DESARROLLO DE MODELOS DE TRÁFICO VEHICULAR EN GALATEA.

Br. Carlos José D. Pérez R.

Trabajo presentado como requisito parcial para optar al título de
LICENCIADO EN FíSICA

UNIVERSIDAD DE LOS ANDES
MÉRIDA, VENEZUELA
Junio 2010

Resumen

El tráfico es un fenómeno que lo vivimos diariamente, pero a pesar de esto no es un fenómeno muy bien comprendido. Las dificultades son múltiples, pero hoy en día ya se cuenta con la información estadística suficiente y con la capacidad de computo necesaria para comenzar a desarrollar modelos realistas y detallados de tráfico vehicular. Con este documento se busca implementar algunos modelos de tráfico que sirvan de prototipo, y así ampliar el espectro de sistemas modelados usando la plataforma de simulación Galatea[12, 13, 14, 15], desarrollada en el Centro de Simulación y Modelos de la Universidad de Los Andes. Se implementaron tres tipos de modelos microscópicos de tráfico y en cada uno de ellos se estudiaron tres escenarios diferentes, un conjunto de modelos base, incorporación a la vía e implementación de un semáforo.

Índice general

Índice de gráficas y figuras

Introducción

Una de las cualidades que se ha observado durante la evolución de las sociedades ha sido el desarrollo de la capacidad de movilidad, hecho que ha incidido directamente en la calidad de vida de sus integrantes. Pero con el incremento poblacional en el mundo comenzaron las dificultades de movilidad que, a su vez, mermaron la libertad y así mismo produjeron efectos nocivos en el normal desenvolvimiento de las actividades socio-económicas. Mediante el desarrollo de sistemas de modelado se han podido realizar estudios que describen algunos fenómenos de tráfico como lo son las ondas de arranque-frenado en una congestión, o la evolución de un sistema en atascos.

Las descripciones matemáticas de la dinámica de flujo de tráfico a nivel científico comienza en los años treinta del siglo pasado con el trabajo de Greenshields[1], donde se estudiaron las relaciones fundamentales entre flujo de tráfico, velocidad y densidad de tráfico. Posteriormente en los años cincuenta se describió la física de propagación de los flujos de tráfico por medio de los modelos de dinámica macroscópica y microscópica [2], pero no fue sino hasta los años noventa que el número de científicos en esta área de investigación comienza a hacerse importante. Actualmente, con el incremento de la cantidad de información estadística del tráfico y del poder de cómputo, es posible simular modelos que preserven muchos de los detalles del tráfico real.

Una de las cualidades que distinguen los diferentes modelos son las escalas de tiempo para las cuales se hace el estudio de la evolución del sistema, esto determinará la relevancia de los aspectos del sistema. El cuadro 1 muestra los rasgos que caracterizan los modelos según su escala de tiempo y los atributos que se le asocian a dichas escalas.

Modelos microscópicos			
Caso	Modelo	Escalas de tiempo	Aspecto
Dinámica de vehículo	Sub-microscópico	0.1 segundo 1 segundo	Cola-carros, frenado reacción y brecha tiempo
Dinámica de tráfico	Carro siguiente	10 segundos	Aceleración y frenado
Modelos macroscópicos			
Caso	Modelo	Escalas de tiempo	Aspecto
Dinámica de tráfico	Dinámica de fluidos	1 minuto 10 minutos 1 hora	período de luz de tráfico ondas de arranque-frenado hora pico
	Asignación de tráfico	1 día	Comportamiento humano día-día
Planificación de transporte	Demanda de tráfico Estadística Pronóstico	1 Año 5 Años 50 Años	Medidas de construcción Cambios espaciales Cambios demográficos

Cuadro 1: Relación Modelos - Escalas de tiempo

Nótese que por un lado tenemos a los modelos macroscópicos en los cuales por lo general hacen uso de videos de tráfico real y de ecuaciones de dinámica de fluidos, es decir, que la dinámica se describe en términos de sus cualidades macroscópicas como densidades, velocidades promedio, etc. Una condición importante que se asume usualmente para estos modelos es la conservación de la cantidad de vehículos, lo cual fue considerado inicialmente en los trabajos de Lighthill y Whitham[3], y de Richards[4]. Otro modelo importante en este particular es el de Prigogine y Herman[5] que se basa en la teoría de ecuaciones cinéticas de gases.

Los modelos de tráfico microscópicos, por otro lado, describen el movimiento de cada vehículo individualmente. Estos modelos se enfocan en eventos de aceleración, desaceleración, cambios de carril y otros; todos ellos como respuesta de cada conductor al tráfico que tiene a su alrededor. Los modelos de tráfico microscópico son especialmente adecuados para el estudio de flujos de tráfico heterogéneos, conformados por tipos individuales de conductores de vehículos. Estos modelos se pueden dividir en tres subclases, los primeros son los modelos de tiempo continuo, formulados con ecuaciones diferenciales ordinarias, donde el espacio-tiempo es tratado como una variable continua. Algunos modelos importantes de este tipo son el del carro siguiente de Wiedemann[6] y el del conductor inteligente de Helbing, Hennecke y Treiber[7]. En segundo lugar tenemos a los modelos de autómatas celulares, que por medio de variables discretas muestran los estados dinámicos del sistema. En ellos el tiempo es discreto y la carretera se divide en celdas, las cuales pueden estar vacías u ocupadas por un vehículo. La primera propuesta en tráfico con autómatas celulares fue planteada con el conocido modelo de Nagel-Schreckenberg[8] que se convirtió en la base para otros modelos discretos, gracias a su simplicidad y capacidad para modelar correctamente los fenómenos de congestionamiento en la carretera. Por último están los modelos de mapas acoplados, que son modelos intermedios entre los continuos y

los de autómatas celulares. En estos modelos el tiempo está discretizado mientras que las coordenadas espaciales son continuas, dos modelos populares de este tipo son el de Gipps[9] y el de Newell[10], y típicamente se les asocia con los modelos del carro siguiente. En este trabajo sólo nos centraremos en los modelos microscópicos, a los que implementaremos utilizando la plataforma de simulación `GALATEA`[12, 13, 14, 15].

Objetivo General

- Desarrollar e implementar modelos para tráfico vehicular unidireccional utilizando tres variantes del enfoque microscópico, es decir, modelos continuos, modelos de mapas acoplados y modelos de autómatas celulares.

Objetivos Específicos

- Desarrollar tres modelos de tráfico vehicular unidireccional, con ecuaciones diferenciales ordinarias, mapas acoplados y de autómatas celulares.
- Implementar los tres modelos en `GALATEA`.
- Realizar algunas simulaciones de cada modelo en tres escenarios diferentes: Un canal unidireccional, un canal unidireccional con semáforo y un canal unidireccional con incorporación de vehículos.

Este manuscrito está estructurado de la siguiente forma; en el capítulo 1 se hace una revisión corta de los modelos más importantes que se han creado para simulación de tráfico vehicular y se presenta brevemente el modelo de dinámica de fluidos con ecuaciones de Euler de Lighthill y Whitham, también se presentan el modelo de conductor inteligente de Wiedeman; el modelo de mapas acoplados de Newell, el modelo de autómatas celulares para tráfico vehicular de Nagel-Schreckenberg y finalmente se dan algunos conceptos básicos de la plataforma de simulación `GALATEA`. En el capítulo 2 se describe cada uno de los tres modelos que se implementaron para esta tesis. Además, se presentan las dos variantes, esto es, la implementación del semáforo y la incorporación de vehículos a la vía. Finalmente, en el capítulo 3 se muestran los resultados junto con algunas mediciones de eficiencia de los modelos base y se hacen algunas comparaciones de los mismos, para culminar el manuscrito con las conclusiones.

Capítulo 1

Marco teórico

Para tener una idea clara y objetiva de algunos aspectos que subyacen en lo que a tráfico vehicular se refire se explicaran algunos modelos de simulación que para efectos de este trabajo considerados de importancia. Por otra parte, la mayoría de estos servirán de base teórica para las simulaciones que se implementarán.

1.1. Modelo macroscópico de Dinámica de fluidos

Este modelo, propuesto por Lighthill, Whitham[3] y Richards[4], toma como base las ecuaciones de Euler de dinámica de fluidos y surge de la idea de que una cantidad grande de vehículos se puede tratar como un continuo. En el modelo no hay choques, tampoco pueden desaparecer ni aparecer de la nada las cantidades. Su dinámica viene dada por

$$\frac{n(x)\partial C(x,t)}{\partial t} + \frac{\partial q(x,t)}{\partial x} = 0 \ , \tag{1.1}$$

donde, $n(x)$ es el número de carriles en la posición x; $C(x,t)$ es la densidad de tráfico en la posición x y en el instante t y $q(x,t)$, es el flujo de tráfico de vhículos dada por

$$q(x,t) = C(x,t)v(x,t)n(x) \ , \tag{1.2}$$

donde, $v(x,t)$ es la velocidad del flujo. Tanto Lighthill y Whitham[3] como Richards[4] han propuesto que la velocidad del flujo sea una función de la densidad de tráfico de la forma

$$v(x,t) = F(C(x,t)) \ . \tag{1.3}$$

En la práctica, este tipo de modelos generalmente son discretizados en el tiempo y el espacio. La discretización en el tiempo se hace al considerar pasos de tiempo $\triangle t$, y la

discretización en el espacio se hace con un paso de integración $\triangle x$. Para garantizar la estabilidad numérica de las soluciones se debe cumplir que $\triangle x \gg v\triangle t$. Por su forma de ecuación diferencial en derivadas parciales es posible modelar sistemas de muchos vehículos; pero, por esta misma condición en estos modelos no se puede seguir el estado de cada vehículo del sistema.

1.2. Modelo microscópico del conductor Inteligente

Este modelo, desarrollado por Helbing, Hennecke y Treiber[7], se caracteriza por establecer para cada vehículo un sistema de ecuaciones diferenciales, que controlan la aceleración y el frenado, dado por

$$\begin{cases} \dfrac{dv_i}{dt} = a_i \left[1 - \left(\dfrac{v_i}{v_i^*} \right)^{\delta} - \left(\dfrac{s_i^*}{s_i} \right)^2 \right] \\ \\ \frac{dx_i}{dt} = v_i \end{cases} , \tag{1.4}$$

donde v_i es la velocidad y a_i es la aceleración máxima del i-esimo vehículo, x_i su posición, v_i^* es su velocidad deseada, el exponente δ determina la proporción de agresividad o timidez de los conductores, s_i^* es la brecha deseada. La brecha entre el vehículo i y el anterior a él es s_i y se define como,

$$s_i = x_j - x_i \ , j \ \ \forall \ \ x_j \ \ min(x_k / x_k > x_i) \tag{1.5}$$

donde, la brecha deseada s_i^* es,

$$s_i^*(v_i, \Delta v) = s_0 + v_i T + \frac{v_i(v_i - v_{i-1})}{2\sqrt{a_i b_i}} \ , \tag{1.6}$$

donde, s_0 es la brecha mínima entre vehículos a velocidad cero, T es la brecha de tiempo deseada, b_i es la des-aceleración cómoda que una fracción de la aceleración máxima a_i, del sistema 1.4 se puede extraer que la estrategia de aceleración cuando la carretera está libre, es

$$\frac{dv_l}{dt} = a_i \left[1 - \left(\frac{v_i}{v_i^*} \right)^{\delta} \right] \ , \tag{1.7}$$

y la estrategia de frenado,

$$\frac{dv_f}{dt} = -a_i \left[\left(\frac{s_i^*}{s_i} \right)^2 \right] . \tag{1.8}$$

1.3. Modelo microscópico con redes de mapas acoplados

Este modelo, propuesto por Newell[10], incorpora ecuaciones de tiempo discreto y estados continuos. La dinámica de cada vehículo se describe mediante el sistema de dos mapas acoplados

$$\begin{cases} v_i(t + \Delta t) = G_i(s_i(t), v_i(t)) \\ \\ x_i(t + \Delta t) = v_i(t)\Delta t + x_i(t) \end{cases} , \tag{1.9}$$

donde $v_i(t)$ y $x_i(t)$ son la velocidad y posición del i-esimo vehículo en el instante t respectivamente, Δt es el paso de tiempo, G_i es una función que depende de la velocidad y la brecha entre vehículos s_i, y de la cual hay varias proposiciones en el artículo de Newell[10].

1.4. Modelo con autómatas celulares

Propuesto por primera vez por Nagel-Schreckenberg[8], el modelo es un autómata celular en una dimensión, donde cada celda puede tomar un valor binario, uno si hay un vehículo en ella y cero en caso contrario. Suponiendo que los vehículos se mueven en una carretera de un solo canal, de izquierda a derecha, la evolución de los estados en el modelo se rige por la regla 184 de Wolfram[11]; dado por

$$p_i(t+1) = (p_i(t) \ \wedge \ p_{i+1}(t)) \vee (p_{i-1}(t) \ \wedge \neg \ p_i(t)) \ , \tag{1.10}$$

donde $p_i(t)$ es el estado de la celda i-esima en el instante t.

1.5. Algunos conceptos importantes

`GALATEA` es una familia de lenguajes, compiladores e interpretadores, que contiene distintas herramientas de simulación entre las que se tiene un integrador de ecuaciones diferenciales ordinarias, un simulador `DEVS`[16],y una colección de bibliotecas de

apoyo y plantillas genéricas para el desarrollo de modelos de simulación de eventos discretos, sistemas continuos y sistemas multi-agentes que permite explorar alternativas para la integración con un formalismo general de modelado y simulación[14]. Las simulaciones con `GALATEA` se implementan a través de nodos; en un sistema de modelado un nodo es un subsistemas que intercambian mensajes que en nuestro caso serán los vehículos. Cada subsistema puede almacenar, transformar, transmitir, crear y eliminar mensajes del sistema, a través de un código enlazado. Existen distintos tipos de nodos según su función y nombraremos los nodos usados para las distintas simulaciones implementadas en este trabajo, que son: los nodos tipo `Gate (G)` que controlan el flujo de mensajes, los tipo `Input (I)` que generan mensajes de entrada, los tipo `Resource (R)` que simulan recursos usados por los mensajes, los nodos `Exit (E)` que son los nodos que destruyen los mensajes, los nodos `Autonomous (A)` que no recibe ni envía mensajes, y en su código asociado se permite programar eventos[12], estos nodos junto con el paquete `gSpace`, el cual contiene los mensajes, recrea el espacio (la carretera) con sus cualidades físicas, y de acuerdo con estas características es capaz de reproducir eventos discretos con estados continuos. La herramienta permite programar una función de movimiento (`Regla`) para la interacción de los mensajes que se encuentran en el o los distintos espacios, los mismo se estructuran con paredes y puertas que los comunican; éstos son modificables en tiempo de ejecución, y están compuestos por planos continuos [15].

Capítulo 2

Implementación de los modelos y algunos escenarios

En este capítulo nos enfocaremos en las especificaciones de funcionamiento de cada uno de los distintos ejemplos que se han desarrollado, desglosándolos con la mayor sencillez posible para describir su funcionamiento en base a las distintas clases que los componen. Los modelos de simulación para tráfico vehicular desarrollados podrán ser explotados para tratar problemas reales, aún cuando se trata de modelos de ejemplo en un canal. Se desarrollaron tres ejemplos base: autómatas celulares, mapas acoplados y ecuaciones diferenciales; esto persiguiendo las siguientes metas: los ejemplos deben funcionar sobre la plataforma `GALATEA`, utilizando varias de sus herramientas; probar que la plataforma de simulación es capaz de sustentar los modelos de tráfico de manera eficiente; y permitir la comparación de eficiencia individual de cada uno de los modelos.

2.1. Detalles de los modelos base

Es intuitivo imaginar el modo como se trasladan los vehículos en una carretera de un solo canal como la que se muestra en la figura 2.1; por ejemplo, los vehículos no se adelantan pues físicamente es imposible. A partir de ésta y otras ideas se construye el diseño que sustenta los modelos. En general, los vehículos en la carretera se comportan de la siguiente forma:

- Una vez que los vehículos se incorporan en la vía conservaran el orden en que entran a la carretera hasta el momento de la salida, el vehículo i-esimo siempre será el siguiente del $(i-1)$-esimo.

- Los vehículos ocupan un área determinada y, por tanto, una posición en el espacio sólo puede ser ocupada por un vehículo.
- Los vehículos deben tener aceleración, velocidad, y capacidad de frenado finita.
- En los modelos no habrá estudio de colisiones ni se hará simulaciones que las incluyan.
- La carretera es unidireccional, por lo que todos los vehículos se movilizan de izquierda a derecha.
- Los vehículos guardan una distancia mínima s_i^* entre sí que no dependerá directamente del modelo implementando.
- La posición, velocidad y aceleración de cada uno de los vehículos son individuales, es decir, el vehículo i tendrá posición x_i, velocidad v_i y aceleración a_i.

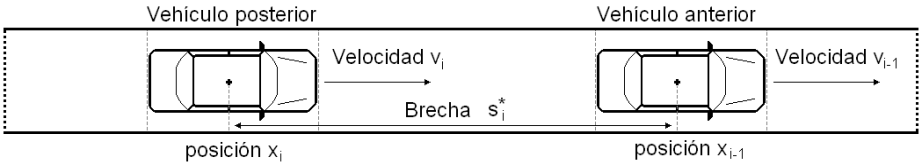


Figura 2.1: Ejemplo del sistema a modelar.

2.2. Modelo general

En síntesis todos los modelos siguen una estructura básica, y están compuestos por los nodos `entrada`, que es un nodo **I**; `wait, move` y `door`, que son los tres que conforman la vía unidireccional definida por `gSpace` y finalmente el nodo `salida` que es de tipo **E**. La figura 2.2 muestra en forma esquemática al modelo El modelo

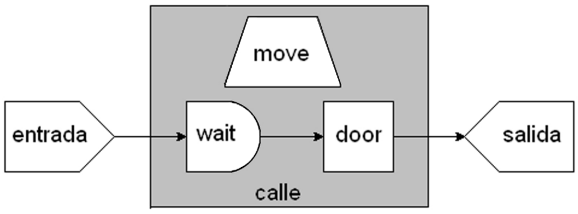


Figura 2.2: Esquema del modelo base en Galatea.

está compuesto por las siguientes clases:

Código 1 Clase `Via.java`. Clase principal del modelo.

```
import galatea.glider.*;
import galatea.gspaces.*;
public class Via {
  static double l;
  static Entrada entrada = new Entrada();
  static Space calle = new Space("calle",0,6.0,6.0,1.0,1.0,"Regla");
  static Salida salida = new Salida();
  public static void main(String[] args) {
      setExperiment(args);
      calle.addWall(0.0, 0.0, l, 0.0);
      calle.addWall(0.0, 12.0, l, 12.0);
      calle.addWall(0.0, 0.0, 0.0, 12.0);
      calle.addWall(l, 0.0, l, 12.0);
      calle.addDoor(l, 0.0, l, 12.0, salida, 'U', 1, 0.0, 0.0);
      calle.build();
      Glider.trace("Via.trc");
      Glider.stat("Via.sta");
      Glider.setTitle("Via");
      Glider.setTsim(10000);
      Glider.act(entrada, 0);
      Glider.act(calle.getMove(), 1);
      Glider.process();
  }
  static void setExperiment(String[] args) {
      args = Galatea.checkArgs(args);
      l = Galatea.doubleArg(args, "Longitud", "01000.0");
       Glider.addPath(Galatea.getExperDir());
  }
}
```

`Via:` Como se muestra en el código 1, ella es la clase principal y allí se instancian los nodos `entrada` y `salida` que crean y eliminan respectivamente a los mensajes (vehículos) del sistema. También, se instancia al espacio, `calle`, donde circulan los mensajes. La clase tiene dos métodos `main` y `setExperiment`. El primero es el método principal, que se invoca al ejecutar la clase. Es el encargado de: llamar al método `setExperiment`, que es el que procesa los parámetros con los que se correrá el modelo; establecer los límites de espacio `calle` e iniciar la simulación estableciendo, entre otras cosas, el tiempo de simulación, el nombre de los archivos de salida y las primeras activaciones de nodos.

`Entrada:` Extiende al nodo Galatea tipo **I**. Como se muestra en el código 2, ésta clase implementa al constructor y al método `fact` que programa su próxima activación, asignan los valores iniciales a los campos del los mensajes (vehículos) y por último, envía al mensaje recién creado al espacio `calle`. Entre los campos tenemos a las posiciones `x` y `y`, la velocidad deseada `v0`, y las velocidades iniciales en cada eje `vvx` y `vy`.

Código 2 `Entrada.java`. Crea los mensajes y se inicializa sus campos.

```
import galatea.glider.*;
import galatea.gspaces.*;
class Entrada extends Node {
    Entrada() {
        super("entrada", 'I');
        Glider.nodesl.add(this);
    }
    public void fact() {
        it(GRnd.unif(1, 4));
        Glider.mess.addField("x", 0.1);
        Glider.mess.addField("y", Via.calle.getYIn());
        Glider.mess.addField("v0", 12);
        Glider.mess.addField("tU",Glider.mess.getNumber()%2);
        Glider.mess.addField("p", null);
        Glider.mess.addField("c", 0);
        Glider.mess.addField("s", 0);
        Glider.mess.addField("d", null);
        Glider.mess.addField("cut", null);
        Glider.mess.addField("vvx", 0);
        Glider.mess.addField("vy", 0);
        /* ****************************************** */
        /* ACA VA EL CODIGO PARTICULAR DE CADA MODELO */
        /* ****************************************** */
        GSpace.sendto(Glider.mess, this, Via.calle);
    }
}
```

`Salida:` Extiende al nodo Galatea tipo **E**. En esta clase, que se muestra en el código 3, se implementa su constructor y el método `fscan`, que de ser necesario puede procesar al mensaje saliente.

Código 3 `Salida.java`. Eliminan los mensajes que salen del sistema.

```
import galatea.glider.*;
class Salida extends Node {
    Salida() {
        super("salida", 'E');
        Glider.nodesl.add(this);
    }
    public void fscan() {
        /* ************************************** */
        /* ACA VA EL CODIGO PARTICULAR DE CADA MODELO */
        /* ************************************** */
    }
}
```

`Regla:` Es la clase donde se implementan las reglas de movimiento de cada modelo. En general tiene la forma que se muestra en el código 4, pero el cálculo de las nuevas posiciones y velocidades de cada mensaje dependerá de cada modelo en particular. La clase `Regla` se asocia a espacio `calle` al momento de instanciarlo en la clase `Via` como se muestra en el código 1.

Código 4 `Regla.java`. Implementa el movimiento en cada modelo.

```
import galatea.glider.*;
import galatea.gspaces.*;
public class Regla {
    public Regla(){}
    public double[] move(Move e, Message m, Cell c) {
        double[] nCoord = new double[4];
        /* ************************************** */
        /* ACA VA EL CODIGO PARTICULAR DE CADA MODELO */
        /* ************************************** */
        nCoord = e.dChecking(m, c, nCoord, e.getDoor(1));
       return nCoord;
    }
}
```

2.2.1. Simulación con autómatas

En los modelos con autómatas la vecindad es factor determinante para realizar movimientos en el espacio que está compuesto por celdas. Para cada instante discreto

de tiempo los vehículos solamente se desplazaran a una nueva posición si se cumplen las condiciones para que esto ocurra.

En este modelo la posición a la que tendrá acceso cada vehículo estará limitada por su velocidad v_i para un intervalo de tiempo $it = 1$ y su posición en el instante anterior, lo que se puede escribir como

$$x_i(t + it) = x_i(t) + v_i(t + it) \ , \tag{2.1}$$

En esta simulación la única `clase` que cambia con respecto al modelo general de la sección 2.2 es `Regla`, por lo que sólo se hará la descripción de la misma.

`Regla:` Como se muestra en el código 5, para cada iteración se llama al método `desplazamiento`, código 6, previa declaración de la variable `lastUpdate`, para luego cambiar la posición del vehículo según la ecuación (2.1) expresada en la actualiza-

Código 5 `Regla.java` Implementación del movimiento con Autómatas

```
if(lastUpdate < Glider.getTime()){
    lastUpdate=Glider.getTime();
    desplazamiento();
}
double[] nCoord = new double[4];
nCoord[0] = m.getdoubleValue("x")+ m.getdoubleValue("vvx");
nCoord[1] = m.getdoubleValue("y");
```

ción de `nCoord[0]`. El método `desplazamiento` llamado en `Regla` revisa todas las celdas del espacio, y si encuentra una celda que no está vacía, es decir, que contenga un mensaje, y mientras dicho mensaje tenga una velocidad `vx1` menor que la velocidad `v0`, se añade una unidad a `vx1` y se revisa si las celdas correspondientes al desplazamiento producido por la nueva velocidad `vx1` se encuentran vacías, se hace la actualización del campo de velocidad del mensaje, y el mensaje avanza. El método `desplazamiento` regula la velocidad máxima v_0 al que los vehículos se moverán siguiendo la regla general de autómatas celulares.

2.2.2. Simulación con mapas acoplados

Éste modelo se encuentra compuesto por un sistema de dos mapas acoplados, el de posición x y el de velocidad v, el tiempo es discreto y los estados del sistema son

Código 6 `desplazamiento`. Método declarado en la clase `Regla.java`.

```
void desplazamiento(){
    Cell[] c = Via.calle.getMove().getCells();
    Message m;
    int i;
    int v0;
    int vx;
    int vx1;
    for(i=0;i<c.length;i++){
        if(!c[i].isEmpty()){
            m = (Message)(c[i].getAg().getDat(1));
            v0 =m.getintValue("v0");
            vx = m.getintValue("vvx");
            vx1 = 0;
            while(vx1 < v0 && vx1 < vx + 1){
                if(c[i + vx1 + 1].isEmpty())
                    vx1++;
                else
                    break;
            }
            m.setField("vvx", vx1);
        }
    }
}
```

continuos, dichos mapas se escriben mediante el siguiente sistema de ecuaciones;

$$\begin{cases} x_i(t+it) = x_i(t) + v_i(t)\ it + \frac{a_i(t+it)\ it^2}{2} \\ \\ v_i(t+it) = a_i(t+it)\ it + v_i(t) \end{cases} , \tag{2.2}$$

de donde $x_i(t)$, $v_i(t)$ y a_i son la posición, la velocidad y la aceleración respectivamente del i-esimo vehículo, it es tiempo entre iteraciones, además a_i se define como;

$$a_i(t+it) = a_i^* \left[1 - \left(\frac{v_i(t)}{v_i^*} \right)^2 - \left(\frac{s_i^*(t)}{s_i(t)} \right)^2 \right] , \tag{2.3}$$

donde a_i^* es la aceleración deseada, v_i^* es la velocidad deseada, s_i es la brecha actual y se escribe;

$$s_i(t) = x_{i-1}(t) - x_i(t) . \tag{2.4}$$

y s_i^* es la brecha deseada, que se escribe,

$$s_i^*(t) = s_0 + v_i T + \frac{v_i(v_i - v_{i-1})}{2\sqrt{a_i^* b}} , \tag{2.5}$$

donde s_0 es la brecha mínima entre vehículos a velocidad cero y b es la des-aceleración deseada que es directamente proporcional a la aceleración deseada. Para construir la simulación con mapas acoplados se hacen las siguientes modificaciones del modelo base;

`Regla:` En primer lugar el método `move` de `Regla` actualiza todas las velocidades de los vehículos invocando al método `desplazamiento`; que mediante el uso de la varia-

Código 7 `Regla.java`

```
if(lastUpdate < Glider.getTime()){
    lastUpdate=Glider.getTime();
    desplazamiento();
}
nCoord[0] = m.getdoubleValue("x") + m.getdoubleValue("vvx") * e.getIt()
                                  + (1/2) * m.getdoubleValue("a")
                                  * Math.pow(e.getIt(),2);
nCoord[1]=m.getdoubleValue("y");
```

ble `lastUpdate` solamente es invocado una vez en cada iteración. Luego se actualiza `nCoord[0]` según la implementación en el código 7 de la ecuación (2.2.a). El método `desplazamiento` tal y como se muestra en el código 8 en primer lugar recorre la lista correspondiente al espacio `calle` para ubicar los mensajes en ésta. Luego para cada mensaje encontrado con `anterior` no nulo se calcula la brecha entre vehículos `g` como se ve en la figura 2.3. Si la brecha es mayor a `g0 + 10`, siendo `g0` la brecha mínima

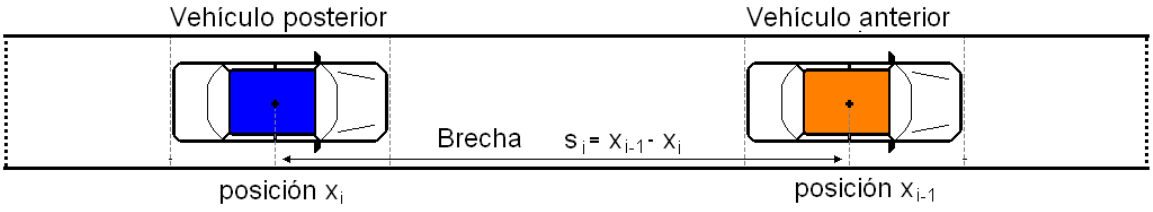


Figura 2.3: Esquema del modelo base en Galatea.

entre carros a velocidad cero, entonces se calcula la brecha deseada `gg` y la aceleración `a` para el vehículo, En caso de que no haya vehículo `anterior` simplemente se calcula la aceleración para un vehículo libre en el vía, la aceleración resultante se actualiza en el campo a la misma, se calcula la velocidad nueva del vehículo con la implementación en el código de la ecuación (2.2.b) y finalmente se actualiza el campo de velocidad.

Código 8 desplazamiento.

```
void desplazamiento(){
    double t = Via.calle.getIt();
    for(int n=1; n<=Via.calle.getWait().getEl().ll();n++){
      Message m = (Message)Via.calle.getWait().getEl().getDat(n);
      Message ant = (Message)m.getValue("anterior");
      double v_m = m.getdoubleValue("vvx");
      double T = m.getdoubleValue("T");
      double b = m.getdoubleValue("b");
      double a0 = m.getdoubleValue("a0");
      double x_m = m.getdoubleValue("x");
      double v0 = m.getdoubleValue("v0");
      double g0 = m.getdoubleValue("g0");
      double stop =m.getdoubleValue("g0")+ 10;
      double a = m.getdoubleValue("a");
      if(ant!=null){
        double x_ant = ant.getdoubleValue("x");
        double v_ant = ant.getdoubleValue("vvx");
        double g = x_ant - x_m;
        if(g>stop){
          double gg =g0+v_m*T+v_m*((v_m-v_ant)/(2*Math.sqrt(a0*b)));
          a = a0*(1-Math.pow((v_m/v0),2)-(Math.pow((gg/g),2)));
        }else{
          if(ant.getdoubleValue("vvx")!=0){
              a = 0;
          }else{
              a = 0;
              v_m = 0;
          }
        }
      } else {
        a = a0*(1-Math.pow((v_m/v0),2));
      }
      if(a >= 0)
        m.setField("a", a);
      vx = a*t + v_m;
      m.setField("vvx", vx);
    }
}
```

`Entrada:` En éste se añaden varios campos, como muestra el código 9 y se declara la variable `Message ant = null` para inicializarla, luego, si `anterior` no es nulo se inicializa el campo `posterior` para este mensaje.

Código 9 `Entrada.java` Extracto del método `fact` para el modelo de mapas acoplados.

```
Glider.mess.addField("a", 0.0);
Glider.mess.addField("b",0.9);
Glider.mess.addField("a0",11.1);
Glider.mess.addField("g0",4.0);
Glider.mess.addField("T",1.5);
Glider.mess.addField("anterior",ant);
if(ant!=null)
    ant.addField("posterior", Glider.mess);
ant = Glider.mess;
```

`Salida:` En el método `fscan` se anula el atributo `anterior` del mensaje `posterior` al mensaje que está saliendo como se muestra en el código 10. Para ésta simulación

Código 10 `Salida.java` Eliminación del `anterior` en `posterior`

```
Message post = (Message)(Glider.mess.getValue("posterior"));
if(post != null){
    post.setField("anterior", null);
```

el código de `Via.java` es idéntico al del modelo general de la sección 2.2.

2.2.3. Simulación con el conductor inteligente

Éste es un modelo de espacio-tiempo continuo, el cual actualiza la posición y velocidad de cada vehículo resolviendo un sistema de ecuaciones diferenciales ordinarias. Las ecuaciones diferenciales se resuelven por medio del integrador de `GALATEA`. En la simulación implementada se usa el siguiente sistema,

$$\begin{cases} \frac{dx_i}{dt} = v_i \\ \\ \frac{dv_i}{dt} = a_i^* \left[1 - \left(\frac{v_i}{v_i^*} \right)^2 - \left(\frac{s_i^*}{s_i} \right)^2 \right] \end{cases} , \tag{2.6}$$

donde x_i es la posición del vehículo i, v_i es su velocidad y depende de la solución de la ecuación 2.6.b, a_i^* es su aceleración deseada, v_i^* su velocidad deseada, s_i^* su brecha deseada y s_i la brecha entre el i-esimo vehículo y su anterior. Las `clases` para este modelo toman la siguiente forma:

`Regla:` En esta clase se instancia el sistema de ecuaciones diferenciales ordinarias `Odes`, luego se le asigna como paso de integración de las ecuaciones diferenciales al tiempo entre iteraciones del espacio, posteriormente se resuelven las ecuaciones diferenciales para el tiempo que lleva la simulación mas el `it`; y por último se asigna la primera solución al campo de velocidad representado por `vvx` y se asigna la segunda solución al campo `nCoord[0]` que finalmente actualiza la posición del mensaje, todo esto como se muestra en el código 11.

Código 11 `Regla.java` Actualización de la velocidad y la posición según las soluciones de las ecuaciones diferenciales.

```
Odes ode = (Odes)m.getValue("ode");
ode.setStep(e.getIt());
ode.solve(Glider.getTime() + e.getIt());
m.setField("vvx", ode.getState(0));
nCoord[0] = ode.getState(1);
nCoord[1] = m.getdoubleValue("y");
```

`Entrada:` En este nodo de añaden nuevos atributos al mensaje entre los que se el `anterior`, `posterior` de `anterior`, los campos de actualización de las ecuaciones

Código 12 `Entrada.java`

```
Glider.mess.addField("a", 2.8);
Glider.mess.addField("b",0.9);
Glider.mess.addField("g", 0);
Glider.mess.addField("g0", 10.0);
Glider.mess.addField("T",1.5);
Glider.mess.addField("anterior",ant);
if(ant != null )
    ant.addField("posterior", Glider.mess);
Odes ode = new Odes(true, 0.0, 0.01);
ode.addEqs(new vOde("vvx",0.0,Glider.mess));
ode.addEqs(new xOde("x",Glider.mess.getdoubleValue("x")));
Glider.mess.addField("ode",ode);
ant = Glider.mess;
```

diferenciales como se ve en el código 12 y las clases instanciadas de `Odes` que contiene a

las dos encargadas de obtener las ecuaciones diferenciales velocidad y posición según el sistema ecuaciones (2.6). El código 13 muestra la implementación de estas ecuaciones.

`Salida:` Como en el modelo de mapas acoplados, en salida una vez que entra un mensaje y éste tiene `posterior`, es anulado el campo `anterior` en `posterior`, esto se muestra en el código 10. En ésta simulación el código `Via.java` es igual al del modelo general presentada en el código 1.

2.3. Algunas variantes de interés

En esta sección nos enfocaremos en examinar dos variantes que hemos considerado trascendentes para las simulaciones de tráfico vehicular. Por una parte, la incorporación de vehículos en la vía, cuya descripción caracteriza una condición que es natural. Además también se implementará el escenario de un semáforo en la vía. Aunque no se profundizará en estos, se dejará con esta contribución un terreno el cual puede ser estudiado posteriormente.

2.3.1. Incorporación de vehículos

Esta variante mostrada en la figura 2.4 añade a la simulación un nodo de `Incorporacion`, y se modifica el nodo de entrada para enviar mensajes en una nueva ubicación y con una taza de activación independiente a los mensajes entrantes al comienzo de `Via`. Entre los cambios hechos en el código tenemos:

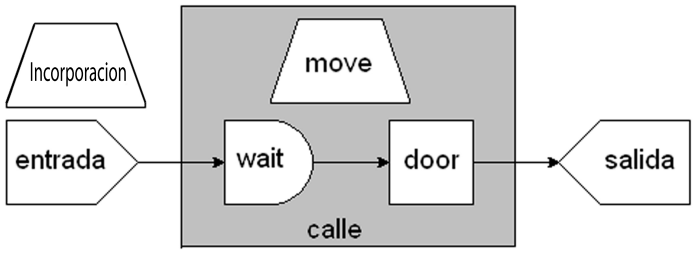


Figura 2.4: Como se observa contiene `Incorporacion` como nuevo nodo.

`Entrada:` Tal y como se muestra en el código 14, este nodo añade al nuevo mensaje un grupo de atributos que representa el grupo de mensajes que se incorpora a `Via`, está parte del código se ejecuta según lo indica el valor de `Incorporacion.activa` controlado en el nodo `Incorporacion`. Además cambia la etiqueta `anterior` del `posterior` por la etiqueta del mensajes entrante y la etiqueta `posterior` del `anterior`

Código 13 `vOde` y `xOde`. Acá se muestran los integradores.

```
        class vOde extends Cont {
         Message m;
            vOde(String l, double ic,Message m){
                super(l,ic);
                this.m = m;
            }
            public double feval(double x, Vector y){
                double T = m.getdoubleValue("T");
                double b = m.getdoubleValue("b");
                double g0 = m.getdoubleValue("g0");
                double v0 = m.getdoubleValue("v0");
                double v_m = m.getdoubleValue("vvx");
                double a = m.getdoubleValue("a");
                Message ant = (Message)m.getValue("anterior");
                double x_m = m.getdoubleValue("x");
                double x_ant;
                double v_ant;
                double g;
                double gg;
                double Vode = 0;
                if (ant != null) {
                    x_ant = ant.getdoubleValue("x");
                    g = x_ant - x_m;
                    v_ant = ant.getdoubleValue("vvx");
                    gg = g0 + v_m * T + v_m * ((v_m - v_ant) / (2 * Math.sqrt(a * b)));
                    Vode = a * (1 - Math.pow((v_m / v0), 2) - (Math.pow((gg / g), 2)));
                } else {
                    Vode = a * (1 - Math.pow((v_m / v0), 2));
                }
                return Vode;
            }
        }
        class xOde extends Cont {
            Message m;
            xOde(String l, double ic, Message m){
                super(l,ic);
                this.m=m;
            }
            public double feval(double x, Vector y){
             double Xode = m.getdoubleValue("vvx");
                return Xode;
            }
        }
    }
```

por la etiqueta de éste mismo mensajes entrante, esto último para ubicar el mensaje que se incorpora en orden a la lista de los mensajes de `Via`.

Código 14 `Entrada.java` Nuevo código.

```
        if(Incorporacion.activa){
            Incorporacion.activa=false;
            Glider.mess.addField("x", Via.l/2);
            Glider.mess.addField("y",Via.calle.getYIn());
            Glider.mess.addField("v0", 5.0);
            Glider.mess.addField("tU",0);
            Glider.mess.addField("p", null);
            Glider.mess.addField("c", 3);
            Glider.mess.addField("s", 0);
            Glider.mess.addField("d", null);
            Glider.mess.addField("cut", null);
            Glider.mess.addField("vvx", 0.0);
            Glider.mess.addField("vy", 0.0);
            Glider.mess.addField("a", 2.8);
            Glider.mess.addField("b",0.9);
            Glider.mess.addField("g", 0);
            Glider.mess.addField("g0", 10.0);
            Glider.mess.addField("T",1.5);
            Message p = Via.incorporacion.getPosterior();
            if(p != null){
                Message a = (Message)(p.getValue("anterior"));
                Glider.mess.setField("anterior",a);
                Glider.mess.setField("posterior", p);
                p.setField("anterior", Glider.mess);
                a.setField("posterior", Glider.mess);
            }
    } else {
            /* ****************** */
            /* ACA VA CODIGO BASE */
            /* ****************** */
        }
```

`Via.` En ésta clase se hace la instanciación del nodo `Incorporacion` y también la primera activación de éste mismo nodo, así como se muestra en el código 15.

`Incorporacion:` Este nodo es de tipo **A** e implementa el método `fact`. Este último, aunque conceptualmente tiene la misma función para todos los modelos, para el modelo de autómatas celulares se implementa con algunas ligeras diferencias para conservar el paradigma de autómata celular. Para los modelos de mapas acoplados y conductor inteligente, la implementación se muestra en código 16. En ambos casos en primer lugar se determina que hay suficiente espacio para que se de la incorporación. De ser este el caso se activa el nodo `Entrada` con el atributo `Incorporación.activa` en verdadero de no ser suficiente el espacio se re-programa la activación del nodo hasta lograr la incorporación del vehículo.

Código 15 `Via.java` Instanciación y activación del nodo `Incorporacion`.

```
static Incorporacion incorporacion = new Incorporacion();
    /* ****************** */
    /* ACA VA CODIGO BASE */
    /* ****************** */
    Glider.act(incorporacion, 60);
```

Código 16 `Incorporacion.java` Código para mapas acoplados y conductor inteligente.

```
import galatea.glider.*;
import galatea.gspaces.*;
class Incorporacion extends Node {
   static boolean activa;
   Message posterior;
   Incorporacion() {
      super("incorporacion", 'A');
      activa=false;
      Glider.nodesl.add(this);
   }
   public void fact(){
      posterior = null;
      Message m;
      int ll = Via.calle.getWait().getEl().ll();
      double x;
      for(int n=1; n<=ll;n++){
         m = (Message)Via.calle.getWait().getEl().getDat(n);
         x = m.getdoubleValue("x");
         if(x<=Via.l/2){
            Message a = (Message)(m.getValue("anterior"));
            if(x<Via.l/2-10 && (a==null||a.getdoubleValue("x")>Via.l/2+10)){
               posterior = m;
               Glider.act(Via.entrada, 0);
               it(40);
               activa=true;
               return;
            }else{
               it(0.1);
               return;
            }
         }
      }
      it(40);
      return;
   }
   public Message getPosterior(){
       return posterior;
   }
}
```

2.3.2. Semáforo

Para el caso del semáforo el esquema de los tres modelos se muestra en la figura 2.5. El modelo está conformado ahora por dos espacios distintos `calle` y `otraCalle`,

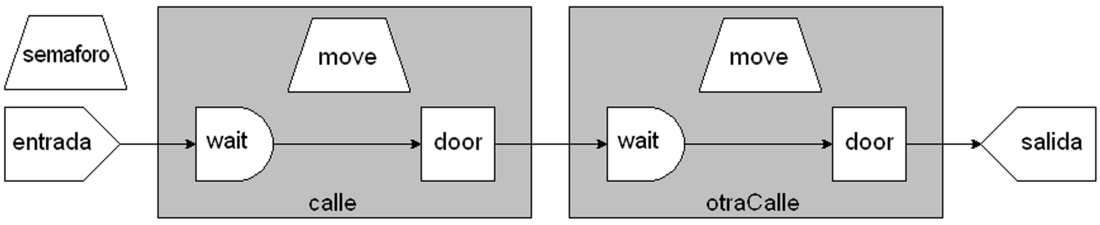


Figura 2.5: El semáforo contiene dos espacios, antes y después del semáforo y un nodo autónomo que simula al semáforo.

además se incluye un nodo autónomo nuevo `semaforo` que controla las acciones del semáforo en la vía, representado por la puerta que comunica `calle` y `otraCalle`. Las clases `Entrada` y `Salida` quedan sin cambios con respecto al modelo base de cada modelo en particular. En cambio las clases `Via` y `Regla` se ven modificadas.

`Via:` Como se ve en el código 17, se instancian un nuevo espacio y un nuevo nodo que llamamos `otraCalle` y `Semaforo` respectivamente, se añade la geometría referente al nuevo espacio y finalmente se instancian el tiempo de la primera activación del `semáforo` y `otraCalle`.

Código 17 `Via.java` Extracto de la implementación de la clase principal para los modelos con semáforo.

```
static Space otraCalle = new Space("otraCalle",0,3*l/4,6.0,1.0,0.1,"Regla");
static Semaforo semaforo = new Semaforo();
     /* ****************** */
     /* ACA VA CODIGO BASE */
     /* ****************** */
     otraCalle.addWall(l/2, 0.0, l, 0.0);
     otraCalle.addWall(l/2, 12.0, l, 12.0);
     otraCalle.addWall(l/2, 0.0, l/2, 12.0);
     otraCalle.addWall(l, 0.0, l, 12.0);
     otraCalle.addDoor(l, 0.0, l, 12.0, salida, 'U', 1, 0.0, 0.0);
     otraCalle.build();
     display.addSpace(otraCalle);
     /* ****************** */
     /* ACA VA CODIGO BASE */
     /* ****************** */
     Glider.act(otraCalle.getMove(), 0);
     Glider.act(semaforo,0);
```

`Regla:` Aquí además de la información de cada modelo en particular se añade el código, que produce la reducción de la velocidad a cero del vehículo que corte la puerta en el momento en que ésta esté cerrada, lo que representa el semáforo en rojo, como muestra el código 18.

Código 18 `Regla.java` Reducción de velocidad en el semáforo.

```
if(nCoord[2]==1 && e.getDoor(1).isClose()){
    m.setField("vvx", 0);
}
```

`Semaforo:` Clase extendida a nodo tipo **A**, con un método `fact` cuya única función es mantener abierta la puerta durante 55 unidades de tiempo y luego mantenerla cerrada durante 35 unidades de tiempo, tal y como se ve en el código 19.

Código 19 `Semaforo.java`

```
import galatea.glider.*;
class Semaforo extends Node {
    Semaforo(){
        super("semaforo",'A' );
        Glider.nodesl.add(this);
    }
    public void fact() {
        Via.calle.getMove().getDoor(1).setClose(!Via.calle.getMove()
                                                .getDoor(1).isClose());
        if(Via.calle.getMove().getDoor(1).isClose()){
            it(35);
        }else{
            it(55);
    }
    }
}
```

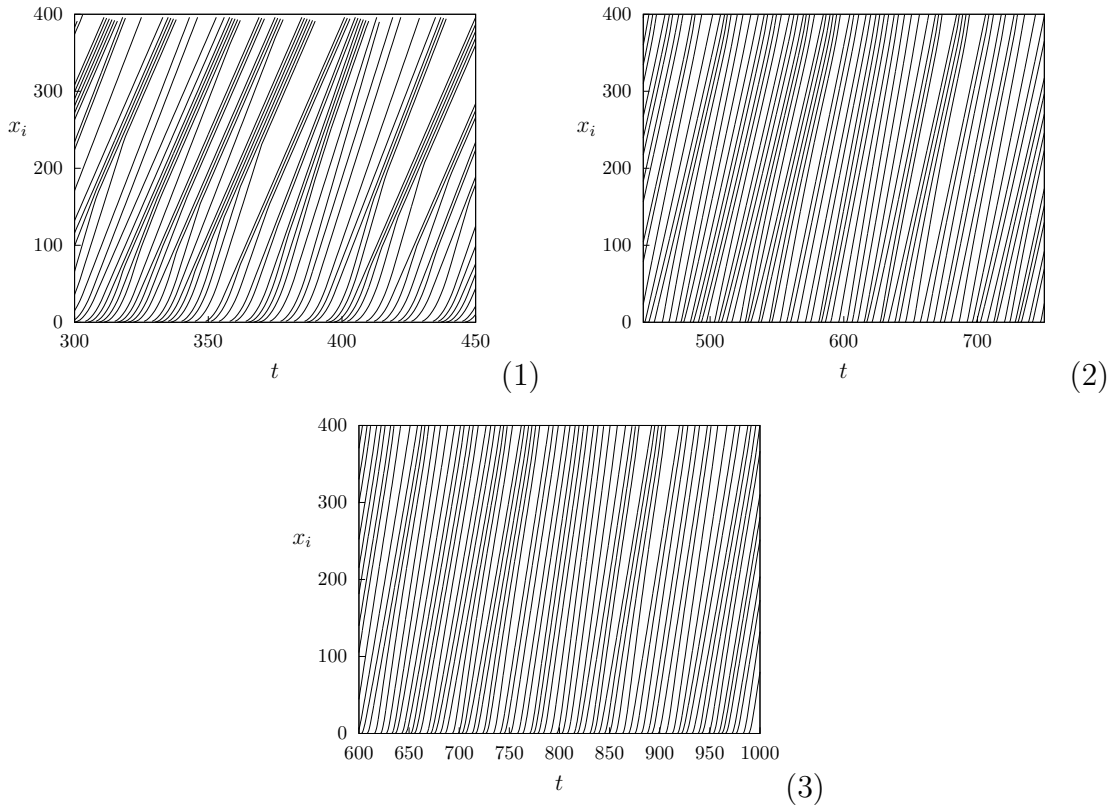
Capítulo 3

Resultados

En este capítulo mostraremos en primer lugar el análisis de un grupo de gráficas que representan la posición de los vehículos respecto al tiempo en los distintos modelos, por una parte las simulaciones base, luego la incorporación, el semáforo y finalmente se muestra una gráfica de tiempo de duración del modelo respecto al modelo y a la longitud de la vía.

3.1. Análisis del modelo general

En el conjunto de gráficas 3.1 se observa una muestra del cambio de posición con respecto al tiempo de un conjunto de vehículos que recorre una vía de longitud 400 celdas. Una de las cualidades más importantes que se observa es que en la representación no hay cruce de líneas hecho que evidencia que no se produce adelantamiento de vehículos. Como la velocidad deseada `v0` de cada vehículo se distribuye aleatoriamente entre 8 y 12 unidades, el sistema a partir de las doscientas celdas presentan clusters de vehículos producto de la reducción de velocidad de conjuntos de vehículos para mantener su distancia de acuerdo con las condiciones de los modelos.

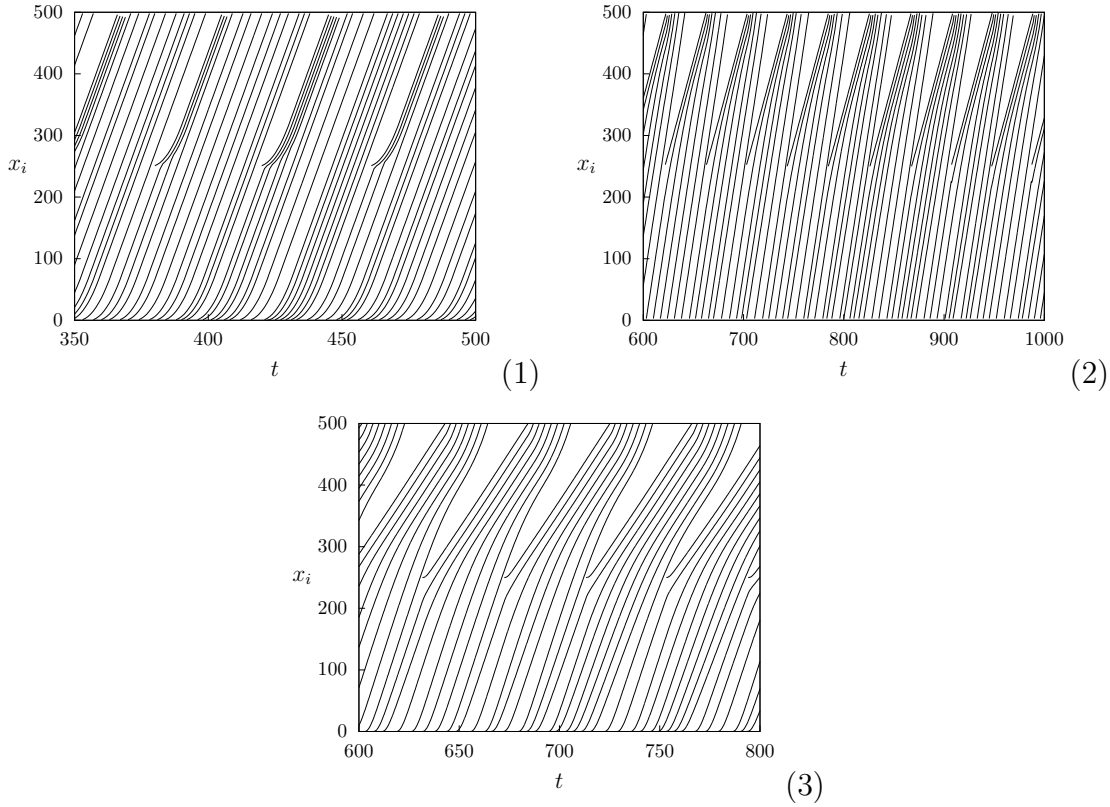


Gráficas 3.1: posición de cada vehículo vs. tiempo. Cada línea representa un vehículo atravesando la vía. Modelo Base, (1) autómatas celulares, (2) mapas acoplados, (2) conductor inteligente.

3.2. Incorporación

En esta sección se muestra algunas observaciones correspondientes a la incorporación de las tres distintas simulaciones que se hicieron. Se graficó una vía de longitud 500 celdas con incorporación en la celda 250, en distintos intervalos. El conjunto de vehículos incorporados a la vía tenían una velocidad deseada menor a la velocidad deseada de los vehículos que se movían desde el comienzo de la misma, lo que obliga a los sistemas a producir ondas de frenado que finalmente permiten la observación de clusters de vehículos.

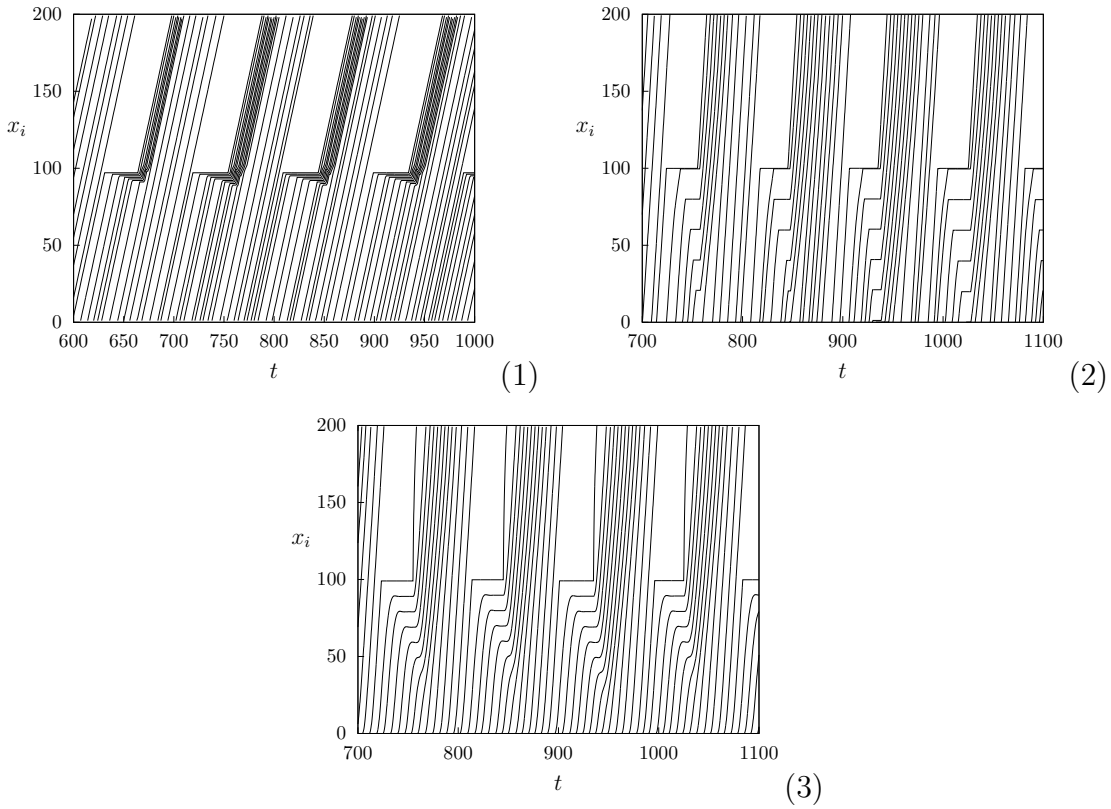
Para el conjunto de gráficas 3.2, en primer lugar se puede observar que el vehículo que se incorpora está representado por una línea que arranca en la celda 250, el modelo funciona correctamente.



Gráficas 3.2: posición de cada vehículo vs. tiempo. Cada vehículo está representado por una línea en los gráficos. Incorporación, (1) autómatas celulares, (2) mapas acoplados, (3) conductor inteligente.

3.3. Semáforo

En cuanto al funcionamiento del semáforo se refiere, tenemos el conjunto de gráficas 3.3 donde se muestra aproximadamente cuatro cambios del semáforo al rojo en una vía de doscientas celdas. Entre las observaciones que se pueden hacer de funcionamiento se puede decir, que los modelos cumplen con las mentas fijadas, entre otras cosas no existen errores de adelantamiento de vehículos, se respetan las distancias entre vehículos, en el momento en que la puerta se cierra, es decir, el semáforo entra en rojo, los vehículos reducen su velocidad de manera eficiente y frenan conservando la distancia de entre ellos.



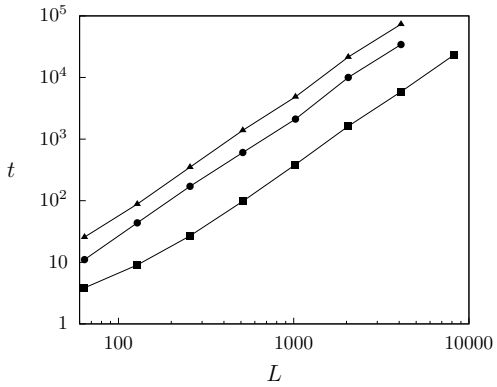
Gráficas 3.3: posición de cada vehículo vs. tiempo. Cada vehículo está representado por una línea en los gráficos. Semáforo, (1) autómatas celulares, (2) mapas acoplados, (3) conductor inteligente.

3.4. Tiempos de ejecución de los modelos

Para que los futuros usuarios de estos modelos tengan una idea del tiempo de computo que les llevaría hacer las simulaciones hemos considerado importante medir los tiempos de ejecución para los tres modelos base y ver cómo escala el tiempo en función del número de vehículos en el sistema. Como se observa en la gráfica 3.4 se tiene un conjunto de tres curvas que representan el tiempo que tarda cada simulación en su modelo base con respecto a la longitud de la vía. La curva con indicador cuadrado representa los autómatas celulares, la curva con indicador circular representa la red de mapas acoplados y la curva con indicador triangular representa el conductor inteligente, como era de esperarse una de las primeras observaciones que se puede hace es que el tiempo que tarda cada modelo es proporcional al grado de complejidad de los cálculos del mismo. Por otra parte, la función con la que se puede representar el conjunto de curvas es,

$$t \sim L^{\alpha} \,, \tag{3.1}$$

Ahora bien, para el conjunto de datos obtenidos en cada una de las distintas simulaciones, el tiempo efectivo a mayor longitud aumenta con un $\alpha = 1,96 \pm 0,06$, es como podría esperarse.



Gráfica 3.4: Tiempo de ejecución en segundos versus la longitud de la vía, ambos ejes están escala logarítmica. (cuadrado) Autómatas celulares, (circulo) Mapas acoplados, (triángulo) Conductor inteligente.

Conclusiones

En este documento se desarrollaron e implementaron modelos para tráfico vehicular unidireccional sustentados en los modelos de conductor inteligente, mapas acoplados y autómatas celulares que se habían planteado. Se implementaron los tres modelos en `GALATEA` y realizaron algunas simulaciones de cada modelo en los distintos escenarios que fueron; la vía simple, la incorporación de vehículos y el semáforo. Adicionalmente se hizo varias mediciones entre las cuales esta la eficiencia de las simulaciones en la vía simple, el comportamiento de muestras de vehículos en la vía, y se pudo observar los patrones de movimiento reproducidos por los modelos que son coherentes con los que se observarían en implementaciones de este tipo.

El conjunto de gráficas 3.1, 3.2 y 3.3 aseguran que la evolución en el tiempo de los modelos es correcta, no se produce adelantamiento de vehículos y los patrones de continuidad consolidan las técnicas de simulación usadas. En el caso de los autómatas celulares las curvas de las gráficas 3.2.(1) y 3.3.(1) se corresponden con una simulación en la cual el espacio es discretizado por celdas, igualmente de las gráficas 3.2.(2) y 3.3.(2) y (3) se muestran patrones donde se suavizan las curvas. Y por ultimo se tiene de la gráfica 3.4, que muestra como escala el tiempo en función del número de vehículos, esto para dar una referencia aproximada de los tiempos de simulación según la longitud de la vía y el número de vehículos.

Referencias

[1] B.D. Greenshields. *A study of traffic capacity.* In Proceedings of the Highway, Highway Research Board, Proceedings, **14**, 458 (1935).

[2] A. Kesting, M. Treiber, D. Helbing. *Agents for Traffic Simulation,* en Multi-Agent Systems: Simulation and Applications, Capítulo, 325-356. Editores A. Uhrmcher and D. Weyns, CRC (2009).

[3] M.J. Lighthill, G.B. Whitham. *On kinematic waves: II. A theory of traffic on long crowded roads.* Proc. Roy. Soc. of London A, **229**, 317-345 (1955).

[4] P.I. Richards. *Shock waves on the highway,* Operations Research, **4**, 42-51 (1956).

[5] I. Prigogine, R. Herman, *A Two-fluid Approach ot Town Traffic,* Science, Reprint Series, **204**, 148-151 (1979).

[6] R. Wiedemann. *Simulation des Strabenverkehrsflusses.* En Heft 8 der Schriftenreihe des IfV. Institut föur Verkehrswesen, Universitöat Karlsruhe, (1974).

[7] M. Treiber, A. Hennecke, D. Helbing, *Congested traffic states in empirical observations and microscopic simulations,* Phys. Rev. E, **i62** 1805-1824 (2000).

[8] K. Nagel, M. Schreckenberg. *A cellular automaton model for freeway traffic,* J. Phys. I France, **2**, 2221-2229 (1992).

[9] P.G. Gipps. *A behavioural car-following model for computer simulation,* Transportation Research Part B: Methodological, **15**, 105-111 (1981).

[10] G.F. Newell. *Nonlinear effects in the dynamics of car following.* Operations Research, **9**, 209 (1961).

[11] B. Chopard, M. Droz, *Cellular automata Modeling of Physical Systems,* 1998.

[12] J. Dávila, M. Uzcátegui, K. Tucci. *Simulación Multi-agente con GALATEA.*ULA, 2004.

[13] M. Uzcátegui, *Diseño de la plataforma de simulación de sistemas multi-agentes GALATEA,* Tesis de Maestría. Universidad de Los Andes. (2002).

[14] M. Uzcátegui, J. Dávila, K. Tucci, *GALATEA: Plataforma de Simulación de Sistemas Multi-Agentes,* Memorias de VI Jornadas científico técnicas de la Facultad de Ingeniería, Universidad de Los Andes, Mérida, Venezuela (2007).

[15] K. Laffaille, *Gspace, meta-modelo para simular desalojos de Espacios Urbanos y Arquitectónicos basado en GALATEA* Tesis de Maestría. Universidad de los Andes, Mérida, Venezuela (2005).

[16] B. P. Zeigler, *Theory of modelling and simulation,* ser. Interscience. New York: Jhon Wiley and Sons, (1976).