Universidad Zaragoza
1542

Trabajo Fin de Grado

Modelado y control de sistemas de tráfico mediante redes flexibles

Modeling and controlling traffic systems by flexible nets

Autor/es

Carlos Cunchillos Andicoberry

Director/es

Jorge Emilio Júlvez Bueno

Grado en ingeniería electrónica y automática

Departamento de Informática e Ingeniería de Sistemas

Escuela de Ingeniería y Arquitectura de Zaragoza

2021

RESUMEN

El control óptimo de los semáforos de las ciudades es un trabajo delicado debido a que el flujo de coches es incierto y presenta amplias fluctuaciones a lo largo del día. Hay muchas formas de afrontar este problema, pero este proyecto lo analiza de manera macroscópica, de tal forma que el sistema global de tráfico es una interconexión de subsistemas o secciones, que son carreteras de pequeña longitud y un único sentido. En cada intersección entre carreteras de distintos sentidos se colocará un semáforo.

El principal objetivo es automatizar las frecuencias de los semáforos en función del flujo de coches para minimizar el tiempo total de espera. Inicialmente se ha modelado el sistema con redes flexibles para facilitar el análisis con respecto a modelos previos basados en redes de Petri continuas. El tiempo de cómputo para la optimización de este modelo resultó ser demasiado alto, por lo que se buscó un método alternativo. Este ha consistido en el diseño de un modelo de tráfico propio que ha sido programado en Python, con el que se ha conseguido analizar y posteriormente optimizar las frecuencias de activación de los semáforos. El método de control que se ha utilizado es el control predictivo por modelo.

En primer lugar, se explica qué conforma el modelo de tráfico. Posteriormente se hace una breve introducción sobre las redes de Petri continuas. En ellas está fundamentado el modelo de tráfico del que parte este proyecto. Luego se introducen las redes flexibles. Sobre estas redes flexibles se diseñará el modelo de tráfico y se analizará. Finalmente, con el inconveniente del costoso tiempo de cálculo, se explica cómo se ha diseñado el modelo programado, su posterior control y su potencial.

INDICE

1. SISTEMAS DE TRÁFICO

La definición de un sistema de tráfico puede ser muy extensa y compleja, pero en la realización de este proyecto se ha optado por una aproximación a este tipo de sistemas de forma macroscópica. Los elementos que lo componen son: la carretera, los coches y los semáforos.

El modelo se compone de la unión de tramos de carretera (secciones), cuyas características son tomadas de [1]: un segmento de carretera de 200 metros, 2 carriles de una sola dirección y una capacidad máxima de 60 coches.

Son las intersecciones perpendiculares un aspecto destacado del proyecto, ya que son en ellas donde se colocan los semáforos. Alternando la activación de estos semáforos se consigue controlar el flujo del sistema. En cada sección se considerarán tres variables: la densidad de coches, la velocidad media y el flujo de coches, donde el flujo es igual a la densidad de coches por la velocidad media.

De forma esquemática, se entrará en detalle más adelante, se pretende obtener un modelo semejante al *diagrama fundamental de tráfico* [2]. Dicho diagrama, ver figura 1, captura la relación entre la densidad y el flujo de una sección. El diagrama se compone de tres tramos bien definidos: El primero, en el cual el flujo crece de manera proporcional a la densidad de coches, se llamará flujo libre. En el segundo, el flujo ya no crece, si no que pasa a ser "constante". El último tramo, refleja la reacción que se produce cuando hay una densidad muy elevada de coches, el flujo se reduce a medida que la densidad aumenta.

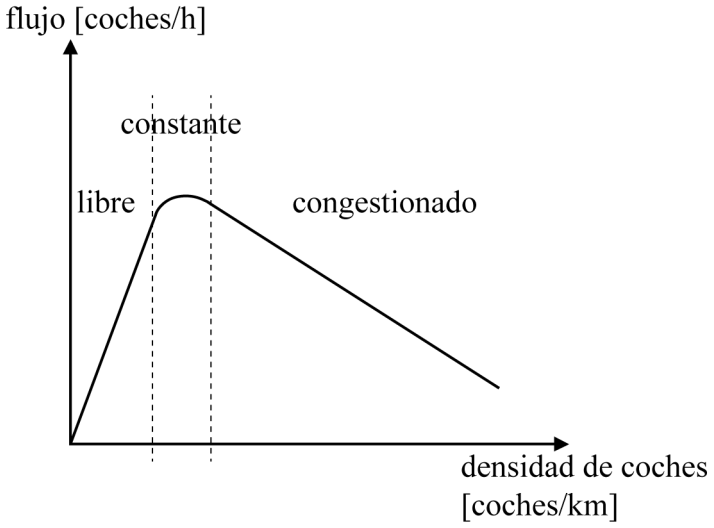


Figura 1. Diagrama fundamental del tráfico

El gran potencial de utilizar un enfoque macroscópico composicional es que se pueden modelar sistemas de tráfico muy variados. Posteriormente, con el uso del control predictivo por modelo (explicado en una sección posterior), se conseguirá adecuar la secuencia de activación de los semáforos a cualquier tipo de fluctuación en el flujo de entrada de coches al sistema, ocasionadas por las diferentes horas del día o incluso por detenciones esporádicas, debido a accidentes, etc.

2. REDES DE PETRI CONTINUAS

Redes de Petri

Para poder entender el funcionamiento de la red flexible sobre la que se modelará el sistema de tráfico y que se explicará en la próxima sección es necesaria una introducción breve sobre el funcionamiento de las redes de Petri convencionales.

Una red de Petri es un formalismo de modelado para sistemas de eventos discretos. Las redes de Petri tienen muchos ámbitos de aplicación, pero normalmente se usan para modelar centros de manufactura o logísticos. Se van a definir de forma resumida los conceptos de las redes de Petri, basándose en [3], [4].

Una red de Petri, N, es una tupla N= {P, T, Pre, Post} donde P es el conjunto de lugares, T es el conjunto de transiciones, Pre y Post son matrices de enteros de |P| filas y |T| columnas que contienen los pesos de los arcos. Por ejemplo, la red de la figura 2, tiene dos lugares P={p1, p2}, tres transiciones T={t1, t2, t3} y arcos que conectan los lugares y transiciones. Si no se asocia explícitamente un peso a un arco, se considera que el peso es uno. Los lugares se consideran las entradas y salidas de una transición. Cada lugar tiene asignado un marcado o número discreto de tokens.

Tomando como ejemplo la situación mostrada en la figura se pueda explicar la evolución del marcado de la red: cuando una transición está habilitada, por ejemplo, t2, puede dispararse. Su disparo consume tokens de los lugares de entrada (tantos como indiquen los pesos de los arcos) y produce tokens en los lugares de salida (tantos como indiquen los pesos de los arcos). Por ejemplo, el disparo de t2 consume un token de p1 y produce un token en p2.

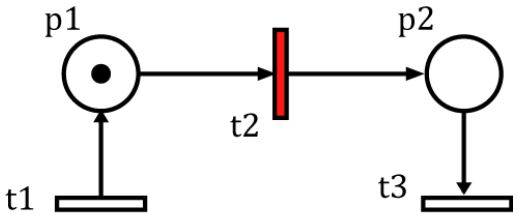


Figura 2. Red de Petri simple

Redes de Petri Continuas

Las redes de Petri continuas son una variante de las discretas y sobre ellas está fundamentado el modelo de tráfico del que se parte [1] para la realización de este proyecto. La mayor diferencia es que como su propio nombre indica, el marcado deja de ser discreto y pasa a ser un valor real no negativo. Debido a esto, se evita el problema de la explosión de estados, inherente a los modelos discretos con excesivo marcado.

Una transición t está *activada* cuando los marcados de todos sus lugares de entrada son positivos (m corresponde al marcado del lugar p). El arco que une el lugar p con la transición t tiene peso, indicado como Pre[p, t]. El *grado de activación* de t se define como:

$$activ(t,m) = min\left\{\frac{m[p]}{Pre[p,t]}\right\} \quad (1)$$

El flujo de una transición se define de acuerdo a la semántica de servidores infinitos [5], es decir, es proporcional a su grado de activación:

$$f[t](\tau) = \lambda[t] \cdot min\left\{\frac{m[p](\tau)}{Pre[p,t]}\right\} \quad (2)$$

donde $\lambda[t]$> 0 es un parámetro constante que representa la velocidad de la transición. Como resultado, se expone que el flujo depende del marcado de los lugares de entrada, del peso de los arcos y de $\lambda[t]$. La evolución del marcado de un lugar, viene dado por la expresión:

$$\dot{m}[p] = \sum_{t \in \cdot p} Post[p,t] \cdot f[t](\tau) - \sum_{t \in p \cdot} Pre[p,t] \cdot f[t](\tau) \quad (3)$$

Por tanto, tomando como referencia la estructura típica de una red de Petri, se obtiene que la evolución de m[p], donde p es un lugar de entrada a t, es:

$$\dot{m}[p] = -Pre[p,t] \cdot f[t] = -\lambda[t] \cdot m[p] \quad (4)$$

Por lo que, en este caso, la evolución de m[p] es independiente de Pre[p,t].

Para que esto no ocurra y se pueda modelar el diagrama fundamental de tráfico, se idea una nueva forma de interconectar los lugares y transiciones. Se añaden arcos en ambos sentidos (figura 3 [1]). Los pesos q y q-a siempre deberán ser positivos, de tal forma que el flujo de t_2 será: $f[t_2] = \lambda[t_2] \cdot m\,[p_2]/q$ y el marcado de p2 evolucionará según la ecuación: $\dot{m}[p_2] = (q - a - q) \cdot f[t_2] = \frac{-a}{q} \cdot \lambda[t_2] \cdot m[p_2]$. Esto significa que el nuevo marcado dependerá del peso de los arcos, algo realmente interesante para el sistema de tráfico.

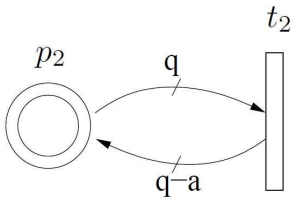


Figura 3. Bucle entre lugar y transición

Redes de Petri continuas aplicadas a un modelo de tráfico

Aplicando todos estos conceptos y tomando como referencia el diagrama fundamental del tráfico [2], se llega al siguiente sistema de tráfico modelado con una red de Petri continua en [1]:

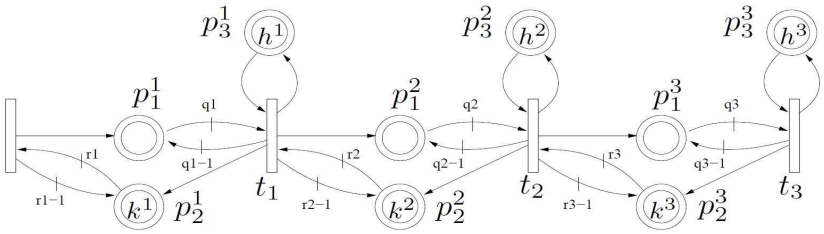


Figura 4. Modelo de tráfico de tres secciones con red de Petri continua

En él se representan 3 secciones de tráfico conectadas. Cada sección va de una transición a la siguiente. Como se ha comentado anteriormente, el diagrama fundamental del tráfico tiene 3 tramos diferenciados que coinciden con los 3 lugares de cada sección.

Analizando primeramente una sección suelta. El flujo de salida de la transición t_i depende de los lugares p_1^i y p_3^i de tal forma que, aplicando la ecuación 2: $f[t_i] = \lambda[t_i] \cdot min \cdot \left\{\frac{m[p_1^i]}{q}, h\right\}$. Es decir, que cuando el marcado de p_1^i sea menor que el de p_3^i (h), el flujo de salida será proporcional al marcado y al peso del arco de p_1^i. Y cuando ocurra lo contrario, el flujo de salida será constante.

El lugar p_2^i representa los huecos disponibles de cada sección, esto es, la diferencia entre el número máximo de coches admisibles (*maxcar*) y los coches actuales que hay en la sección. Se empleará más adelante cuando se unan varias secciones, de tal forma que se tenga una forma de saber cuál es el marcado de la sección posterior, para así poder calcular el flujo correctamente de la sección anterior, ya que, si la sección posterior está obstruida, no podrán circular los coches. Se emplea cuando se unen dos secciones, de esta forma, t_i pasa a tener otro lugar de entrada, por lo que la ecuación del flujo de salida quedaría así:

$$f[t_i] = \lambda[t_i] \cdot min \cdot \left\{\frac{m[p_1^i]}{q}, h, \frac{maxcar - p_2^{i+1}}{r}\right\} \quad (5)$$

Representando esta ecuación lineal a tramos quedaría el diagrama de la Figura 5 para el flujo de una transición t:

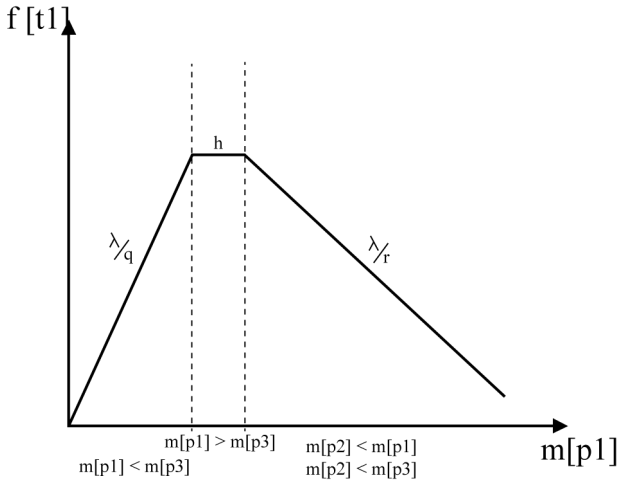


Figura 5. Representación ecuación lineal a tramos

El diagrama se asemeja bastante a lo que debería ser una forma de campana como la de la figura 1 (diagrama fundamental de tráfico).

Posteriormente, con la introducción de las redes flexibles, se explicará la incorporación de intersecciones y semáforos con el consecuente uso de una función objetivo para optimizar esta secuenciación de encendido y apagado de los semáforos.

3. REDES FLEXIBLES

Las redes flexibles [6] se componen de dos redes, la de eventos y la de intensidades, que gestionan la relación entre los estados y los procesos de cada sistema. Las inecuaciones asociadas a ambas redes permiten trabajar con parámetros que tengan incertidumbre, como es el caso de la llegada de coches a una intersección.

Son cuatro elementos los que forman una red flexible: lugares (p), transiciones (t), manejadores de eventos (v) y manejadores de intensidades (s). Los lugares se representan con círculos, las transiciones con rectángulos y los manejadores con puntos. La unión de ambas redes es la que gobierna el sistema, de tal forma que en la red de intensidades se calcula la velocidad a la que se generarán las acciones dependiendo del estado del marcado de la red de eventos, la cual, especificará las acciones que se deben ejecutar y cuál será el próximo estado. Tómese como acciones cualquier evento cuya ocurrencia pueda cambiar el estado del sistema, en el caso que concierne a este proyecto, el movimiento de coches de una sección a otra.

Las variables de estas redes no requieren de distribuciones de probabilidad, solamente de los intervalos entre los que se encuentran. Con esto se pueden conseguir análisis mucho más eficientes basados en programación lineal. Una de las ventajas también es que el sistema se representa de forma más sencilla gráficamente. El contenido posterior está fundamentado en [6].

Red de eventos

En esta red, los manejadores de eventos conectan los lugares con las transiciones. Son estos los que producen el cambio en el marcado según las acciones de las transiciones. Los arcos de la red de eventos van de los lugares hacia los manejadores de eventos y terminan con una flecha, mientras que las aristas van desde las transiciones hacia los manejadores de eventos y son una línea. Al contrario que en las redes de Petri, no se pueden conectar los lugares con las transiciones directamente.

Los lugares tienen el marcado y las transiciones contienen la cantidad de acciones que se han producido. Estas acciones requieren un tiempo para ser producidas (este tiempo depende de la red de intensidades). El estado de una red de eventos depende del marcado, del número de acciones, de los cambios en el marcado y de la ejecución de estas acciones.

Con un pequeño ejemplo se muestra el potencial de estas redes, pudiendo añadir inecuaciones:

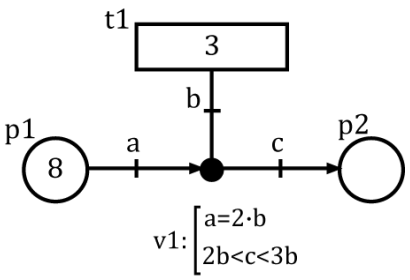


Figura 6. Red de eventos

Las inecuaciones asociadas al manejador de eventos v_1 de la figura 6 definen el cambio de marcado en el sistema, donde *a, b* y *c* son las etiquetas de los arcos/aristas. La ecuación a=2·b indica que por cada acción de t_1 ejecutada por v_1 se consumen dos tokens de p_1.

Por otro lado, la inecuación 2b<c<3b significa que la ejecución de una acción de t_1 por v_1, producirá una cantidad c de tokens en p_2 donde c puede ser cualquier valor en el intervalo[2, 3].

Análogamente a las redes de Petri continuas, la ecuación que hace que el marcado evolucione [6] tiene en cuenta el marcado que entra, desde los manejadores de eventos hacia el lugar y el marcado que se va, desde el lugar hacia los manejadores de eventos:

Red de intensidades

De igual forma, esta red conecta los manejadores de intensidades con las transiciones y lugares. El marcado de los lugares produce y consume las intensidades de las transiciones a través de los manejadores de intensidades. La intensidad de la transición es la velocidad a la que se generan acciones en esta, es decir, la integral en el tiempo de la intensidad de la transición es el número total de acciones que se producen.

Al igual que la red de eventos, cada transición tiene una intensidad inicial (o por defecto) asociada $\lambda_0[t_j]$. Entonces, la intensidad en un instante dado es igual a $\lambda_0[t_j]$ más los cambios positivos de intensidad menos los cambios negativos de intensidad, llevados a cabo por las inecuaciones de los manejadores de intensidades.

La Figura 7 muestra un pequeño ejemplo de red de intensidades:

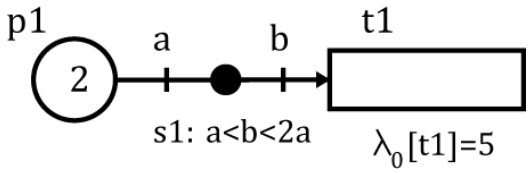


Figura 7. Red de Intensidades

De manera análoga al otro ejemplo, la inecuación asociada al manejador de intensidades s_1 indica que cada token de p1 produce entre una y dos acciones en t1 por unidad de tiempo. En otras palabras la intensidad de t_1 es $\lambda[t_1] = \lambda_0[t_1] + b$ donde $b \in [m[p], 2 \cdot m[p]]$.

4. MODELO DE UN SISTEMA DE TRÁFICO CON REDES FLEXIBLES

Una vez explicadas las bases de las redes flexibles y tomando como punto de partida la red de Petri continua de la figura 4, esta sección muestra como modelar un sistema de tráfico mediante redes flexibles. Se añade el Anexo I para más información acerca del análisis y modelado de esta parte.

Sección simple

Inicialmente se ha diseñado la red flexible del sistema para una sola sección (figura 8). Como se puede observar, es mucho más simple visualmente que la red de Petri continua, pero requiere una base de conocimiento acerca de su funcionamiento más amplia. En resumen: t_0 simula la entrada de coches al sistema, v_0 hace evolucionar esos coches para que lleguen al lugar p_1 de la sección (la carretera). Posteriormente, dependiendo de la

cantidad de coches que haya dentro de la sección, s_1 se encarga de gestionar la velocidad que van a llevar los coches, activando la transición t_1 y regulando la salida de estos coches a la próxima sección por v_1.

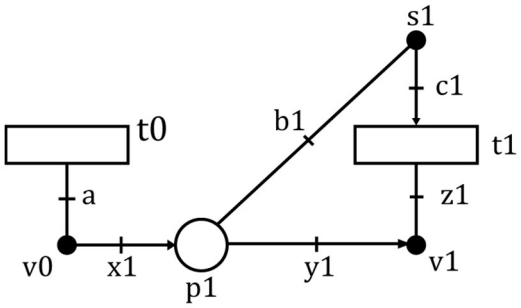


Figura 8. Red flexible de una sección de tráfico

Ahora se va a analizar en detalle el funcionamiento de esta red y la adaptación de las ecuaciones de la red continua, para después introducir las intersecciones y los semáforos, conformando así el modelo de tráfico global.

Empezando por la entrada de coches al sistema, t_0 representará una función que en cada periodo de simulación tomará un valor aleatorio dentro de una distribución uniforme (coches/segundo). Se ha elegido esta distribución por simplicidad de diseño, se podrían optar por modelos más complejos que tuvieran en cuenta las horas punta de tráfico, aleatoriedades de accidentes, etc… El manejador de eventos v_0 con la ecuación asociada v_0: $a = x_1$ implica que todos los coches que se "generan" en t_0 pasen al lugar p_1 en la misma proporción. En p_1 se encuentra el marcado, los coches que hay actualmente en esa sección. De aquí se parte hacia s_1, el manejador de intensidades que genera la ecuación lineal a tramos de la figura 5 (sin el último tramo ya que no tiene ninguna sección después):

$$s_1: \begin{cases} c_1 = b_1 * ff & si \quad m[p_1] < endff \quad (6) \\ c_1 = c_f & en\ otro\ caso \quad (7) \end{cases}$$

(6) Es la ecuación que relaciona el **flujo libre** de coches, *ff* viene de "free flow", la pendiente que da lugar a un incremento lineal de coches. Este tramo corresponde a cuando hay menos coches que *endff.*

(7) Es el tramo de **flujo constante**, en el que los coches llevan una velocidad constante c_f definida.

t_1 produce tantas acciones como le llegan desde c_1, que en este caso es el número de coches que pasan de una sección a otra. Por último, v_1 con v_1: $y1 = z1$, es el manejador de eventos encargado de trasladar tantos coches como acciones se hayan producido en t_1, restando esta cantidad a p_1.

Conectar secciones simples

Ahora, para encadenar estas redes simples y formar primero un tramo de carretera recto y posteriormente uno que contenga intersecciones y semáforos, basta con añadir tantos lugares, transiciones y manejadores de eventos por cada tramo de carretera, de igual forma que en la sección simple de la figura 8.

El funcionamiento sería idéntico, anidando una sección detrás de otra, el marcado de la anterior pasa a la siguiente. Pero para adaptar el funcionamiento de la red a la realidad, hay que tener en cuenta que si la sección posterior está saturada, esto es, el lugar p_{i+1} tiene un marcado mayor a un límite establecido, afecta a la velocidad de la sección anterior, reduciendo esta cuanto más coches haya en la siguiente sección. Por lo que se necesita añadir el tercer tramo de la ecuación (4). La figura 9 representa dos secciones unidas, añadiendo un semáforo en cada una de ellas.

Quedaría así:

$$s1:\begin{cases} c_1 = b_1 * ff & si \quad m[p_1] < endff \quad (8) \\ c_1 = c_f & si \quad m[p_1] \geq endff \quad (9) \\ c_1 = max_{car} - b_2 & si \quad max_{car} - m[p_2] \geq endcf \quad (10) \end{cases}$$

Este tramo es una ecuación lineal con pendiente negativa unitaria, en el que cuando en la sección posterior hay menos huecos que *endcf ("end constant flow")*, el flujo de la sección anterior depende de ese número de huecos disponibles, es decir, si no hay huecos, el flujo equivale a 0.

Como se puede observar, por la arquitectura de las redes flexibles, las ecuaciones (8), (9), (10) se tienen que re escribir en forma de inecuaciones, que van asociadas al manejador de intensidades s_1.

Intersecciones y semáforos.

La optimización del tráfico se realizará mediante el control de los semáforos que regulen las intersecciones. Tomando el modelo de semáforo de la red continua de tráfico y su máquina de estados [1], en la figura 9 se muestran dos secciones unidas con un semáforo cada una, ambas siguiendo la misma dirección. La intersección de las carreteras en ambas direcciones se modelaría gráficamente de igual forma que la figura 9, pero con otra red flexible que simulase el recorrido de la otra dirección y como punto de unión, la transición ts1, gobernando ambos sentidos. El semáforo se modela como una transición conectada a la salida de cada sección. Las aristas que unen el lugar de la sección posterior (p_{i+1}) al manejador de intensidades (s_i) de la sección anterior son las que permiten tener en cuenta el tráfico de secciones posteriores.

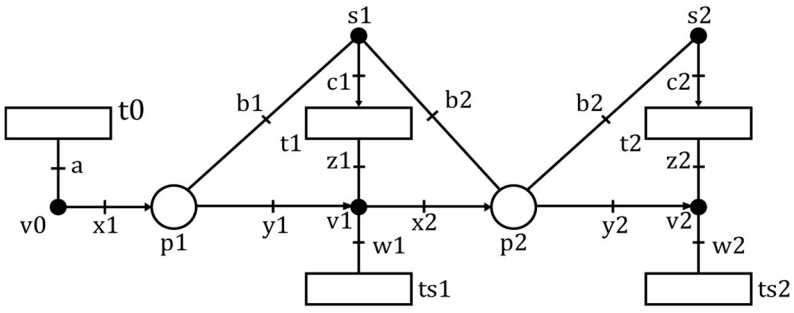


Figura 9. Intersección entre dos secciones con semáforo.

Se añade una breve explicación de la máquina de estados del semáforo de [1]. Esta máquina de estados está asociada a una dirección, para la otra dirección, la secuencia del semáforo será la inversa (cuando uno este verde, el otro rojo). Como se explica en [1], cada estado del semáforo dura un periodo de muestreo Δ, el tiempo que se tarda en pasar de verde a rojo es α y de rojo a verde β, con lo que $\alpha + \beta < \Delta$.

La explicación de los diferentes estados está en la siguiente tabla:

Verde	El flujo de coches es normal, no sufre ninguna modificación
Verde hacia rojo (ambar)	Estado correspondiente a la deceleración. Los coches se van frenando y el flujo de coches durante este estado viene dado por la expresión: $f \cdot \frac{\alpha}{2 \cdot \Delta}$, donde f es el flujo cuando el semáforo esta verde. Una vez calculado el flujo que pasaría en estado verde, se aplica esta medida correctora para saber cuántos han pasado en α.
Rojo	El flujo de coches es 0, están parados.
Rojo hacia verde	Estado representativo del arranque, la aceleración. se da en un tiempo β corresponde a: $f \cdot \frac{\beta}{2 \cdot \Delta}$ donde f es el flujo cuando el semáforo está verde.

Optimización de la red flexible.

Una vez se tiene definido todo el modelo del sistema, hay que definir una función objetivo que debe ser optimizada por los tiempos de conmutación del semáforo. La función objetivo será el tiempo total de espera de los coches del sistema. La minimización de esta función equivale a la maximización de la suma de los flujos de salida [1]:

$$max \sum_{i=1}^{n} \cdot \sum_{t \in T_{out}} f^i[t] \tag{11}$$

Una vez definido el modelo matemático que define la red flexible, se ha usado fnyzer [7], un paquete de Python de software libre diseñado para el análisis y la optimización de las redes flexibles. Con él, siguiendo la estructura de diccionario de Python que emplea, he conseguido modelar la red y analizarla. Este diccionario se puede consultar en el Anexo I.

Para empezar con el uso de fnyzer, se ha modelado la unión de dos secciones de tráfico sin semáforos (figura 9 sin las dos transiciones ts1 y ts2), solamente conectadas, de tal forma que el marcado de coches pasa de una sección a la siguiente. Se verifica su correcto funcionamiento en la figura 10.

La primera sección comienza con un marcado inicial de 45 coches mientras que la segunda con un marcado inicial de 0. Se aprecia como la primera se vacía a un ritmo constante (pendiente idealmente recta) ya que la sección posterior está vacía y por ende la segunda se va llenando, al mismo tiempo que se va vaciando.

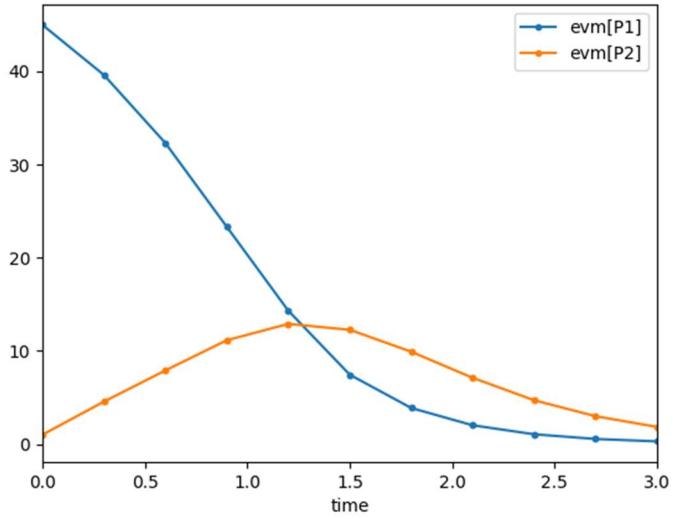


Figura 10. Dos secciones anidadas en Fnyzer

El problema que tiene esta herramienta de análisis es que tarda mucho en encontrar la solución óptima pues se basa en la resolución de un problema de programación lineal-entera. Por ello, el número de muestras recogidas por cada gráfica no puede ser muy elevado, ya que se demoraría mucho el proceso de cálculo.

Por ejemplo, obtener la figura 10 ha llevado 7 minutos y consiste en la búsqueda de la solución de 2 secciones. Se intentaron anidar 3 secciones y costó más de 20 minutos. Esto sin incluir las intersecciones ni los semáforos. El coste en tiempo aumenta con el número de secciones de forma exponencial debido a las variables binarias que son necesarias.

Debido a este costoso proceso, se ha decidido no utilizar fnyzer ni redes flexibles para el control y la optimización de modelos más complejos, optando por diseñar un modelo programado propio.

5. PROGRAMACIÓN DE UN SISTEMA DE TRÁFICO

Introducción

La elaboración del modelo programado de tráfico se ha hecho con la intención de que sea altamente composicional, al igual que los modelos previos. De tal forma que se ha optado por crear una clase que contenga el objeto sección (definida previamente en 1. Sistemas de tráfico), almacenando en vectores todas estas secciones se podrá operar fácilmente entre ellas.

Los dos atributos principales de esta clase son el marcado y el flujo, en ellos se va almacenando la evolución de la sección en cada periodo de tiempo. Además, esta clase tiene algunas constantes que se podrían modificar si se quisiese cambiar el modelo del sistema, estas constantes son: *maxcar*, q, r, h y λ. Estas cuatro últimas hacen referencia a unas constantes que se introdujeron en relación al modelo para calcular el flujo de cada sección, estudiadas en [1] y explicadas en la sección *2. Redes de Petri continuas*. Para trabajar con esta clase, se ha hecho uso de las listas que ofrece Python para almacenar

cualquier tipo de dato en una sola variable. La construcción de este objeto, como el desarrollo de todo el proyecto se puede consultar en el Anexo II.

Modelo de un sistema de tráfico

Una vez determinada la arquitectura, se tiene que realizar el algoritmo que procese el movimiento de marcado por las secciones. En el modelo se emplea un tiempo discretizado en periodos de $\Delta = 8segundos$. Este parámetro es lo que le cuesta recorrer a un coche cada tramo de carretera. Depende del resto de parámetros del sistema [1].

El algoritmo en cuestión se resume en: dado un marcado $m[p_i]$, calcular el $f[t_i]$ que tendría cada sección según la ecuación (5). Habrá que diferenciar dos tipos de secciones, las que a su salida tienen otra sección, por lo que el cálculo de su flujo se ve afectado por el marcado de la siguiente y las que no tienen ninguna sección posterior. Una vez se ha calculado el flujo de coches que se van, habrá que calcular el nuevo marcado. Esto es: el marcado actual, menos el marcado que se va en Δ, más el marcado que entra de la sección anterior en Δ.

$$m_{j+1}[p_i][\tau] = m_j[p_i] - \Delta \cdot f_j[t_i] + \Delta \cdot f_{j-1}[t_i] \tag{12}$$

Dado que todavía no se han implementado las intersecciones ni los semáforos, este proceso se repite hasta que se vacíe la red.

Primeramente se ha diseñado un sistema con dos secciones conectadas, para comprobar los resultados con respecto al análisis de fnyzer [7]. En estos primeros casos no se simula la entrada de coches al sistema ya que solo interesa la correcta evolución del sistema.

Como se puede ver en la figura 11, el resultado es prácticamente el mismo, pero el tiempo de ejecución han sido 5 segundos. Se aprecia como la primera sección (azul) tiene un marcado inicial de 50 coches y se va vaciando con una pendiente fija mientras la segunda sección (roja) se va llenando al mismo tiempo que se va vaciando y por ello cuando la velocidad de entrada de coches es menor que la de salida de coches, se va vaciando la red. Estas figuras se han conseguido haciendo uso de la librería matplotlib [8] de Python.

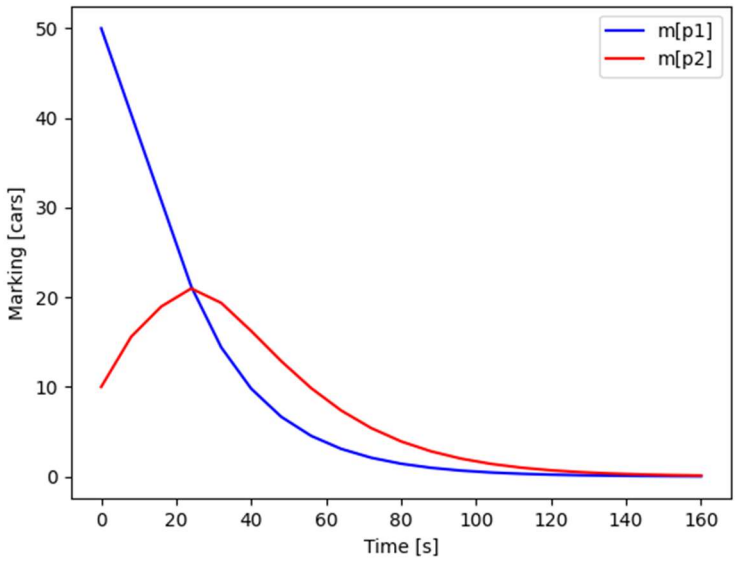


Figura 11. Modelado en Python. Dos secciones.

En la figura 12 a) hay otra prueba con distintos marcados iniciales para que se aprecie la obstrucción del flujo causada por la sección posterior. Se puede ver en la figura 12 b) que se cumple la semejanza con la forma de campana. La forma tan lineal a tramos de la gráfica es debida a que el sistema tiene un periodo de muestreo de 8 segundos.

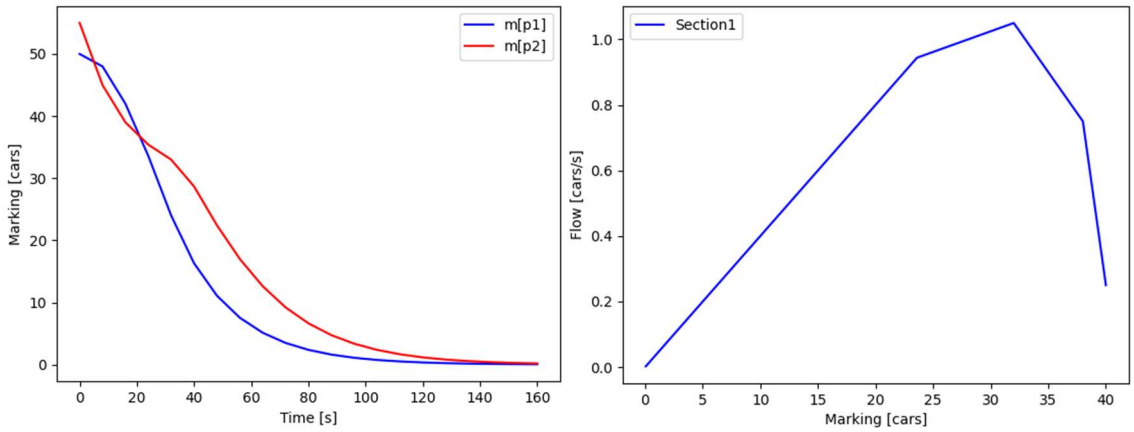


Figura 12. a) Verificación de congestión en las secciones anteriores (m[p1]) b) Densidad de coches frente a flujo

Se puede ver en la figura 13, cuatro secciones concatenadas, en las que los coches van fluyendo de una sección a la siguiente. Con este esquema se podrían configurar cuantas secciones anidadas se quisieran, pero no tendría mucho sentido ya que solamente se estaría construyendo una línea recta en la que no habría nada que controlar.

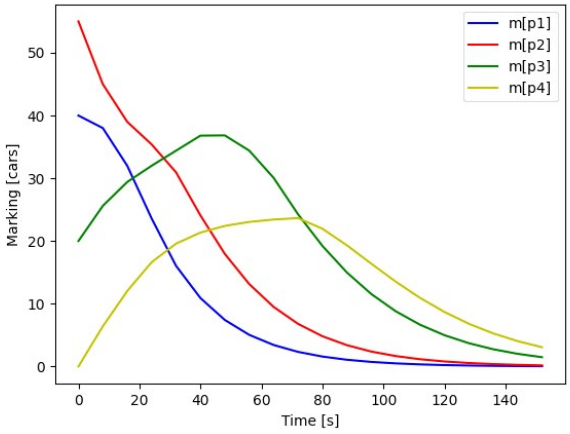


Figura 13. Flujo de coches entre secciones

Semáforos e intersecciones

Para añadir una intersección, es necesario que tenga un semáforo que controle el flujo de coches. Se ha optado por incluir el semáforo como parte de la propia sección, de tal forma que hay una variable cuya función es saber si la sección tiene un semáforo o no. Como está explicado en [1] y he resumido anteriormente, un semáforo es una máquina de estados.

Los atributos asociados al semáforo son: la variable que dice si tiene o no semáforo, el número de periodos que pasa en verde y en rojo (solamente pensado para un control fijo de la secuencia de los semáforos, figura 16 y 17), las constantes α y β, la lista de los estados por los que ha pasado el semáforo (para después poder mostrarlo en la gráfica) y

una variable que se usa para mover la máquina de estados. Estos posibles estados son: 0 para verde, 1 para verde hacia rojo, 2 para rojo y 3 para rojo hacia verde.

Para añadir las intersecciones, realmente lo único necesario es que la máquina de estados del semáforo de la sección horizontal sea igual y opuesta a la de la sección vertical. Por ello, se ha implementado una función que busque las carreteras horizontales con semáforo y en orden las vaya asociando a las secciones verticales con semáforo, esto conformará las intersecciones.

Una vez definido el nuevo modelo, se crea una pequeña red de tráfico que consta de dos secciones horizontales y dos secciones verticales que forman una intersección con un semáforo. El esquema está representado en la figura 14.

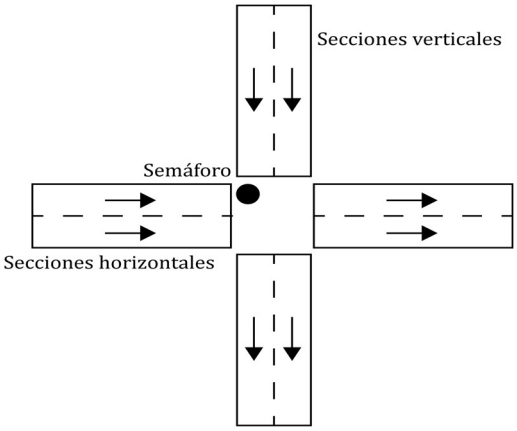


Figura 14. Esquema de una intersección.

Dibujando la red de Petri continua que representa el esquema anterior, se obtiene la siguiente figura:

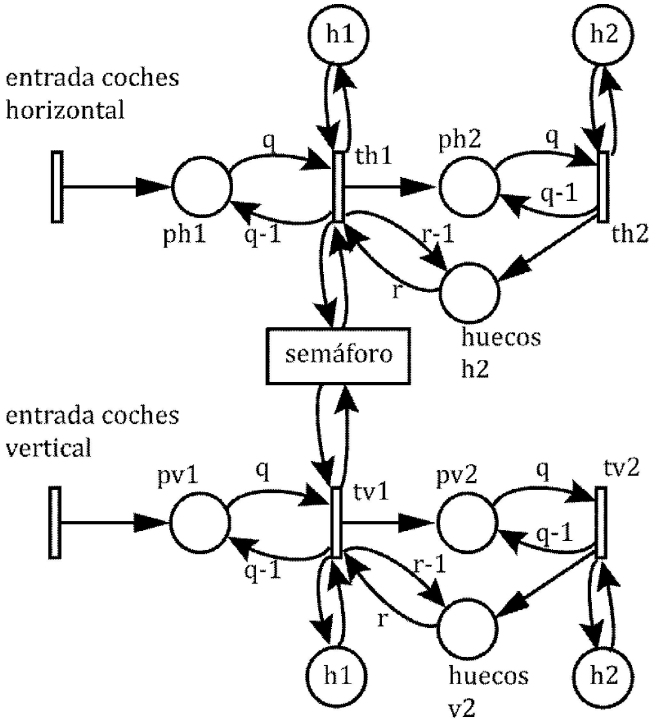


Figura 15. Red de Petri continua. Intersección con semáforo

Para comprobar la funcionalidad de este modelo, es necesario añadir una entrada de coches constante (pero aleatoria) al sistema. Para una primera prueba se ha decidido que esta entrada de flujo sea una distribución uniforme de [0.2, 0.3] coches por segundo para la carretera horizontal y de [0.4, 0.6] coches por segundo para la vertical. En cuanto al resto de constantes del modelo, para la secuencia de los semáforos se ha considerado que sea igual en estado rojo que en verde, 24 segundos (3 estados de $\Delta = 8$ segundos), α=3 y β=2.

El marcado inicial corresponde a: $m_0[p_{h_1}] = 15, m_0[p_{h_2}] = 10, m_0[p_{v_1}] = 20, m_0[p_{v_2}] = 15$ y $\lambda[t_{h_1}] = \lambda[t_{v_1}] = 4; \lambda[t_{h_2}] = \lambda[t_{v_2}] = 5$.

El resultado se puede ver en la figura 16. Con estos parámetros se tiene un sistema medianamente estable, en el que se aprecia cómo se van acumulando coches en la sección vertical debido a que el flujo de coches por esta entrada es mayor que por la horizontal. En color rojo se tiene la secuencia de la máquina de estados del semáforo para la carretera horizontal 3 periodos en verde y 3 en rojo más un periodo para conmutar entre verde y rojo y viceversa. En la figura se representa de tal forma que el estado más bajo de la línea de cruces es el semáforo en verde (para la línea horizontal), el siguiente es la conmutación de verde a rojo, el próximo rojo y las cruces superiores significan el estado de conmutación de rojo a verde de nuevo.

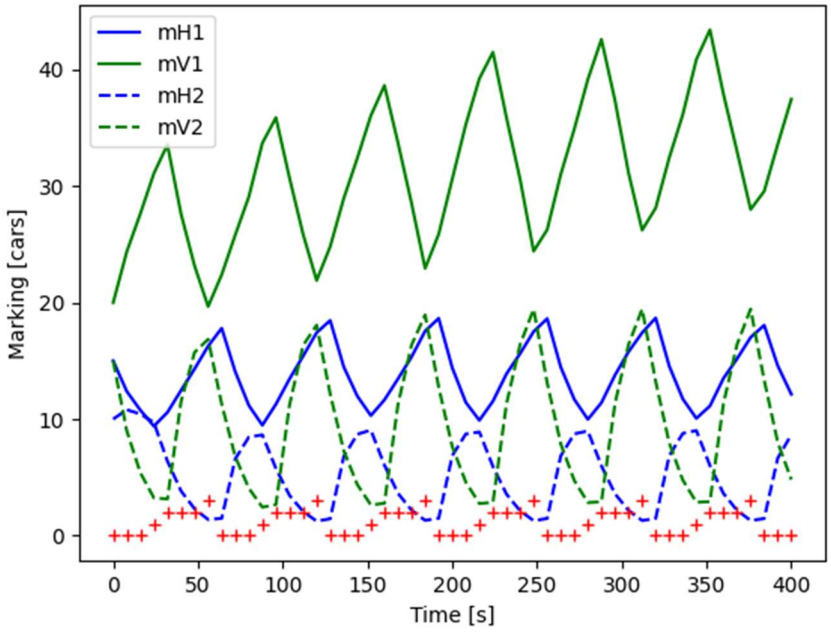


Figura 16. Evolución del modelo de la figura 12. Sin control.

Otro ejemplo aumentando mínimamente la entrada de coches por la sección vertical, pasando de [0.4, 0.6] a [0.5, 0.7] coches por segundo. Se puede ver en la figura 17 como se desestabiliza completamente en poco menos de 5 minutos, lo que derivaría en un atasco, teniendo en cuenta una variabilidad de flujo tan pequeña.

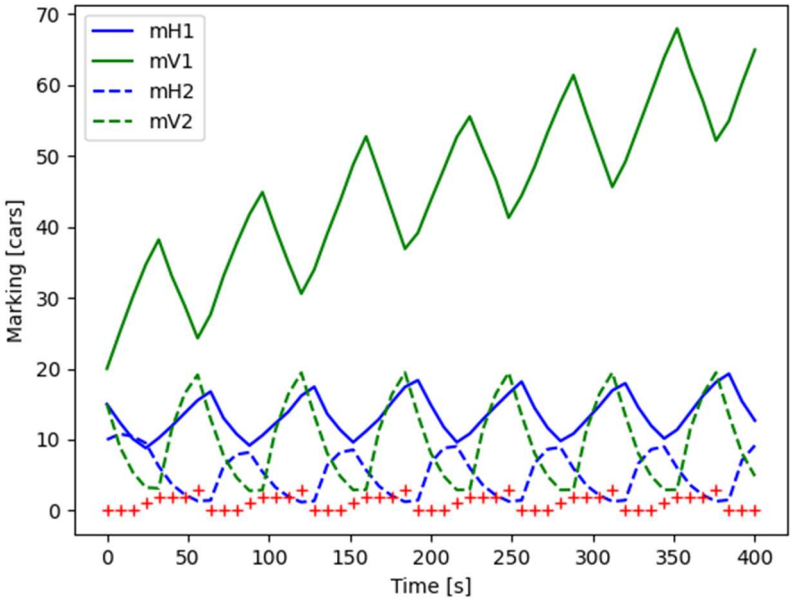


Figura 17. Evolución del modelo de la figura 12. Sin control, aumentado el flujo de entrada.

Control predictivo por modelo (MPC)

Al ser un sistema dinámico tan irregular y poco predecible, se ha decidido implementar un algoritmo de control predictivo por modelo o *Model Predictive Control* [9] ("MPC").

Este modelo de control se basa en que, con el estado actual del sistema, calcula los resultados que dan las diferentes posibilidades que puede haber dentro de un horizonte finito definido y aplica de la mejor posibilidad, solamente durante el primer periodo. Después de aplicar ese primer periodo vuelve a recalcular. La gran virtud de este control es que se anticipa a eventos, como puede ser un cambio en el flujo de entrada debido a las diferentes horas del día o a algún cambio repentino como el de un accidente, y puede tomar acciones para controlar el nuevo sistema.

Para implementar entonces este control al sistema, es necesario definir primero un horizonte de control, que serán 4 periodos, y las siguientes posibles combinaciones que puede llegar a adaptar el sistema (Anexo II), que en el caso de un semáforo, serán 8 viniendo desde verde y 8 viniendo desde rojo. El algoritmo a alto nivel sería:

1. *Estado_Actual = Estado_inicial*
2. *Bucle*
    a. *Calcular los resultados de las combinaciones posibles*
    b. *Guardar la combinación que minimice la función objetivo (maximice el flujo de salida. (Ecuación (12))*
    c. *Aplicar la primera acción de esa combinación a los semáforos*
    d. *Calcular el nuevo estado del sistema (flujos y marcados)*
    e. *Estado_Actual = Nuevo_Estado*
3. *Fin Bucle*

El tiempo de cómputo del modelo con este algoritmo, está alrededor de los 5 segundos, mucho más conveniente.

Ahora, ejecutando este algoritmo con los mismos parámetros que con el tiempo de conmutación de semáforo fijo, se obtiene la figura 18. En ella se observa cómo se auto

regula el marcado entre la carretera horizontal y vertical para que sean casi parejos, teniendo en cuenta que la vertical tiene un flujo de entrada de entre 0.4 y 0.6 coches por segundo y la horizontal de entre 0.2 y 0.3. Como es de esperar, para que esto ocurra, el semáforo de la carretera horizontal (mostrado en la gráfica con la línea roja) es necesario que esté más tiempo en rojo (5 periodos) que en verde (2 periodos).

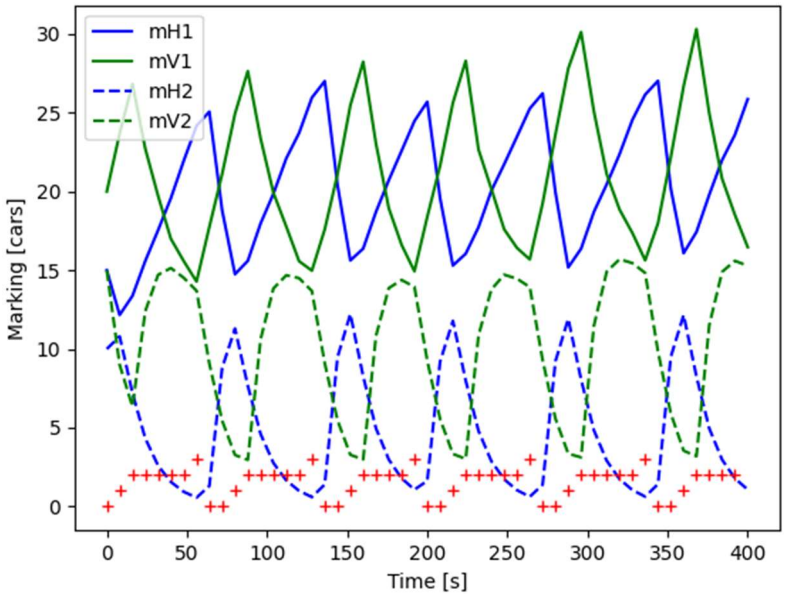


Figura 18. Evolución del sistema de la figura 12. Control con MPC.

Simulando ahora por ejemplo un cambio en el flujo a mitad de la simulación de la entrada de coches de la línea vertical, que pase a ser un flujo constante de 0.3 coches por segundo. Se obtiene el resultado de la figura 19. A partir de los 200 segundos, el algoritmo de control predictivo se adapta y al ser el flujo de entrada horizontal igual al vertical, la secuencia de activación del semáforo se iguala. Aquí se puede apreciar realmente la potencia de este algoritmo.

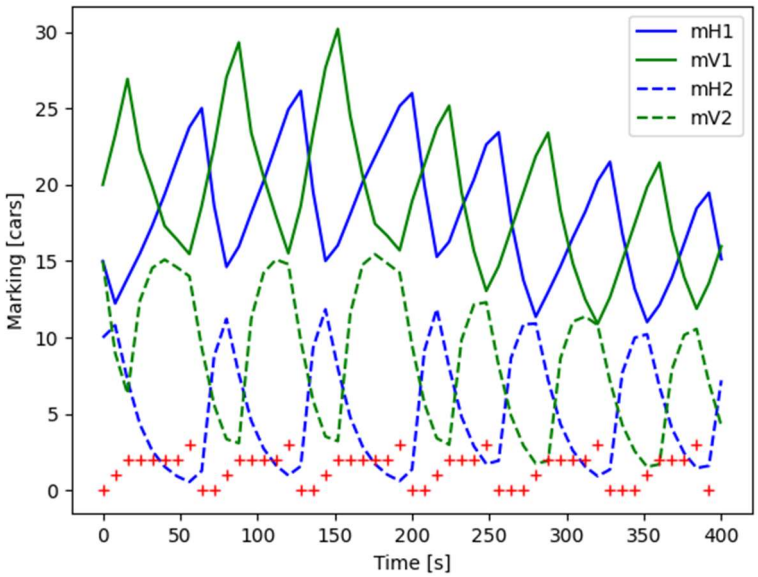


Figura 19. Evolución del sistema de la figura 12. Control con MPC y cambio de flujo de entrada.

Se podrían realizar numerosos experimentos, como programar el algoritmo para un día entero en el que el flujo este programado para variar según la entrada y salida de la jornada laboral, pero el correcto funcionamiento del mismo ya se ha demostrado.

El inconveniente de este algoritmo, es que las posibles combinaciones dentro de las que tiene que calcular la mejor opción, aumentan exponencialmente conforme introducimos nuevas intersecciones, ya que el flujo de la sección posterior afecta a la anterior y por ello hay que tenerlo en cuenta en la combinación. Pasaríamos a tener 64 posibilidades con un horizonte de control de 4 periodos, frente a las 8 anteriores (64 desde verde y 64 desde rojo). No obstante, no es una tarea demasiado compleja computacionalmente y se podría llevar a cabo.

El código de este proyecto está disponible en GitHub. En el propio código está incluida la guía de cómo generar el modelo.

6. CONCLUSIONES Y TRABAJO FUTURO

El principal objetivo de este proyecto se ha alcanzado ya que ha sido capaz de automatizar las frecuencias de activación de los semáforos en función del flujo de entrada. Se ha diseñado un modelo mediante redes flexibles y se ha intentado analizar con fnyzer. No ha sido posible conseguir el control mediante el modelo de las redes flexibles debido a su coste de cómputo, pero se ha ideado otra forma de conseguirlo.

Esta forma de análisis resulta muy competente debido a que se ajusta de una forma muy precisa al diagrama fundamental de tráfico en forma de campana, que era uno de los principales objetivos de este proyecto también.

Con el control predictivo por modelo se ha conseguido que, en una intersección, los semáforos reaccionen a cambios en el flujo de entrada de coches. No se ha podido implementar el análisis de un sistema con más de una intersección, ya que pasaría de tener 8 posibles combinaciones a 64. Sería algo interesante de analizar, aunque tendría la misma estructura de control y por ello y debido al limitado tiempo de realización del proyecto de fin de grado, se ha decidido no abordar.

La implementación de este modelo a la realidad es factible, o al menos la idea de control predictivo por modelo y se podría llevar a cabo de dos formas distintas:

La primera consiste en que se hace un estudio exhaustivo de la entrada de coches en las intersecciones interesadas según la hora del día y posteriormente se introducen en este modelo. De tal forma que se consigue la secuencia de activación óptima según la hora. Esta técnica es la que se usa actualmente.

La segunda, algo más compleja, sería implementar a esta primera la posibilidad de que se regulase de forma automática. Esto quiere decir que, si se pudiera medir el flujo de coches de cada intersección en tiempo real, los semáforos podrían adaptarse a ese flujo si fuera muy diferente al habitual. Se ha visto que el modelo de control es capaz de reaccionar a este tipo de imprevistos.

Esta última opción daría lugar a que cuando se produjesen accidentes de tráfico, durante las vacaciones o durante fines de semana en los que hay mucha congestión de tráfico en una dirección, los semáforos se adaptasen.

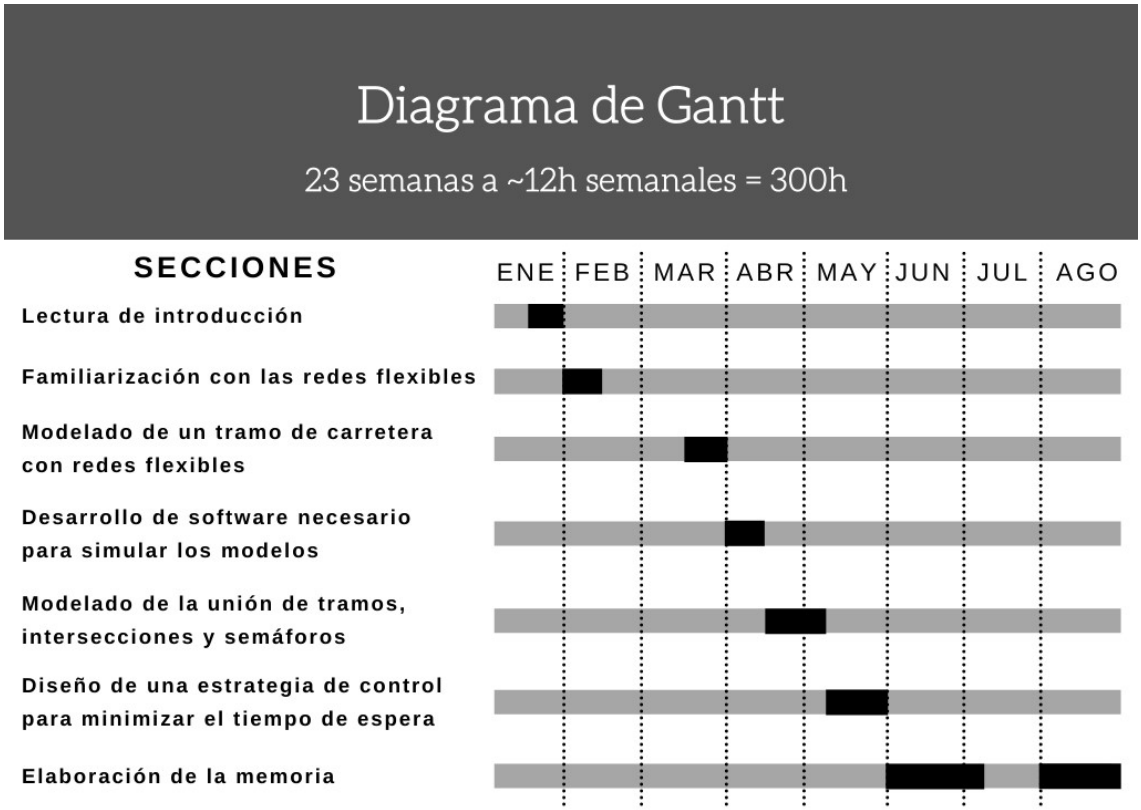
Como conclusión personal, con todo el trabajo realizado he profundizado en los modelos basados en las redes de Petri y redes flexibles, especialmente en los sistemas de tráfico, un

tema en el que tenía especial interés. Además, he descubierto un nuevo algoritmo de control, que es el control predictivo por modelo, aplicable al comportamiento de los sistemas dinámicos complejos. Por último, he profundizado mis conocimientos en Python, uno de los lenguajes más utilizados y potentes de la actualidad.

7. PLANIFICACIÓN TEMPORAL

A continuación, se muestra un diagrama de Gantt con las actividades establecidas para el desarrollo del proyecto, así como la duración de estas.

La duración total del proyecto está estimada en unas 300h, ya que se ha trabajado 12~13h semanales las semanas que están indicadas en el diagrama. He compaginado la realización de este proyecto con el trabajo en empresa a jornada completa, de ahí que se haya demorado más de lo previsto.



8. REFERENCIAS

[1] J. Júlvez y R. Boel, «A continuous Petri net approach for model predictive control of traffic systems,» *IEEE Transactions on Systems, Man, and Cybernetics,* pp. 686-697, 2010.

[2] M. Papageorgiou., «Applications of Automatic Control Concepts to Traffic Flow Modeling and Control,» *Springer,* 1983.

[3] M. Silva, «Introducing Petri Nets. In Practice of Petri Nets in Manufactuing.,» *Chapman & Hall,* pp. 1-62, 1993.

[4] R. D. a. H. Alla, «Discrete, Continuous and Hybrid Petri Nets,» *Springer,* 2004.

[5] L. R. a. M. Silva, «Petri Nets Fluidification revisited: Semantics and Steady state,» *APII-JESA,* pp. 435-449, 2001.

[6] J. J. &. S. G. Oliver, «Flexible Nets: a modeling formalism for dynamic systems with uncertain parameters,» *Springer,* 2019.

[7] J. Júlvez, «Fnyzer,» [En línea]. Available: https://fnyzer.readthedocs.io/en/latest/.

[8] Matplotlib, «matplotlib.org,» [En línea]. Available: https://matplotlib.org/stable/tutorials/introductory/pyplot.html.

[9] J.M.Maciejowski, «Predictive control with constraints,» *Prentice Hall,* 2001.

9. ANEXO I. Uso de Fnyzer y resultados.

A continuación, se muestra el modelo realizado con fnyzer de la conexión de dos secciones de forma directa, sin semáforos ni intersecciones, como lo explicado en la Sección 4. Modelo de un sistema de tráfico con redes flexibles.

Este modelo se ha creado como un diccionario de Python de red flexible, aplicando la estructura definida de la librería de fnyzer. En este diccionario se definen los lugares, las transiciones y los manejadores de eventos y de intensidades con sus desigualdades y ecuaciones. Luego, las características relativas al muestreo con la técnica MPC y se escribe la función objetivo que se quiere alcanzar, esto es, minimizar el marcado de P1.

```
section_two_mpc = {
    'name': 'section_two_mpc',
    'solver': 'gurobi',
    'places': {'P1': 45, 'P2': 1},
    'trans': {'T0': {'l0': 0, 'a0': 0}, 'T1': {'l0': 0, 'a0': 0}, 'T2': {'l0': 0, 'a0':
0}},
    'vhandlers': {
        'v0': [{'a': ('T0', 'v0'), 'x1': ('v0', 'P1')}, 'a == x1'],
        'v1': [{'y1': ('P1', 'v1'), 'z1': ('T1', 'v1'), 'x2': ('v1', 'P2')}, 'y1 == z1',
'y1 == x2'],
        'v2': [{'y2': ('P2', 'v2'), 'z2': ('T2', 'v2')}, 'y2 == z2']
    },
    'regs': {
        'freeflow1': ["m['P1'] <= " + str(endff)],
        'constant1': [str(endff) + " <= m['P1']", "m['P1'] <= " + str(endcf)],
        'stuck1': [str(endcf) + " <= m['P1']"],

        'freeflow2': ["m['P2']<= " + str(endff)],
        'constant2': [str(endff) + " <= m['P2']", "m['P2'] <= " + str(endcf)],
    },
    'parts': {'Part1': ['freeflow1', 'constant1', 'stuck1'],
              'Part2': ['freeflow2', 'constant2']},
    'shandlers': {
        's1': [
            {'b1': ('P1', 's1'), 'c1': ('s1', 'T1'), 'b2': ('P2', 's1'), 'c2': ('s1',
'T2')},
            {'freeflow1': ["c1 == (b1)*" + str(ff1)],
            'constant1': ["c1 == " + str(cf1)],
            'stuck1': ["c1 == " + str(endsf) + "+" + str(stuckslope1) + "*(b2)"],
            'freeflow2': ["c2 == b2*" + str(ff1)],
            'constant2': ["c2 == " + str(cf1)],
            }],

    },
    # 'mbounds': ["m['P1']>=0"],
    'obj': {'f': "m['P1']", 'sense': 'min'},
    'W': 100,
    'wl': 0,
    'wu': 500,
    'options': {
        'antype': 'mpc',
        'mpc': {
            'firstinlen': 0.3,
            'numsteps': 10,
            'maxnumins': 1,
            'flexins': False
        },
        'writevars': {'m': ['P1', 'P2'], 'L': 'all', 'U': 'all'},
        'plotres': True,
        'plotvars': {'evm': ['P1', 'P2']},
        # 'xlsfile': 'seccion.xls',
        'printres': False,
    },
    'actfplaces': 'all',
    'exftrans': 'all',
    'actavplaces': 'all',
    'exavtrans': 'all',
}
```

10. ANEXO II. Modelo programado de un sistema de tráfico

El código referente a lo que se explica a continuación está subido a un repositorio de GitHub para poder ser consultado de manera libre.

La clase *SectionLight* contiene los atributos característicos de una sección de tráfico, esto es, el vector del marcado y del flujo más las constantes inherentes al comportamiento del sistema, q, h, r, maxcar. Además, posee los atributos del semáforo, un vector que almacena todos los estados anteriores del semáforo, α y β y un valor boolean que dice si esa sección tiene o no semáforo.

La unión de estas secciones será lo que conforme nuestra carretera/modelo de tráfico. Estas secciones se almacenarán en un vector que se crea de la siguiente forma:

```
hsections = [Tml.SectionLight(15, True, 3, 3), Tml.SectionLight(10, False, 0, 0)]
```

Se ha creado un vector con dos secciones conectadas.

Se han definido muchos métodos que se pueden encontrar en el repositorio que se ha nombrado anteriormente, voy a explicar los más relevantes:

Calculo del flujo:

Esta función pasa por cada sección del vector *Section*. Primero, se calcula el flujo que tiene la sección según la ecuación 4, posteriormente, si la sección tiene semáforo, se aplica el factor corrector: en rojo flujo=0, en ambar se reduce la velocidad.

```
def new_flow_horizontal_mpc(section, i, lambd, delta, sequence, m):
    for j in range(0, len(section)):
        if j < len(section) - 1:
            f = lambd[j] * min(section[j].m[i+m] / section[j].q, section[j].h, (sec-
tion[j].maxcar - section[j+1].m[i+m]) / section[j+1].r)
            if section[j].tl:
                if sequence[m] == 1:
                    f = f * section[j].alfa / (2 * delta)
                elif sequence[m] == 2:
                    f = 0
                elif sequence[m] == 3:
                    f = f * section[j].beta / (2 * delta)
            section[j].f.append(f)
        else:
            f = lambd[j] * min(section[j].m[i+m] / section[j].q, section[j].h)
            if section[j].tl:
                if sequence[m] == 1:
                    f = f * section[j].alfa / (2 * delta)
                elif sequence[m] == 2:
                    f = 0
                elif sequence[m] == 3:
                    f = f * section[j].beta / (2 * delta)
            section[j].f.append(f)
```

Calculo del marcado:

El marcado se calcula según la ecuación 13. Sobre el vector *Section,* se coge el marcado actual, se resta los coches que se van y se suman los que entran, en el periodo delta.

```
def new_marking_horizontal_mpc(section, inputcars, i, delta, m):
    for j in range(0, len(section)):
        if j == 0:
            section[j].m.append(section[j].m[i+m] + delta * inputcars[m] - delta * sec-
tion[j].f[i+m])
        else:
            section[j].m.append(section[j].m[i+m] + delta * section[j-1].f[i+m] - delta
* section[j].f[i+m])
```

Máquina de estados:

Esta función solamente es útil cuando no se aplica ninguna medida correctiva, esto es, que la secuencia de activación de los semáforos está predeterminada.

Según el estado actual del semáforo, se calcula el próximo estado en función de los periodos que le quedan en ese estado. Por ejemplo, si tiene que estar en verde 4 periodos y lleva 2, el próximo estado será verde.

```
def state_machine(section, i):
    if section.lstate[i] == 0:
        section.cnt += 1
        if section.cnt == section.gg:
            section.cnt = 0
            section.lstate.append(1)
        else:
            section.lstate.append(0)
    if section.lstate[i] == 1:
        section.lstate.append(2)
    if section.lstate[i] == 2:
        section.cnt += 1
        if section.cnt == section.rr:
            section.cnt = 0
            section.lstate.append(3)
        else:
            section.lstate.append(2)
    if section.lstate[i] == 3:
        section.lstate.append(0)
```

Invertir máquina de estados:

Hay una pequeña función para asignar a la otra dirección de la intersección la máquina de estados inversa. Si en la dirección horizontal esta en verde, a la dirección perpendicular se le asignará el estado rojo.

```
def invert_nextstate(i):
    j = 0
    if i == 0:
        j = 2
    elif i == 1:
        j = 3
    elif i == 2:
        j = 0
    elif i == 3:
        j = 1
    return j
```

Control MPC - algoritmo:

Como se ha descrito en la Sección 5. Programación de un sistema de tráfico, se ha implementado la técnica de control MPC:

- Estado actual = Estado inicial.

```
"""CURRENT STATE = INITIAL STATE"""
if i < samples:
    inputV = Tml.inputcars(vinputcars, i)
inputH = Tml.inputcars(hinputcars, i)
sectionH = copy.deepcopy(hsections)
sectionV = copy.deepcopy(vsections)
state = hsections[0].lstate[i-1]
next_state = 0
```

- Bucle:

Calcular los resultados de las combinaciones posibles, guardar la combinación que dé el máximo flujo de salida

```
for j in range(0, len(sequenceH)):
# Iterate among the possible 8 combinations to find the maximum ec 12.
    for m in range(0, len(sequenceH[0])):  # Iteration for the next 4 states
        """FLOW"""
        Tml.new_flow_horizontal_mpc(sectionH, i, hlambda, delta, sequenceH[j], m)
        Tml.new_flow_vertical_mpc(sectionV, i, vlambda, delta, sequenceV[j], m)

        """MARKING"""
        Tml.new_marking_horizontal_mpc(sectionH, inputH, i, delta, m)
        Tml.new_marking_vertical_mpc(sectionV, inputV, i, delta, m)

    """GET FLOW"""
    # appends the flow for each combination so later I know the index of the maxi-
mum flow combination
    flow.append(sectionH[1].f[i + 3] + sectionV[0][1].f[i + 3])
    Tml.remove(sectionH, sectionV)

"""GET THE INDEX FOR THE MAXIMUM FLOW COMBINATION"""
maxflow = max(flow)
index = 0
for j in range(0, len(sequenceH)):
    if flow[j] == maxflow: index = j

"""GET THE FIRST STATE OF THAT SEQUENCE"""
next_state = sequenceH[index][0]
```

Por último, aplicar la primera acción del nuevo estado

```
"""APPLY NEXT STATE"""
hsections[0].lstate.append(next_state)
vsections[0][0].lstate.append(Tml.invert_nextstate(next_state))

"""CALCULATE NEW FLOW"""
Tml.new_flow_horizontal_blind(hsections, i, hlambda, delta)
Tml.new_flow_vertical_blind(vsections, i, vlambda, delta)

"""CALCULATE NEW MARKING"""
Tml.new_marking_horizontal_blind(hsections, hinputcars, i, delta)
if i < samples/2:
    Tml.new_marking_vertical_blind(vsections, vinputcars, i, delta)
else:
    Tml.new_marking_vertical_blind(vsections, vinputcars2, i, delta)
```