Universidad Zaragoza

Trabajo Fin de Grado

Simulación de Redes de Vehículos desde una Perspectiva de Gestión de Datos

Vehicle Network Simulation from a Data Management Perspective

Autor

Alvaro Echavarri Sola

Director

Sergio Ilarri Artigas

GRADO EN INGENIERÍA INFORMÁTICA – RAMA DE COMPUTACIÓN
ESCUELA DE INGENIERÍA Y ARQUITECTURA
2022

AGRADECIMIENTOS

Para comenzar me gustaría agradecer a mi tutor, Sergio Ilarri, su orientación, apoyo y su completa disposición a la hora de resolver mis dudas e inquietudes durante la realización de todo el proyecto.

También agradecer a todos aquellos profesores que han dedicado su esfuerzo en impartirme las diferentes asignaturas que he cursado a lo largo de la carrera.

Por último, agradecer a mi familia y amigos por estar día a día conmigo y ayudarme a superar los problemas. No habría llegado hasta aquí si no fuera por ellos.

RESUMEN

Hoy en día, conocemos diferentes enfoques que intentan resolver con precisión el complejo problema de la simulación de redes vehiculares. El más utilizado busca utilizar un simulador de tráfico para generar trazas de movilidad vehicular realistas que se utilizarán como entrada para un simulador de red móvil ad hoc. Otra opción sería utilizar una herramienta de simulación de redes vehiculares especialmente diseñada para ello. Finalmente, otra opción bastante llamativa es que algunos entornos de programación MANET permiten al desarrollador probar las aplicaciones VANET a través de simulaciones.

El objetivo de este trabajo es analizar diferentes simuladores presentes en el mercado y comparar su comportamiento y rendimiento para poder obtener una imagen general del estado de los simuladores de VANETs en la actualidad. Existen generalmente 3 enfoques a la hora de crear un simulador de redes VANET: generar trazas de movilidad vehicular y simularlas como una red móvil, crear un programa VANET como tal que cree y simule la red de coches o construir un simulador de VANETs dentro de una aplicación MANET que proporcione las herramientas necesarias. Los simuladores que se han elegido para este análisis comparativo son: SUMO, VanetMobisim, Veins y MAVSIM. Estos serán explicados con mayor profundidad en sus respectivos apartados.

Para realizar el análisis de los simuladores de VANETs el primer paso ha sido instalar y configurar cada uno de ellos. SUMO ha sido el más sencillo de preparar, en Linux se puede instalar con apt-get y funciona perfectamente. Veins es algo complejo, ya que primero hay que hacer funcionar el entorno OMNet++ y luego importar las librerías. Con Mavsim tuve algunos problemas para compilarlo, ya que es necesario utilizar una versión de Java en concreto, pero una vez compila todo va perfectamente. Por último, VanetMobisim es con diferencia el peor de todos a la hora de instalar y configurar, la documentación sobre su instalación es inexistente, el código fuente no se puede conseguir y no hay ningún ejemplo en su página oficial. Tras tener todos los programas listos fue necesario escribir un programa en Python que me ayudara a generar las pruebas en el formato de cada simulador. Por último, se decidió qué parámetros se iban a modificar y qué resultados queríamos guardar para la posterior comparación (tiempo de simulación/ejecución, número de coches, rutas aleatorias/no aleatorias, uso de RAM/CPU, etc.) .

Índice

1. Introducción y objetivos **1**

1.1. Motivación 1

1.2. Objetivos 3

1.3. Metodología y herramientas 4

2. Simulación de una VANET **5**

2.1. Simulación VANET 5

2.2. Parámetros de simulación 6

2.2.1. Tiempo simulado 6

2.2.2. Paso de simulación 6

2.2.3. Número de coches 7

2.3. Recursos computacionales 7

2.4. Ejecución de la simulación 8

2.4.1. Velocidad de ejecución 8

2.4.2. Bloqueos y errores de ejecución 9

3. Simuladores de VANETs y programa en Python **11**

3.1. Instalación y uso de los programas 12

3.2. Simuladores 12

3.2.1. SUMO 12

3.2.2. MAVSIM 13

3.2.3. Veins 14

3.2.4. VanetMobisim 16

3.3. Programa createNet.py 18

4. Análisis comparativo **20**

4.1. SUMO 21

4.1.1. Facilidad de uso y ejecución 21

4.1.2. Rendimiento 22

4.1.3. Efectividad 26

4.2. Veins . . . 27
4.2.1. Facilidad de uso y ejecución . . . 27
4.2.2. Rendimiento . . . 28
4.2.3. Efectividad . . . 30
4.3. MAVSIM . . . 31
4.3.1. Facilidad de uso y ejecución . . . 31
4.3.2. Rendimiento . . . 33
4.3.3. Efectividad . . . 35
4.4. VanetMobisim . . . 35
4.4.1. Facilidad de uso y ejecución . . . 36
4.4.2. Rendimiento . . . 37
4.4.3. Efectividad . . . 40
4.5. Comparativa . . . 41
4.5.1. Gráfica comparativa de mapas aleatorios . . . 41
4.5.2. Uso de los recursos . . . 42
4.5.3. Rendimiento en tiempo . . . 44
4.5.4. Facilidad de uso e interfaz . . . 44

5. Conclusiones **46**

6. Trabajo futuro **47**

7. Bibliografía **48**

Lista de Figuras **49**

Anexos **51**

A. Gestión del proyecto **52**

B. Imágenes de mapas aleatorios de apartado 4.5 **54**

C. Documento de pruebas **59**

Capítulo 1

Introducción y objetivos

1.1. Motivación

El paradigma del tráfico y su simulación presenta diferentes factores a tener en cuenta; por ejemplo, se debe representar de forma precisa las carreteras y sus intersecciones al igual que su orientación y posibles obstáculos como otros coches o semáforos. Una red VANET [1] (Vehicular Ad Hoc Networks en inglés) es un tipo de red de comunicación que utiliza a los vehículos como nodos de la red. Dado el reducido alcance de su comunicación (1 km), la conectividad se establece de forma esporádica. Por este motivo, estas redes se consideran un tipo específico de red móvil de comunicación o MANET[2]. Sin embargo, suponen una serie de problemas adicionales a la red MANET típica, como la alta volatilidad de las redes, la velocidad de los nodos y la concentración de los nodos en un área pequeña.

En el mercado actual existen una gran variedad de *software* que intenta reproducir este paradigma de redes VANET. Mediante diferentes desarrollos, cada programa intenta acercarse al problema de diferente forma. Algunos realizan una aproximación más directa creando una aplicación que represente esta comunicación y tráfico de los coches desde cero. Por otro lado, otros intentan aplicar algunas de las tecnologías existentes (entornos de MANET como OMNet++ o CanuMobiSim) para generar escenarios que simulen estos escenarios de VANET[3].

Para este proyecto se ha decidido seleccionar para analizar 4 simuladores existentes en el mercado actual de la simulación de redes vehiculares:

- SUMO → Simulador más usado que usa su propio framework para sus simulaciones. [1]
- Veins → Librerías para el entorno OMNet++ (MANET) que permiten crear simulaciones VANET (utiliza Sumo internamente). [2]
- MAVSIM → Aplicación Java desarrollada en la EINA que crea y representa simulaciones VANET. [3]
- VanetMobisim → Programa en Java que reproduce simulaciones VANET ya definidas, sin muchos ajustes posibles. [4]

[1] https://www.eclipse.org/sumo/about
[2] https://veins.car2x.org
[3] http://webdiis.unizar.es/~silarri/prot/MAVSIM/
[4] https://neo.lcc.uma.es/staff/jamal/vanet/index.html

1.2. Objetivos

El objetivo buscado en este trabajo es explorar la simulación de redes vehiculares en cada uno de los programas elegidos, observar su comportamiento, formato de simulación, resultados y salida mostrada, errores de ejecución observados, etc. Todos estos datos nos permitirán perseguir los siguientes objetivos propuestos:

- Experimentación de los simuladores y sus herramientas disponibles.
- Análisis de los resultados obtenidos, precisión de la simulación, comportamiento y manejo de los programas, rendimiento (tanto en tiempo como en coste computacional, RAM/CPU) y facilidad de instalación y uso.
- Comparativa final entre los simuladores evaluados, tanto cuantitativa como cualitativa. Como reaccionan a diferentes mapas reales o aleatorios, efecto de los parámetros (si es que dejan modificar parámetros) y eficiencia a la hora de ejecutar estas simulaciones.
- Llegar a una conclusión del estado del paradigma de la simulación VANET en el mercado actual. Discutir cómo de factible es su puesta en marcha en un entorno de tráfico real.

1.3. Metodología y herramientas

El proyecto consiste en la exploración y experimentación de los diferentes programas para formar un análisis comparativo y formar una visión global del estado de la simulación de VANETs. Para ello va a ser necesario instalar y configurar cada simulación para cada uno de los simuladores y, de alguna forma, procesar el rendimiento y resultado observado. Para ello se han seguido los siguientes métodos y herramientas:

- Desarrollo e implementación en Python del programa encargado de leer los mapas OSM y generar las respectivas simulaciones de cada simulador. Permite semi-automatizar la ejecución de las pruebas.
- El diferente software necesario para la instalación y uso de cada simulador: Sumo (Java), Veins (OMNet++[5], Java y Sumo), VanetMobisim (Java) y Mavsim (Java).
- Representación de los resultados mediante gráficas y tablas que permitan establecer los datos a gestionar y facilite la comparacion entre los resultados obtenidos de cada programa.
- Mapas fuente extraídos de la librería de OpenStreetMap (Overpass API) [6]
- LaTeX para la redacción del documento.

[5] `https://omnetpp.org/`
[6] `https://www.openstreetmap.org/`

Capítulo 2

Simulación de una VANET

2.1. Simulación VANET

La representación de una simulación de VANET es un conjunto de datos de diferentes formatos que permiten representar esta red VANET. Incluyendo sus actores móviles, nodos, aristas y rutas que la componen. Cada programa sigue su propio formato; algunos prefieren una representación basada en archivos de configuración y otros utilizar su interfaz. Estos formatos siguen una idea general basada en: identificar los nodos por su posición y nombre, dibujar aristas entre ellos (representando segmentos de carreteras), definir las rutas y el número de coches que van a recorrerlas y ejecutar la simulación mediante eventos o iteraciones.

En la figura 2.1 se puede observar el fichero de configuración de una simulación en su formato más utilizado. En él quedan definidas las partes de la simulación. El fichero de red (*net-file*) define los nodos y rutas de la simulación y el fichero de rutas (*route-files*) define las rutas que seguirán los coches a través de la red.

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
    <input>
        <net-file value="Random.net.xml"/>
        <route-files value="Random.rou.xml"/>
    </input>
    <time>
        <begin value="0"/>
        <end value="10000"/>
    </time>
</configuration>
```

Figura 2.1: Configuración de una simulación en *SUMO*

2.2. Parámetros de simulación

2.2.1. Tiempo simulado

El tiempo de simulación indica la duración de la simulación, que no es lo mismo que la duración real de la simulación. Este parámetro afecta en todos los aspectos al resultado de la simulación, decide cuándo se detiene la simulación y esto puede provocar que termine la ejecución antes de tiempo o, por el contrario, que termine mucho más tarde que los coches simulados.

Por otro lado, cada simulador tiene una forma de interpretar el tiempo de simulación. SUMO y VanetMobisim, por ejemplo, no terminan la simulación hasta que se acaba el tiempo, pero Veins termina la simulación en cuanto se acaban las rutas, sin esperar al tiempo. MAVSIM tiene su propia forma de interpretar el tiempo mediante el número máximo de iteraciones y tiempo de simulación por iteración. Este tratamiento del tiempo es tan distinto del de los otros programas que hace imposible compararlos de igual forma.

Normalmente, se hace variar el tiempo de simulación máximo y el número de coches para ver cuanto tiempo necesita el simulador para ejecutar las rutas. Sin embargo, esto no tiene sentido cuando la ejecución de una simulación está limitada por el número de iteraciones que puede hacer. Esta ejecución siempre se detendrá tras X iteraciones, da igual cuanto haya ejecutado del total.

2.2.2. Paso de simulación

El tema del paso de simulación es algo complicado, ya que cada simulador trata sus iteraciones y como se ejecutan durante la simulación de diferente forma. El paso de simulación es un parámetro que indica cuanto transcurre entre iteraciones en una simulación. En SUMO y Veins el paso es de 100 ms, en MAVSIM la simulación se ejecuta hasta llegar a un número fijo de iteraciones así que lo que se ha hecho es ajustar el tiempo de cada iteración para aproximarlo al comportamiento de SUMO y Veins (100 ms de paso). VanetMobisim, por su parte, no permite ajustar ningún parámetro, excepto el tiempo de simulación. Por ello los tiempos se han ajustado lo máximo posible a otros simuladores.

2.2.3. Número de coches

El número de coches está a la par que el tiempo de simulación en cuanto a importancia a la hora de definir los parámetros de una simulación. Los coches son la base de la ejecución, recorren el mapa, siguen sus rutas e interaccionan entre sí. Sus iteraciones provocan variaciones en el rendimiento del programa a diferentes grados, según la forma que tiene el mapa o la densidad del tráfico.

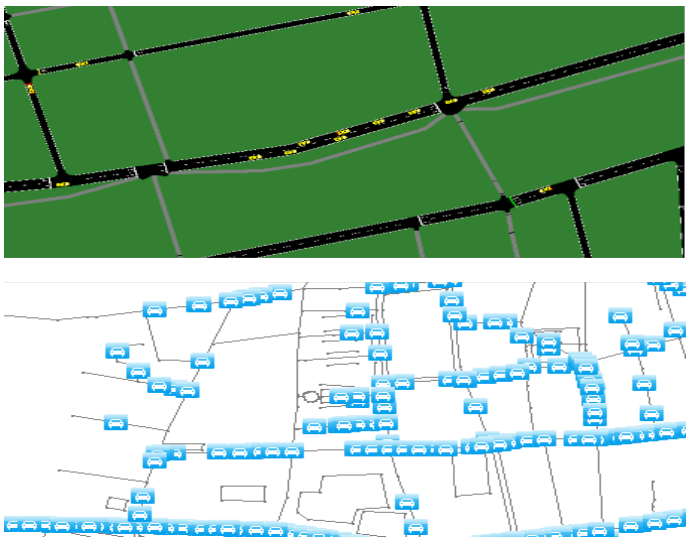


Figura 2.2: Representación de los coches

2.3. Recursos computacionales

Otro factor a tener en cuenta a la hora de ejecutar simulaciones como estas es la cantidad de recursos de memoria y uso del CPU que utiliza cada simulador en según qué situaciones. Esto puede dar otro punto de vista al rendimiento. En vez de medir el rendimiento de una simulación por el tiempo que cuesta hacer una simulación puede ser interesante comparar los recursos que utiliza cada programa para saber cómo de eficiente, en términos de cómputo, es cada uno.

De esta forma sabremos cómo de exigente es cada simulador con los recursos necesarios para que ejecute una simulación en concreto. No es lo mismo ejecutar un mapa con poco coste de CPU o RAM que uno con mucho coste. El coste de CPU es el que limita el tiempo real de ejecución. Una simulación que use mucho CPU tardará más en completarse que otra que exija menos, debido a que exige más al ordenador y puede ralentizar su ejecución con mayor facilidad (sobre todo cuando se trata con mapas muy grandes o con ejecuciones de horas de duración).

2.4. Ejecución de la simulación

Una simulación siempre empieza de la misma forma. Se carga cada nodo y arista del mapa, después se leen las rutas y se colocan los coches al principio de sus rutas. Algunas de las propiedades de la ejecución de una simulación son la velocidad o modo de ejecución y el tratamiento de bloqueos/errores de cada simulador.

2.4.1. Velocidad de ejecución

La velocidad de ejecución o de simulación está relacionada con el parámetro de paso, aunque no afecte directamente a su funcionamiento. Esta velocidad acelera artificialmente el programa para que el tiempo de simulación transcurra más rápido. De esta forma es posible simular las mismas iteraciones en un lapso menor de tiempo y ejecuciones que tardarían horas se reducen a minutos. Cada simulador trata la velocidad de ejecución de diferente forma:

- **SUMO:** SUMO solo permite 1 modo de velocidad. Sin embargo, se puede detallar en cada arista la velocidad de los coches en ese tramo. Esto provocará, que según la ruta de cada coche, algunos terminen más rápido su recorrido que otros, pero no hay ninguna forma de cambiar la velocidad de la simulación una vez iniciada.
- **Veins:** El entorno que utiliza Veins para lanzar sus simulaciones, OMNet++, permite una gran variedad de velocidades:
  - *Step*, que permite ir iteración a iteración.
  - *Run*, ejecuta la simulación mostrando en detalle cada iteración y evento.
  - *Fast*, ejecuta la simulación sin mostrar algunas animaciones.
  - *Express*, ejecuta la simulación ignorando todas las animaciones y *output* de texto
  - *Until*, ejecuta la simulación en modo *Express* hasta cierto evento especificado.

  Cada modo permite ejecutar la simulación de un modo u otro, el tiempo de simulación sigue siendo el mismo, pero el tiempo real varía mucho. Esto puede resultar muy útil cuando quieres obtener diferentes resultados de una misma ejecución (medir tiempos, CPU, etc.).

– **MAVSIM:** El simulador MAVSIM permite ejecutar la simulación con dos modos de velocidad: normal y rápido. Mediante el modo normal se puede observar cada paso de los coches de forma más clara. Por otro lado, con el modo rápido todo es acelerado y es casi imposible seguir los movimientos de un coche: está más pensado para observar el comportamiento general del tráfico y para obtener resultados experimentales.

– **VanetMobisim:** Este simulador no permite ninguna opción mediante interfaz gráfica y mediante terminal las opciones son muy reducidas. En su documentación no especifica ningún modo de velocidad ni cómo variar el paso. Solo permite establecer el tiempo de la simulación.

Figura 2.3: Modos de velocidad de Veins

2.4.2. Bloqueos y errores de ejecución

Uno de los problemas más comunes durante la ejecución de una simulación de VANET son los bloqueos causados por las iteraciones entre los coches. Estos bloqueos son provocados por varios motivos: el tráfico actual no permite tomar el siguiente paso en la ruta o el coche llega a un camino sin salida y no puede terminar su ruta. Los bloqueos pueden causar problemas serios que acaben con la ejecución de la simulación. Por tanto, es importante tratar estos errores de la manera más rápida y eficiente.

Cada simulador tiene su forma de tratar los bloqueos. Algunos los solucionan modificando la simulación y otros simplemente los muestran por terminal y los ignoran.

– **SUMO:** SUMO trata los bloqueos mediante traslaciones, de modo que cuando un coche se encuentra en un bloqueo (no puede completar su ruta) predeterminado a 100 ms de simulación, este lo traslada a otra vía para que prosiga su ruta. Existe un máximo de iteraciones permitido en una ejecución. Si se sobrepasa este número de iteraciones se da la ejecución como fallida y se detiene.

– **Veins:** El entorno OMNet++ no evita los bloqueos. Si ocurren, saca un mensaje por terminal y prosigue la simulación ignorándolos. Es más, es difícil ver los bloqueos cuando ocurren, ya que los eventos se tratan de forma breve.

– **MAVSIM:** En MAVSIM no es posible que ocurran bloqueos. Los coches siguen sus rutas sin importar la posición de los otros coches y las rutas que se generan siempre es posible que se completen.

– **VanetMobisim:** El simulador VanetMobisim no trata los errores de ninguna forma posible, ni siquiera tiene en cuenta las interferencias entre coches. Cuando ocurre un bloqueo, es decir, un coche no puede terminar su ruta, el programa se bloquea y termina la simulación sin advertencia.

Capítulo 3

Simuladores de VANETs y programa en Python

Para poder leer y ejecutar estas simulaciones de VANET se utilizan diferentes programas que leen y representan la red de vehículos. Por norma general, se sigue el formato descrito en el apartado 2.1: un archivo define la simulación y otros dos archivos definen el mapa y las rutas. Sin embargo, algunos simuladores siguen otros procedimientos o formato en sus archivos.

En este apartado se describirá el funcionamiento de cada simulador, es decir, el proceso necesario para que, partiendo desde el mapa OSM (un formato de mapa de código abierto), se pueda ejecutar una simulación de redes vehiculares con el número de coches y tiempo de simulación deseado.

3.1. Instalación y uso de los programas

3.2. Simuladores

3.2.1. SUMO

SUMO es el simulador de VANETs más utilizado del mercado, su instalación es muy sencilla y no requiere de ningún otro software externo como otros programas. Además, ofrece otras herramientas útiles a la hora de generar las simulaciones.

Para su instalación en Linux solo hay que utilizar apt-get con el siguiente comando: *apt-get install SUMO*. Tras esto la instalación está completa y ya acepta simulaciones. la simulación debe estar especificada con el formato ya introducido.

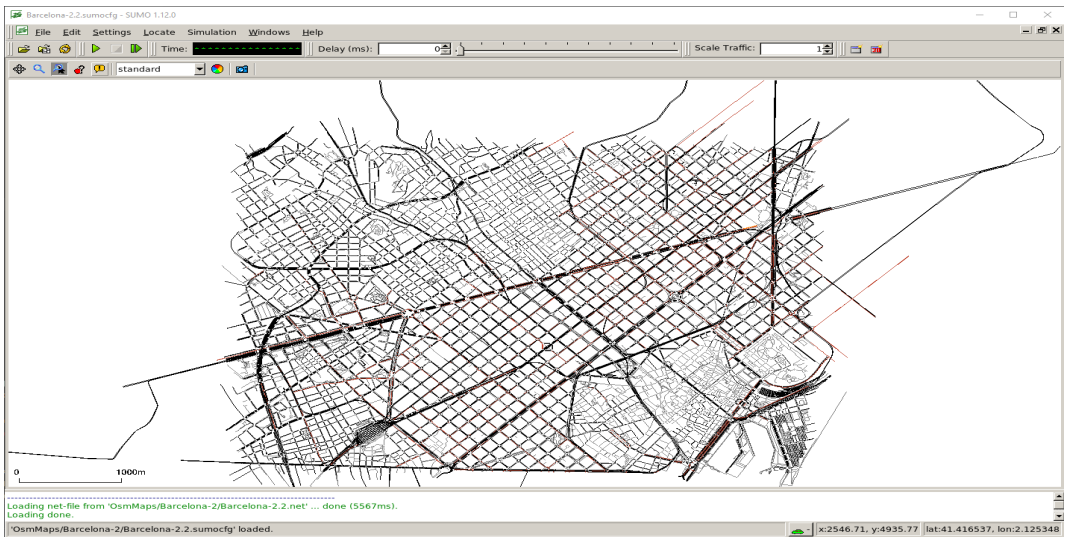


Figura 3.1: Programa SUMO iniciado

SUMO también ofrece una herramienta para ejecutar simulaciones en terminal aunque, obviamente, con menos detalles que su variante con interfaz, pero resulta interesante para pruebas automáticas de un largo periodo de tiempo. Otras herramientas que ofrece SUMO son muy útiles para crear simulaciones. randomTrip permite, a partir de un mapa y un número de coches, generar el archivo de rutas aleatoriamente, y otra herramienta llamada osmconvert actualiza el formato de los mapas OSM o los traduce a otros formatos. Estas herramientas han sido clave para ejecutar simulaciones en otros programas.

3.2.2. MAVSIM

Este simulador ha sido desarrollado por un equipo de la Universidad de Zaragoza y se puede encontrar en su repositorio oficial `http://webdiis.unizar.es/~silarri/prot/MAVSIM/`[4]. Para su instalación ese necesario clonar su GitHub y compilarlo mediante Apache Ant[1]. Todo este proceso está incluido en los scripts compile.sh y MAVSIM.sh proporcionados. Hay que tener en cuenta que el programa ha sido compilado en java *openjdk 17.0.4 2022-07-19*.

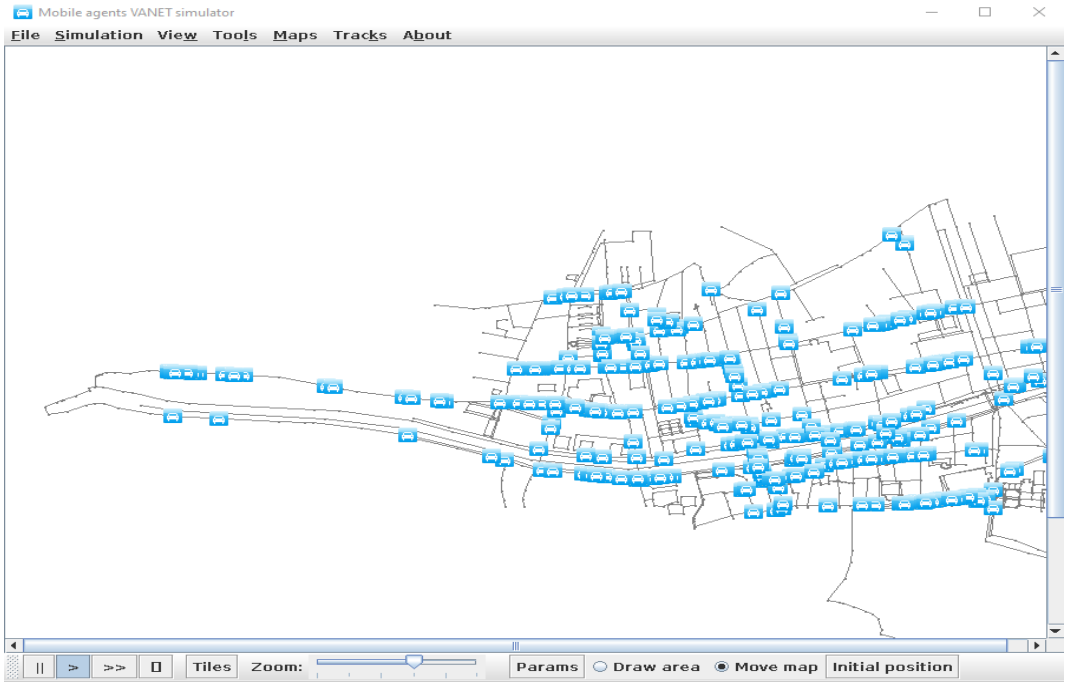


Figura 3.2: Programa MAVSIM iniciado

El propio MAVSIM trae incorporado un gestor de mapas OSM que permite descargar un mapa dadas sus coordenadas, se guarda en su carpeta *graphs* y es ejecutable al momento. Por otro lado, en vez de leer dónde está el mapa, las rutas o la configuración de la simulación, MAVSIM lee el mapa desde su carpeta y crea directamente las simulaciones con parámetros por defecto. Durante la simulación estos parámetros pueden ser modificados, incluyendo la generación de las rutas, la estrategia de salto, el número de coches o su velocidad.

[1] `https://ant.apache.org/`

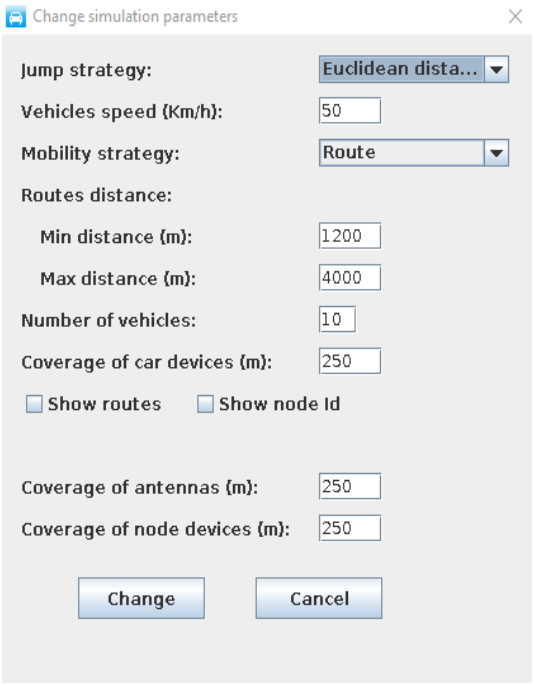


Figura 3.3: Parámetros de MAVSIM

3.2.3. Veins

La instalación del simulador Veins conlleva algún que otro paso adicional que otros simuladores ya que es un conjunto de librerías para otro programa, OMNet++. OMNet++ es un entorno de desarrollo, basado en Eclipse, que permite crear simulaciones MANET de todo tipo, desde redes ad-hoc hasta redes de comunicación para aviones, entre otros.

La instalación de OMNet++ es algo compleja: tras descargarte de su página web[2] la versión adecuada, hay que compilar el programa mediante el script *configure*. Este archivo contiene todas las utilidades de OMNet++ que se pueden desactivar o activar. Este proceso deja listo el programa para su compilación mediante *make*.

[2] `https://omnetpp.org/download/`

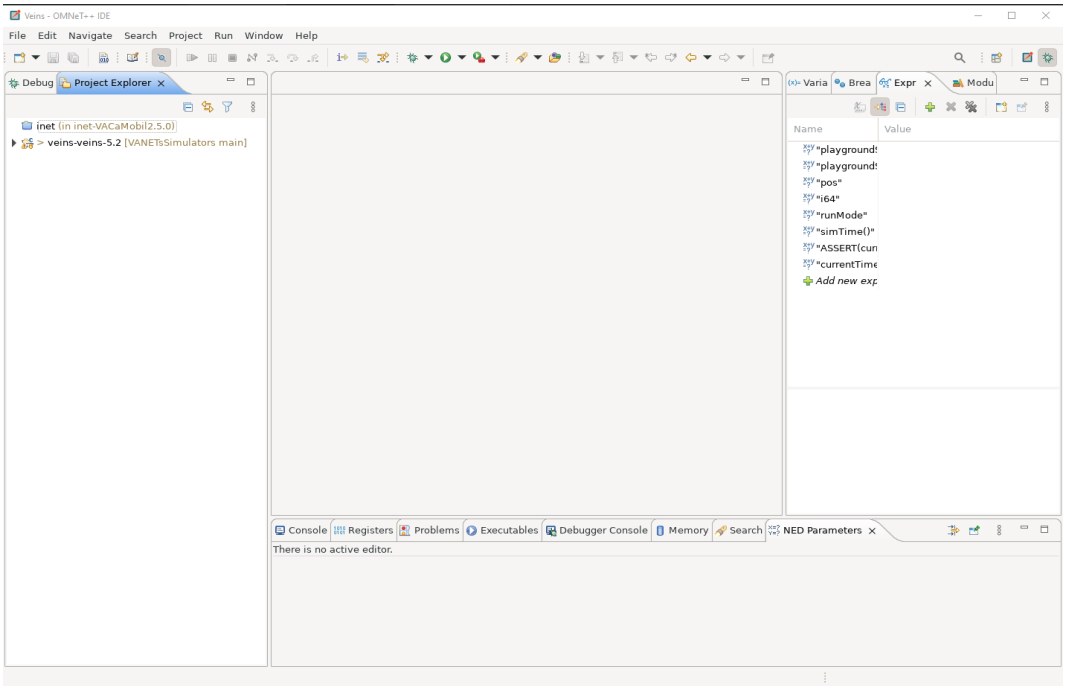


Figura 3.4: Entorno OMNet++

Con el entorno OMNet++ lanzado se exporta el proyecto de Veins con sus librerías y ejemplos además de las librerías de INET [5]. INET es un conjunto de librerías de código abierto para OMNET++. Proporciona protocolos, agentes y otros modelos para trabajar con redes de comunicación. Para la realización de este proyecto se ha utilizado uno de los ejemplos ya hechos de Veins para hacer funcionar los distintos mapas y simulación, gracias al programa en Python que se introducirá más adelante.

El formato de la simulación es el mismo que para SUMO, pero es necesario indicar en el fichero de configuración de Veins (*omnetpp.ini*) el fichero que contiene la configuración de la simulación (entre otros parámetros como el tiempo o el paso). A partir de este fichero la simulación se construye de la misma forma que SUMO. Sin embargo, es necesario establecer un puerto de escucha entre Veins y SUMO, ya que este primero utiliza al segundo para resolver los eventos durante la ejecución. En la figura 3.5 se puede observar una simulación ejecutándose en Veins.

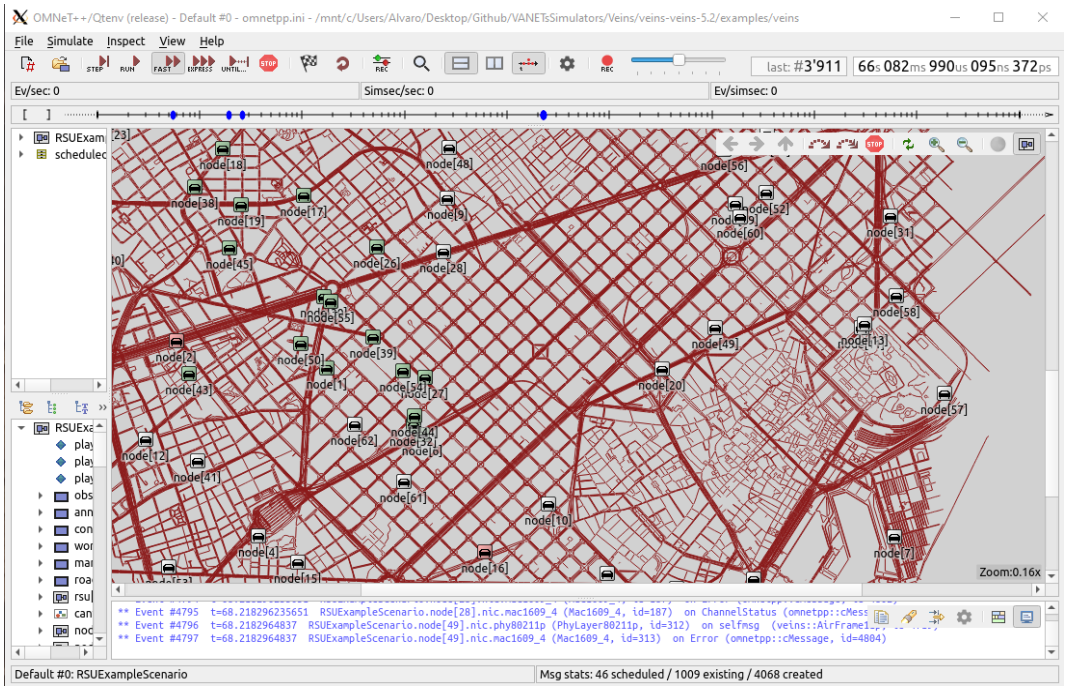


Figura 3.5: Simulación en Veins ejecutándose

3.2.4. VanetMobisim

El soporte disponible para VanetMobisim es muy pobre, la página oficial y sus descargas están tan mal cuidadas que la versión que se va a usar en el proyecto ha sido encontrada en una página en portugués tras buscar varios días. Obviando esto, el programa no necesita ninguna instalación y basta con ejecutar su fichero .jar con el siguiente comando: *java -jar VanetMobiSim-1.1.jar*.

En cuanto a la ejecución de las simulaciones, VanetMobisim funciona de forma completamente distinta y solo lee un mapa OSM a partir del cual crea la simulación. Aunque se podría decir que funciona igual que MAVSIM, no sería del todo cierto ya que MAVSIM permite modificar prácticamente todos los parámetros de la simulación tras crearla y VanetMobisim no permite modificar nada y solo acepta tiempo de simulación y velocidad de los coches.

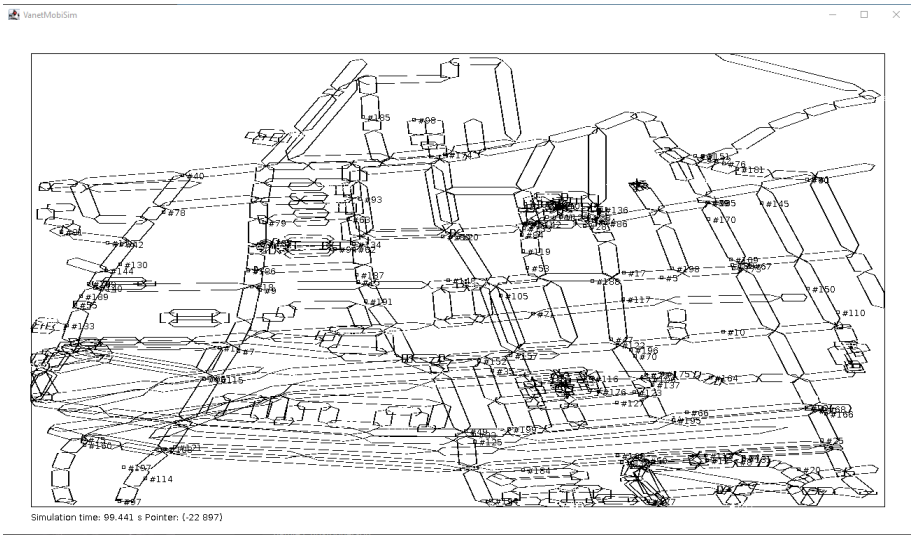


Figura 3.6: Mapa de Dublín en VanetMobisim

Otro de los problemas de este simulador es que utiliza un formato completamente distinto a la hora de crear simulaciones. Su característica más importante es que las coordenadas de los nodos se definen mediante coordenadas cartesianas. Esto implica que hay que traducir los mapas OSM de coordenadas geográficas a coordenadas cartesianas y establecer el centro, mínimo y máximo de las coordenadas nuevas. En cuanto a su estructura HTML, las simulaciones siguen un esquema de clases estilo Java. Por ejemplo, la etiqueta *eurecom.usergraph.UserGraph* indica el grafo que representa el mapa OSM, *de.uni$_s$tuttgart.informatik.canu.mobisimadd.extensions.GUI* el tamaño de la interfaz o *de.uni$_s$tuttgart.informatik.canu.mobisim.simulations.TimeSimulation* el tiempo de simulación. Este formato propio hace muy complicado adaptar los mapas de otros simuladores para que funcionen de igual forma en VanetMobisim.

A todo esto hay que añadir la escasez de ejemplos que hay. En la documentación oficial no especifican ningún ejemplo ni qué clases hay disponibles ni cuáles son necesarias para hacer funcionar una simulación. Así que de nuevo se han utilizado unos ejemplos encontrados en páginas ajenas para, a partir de lo que estos contienen, construir las simulaciones deseadas. Para ello se definen los nodos, las aristas que existen entre los nodos, las velocidades y el tamaño de la interfaz entre otros.

Figura 3.7: Formato de una simulación en VanetMobisim

3.3. Programa createNet.py

Para poder facilitar la realización de las pruebas se decidió implementar un programa en Python que permitiera semi-automatizar la generación de simulaciones para cualquier simulador y para cualquier parámetro. Este programa es capaz de, dado un mapa OSM y ciertos parámetros que se describirán más adelante, generar la simulación deseada en una sola ejecución y poder ejecutarla al momento. El programa acepta 3 modos de ejecución:

- **python3 createNet.py -[gmiv]:** Crea una simulación para el simulador introducido [gmiv] con un mapa aleatorio simple (2 aristas por nodo).
- **python3 createNet.py -osm -[gmiv] [mapaOsm]:** Crea una simulación para el simulador introducido [gmiv] a partir del mapa OSM [mapaOsm] .

El procedimiento que se sigue para ejecutar simulaciones se divide en 3 pasos, más o menos similares para los diferentes simuladores:

- **Creación del mapa de nodos y aristas:** El programa, en sus dos modos, primero tiene que generar un mapa de nodos y aristas, normalmente representado con una matriz. Esta matriz puede ser generada de forma aleatoria o directamente desde el mapa OSM (como es el caso de VanetMobisim). Para gran parte de los simuladores el propio mapa OSM original es válido.

Si el mapa original provoca problemas se utiliza una versión de este mapa con un formato ligeramente distinto (osmconvert facilita este cambio).

- **Generación de rutas y otras configuraciones:** A partir del mapa formado en el paso anterior y la herramienta randomTrip de SUMO se genera el fichero de rutas aleatorias. Durante este paso también se generan los ficheros de configuración para los 4 simuladores. Aunque no es obligatorio el uso de randomTrip, ahorra mucho tiempo que habría que invertir en generar estas rutas mediante bucles en Python.

- **Escritura de archivos y lanzamiento de simuladores:** Por último se guardan los ficheros necesarios para la ejecución de la simulación en los directorios correspondientes y se ejecuta la simulación, cuando es posible lanzar la simulación por terminal (en Veins es imposible ya que es necesario lanzar el proyecto Veins dentro del entorno OMNet++). En caso de que no se puede ejecutar por terminal el simulador, se prepara los archivos lo máximo posible para que el usuario tenga que hacer lo mínimo para comenzar la simulación.

Capítulo 4

Análisis comparativo

Este análisis se ha basado en las pruebas realizadas a lo largo del proyecto y que están detalladas en el apéndice C. Los tiempos de simulación han sido medidos en situaciones de disponibilidad de recursos similares y los recursos disponibles son 16 Gb de RAM y el procesador Intel Core i5-8600K CPU @ 3.60GHz.

En este apartado se va a analizar cada programa por separado, haciendo hincapié en su funcionamiento y facilidad de uso, rendimiento en tiempo y recursos y como de fácil es su uso a la hora de experimentar con diferentes simulaciones. Los análisis intentarán tratar de describir los diferentes elementos de los programas para luego realizar una comparativa cuantitativa y cualitativa de los datos obtenidos[6]. Sin embargo, algunos de los campos incluidos son algo difíciles de contabilizar y comparar debido a la naturaleza de los simuladores.

El análisis explorará los resultados obtenidos en las pruebas de los siguientes mapas:

- *Dublin.osm*
- *Barcelona-2.osm*
- *London.osm*

Se puede encontrar imágenes y pruebas más extensas de cada uno de los mapas en él apéndice C.

4.1. SUMO

Lo primero a destacar de SUMO es su facilidad de uso y la simpleza de su instalación, además proporciona una gran variedad de herramientas muy útiles para generar o modificar simulaciones de redes vehiculares. Es el simulador más utilizado del mercado debido a todas estas utilidades y al rendimiento y comportamiento estable. Además, también tienes a tu disposición su extensa documentación, variedad de ejemplos y excelente soporte.

4.1.1. Facilidad de uso y ejecución

En el apartado de facilidad de uso, SUMO destaca por encima del resto por su simple y, a la vez, completa interfaz, su instalación sencilla y un buen control de la simulación y su ejecución. A esto hay que incluir que SUMO incluye una versión sin interfaz para poder lanzarse por terminal. Sin embargo, la salida que muestra por terminal (figura 4.1) no muestra claramente los eventos de la simulación y hace muy difícil entender la simulación, más allá de cuando empieza o acabe o cuantos vehículos quedan activos.

```
Step #0.00 (0ms ?*RT. ?UPS, vehicles TOT 0 ACT 0 BUF 0)
Step #100.00 (0ms ?*RT. ?UPS, vehicles TOT 76 ACT 67 BUF 0)
Step #200.00 (0ms ?*RT. ?UPS, vehicles TOT 152 ACT 107 BUF 0)
Step #300.00 (0ms ?*RT. ?UPS, vehicles TOT 219 ACT 115 BUF 0)
Step #400.00 (1ms ~= 1000.00*RT, ~119000.00UPS, vehicles TOT 297 ACT 119 BUF 1)
Step #500.00 (0ms ?*RT. ?UPS, vehicles TOT 366 ACT 118 BUF 0)
Step #600.00 (0ms ?*RT. ?UPS, vehicles TOT 436 ACT 128 BUF 0)
Step #700.00 (0ms ?*RT. ?UPS, vehicles TOT 502 ACT 119 BUF 0)
Step #800.00 (1ms ~= 1000.00*RT, ~123000.00UPS, vehicles TOT 574 ACT 123 BUF 0)
Step #900.00 (0ms ?*RT. ?UPS, vehicles TOT 641 ACT 125 BUF 0)
Step #1000.00 (1ms ~= 1000.00*RT, ~126000.00UPS, vehicles TOT 712 ACT 126 BUF 0)
Step #1100.00 (0ms ?*RT. ?UPS, vehicles TOT 779 ACT 118 BUF 0)
Step #1200.00 (0ms ?*RT. ?UPS, vehicles TOT 845 ACT 111 BUF 0)
Step #1300.00 (1ms ~= 1000.00*RT, ~116000.00UPS, vehicles TOT 914 ACT 116 BUF 1)
Step #1400.00 (0ms ?*RT. ?UPS, vehicles TOT 989 ACT 133 BUF 1)
Step #1500.00 (0ms ?*RT. ?UPS, vehicles TOT 1054 ACT 124 BUF 0)
Step #1600.00 (0ms ?*RT. ?UPS, vehicles TOT 1129 ACT 124 BUF 0)
Step #1700.00 (1ms ~= 1000.00*RT, ~124000.00UPS, vehicles TOT 1206 ACT 124 BUF 0)
Step #1800.00 (1ms ~= 1000.00*RT, ~131000.00UPS, vehicles TOT 1280 ACT 131 BUF 0)
Step #1900.00 (0ms ?*RT. ?UPS, vehicles TOT 1338 ACT 123 BUF 3)
Step #2000.00 (0ms ?*RT. ?UPS, vehicles TOT 1405 ACT 120 BUF 0)
Step #2100.00 (1ms ~= 1000.00*RT, ~120000.00UPS, vehicles TOT 1471 ACT 120 BUF 0)
Step #2200.00 (1ms ~= 1000.00*RT, ~126000.00UPS, vehicles TOT 1548 ACT 126 BUF 0)
```

Figura 4.1: Salida por terminal de SUMO

En cuanto a la ejecución, para que SUMO inicie la simulación solo es necesario proporcionarle el archivo de configuración donde se especifica todo lo necesario. La simulación comienza eligiendo uno de los modos de velocidad (en el caso de SUMO solo hay un modo). Tras eso, los coches empiezan su recorrido aunque van iniciando poco a poco. La respuesta del programa es prácticamente inmediata y la representación del mapa y los coches es precisa y con gran detalle. Sin embargo, esta precisión y detalle hace que los coches sean pequeños y sea casi imposible seguir la simulación globalmente. Aunque SUMO permite ver la simulación de diferentes modos, como: *standart, faster standart, real world, rail y selection.* Algunos como *real world* hace más fácil distinguir los coches y las vías.

Figura 4.2: Vista global de la simulación en SUMO

SUMO también ofrece un panel de información donde se muestra el estado de la simulación, pero sobre todo muestra con gran detalle el estado de los coches y su progreso. Esto facilita mucho determinar cuándo ha acabado la simulación, ya que por defecto SUMO no acaba su ejecución hasta que llega al tiempo de simulación especificado, incluso cuando no quedan coches por finalizar.

4.1.2. Rendimiento

SUMO destaca en gran medida en el apartado de rendimiento respecto al resto de programas, esto es debido en gran parte al uso del almacenamiento en disco para intentar ahorrar recursos. Aunque esto podría explica la ligera diferencia en tiempos obtenidos respecto a Veins, ya que el uso del disco puede ralentizar el procesado de las iteraciones provocando un aumento en el tiempo que tarda la simulación en terminar.

Los resultados obtenidos en las pruebas reflejan que SUMO utiliza en torno al 2 % y 3 % de la RAM disponible y alrededor del 38 % del CPU con picos de 50 %. Los valores de memoria suben hasta un 11 % (el uso de CPU se mantiene similar) con mapas mucho más grande como el mapa *Barcelona-2*.

La tendencia que se puede observar en las simulaciones es que el uso de RAM aumenta de forma directa según el tiempo que tarde la simulación en terminar. Como el rendimiento en tiempo de SUMO es bastante estable, en un mapa en concreto, y sabemos que existe esta relación tiempo-memoria, podemos decir que la memoria también será estable, tal y como refleja la figura 4.3.

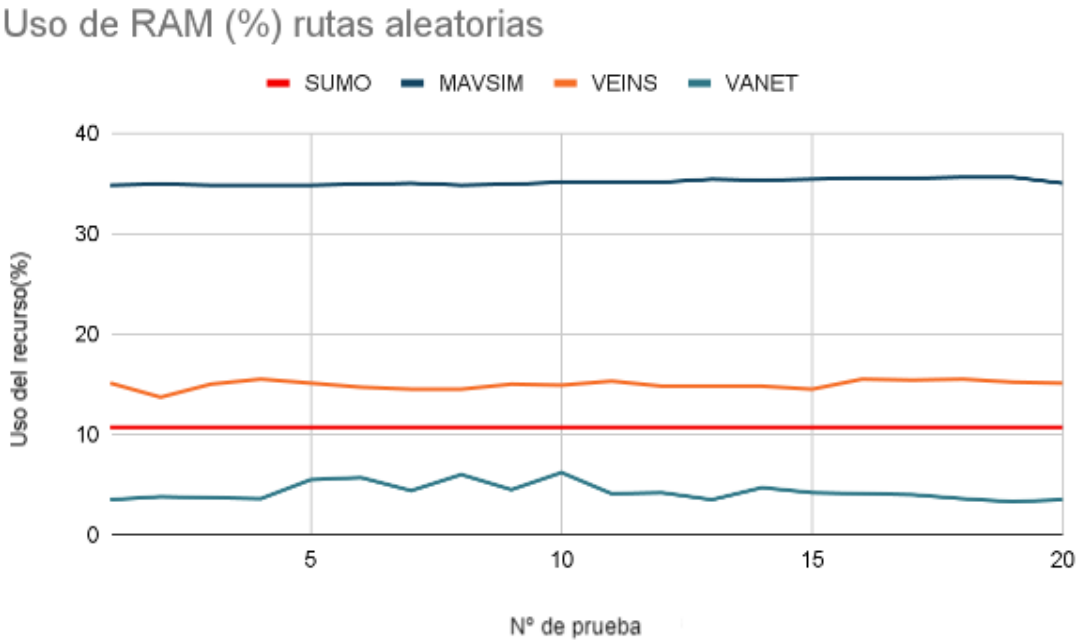


Figura 4.3: Uso de RAM *Barcelona-2*

Analizando más a fondo los tiempos obtenidos en las tablas del documento de pruebas se puede ver con más detalle cómo afecta el número de coches a SUMO. Por ejemplo, en la figura 4.4 podemos observar la tabla de los tiempos obtenidos, en las columnas está representado el número de coches y las filas el tiempo de simulación máximo configurado en la simulación *Dublin.osm*. El número de coches no parece seguir un aumento lineal, ya que en las primeras iteraciones de la tabla pasa de 9 segundos a 10, lo que sería un aumento del 11 %, pero con 750 y 1900 coches el aumento es del 90 %.

Esto indica que el tiempo escala de forma exponencial con el aumento del número de coches, lo que sumado al tamaño del mapa puede llegar a provocar problemas graves de rendimiento en SUMO. Para comprobar la magnitud del problema, se han realizado pruebas sobre el mapa *Barcelona-2.osm* ya que es un de un mayor tamaño respecto al

resto. Los tiempos reflejados en su tabla, figura 4.5, son terriblemente grandes, llegando a magnitudes de varias horas de ejecución.

	50 coches	125 coches	300 coches	750 coches	1900 coches
250 ms	6s*	6 s*	6 s*	6 s*	6 s*
625 ms	9s	10s	15s	16s*	16s*
1560 ms	9s	10s	15s	27s	41s*
3900 ms	9s	10s	15s	27s	57s
10000 ms	9s	10s	15s	27s	57s

Figura 4.4: Evolución del tiempo respecto al tiempo de simulación y nº de coches

	50 coches	125 coches	300 coches	750 coches	1900 coches
250 ms	320s *1	320 s *1	320s *1	320s *1	320s *1
625 ms	988s *1	988s *1	988s *1	988s *1	988s *1
1560 ms	1516s *2	2400s *2	2736s *2	3128s *1	3128s *1
3900 ms	1516s *2	2400s *2	2736s *2	6396s *1	6396s *1
10000 ms	1516s *2	2400s *2	2736s *2	6396s *1	13128 s *2

Figura 4.5: Evolución del tiempo mapa *Barcelona-2.osm*

Otro factor a tener en cuenta es el efecto en el rendimiento que tiene la complejidad y tamaño de cada mapa. Para el mapa *Dublin.osm*, los resultados de tiempo son los ya comentados en la figura 4.4 (se mantienen entre un mínimo de 10 segundos y un máximo de 1 min, algo bastante factible). Sin embargo, en el mapa Barcelona-2 los resultados observados son mucho peores, llegando a ejecuciones de varias horas, como ya se ha comentado. El mapa *London* refleja resultados de tiempo intermedios, con un mínimo de 77 segundos y un máximo de 28,4 minutos, por tanto, se coloca a medio camino entre los tiempos de *Dublin.osm* y *Barcelona-2.osm*.

Por último, la aleatoriedad de las rutas puede llegar a afectar el resultado de una simulacion, ya que unas rutas pueden crear más conflictos que otras entre los coches. En la figura 4.6 se encuentra representado el tiempo de una simulación variando las rutas de los coches del mapa *Barcelona-2* y en figura 4.7 los tiempos de *London*. En los tiempos de *Barcelona-2* se puede ver que los valores no varían mucho de la media, se desvían un máximo del 30 % de su valor. Pero si se observa la gráfica del mapa *London*, las rutas afectan en mayor medida a los tiempos de ejecución, aumentando hasta en un 146 % sus valores.

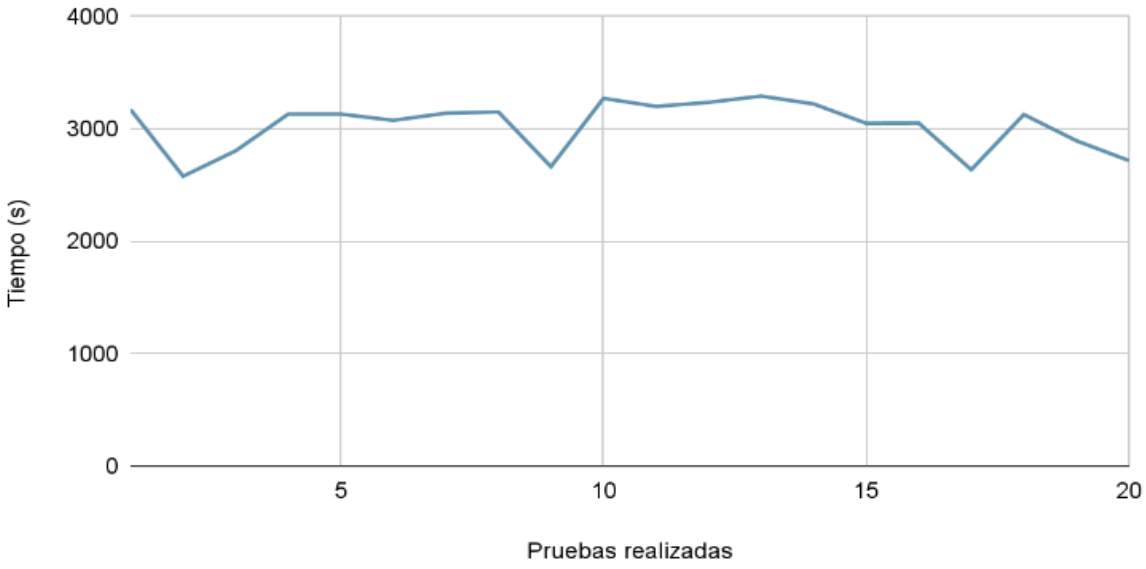


Figura 4.6: Tiempos de 20 simulaciones con rutas aleatorias en SUMO

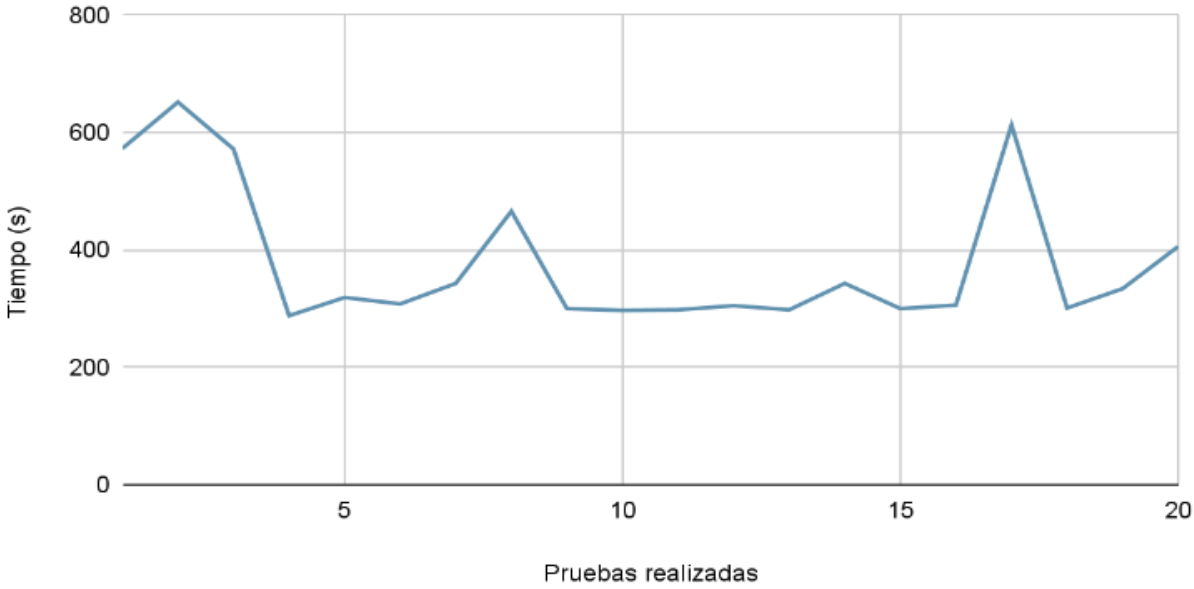


Figura 4.7: Tiempos rutas aleatorias mapa London en SUMO

4.1.3. Efectividad

La efectividad de un simulador se refiere a la facilidad con la que un simulador ejecuta diferentes simulaciones con diferentes parámetros y mapas de diferentes tamaños y formas. Esto incluye, obviamente, el rendimiento de tiempo y recursos, pero también cómo de fácil es preparar esta simulación y cómo de manejable y comprensible es esta simulación.

Teniendo todo esto en cuenta, SUMO cumple excepcionalmente bien en este campo, ya que las simulaciones se lanzan simplemente cambiando el parámetro que se le pasa por terminal e incluso mapas muy grandes como *Barcelona-2* tardan apenas unos 20 segundos en abrirse, cuando en otros simuladores tardarían varios minutos. En cuanto al manejo de la simulación no hay nada que decir, es excepcional. El manejo cumple con lo que se le pide a este tipo de programas y aun con mapas muy grandes sigue pudiendo moverse bien, otros simuladores ven su desplazamiento y *zoom* del mapa ralentizado o incluso congelado por completo.

Sin embargo, la mala visibilidad global de la simulación se hace más evidente con mapas más grandes, ya que aún es más necesario acercarse a los coches para verlos. Aunque el rendimiento ya ha sido explicado en el apartado anterior, hay que recalcar que el uso de recursos apenas aumenta entre simulaciones de distintos mapas, sobre todo cuando lo comparamos con el aumento en el tiempo de ejecución, que aumenta de forma exponencial con mapas muy grandes. Este anormal aumento de tiempo es provocado por el tamaño del mapa, que provoca una disminución en el rendimiento del programa ralentizándolo drásticamente.

4.2. Veins

Veins no es un simulador de redes vehiculares, como sí lo son el resto de programas. En realidad, Veins es un conjunto de librerías que permiten crear estas redes vehiculares dentro de OMNet++. Esto quiere decir que los problemas y rendimiento que este entorno contiene afectan también a Veins, además de los que añade estas librerías.

4.2.1. Facilidad de uso y ejecución

La facilidad de uso es un apartado en el que Veins destaca, pero en el mal sentido. Su instalación es un proceso relativamente complejo y lleno de posibles fallos, ya que para hacer funcionar una simulación hay que, primero, descargar e instalar OMNet++ y después importar el proyecto con las librerías de Veins.

Además, este proyecto necesita otras librerías (INET) para funcionar y todo tiene que compilarse entre sí sin fallos y, por si fuera poco, también es necesario una conexión con SUMO mediante un puerto de escucha. Sin embargo, el soporte que proporciona su web oficial [1] para la instalación y su uso es bastante detallado.

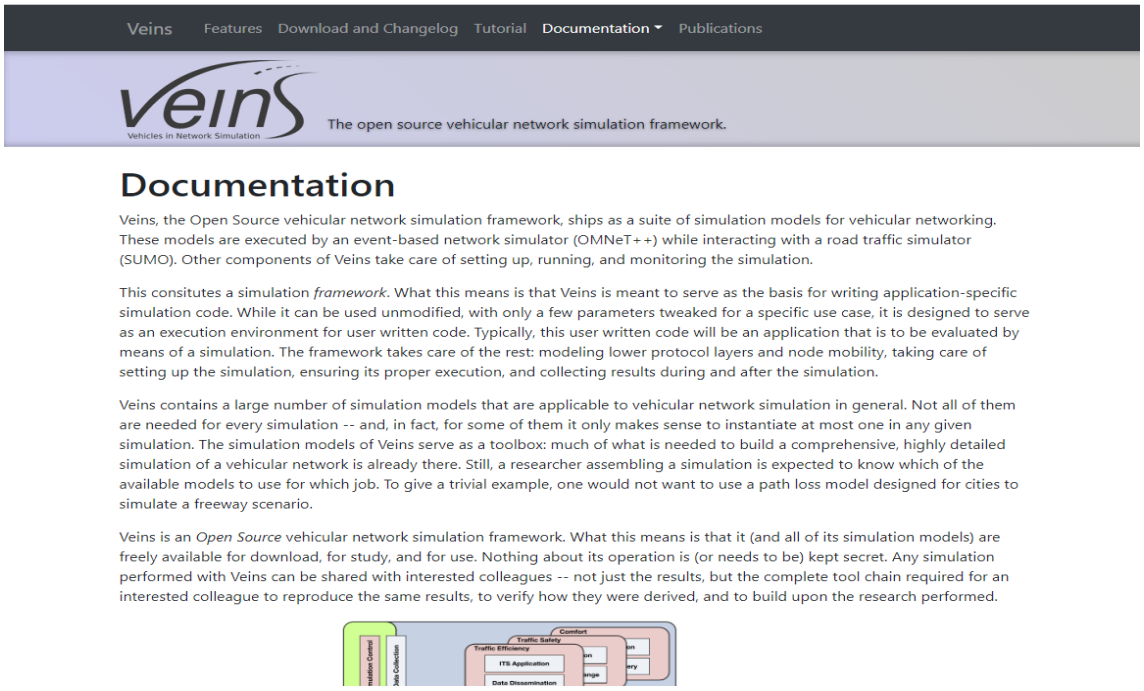


Figura 4.8: Página de documentación de Veins

[1] https://veins.car2x.org/documentation/

Al contrario que SUMO, Veins pone a disposición del usuario hasta 4 modos distintos de ejecutar una simulación, pero hay que destacar que ninguno de los 4 modos permite seguir la simulación de una forma rápida y comprensible como lo permite SUMO. El modo de ejecución normal es demasiado detallista y lento, describiendo cada uno de los eventos, y los modos rápidos muestran a saltos las rutas de los coches, por lo que es imposible ver cómo interaccionan entre sí.

4.2.2. Rendimiento

El rendimiento de Veins es algo complejo de medir, ya que depende de varios componentes para funcionar correctamente. Uno de estos procesos está representado en la figura 4.3, ya vista y es quien ejecuta la simulación como tal.

Como se puede observar en la gráfica, el uso de recursos es similar al de SUMO pero algo menos estable y un poco más elevado. Aun así, no hay que olvidar que a esto hay que sumar el coste de ejecutar SUMO internamente para resolver los eventos. Este coste no es constante, ya que se mantiene bajo mínimos excepto en los momentos en los que se comunican ambos programas. el máximo que alcanza en ese instante está entre el 7,9 % y 11,2 % de RAM y 3 % y 7,7 % de CPU. Si tenemos en cuenta el sumatorio total de los procesos que intervienen en Veins resulta evidente que el uso de recursos es mucho mayor que SUMO, sobre todo el uso de RAM.

Por otro lado, las tablas de tiempos, como la de la figura 4.9 reflejan un comportamiento exponencial, pero en este caso, la gran mayoría de ejecuciones terminan porque los coches finalizan sus rutas antes del tiempo de simulación. Pero eso no tiene por qué significar que obtiene mejor rendimiento en tiempo que SUMO, si comparamos los tiempos obtenidos, son parecidos o al menos de la misma magnitud.

	50 coches	125 coches	300 coches	750 coches	1900 coches
250 s	44s	92s	176s	176s*	176s*
625 s	44s	92s	176s	1228s	1228s*
1560 s	44s	92s	176s	1228s	6246s*
3900 s	44s	92s	176s	1228s	12.821s
10000 s	44s	92s	176s	1228s	12.821s

Figura 4.9: Evolución del tiempo con el mapa *Barcelona-2.osm*

En cuanto a la comparación con otros mapas, el rendimiento que se muestra en *Dublin.osm* o *London.osm* es el esperado. La mayoría de simulaciones terminan sus rutas sin problemas y los valores de tiempo obtenidos siguen aumentando a un ritmo exponencial. Esto se puede observar en la figura 4.10 donde en las primeras columnas el tiempo pasa de 3 a 8 (167 %) y más adelante pasa de 8 a 32 (300 %).

	50 coches	125 coches	300 coches	750 coches	1900 coches
250 s	1s	2s	5s*	5s*	5s*
625 s	1s	3s	7s	24s*	24s*
1560 s	1s	3s	8s	32s	83s*
3900 s	1s	3s	8s	32s	102s*
10000 s	1s	3s	8s	32s	126s

Figura 4.10: Evolución del tiempo con el mapa *Dublín*

Sin embargo, todos los mapas terminan más o menos el mismo porcentaje de ejecuciones por tiempo de simulación, aunque luego el tiempo de ejecución obtenido sea completamente distinto. Esto aunque en principio no parezca tener explicación, está relacionado con cómo trata Veins los eventos de las simulaciones.

Veins ejecuta las simulaciones mediante listas de eventos y ejecuta eventos cada X milisegundos de tiempo de simulación. Lo que no tiene en cuenta es cuánto dura cada uno de esos eventos. No es lo mismo ejecutar 300.000 eventos que duran 1-2 segundos reales que el mismo número de eventos y que duren 5 segundos cada uno. El tiempo de simulación será el mismo, pero el de ejecución o real no. Cuanto mayor sea el coste de cada evento por separado mayor será la deformación entre el tiempo de simulación y el tiempo real.

En cuanto al efecto que tiene la generación de rutas aleatorias, como muestra la figura 4.11, las rutas no parecen afectar mucho a los tiempos obtenidos. El máximo observado es de 100 s y el mínimo de 77 s la diferencia no llega al 25 %, no es una diferencia muy remarcable si lo comparamos con datos anteriores.

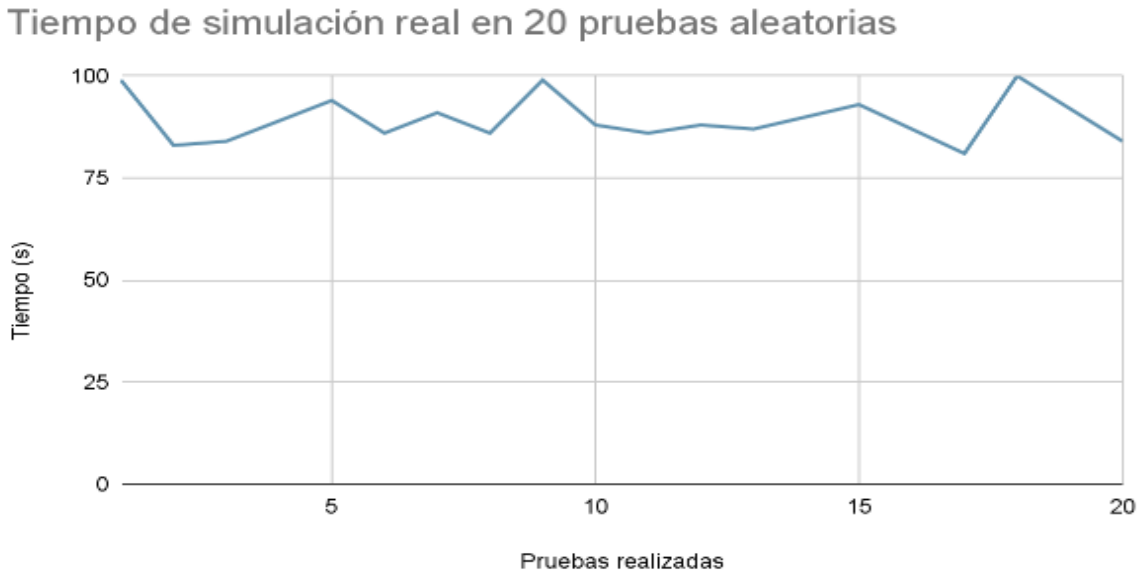


Figura 4.11: Tiempos con rutas aleatorias London

4.2.3. Efectividad

En el caso de Veins, preparar el simulador para ejecutar simulaciones pasa por, como ya se ha comentado, configurar e iniciar OMNet++, preparar un puerto de escucha de SUMO e iniciar Veins como tal. Una vez iniciado Veins, cambiar de una simulación a otra es muy sencillo; solo hay que modificar el archivo *omnetpp.ini* e introducir los archivos de la nueva simulación en la carpeta del proyecto. Veins se inicia, pero no carga la simulación hasta que se elige un modo de ejecución. Esta carga no suele llevar más de 10 segundos, excepto en mapas muy grandes como *Barcelona-2.osm*.

Sin embargo, la ejecución en sí de Veins no ofrece todo lo que se busca en un simulador de redes vehiculares. Aunque sí es verdad que el programa permite observar cada uno de los eventos de la simulación e incluso pausar uno a uno y ver las comunicaciones entre los coches, no permite un modo de ejecución con el que se pueda observar el comportamiento global de la simulación. O el usuario elige un modo muy lento y tardaría demasiado tiempo en ver la ejecución entera o elige un modo rápido que muestra los eventos muy de vez en cuando, haciendo parecer que los coches dan saltos por el mapa y no es posible seguir el recorrido de los coches.

4.3. MAVSIM

MAVSIM es un simulador creado en la Universidad de Zaragoza. El equipo que lo diseño optó por incorporar un gestor de mapas propio. Con este gestor puedes descargar mapas dadas las coordenadas y más tarde abrir el mapa deseado de la colección descargada. Aunque este gestor propio sí aporta mucha comodidad, otras particularidades como tratar el tiempo de simulación mediante número de iteraciones o modificar sus parámetros durante la ejecución pueden resultar en algunos contratiempos. Estos contratiempos están relacionados con la importación de mapas y el tratamiento de algunos de los parámetros (serán explicados con más profundidad a continuación).

4.3.1. Facilidad de uso y ejecución

Por todo lo comentado en la introducción anterior, MAVSIM tiene su modo de ejecutar simulaciones que puede diferir de los otros simuladores. Aunque ofrece algunas utilidades únicas pueden venir acompañadas de algunos contratiempos, como: no poder tratar el rendimiento en tiempo con la misma metodología que otros programas o que los parámetros de la simulación estén limitados en parte (por ejemplo la generación de rutas es interna y puedes elegir la estrategia a utilizar, pero no especificar las rutas como tal).

La ejecución de simulaciones está bastante simplificada, aunque si quieres usar un mapa externo hay que incluirlo en el fichero de mapas descargados y crear su directorio. Si usas un mapa del gestor de mapas de MAVSIM todo el proceso se puede realizar sin cerrar el programa ni modificar ningún archivo. En cuanto a la visibilidad de la ejecución, los vehículos y las vías que recorren se identifican claramente y la simulación es comprensible y fácil de seguir. Sin embargo, los detalles que se dan de cada coche o evento son muy reducidos, haciendo difícil entender qué está ocurriendo por detrás.

En la figura 4.12 se puede ver el gestor de ficheros mencionado. En esta pestaña en concreto se especifican las coordenadas geográficas del mapa que se quiere descargar, su nombre y otras opciones. MAVSIM utiliza la API Overpass de *OpenStreetMap* para descargar el mapa designado. El formato de este mapa es muy similar a los obtenidos directamente desde la página web de *OpenStreetMap*, pero con algunos pequeños detalles. Debido a algunos problemas causados por la estructura del tabulado de estos mapas descargados, se ha tratado como un formato distinto a los utilizados en el resto de programas.

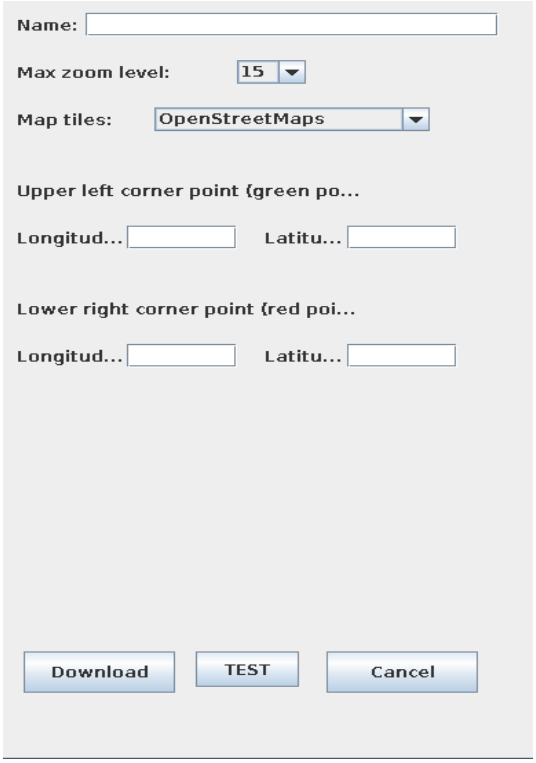


Figura 4.12: Gestor de Mapas de MAVSIM

Como ya se ha comentado al principio de este apartado, MAVSIM tiene un método algo distinto de ejecutar simulaciones. Este método restringe en parte los parámetros que se pueden configurar de las simulaciones. En la **??** se puede observar el panel de parámetros de simulación de MAVSIM. En este panel se pueden configurar diferentes parámetros como: estrategia de salto, velocidad de los vehículos, estrategia de movimiento de los coches (rutas, random, etc.) y otros muchos. Sin embargo, algunos parámetros como el número de coches o la velocidad máxima de los vehículos tienen sus valores limitados. En el caso de la velocidad máxima de los coches está limitado a 9999 km/h que es imposible que limite la simulación. Pero por parte del número de vehículos, su valor está limitado a 999 y este valor puede quedarse pequeño para mapas muy grandes como *Barcelona-2*

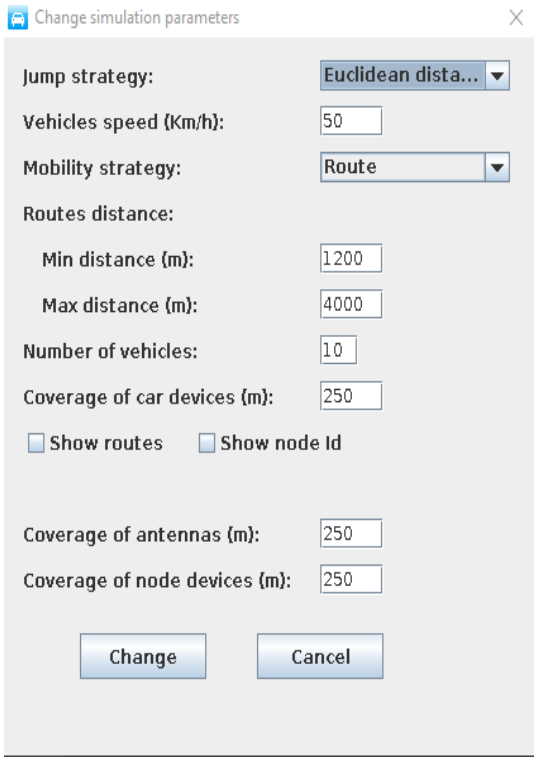


Figura 4.13: Parámetros de una simulación en MAVSIM

4.3.2. Rendimiento

El análisis del rendimiento de MAVSIM tiene el problema fundamental de que no puede ser comparado directamente con el rendimiento mostrado por otros programas ya que MAVSIM permite limitar con precisión la duración de la ejecución mediante el número de iteraciones. En cambio, el resto de programas limitan sus ejecuciones por un tiempo de simulación máximo. Para que MAVSIM pudiera ser comparada en rendimiento de tiempo con el resto habría que diseñar pruebas adicionales en las que se ponga a prueba cuanto porcentaje de simulación es capaz de recorrer cada programa en un tiempo limitado. Y de esta forma saber qué simulador muestra más simulación con el menor tiempo disponible.

En la figura 4.14 se puede observar el uso de CPU para el mapa *Dublin.osm*. Estos valores varían entre el 10 % y el 23 % y son los valores más bajos de uso de CPU para este mapa. Por parte de la RAM, como su uso depende del tamaño del mapa y del tiempo de simulación los valores que se obtienen siempre para un mismo mapa entre diferentes ejecuciones son siempre iguales. Debido a que siempre va a ejecutar el mismo número de iteraciones, una ejecución siempre resulta en la misma representación de la simulación, da igual los parámetros que se introduzcan. La ejecución viene limitada directamente por el número de iteraciones. Este comportamiento completamente estable se puede observar en figura 4.3 de apartados anteriores.

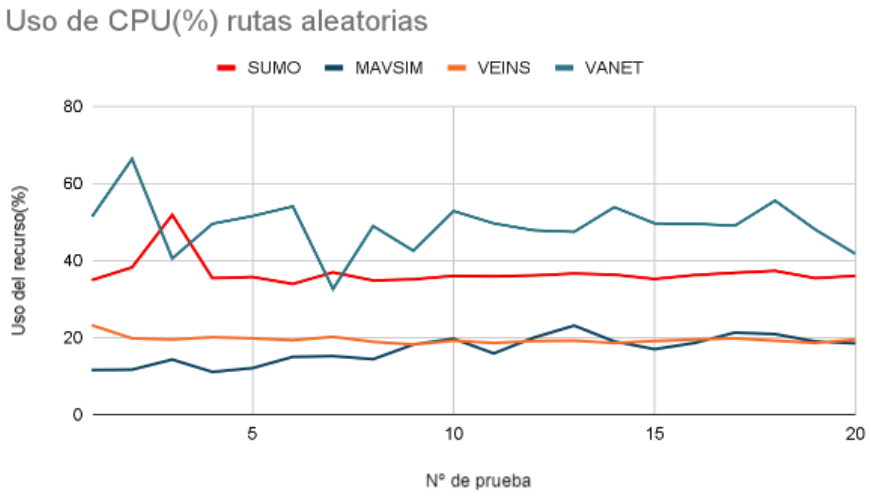


Figura 4.14: Uso de CPU con el mapa *Dublin.osm*

Como ya se ha comentado el rendimiento en tiempo de MAVSIM es algo difícil de analizar ya que las simulaciones tienen el mismo tiempo de ejecución debido al tratamiento del tiempo mediante iteraciones. Como se observa en **??** el tiempo es constante, da igual el número de coches, sin embargo, se puede observar que no hay datos para parte de la gráfica. Esta falta de datos está provocada por el hecho de que MAVSIM no permite establecer más de 999 coches en una simulación, de esta forma ha sido imposible completar las pruebas, como con el resto de simuladores.

Otro factor a tener en cuenta es el efecto en el rendimiento que tiene la complejidad y tamaño de cada mapa. Para el mapa *Dublin.osm* el tiempo observado es de 7 segundos, en cambio, para los mapas *Barcelona-2.osm* y *London.osm* es de 16 segundos en ambos. Esos 9 segundos de diferencia es lo que tarda en cargar el mapa de Dublin y el resto, ya que este mapa es muy pequeño en comparación. El tiempo que la simulación está ejecutándose es el mismo.

En el caso de las rutas aleatorias, el comportamiento de MAVSIM es el mismo. La ejecución de la simulación está limitado por el número de iteraciones. Por tanto, da igual como de complejas sean las rutas, ya que el tiempo que tarda la simulación en finalizar será el mismo.

4.3.3. Efectividad

Como ya se ha comentado, MAVSIM tiene una forma particular de ejecutar simulaciones haciendo que sea fácil cambiar de mapa o permitir cambiar los parámetros de simulación durante la ejecución. Sin embargo, si quieres añadir un mapa externo, por ejemplo si quieres ejecutar el mismo mapa en los 4 simuladores, es necesario crear y editar archivos y directorios. Esto hace complicado replicar los mapas de otros simuladores en MAVSIM ya que no es un mapa de su gestor de mapas.

Lo más molesto a la hora de trabajar con MAVSIM es que utiliza un formato propio para tratar el tiempo de ejecución mediante número de iteraciones. Al ser tan distinto al del resto de simuladores hace difícil evaluar el rendimiento del programa de la misma forma que SUMO, por ejemplo. No es un problema que afecte al funcionamiento como tal del simulador, y si lo tratamos de forma aislada no es ningún problema, pero se hace muy molesto intentar evaluar los resultados cuando el tiempo es tratado de forma completamente distinta.

En cuanto a la visibilidad de la simulación, el mapa y los coches se pueden seguir perfectamente, tanto en el modo normal como en el rápido. Pero en mapas muy grandes, como es el caso de *Barcelona-2.osm* la simulación en sus dos modos se ve altamente ralentizada y los coches no siguen sus rutas de forma fluida, haciendo que entender quÉ está ocurriendo a nivel global sea complicado. Aun así, el comportamiento de MAVSIM, ante simulaciones con mapas tan grandes como es este, permite entender mejor la ejecución que otros programas.

4.4. VanetMobisim

VanetMobisim es, con mucha diferencia, el simulador que más problemas ha causado durante la realización del proyecto. Desde problemas técnicos, falta de documentación, problemas con su formato, problemas con ejecuciones de simulaciones, problemas provocados por la gran ralentización que sufre el programa frente a mapas grandes, entre otros. Todo esto, sumado a que encontrar ejemplos funcionales del programa ha sido casi imposible, ha hecho trabajar con este simulador una experiencia que no recomendaría a nadie.

4.4.1. Facilidad de uso y ejecución

Como se ha descrito en la anterior introducción, trabajar con VanetMobisim no ha sido fácil. Para conseguir ejecutar una simulación es necesario pasar por una serie de pasos: traducir el mapa *OpenStreetMap* al formato que utiliza el programa, entender o más bien encontrar cómo funciona el sistema de clases que montan la simulación y, por último, que no se cierre sin avisar por un error provocado por un bloqueo.

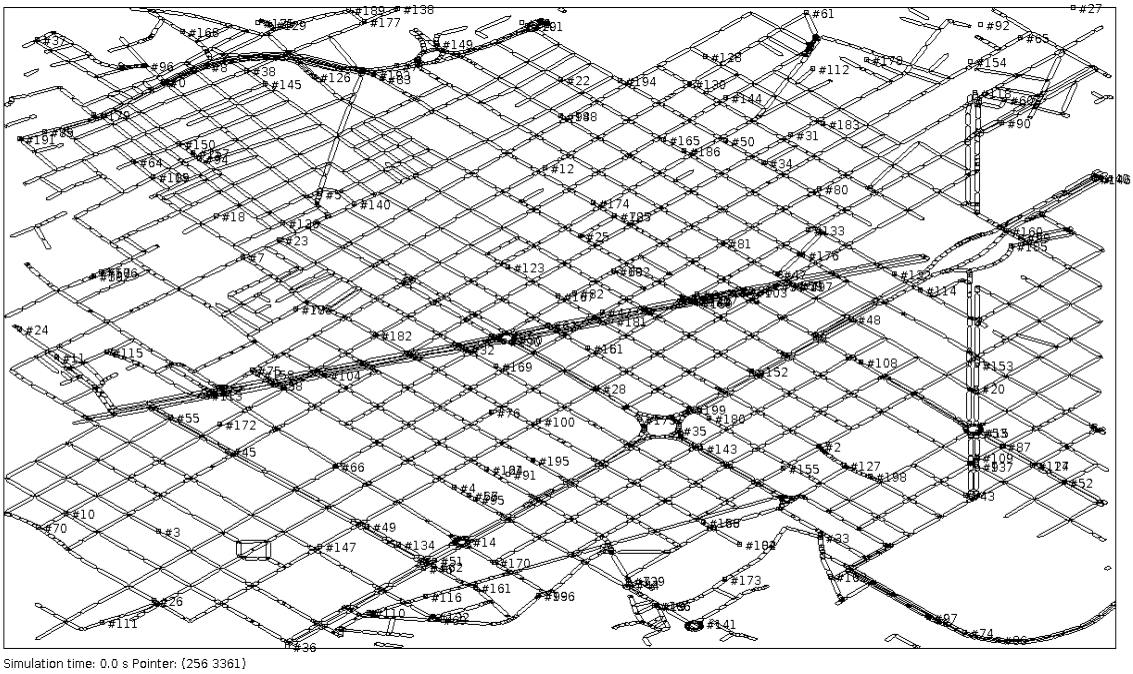


Figura 4.15: Simulación ejecutándose en VanetMobisim

La simulación es lanzada directamente desde la terminal y su ejecución comienza al momento. Una de las características más presentes de VanetMobisim es que, en la gran mayoría de ejecuciones, el tiempo que tarda el programa en leer y preparar el mapa y los coches es mayor que lo que dura la propia ejecución (si la ejecución duraba 3 minutos, 2 minutos eran de lectura y preparación de la simulación).

Esta ejecución puede durar 2 o 3 minutos o 2 horas, depende del tamaño del mapa, porque el número de coches o rutas no es posible definirlas; además tampoco deja claro cómo elige estos coches o cómo genera las rutas. Uno de los factores más importantes en este tipo de programas es cómo resuelves los conflictos entre los coches, sobre todo, cómo se resuelven los bloqueos, ya que una mala gestión de los bloqueos puede llevar a una simulación fallida o muy distorsionada. VanetMobisim no trata los bloqueos de ninguna manera: cualquier clase de comportamiento que no sea el normal acaba con la simulación cerrándose sin previo aviso y con un mensaje autogenerado por *Java*.

Por último, quedaría destacar el horrible rendimiento que tienen las ejecuciones de mapas muy grandes. En la anterior imagen, figura 4.15 se puede observar una de estas ejecuciones; en este caso la imagen se mantiene inmóvil hasta que termina de cargar, en aproximadamente 5 minutos, pero es que aun después de estar tanto tiempo cargando, la simulación realiza algún movimiento cada varios segundos. Funciona tan mal que en vez de describir su rendimiento en FPS (*frames per second*) habría que describirlo en SPF (*seconds per frame*).

4.4.2. Rendimiento

Como se ha dejado entrever en el apartado anterior, el rendimiento de VanetMobisim es muy malo. Los valores que se obtienen son tan distantes que para poder ser comparados con resultados de otros programas se ha reducido los tiempos de simulación utilizados en un 90 %. Es decir, para una simulación que se le dice que acabe su ejecución a los 10000 ms en VanetMobisim se le dice que acabe a los 1000 ms.

Teniendo todo lo anterior en cuenta, la tendencia observada en las gráficas de uso de recursos reflejan que VanetMobisim es, con diferencia, el simulador que menos RAM utiliza. Esto es obvio debido a que los tiempos de simulación son 10 veces menores y la memoria y el tiempo están directamente relacionados. Sin embargo, si miramos la gráfica de uso de CPU podemos observar como VanetMobisim no es para nada estable, es más, es casi imposible predecir qué uso de CPU va a tener.

En la figura 4.16 se puede observar este comportamiento completamente irregular de la CPU: sus valores van desde el 27 % hasta el 50 % y saltan de mínimos a máximos en cualquier momento. Esto es debido a que VanetMobisim genera sus propias rutas aleatorias en cada ejecución (no permite introducir rutas como parametro de entrada) y los otros simuladores no están utilizando rutas aleatorias, por lo que cada ejecución usa más o menos los mismos recursos. Además, VanetMobisim es increíblemente susceptible a los efectos de rutas aleatorias, ya que aun con rutas aleatorias no es normal que haya una diferencia de más del 20 % de CPU, sobre todo cuando la RAM se mantiene bastante estable en comparación.

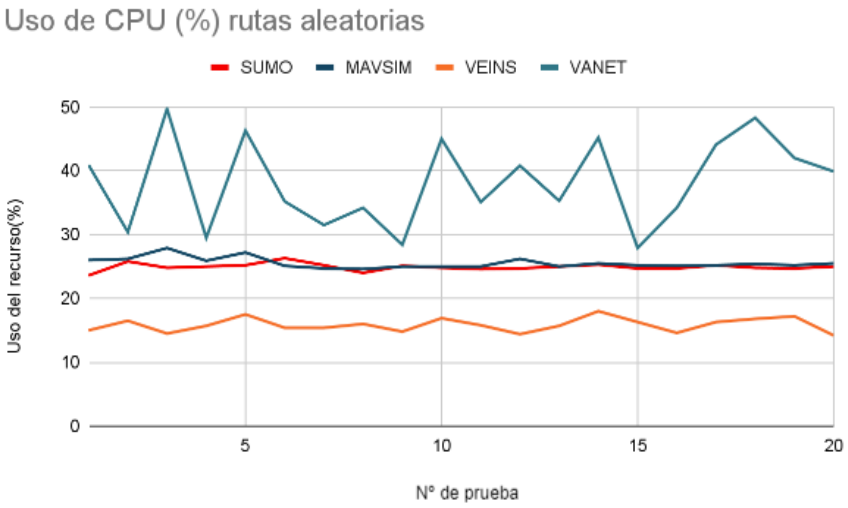


Figura 4.16: Uso de CPU con el mapa *London.osm*

Como VanetMobisim no permite establecer el número de coches ni tampoco deja claro cuantos utiliza, se ha sustituido la tabla de tiempos por una gráfica de simulaciones finalizadas por un error grave no controlado (bloqueos en la mayoría de casos). En esta gráfica,figura 4.17, se encuentran representados el resultado de 20 pruebas aleatorias y están divididas en pruebas fallidas y pruebas correctas. Las pruebas fallidas son provocadas por todo tipo de fallos: mapas que VanetMobisim dice que le faltan nodos/aristas (no es verdad), bloqueos que VanetMobisim no esta programado para solucionar y otros fallos para los que el programa devuelve una salida tan genérica que es imposible saber el motivo.

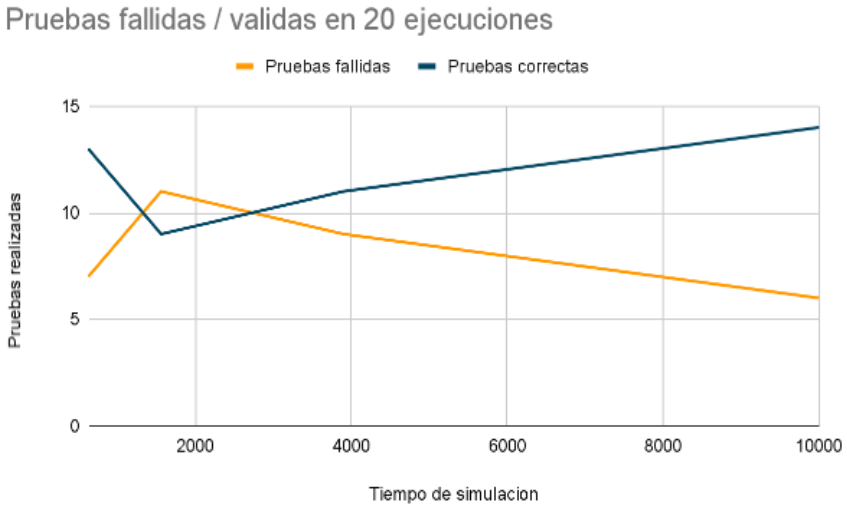


Figura 4.17: Pruebas fallidas/correctas con el mapa *Barcelona-2.osm*

Como se puede observar, el porcentaje de pruebas fallidas disminuye y aumenta sin mucha relación con el tiempo de simulación; esto se debe a que los fallos que devuelve este mapa están relacionados con la creación del mapa y sus rutas y, por tanto, no es afectado por el tiempo.

Sin embargo, si comparamos esta gráfica de pruebas fallidas/correctas con la de otros mapas obtenemos diferentes comportamientos. La gráfica de *Dublin.osm* muestra un comportamiento más predecible, aumentando los fallos conforme aumenta el tiempo de simulación (más tiempo, mayor probabilidad de bloqueo) y en el caso de *London.osm* no muestra ningún fallo en absoluto

Por último, la gráfica de tiempos con rutas aleatorias de la figura 4.18 muestra unos valores relativamente irregulares, con un máximo de 400 segundos y un mínimo de 200. Estos valores suponen un máximo del 50 % de diferencia, que es bastante notable, sobre todo cuando estamos hablando de minutos de tiempo de ejecución. Si lo comparamos con la ejecución con otros mapas como *Dublin.osm*, cuyas rutas no parecen afectar en lo más mínimo a los tiempos, o *London.osm*, donde las rutas parecen afectar aún más al tiempo, llegamos a la conclusión de que cuanto más grande es un mapa más posibilidades hay de que la generación de rutas afecte mucho a una simulación. En el caso de *Dublin.osm* la simulación es tan corta que las rutas no tienen tiempo de afectar a la ejecución.

Figura 4.18: Tiempos con rutas aleatorias *Barcelona-2.osm*

4.4.3. Efectividad

Crear el archivo de simulación con los parámetros deseados y lanzar VanetMobisim con este archivo es un paso rápido entre simulación y simulación. Pero, por otra parte, muchas de estas simulaciones fallan en su ejecución por cualquier tipo de bloqueo o fallo porque el programa no trata ningún tipo de fallo. Ya sea por un comportamiento de la simulación o por el formato del mapa, el error que se recibe por terminal es un error genérico y muchas veces casi imposible deducir porque ha ocurrido.

La peor parte a la hora de importar mapas es el formato. VanetMobisim utiliza un formato propio y además utiliza coordenadas cartesianas, a diferencia del resto de simuladores que utilizan coordenadas geográficas. Para poder traducir estos mapas al formato de VanetMobisim ha sido necesario crear un sistema de coordenadas cartesianas propio y establecer el punto de origen como el nodo de latitud y longitud mínima y definir los diferentes nodos como la distancia en el eje X y eje Y hasta este nodo mínimo.

En cuanto a cómo se adapta a cada simulación, su peor característica es el tiempo de preparación de la simulación. Este tiempo en el que se prepara la simulación para ser ejecutada puede llegar a ser de varios minutos en mapas grandes. Pero ,lo que es verdaderamente preocupante es que, este tiempo de preparación es relativamente grande cuanto menos dura la simulación. Por ejemplo para el mapa *London.osm* los tiempos obtenidos se encuentran entre los 3 minutos y 3 minutos y medio, sin embargo, el tiempo de preparación es de más de 2 minutos. Esto significa que de media el tiempo de preparación supone un 57 % del tiempo total de ejecución de la simulación Para *Dublin.osm* este tiempo de preparación es de unos 10 segundos (menos del 20 % del tiempo de ejecución) y para *Barcelona-2* es de 5 minutos (el tiempo total es de más de casi 15 minutos, 30 %).

Personalmente, creo que este simulador ganaría mucho en cuanto a su efectividad a la hora de trabajar con el sí tuviera una interfaz y la simulación no se lanzara inmediatamente cuando se abre el programa. También es extraño que se tome un tiempo para cargar el programa (o la simulación porque no muestra ninguna salida de lo que hace) y luego al abrirse el programa se toma aún más tiempo para preparar la simulación. Podría hacer todo este proceso durante el arranque del programa y luego mediante interfaz permitir iniciar o parar la ejecución.

4.5. Comparativa

4.5.1. Gráfica comparativa de mapas aleatorios

Decidir que un simulador ejecuta mejor una simulación que otro es algo complicado y que puede abarcar muchos matices. En un simulador el rendimiento es importante, un programa que utilice más recursos o que le tome más tiempo hacer algo que otros siempre será peor. Pero hay que tener en cuenta que la parte visual de un simulador también es importante ya que afecta a como se representa la simulación y como la percibe el usuario. Pero tampoco hay que olvidar la facilidad con la que el usuario descarga el simulador, lo instala y configura, lo inicia y genera cada simulación.

En la figura 4.19 se encuentran representadas 50 ejecuciones con mapas aleatorios para los 4 simuladores. Estos mapas son mapas simples con 2 aristas por nodo y de 10 a 100 nodos. Esta representación nos permite ver como reacciona cada simulador a los diferentes mapas, además para cada simulador se ha intentado utilizar los mismos parametros. Lo más destacable de la gráfica es los tiempos obtenidos de MAVSIM.

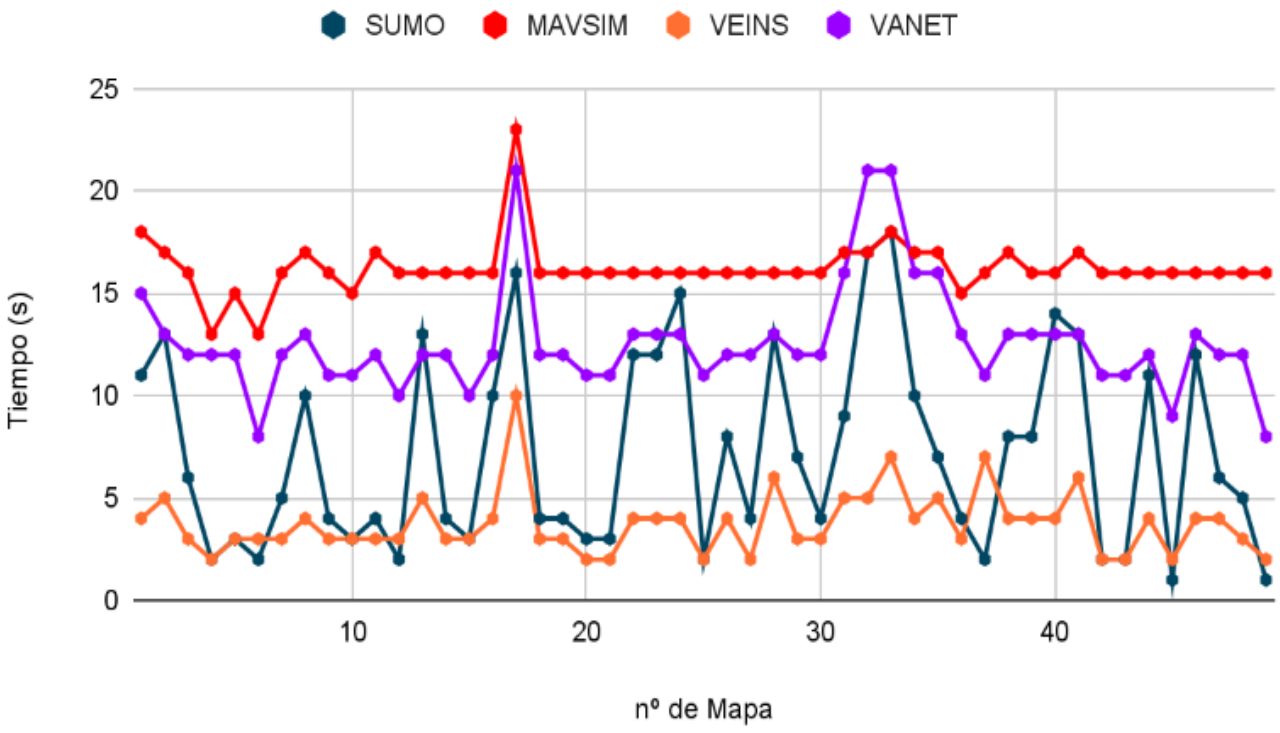


Figura 4.19: Tiempos con rutas aleatorias con el mapa *Barcelona-2.osm*

Como ya se ha explicado, los tiempos de MAVSIM son siempre los mismos porque están limitados por el número de iteraciones y el resto de simuladores se ven limitados por un tiempo de simulación. Por ello los tiempos de MAVSIM son siempre iguales, salvo diferencias de décimas de segundo.

Sin embargo, se pueden observar variaciones en sus valores. En los mapas 1,4,6 y 17 se puede ver como en algunos casos este valor llega hasta 23 segundos. Sí nos fijamos en la composición de estos mapas (incluidos en el apéndice B), podemos ver el que mapa 1 está cerca del máximo de 100 nodos y que el mapa 4 y 6 tienen muy pocos nodos. Esto parece mostrar que, aunque MAVSIM se bloque en un máximo de iteraciones, una saturación de nodos o la falta de esta puede provocar cambios en sus tiempos de ejecución. Esto se ve acentuado en el mapa 17, donde el tiempo se dispara hasta los 23 segundos.

Si observamos otros simuladores, sus valores también se disparan en este mapa. SUMO sorprendentemente se mantiene dentro de su comportamiento, que es muy reactivo como se observa por toda la grafica. En el caso de VanetMobisim se puede observar un aumento en el tiempo de ejecución muy grande, como ya ocurre en los mapas 32 y 33. Veins por su parte también reacciona aumentando de forma similar, pero en menor medida debido al modo de ejecución elegida para las pruebas. De esto podemos confirmar que SUMO es el simulador que mejor se adapta al tamaño del mapa, que VanetMobisim y Veins aunque si muestran algo de reacción, el tiempo que da a cada mapa no es suficiente y que MAVSIM puede llegar a ralentizarse tanto que se aleja por completo de las interacciones que tiene configurado.

4.5.2. Uso de los recursos

En el apartado de cada simulador se ha descrito su rendimiento, tanto en tiempo como en el uso de recursos. De entre todos los simuladores quien consume más RAM promedio es MAVSIM, seguida por Veins cuyo total, incluido el coste de usar SUMO, es alrededor del 25 %. El simulador que menos RAM utiliza es VanetMobisim aunque esto repercute en su uso de CPU como veremos a continuación.
El uso de CPU refleja un comportamiento contrario al observado en la figura 4.20. En esta gráfica (figura 4.22) MAVSIM muestra el menor uso de CPU y VanetMobisim el mayor uso, además de ser el más irregular. Por su parte SUMO y Veins se mantienen en el medio de la gráfica, con Veins siendo el menor de los dos. Invirtiendo de esta forma lo mostrado en el uso de RAM. Los comportamientos del uso del CPU son algo más irregulares, algo normal ya que el uso de CPU indica el coste computacional. Además, este puede variar fácilmente según las interacciones de los coches al recorrer sus rutas.

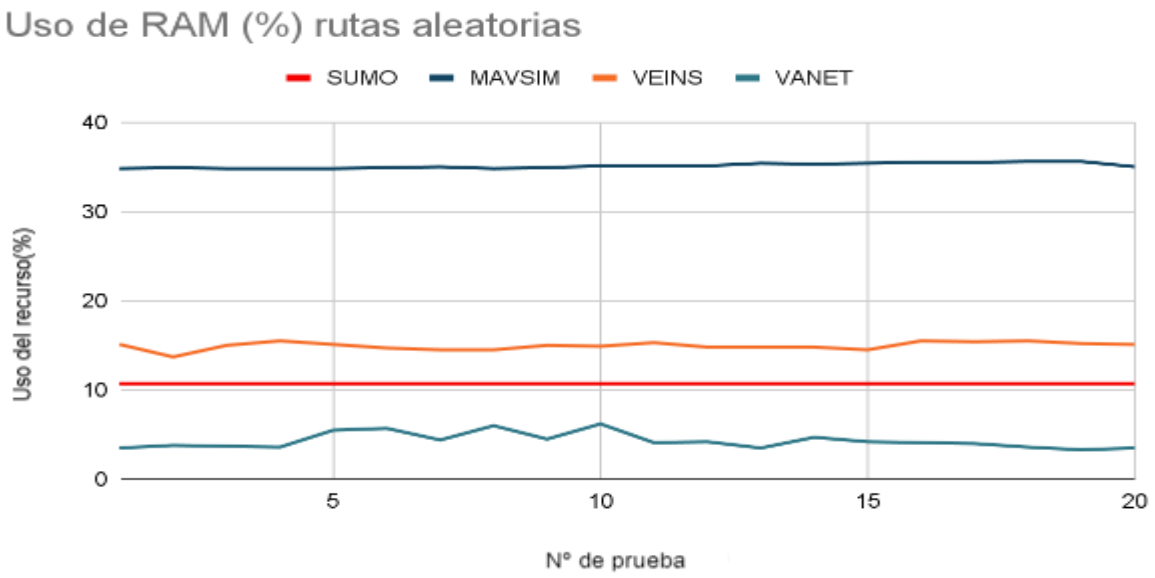


Figura 4.20: Uso de RAM *Barcelona-2.osm*

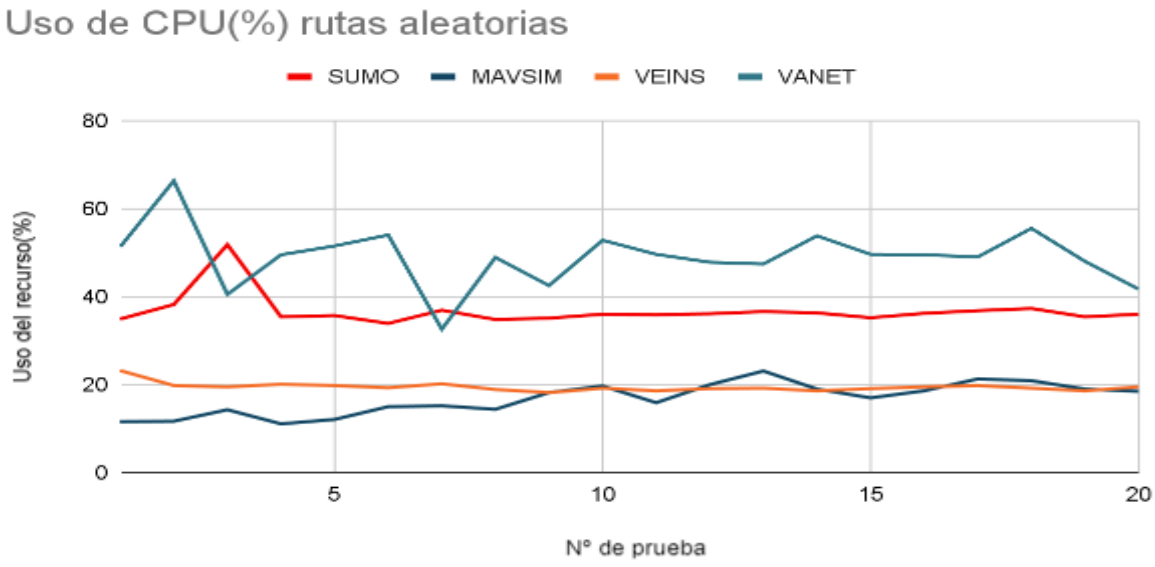


Figura 4.21: Uso de RAM *Dublin.osm*

Otro factor relacionado con el rendimiento es que SUMO es el único simulador de los 4 que utiliza el almacenamiento en disco para procesar sus ejecuciones. De esta forma está reduciendo el uso de RAM almacenando lo que no necesita procesar temporalmente en memoria. Esta es la principal razón por la que el uso de RAM de SUMO es increíblemente estable, mientras que el uso de CPU si es variable.

4.5.3. Rendimiento en tiempo

Centrando la mirada ahora en el apartado de rendimiento en tiempo, hay un problema base en su análisis. Cada simulador trata la ejecución de su simulación de una forma distinta (por iteraciones, por eventos, por milisegundos de ejecución, etc) y esto hace complicado comparar tiempos entre sí. El enfoque que se ha tomado ha sido el de comparar como se adapta el tiempo de cada una de las simulaciones entre sí. De esta forma podemos valorar como de efectivo es el rendimiento en tiempo de cada simulador según como se han adaptado a cada cambio en la simulación.

El simulador que mejor parece adaptar su rendimiento en tiempo es Veins, sus simulaciones son las más rápidas e incluso en el mapa *Barcelona-2.osm* las ejecuciones terminan dentro del tiempo de simulación. Aunque SUMO obtiene tiempos similares muchas ejecuciones terminan por llegar al tiempo de simulación dejando muchos coches sin terminar sus recorridos.

Por parte de VanetMobisim los tiempos que muestra en los distintos mapas reflejan un comportamiento marcado por el tiempo de preparación que necesita para iniciar la simulación. Por ejemplo para *Dublin.osm* los tiempos son muy bajos, alrededor de 10 segundos, pero para *Barcelona-2.osm* o *London.osm* los tiempos se disparan hasta los 4 o 6 minutos. La diferencia radica en que *Dublin.osm* es un mapa muy pequeño en comparación con el resto y su preparación tarda mucho menos, unos 5 segundos.

4.5.4. Facilidad de uso e interfaz

Después del rendimiento, la interfaz o como de fácil es usar cada simulador es lo más importante a la hora de elegir el programa a utilizar. No es lo mismo lanzar simulaciones con un par de clics y poder establecer los parámetros que se quiera que tener que modificar archivos/directorios y relanzar el simulador.

Teniendo lo anterior en cuenta, el programa que ofrece la experiencia más sólida al usuario es SUMO. Su interfaz es simple y fácil de entender, ofrece muchos detalles sobre la simulación y se puede observar con gran detalle las simulaciones de los diferentes mapas. Sin embargo, carece de opciones a la hora de ejecutar la simulación, ya que solo permite o parar o ejecutar. No tiene modos de ejecución para poder ralentizar la ejecución y observar con más detalle las interacciones. Además, por los colores y las formas utilizadas para su representación no se entiende fácilmente a nivel global que está ocurriendo en la simulación.

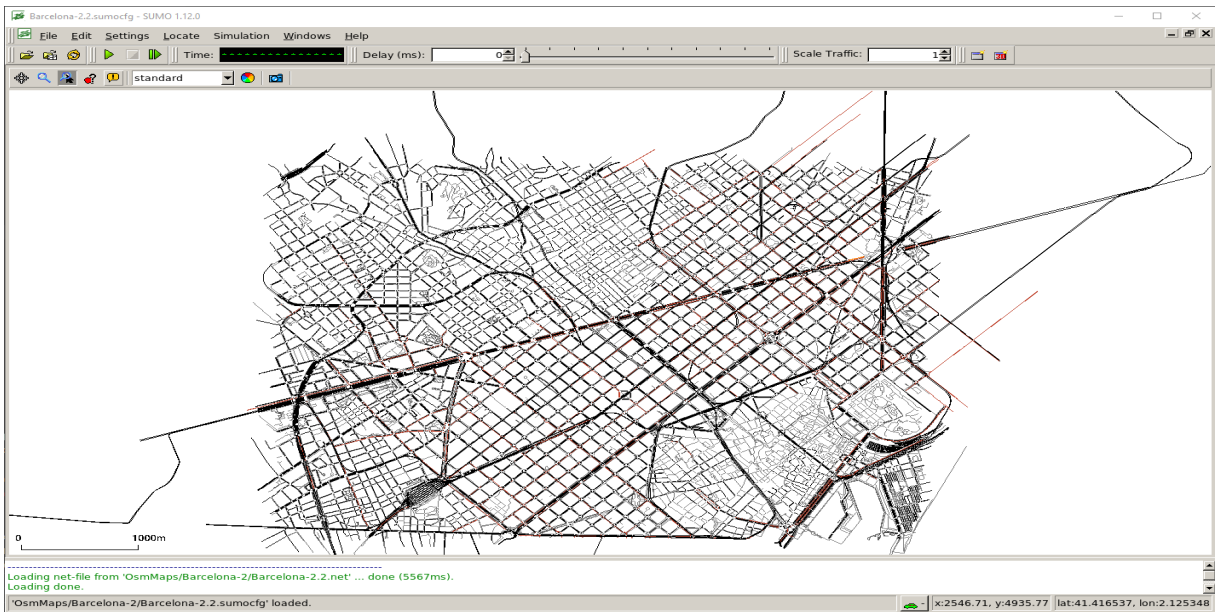


Figura 4.22: Uso de RAM *Dublin.osm*

MAVSIM y Veins sí que tienen interfaces simples y con detalle, pero ambos flaquean por algún lado. Veins tiene una instalación y configuración larga y compleja en la que varios programas tienen que funcionar correctamente. Por su parte, MAVSIM sí que tiene una instalación fácil, no es necesario ninguna configuración y trae incorporado su propia descarga de mapas. Pero tiene un entorno cerrado, es algo molesto introducir mapas si no es usando su gestor de mapas y los parámetros deben ser introducidos mediante su interfaz.

Capítulo 5

Conclusiones

El paradigma de la simulación de redes vehiculares es un problema muy presente en la actualidad. La posibilidad de un futuro en el que el tráfico en las ciudades este controlado por programas con gran precisión es muy llamativa, pero está lejos de ser posible hoy en día. Los programas que se identifican como simuladores de redes VANET tratan de aproximarse a una solución que permita implementar la tecnología de las redes VANET en un escenario real. Y cada simulador trata de llegar a esta solución desde un punto de vista, aunque algunos se inspiren en las estrategias más populares.

Algunos de los programas incluidos en este proyecto consiguen aproximaciones precisas y eficientes, pero fallan en algunos aspectos como las funcionalidades disponibles, una representación clara de la simulación o un rendimiento mejorable. Sin embargo, personalmente, no veo que estas aplicaciones sean lo suficientemente robustas como para una aplicación real. El programa que parece conseguir la solución más robusta es SUMO. El programa es sencillo de usar, muy preciso en sus simulaciones y un uso de los recursos minucioso. También tiene un control de los coches y sus rutas muy preciso que permite, incluso, describir los eventos de cada cruce con gran detalle.

Aun así, el rumbo que están tomando los simuladores sí que apunta a que en un futuro se llegara a desarrollar programas capaces de poner en funcionamiento soluciones para el paradigma de la simulación de redes vehiculares. Si algún día se quiere llegar a aplicaciones reales, será necesario mejorar la tecnología de comunicacion ya que los rendimientos observados en este proyecto más los problemas que conlleva establecer comunicaciones podrían echar por tierra cualquier solución realista a la que se llegue.

Capítulo 6

Trabajo futuro

Como trabajo futuro, lo más llamativo sería realizar pruebas de mayor magnitud (varios días de tiempo de simulación o con tamaños de mapa y número de coches mucho mayor) utilizando máquinas de la universidad. Esto podría mostrar otros comportamientos de los simuladores de VANET u obtener resultados más útiles para una aplicación real de esta tecnología.

Otras opciones para un trabajo futuro serian:

- Diseñar más pruebas que pongan a prueba otros aspectos de los simuladores o, por ejemplo, que permitan medir el rendimiento de los simuladores según el porcentaje de una simulación que son capaces de ejecutar (de esta forma se podría comparar MAVSIM con el resto de simuladores).
- Automatizar por completo las pruebas de simuladores VANET para poder realizar pruebas en masa durante periodos largos de tiempo o con gran variedad de mapas y simuladores.
- Explorar otros simuladores del mercado actual.
- Realizar cambios en el simulador MAVSIM de la universidad, a raíz de lo aprendido en otros simuladores.
- Investigar sobre una posible aplicación a un escenario real, aunque en escala sea pequeño, para poder ver resultados reales en un entorno donde todos los posibles factores de riesgo estén presentes.

Capítulo 7

Bibliografía

[1] Oscar Orozco Sarasti and Gonzalo Llano Ramírez. Vanet applications focused on environmental sustainability, a systematic review, 2014. Referencia: O. Orozco Sarasti, G. Llano Ramírez. (2014). Aplicaciones para redes VANET Estimación del exponente de Hurst y dimensión fractal para el análisis de series de tiempo de absorbancia UV-Vis. Ciencia e Ingeniería Neogranadina, 24 (2), pp. 111-132.

[2] Trapti Jain and Savita Shiwani. Analysis of olsr, dsr, dymo routing protocols in mobile ad-hoc networks using omnet++ simulation, 2014. Global Journal of Computer Science and Technology. 14, E1 (Jan. 2014).

[3] Óscar Urra, Sergio Ilarri, and Eduardo López. Simulating mobile agents in vehicular networks, 2014. XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2014), Cádiz (Spain), ISBN-13: 978-84-697-1152-1, ISBN-10: 84-697-1152-0,Editores: Javier Tuya.

[4] Óscar Urra and Sergio Ilarri. Chapter 10. mavsim: Testing vanet applications based on mobile agents, 2016. in Cognitive Vehicular Networks, pages=199–224, Anna Maria Vegni, Dharma P. Agrawal (editors), CRC Press - Taylor Francis Group, Print ISBN 978-1-4987-2191-2, eBook ISBN 978-1-4987-2192-9, (DOI: 10.1201/b19351-14).

[5] Inet framework open-source model library for omnet++, 2022. Jul 27, 2022.

[6] Sergio Ilarri, Thierry Delot, and Raquel Trillo-Lado. A data management perspective on vehicular networks, 2015. ISSN 1553-877X, volume 17, number 4, IEEE, Fourthquarter. (DOI: 10.1109/COMST.2015.2472395).

Lista de Figuras

2.1. Configuración de una simulación en *SUMO* 5
2.2. Representación de los coches 7
2.3. Modos de velocidad de Veins 9

3.1. Programa SUMO iniciado 12
3.2. Programa MAVSIM iniciado 13
3.3. Parámetros de MAVSIM 14
3.4. Entorno OMNet++ 15
3.5. Simulación en Veins ejecutándose 16
3.6. Simulación ejecutándose en VanetMobisim 17
3.7. Formato de una simulación en VanetMobisim 18

4.1. Salida por terminal de SUMO 21
4.2. Vista global de la simulación en SUMO 22
4.3. Uso de RAM *Barcelona-2* 23
4.4. Evolución del tiempo con el mapa *Dublin.osm* 24
4.5. Evolución del tiempo con el mapa *Barcelona-2.osm* 24
4.6. Tiempos de 20 simulaciones con rutas aleatorias en SUMO 25
4.7. Tiempos rutas aleatorias mapa London en SUMO 25
4.8. Página de documentación de Veins 27
4.9. Evolución del tiempo con el mapa *Barcelona-2.osm* 28
4.10. Evolución del tiempo con el mapa *Dublín* 29
4.11. Tiempos con rutas aleatorias London 30
4.12. Gestor de Mapas de MAVSIM 32
4.13. Parámetros de una simulación en MAVSIM 33
4.14. Uso de CPU con el mapa *Dublin.osm* 34
4.15. Simulación ejecutándose en VanetMobisim 36
4.16. Uso de CPU con el mapa *London.osm* 38
4.17. Pruebas fallidas/correctas con el mapa *Barcelona-2.osm* 38
4.18. Tiempos con rutas aleatorias *Barcelona-2.osm* 39

4.19. Tiempos con rutas aleatorias con el mapa *Barcelona-2.osm* 41
4.20. Uso de RAM *Barcelona-2.osm* 43
4.21. Uso de CPU *Dublin.osm* 43
4.22. Interfaz 45

A.1. Diagrama de Gantt del proyecto. 53

Anexos

Anexos A

Gestión del proyecto

El proyecto se ha llevado a cabo desde mediados de Marzo de 2022 hasta Mediados de Noviembre de 2022, con un aumento en horas invertidas a partir de Agosto/Septiembre. Las tareas realizadas así como el tiempo dedicado a cada tarea se puede observar en la tabla A.1. Durante estos meses ha habido muchas horas muertas, no reflejadas en esta tabla debido a problemas técnicos con los simuladores, ya sea por falta de documentación o fallos del propio código del simulador en cuestión, que se verán reflejados en los apartados de cada programa.

Tarea	*Horas dedicadas*
Estudio y elección de los simuladores a evaluar	20 h
Obtención de mapas, fuentes y realización de pruebas iniciales	128 h
Implementación del programa que permita (semi-)automatizar las pruebas a realizar	96 h
Diseño de las pruebas y decisión de los parámetros a utilizar en las simulaciones	96 h
Documentación del rendimiento y de los resultados obtenidos	128 h
Comparativa final entre los simuladores evaluados, tanto cuantitativa como cualitativa	96 h
Memoria	64 h
Reuniones	32 h
Total:	660 h

Tabla A.1: Horas/Tarea dedicadas al proyecto.

En la figura A.1 se muestra el diagrama de Gantt que refleja qué tareas han sido realizadas cada semana durante la duración del proyecto.

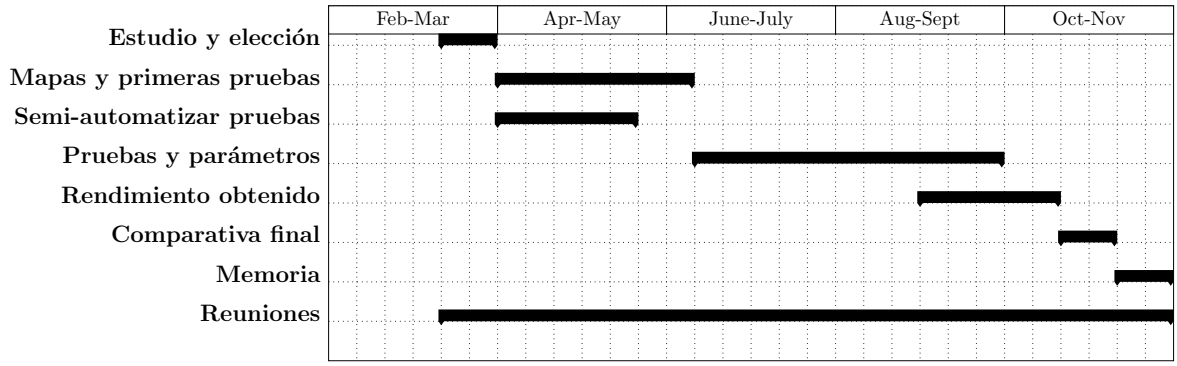
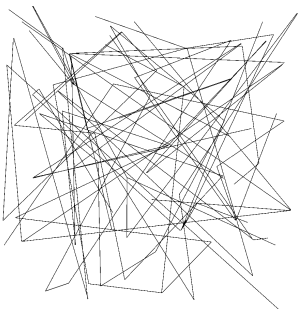


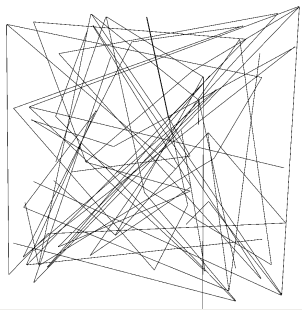
Figura A.1: Diagrama de Gantt del proyecto.

Anexos B

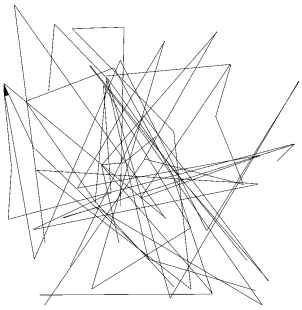
Imágenes de mapas aleatorios de apartado 4.5



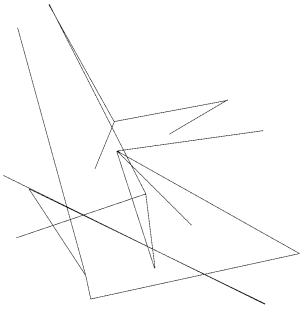
Mapa 1



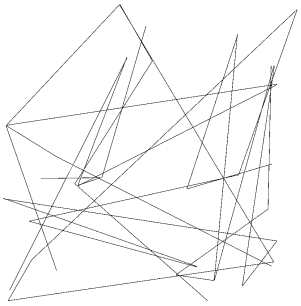
Mapa 2



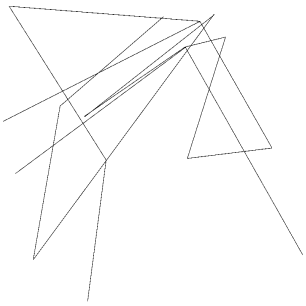
Mapa 3



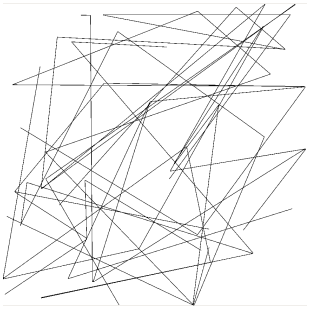
Mapa 4



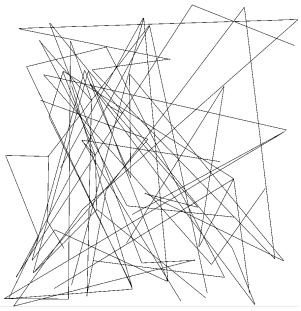
Mapa 5



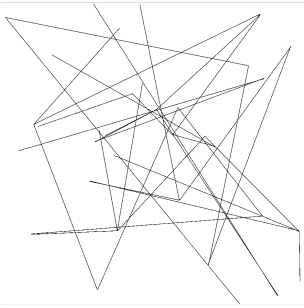
Mapa 6



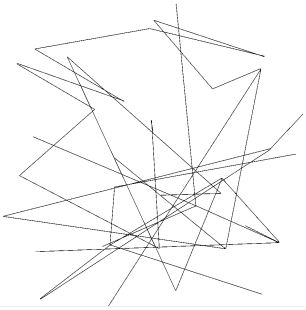
Mapa 7



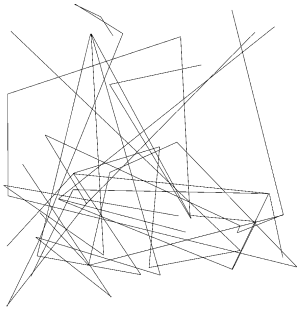
Mapa 8



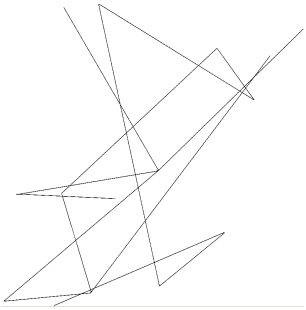
Mapa 9



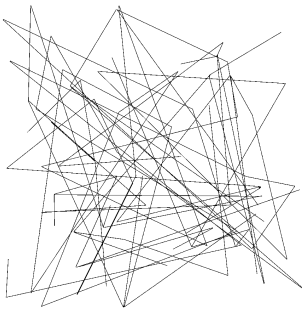
Mapa 10



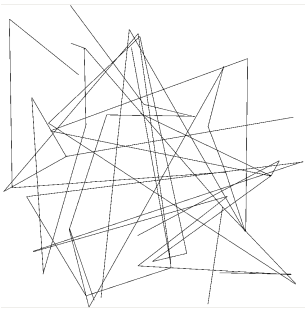
Mapa 11



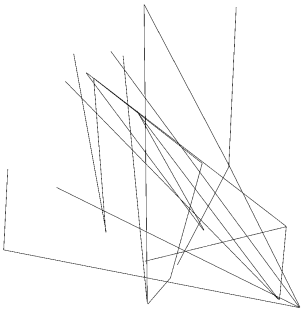
Mapa 12



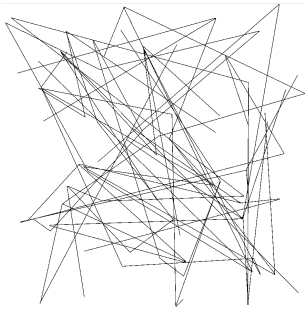
Mapa 13



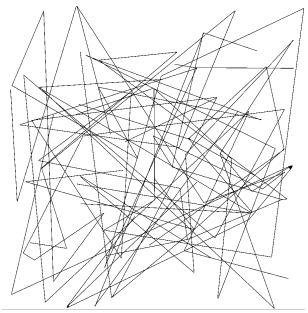
Mapa 14



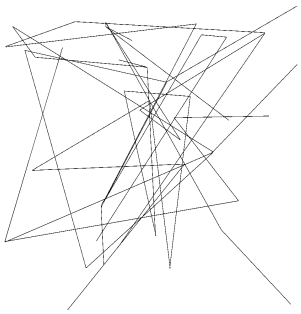
Mapa 15



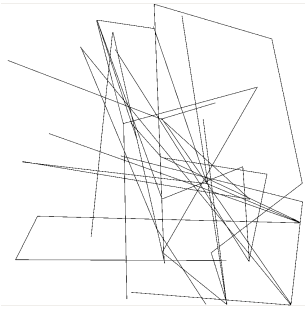
Mapa 16



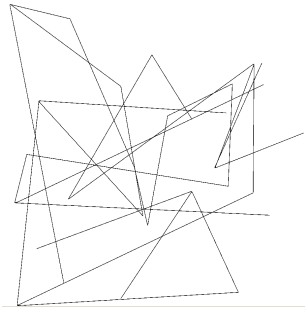
Mapa 17



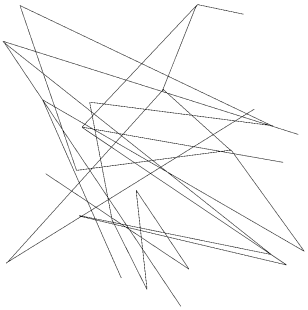
Mapa 18



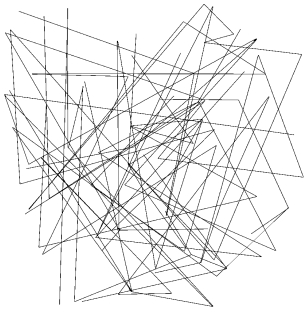
Mapa 19



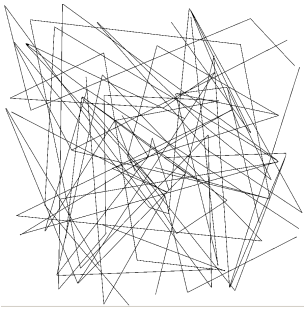
Mapa 20



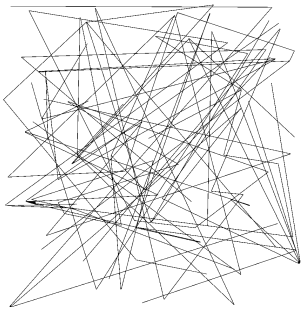
Mapa 21



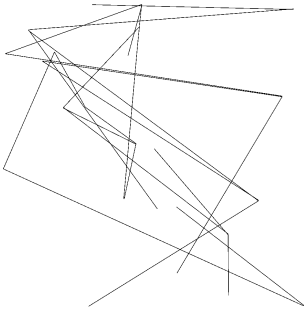
Mapa 22



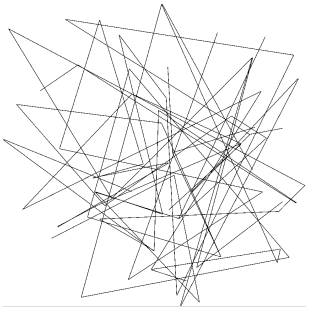
Mapa 23



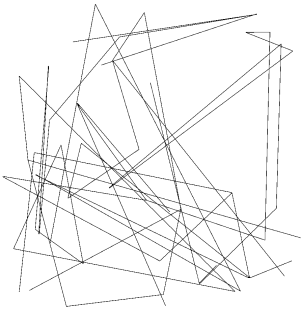
Mapa 24



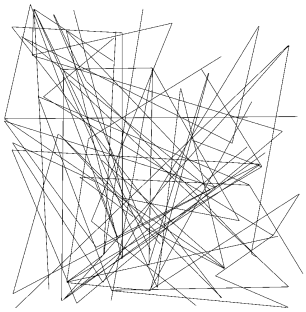
Mapa 25



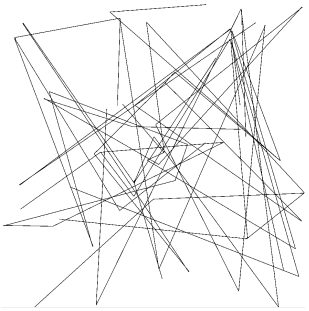
Mapa 26



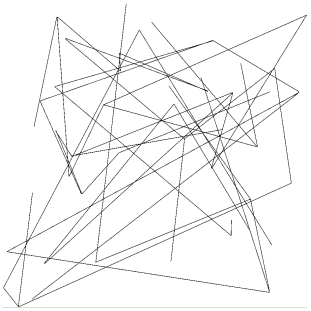
Mapa 27



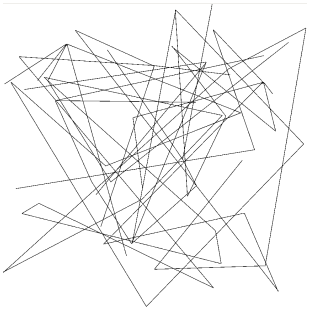
Mapa 28



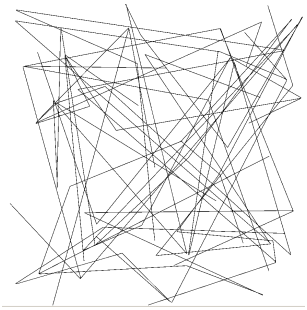
Mapa 29



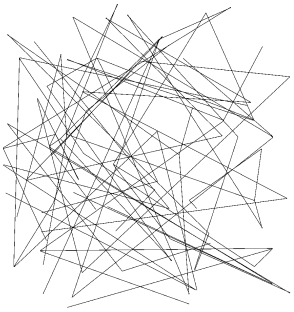
Mapa 30



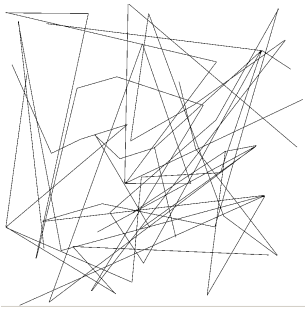
Mapa 31



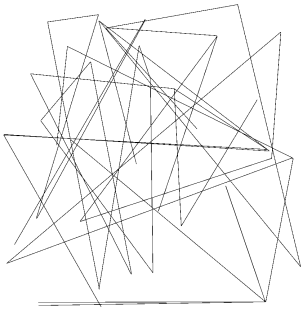
Mapa 32



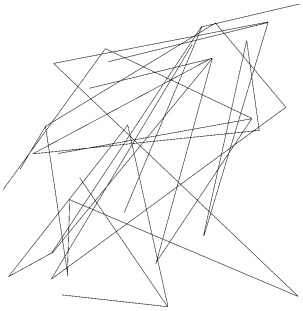
Mapa 33



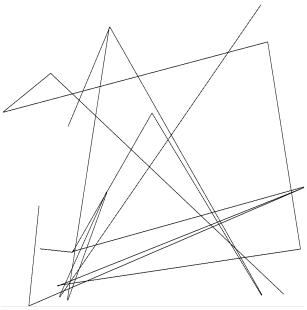
Mapa 34



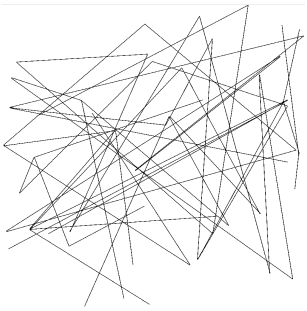
Mapa 35



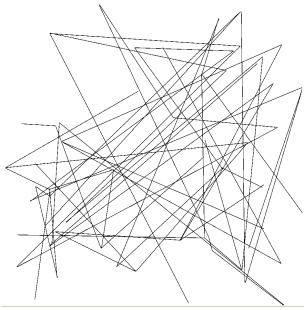
Mapa 36



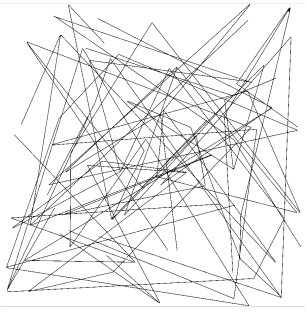
Mapa 37



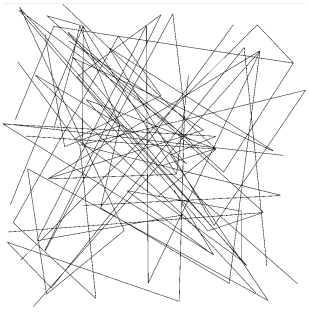
Mapa 38



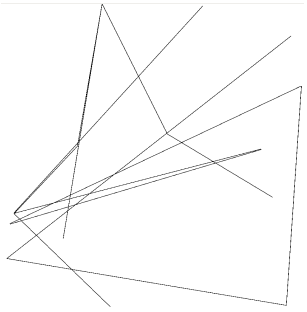
Mapa 39



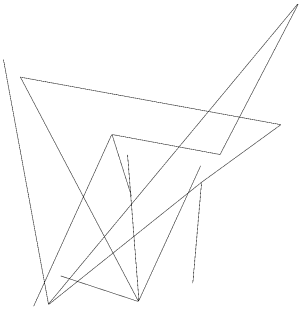
Mapa 40



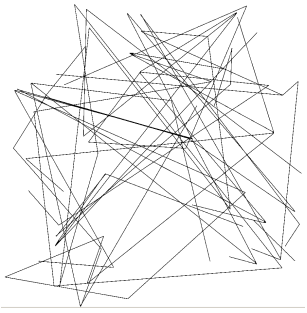
Mapa 41



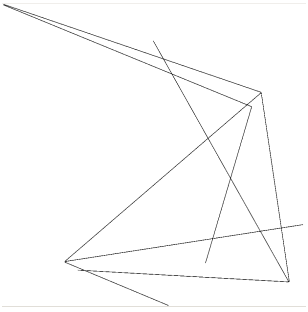
Mapa 42



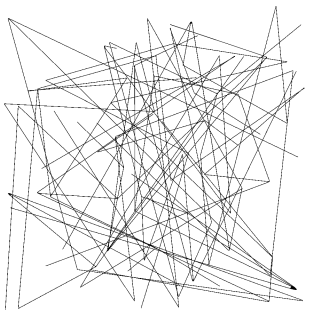
Mapa 43



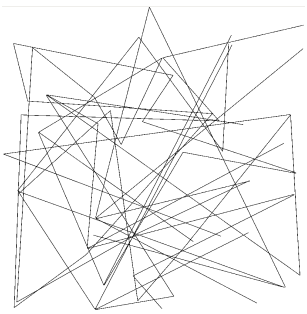
Mapa 44



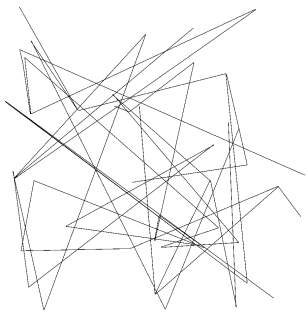
Mapa 45



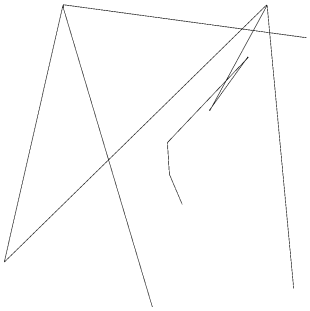
Mapa 46



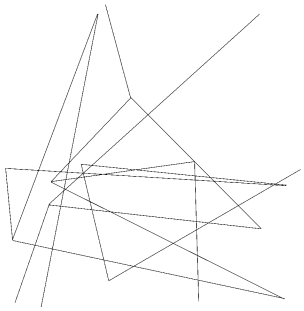
Mapa 47



Mapa 48



Mapa 49



Mapa 50

Anexos C

Documento de pruebas

En este documento de pruebas se encuentran todas las pruebas realizadas durante el proyecto. La gran mayoría de ellas va acompañado de cierto análisis de los resultados, además de algunas explicaciones de como se han hecho las tablas y algunas anotaciones.

ÍNDICE

Dublin.osm **2**
- SUMO 3
- MAVSIM 5
- VEINS 7
- VANETMOBISIM 9
- RECURSOS UTILIZADOS 12

Barcelona-2.osm **13**
- SUMO 14
- MAVSIM 17
- VEINS 20
- VANETMOBISIM 23
- RECURSOS UTILIZADOS 26

London.osm **27**
- SUMO 28
- MAVSIM 31
- VEINS 34
- VANETMOBISIM 37
- RECURSOS UTILIZADOS 41

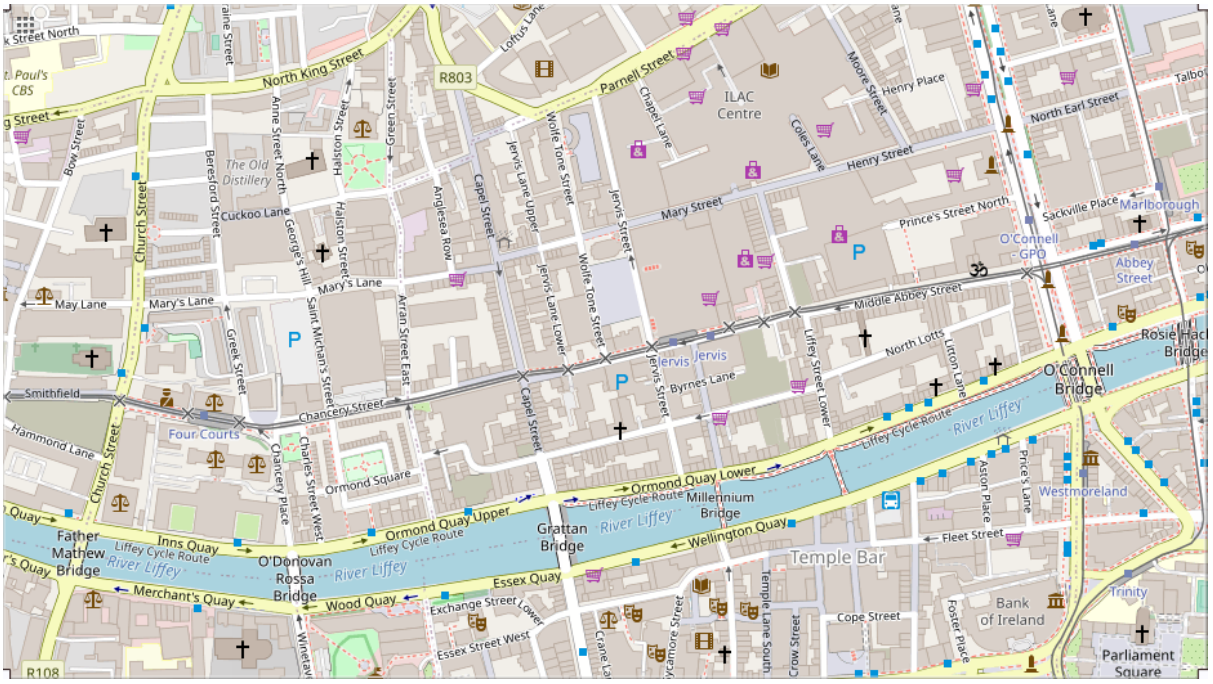
RESULTADOS MAPAS ALEATORIOS **43**
- TIEMPO DE SIMULACIÓN MAPAS ALEATORIOS 44
- IMÁGENES DE LOS MAPAS ALEATORIOS 45

Prueba con Mapas OpenStreetMap reales

Dublin.osm

Dimensiones del mapa

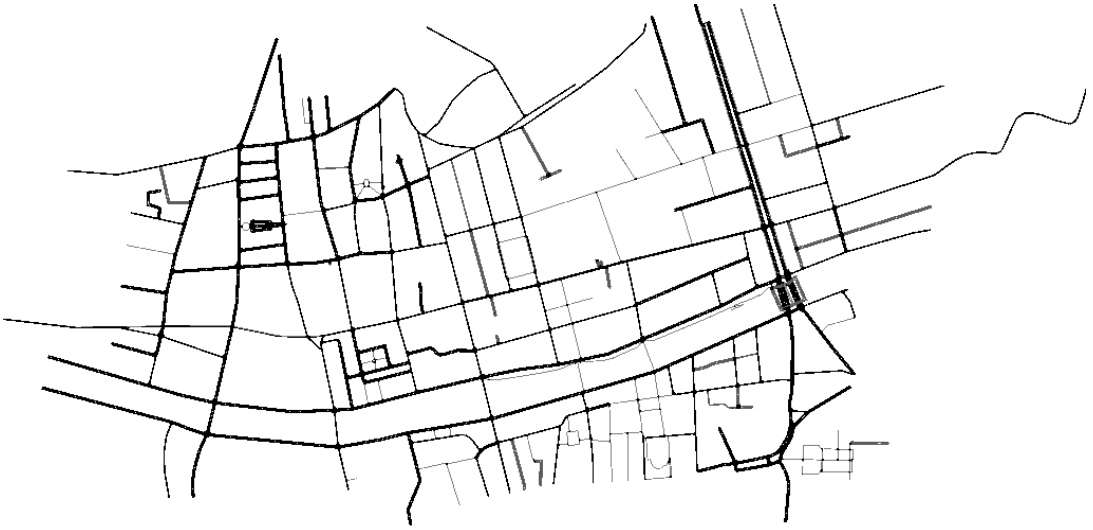
- 695 x 1.334 m de Longitud
- 927.130 m2 de Superficie



Tiempo de simulación: 10.000 ms

Número de coches: 500

SUMO



Tiempo real de simulación: 21 segundos

Gestión de bloqueos: 0 bloqueos. Sumo resuelve los bloqueos mediante teletransportaciones, en un mapa muy grande con solo 500 coches los bloqueos son inexistentes.

Configuración de la simulación:
A los 21 segundos de iniciar la simulación los 500 coches habían desaparecido. Probablemente porque ya han cumplido sus rutas preestablecidas.

nº Teletransportaciones: 0. No ha ocurrido ninguna teletransportación, dependiendo del timeout establecido en las mismas.
Si se aumenta el número de coches o el escalado del tráfico las teletransportaciones enseguida crecen hasta detener la simulación por sobrepasar el límite de 50 establecido.

TABLA DE VARIACIÓN DEL TIEMPO DE SIMULACIÓN REAL (TIEMPO DE SIMULACIÓN y COCHES)

	50 coches	125 coches	300 coches	750 coches	1900 coches
250 ms	6s*	6 s*	6 s*	6 s*	6 s*
625 ms	9s	10s	15s	16s*	16s*
1560 ms	9s	10s	15s	27s	41s*
3900 ms	9s	10s	15s	27s	57s
10000 ms	9s	10s	15s	27s	57s

*Estas simulaciones terminaron porque se llegó al tiempo de simulación estipulado, los coches no terminaron sus rutas.

Como las rutas se generan aleatoriamente también se ha realizado una serie de pruebas con los valores por defecto (10.000 ms de tiempo de simulación y 500 coches) para observar el efecto que tienen estas rutas aleatorias.

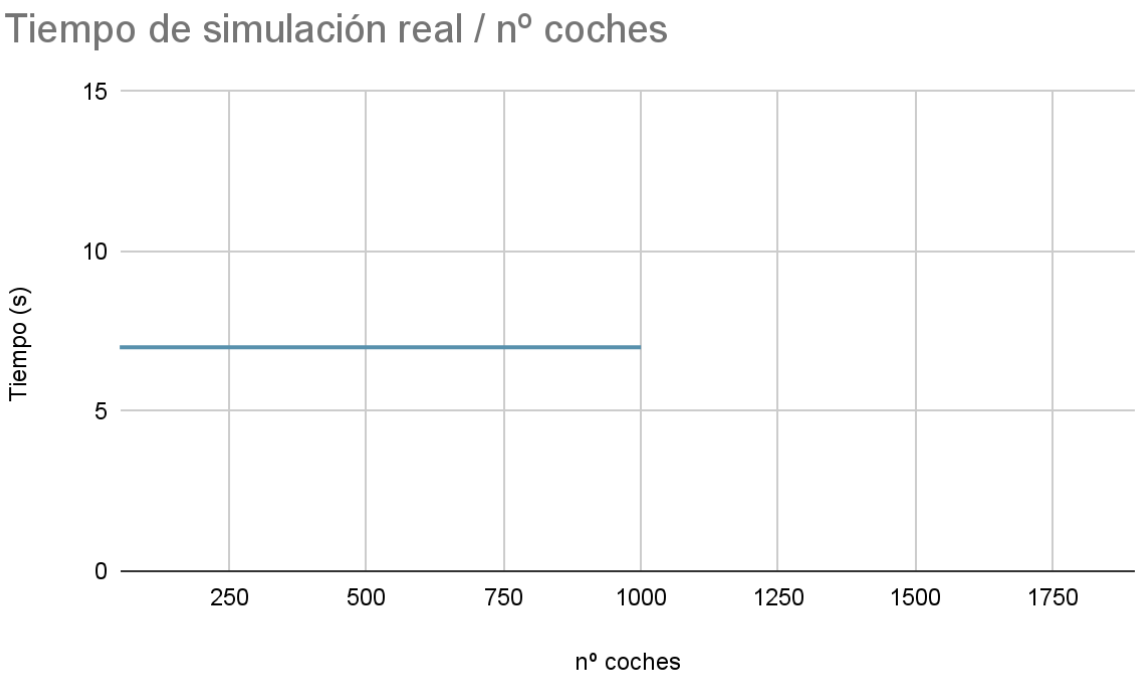
TIEMPO DE SIMULACIÓN DE 20 PRUEBAS (RUTAS ALEATORIAS)

MAVSIM

Tiempo real de simulación: 15 segundos. Al contrario que SUMO, no existen las teletransportaciones, la simulación acaba siempre cuando indica el tiempo de simulación configurado. Permite dos velocidades, normal y muy rápido. No especifica qué velocidad es cada una (Paso en milisegundos)

Pero si que se ve mas claro el tráfico que con SUMO, los coches se ven mucho mejor, aunque cuantos mas coches peor va a funcionar.

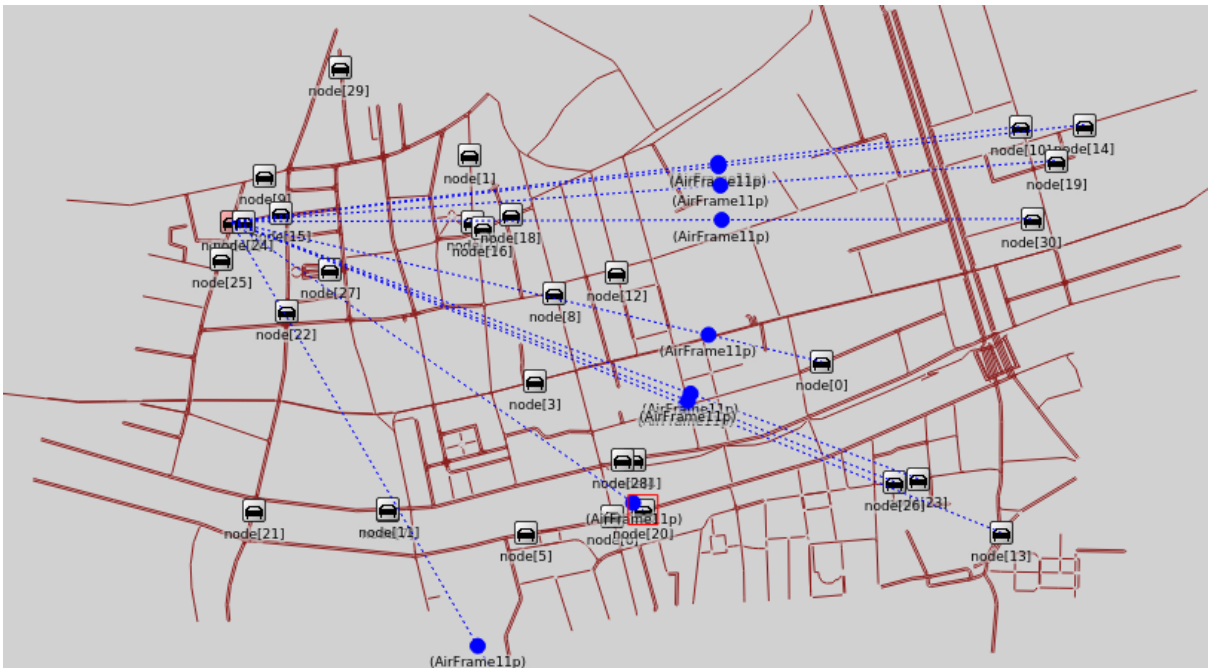
TIEMPO DE SIMULACIÓN REAL VARIACIÓN DEL Nº DE COCHES



Todas las pruebas se detienen exactamente a la vez, ya que la simulación va por iteraciones, esto significa que por más coches que se añadan o la variedad de rutas aleatorias que se generen el tiempo real de simulación será siempre el mismo. Además, hay que añadir que hay un número máximo de coches en 999, esto dificulta la ejecución de las pruebas como se observa en la gráfica resultante.

TIEMPO DE SIMULACIÓN DE 20 PRUEBAS (RUTAS ALEATORIAS)

VEINS



Tiempo real de simulación: 26 segundos (con la velocidad intermedia o “Fast”). La simulación en concreto acaba a los 10000 segundos de “simulación”, aunque en realidad cuando no quedan coches la simulacion salta directamente al final.

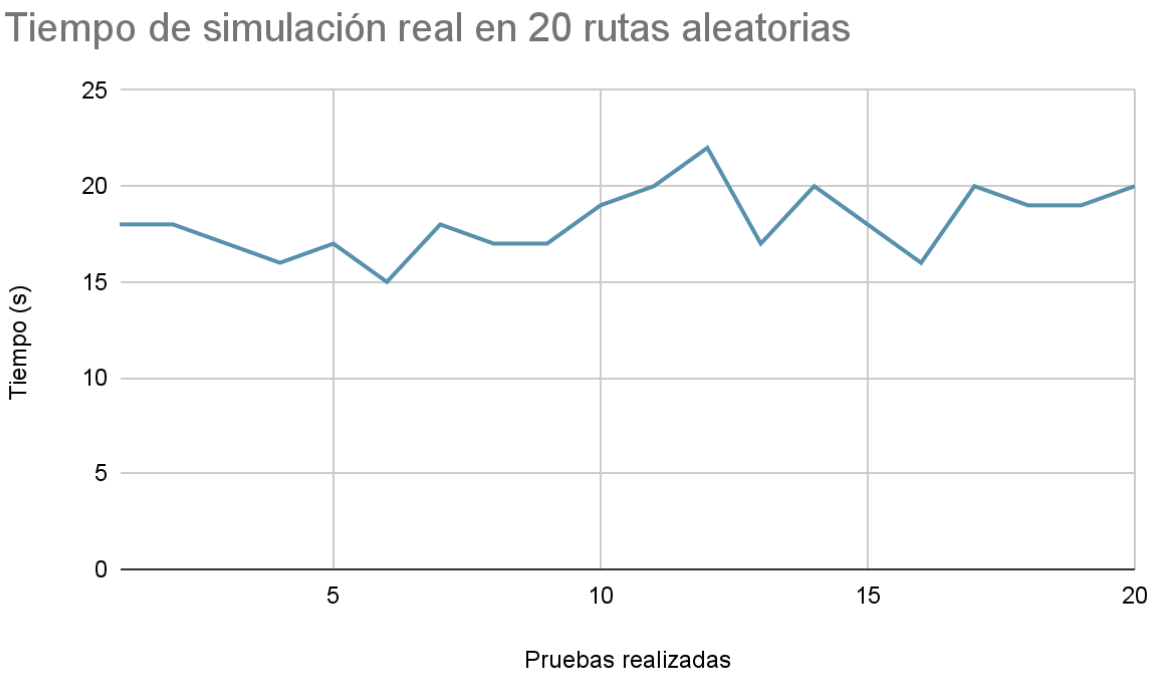
Veins permite 3 velocidades: la primera permite seguir los movimientos de cada uno de los coches y cada uno de los nodos que atraviesan, además de cómo se envían paquetes de información entre sí, la segunda velocidad no añade retraso entre los output mostrados en la terminal, esto acelera mucho la simulación aunque dificulta en gran medida seguir los pasos de los coches tanto en terminal como en el mapa y la última velocidad que evita cualquier feedback de terminal y mapa haciendo imposible seguir la ejecución.

TABLA DE VARIACIÓN DEL TIEMPO DE SIMULACIÓN REAL (TIEMPO DE SIMULACIÓN y COCHES)

	50 coches	125 coches	300 coches	750 coches	1900 coches
250 s	1s	2s	5s*	5s*	5s*
625 s	1s	3s	7s	24s*	24s*
1560 s	1s	3s	8s	32s	83s*
3900 s	1s	3s	8s	32s	102s*
10000 s	1s	3s	8s	32s	126s

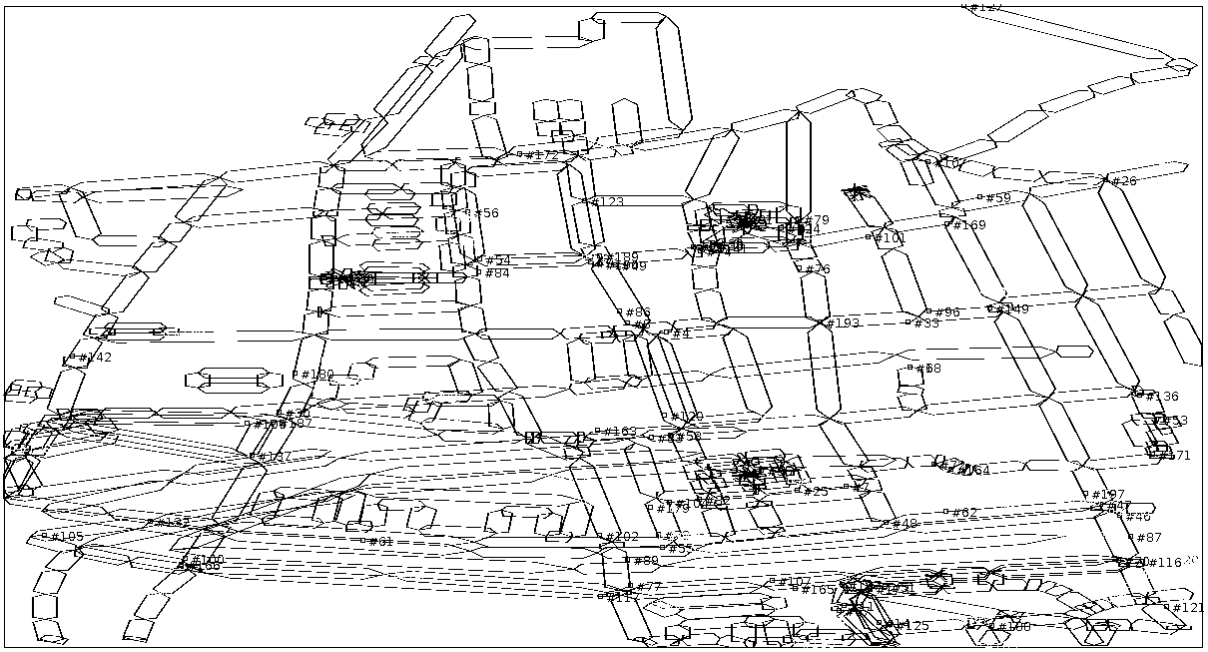
Aunque estas pruebas muestran tiempos de simulación reales muy bajos, no se puede observar el comportamiento de la simulación con esta velocidad, la anterior velocidad es demasiado lenta (para 250 s y 50 coches tarda casi 1 minuto en finalizar) pero muestra con buen detalle la iteración entre los coches y los mensajes que estos se envían sobre sus posiciones y rutas.

TIEMPO DE SIMULACIÓN DE 20 PRUEBAS (RUTAS ALEATORIAS)



Estas rutas aleatorias han sido simuladas con 500 coches y 10.000 segundos como parámetros. Como se observa en la gráfica el comportamiento gira alrededor de los 17 segundos, con un mínimo de 15 y un máximo de 22, el efecto de las rutas aleatorias es relativamente mayor a la observada, por ejemplo, en el simulador SUMO.

VANETMOBISIM



Tiempo real de simulación: 10 segundos (la simulación se detiene tras 390 ms de simulación). El programa devuelve un error, no da ningún *feedback* del error asi que supondremos que es porque algún coche no encuentra un siguiente destino, ya que al ser rutas aleatorias puede ocurrir).

Configuración de la simulación: De entre todos los simuladores este es el que peor interfaz muestra, el que peor representa los mapas reales OpenStreetMap, el único que utiliza un formato propio sin dar las herramientas necesarias para usarlo, la interfaz no te da opción alguna a modificar los parámetros de la simulación u opción de visualización y como se observa en la imagen de ejemplo del mapa Dublin.osm el mapa resultante es inteligible e imposible de seguir durante la ejecución.

A todo lo anterior hay que añadir que no permite especificar número de coches, o si lo permite, la escasa documentación no lo menciona (solo explica como definir escenarios) y tampoco pone ningún ejemplo funcional.

VARIACIÓN DEL TIEMPO DE SIMULACIÓN REAL

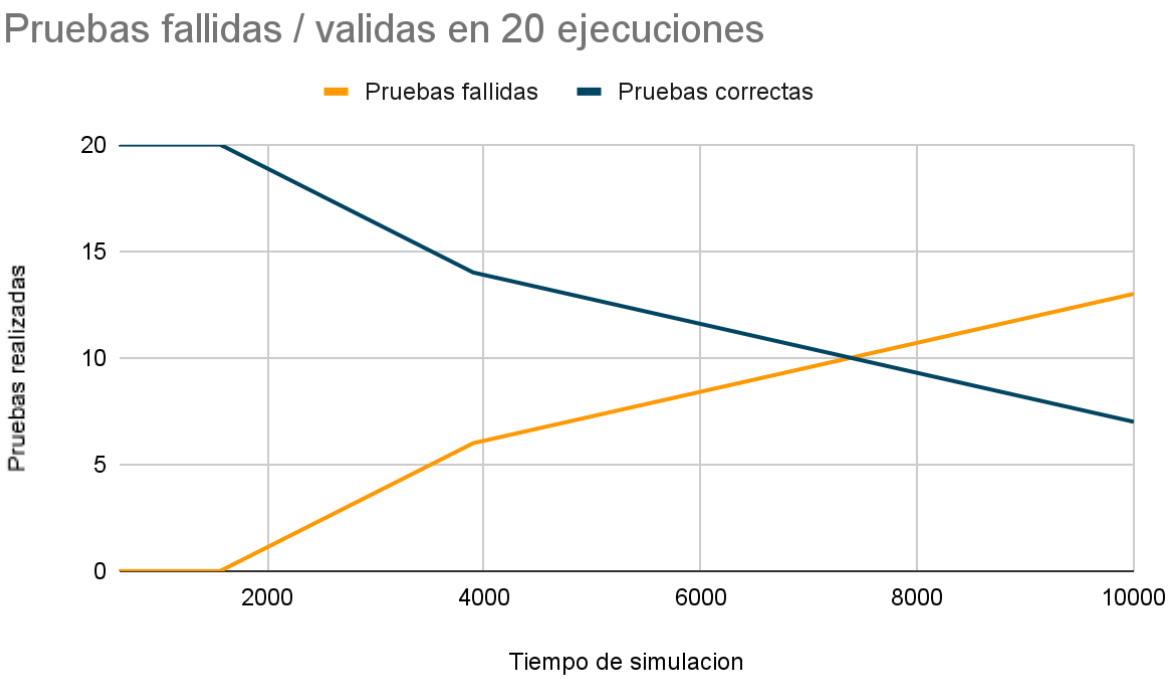
En cuanto a las pruebas de tiempo de simulación y número de coches el simulador no permite realizarlas de igual manera que el resto de programas, por tanto se va a optar por ejecutar VanetMobisim para 25 pruebas aleatorias para comprobar cómo de estable es el rendimiento que este muestra variando el tiempo de simulación. Sin embargo se van a descartar las pruebas fallidas (que devuelve el error ya mencionado en el apartado anterior).

Los parámetros para estas pruebas aleatorias serán 500 coches y 3900 ms (390 ms) de tiempo de simulación (hay que recordar que los tiempos de simulación reales para VanetMobisim están reducidos en un factor de 1/10 respecto al resto de simuladores, esto es debido al horrible rendimiento en tiempo que este muestra).

La gráfica que se obtiene parece mostrar un comportamiento muy regular, no parece que las rutas aleatorias afecten al tiempo de simulación, esas pequeñas variaciones deben ser por el número de coches (el simulador elige el número que quiere de coches).

Podemos suponer que en este caso hay menos variación en los tiempos obtenidos ya que el número de coches afecta poco en un mapa pequeño, principalmente porque en 390 ms de simulación no da tiempo a que recorran sus rutas. Para simulaciones lo suficientemente largas los coches terminarán sus rutas y la simulación debería verse afectada en mayor medida.

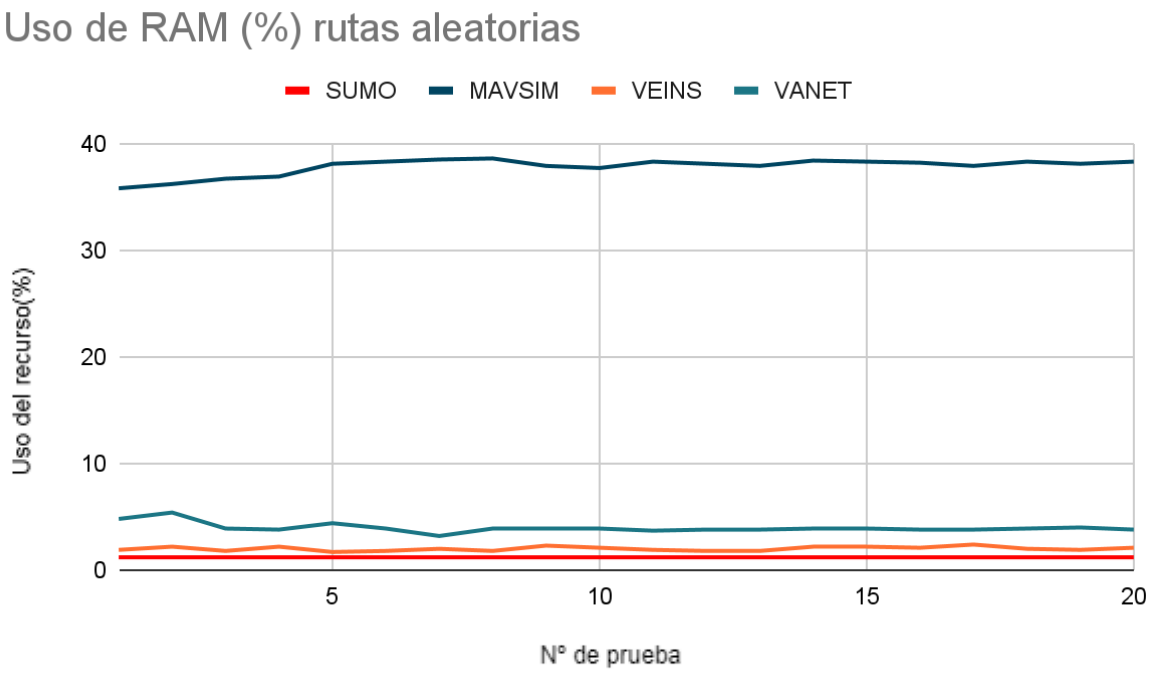
FALLOS DE SIMULACIÓN

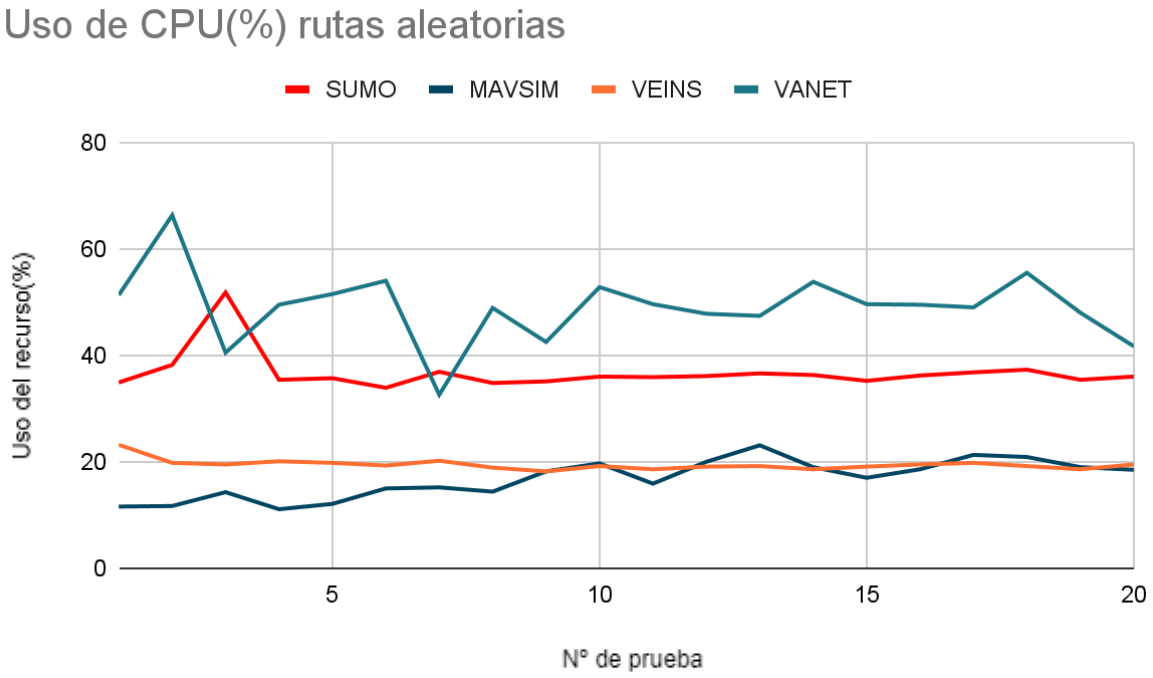


Como se puede observar en la gráfica de arriba mientras que el tiempo de simulación aumenta la probabilidad de fallo de la ejecución aumenta. Cuanto mas dura una ejecución mas posibilidades hay de que alguna ruta llegue a un camino sin salida y el programa termine sin previo aviso. En estas pruebas se ve reflejado uno de los mayores problemas de VanetMobisim, si alguno de los coches está siguiendo su ruta aleatoria y llega a un camino sin salida este detiene el programa y da un error fatal parando la ejecución sin reacción posible.

RECURSOS UTILIZADOS

En esta sección se va a analizar el consumo de recursos de cada uno de los simuladores, la idea es representar la gráfica de recursos utilizados en 20 ejecuciones aleatorias para cada simulador y, de esta forma, comparar el rendimiento de cada programa y su adaptabilidad.



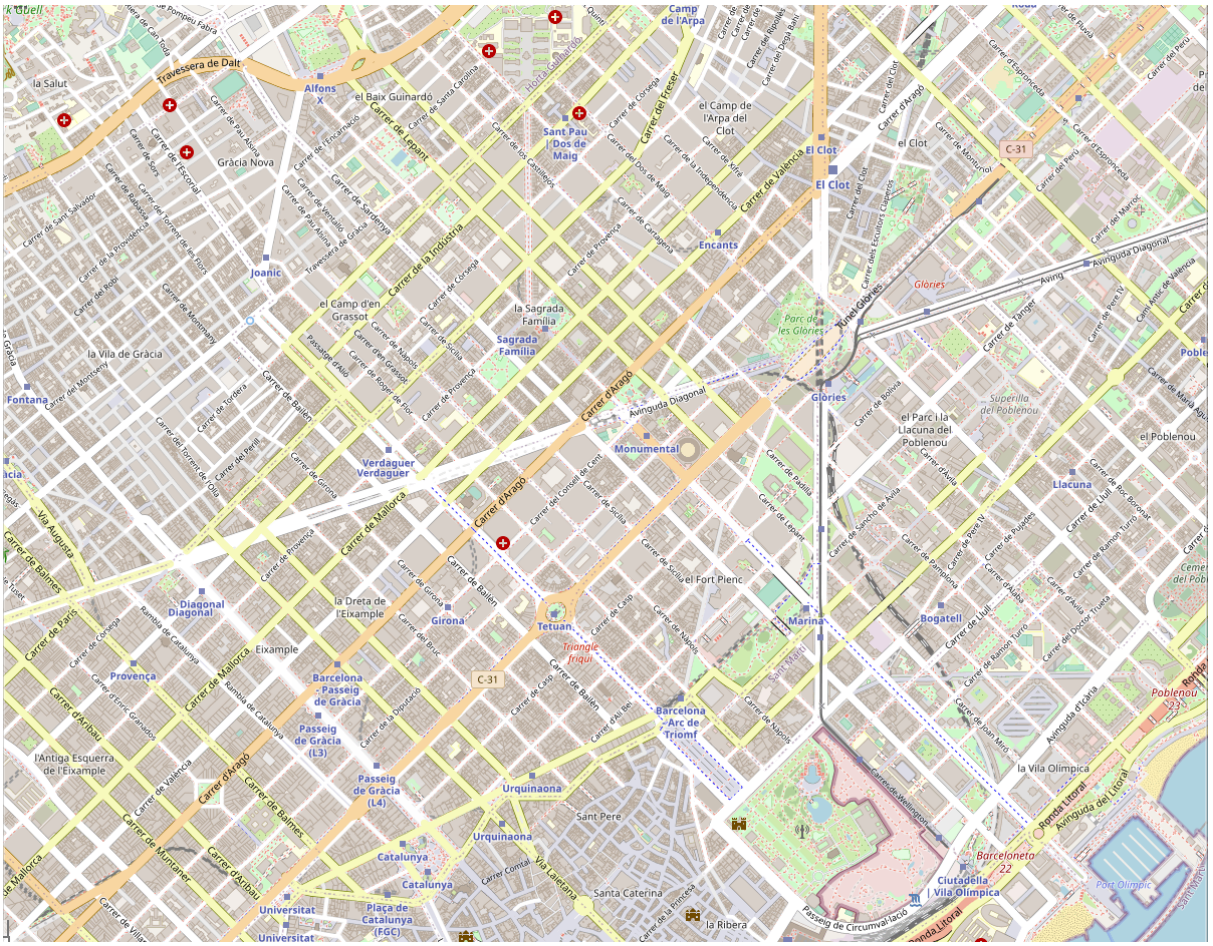


Coste Sumo: RAM(%) = 7,9 - 11,2 CPU(%) = 3 - 7,7

Barcelona-2.osm

Dimensiones del mapa:

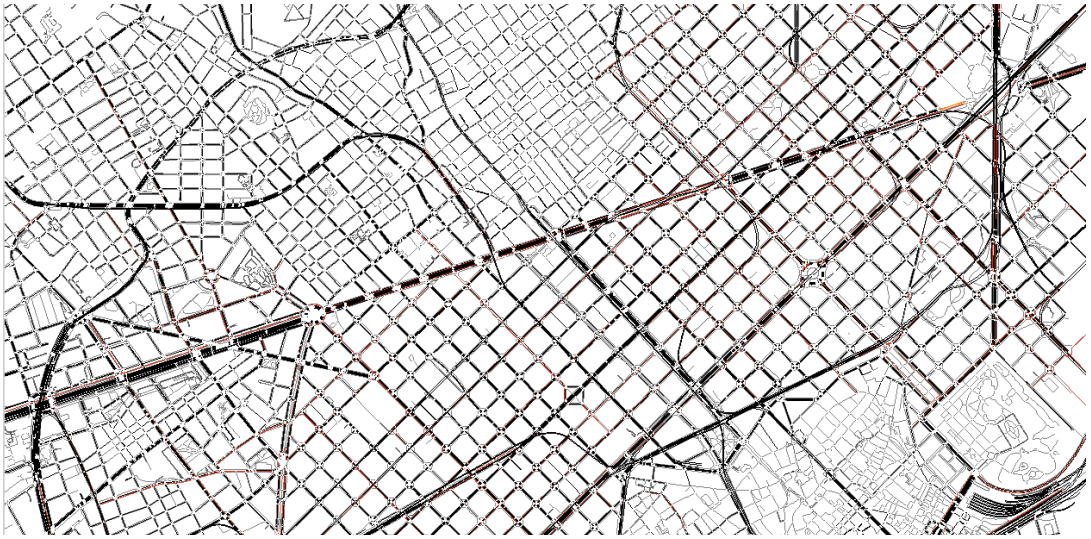
- 3.383 x 4.325 m de Longitud
- 14.631.475 m2 de Superficie



Tiempo de simulación: 10.000 ms

Número de coches: 500

SUMO



Tiempo real de simulación: 16.500 segundos si el tiempo de simulación terminara, pero , dado el bajo número de coches, la simulación tardaría mucho menos. Si el ritmo de los coches al recorrer las rutas se mantiene constante, calculo que terminaran sus recorridos en ~3100 segundos suponiendo que las rutas son de media tan largas como en simulaciones anteriores.

Gestión de bloqueos: 0 bloqueos. Sumo resuelve los bloqueos mediante teletransportaciones, en un mapa muy grande con solo 500 coches los bloqueos son inexistentes.

Configuración de la simulación: Tanto la simulación como el propio manejo del zoom y movimiento del mapa se ve increíblemente ralentizado, esto se traduce en un tiempo real de simulación horrible. La escala provoca que los iconos de los coches apenas se distingan lo que hace muy difícil de seguir la simulación sin acercarse tanto que pierdas la perspectiva de lo que estás viendo. También cabe recalcar que Sumo no permite acelerar la simulación de ninguna forma como hace por ejemplo Veins, por ello en 5 min tan solo el 10% de los coches han terminado.

nº Teletransportaciones: 0 teletransportaciones. 500 coches no son suficientes para que ocurran teletransportaciones, el mapa está completamente desértico, serían necesarios una cantidad de coches de otro orden de magnitud para que globalmente ocurrieran suficientes teletransportaciones como para ser algo significativo.
Claro que una simulación con tantos coches con un mapa así tardaría días en finalizar, no tiene mucho sentido si quiera intentarlo.

TABLA DE VARIACIÓN DEL TIEMPO DE SIMULACIÓN REAL (TIEMPO DE SIMULACIÓN y COCHES)

	50 coches	125 coches	300 coches	750 coches	1900 coches
250 ms	320s *1	320 s *1	320s *1	320s *1	320s *1
625 ms	988s *1	988s *1	988s *1	988s *1	988s *1
1560 ms	1516s *2	2400s *2	2736s *2	3128s *1	3128s *1
3900 ms	1516s *2	2400s *2	2736s *2	6396s *1	6396s *1
10000 ms	1516s *2	2400s *2	2736s *2	6396s *1	13128 s *2

*1 Estas simulaciones terminaron porque se llegó al tiempo de simulación estipulado, los coches no terminaron sus rutas.

*2 Estas simulaciones han sido detenidas sin terminar su ejecución debido a que a partir de cierto tiempo de simulación Sumo se ve increíblemente ralentizado, llegando incluso a los 4 frames por segundo (fps) según indica la interfaz del propio programa.Los tiempos de estas simulaciones son aproximados ya que cada ejecución podría tardar horas o incluso días en procesarse. En estas simulaciones también están incluidas aquellas simulaciones que no podemos saber si termina antes el tiempo de simulación o los coches, por tanto se utilizará el tiempo aproximado de finalización de los coches cuando este cálculo no quede claro.

Una observación bastante curiosa es el hecho de que para la prueba de 50 coches/ 1560 ms el resultado aproximado es de 1516 s, para la prueba de 125 coches / 1560 ms aumenta en un 58% (aumento similar al número de coches 50->125) pero el aumento de la prueba de 125 coches / 1560 ms a la de 300 coches / 1560 ms es de 14%. Esto es debido a que la aproximación del tiempo de simulación se ha realizado utilizando el ritmo al que los coches terminan su ruta, siendo la métrica segundos / coche.

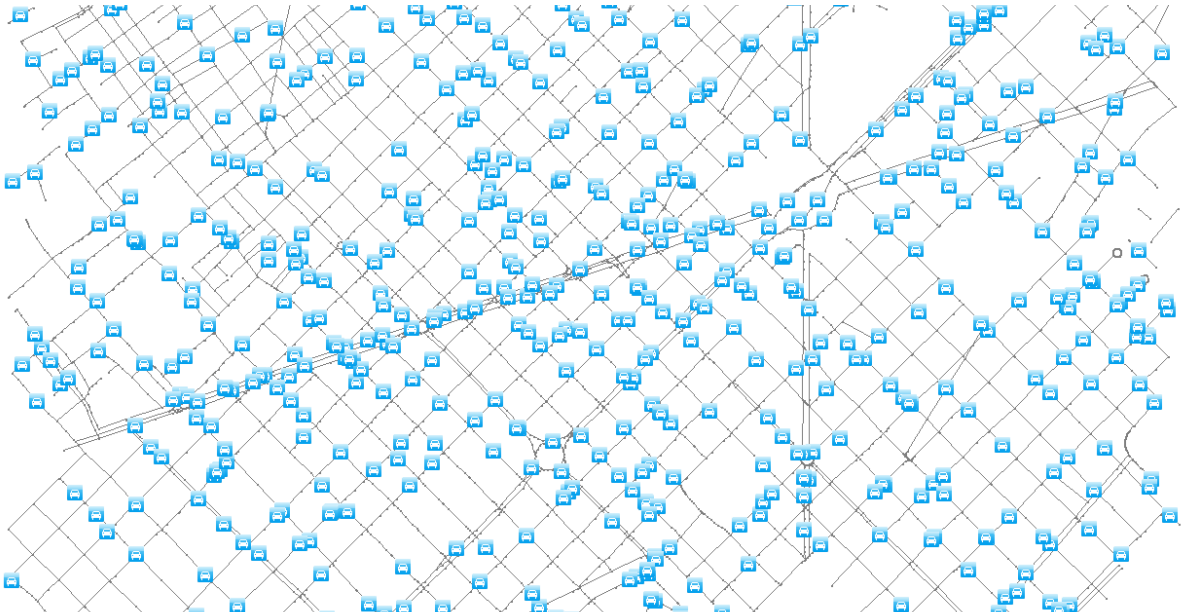
Esta métrica s/c (segundos / coche) se calcula tras finalizar al menos el 25% de los coches totales, por tanto cuanto mayor es el número de coches mayor será el máximo error cometido, llegando a errores considerables a partir de los 1000 coches.
Aun así es necesario aplicar esta aproximación para evitar ejecuciones excesivamente largas, además los datos siguen sirviendo para realizar comparaciones entre las métricas, entre simuladores o entre mapas.

Teniendo esto en cuenta para el resto de la tabla se ha utilizado el ratio s/c observado tras finalizar el 25% de los coches, esto sigue permitiendo ahorrar tiempo de ejecución (varias horas) conservando el error en un margen aceptable.

Como las rutas se generan aleatoriamente también se ha realizado una serie de pruebas con los valores por defecto (10.000 ms de tiempo de simulación y 500 coches) para observar el efecto que tienen estas rutas aleatorias. (Para estas pruebas también se ha aproximado el resultado usando el ratio s/c tras el 25% de los coches)

TIEMPO DE SIMULACIÓN DE 20 PRUEBAS (RUTAS ALEATORIAS)

MAVSIM

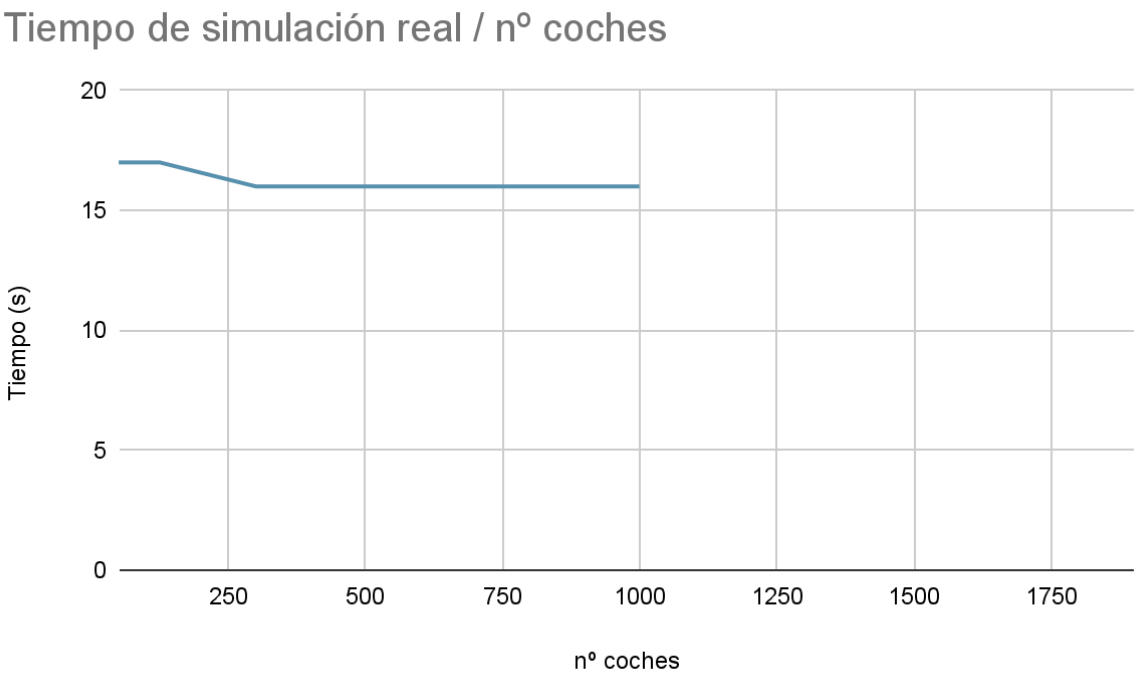


Tiempo real de simulación: 17 segundos. Al contrario que SUMO, no existen las teletransportaciones, la simulación acaba siempre cuando indica el tiempo de simulación configurado. Permite dos velocidades, normal y muy rápido. No especifica que velocidad es cada una (Paso en milisegundos). Para las pruebas realizadas se ha utilizado el modo de velocidad “muy rápido”.

Configuración de la simulación: La simulación comienza con todos los coches simultáneamente, no como Sumo que los va arrancando 1 a uno mientras se ejecuta la simulación. La situación de los coches y el mapa de la simulación se pueden observar y entender sin siquiera hacer ningún zoom ni mover el mapa, al contrario que en Sumo que era imposible siquiera ver los coches sin acercarte a la carretera en concreto.

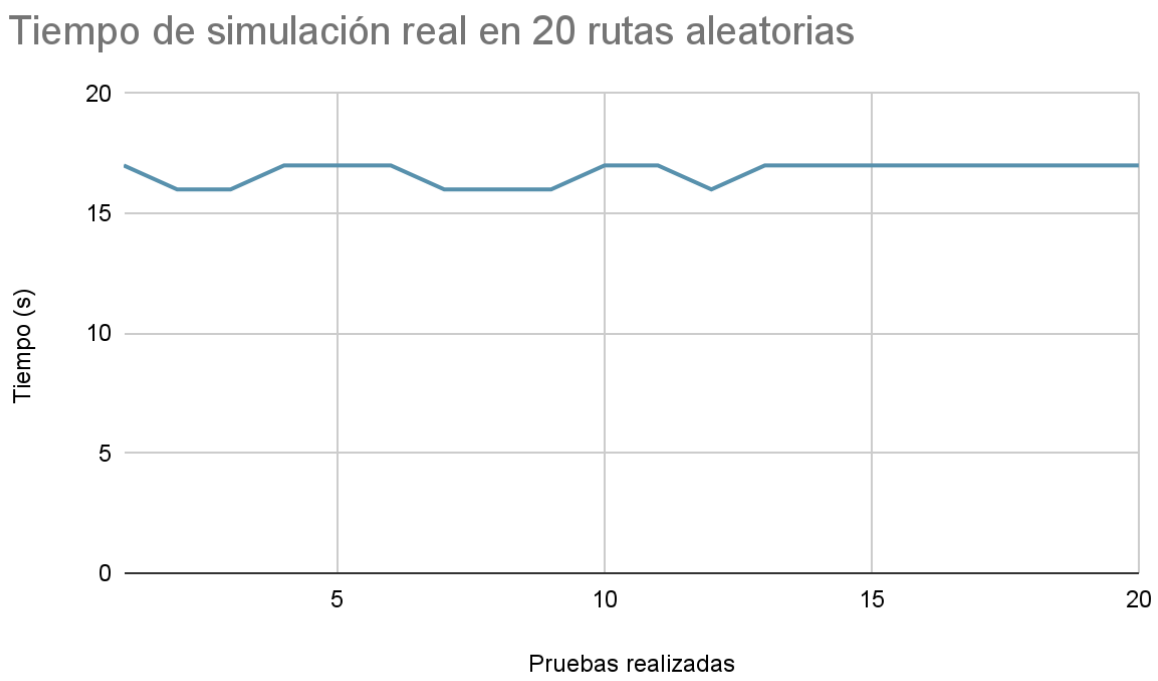
Aunque esta representación hace mucho mas fácil entender la situación del mapa de un vistazo, sí que puede llevar a una disminución en su rendimiento. Esto se puede observar ligeramente en las pruebas realizadas a continuación.

TIEMPO DE SIMULACIÓN REAL VARIACIÓN DEL Nº DE COCHES



De nuevo se obtienen unos resultados similares a otras pruebas de MAVSIM, tiempos prácticamente iguales debido al límite de iteraciones que tiene el simulador y pruebas sin terminar ya que no permite mas de 1000 coches. Adicionalmente se puede observar cómo a partir de un número de coches la simulación empieza a ralentizarse, esto es debido al tamaño del mapa y la gran cantidad de carreteras, como ya se ha mencionado.

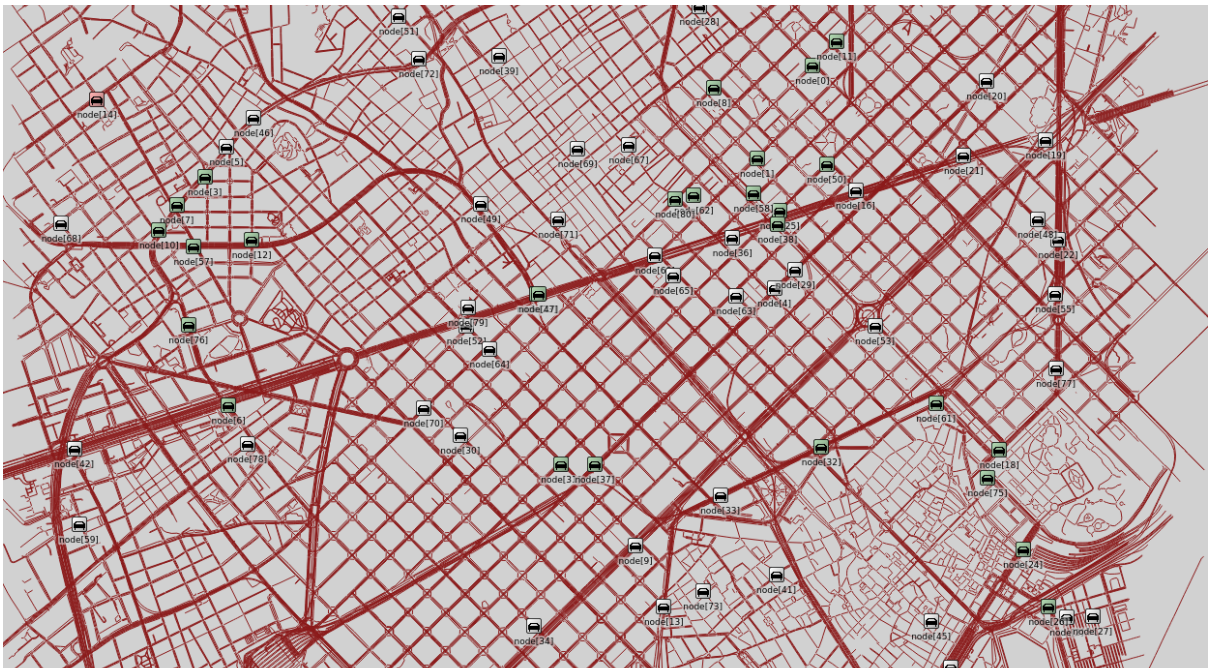
TIEMPO DE SIMULACIÓN DE 20 PRUEBAS (RUTAS ALEATORIAS)



Como en otras simulaciones de MAVSIM las pruebas de rutas aleatorias no aportan ninguna información, el tiempo obtenido es siempre el mismo, salvo en este caso que el tamaño del mapa y la cantidad de coches hace que el programa se ralentice y varían un poco los resultados.

En definitiva estas simulaciones consiguen justo lo contrario que las de Sumo, Sumo te representa todas las rutas de todos los coches y la simulación termina cuando la última ruta acaba o se llega al tiempo programado. Esto provoca que con un mapa tan grande y la gran ralentización que ocurre las simulaciones duren horas y sea necesario aproximar.
Por otro lado las simulaciones en MAVSIM siempre duran el tiempo que se le especifica con las iteraciones, aunque esto signifique que no se cumplan las rutas de los coches.

VEINS



Tiempo real de simulación: 1.335 segundos (con la velocidad intermedia o “Fast”). En este caso la simulación Veins se ralentiza enormemente, congelándose el programa durante hasta 10 segundos. Esto hace que una ejecución normal de 15 segundos tarde mas de 20 minutos, por ello, se va a aproximar la ejecución de la misma forma que la simulación Sumo. El ratio a utilizar será c/s pero se calculara una media de c/s de unas pocas ejecuciones y se utilizara esa medida para aproximar el resto de ejecuciones y aproximar si acaban antes los coches o el tiempo de simulación.

Configuración de la simulación: La simulación al ejecutarse tarda unos 20 segundos en generar el mapa, debido a su tamaño, tras comenzar la simulación se para durante aún mas segundos hasta que empiezan a aparecer los primeros coches. Estos aparecen muy lentamente y de golpe ya que la simulación está muy ralentizada.
La simulación transcurre muy lentamente debido al lag que se genera y es aún mas difícil seguir su ejecución que con su comportamiento normal. De nuevo cuando los coches terminan todas sus rutas la simulación termina, la gran diferencia con otras simulaciones es que transcurren varios minutos o incluso, en casos extremos, varias horas.

TABLA DE VARIACIÓN DEL TIEMPO DE SIMULACIÓN REAL (TIEMPO DE SIMULACIÓN y COCHES)

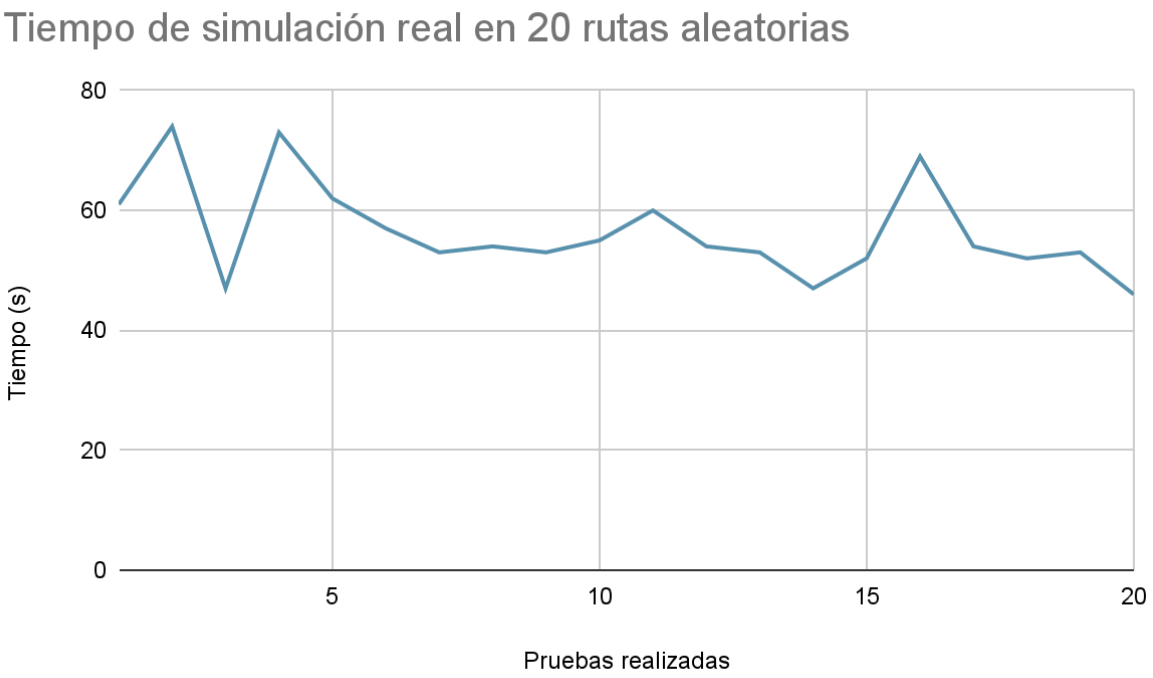
	50 coches	125 coches	300 coches	750 coches	1900 coches
250 s	44s	92s	176s	176s*	176s*
625 s	44s	92s	176s	1228s	1228s*
1560 s	44s	92s	176s	1228s	6246s*
3900 s	44s	92s	176s	1228s	12.821s
10000 s	44s	92s	176s	1228s	12.821s

*1 Estas simulaciones terminaron porque se llegó al tiempo de simulación estipulado, los coches no terminaron sus rutas.

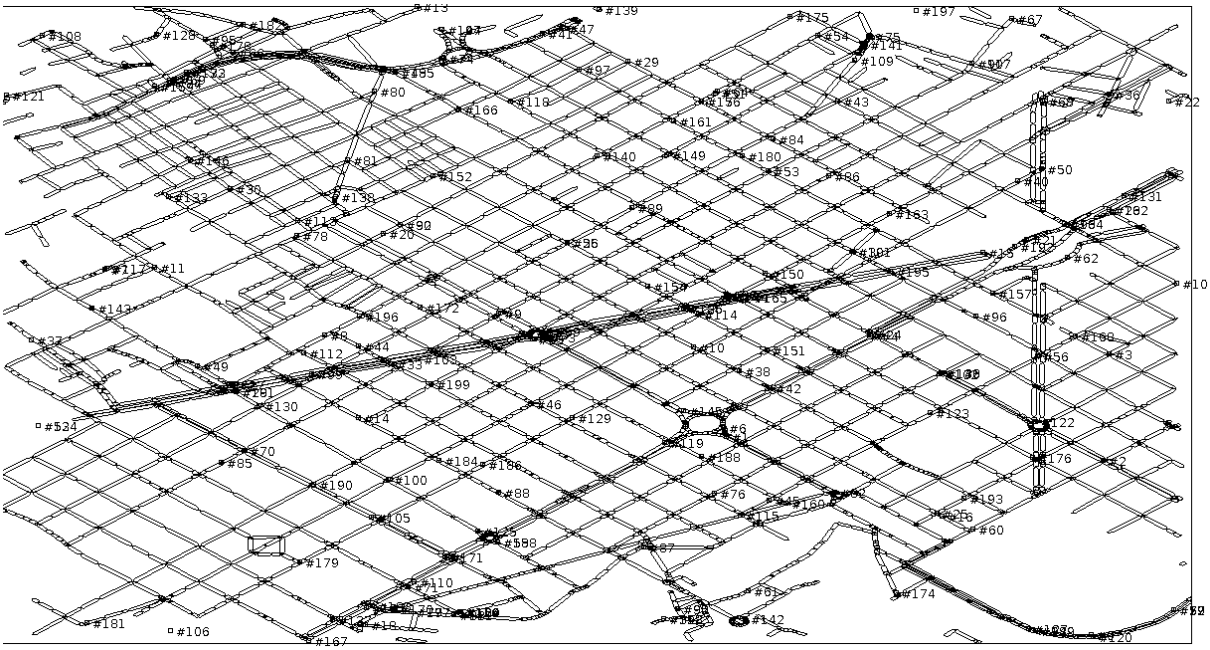
De nuevo estas pruebas de variación de parámetros reflejan algo similar a los observado con el simulador Sumo. Sin embargo los resultados de la tabla de arriba indican que la mayoría de ejecuciones terminan antes porque los coches acaban sus rutas por tiempo de simulación, justo al contrario que Sumo. Pero esto no significa que tenga mejor rendimiento, aunque las primeras columnas de la tabla son mejor en Veins que en Sumo, enseguida se observa un aumento significativo en los tiempos según aumenta el número de coches, pero las simulaciones siguen terminando antes del tiempo de simulación.

De esto podemos deducir que, aunque las simulaciones terminan antes en Veins que en Sumo, la ralentización que se produce por el número de coches y el tamaño del mapa provoca un mayor retraso en Veins que en Sumo. La deformación entre el tiempo de simulación y el tiempo real es mucho mayor en Veins, podemos decir que tiene mucho peor rendimiento.

TIEMPO DE SIMULACIÓN DE 20 PRUEBAS (RUTAS ALEATORIAS)



Estas rutas aleatorias han sido simuladas con 50 coches y 10.000 segundos como parámetros (con 500 coches se acerca a la media hora por prueba). El comportamiento mostrado en las pruebas con rutas aleatorias es cuanto menos impredecible, el tiempo de ejecución alcanza máximos de 74 y mínimos de 52. Se puede observar de un solo vistazo que la aleatoriedad de las rutas generadas afectan en gran medida al tiempo de ejecución final, esto es debido a que Veins comunica en todo momento los coches entre sí.
Esto provoca que los cruces y los conflictos que en estos se generan aumentan rápidamente las iteraciones entre los coches haciendo que una ruta con pocos cruces pueda tardar 50 segundos y otra con algún cruce mas tarde hasta un 50% mas.



VANETMOBISIM

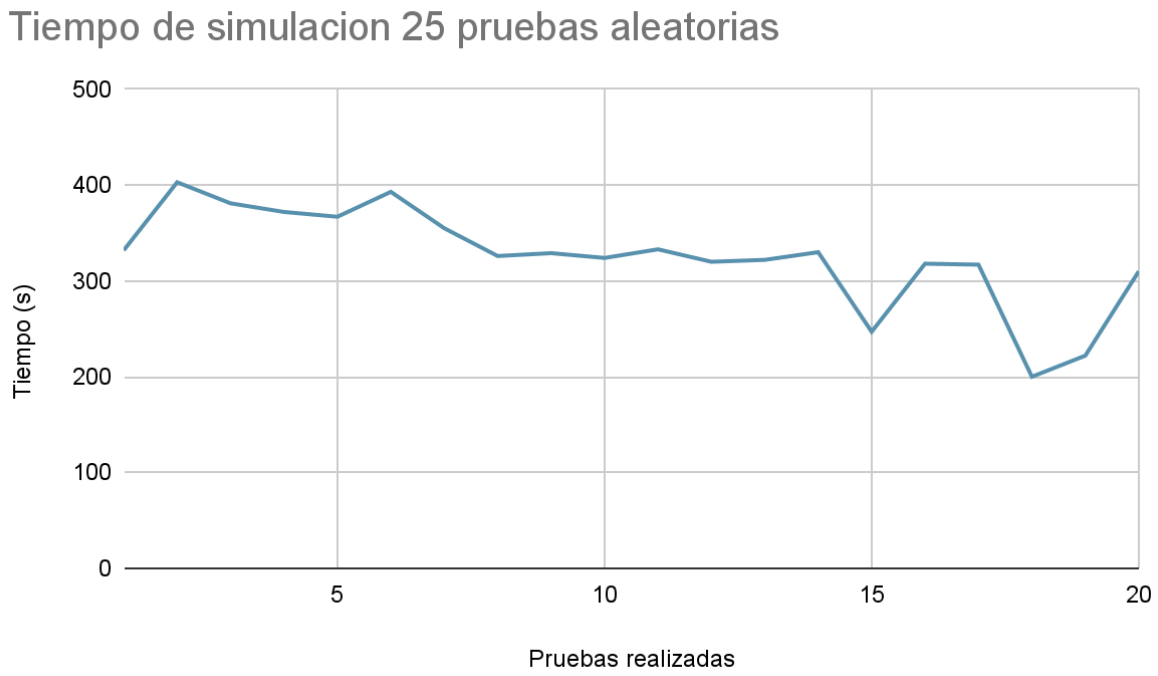
Tiempo real de simulación: 361 segundos. El programa termina cuando todos los coches terminan sus rutas, pero al contrario que otros simuladores VanetMobisim no permite establecer rutas

Configuración de la simulación: Los primeros 2 minutos de ejecución el simulador está simplemente iniciando la simulación. Tras esto los coches comienzan sus rutas, debido al tamaño del mapa los coches se mueven muy poco a poco ralentizando mucho su recorrido y aumentando la duración de sus rutas.

VARIACIÓN DEL TIEMPO DE SIMULACIÓN REAL

En cuanto a las pruebas de tiempo de simulación y número de coches el simulador no permite realizarlas de igual manera que el resto de programas, por tanto se va a optar por ejecutar VanetMobisim para 25 pruebas aleatorias para comprobar cómo de estable es el rendimiento que este muestra variando el tiempo de simulación. Sin embargo se van a descartar las pruebas fallidas (que devuelve el error ya mencionado en el apartado anterior).

Los parámetros para estas pruebas aleatorias serán 500 coches y 3900 ms (390 ms) de tiempo de simulación (hay que recordar que los tiempos de simulación reales para VanetMobisim están reducidos en un factor de 1/10 respecto al resto de simuladores, esto es debido al horrible rendimiento en tiempo que este muestra).

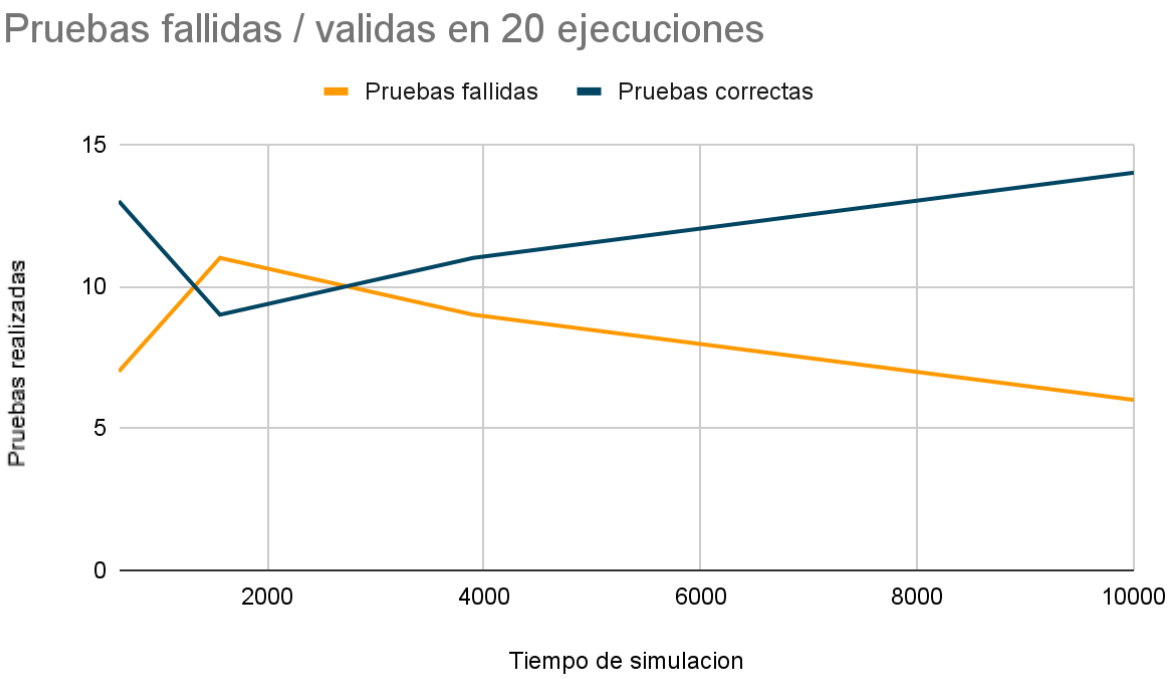


Durante la ejecución de estas pruebas se ha podido observar que aunque el tiempo de ejecución es muy elevado para un tiempo de simulación tan bajo, 390 ms reales, esto se debe a que tarda mas de 300 segundos en leer, crear y comenzar la simulación.
Las pruebas realizadas reflejan un comportamiento muy impredecible con subidas y bajadas bruscas y valores muy dispares. Esto se puede traducir en que el simulador VanetMobisim es muy sensible a las rutas aleatorias, pero como no sabemos cómo genera las rutas o cuántos coches utiliza no podemos afirmar nada.

Si tuviera que decir algo apostaría a que el número de coches varía entre un valor mínimo y máximo ya que las rutas aleatorias no producen un efecto tan irregular y contrario para cada simulación (como se ha observado en otros programas). Aunque también se puede deber a

que con un mapa tan grande no acaben sus rutas los coches y siempre se acabe por tiempo de simulación, lo que hace que el número de coches no sume tanto tiempo de simulacion.

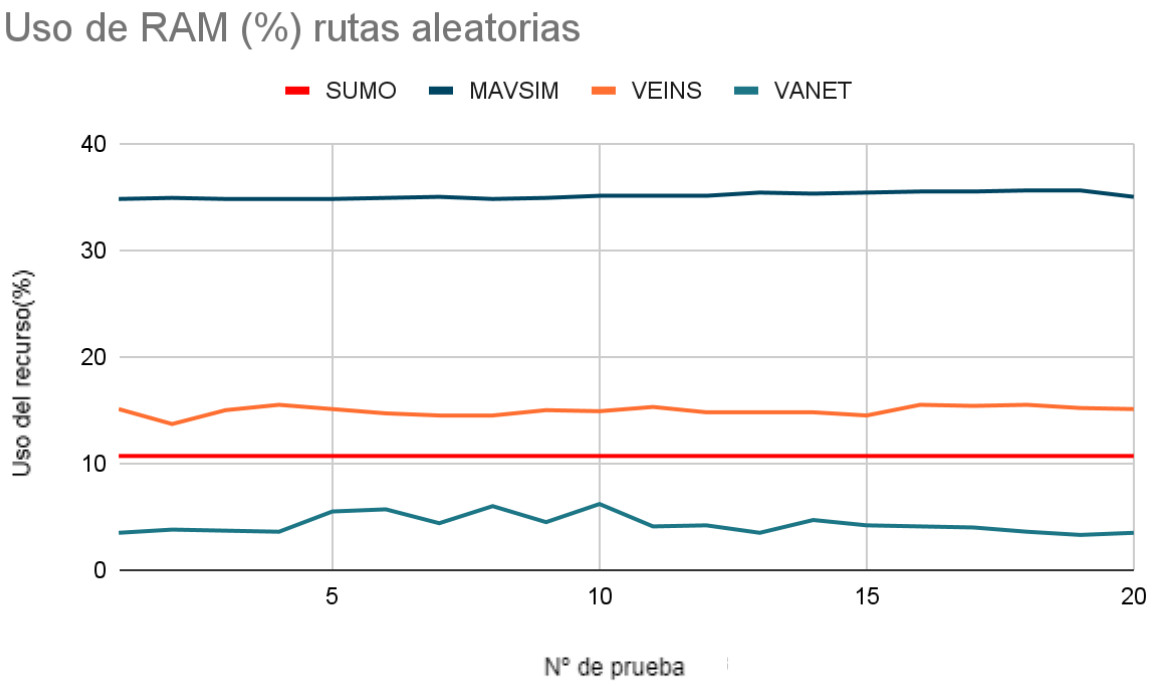
FALLOS DE SIMULACIÓN

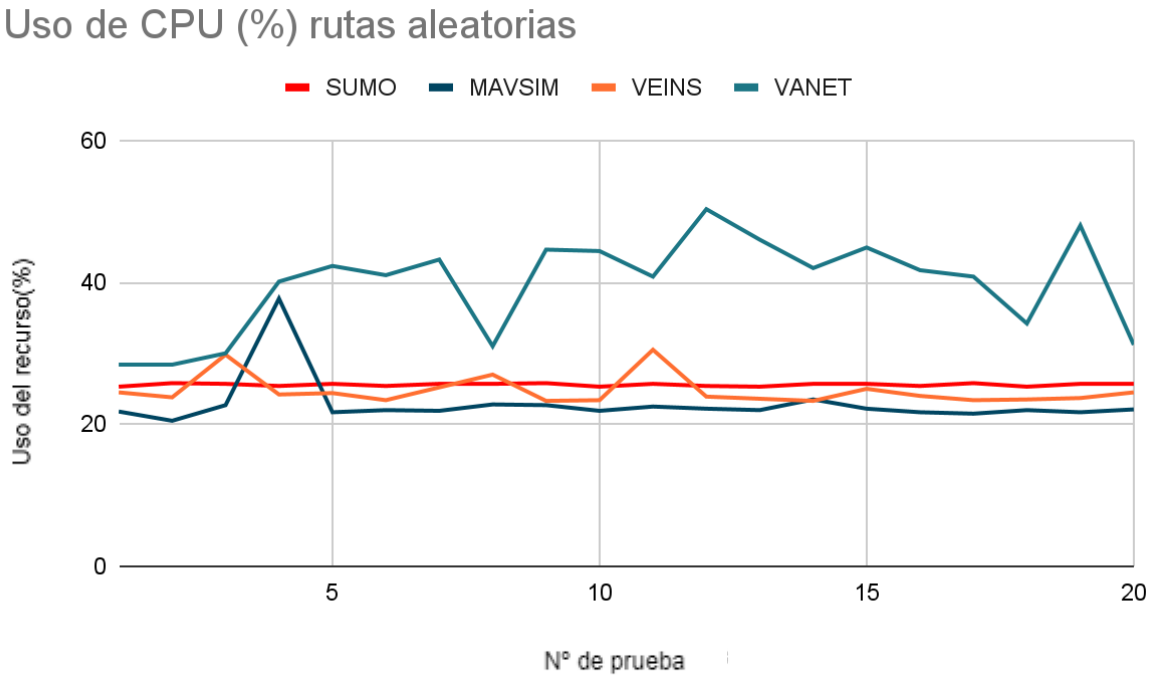


A primera vista no parece haber una correlación entre el tiempo de ejecución y que la ejecución acabe de forma correcta o por un error. Lo que da a pensar que deben ser algunas rutas en concreto que provoquen caminos sin salida y como los coches nunca terminan sus rutas el error es completamente aleatorio que ocurra antes o después del tiempo de simulación indicado.

RECURSOS UTILIZADOS

En esta sección se va a analizar el consumo de recursos de cada uno de los simuladores, la idea es representar la gráfica de recursos utilizados en 20 ejecuciones aleatorias para cada simulador y, de esta forma, comparar el rendimiento de cada programa y su adaptabilidad.



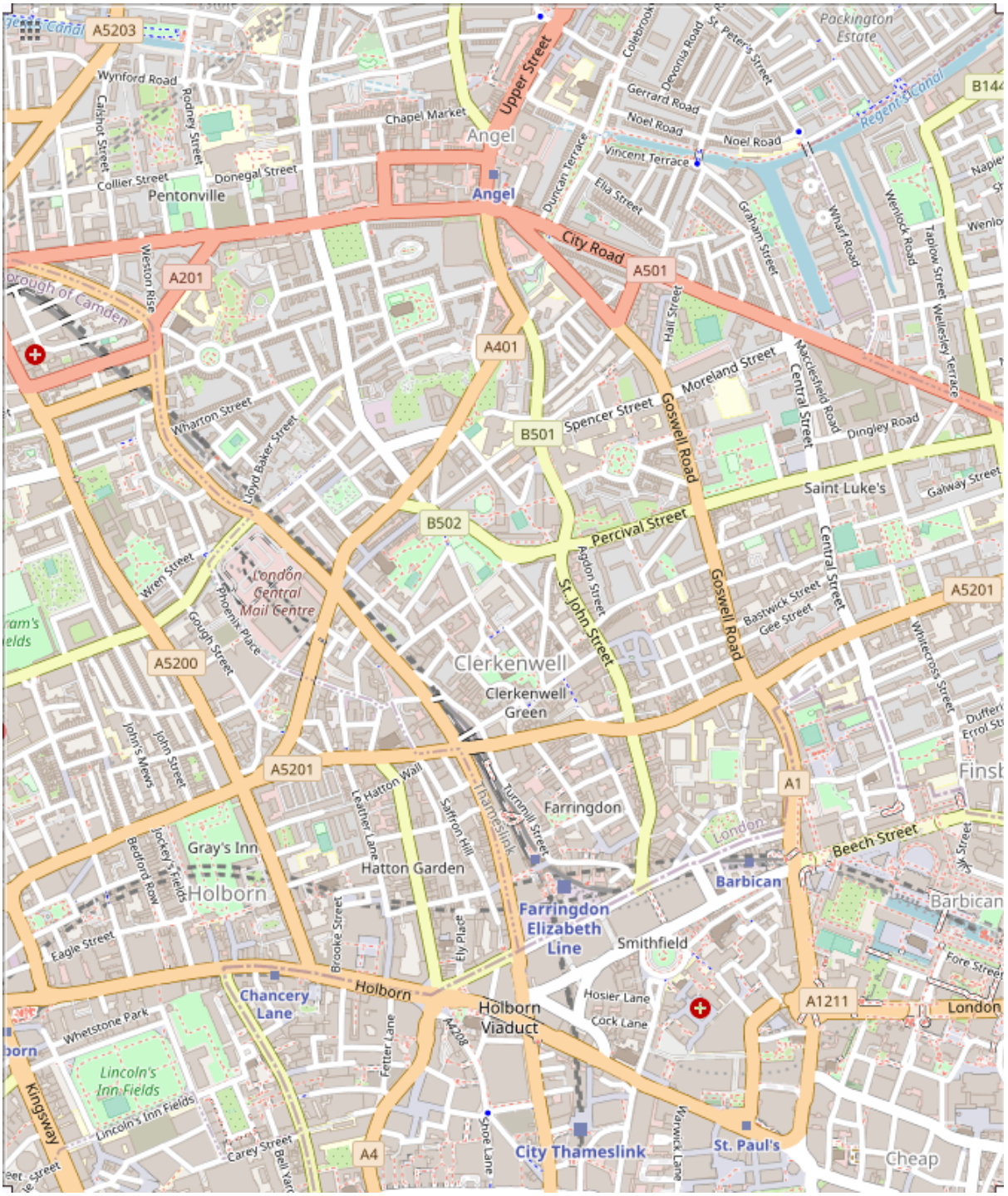


Coste Sumo: RAM(%) = 7,9 - 11,2 CPU(%) = 3 - 7,7

London.osm

Dimensiones del mapa:

- 2336 x 1992 m de Longitud
- 4.653.312 m2 de Superficie



Tiempo de simulación: 10.000 ms

Número de coches: 500

SUMO

Tiempo real de simulación: 314 segundos

Gestión de bloqueos: 0 bloqueos, el mapa es lo suficientemente grande como para que no ocurran bloqueos. Y aunque ocurran nunca serán suficientes para provocar el fin de la simulación.

Configuración de la simulación: Durante los primeros 2 minutos la simulación está generando los coches y sus rutas, tras esto, los coches recorren sus rutas hasta que se

cumple el tiempo de simulación introducido (recordar que los tiempos de ejecución están reducidos para VanetMobisim).

nº Teletransportaciones: 0. No hay teletransportaciones porque son la solución a los bloqueos, y no hay de estos.

TABLA DE VARIACIÓN DEL TIEMPO DE SIMULACIÓN REAL (TIEMPO DE SIMULACIÓN y COCHES)

	50 coches	125 coches	300 coches	750 coches	1900 coches
250 ms	77s*	77s*	77s*	77s*	77s*
625 ms	166s	172s	229*	229*	229*
1560 ms	166s	176s	340s	498*	498*
3900 ms	166s	176s	340s	696s	1201s*
10000 ms	166s	176s	340s	696s	1763s

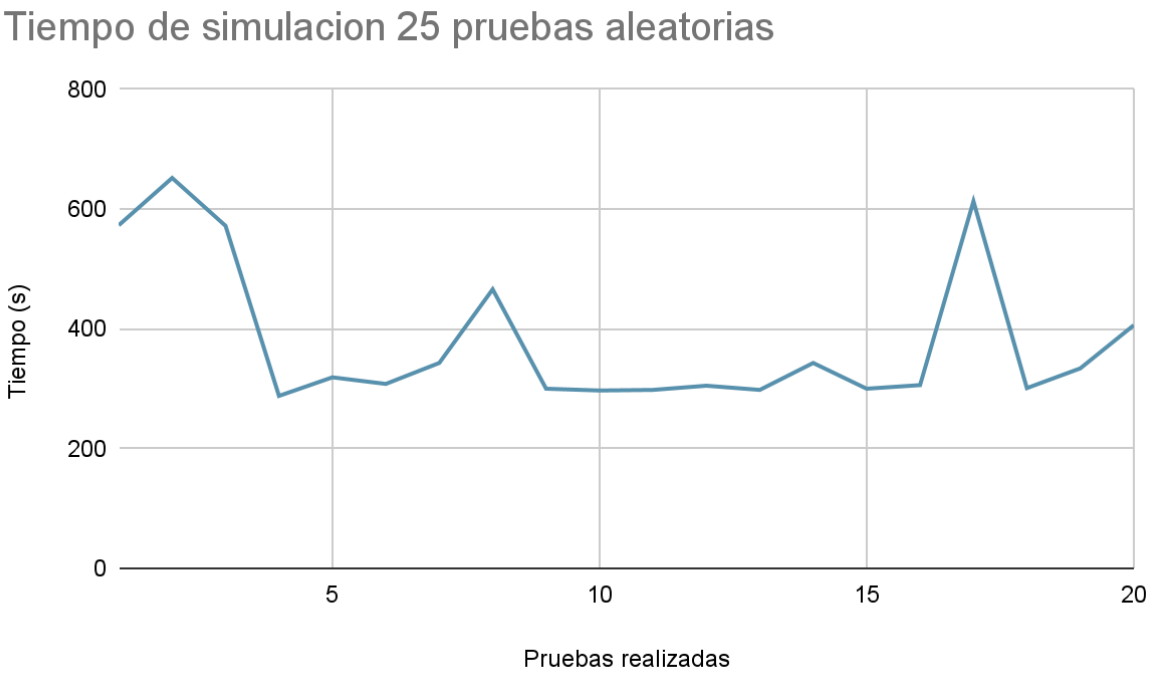
*Estas simulaciones terminaron porque se llegó al tiempo de simulación estipulado, los coches no terminaron sus rutas.

El comportamiento observado en los resultados de la tabla es el esperado para este simulador, por lo visto en otros mapas los tiempos de ejecución de Sumo suelen crecer mas o menos de forma lineal creando una especie de escalera con las ejecuciones terminadas por tiempo de simulación y por coches.

En el caso de este mapa no hay problema, pero en mapas anteriores como *Barcelona-2* el tamaño del mapa genera una ralentización que va escalando de forma exponencial según aumenta la memoria y el uso del cpu. Esto se traduce en ejecuciones de varias horas con +10.000 ms de tiempo de simulación.

TIEMPO DE SIMULACIÓN DE 20 PRUEBAS (RUTAS ALEATORIAS)

Como las rutas se generan aleatoriamente también se ha realizado una serie de pruebas con los valores por defecto (10.000 ms de tiempo de simulación y 500 coches) para observar el efecto que tienen estas rutas aleatorias.

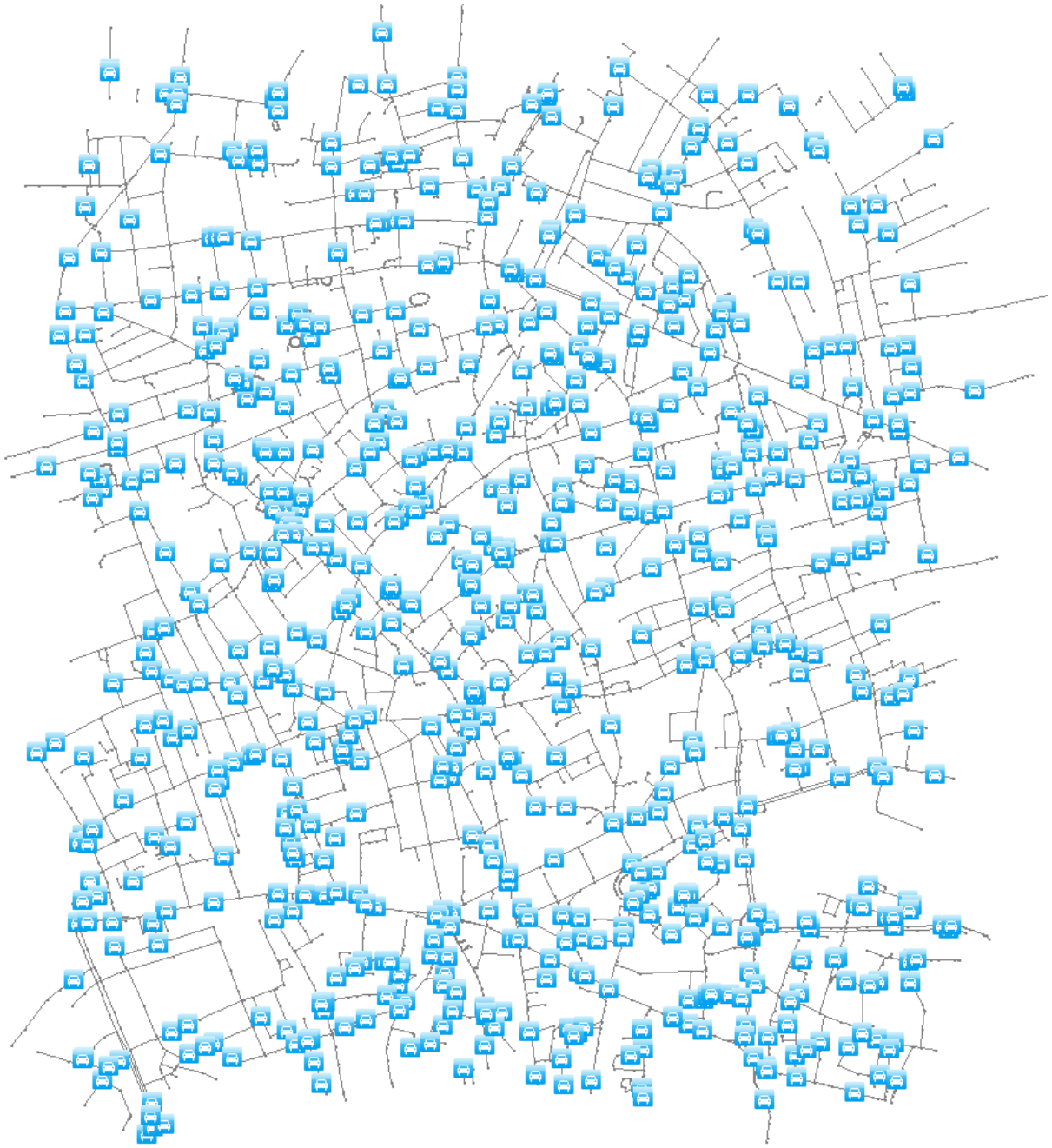


A lo largo de estas pruebas se han observado algunas teletransportaciones, pero nada que merezca la pena resaltar. La gráfica obtenida refleja unos resultados bastante dispares a los obtenidos con los mapas *Barcelona-2* y *Dublin*.

El gráfico muestra picos de mas de 600 s y mínimos de menos de 300 s, son unos valores muy extremos que muestran un comportamiento irregular, esto se debe a que en este caso las simulaciones si terminan porque los coches acaban sus rutas y no por tiempo de simulación. Gracias a que nunca se cumple el tiempo de simulación, las pruebas reflejan el rendimiento del simulador afectado únicamente por el RNG usado para generar las rutas.

Algunas simulaciones pueden mantenerse con el mismo número de coches durante varios segundos debido a que esas rutas tengan recorridos mayores de lo normal o también pueden terminar sus recorridos varios coches en 1 segundo porque sus rutas sean cortas. También puede influir en estos comportamientos en que momento es iniciado el coche ya que no es lo mismo empezar su recorrido al principio de la simulación que al final cuando hay menos coches y el programa está mas libre (mayor número de fps y por tanto velocidad en la ejecución).

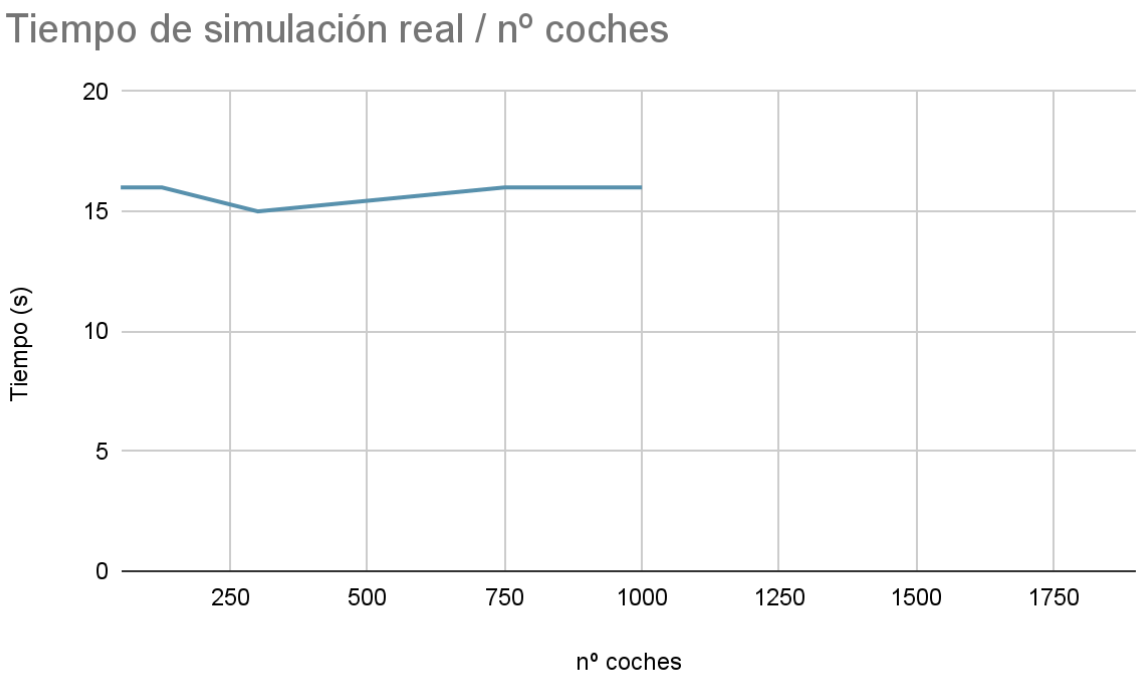
MAVSIM



Tiempo real de simulación:16 segundos. La simulación termina cuando el número de iteraciones llega al indicado (1000 iteraciones por defecto). A menos que el programa se ralentice por el uso excesivo de memoria o cpu, siempre se obtendrá el mismo tiempo de ejecución.

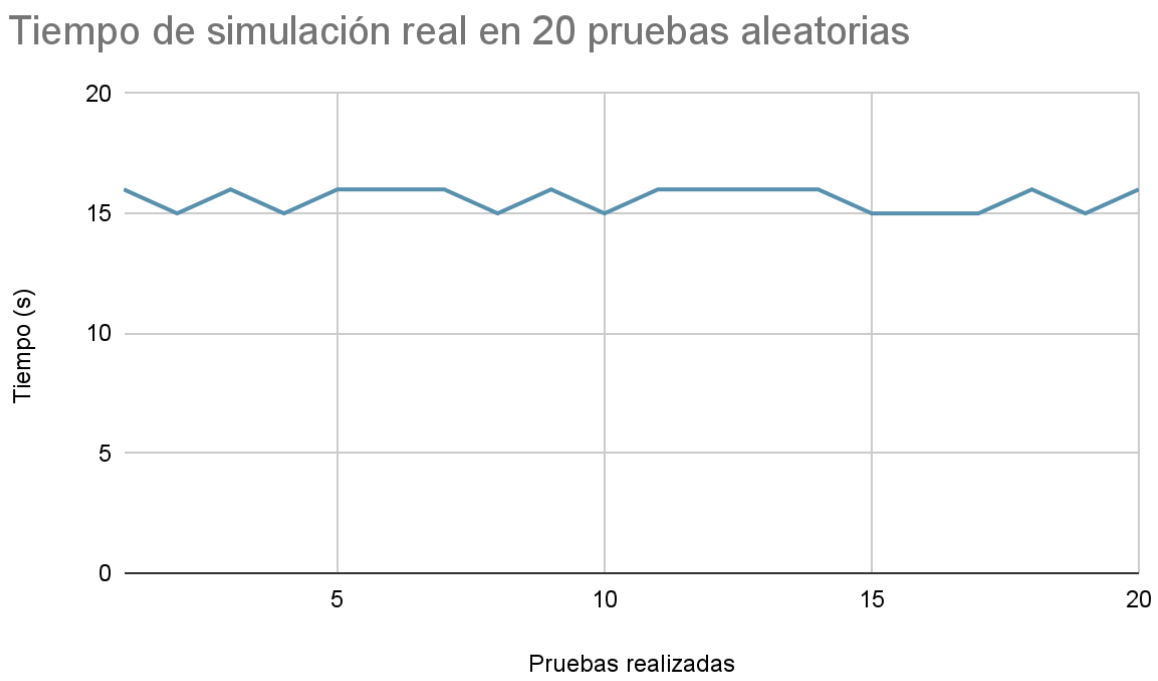
Configuración de la simulación: La simulación comienza generando los coches y sus rutas, tras esto los coches empiezan sus rutas hasta que se llega al máximo de iteraciones y termina la ejecución.

TIEMPO DE SIMULACIÓN REAL VARIACIÓN DEL Nº DE COCHES



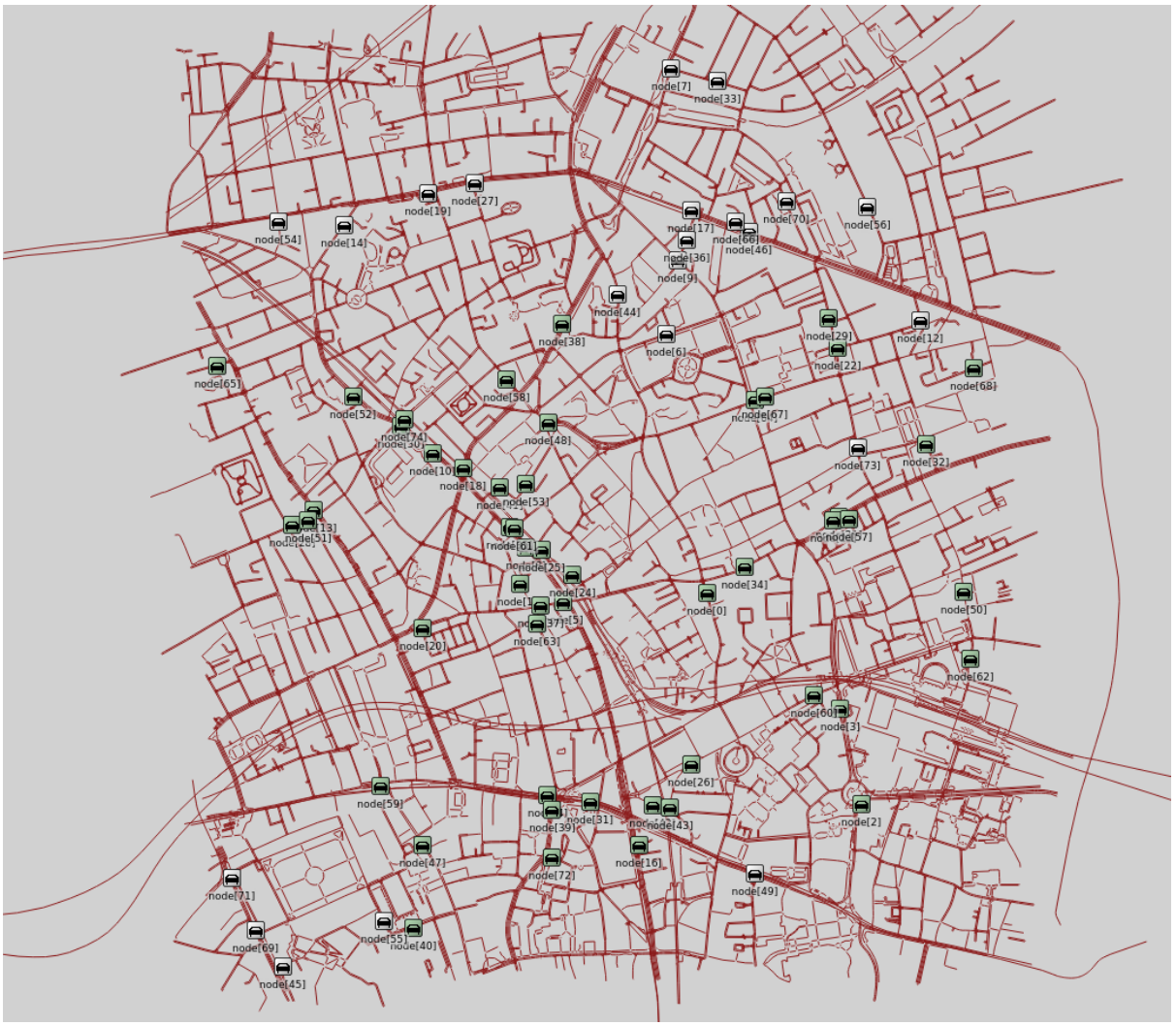
Los resultados obtenidos reflejan casi a la perfección el comportamiento de MAVSIM, como la ejecución depende completamente del número de iteraciones y su duración da igual que RNG y parámetros tenga la simulación. Las pequeñas irregularidades que aparecen son pequeñas diferencias causadas por variaciones en el rendimiento del programa, pero son decenas de segundo de diferencia, nada importante.

TIEMPO DE SIMULACIÓN DE 20 PRUEBAS (RUTAS ALEATORIAS)



De nuevo se observa el mismo comportamiento del programa, los datos se mantienen muy cerca de la marca de los 16 segundos y las variaciones causadas por las rutas aleatorias no son capaces de alejar los tiempos de simulación del definido por las iteraciones.

VEINS



Tiempo real de simulación: 93 segundos (con la velocidad intermedia). Los coches rápidamente inician y terminan sus rutas, la simulación termina cuando todos los coches han terminado su recorrido.

Configuración de la simulación:Cómo estamos utilizando la velocidad intermedia es difícil seguir la ejecución, los coches aparecen y desaparecen tras completar sus rutas. Además cuando se acumulan varios coches (100 o mas) el programa se ralentiza haciendo aún mas complicado entender la ejecución. Las otras velocidades son o demasiado rápidas o tan lentas que se pierde la noción de la simulación.

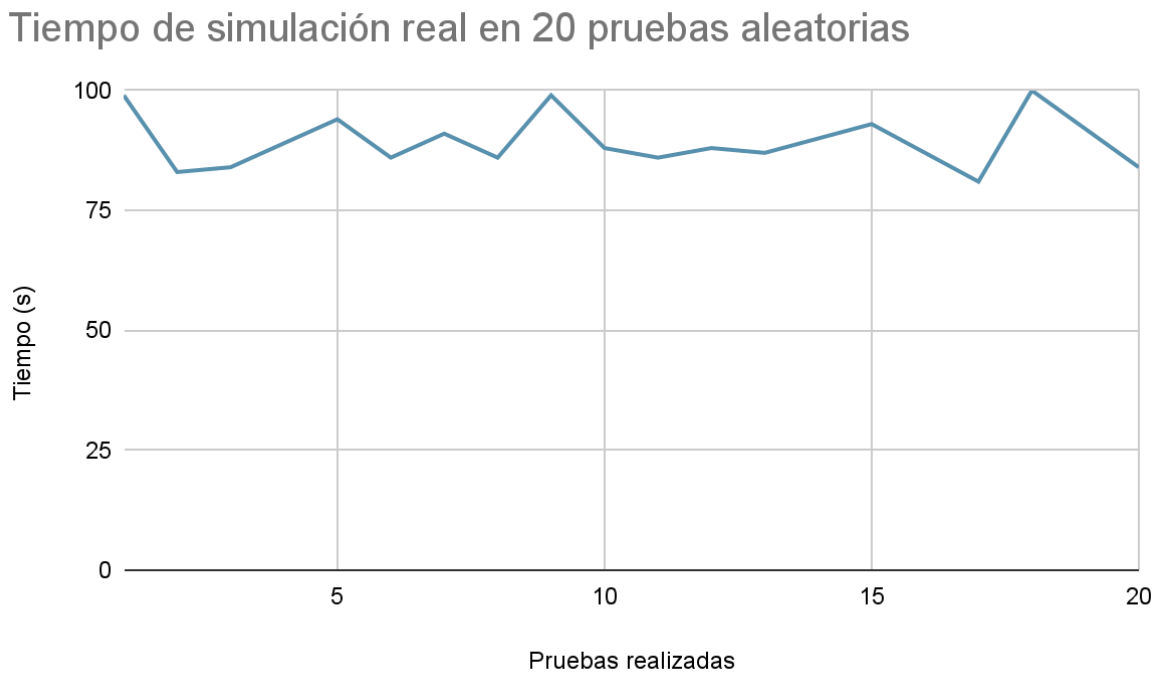
TABLA DE VARIACIÓN DEL TIEMPO DE SIMULACIÓN REAL (TIEMPO DE SIMULACIÓN y COCHES)

	50 coches	125 coches	300 coches	750 coches	1900 coches
250 s	8s	17s	30s	30s*	30s*
625 s	8s	17s	30s	109s*	109s*
1560 s	8s	17s	30s	143s	398s*
3900 s	8s	17s	30s	143s	587s
10000 s	8s	17s	30s	143s	587s

La tabla muestra los resultados obtenidos, en general el comportamiento de la simulación es el esperado, la gran mayoría de ejecuciones termina por número de coches y no por tiempo de simulación. Los tiempos escalan de forma mas o menos lineal lo que indica que el programa no sufre ninguna ralentización apreciable, al contrario que en el mapa *Barcelona-2*.

Si comparamos los datos obtenidos con otros mapas simulados en Veins se puede observar una reducción en los tiempos de ejecución bastante visible y además estos tiempos no escalan tan rápido a cifras tan elevadas.

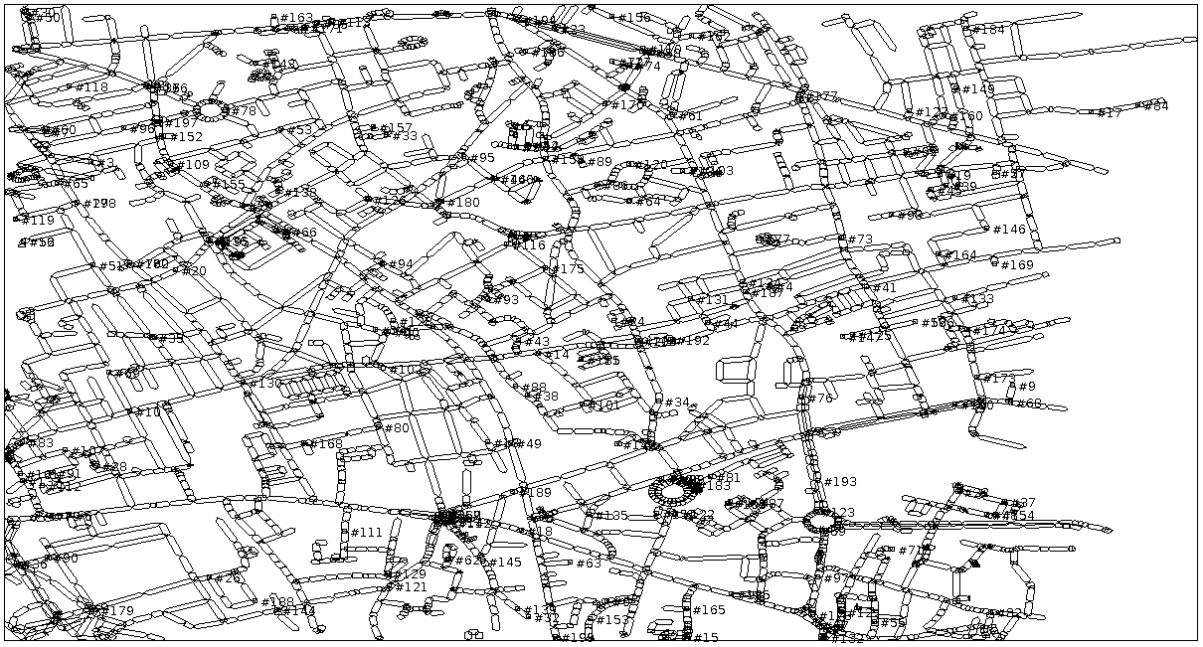
TIEMPO DE SIMULACIÓN DE 20 PRUEBAS (RUTAS ALEATORIAS)



De nuevo las pruebas de rutas aleatorias devuelven los resultados esperados, las ejecuciones vuelven a terminar por coches, esperable ya que para 500 coches este mapa tiene de sobra con 10.000 ms de simulación.

También se observa que los tiempos obtenidos son mas elevados que las simulaciones de *Barcelona-2*, lo que es normal por la diferencia del tiempo de simulación configurado, y que las simulaciones de *Dublin*. Esta diferencia entre *Dublin* y *London* está basada en la diferencia de tamaño del mapa lo que provoca que las rutas puedan ser mas variadas y el tiempo que tardan los coches en recorrer sus rutas pueden variar mas. En un mapa mas pequeño se puede complicar menos cada recorrido.

VANETMOBISIM



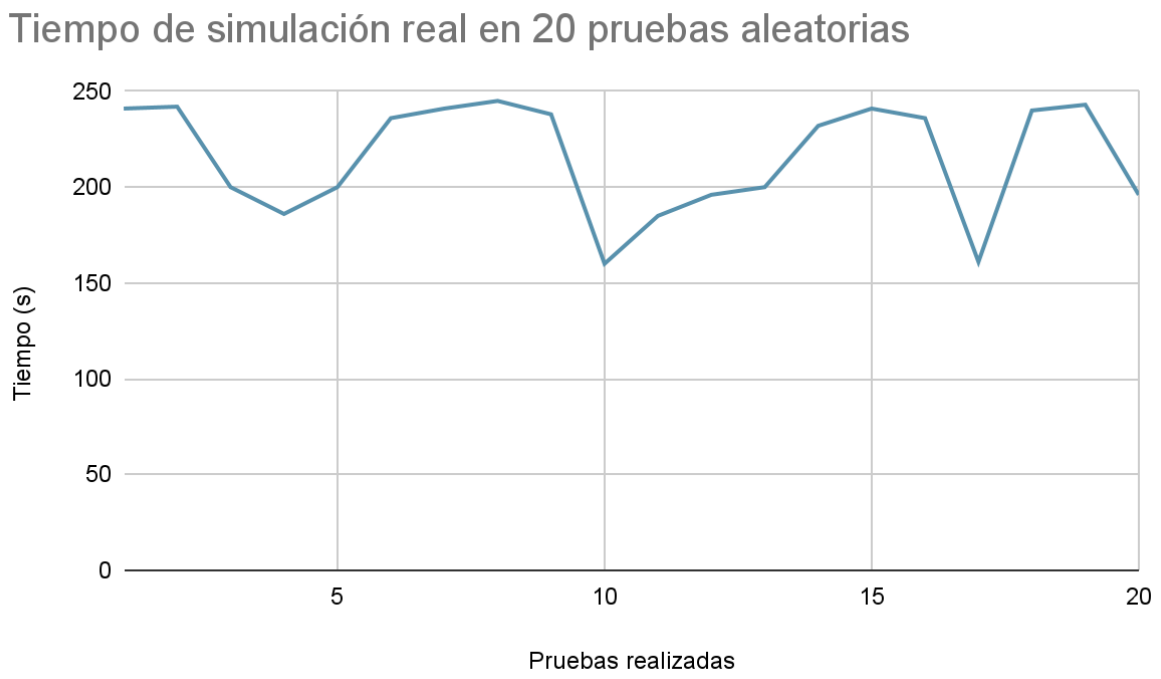
Tiempo real de simulación: 260 segundos. El programa termina cuando todos los coches terminan sus rutas, en este caso los coches no terminan y la simulación acaba por llegar al tiempo de simulación. En este caso este tiempo es de 1.000 ms en vez de 10.000 ms ya que aun asi una simulación tarda casi 5 minutos.

Configuración de la simulación: La principal causa de que una simulación de 1.000 ms tenga una duración de casi 5 minutos es que el programa tarda mas de 3 en generar los coches y comenzar la simulación. Sin contar los 10-20 segundos que tarda en abrirse el programa.

Aun si no tenemos en cuenta esta carga inicial, 2 minutos para mover 500 coches durante 1.000 ms de simulación se aleja mucho del rendimiento mostrado por los otros simuladores. Que en un tiempo mucho mayor (10.000 ms) Sumo o Veins ejecuten todas las rutas en un tiempo similar al que VanetMobisim simula 1.000 ms es un rendimiento nefasto y para nada útil en un escenario real.

VARIACIÓN DEL TIEMPO DE SIMULACIÓN REAL

El simulador VanetMobisim no permite asignar número de coches o rutas de los mismos, por tanto las pruebas serán simplemente con rutas aleatorias. Aun así, se analizará el porcentaje de ejecuciones terminadas antes de tiempo por fallos del simulador en siguientes pruebas.

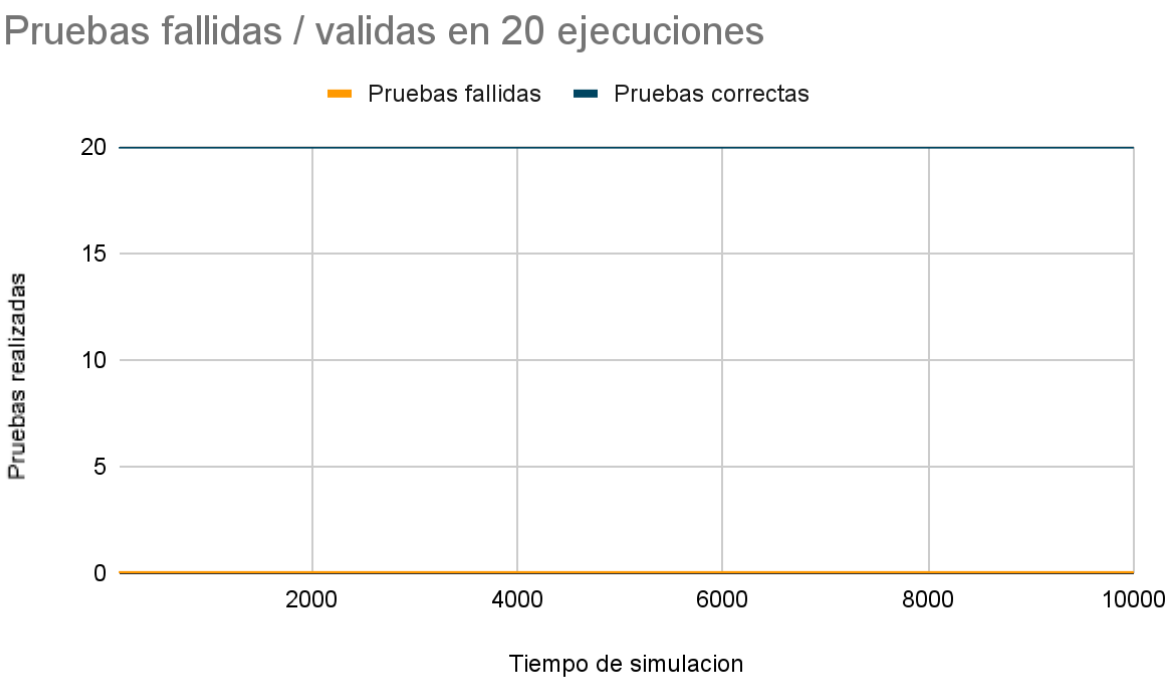


De nuevo se puede observar de las pruebas la pausa inicial para cargar la simulación que se ha comentado. En este caso la carga de los coches y rutas se toma mas de 2 minutos de simulación, cuando la simulación dura de media aproximadamente 215 segundos (3 minutos y medio).

En cuanto a los resultados en si, cumplen con lo esperable, un comportamiento nada regular, sobre todo cuando la simulación no está terminando por el tiempo de simulación. Esto indica que parando en el mismo momento de la simulación (390 ms) la ejecución ha variado mucho en su ejecución, una de los motivos para este comportamiento podría ser que VanetMobisim no sea muy tolerante a las intersecciones y que las rutas aleatorias hagan variar mucho el tiempo de ejecución, según las intersecciones que estas contengan.

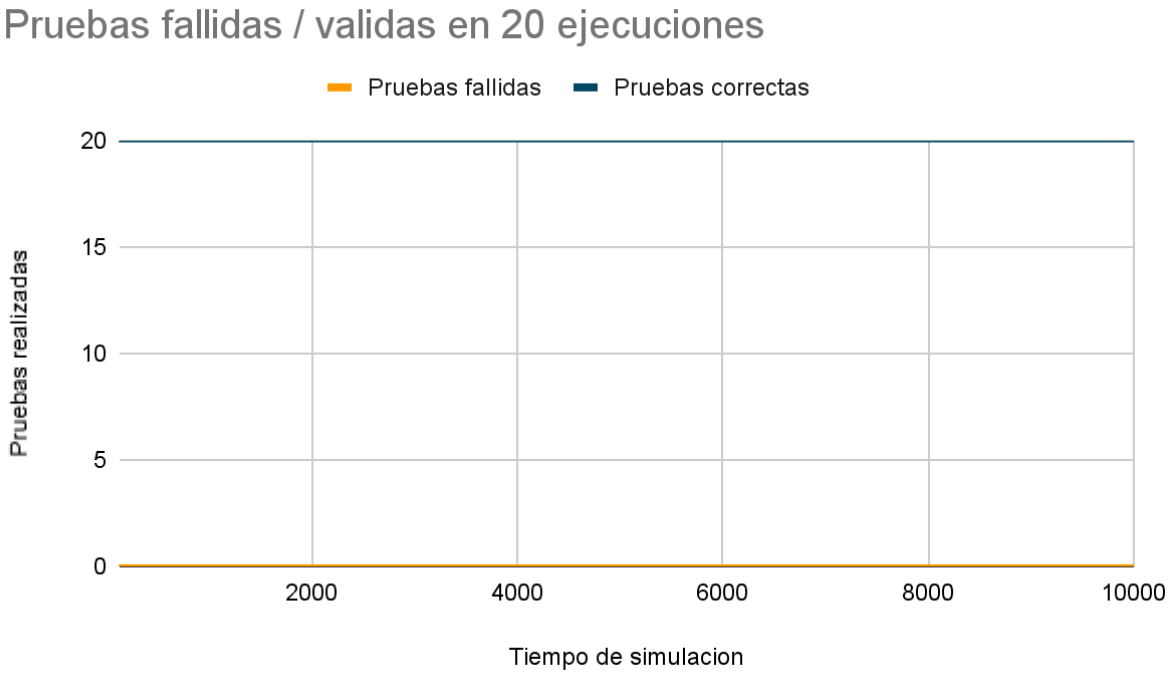
Pero tampoco podemos asegurar el motivo de este comportamiento porque no podemos saber cómo genera las rutas el simulador, no aparece nada sobre este aspecto en la ayuda del programa ni en su documentación. Si que aparece el algoritmo de camino mas corto Randomized Dijikstra pero esto solo nos indica cómo se resuelven las intersecciones, no como se generan las rutas.

FALLOS DE SIMULACIÓN



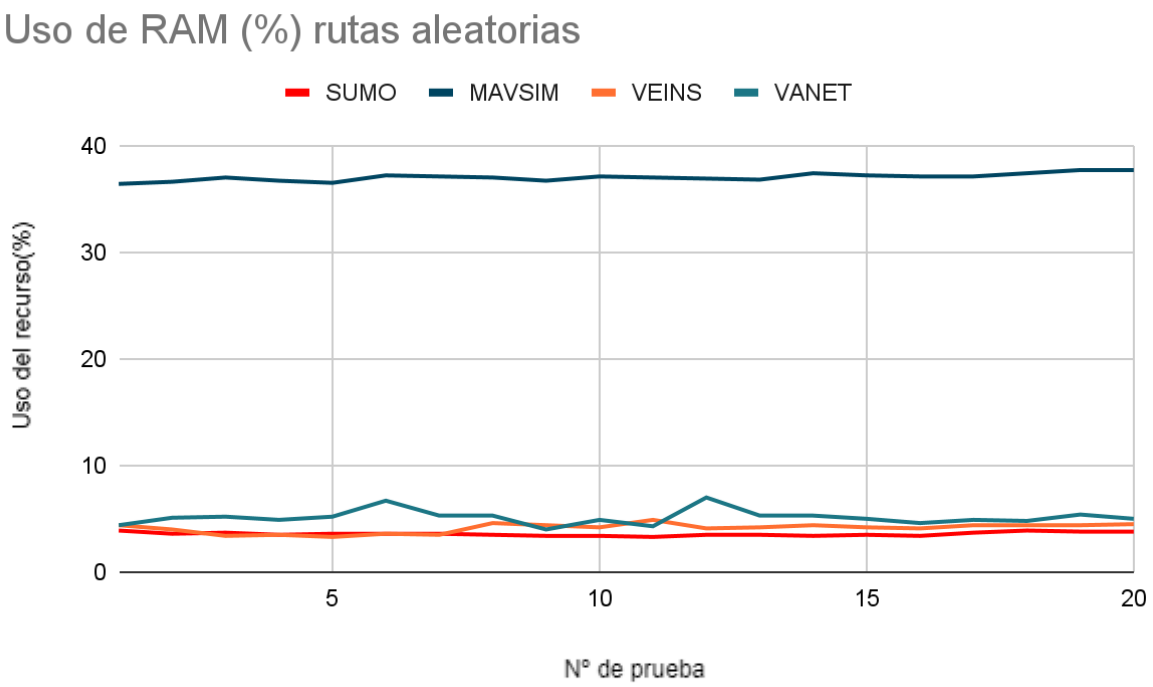
En este caso en ninguna ejecución ha surgido ningún error, todas las ejecuciones se inician sin fallos y todas finalizan dentro del tiempo de ejecución que se espera sin mostrar ningún error. Los fallos al iniciar no se entiende porqué ocurre (el simulador no da ninguna información, a veces falla al procesar la simulación y a veces no), como solo ocurre con el mapa *Barcelona-2*, que es de un tamaño gigantesco comparado con los otros mapas.
Por ello podemos decir que VanetMobisim tiene problemas al procesar mapas muy grandes, a parte del considerable efecto que estos mapas tienen en el rendimiento, pueden producirse errores al leer el mapa o al generar las rutas de estos mapas.

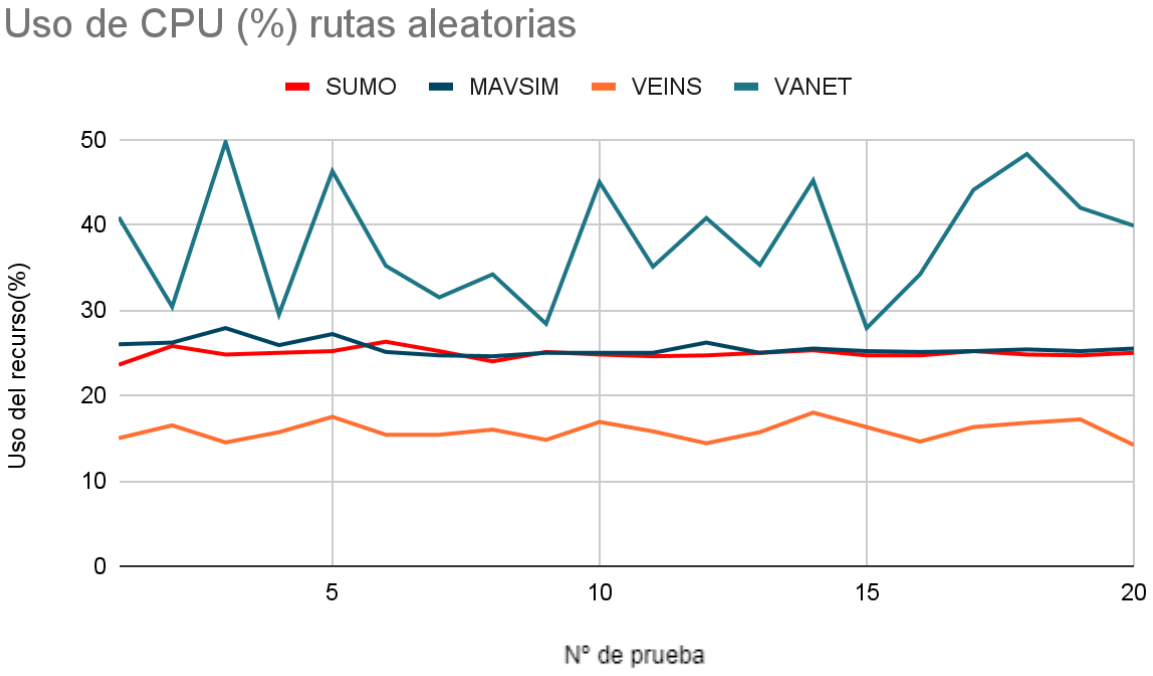
El otro tipo de errores es el provocado cuando el simulador llega a un punto donde la ruta que está siguiendo no tienen un posible siguiente destino. Esto comúnmente se llama bloqueo, y los simuladores de tráfico Vanet normalmente tratan estos bloqueos de diferentes formas (en el caso de Sumo lo soluciona con teletransportaciones). El simulador VanetMobisim no los soluciona de ninguna forma, ni siquiera plantea su existencia, cuando encuentra un bloqueo lo trata como un fallo en el mapa y detiene la simulación sin previo aviso.


Pruebas fallidas / validas en 20 ejecuciones
Pruebas fallidas
Pruebas correctas
20
15
10
5
0
Pruebas realizadas
2000
4000
6000
8000
10000
Tiempo de simulacion

RECURSOS UTILIZADOS

En esta sección se va a analizar el consumo de recursos de cada uno de los simuladores, la idea es representar la gráfica de recursos utilizados en 20 ejecuciones aleatorias para cada simulador y, de esta forma, comparar el rendimiento de cada programa y su adaptabilidad.





Coste Sumo: RAM(%) = 7,9 - 11,2 CPU(%) = 3 - 7,7

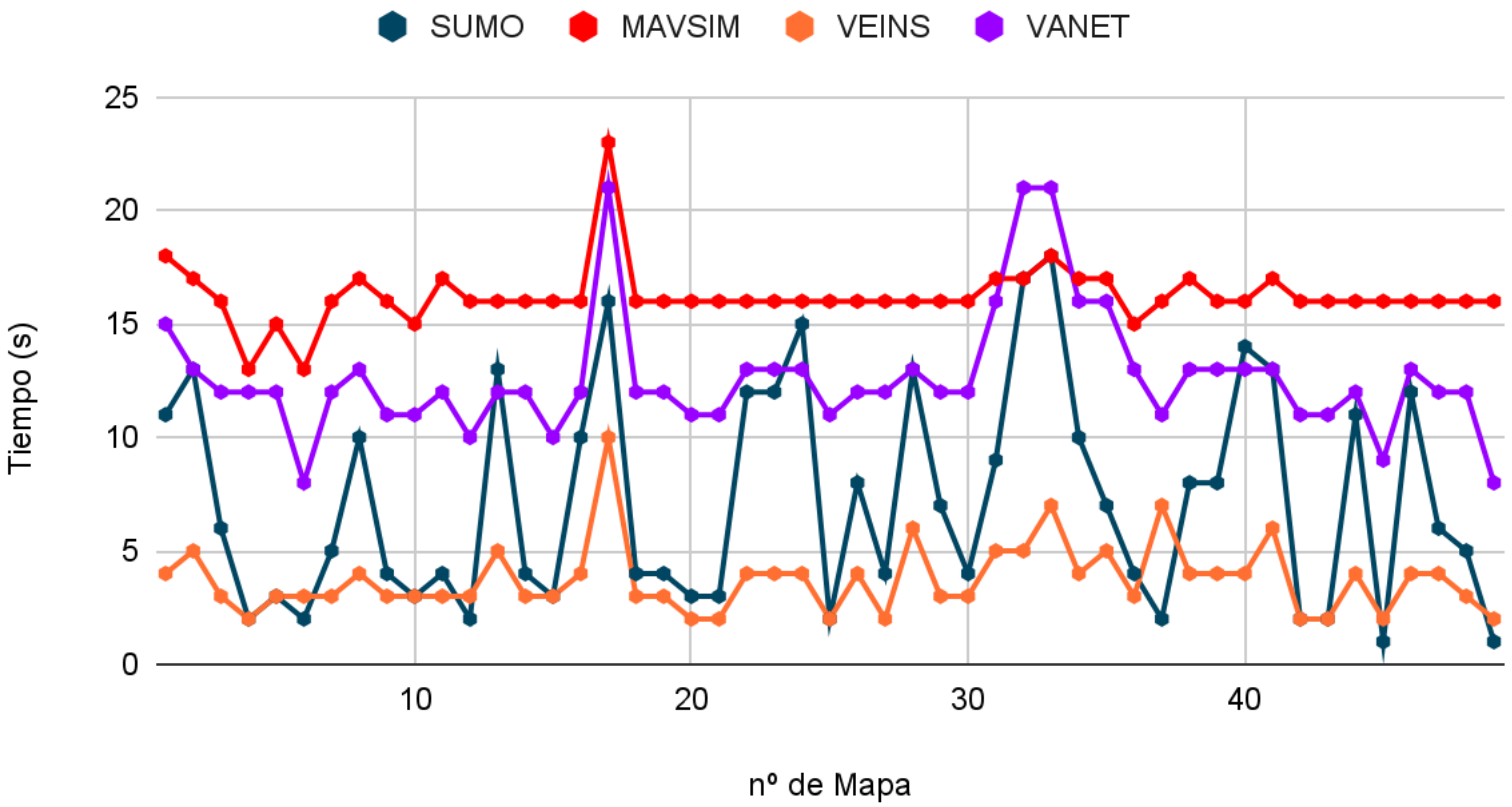
Coste disco Veins extra (%) : 20-30

RESULTADOS MAPAS ALEATORIOS

Para las pruebas de mapas aleatorios se han establecido ciertos parámetros por defecto para la generación de estas simulaciones aleatorias. El número de nodos será un número aleatorio entre 10 y 100, el número de coches será 125 y el tiempo de simulación será de 10.000 ms en el caso de SUMO, MAVSIM y VANET (1.000 MS) y 10.000 s en el caso de VEINS.

TIEMPO DE SIMULACIÓN MAPAS ALEATORIOS

Estas pruebas tienen el objetivo de comparar el rendimiento en tiempo y el efecto de cada mapa aleatorio en los 4 simuladores. De esta forma se puede observar de un vistazo cuál es el rendimiento de cada simulador para una gran variedad de mapas.



IMÁGENES DE LOS MAPAS ALEATORIOS