**Simulación del tráfico en carreteras transitadas**

Gabriel Martínez Fresneda

Sergio López de la Flor

1. **Introducción y objetivo**

La idea de este proyecto surge debido al gran número de atascos y tráfico que se producen en muchos puntos estratégicos de cualquier ciudad grande en horas transitadas.

El objetivo de este proyecto es poder reducir el tiempo de espera de los coches en un cruce de semáforos gracias al control automático del estado de estos mismos, de tal manera que ellos sean capaces por si solo de ir modificándose según el tráfico que se acumule en la carretera.

Gracias a todos los avances en machine learning de hoy en día, podemos desarrollar este proyecto con una técnica llamada q-learning o también conocida como aprendizaje automático.

Como en todo cualquier proyecto, antes de realizarse a escala real, se debe realizar una primera simulación. Para eso, hemos usado SUMO. SUMO SUMO es nuestro entorno de simulación de tráfico en carreteras con una amplia librería en la que podremos realizar nuestro siguiente proyecto.

A continuación mostraremos los pasos seguidos para desarrollar la red y realizar nuestra simulación.

1. **Generación de la red**

En esta primera parte del proyecto hemos desarrollado una red simple en SUMO. Nuestra red se basa en una simple intersección entre dos cruces, donde cada cruce tiene camino de ida y vuelta.

Para generar esta red necesitamos saber el número de nodos, “nodes”, que serán los extremos de la red además del punto que cruza ambas líneas, y el número de bordes, “edges”, que son el número de líneas entre cada nodo que dispone nuestro sistema. Cada edge se puede componer de tantas líneas como se quiera. En nuestro caso usaremos dos líneas por cada edge. La red quedaría de la siguiente manera:

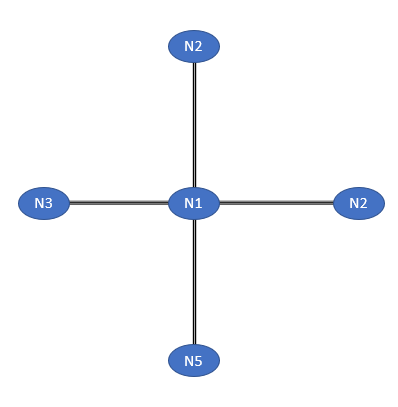
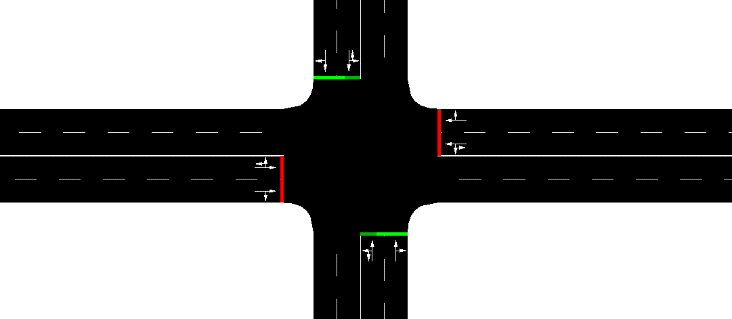


Figura 1: Nodos de la red a simular

Figura 2: Muestra de la red en SUMO

En la figura de la izquierda aparecen los nodos que componen la red y entre ellos los bordes o también llamadas “edges”. En la figura de la derecha podemos observar el nodo principal. Alrededor suyo se observa como cada recta está compuesta de dos líneas.

Para conseguir esta red, se han creado diferentes archivos con diferente información, todos ellos en la carpeta *Net*, la cual se accede desde la carpeta inicial:

* **Configuración de los nodos (my\_nodes.nod):** Creamos un archivo donde configuramos la posición de cada nodo.
* **Configuración de los bordes o “edges” (my\_edges.edg**): En este archivo configuramos los bordes que unen los nodos, de tal manera que conseguiremos una red.
* **Configuración de los tipos de líneas (my\_types.type):** En este archivo contendremos los diferentes tipos de líneas que habrá en cada edge. El tipo de Edge indica el número de líneas y su prioridad. En nuestro caso siempre consideramos que hay dos líneas por cada edge.
* **Configuración aleatoria de coches(trips.trips):** De todos los archivos mencionados, este será el que tenga mayor importancia, ya que con el configuraremos las rutas aleatorias de los coches.
* **(Adicional)** **Configuración manual de rutas de coches (my\_route.rout):**  Adicionalmente, y opuestamente al archivo de trips.trips, se puede realizar una configuración manual de cada ruta de un coche, en lugar de la generación aleatoria que ofrece trips. En ella se debe configurar el trayecto entre nodos que va a cursar, así como la longitud del coche y su velocidad.
* **Configuración de la red**

Una vez están creados estos archivos, se puede crear la red mediante el siguiente comando y desde la carpeta de Net:

*netconvert --node-files my\_nodes.nod.xml -- edge-files my\_edge.edg.xml -t my\_type.type.xml -o my\_net.net.xml*

Con la red ya creada, podemos crear el archivo de configuración de SUMO (config.file), que será el que permita desplegar nuestra red en el programa. Los parámetros importantes que debemos introducir son la red creada anteriormente, la ruta manual de coches generada previamente si se quiere añadir, y archivos adicionales como pueden ser detectores de tráfico. Además, se configura el tiempo de simulación y el enlace de las estadísticas de salida.

* **Configuración de la ruta aleatoria de coches en nuestra red**

En este apartado introducimos el archivo trips.trips en nuestra red. Para ello usaremos el siguiente comando y desde la carpeta de Net:

*python randomTrips.py -n ./simple.net.xml*

* **Archivo de configuración**

Por último, nos faltaría desplegar el archivo de configuración mediante la siguiente línea de comandos desde la carpeta de Net:

*sumo-gui config.sumocfg*

*sumo -c config.sumocfg*

Como se puede observar, sólo es necesario introducir uno de los dos comandos. La diferencia se basa en la interfaz gráfica que se abriría en el primero, mientras que en el segundo se simularía directamente desde la consola de comandos.

1. **Generación de estadísticas**

Como se ha mencionado anteriormente, la importancia de esta simulación es poder reducir el tiempo de espera de los coches en un cruce de calles mediante el cambio óptimo de los semáforos. Para ello, necesitamos unas estadísticas que sea útiles para posteriores aprendizajes.

SUMO nos proporciona una gran cantidad de datos que se pueden extraer de cada simulación. Dependiendo de lo que nos interese tendremos que escribirlo en la línea de comandos. Estas estadísticas se guardarán en la carpeta que se ha mencionado en el config.file.

* Extraer estadísticas generales:

*sumo -c config.sumocfg –summary <FILEOUTPUT>*

* Información del viaje:

*sumo -c config.sumocfg --tripinfo-output < FILEOUTPUT>*

* Información de la simulación total:

*sumo -c config.sumocfg --full-output < FILEOUTPUT >*

* Información del tráfico agregado:

*sumo -c config.sumocfg --duration-log.statistics < FILEOUTPUT>*

* Información del estado de los semáforos:

En este caso, se debería añadir en el config.file la siguiente línea:

*<timedEvent type="SaveTLSStates" source="<TLS\_ID>" dest="<OUTPUT\_FILE>"/>*

1. **Traci**

Como ya se ha dicho anteriormente, nuestro objetivo principal es reducir el tiempo de espera de cada coche en nuestras simulaciones para poder escalarlo a un cruce real de semáforos.

Como ya hemos definido dicho cruce, tenemos que ser capaces de poder variar los semáforos a nuestra libre elección. Traci es una librería de SUMO que ayuda con ello. Permite cambiar el estado de los semáforos, provee información del tiempo de espera de los coches, etc.

En una primera aproximación usamos traci para detectar si un coche se encuentra esperando en el semáforo. Para ello hemos añadido una serie de detectores a unos ligeros metros del semáforo.

En esta simulación, damos preferencia al estado del semáforo este y oeste cuando pasan algún coche por dichas líneas, de tal manera que se abren ambos semáforos mientras que los del norte y sur permanecen cerrados. Esta no es la mejor solución ya que se forman grandes atascos en las líneas norte y sur como se observa en la siguiente figura:

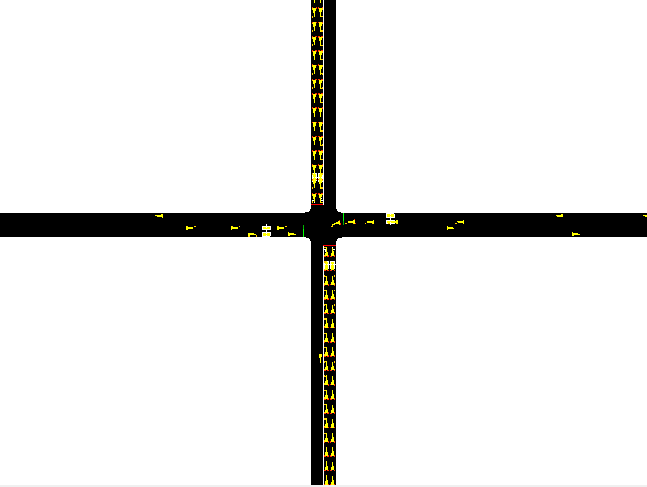


Figura 3: Simulación con Traci

Para probar este modelo tenemos que entrar en el siguiente comando desde la carpeta inicial:

*python “No Reinforcement Learning”/testTracy.py*

Una vez ejecutado se abre la gui de sumo, en donde se debe hacer click en start.

1. **Reinforcement Learning**

Como hemos observado, usar solo traci no es una buena opción ya que se debería configurar manualmente el cambio de estados de los semáforos y siempre habrá alguno que tenga la preferencia. Como el objetivo de la práctica es que el semáforo sea capaz de aprender por sí solo la manera de evitar atascos tenemos que introducir en esta configuración de red el aprendizaje automático.

* 1. **Q-Learning**

Q-Learning se basa en ir adaptando tu modelo en función a lo que deseas. En q-learning se tiene un agente, que será el que va a ir siendo adaptado, un estado el cual será el estado en el que se encuentra actualmente el agente y una recompensa. Como recompensa usaremos el tiempo de espera.

Para usar el q-learning realizado simplemente hay que escribir el siguiente comando desde el path inicial:

*python "Reinforcement Learning"/QL/Ql\_2way\_intersection.py*

Este comando ejecuta nuestro q-learning en sumo y nos abre la gui en el que se puede ver cómo se van comportando los coches gracias a esta técnica. Cómo se puede observar en la siguiente imagen, ya no usaremos detectores gracias a que las funciones de traci de tiempo de espera de los vehículos nos devuelve información muy útil.

Además, esta técnica por sí sola no resulta ser muy efectiva ya que se observa que comienza a haber tráfico en el este y oeste, mientras que en el norte y sur está aparentemente despejado.

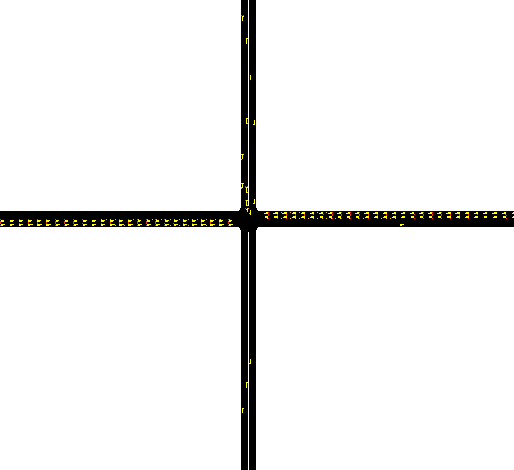


Figura 4: Simulación con Q-Learning

También podemos cambiar diferentes parámetros de la simulación y ver qué cambios produce. Los parámetros que podemos añadir a nuestra línea de comandos superior son:

|  |  |  |
| --- | --- | --- |
| **Comando** | **Descripción** | **Default** |
| -route | Ruta de coches asignada | './Net/trips.trips.xml' |
| -a | Alpha learning rate | 0.1 |
| -g | Gamma discount rate | 0.79 |
| -e | Epsilon | 0.05 |
| -me | Minimo epsilon | 0.005 |
| -d | Epsilon decay | 1.0 |
| -mingreen | Minimo tiempo en verde | 10 |
| -maxgreen | Maximo tiempo en verde | 30 |
| -gui | Gui en SUMO | True |
| -fixed | Ejecución con tiempo de las señales de tráfico fijo | False |
| -s | Número de segundos de la simulación | 5000 |
| -r | Recompensa por media de tiempo (-r wait) o por media de espera en cola (-r queue) | wait |
| -runs | Número de veces ejecutada la simulación | 1 |

Una vez se ha ejecutado la simulación, se guardará una csv en la carpeta *"Reinforcement Learning"/QL/Outputs* con los valores dados en la simulación. En este csv aparece cada paso de la simulación, la recompensa, el número total de coches parados y el tiempo total de espera.

De este csv podemos mostrar dos tipos de gráficas:

* **Tiempo total de espera por coche por step**

En esta gráfica se muestra el tiempo de espera de cada vehículo en cada paso de la simulación. Como se puede observar, al inicio de la simulación los coches no esperaban mucho tiempo parados. En cambio, al final de esta el tiempo de espera de cada coche fue altamente incrementado. Si observamos la simulación en SUMO, se puede comprobar que se forman grandes atascos en este y oeste por lo que el tiempo de espera aumenta.

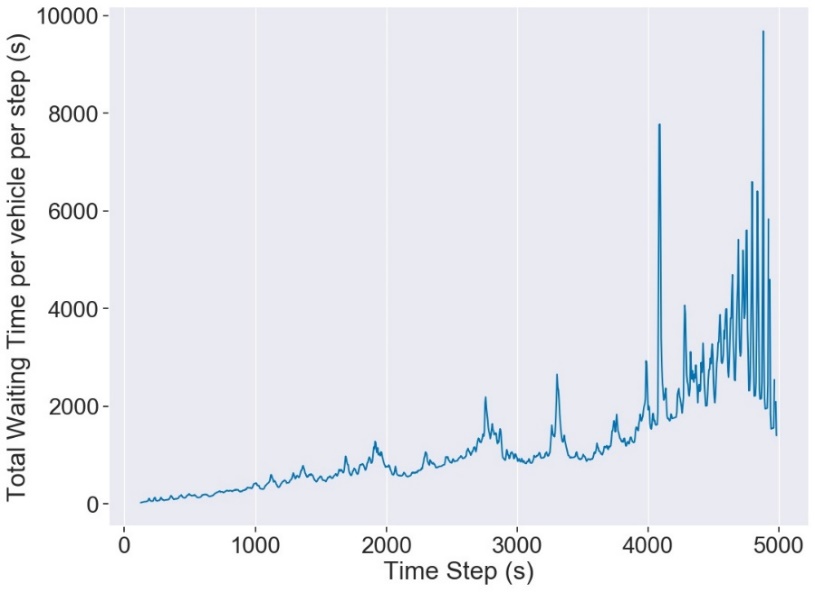


Figura 5: Tiempo total de cada vehículo en cada step

Para obtener está grafica deberemos usar el siguiente comando desde la carpeta inicial:

*python "Reinforcement Learning"/Plots/plot.py -f "Reinforcement Learning"/QL/Outputs -out "Reinforcement Learning"/QL/Plots/Total\_Waiting\_Time\_Per\_Vehicle\_Per\_Step*

Cabe destacar que en la carpeta Outputs solo deberemos tener el csv a graficar. En la carpeta plots se guardará esta gráfica.

* **Tiempo total de espera de todos los coches por step**

En esta gráfica se observará el tiempo total de todos los coches en un mismo step. Como se observa en la figura, el tiempo total es mucho mayor, alcanzando casi los

100.000 segundos. Estos datos siguen en concordancia con lo que hemos visto en la visualización de SUMO.

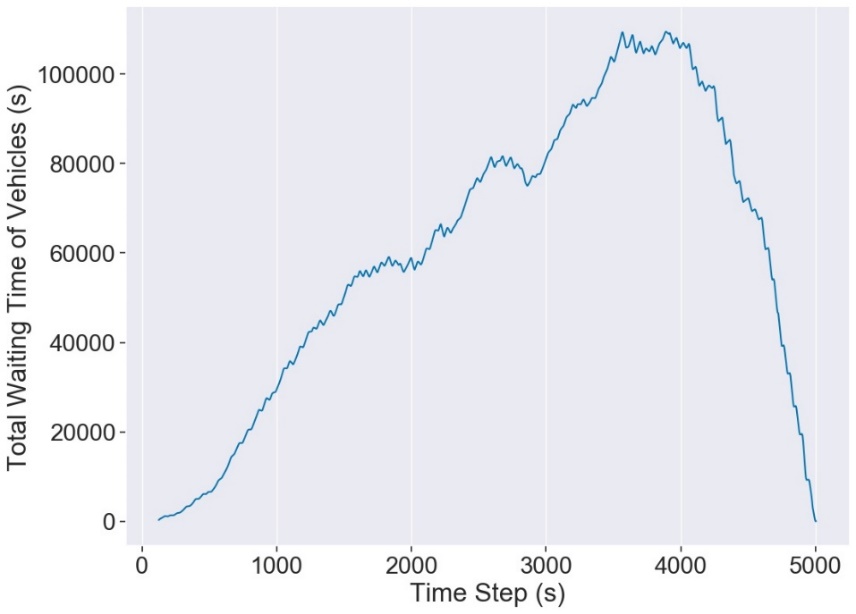


Figura 6: Tiempo total de todos los vehículos

Para ejecutar esta gráfica debemos irnos a la carpeta inicial y desde allí ejecutar el siguiente comando:

*python "Reinforcement Learning"/Plots/plot2.py -f "Reinforcement Learning"/QL/Outputs -out "Reinforcement Learning"/QL/Plots/Total\_Waiting\_Time\_Of\_Vehicles*

Al igual que en el primer plot, solo debemos mantener el csv que se quiere graficar en la carpeta de Outputs.

* 1. **A2C**

Como ya hemos visto, el q-learning por sí solo no nos reduce el tiempo de espera de los coches. Para ello introducimos A2C. A2C es una librería de OpenAI útil para proyectos de reinforcement learning. En esta librería introduciremos nuestro enviroment ya creado por nosotros.

Para ejecutar la simulación debemos usar el siguiente comando desde la carpeta inicial:

*python "Reinforcement Learning"/A2C/A2C\_2way\_intersection.py*

Esto nos devuelve otro csv de las mismas características que el anterior y con el que podremos dibujar ambas gráficas:

* **Tiempo total de espera por coche por step**

Al igual que en el anterior apartado, dibujamos el tiempo total de espera de cada vehículo en cada paso de la simulación con el siguiente comando:

*python "Reinforcement Learning"/Plots/plot.py -f "Reinforcement Learning"/QL/Outputs -out "Reinforcement Learning"/QL/Plots/Total\_Waiting\_Time\_Per\_Vehicle\_Per\_Step*

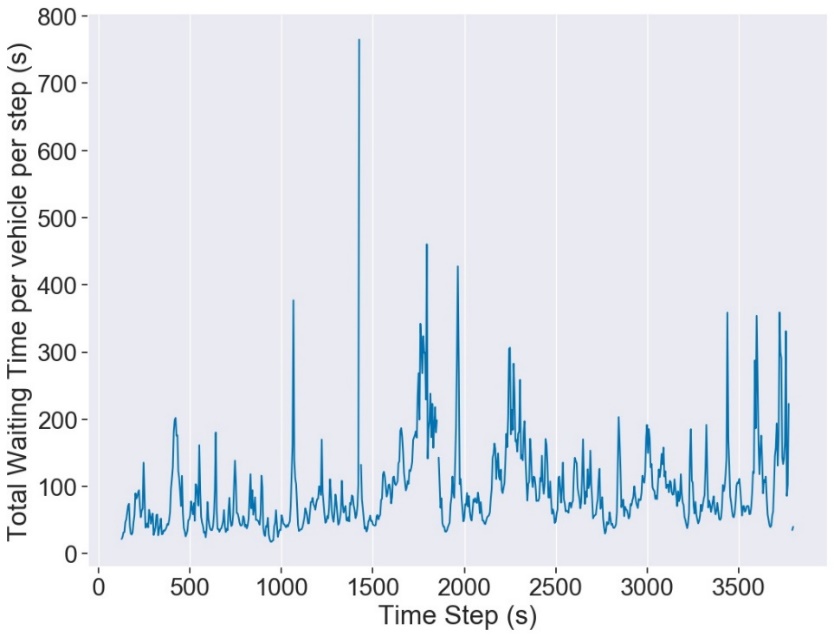


Figura 7: Tiempo total de cada vehículo en cada step

Como se puede observar, las magnitudes de los tiempos de espera han bajado notablemente. Si que es verdad que existen aún algunos picos, pero la mayoría del tiempo se mantiene entre 100 y 200 segundos.

* **Tiempo total de espera de todos los coches por step**

Siguiendo la misma fórmula, podemos graficar el tiempo total de los coches con el siguiente comando:

*python "Reinforcement Learning"/Plots/plot2.py -f "Reinforcement Learning"/A2C/Outputs -out "Reinforcement Learning"/A2C/Plots/ Total\_Waiting\_Time\_Of\_Vehicles*

Si observamos la gráfica, comprobamos que los picos aparecen en aproximadamente los mismos steps que en la gráfica de tiempo total de espera por coche, lo que se puede determinar que ambas gráficas están correladas. Además, hemos reducido el tiempo total en el máximo de 100.000 segundos a 2.500, por lo que demostramos que la librería A2C nos ha ayudado en la reducción del tiempo de espera de la simulación.

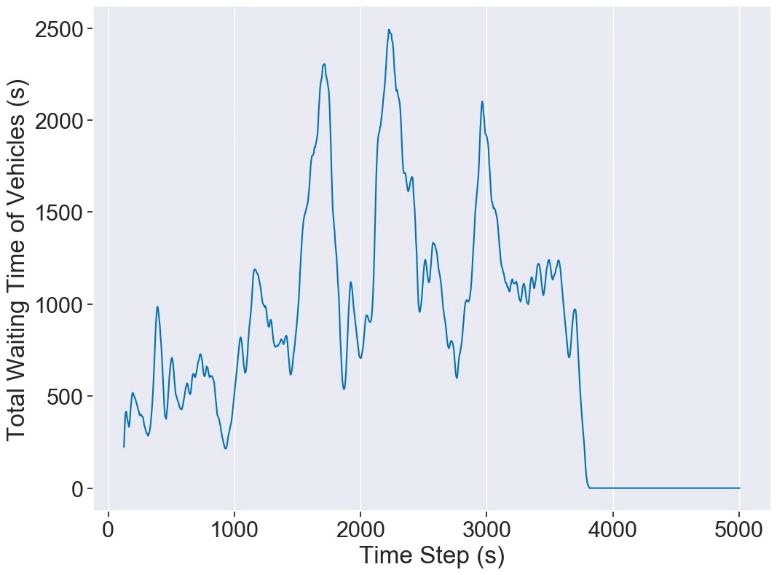


Figura 8: Tiempo total de todos los vehículos