extensiones
    ruta="./"+"brooklyn10/"#ruta donde esta el fichero .net
    options.net_file = ruta+nombre+".net.xml"
    options.output="pruebacruce2loopsduplicados.add.xml"
    print("net file es ",options.net_file)
    print("el fichero de salida es ",options.output)

    if not options.net_file:#esto te indica que no has elegido un
fichero de red y que falta ese argumento
        print("Missing arguments")
        option_parser.print_help()
        exit()
```

```
    logging.info("Reading net...")#esto te indica que se esta leyendo la
red
    net = sumolib.net.readNet(options.net_file)#esto te devuelve el
fichero de red, es decir lee el fichero de red
    logging.info("Generating detectors...")
    detectors_xml = sumolib.xml.create_document("additional")#esto es
para crear el fichero adicional que contiene a los detectores
    lanes_with_detectors = set()
    lanes_with_detectors2=set()
    #importante porcentaje 1 y 2 son imprescindibles para que funcione
bien ya que asi crea duplicados algunos loops pero al ser completamente
idénticas sus características se pueden eliminar los duplicados
    porcentaje1=0.4
    porcentaje2=0.8

    for tls in net._tlss:
        #podrian llegar a repetirse _> creo que no porque solo lo
ejecutarias una vez por linea
        for connection in tls._connections:#esto recorre todas las
conexiones que tiene un semaforo
            print("la lineas a las que esta conectado el semaforo son
",connection)
            lane = connection[0]#la linea es la primera conexion del
semaforo, esto es para los nodos de entrada
            lane2=connection[1]#esto es para los nodos de salida
            lane_length = lane.getLength()#aqui obtengo la longitud de
la linea, puedo ponerlo al 5% y al 95% el detector
            lane_id = lane.getID()#obtengo el ID de la linea
            lane_length2 = lane2.getLength()  #aqui obtengo la longitud
de la linea, puedo ponerlo al 5% y al 95% el detector
            lane_id2 = lane2.getID()  #obtengo el ID de la linea

            logging.debug("Creating detector for lane %s" %
(str(lane_id)))#esto te dice que se ha creado un detector para la linea
tal


            if lane_id in lanes_with_detectors:#esto te dice que el
detector para una linea detecminada ya ha sido generado
                logging.warn("Detector for lane %s already generated"
%(str(lane_id)))
                #continue
            else:
                lanes_with_detectors.add(lane_id)
                print("la longitud de la linea es ", lane_length)
                final_detector_position = lane_length * porcentaje1 #es
decir dejo esta longitud al 2% de ela longitud de la linea
                print("la posicion en que se colocara es ",
final_detector_position)
                #este codigo es el que hay que escibir para el fichero
con los atributos de cada detector
                detector_xml = detectors_xml.addChild("e1Detector")
#aqui se crea un fichero por cada loop
                detector_xml.setAttribute("file", options.results)
#esto el el nombre del fichero, esto sera el fichero de nombre propio
del detector
                detector_xml.setAttribute("freq",
str(options.frequency))  #esta el la frecuencia del detector, lo pondre
seguramente en 1000
                detector_xml.setAttribute("friendlyPos", "x")
                detector_xml.setAttribute("id", "e1det1_" +
```

```
str(lane_id))
                detector_xml.setAttribute("lane", str(lane_id))
                #detector_xml.setAttribute("length",
str(final_detector_length))#esto es para definir la longitud del
detector normalmente se dejara por defecto, DA ERROR POR ESO LO COMENTO
                detector_xml.setAttribute("pos",
str(final_detector_position))  #esta es la posicion donde voy a poner el
detector, es un caso sera a un porcentage del inicio, y en otro caso
sera a un porcentage del final

                #aqui para el segundo detector
                final_detector_position = lane_length * porcentaje2  #es
decir dejo esta longitud al 95% de la longitud de la linea
                print("la posicion en que se colocara es ",
final_detector_position)
                detector_xml = detectors_xml.addChild("e1Detector")
#esto dice el tipo de detector que queremos crear
                detector_xml.setAttribute("file", options.results)
                detector_xml.setAttribute("freq",
str(options.frequency))
                detector_xml.setAttribute("friendlyPos", "x")
                detector_xml.setAttribute("id", "e1det2_" +
str(lane_id))
                detector_xml.setAttribute("lane", str(lane_id))
                #detector_xml.setAttribute("length",
str(final_detector_length))
                detector_xml.setAttribute("pos",
str(final_detector_position))  #esto habria que editarlo para poner 2
decectores por linea uno al final y otro al principio

            if lane_id2 in lanes_with_detectors2:#esto te dice que el
detector para una linea detecminada ya ha sido generado
                logging.warn("Detector for lane %s already generated"
%(str(lane_id2)))
                #continue
            else:
                lanes_with_detectors2.add(lane_id2)
                final_detector_position = lane_length2 * porcentaje1
#es decir dejo esta longitud al 2% de ela longitud de la línea por
ejemplo
                detector_xml = detectors_xml.addChild("e1Detector")
#aqui se crea un fichero por cada loop
                detector_xml.setAttribute("file", options.results)
#esto el el nombre del fichero, esto sera el fichero de nombre propio
del detector
                detector_xml.setAttribute("freq",
str(options.frequency))  #esta el la frecuencia del detector, lo pondre
seguramente en 1000
                detector_xml.setAttribute("friendlyPos", "x")
                detector_xml.setAttribute("id", "e1det1_" +
str(lane_id2))
                detector_xml.setAttribute("lane", str(lane_id2))
                #detector_xml.setAttribute("length",
str(final_detector_length))#esto es para definir la longitud del
detector normalmente se dejara por defecto, DA ERROR POR ESO LO COMENTO
                detector_xml.setAttribute("pos",
str(final_detector_position))  #esta es la posicion donde voy a poner el
detector, es un caso sera a un porcentage del inicio, y en otro caso
sera a un porcentage del final

                #aqui para el segundo detector
```

```
                final_detector_position = lane_length2 * porcentaje2
#es decir dejo esta longitud al 95% de la longitud de la línea por
ejemplo
                print("la posicion en que se colocara es ",
final_detector_position)
                detector_xml = detectors_xml.addChild("e1Detector")
#esto dice el tipo de detector que queremos crear
                detector_xml.setAttribute("file", options.results)
                detector_xml.setAttribute("freq",
str(options.frequency))
                detector_xml.setAttribute("friendlyPos", "x")
                detector_xml.setAttribute("id", "e1det2_" +
str(lane_id2))
                detector_xml.setAttribute("lane", str(lane_id2))
                #detector_xml.setAttribute("length",
str(final_detector_length))
                detector_xml.setAttribute("pos",
str(final_detector_position))

    #esto  ultimo es para manipular el fichero
    detector_file =
open_detector_file(get_net_file_directory(options.net_file),options.outp
ut)#aqui escribo en un fichero conjunto todos los loops
    detector_file.write(detectors_xml.toXML())
    detector_file.close()

    logging.info("%d e1 detectors generated!" %
len(lanes_with_detectors))
#aqui vamos a quitar los duplicados de la lista

    lines_seen = set()  # holds lines already seen
    outfile = open(ruta+nombre+".add.xml", "w")#aqui creo el nombre del
fichero nuevo sin los duplicados
    print("EL FICHERO DE SALIDA ES ",ruta+nombre+".add.xml")
    print("el fichero input es ",options.output)
    options.output=ruta+"pruebacruce2loopsduplicados.add.xml"
    for line in open(options.output, "r"):
        if line not in lines_seen:  # not a duplicate
            print("la linea es ",line)
            outfile.write(line)#escribo la linea
            lines_seen.add(line)#añado la linea a la lista de lineas
vistas
    outfile.close()
```

8.10 Código get_semloops

```
def get_semloops(ficherored):  #esta funcion lo que hace es que te
proporciona la lista de semaforos con sus loop

    numloops = traci.inductionloop.getIDCount()
    loops = traci.inductionloop.getIDList()
    net = sumolib.net.readNet(ficherored)
    semaforos = net.getTrafficLights()
    if (numloops == 0):
        raise ValueError('no hay aros inductivos')
    #en la primera iteracion obtengo los semaforos de la red , cuantos
hay y un objeto lista que los contiene
```

```
    trafli = traci.trafficlight.getIDList()
    if (len(semaforos) == 0):
        raise ValueError('no hay semaforos')



    detectorstrafli = []  # vamos a crear una doble sumlista que dira
los semaforos que hay junto con sus detectores y su sentido(si es de
entrada o salida al nodo), el sentido lo sabemos si esta en el primer o
2 indice
    for i in semaforos:  #aqui iterao para los diferentessemaforos
        #print("los nodos que controla el semaforo ", trafli[i], " son
",traci.trafficlight.getControlledLinks(trafli[i]))
        linescontrolled =
traci.trafficlight.getControlledLinks(i.getID())  #asi obtengo los nodos
que controla el semaforo, (LO QUE PASA ES QUE VIENE EN UNA TRIPLE
SUBLISTA)
        print("las lineas que controla el semaforo ", i.getID(), " son
", linescontrolled, " con una longitud de la lista ",
len(linescontrolled))
        #ahora de estas lineas que controla tengo que mirar en todos los
loops cuales estan en estas lineas
        if(len(i._connections)>1): #si hay mas de una conexion, es
cuando nos interesa procesar el semaforo
            sublist1 = []
            sublist1.append(i.getID())  # creamos una lista del nodo al
que pertenecen, en el que iremos añadiento los loops
            for j in range(0, numloops):  #aqui voy probando todos los
loops para ver si pertenecen a un semaforo
                for k in range(0, len(linescontrolled)):  #esto lo
utilizo para acceder a todos los elementos de la sublista(los loops), es
decir pruebo para cada semaforo todos los loops
                    #print("la linea del loop es
",traci.inductionloop.getLaneID(loops[j])), probando con todos los
carriles que controla el el semaforo, es decir hago un triple
                    if (traci.inductionloop.getLaneID(loops[j]) in
linescontrolled[k][0]) & (
                            loops[j] not in sublist1):  #es una sublista
triple lo que te dice el elemento final de la sublista es carril origen
carril destino e id conexion para el semaforo
                        #compruebo si el carril del loop estan en las
lineas que controla el semaforo & compruebo que  el loop no este ya en
la sublista
                        #luego si esta en el primer elemento (0) es un
carril que entra al nodo si es un 1 es un carril que sale del nodo
                        #con este if obtenemos los los loops que estan
en una carrilo controlado por el semaforo(si hay varios habra que
quedarse con uno)
                        if (True):  #ahora tengo que mirar si el numero
de lineas que hay en ese carril es 1 o mas, en cuyo caso tendria que
quedarme solo con uno de ellos
                            if
(linescontrolled[k][0].index(traci.inductionloop.getLaneID(loops[j])))
== 0:  #esto me sirve para distinguir si son loops de entrada o de
salida al nodo
                                sublist1.append(loops[j])  #aqui le
añado el elemento j que pertenece a esas lista de detectores
                                sublist1.append("in")  #y le añado un
elemento indicador que es de entrada
                            if
```

```
(linescontrolled[k][0].index(traci.inductionloop.getLaneID(loops[j])))
== 1:  #esto me sirve para distinguir sin son loops de entrada o salida
el nodo
                                sublist1.append(loops[j])  #aqui le
añado el elemento j que pertenece a esas lista de detectores
                                sublist1.append("out")  #y le añado un
elemento indicador que es de salida

            #print("tiene ", (len(sublist1) - 1) / 2, " detectores en la
lista")
            detectorstrafli.append(sublist1)  #lo añado a la lista de
cada semaforo
        #sublist1.clear()#aqui limpio la lista para el siguiente
semaforo

    return detectorstrafli  #devuelvo la lista de semaforos asociados y
loops
```

8.11 Código obtener vehículos paso

```
def obtenervehiculospaso(listasemyloops, listavehiculosporsemaforo,
listacorrespondenciarutas, listavehiculosporruta,
listanodoorigenydestinototal,
                         finalciclo):  #lo que va a hacer es que crea un
hilo por cada semaforo que hay para asi hacer el precesamiento
simultaneo y que sea mas rapido y asi simula un sistema distribuido
    #lo primero que miro son el numero de elementos de esta lista


    #***********creo que la lista de vehiculos por semaforo, hay que
crear trantas sublistas como semaforos, y por cada semaforo tantas como
aros****************

    def verflujo(listaloopsporsemaforo, listavehiculosenloop,
listacorrespondenciarutasindividual, listavehiculosporrutaindividual,
listanodoorigenydestinoindividual):  #para un semaforo, le hemos pasado
unicamente su informacion especifica
        #aqui ponemos el codigo que queremos hacer para cada hilo, lo
que tengo que hacer es ver la longitud de la sublista y comparar los IDs
        #la listavehiculosenloop contiene los vehiculos que han pasado
por un loop
        #la llista corresponendeica rutas individual, contiene la
correspondencia de loops a rutas
        #la lista vehiculos por ruta es analogo a lista vehiculosenloop
pero para las rutas
        #la listanodoorigenydestino individual
        #la listanodororigenydestino, contiene una lista de las rutas y
sus nodos origen y destino

        print("\n\n\nla lista 1 es \n",listaloopsporsemaforo)
        print("la lista 2 es \n",listavehiculosenloop )
        print("la lista 3 es \n", listacorrespondenciarutasindividual)
        print("la lista 4 es \n",listavehiculosporrutaindividual )
        print("la lista 5 es \n",listanodoorigenydestinoindividual )

        if (len(listavehiculosporrutaindividual) <
(len(listanodoorigenydestinoindividual) - 1) / 3):  #si es menor, tengo
que añadir a la lista en blanco las rutas
```

```
            for i in range(0,
int((len(listanodoorigenydestinoindividual) - 1) / 3)):  #inicializo la
lista de las rutas, para ver que rutas han sido usadas por que vehiculos
                listavehiculosporrutaindividual.append(list())

listavehiculosporrutaindividual.insert(0,listaloopsporsemaforo[0])#le
inserto el ID del semaforo al principio de la lista


            #*****************he añadido el ID---->hay que modificar como
se llen las listas o tal



        print("la lista de vehiculos por ruta individual es
",listavehiculosporrutaindividual)#************esta longitud tienen que
se igual que el numero de rutas

        for i in range(1, len(listaloopsporsemaforo) - 1, 2):#para cada
loop del semaforo
            if (len(listavehiculosenloop) < (len(listaloopsporsemaforo)
- 1) / 2):  #si es menor que la sublista tengo que crear instancias,la
longitud tiene que ser igual que el numero de flujos entrantes + flujos
salientes
                listavehiculosenloop.append(list())  #aqui añado una
sublista en blanco
                print("la longitud de la lista  de vehiculosenloop es ",
len(listavehiculosenloop))
            listapaso =
traci.inductionloop.getLastStepVehicleIDs(listaloopsporsemaforo[i])
#con esto obtengo la lista de vehiculos en este paso por un loop del
semaforo



            """
            for j in range(0, len(listapaso)):
                print("LA ACELERACION MAXIMA ES  es ",
traci.vehicle.getAccel(listapaso[j]))"""







            #qui tengo que hacer un cambio de nuevo a los indices para
adaptarlo a unos indices normales y ahorrar espacio


            #con esto elimino los duplicados
            listadiferencia =
set(listapaso).difference(set(listavehiculosenloop[int((i - 1) / 2)]))
#con esto hago que aunque en dos pasos aparezca mas de una vez, que no
se añada el mismo de manera extra

            listavehiculosenloop[int((i - 1) /
2)].extend(list(listadiferencia))  #las posiciones de esta lista vienen
dadas por las posiciones en que estan guardados los loops en ese
semaforo, añado aqui los nuevos vehiculos que acaban de pasar en el
ultimo paso
            #print("\n\n\nla lista diferencia es ",
```

```
listavehiculosenloop[int((i - 1) / 2)])
        print("la lista vehiculos en loop es \n",listavehiculosenloop)




        #una vez que termina el bucle ya tengo en la lista todos los
ID's de los coches en ese paso añadido a los que tenia antes, ahora lo
que tengo que hacer es procesarla para vez que vehiculos permanecen en
cada lado

        print("la lista de vehiculos sin quitar
\n",listavehiculosenloop)

        #lo que tengo que hacer ahora es comparar entre los loops para
ver si coinciden quitarlo de la lista de entrada y la lista de salida es
decir tengo que comparar todos los loops menos si mismo
        #el bucle de esta manera evita automaticamente los elementos
repetidos y buscar por duplicado


        #aqui i y j son las rutas del semaforo



        minimo = 0  #el primer indice de la lista
        maximo = len(listavehiculosenloop)  #el indice maximo de la
lista
        k = minimo
        l = k + 1
        i = minimo
        j = i
        while k < maximo:
            j = k + 1
            l = k + 1
            while l < maximo:

                #de esta manera lo elimino de las 2 listas, de la de
entrada y de la de salida de los vehiculos a la vez
                auxiliar = listavehiculosenloop[i]  #copio la lista

                #lo que tengo que hacer es buscar que elementos son
iguales, como esta abajo, reemplazando diferencia por coincidencia, y
con los indices i,j se cual es la entrada y la salida
                #luego con esto puedo obtener que ruta ha seuid, y crear
una lista por cada iteacion, que te diga cuantos vehiculos han ido por
cada ruta
                listacoincidencia =
list(set(listavehiculosenloop[i]).intersection(set(listavehiculosenloop[
j])))  #asi tengo las coincidencias, una vez que las tengo, mirando el
indice de la posicion si es entrada o salida
                #el loop, y por tanto mirando la lista de loops ruta, ya
se que ruta ha seguido, con los indices i y j, tengo la posicion de la
originales de la listasem loop, los loops que son, y por tanto los ids
de los
                #coches que han ido por ahi
                if (len(listacoincidencia) > 0):
                    #print("la lista de correspondencias es
```

```
\n",listacorrespondenciarutasindividual)
                    #hay que darse cuenta que la lista de
correspondencias esta ordenada, es decir, mirando el numero de veces que
sale ese loop en la lista veo cuantas veces como mucho hay que
desplazarse en la lista
                    #para comprobar todas las posibilidades
                    #print(2*i+1,2*j+1)
                    indice1 =
listacorrespondenciarutasindividual.index(listaloopsporsemaforo[2 * i +
1])  #obtnemos el id del loop, obtnemos su indice, que sabemos que la
funcion index, te la primera coincidencia, como mucho hay que
desplazarse
                    #le numero de veces que ese loop aparece en la
lista, ya que en la lista los loops, estan ordenados por nombre de forma
consecutiva, tengo que recorrer todas las posibilidaddes, hasta que de
con la ruta
                    indice2 =
listacorrespondenciarutasindividual.index(listaloopsporsemaforo[2 * j +
1])
                    #print("los indices son",indice1, " y ",indice2)
                    #print("las veces que aparecen son
",listacorrespondenciarutasindividual.count(sublista[2*i+1]),"  y
",listacorrespondenciarutasindividual.count(sublista[2*j+1]))
                    for p in range(0,
listacorrespondenciarutasindividual.count(listaloopsporsemaforo[2 * i +
1])):
                        for q in range(0,
listacorrespondenciarutasindividual.count(listaloopsporsemaforo[2 * j +
1])):
                            #print("los resultados es
",listacorrespondenciarutasindividual[indice1+2*p-1],"
",listacorrespondenciarutasindividual[indice2+2*q-1])#el -1 es para
acceder al id de la ruta
                            if
((listacorrespondenciarutasindividual[indice1 + 2 * p - 1]) ==
(listacorrespondenciarutasindividual[indice2 + 2 * q - 1])):
                                print("coinciden en la ruta ",
listacorrespondenciarutasindividual[indice1 + 2 * p - 1], " los
vehiculos con ID's ", listacoincidencia)

listavehiculosporrutaindividual[listacorrespondenciarutasindividual[indi
ce1 + 2 * p - 1]+1].extend(listacoincidencia)
                    #si se abren rutas copatibles, en un mismo paso de
la simulacioón puedo obtener varios vehiculos en rutas diferentes, pero
estas rutas, siempre son rutas compatibles

                listavehiculosenloop[i] =
list(set(listavehiculosenloop[i]).difference(set(listavehiculosenloop[j]
)))  #miro las diferencias entre las 2 listas
                #las diferencias, son lo que me tengo que quedar, ya que
lo que se repite es es donde han salido y han entrado  coches iguales

                listavehiculosenloop[j] =
list(set(listavehiculosenloop[j]).difference(set(auxiliar)))  #AQUI NO
ES COMPARAR CON EL MISMO CONJUNTO SINO CON EL AUXILIAR, YA QUE EL
CONJUNTO i YA ESTA EDITADO,
                #para lograr esto lo mas sencillo es utilizar una
variable auxiliar, haciendo todo esto la lista de los loops de salida
tambien se ira vaciando a medida que van saliendo los vehiculos

                j += 1  #esto es para ir recorriendo los elementos
```

```
                l += 1  #esta es la condicion para salir del while
            i += 1  #esto es para ir recorriendo los elementos
            k += 1  #este elemento es la condicion para que termine el
primer while

        #print("la lista de vehiculos quitada\n",listavehiculosenloop)

        #OJOOOO,la lista de vehiculos aparece solo informacion en las
posiciones impares, una vez que quitas los vehiculos que han ido
saliendo, es decir despues de esta doble iteriacion, ESTA BIEN


        #ademas, sabemos que se puede hacer en una lista con tamaño
        return  #cuando termina el hilo vuelvo a obtenervehiculospaso









    #SE EJECUTA ESTO DE ABAJO PRIMERO YA QUE LO DE ARRIBA ES UNA
INSTANCIACION DE UN METODO
    threads = list()  #creo una lista con los hilos
    for i in range(0, len(listasemyloops)):  #aqui creo tantos hilos
como semaforos tenga
        if (len(listavehiculosporsemaforo) < len(listasemyloops)):  #si
no se han presituado las sublistas añado una nueva dimension a la lista
(es decir es para presituar las dimensiones
            listavehiculosporsemaforo.append(list())  #añadimos una
nueva lista por cada semaforo
            #print("la longitud de la lista  de vehiculosporsemaforo es
",len(listavehiculosporsemaforo))
        if (len(listavehiculosporruta) < len(listasemyloops)):
            listavehiculosporruta.append(list())  #añado una nueva lista
por cada semaforo
        #necesitamos buscar el ultimo indice para que lo pase
correctamente la lista,extraigo el primer elemento de todaslas listas y
busco el indice

        j=list(map(itemgetter(0),
listanodoorigenydestinototal)).index(listasemyloops[i][0])#asi obtenengo
el primer elemento de todas las sublistas
        obtenervehiculospaso =
threading.Thread(verflujo(listasemyloops[i],
listavehiculosporsemaforo[i], listacorrespondenciarutas[i],
listavehiculosporruta[i],listanodoorigenydestinototal[j]))  #aqui lo que
hago es crear un hilo para cada semaforo en el que medira el flujo de
trafico de dicho semaforo
        threads.append(obtenervehiculospaso)  #añado el hilo a una lista
        obtenervehiculospaso.start()  #inicio el hilo

        if (finalciclo):  #si es el final de ciclo, tengo que añadir al
final de cada sublista un caracter especial de indicacion de final de
            for j in range(1, len(listavehiculosporruta[i])):
                listavehiculosporruta[i][j].extend('#')  #este es el
caracter especial limitador de final de ciclo
```