UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

DESCRIPCIÓN, APLICACIÓN E IMPLEMENTACIÓN DE UN MODELO DE EXTENSIÓN DE FLUJOS EN REDES

T E S I S

QUE PARA OBTENER EL TÍTULO DE

M A T E M Á T I C O

P R E S E N T A :

ACRISIO CABALLERO CABALLERO

FACULTAD DE CIENCIAS
UNAM

DIRECTOR DE TESIS:

DR. RICARDO GÓMEZ AÍZA

MÉXICO, D. F. 2004

VNIVERSIDAD NACIONAL
AVTNOMA DE
MEXICO

ACT. MAURICIO AGUILAR GONZÁLEZ
Jefe de la División de Estudios Profesionales de la Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:
Descripción, aplicación e implementación de un modelo de extensión de flujos en redes.

realizado por Acrisio Caballero Caballero

con número de cuenta 8521113-9 , quien cubrió los créditos de la carrera de: Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis
Propietario Dr. Ricardo Gómez Aíza

Propietario Dr. Francisco Marcos López García

Propietario Mat. José Luis Torres Rodríguez

Suplente Dr. Juan José Montellano Ballesteros

Suplente M. en C. María del pilar Valencia Suravia Pilar Valencia S.

Consejo Departamental de Matemáticas

M. en C. Alejandro Bravo Mojica

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL
DE
MATEMATICAS

AGRADECIMIENTOS

A Dios, por darme la maravillosa oportunidad de lograr una meta tan anhelada en mi vida.

A mis queridos padres, a quienes les pido perdón por el sufrimiento causado en tantas ocasiones.

A mi querida esposa Araceli, por su cariño, su paciencia, su fortaleza, por ser la compañera de mi vida y pilar de nuestro hogar.

A mi amigo Omar, por su nobleza, su madurez, su inteligencia y principalmente, por permitirme sentir día a día el orgullo tan grande de ser el padre de un hijo como él.

A mi pequeña Picos, por su carácter y su ternura, por sus enojos y sus alegrías, por sus gustos y sus disgustos, por tantas contradicciones en un ser tan hermoso que representa la luz de nuestra casa.

A mis hermanas y mis hermanos: Clementina (por su fortaleza), Elías (por su carácter), Gudelia (por su apoyo), Anita (por su ternura), Jesús (mi compañero de mil batallas), Kenia (por su inteligencia), Enna (luchadora incansable), Arquímedes (una persona fuera de serie). Por su ejemplo y comprensión, gracias, muchas gracias.

A mis suegros, esperando que algún día tenga nuevamente la dicha de tomar una cerveza y platicar de tantas cosas con mi querido amigo Don Juan.

A mis cuñadas y mis cuñados: Bernardo, Manuel, Edith, Bertrán, Ceci, Alicia, Gaby, Paty, Sofía, Juan, Ale y Miguel. Gracias por darme el privilegio de convivir con gente tan valiosa como lo son ustedes.

A mis tías, tíos, sobrinas, sobrinos, concuñas, concuños,..., como podrán darse cuenta, la lista pareciera interminable por lo que pido disculpas por no citar los nombres de todos mis seres queridos.

Muy en especial, quiero agradecer por su apoyo, paciencia y dedicación a mi querido amigo Ricardo Gómez Aíza, pero principalmente por haberme brindado su amistad.

A todos mis sinodales, por su apoyo, paciencia y consejos esenciales para el desarrollo de este trabajo.

A mis amigos, muy en especial a José Luis y su hermosa familia, a mi querida amiga Pilar, Marcos, Hugo, Héctor, Rogelio, Mario, Víctor Ricardo, Arturo, Napoleón, etc. Gracias.

Descripción, aplicación e implementación de un modelo de extensión de flujos en redes

Acrisio Caballero Caballero

octubre de 2004

Prefacio

El propósito de esta tesis, es el de presentar un modelo estadístico que aborda el *problema de extensión de flujos en redes*, el cual hemos tomado de [7]. Este problema surge en redes de tránsito vehicular cuando se hace necesario obtener un panorama más completo del estado del tráfico para un mejor control. En su generalidad, el modelo puede ser adaptado a diferentes clases de redes. Aquí lo ponemos a prueba con una implementación en base a una red de tránsito real. La importancia del estudio de redes de tránsito urbano se refleja en muchos trabajos como lo son: [1] [3] [5] [6] [8], [9] [10] y [13], por mencionar algunos.

La tesis comienza con un breve capítulo de introducción cuyas primeras dos secciones contienen conceptos básicos y la planteación del problema; en la última sección de este primer capítulo se presentan como referencia definiciones importantes tales como el concepto de *divergencia* y el de *potencial* entre otros. En la primera sección del segundo capítulo se describe formalmente el modelo, basado en las *funciones de correlación de flujo*. Su segunda sección presenta algunos ejemplos de este tipo de funciones que se utilizaron para elaborar los ejemplos. En el tercer capítulo, se presenta y discute el código de la implementación del modelo. Para este trabajo se utilizó una base de datos que describe una red vial urbana, en este caso de la Ciudad de México, la cual fue proporcionada con consentimiento para su uso en investigación por TeleGis de México S.A. de C.V.

II

Índice general

Capítulo 1

Introducción

1.1. Redes

Definición 1 *Una* red $\mathcal{N}$ *consiste de un conjunto* $\mathcal{V}$ *cuyos elementos llamamos* nodos *o* vértices*, de un conjunto* $\mathcal{A}$ *cuyos elementos llamamos* arcos *o* aristas*, y de una* función de incidencia $e\colon \mathcal{A} \to \mathcal{V} \times \mathcal{V}$.

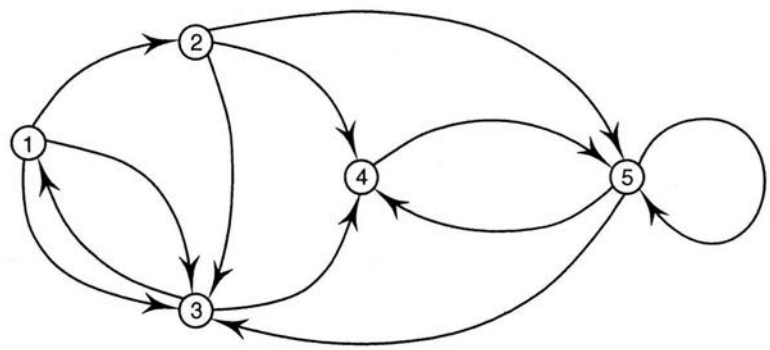

Figura 1. Una red típica.

Sea $\mathcal{N} = (\mathcal{V}, \mathcal{A}, e)$ una red. Para cada $a \in \mathcal{A}$, escribiremos $e(a) = (i(a), t(a)) \in \mathcal{V} \times \mathcal{V}$ y diremos que $i(a)$ es el vértice *inicial* de a y que $t(a)$ es el vértice *terminal* de a. Podemos representar gráficamente a la red $\mathcal{N}$ poniendo un punto en el plano por cada vértice $v \in \mathcal{V}$,

y para cada arista $a \in \mathcal{A}$ colocando una flecha que comienza en el vértice inicial $i(a)$ y que termina en el vértice terminal $t(a)$. Decimos que la red $\mathcal{N}$ es *finita* si $|\mathcal{V}|, |\mathcal{A}| < \infty$. En este trabajo sólo consideraremos redes finitas. Decimos que la red $\mathcal{N}$ tiene *aristas múltiples* si existen $a, b \in \mathcal{A}$, tales que $a \neq b$ y $i(a) = i(b)$ y $t(a) = t(b)$, y decimos que tiene *loops* si existe $a \in \mathcal{A}$ tal que $i(a) = t(a)$. La figura 1 de la página anterior muestra una red con aristas múltiples del vértice 1 al vértice 3 y con un loop en el vértice 5. Los loops pueden ser eliminados mediante la sustituición descrita en la siguiente figura 2:

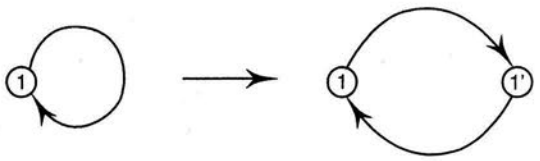

Figura 2. Sustitución para eliminar loops.

Las aristas múltiples pueden ser eliminadas mediante la sustitución descrita en la siguiente figura 3:

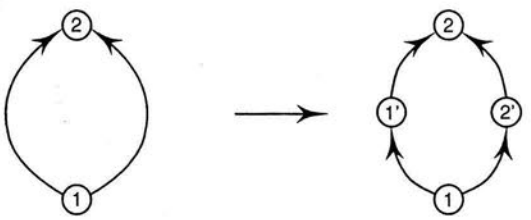

Figura 3. Sustitución para eliminar aristas múltiples.

Así, sólo consideraremos redes sin loops ni aristas múltiples, en cuyo caso podemos identificar al conjunto de aristas $\mathcal{A}$ con un subconjunto de $\mathcal{V} \times \mathcal{V}$ de forma tal que la referencia a la función de incidencia e se vuelve innecesaria.

En la práctica existen muchas clases de redes, mencionaremos algunos ejemplos.

- **Estados de un sistema discreto.** En este ejemplo los vértices representan estados de un proceso en el que una arista $a = (i, j)$ indica que el estado i puede preceder al j. Es común que haya una probabilidad de transición del vértice i al j, como sucede en las cadenas de Markov, que son un modelo muy importante tanto por sus propiedades matemáticas como por sus aplicaciones [2].

- **Redes de transporte.** En las redes de transporte los vértices son ciertos sitios o lugares, como ciudades, fábricas, la esquina de una calle, etc. Las aristas son rutas o caminos, que nos llevan de un sitio a otro. En esta clase de redes es natural el concepto de flujo, que veremos más adelante. La idea es asumir que en cada instante de tiempo, cierta cantidad de material recorre la arista.

- **Redes de comunicación.** Aquí el tipo de redes es muy semejante al anterior. Algunos buenos ejemplos son las redes de internet y las telefónicas como la celular. Este tipo de redes han sido muy estudiadas por su importancia en la vida moderna y pueden formar parte de aquellas redes a las cuales es adaptable el modelo que describimos en este trabajo.

- **Redes hidráulicas.** En este caso es agua la que origina el flujo en una red formada por canales de irrigación, ríos, drenajes, etc. Diferentes niveles de flujo pueden observarse, como en temporadas de escasez o temporadas de lluvia. Un panorama global del flujo hidráulico es sin duda información que puede ser útil.

Concluimos esta sección con una definición básica.

Definición 2 *Sea $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ una red. Una* subred *de $\mathcal{N}$ es una red $\mathcal{N}_0 = (\mathcal{V}_0, \mathcal{A}_0)$ tal que $\mathcal{V}_0 \subset \mathcal{V}$ y $\mathcal{A}_0 \subset \mathcal{A}$.*

Si $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ es una red y $\mathcal{A}_0 \subset \mathcal{A}$ es un subconjunto de aristas, entonces existe una subred $\mathcal{N}_0 = (\mathcal{V}_0, \mathcal{A}_0)$, donde $\mathcal{V}_0$ consiste de los vértices iniciales y finales de las aristas en $\mathcal{A}_0$, de forma que haremos sólo referencia a $\mathcal{A}_0$ que determina la subred $\mathcal{N}_0$.

1.2. Flujos y el problema de extensión

Definición 3 *Sea $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ una red. Un* flujo *en $\mathcal{N}$ es una función $f\colon \mathcal{A} \to \mathbb{R}$.*

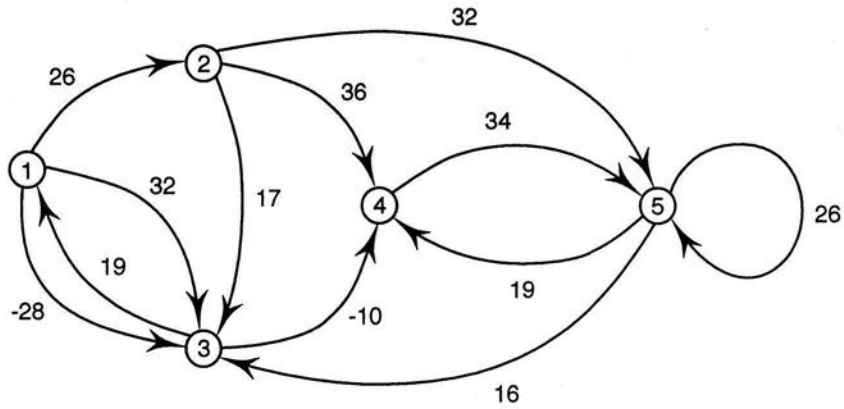

Figura 4. Un flujo en una red.

Sea $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ una red y $f\colon \mathcal{A} \to \mathbb{R}$ un flujo. Para cada $a \in \mathcal{A}$, nos referiremos al valor $f(a)$ como el *flujo* en el arco a y será interpretado como la cantidad de material que fluye a través de a. Si el flujo en a es positivo, entonces se entenderá que el material fluye en la dirección de a, de otra forma se entenderá que fluye en la dirección contraria. La figura 4 muestra un ejemplo de un flujo en una red. Invirtiendo la dirección de los arcos con flujo negativo y luego estos flujos negativos hacerlos positivos resulta en una red con *flujo positivo*, que es una función $f\colon \mathcal{A} \to \mathbb{R}_+$.

Introducimos ahora una definición básica.

Definición 4 *Sea $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ una red y sea $\mathcal{N}_0 = (\mathcal{V}_0, \mathcal{A}_0)$ una subred de $\mathcal{N}$. Sea $f\colon \mathcal{A}_0 \to \mathbb{R}$ un flujo en $\mathcal{N}_0$. Un flujo $F\colon \mathcal{A} \to \mathbb{R}$ en $\mathcal{N}$ es una* extensión *de f si $F|_{\mathcal{A}_0} = f$.*

El problema que estudiamos es el siguiente.

Problema. Sean $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ una red y $F\colon \mathcal{A} \to \mathbb{R}$ un flujo. Sea $\mathcal{N}_0 = (\mathcal{V}_0, \mathcal{A}_0)$ una subred de $\mathcal{N}$ y sea $f = F|_{\mathcal{A}_0}$, de forma que f es un flujo en $\mathcal{N}_0$ y F es una extensión de f. Supongamos que sólamente conocemos el flujo f en $\mathcal{N}_0$. ¿De qué manera es posible encontrar una extensión de f que sea una buena aproximación de F?

Más adelante daremos una definición precisa de lo que entenderemos por "una buena aproximación".

Como ya se ha dicho, el origen de esta pregunta provino del tráfico vehicular. Por naturaleza,

las redes de tránsito son precisamente eso, *redes*, y el flujo vehicular es también eso, un *flujo*. Normalmente se conoce la situación del tráfico vehicular en un instante de tiempo, lo que llamamos el flujo parcial en la red, y un panorama más extenso puede convertirse en información útil para mejor control. En esta tesis presentamos un modelo estadístico que aborda este problema y se dan ejemplos con una implementación en base a una red vehicular real alimentada en base a diferentes situaciones que se dan en la vida común de un día de tránsito.

1.3. Divergencia y otros conceptos.

Sean $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ una red y $f\colon \mathcal{A} \to \mathbb{R}_+$ un flujo positivo. Para cada $i \in \mathcal{V}$, la *divergencia* en i que denotamos por $\mathrm{div}(i)$ es la diferencia entre la suma de los flujos de las aristas que parten de i y la suma de los flujos de las aristas que llegan a i. El principio de divergencia total dice que

$$\sum_{i \in \mathcal{V}} \mathrm{div}(i) = 0.$$

Como ejemplo, podemos observar en el flujo de la red de la figura 4, la divergencia en el nodo 1 es de 11, y la divergencia en el nodo 4 es de -11, así mismo, la divergencia total en la red es igual a 0.

Si se considera una función $g\colon \mathcal{V} \to \mathbb{R}$ lo que se obtiene es un *potencial*. La diferencia de los potenciales en los vértices iniciales y terminales de una arista $a \in \mathcal{A}$ es la *tensión* del arco a y juega un papel de *diferencial* del potencial.

El flujo en una red de tránsito vehicular es comúnmente restringido a *intervalos de capacidad*, que para cada arista $a \in \mathcal{A}$ hay un intervalo $[c_-(a), c_+(a)] \subset \mathbb{R}_+$, con $c_-(a) \leq c_+(a)$, de forma que se requiere $f(a) \in [c_-(a), c_+(a)]$. Clases de intervalos pueden servir para retroalimentar el modelo. En los ejemplos que se presentan en este trabajo, el intervalo de capacidad para todas las arista será de $[0, 80]$ y representará la velocidad promedio de los vehículos que se encuentran cruzando el tramo correspondiente en un momento dado.

Estos conceptos los presentamos a modo de referencia al lector, su estudio más detallado puede verse en [11].

Capítulo 2

Modelo de extensión de flujos

En este capítulo presentamos el modelo de extensión de flujos y un ejemplo de su aplicación práctica al caso de redes de tránsito vehicular.

2.1. Modelo de extensión en base a funciones de correlación de flujo

Se parte del punto en el que el flujo en la red es parcialmente conocido. El objetivo es determinar el flujo en su totalidad mediante una extensión adecuada que preserve la naturaleza intrínseca del flujo. El problema se aborda con la clase de *funciones de correlación de flujo.* La implementación del modelo a un caso específico requerirá de la búsqueda de funciones de correlación de flujo adecuadas que capturen los principales factores que afectan el flujo.

Definición 5 *Sea* $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ *una red. Una* función de correlación de flujo *es una función*

$$\rho\colon \mathcal{A} \times \mathcal{A} \to [0, 1].$$

Para cada $a, b \in \mathcal{A}$, se interpreta al valor $\rho(a, b)$ como una medida de la *influencia* del flujo en a sobre el flujo en b. Si el flujo en b es conocido, sería natural asumir que no existe ninguna influencia del flujo en a sobre el flujo en b a menos que $a = b$. Esto motiva la siguiente definición.

Definición 6 *Sea $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ una red y $\rho\colon \mathcal{A} \times \mathcal{A} \to [0,1]$ una función de correlación de flujo. Si $\mathcal{A}_0 \subset \mathcal{A}$ es un subconjunto de aristas de $\mathcal{A}$, la* función de correlación de flujo con base $\mathcal{A}_0$ asociada a ρ *se define para cada $a \in \mathcal{A}_0$ y $b \in \mathcal{A}$ por*

$$\rho_{\mathcal{A}_0}(a,b) = \delta(a,b)\mathbb{1}_{\mathcal{A}_0}(b) + \rho(a,b)\mathbb{1}_{\mathcal{A}_0^c}(b) \tag{2.1}$$

donde δ es la de Kronecker y $\mathbb{1}$ es la función indicadora del subconjunto subíndice.

Así, si $a \in \mathcal{A}_0$ y $b \in \mathcal{A}$, entonces la función de correlación de flujo con base $\mathcal{A}_0$ asociada a una función de correlación de flujo ρ es tal que si $b \in \mathcal{A}_0$, entonces

$$\rho_{\mathcal{A}_0}(a,b) = \begin{cases} 1 & \text{si } b = a \\ \\ 0 & \text{si } b \neq a \end{cases}$$

y si $b \notin \mathcal{A}_0$, entonces $\rho_{\mathcal{A}_0}(a,b) = \rho(a,b)$.

El flujo promedio en las aristas de $\mathcal{A}_0$ es

$$\mu(\mathcal{A}_0) = \frac{1}{|\mathcal{A}_0|} \sum_{a \in \mathcal{A}_0} f(a). \tag{2.2}$$

Para cada $(a,b) \in \mathcal{A}_0 \times \mathcal{A}$, la combinación lineal convexa

$$f_a(b) = \rho_{\mathcal{A}_0}(a,b) f(a) + (1 - \rho_{\mathcal{A}_0}(a,b))\mu(\mathcal{A}_0) \tag{2.3}$$

se encuentra entre mín$\{f(a), \mu(\mathcal{A}_0)\}$ y máx$\{f(a), \mu(\mathcal{A}_0)\}$, y el valor $f_a(b)$ se aproxima a $f_a(b)$ ó a $\mu(\mathcal{A}_0)$ si $\rho_{\mathcal{A}_0}(a,b)$ se aproxima a 1 ó a 0 respectivamente. Consideremos $a \in \mathcal{A}_0$ y $b \in \mathcal{A}$. Si $b = a$, entonces $f_a(b) = f(b)$, si $b \in \mathcal{A}_0$ pero $b \neq a$, entonces $f_a(b) = \mu(\mathcal{A}_0)$, y finalmente, si $b \notin \mathcal{A}_0$, entonces $f_a(b) \approx f(a)$ si $\rho(a,b) \approx 1$ y $f_a(b) \approx \mu(\mathcal{A}_0)$ si $\rho(a,b) \approx 0$. El promedio de los valores $f_a(b)$ sobre los elementos $a \in \mathcal{A}_0$ es una estimación del flujo en $b \in \mathcal{A}$, concretamente

$$\frac{1}{|\mathcal{A}_0|} \sum_{a \in \mathcal{A}_0} f_a(b),$$

pero en general no es una extensión de f, y además se concentra en el valor $\mu(\mathcal{A}_0)$, en particular si la cantidad de aristas en $\mathcal{A}_0$ con alta correlación de flujo en b es pequeña con respecto a la cantidad total de aristas en $\mathcal{A}_0$. Se contrarresta esta situación con una ponderación.

Definición 7 *Dada una función de correlación de flujo* $\rho\colon \mathcal{A} \times \mathcal{A} \to [0,1]$, la función de ponderación $\tau\colon \mathcal{A} \to \mathbb{R}_+$ asociada a ρ con base $\mathcal{A}_0 \subset \mathcal{A}$ *está definida para cada* $b \in \mathcal{A}$ *por*

$$\tau(b) = \sum_{a \in \mathcal{A}_0} \rho_{\mathcal{A}_0}(a,b). \tag{2.4}$$

Observemos que $\tau(b) = 1$ para toda $b \in \mathcal{A}_0$. El modelo de extensión de flujos asociado a una función de correlación de flujo ρ es el siguiente.

Definición 8 *Sea* $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ *una red y* $\mathcal{A}_0 \subset \mathcal{A}$ *un subconjunto de aristas. Sea* $f\colon \mathcal{A}_0 \to \mathbb{R}$ *un flujo y* $\rho\colon \mathcal{A} \times \mathcal{A} \to [0,1]$ *una función de correlación de flujo.* La extensión del flujo f asociada a ρ *es la función* $F(\cdot \mid \mathcal{A}_0)\colon \mathcal{A} \to \mathbb{R}$ *definida para cada* $b \in \mathcal{A}$ *por la ecuación*

$$F(b \mid \mathcal{A}_0) = \frac{1}{\tau(b)} \sum_{a \in \mathcal{A}_0} \rho_{\mathcal{A}_0}(a,b) f_a(b). \tag{2.5}$$

El hecho de que $\rho_{\mathcal{A}_0}$ es una función de correlación de flujo con base $\mathcal{A}_0$ implica que $F(\cdot \mid \mathcal{A}_0)$ es una extensión de f, es decir, $F(b \mid \mathcal{A}_0) = f(b)$ para toda $b \in \mathcal{A}_0$, lo cual se verifica directamente. Si $\rho_1, \cdots, \rho_n\colon \mathcal{A} \times \mathcal{A} \to [0,1]$ son funciones de correlación de fujo, estas se pueden considerar distribuidas de acuerdo a un vector de probabilidad $\Pi = (\pi_1, \cdots, \pi_n)$ con $\pi_1, \cdots, \pi_n \geq 0$ y $\pi_1 + \cdots + \pi_n = 1$ de forma que se obtiene una extensión usando la función de correlación de flujo (ver figura 5):

$$\rho = \sum_{i=1}^{n} \pi_i \rho_i\colon \mathcal{A} \times \mathcal{A} \to [0,1]. \tag{2.6}$$

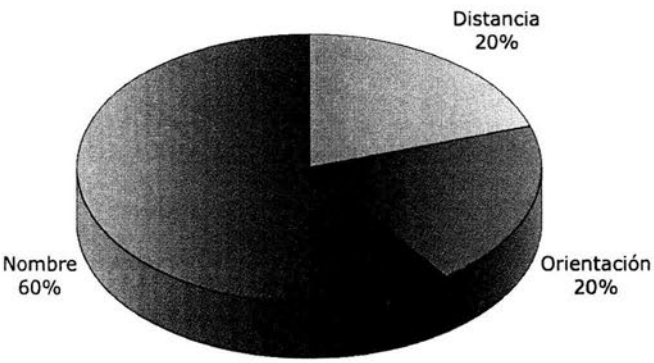

Figura 5. Distribución de funciones de correlación de flujo.

A continuación se presenta una definición de *error*, que servirá más adelante para determinar cuándo tenemos "una buena aproximación" y así medir la confiabilidad de los resultados.

Definición 9 *Sea* $\mathcal{N} = (\mathcal{V}, \mathcal{A})$, $\mathcal{A}_0 \subset \mathcal{A}$ *un subconjunto de aristas,* $f\colon \mathcal{A}_0 \to \mathbb{R}$ *un flujo y* $\rho\colon \mathcal{A} \times \mathcal{A} \to [0,1]$ *una función de correlación de flujo. Para cada* $a \in \mathcal{A}_0$ *se define* el error local en a *como*

$$\varepsilon(a) = f(a) - F(a \mid \mathcal{A}_0 - \{a\}). \tag{2.7}$$

El error global *es*

$$\varepsilon = \frac{1}{|\mathcal{A}_0|} \sum_{a \in \mathcal{A}_0} |\varepsilon(a)|. \tag{2.8}$$

El error local en $a \in \mathcal{A}_0$ es la desviación del modelo del valor real $f(a)$ suponiendo que los valores $f(b)$ son conocidos para toda $b \in \mathcal{A}_0 - \{a\}$, y el error global es el promedio del valor absoluto de estas desviaciones.

Si se utilizan varias funcion de correlación de flujo como en 2.6, entonces tanto los errores locales como el error global dependerán de la distribución del vector de probabilidad $\Pi = (\pi_1, \cdots, \pi_n)$, de forma que puede ser utilizado como parámetro para disminuir dichos errores. Como se ve más adelante en los ejemplos, las funciones de correlación de flujo $\rho_1, \cdots, \rho_n$ pueden a su vez depender de otro tipo de parámetros $\theta = (\theta_1, \cdots, \theta_m)$ que también pueden ser manipulados. Como vemos, diferentes variaciones del modelo pueden obtenerse, por ejemplo, considerar una cadena de longitud k de subredes $\mathcal{A}_0 \subset \mathcal{A}_1 \subset \cdots \subset \mathcal{A}_k = \mathcal{A}$ a forma de iterar el modelo y obtener más extensiones. En este trabajo sólo consideraremos cadenas de longitud $k = 1$, con el mismo intervalo de capacidad $[0, 80]$ para todas las aristas (de forma que 80 km/h es la velocidad máxima a la que un automóvil puede recorrer una arista). Los análisis del error para cadenas de subredes y otros temas se encuentran en [7].

2.2. Redes de tránsito vehicular

Para adaptar el modelo a un caso particular se deben buscar funciones de correlación de flujo adecuadas a los fenómenos característicos que afectan al tipo de flujo que se da en estos casos de redes de tránsito vehicular. Se consideran tres clases de funciones de correlación de flujo. Pero antes se da una descripción más específica de la red que se utiliza y que está definida con una base de datos que describe porciones de la red vial principal de una ciudad, en este caso del Distrito Federal ó Ciudad de México. Cada entrada describe un tramo ó calle con 8 atributos, que son los siguientes (ver figura 6).

1. Identificador.
2. Nombre de la calle.
3. Un código que determina si la calle es de alta, media, baja o nula capacidad de flujo.
4. Un código que determina las *direcciones* en las que es posible atravesar la calle.
5. Coordenada cartesiana x del vértice inicial de la calle.
6. Coordenada cartesiana y del vértice inicial de la calle.
7. Coordenada cartesiana x del vértice terminal de la calle.
8. Coordenada cartesiana y del vértice terminal de la calle.

	A	B	C	D	E	F	G	H
1	#	Nombre de la calle	Tipo	Sentido	Xi	Yi	Xt	Yt
2	1	11 DE ABRIL	-5	b	-99.1809	19.397	-99.1797	19.3972
3	2	11 DE ABRIL	-5	b	-99.1822	19.3969	-99.1814	19.397
4	3	11 DE ABRIL	-5	b	-99.1824	19.3968	-99.1822	19.3969
5	4	11 DE ABRIL	-5	b	-99.1834	19.3965	-99.1824	19.3968
6	5	11 DE ABRIL	-5	b	-99.1843	19.3962	-99.1834	19.3965
7	6	11 DE ABRIL	-5	a	-99.1843	19.3962	-99.1854	19.3959
8	7	11 DE ABRIL	-5	a	-99.1854	19.3959	-99.1859	19.3957
9	8	11 DE ABRIL	-5	a	-99.1859	19.3957	-99.1865	19.3956
10	9	11 DE ABRIL	-5	a	-99.1865	19.3956	-99.1875	19.3956
11	10	11 DE ABRIL	-5	a	-99.1875	19.3956	-99.1878	19.3956
12	11	11 DE ABRIL	-5	a	-99.1878	19.3956	-99.1885	19.3956
13	12	11 DE ABRIL	-5	a	-99.1885	19.3956	-99.1888	19.3956
14	13	1ER CJON AV LA LUZ	-5	c	-99.1418	19.3759	-99.142	19.3745
15	14	1ER RET CORREGGIO	-5	c	-99.1826	19.38	-99.1832	19.38
16	15	1RA CDA ARIZONA	-5	a	-99.179	19.3935	-99.1793	19.3943
17	16	1RA CDA MILLET	-5	c	-99.1803	19.3769	-99.1804	19.3763
18	17	1RA CDA PARQUE	-5	a	-99.1608	19.3607	-99.1613	19.3605
19	18	1RA PRIV CARRACCI	-5	c	-99.1834	19.3767	-99.1839	19.3768
20	19	1RO DE MAYO EJE 5 SUR	-24	b	-99.1352	19.3822	-99.1346	19.3822
21	20	1RO DE MAYO EJE 5 SUR	-24	b	-99.1396	19.3828	-99.1389	19.3828
22	21	1RO DE MAYO EJE 5 SUR	-24	b	-99.1389	19.3828	-99.1384	19.3827
23	22	1RO DE MAYO EJE 5 SUR	-24	b	-99.1379	19.3826	-99.1373	19.3826
24	23	1RO DE MAYO EJE 5 SUR	-24	b	-99.1384	19.3827	-99.1379	19.3826
25	24	1RO DE MAYO EJE 5 SUR	-24	a	-99.1368	19.3825	-99.1362	19.3824
26	25	1RO DE MAYO EJE 5 SUR	-24	b	-99.1373	19.3826	-99.1368	19.3825
27	26	1RO DE MAYO EJE 5 SUR	-24	b	-99.1362	19.3824	-99.1357	19.3823
28	27	1RO DE MAYO EJE 5 SUR	-24	b	-99.1357	19.3823	-99.1352	19.3822
29	28	2 DE ABRIL	-5	b	-99.1805	19.3646	-99.1808	19.3639
30	29	2 DE ABRIL	-5	b	-99.1808	19.3639	-99.181	19.363

Figura 6. Base de datos que describe la cartografía.

Con estos datos se obtiene entonces una red $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ en donde las coordenadas de los vértices iniciales y terminales de cada calle conforman el conjunto $\mathcal{V}$ de los vértices. Nosotros sólo consideraremos calles con la misma capacidad de flujo, es decir, calles con el mismo código dado por 3. Las aristas están determinadas por la condición de dirección

en 4 que toma uno de cuatro posibles valores $\{a, b, c, d\}$, de forma que si es a, entonces es posible atravesar la calle en la dirección de la arista, si es b, entonces es posible atravesar la calle en la dirección contraria de la arista, si es c, entonces es posible atravesar la calle en ambas direcciones, y finalmente si es d, no es posible atravesar la calle en ninguna dirección y no corresponde a ninguna arista. De esta forma podemos asumir que se cuenta con una red en donde los vehículos atraviezan cada tramo de acuerdo con la dirección de la flecha en la red. El valor de un flujo positivo $f\colon \mathcal{A} \to \mathbb{R}_+$ en una arista puede ser interpretado como la cantidad ó bien como la velocidad promedio de los vehículos que atraviesan la calle correspondiente en un instante de tiempo específico. Es común relacionar estas dos unidades en un *diagrama fundamental*, como se describe en [9] (ver figura 7).

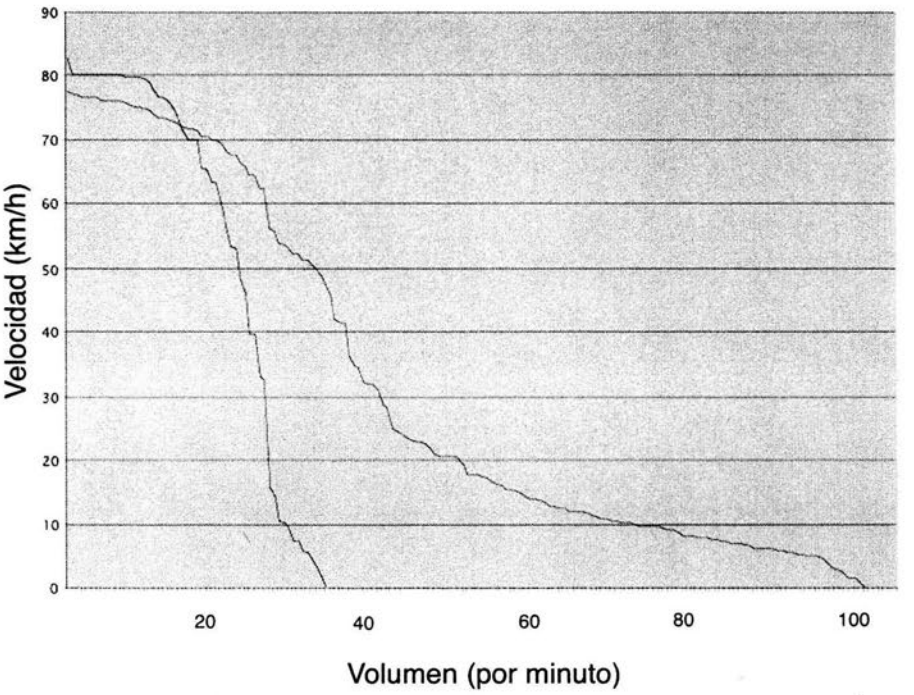

Figura 7. Un diagrama fundamental

Suponemos entonces que un flujo parcial $f\colon \mathcal{A}_0 \to \mathbb{R}_+$ es conocido y queremos obtener extensiones con el modelo descrito en la sección anterior, usando tres tipos de funciones de correlación de flujo que son las que se describen en las siguientes subsecciones, donde para cada $a \in \mathcal{A}$ se denota por $N(a)$ al nombre de a que se obtiene de 2, por $(x_i(a), y_i(a))$ a las

coordenadas xy del vértice inicial de a que se obtienen de 5 y 6, y por $(x_t(a), y_t(a))$ a las coordenadas xy del vértice terminal de a que se obtienen de 7 y 8. La condición de capacidad 3 puede ser usada para obtener extensiones mediante iteraciones, pero como lo mencionamos anteriormente, sólo consideraremos calles con la misma capacidad de flujo. Dos de estas tres funciones de correlación de flujo vienen con una familia de parámetros $\Theta = \{\theta_1, \theta_2\}$ que representan tazas de decaimiento, y está también el vector de probabilidad $\Pi = (\pi_0, \pi_1, \pi_2)$ necesario en 2.6 en caso de considerar combinaciones entre estas funciones de correlación de flujo.

La implementación que se discute en la siguiente sección produce una imagen de la red de tránsito vehicular. Cada calle es dibujada de acuerdo al código de dirección con dos, uno o ningún círculos coloreados adyacentes a los lados del segmento de recta que une los vértices iniciales y terminales. La idea es la de que el color del círculo en uno de los lados del segmento indica el nivel de tráfico que hay de ese lado de la calle, asumiendo que se conduce del lado derecho y no como en Inglaterra, y donde el punto es coloreado en una gama de colores de forma que el corrimiento al verde indica tráfico fluido y el corrimiento al rojo indica tráfico pesado. La siguiente figura 8 muestra ejemplos de las cuatro posibles circunstancias.

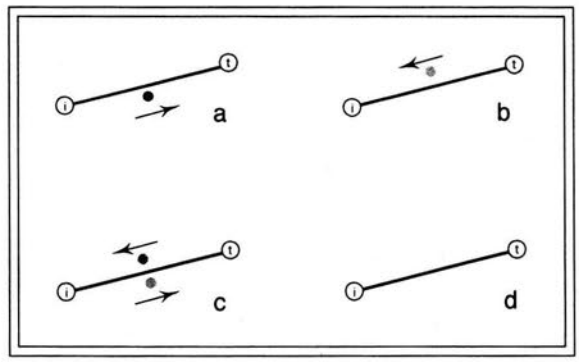

Figura 8. Las cuatro posibles direcciones del flujo en una calle.

2.3. Ejemplos

Los siguientes ejemplos están construidos en base a porciones (delegaciones) de las vías principales de la red de tránsito urbana del Distrito Federal.

2.3.1. Nombre

La primera función de correlación de flujo está definida en términos del nombre del tramo. Concretamente, para cada $a, b \in \mathcal{A}$, sea

$$\rho_0(a,b) = \begin{cases} 1 & \text{si } N(a) = N(b) \\ 0 & \text{si } N(a) \neq N(b) \end{cases}$$

Es natural pensar que si en un tramo con un nombre hay cierto flujo, los tramos con el mismo nombre tendrán un flujo semejante, y este es el razonamiento que da fundamento a esta función de correlación de flujo ρ_0.

El siguiente es un ejemplo de la extensión que se obtiene al sólo considerar la función de correlación de flujo ρ_0 definida en términos del nombre.

Ejemplo 1. La siguiente figura 9 representa el mapa de las principales vías en la delegación Coyoacán, y se observa el flujo vehicular parcial en algunas de sus avenidas.

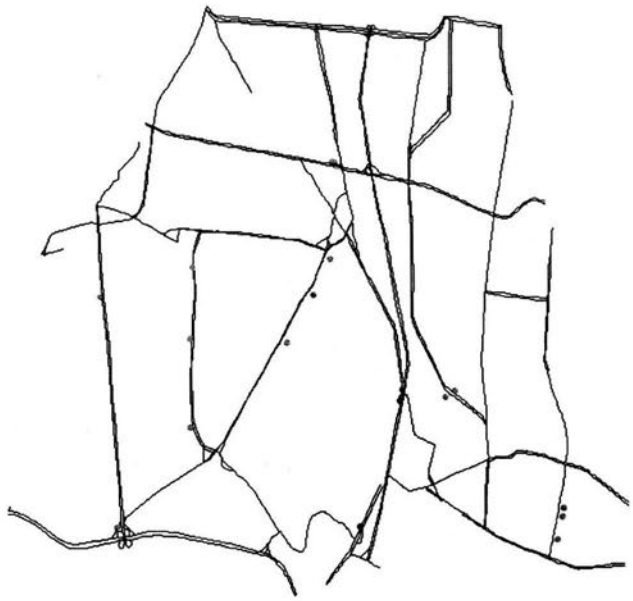

Figura 9. Flujo parcial en la delegación Coyoacán.

Una vez que aplicamos el modelo, obtenemos la extensión correspondiente la cual se muestra en la siguiente figura 10.

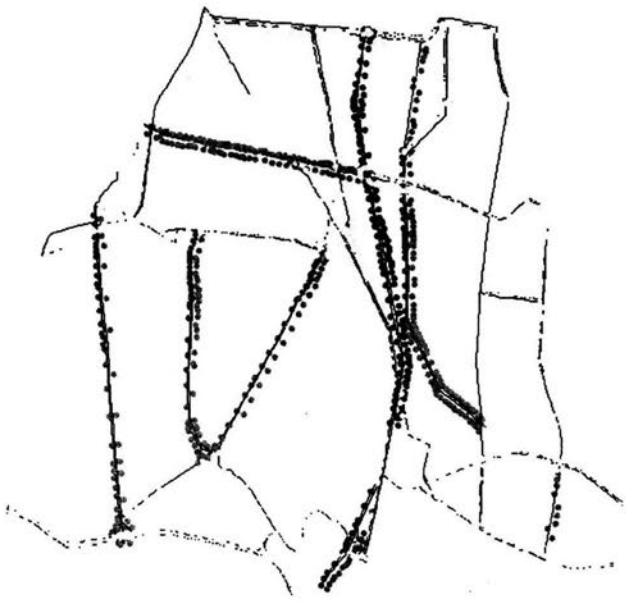

Figura 10. Extensión del flujo de la red en la figura 9.

Tanto los errores locales como el error global asociados a nuestra extensión anterior, se muestran en la figura 11. El punto negro a la derecha indica la velocidad promedio.

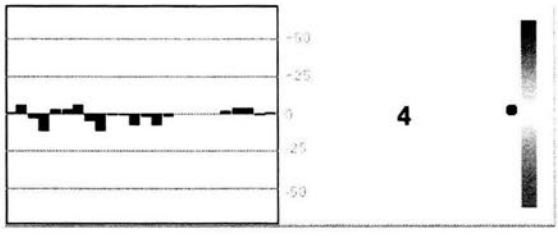

Figura 11. Errores locales, error global y velocidad promedio.

□

2.3.2. Distancia

La segunda función de correlación de flujo está definida en términos de la posición del tramo. La función $d\colon \mathcal{A} \times \mathcal{A} \to \mathbb{R}_+$ definida para cada $a, b \in \mathcal{A}$ por

$$d(a,b) = |x_i(a) - x_i(b)| + |y_i(a) - y_i(b)| + |x_t(a) - x_t(b)| + |y_t(a) - y_t(b)|$$

es una medida de la *cercanía* o *lejanía* entre los tramos a y b. Dada $\theta_1 \geq 0$, se define

$$\rho_1(a,b) = e^{-\theta_1 d(a,b)}.$$

Es natural pensar que si en un tramo hay cierto flujo, los tramos cercanos tendrán flujos semejantes, y este razonamiento es el que da fundamento a esta función de correlación de flujo ρ_1 y cuyo comportamiento está descrito en la figura 12.

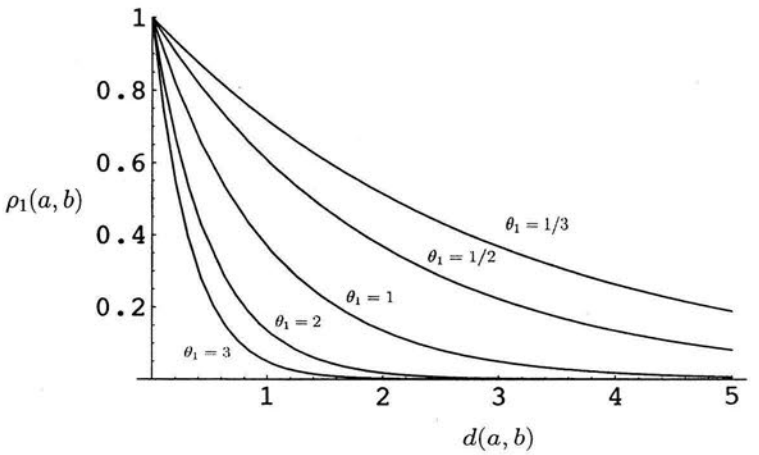

Figura 12. La distancia como función de correlación de flujo.

El siguiente ejemplo muestra la clase de extensión que se obtiene al sólo considerar la función de correlación de flujo ρ_1 definida en términos de la distancia.

Ejemplo 2. La siguiente figura 13, representa el mapa vial de los tramos principales de la delegación Benito Juárez en el cual se observa el flujo vehicular parcial en algunas de sus arterias.

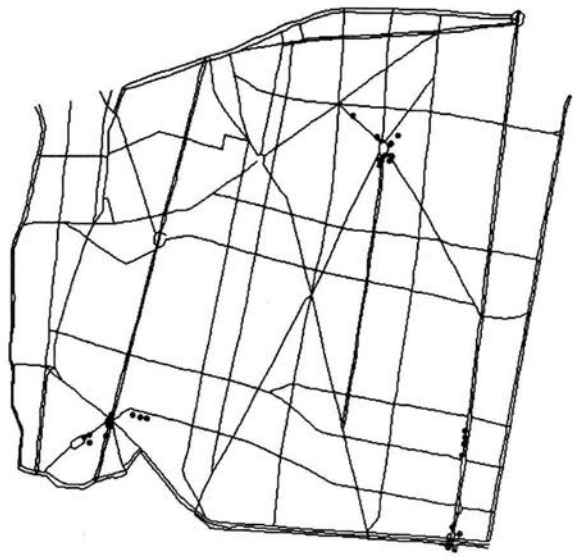

Figura 13. Flujo parcial en la delegación Benito Juárez.

Una vez que aplicamos el modelo, obtenemos la extensión correspondiente la cual se muestra en la figura 14.

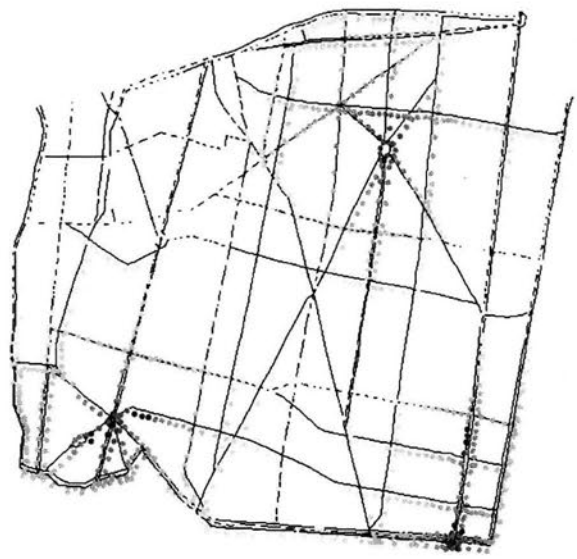

Figura 14. Extensión del flujo de la red en la figura 12.

Tanto los errores locales como el error global y la velocidad promedio asociados a nuestra extensión anterior, se muestran en la figura 15. La taza de decaimiento que se utilizó fue de $\theta_1 = 50$.

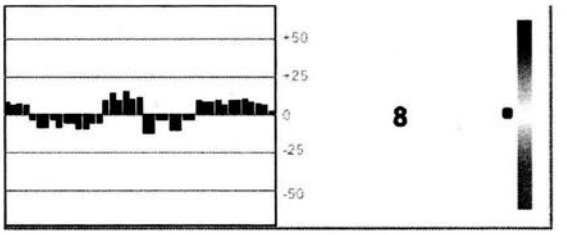

Figura 15. Errores locales, error global y velocidad promedio.

□

2.3.3. Orientación

Finalmente se define una función de correlación de flujo en base a la orientación de los tramos. Para cada $a, b \in \mathcal{A}$, sea $\alpha(a, b)$ el ángulo que determinan los vectores a y b, Dada $\theta_2 \geq 0$, definimos

$$\rho_2(a, b) = \left(\frac{1 + \cos(\alpha(a, b))}{2}\right)^{\theta_2}.$$

Es natural pensar que si en un tramo hay cierto flujo, los tramos con direcciones semejantes también tendrán flujos semejantes, y este razonamiento es el que da fundamento a esta función de correlación de flujo ρ_2 y cuyo comportamiento está descrito en la figura 16.

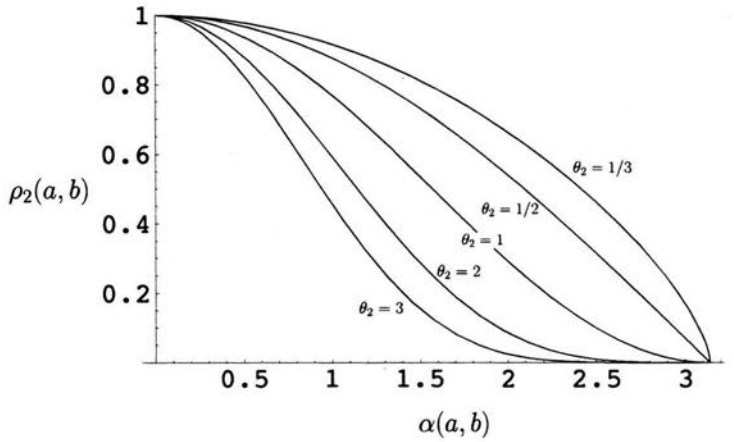

Figura 16. La orientación como función de correlación de flujo.

El siguiente ejemplo muestra la clase de extensión que se obtiene al sólo considerar la función de correlación de flujo ρ_2 definida en términos de la orientación.

Ejemplo 3. La figura 17, muestra de nuevo un flujo parcial en la delegación Benito Juárez.

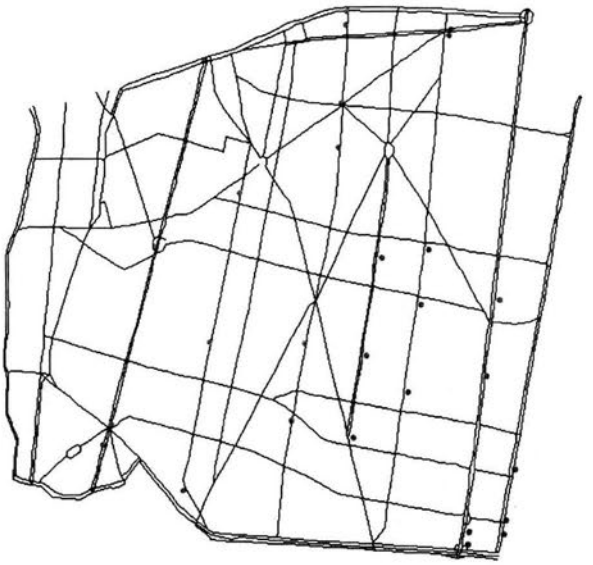

Figura 17. Otro flujo parcial en la delegación Benito Juárez.

Una vez que aplicamos el modelo, obtenemos la extensión correspondiente, la cual se muestra en la figura 18.

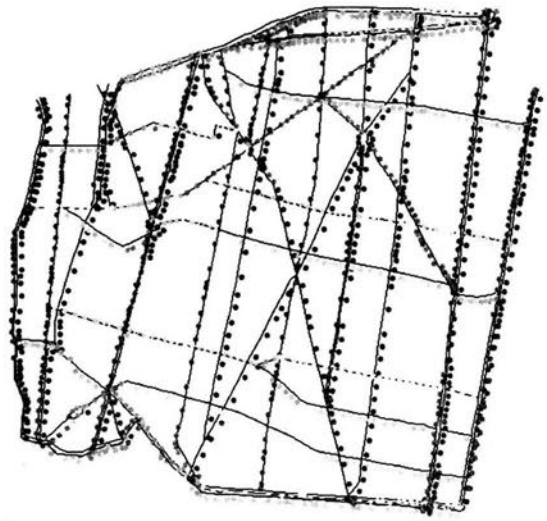

Figura 18. Extensión del flujo de la red en la figura 16.

Tanto los errores locales como el error global y la velocidad promedio asociados a nuestra extensión anterior, se muestran en la figura 19. La taza de decaimiento fue de $\theta_2 = 1$.

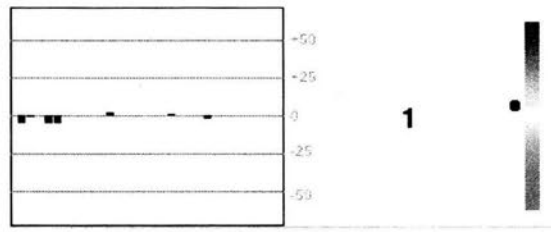

Figura 19. Errores locales, error global y velocidad promedio.

2.3.4. Combinaciones

El siguiente es un ejemplo de la extensión que se obtiene al considerar combinaciones de las tres funciones de correlación de flujo definidas anteriormente.

Ejemplo 4. Nuevamente, el mapa en la figura 20 muestra un flujo vehicular parcial en la delegación Coyoacán en algunas de sus principales arterias

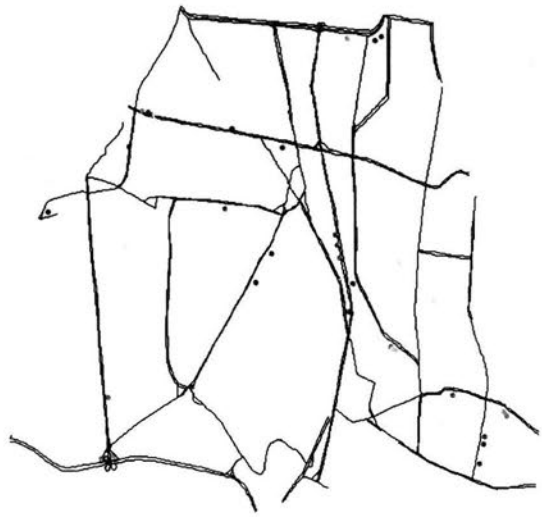

Figura 20. Flujo parcial en la delegación Coyoacán.

Aplicando nuestro modelo, obtenemos la extensión del flujo vehicular que se muestra en la figura 21. En este caso, el vector de distribución de probabilidad fue de $\pi = (60\,\%, 30\,\%, 10\,\%)$ y las tazas de decaimiento $\theta_1 = 100$ y $\theta_2 = 50$.

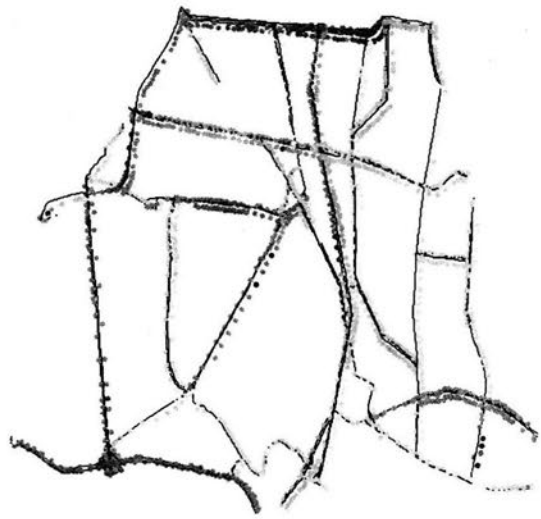

Figura 21. Flujo parcial en la delegación Coyoacán.

Los errores locales, globales y la velocidad promedio se muestran en la figura 22.

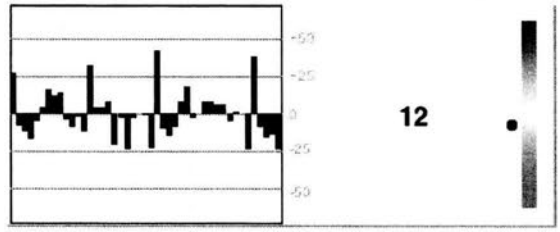

Figura 22. Errores locales, globales y velocidad promedio.

2.4. Comentarios

Como podemos observar, este trabajo es un primer paso para el estudio de las extensiones de flujo y en particular en lo correspondiente a flujos vehiculares. Falta mucho por hacer, como por ejemplo: diseñar un programa que optimice la selección de los parámetros de las funciones de correlación de flujo, así como los decaimientos óptimos de dichas funciones, lo cual puede lograrse aplicando métodos de redes neuronales.

No obstante, del estudio realizado y los resultados hasta ahora obtenidos, se desprende que el camino escogido es el correcto y se tendrá que continuar estudiando generalizaciones del modelo a fin de obtener resultados cada vez más confiables.

En el caso particular de las redes de tránsito vehicular, la eficiencia del modelo que presentamos depende en gran medida de las fuentes de información con las que se cuenta y que determinan el conjunto $\mathcal{A}_0$. Estas pueden ser de varios tipos, como cámaras de video y contadores ó *loops* (ver figura 23.a), localizadores satelitales ó GPS (Global Positioning System), fuentes estadísticas como las realizadas por la Secretaría de Transporte y Vialidad del Distrito Federal (ver [12]), diferentes clases de levantamientos en campo por medio de diferentes tecnologías como el uso de PDA (Personal Digital Assistant, ver figura 23.b), monitoreo mediante helicópteros, patrullas, etc. La obtención de ésta información, su clasificación, organización y todos los pasos que se requerirían con el fin de agruparla para alimentar el modelo constituyen una problemática claramente fuera del alcance de este trabajo. Nuestra principal intención es la de presentar el modelo matemático y mostrar su comportamiento.

(a)

(b)

Figura 23.
(a) Un ejemplo de un contador del volumen vehicular en una calle.
(b) Un ejemplo de una PDA para levantamiento de estadísticas vehiculares.

Capítulo 3

Implementación del modelo

En este capítulo presentamos la implementación del modelo que se realizó para obtener los ejemplos que se describieron en el capítulo 2, incluyendo los fragmentos del código en JAVA que se escribieron para realizar los distintos cálculos que se requieren para obtener las extensiones (una introducción a JAVA lo es [4]). Incluiremos todo el código del programa, inlusive el de la GUI (Graphic User Interface) que generó automaticamente NetBeans (ver figura 23), que fue el IDE (Interface Developer Environment) que se utilizó, y también las rutinas de lectura de archivos, que son las que determinan la base de datos almacenada en la memoria del programa, así como también las clases `map` y `error` que son las que generaron las gráficas que se presentaron.

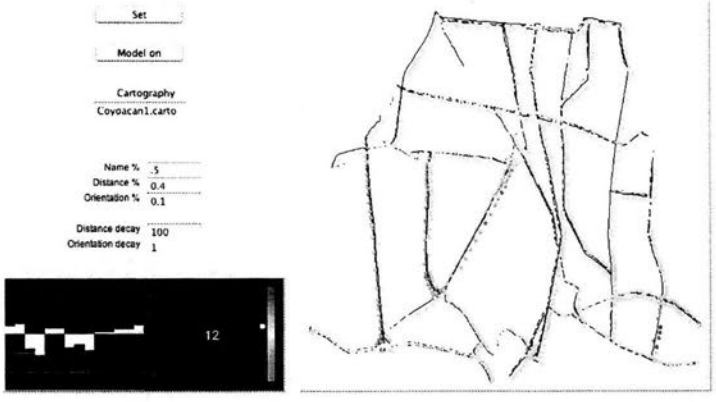

Figura 23. Implementación del modelo.

3.1. La base de datos almacenada en el código

Las entradas de la base de datos se almacenaron en un objeto determinado por la clase `street` cuyo código es el siguiente.

```
public class street extends Object{
    private String name; /* Nombre de la calle */
    private double startX; /* Coordenada X inicial */
    private double startY; /* Coordenada Y inicial */
    private double endX; /* Coordenada X terminal */
    private double endY; /* Coordenada Y terminal */
    private double traffic; /* Flujo de la calle */
    private int code; /* Determina si se conoce el flujo */

    public street() { /* Constructor (inicializa todos los
                datos del objeto street */
        setName("");
        setStartX(0.0);
        setStartY(0.0);
        setEndX(0.0);
        setEndY(0.0);
        setTraffic(0.0);
        setCode(0);
    }
    /* Metodos para establecer la informacion del objeto street */
    public void setName(String n) {name = n;}
    public void setStartX(double xi) {startX = xi;}
    public void setStartY(double yi) {startY = yi;}
    public void setEndX(double xt) {endX = xt;}
    public void setEndY(double yt) {endY = yt;}
    public void setTraffic(double t) {traffic = t;}
    public void setCode(int c) {code = c;}
    /* Metodos para obtener la informacion del objeto street */
    public String getName() {return name;}
    public double getStartX() {return startX;}
    public double getStartY() {return startY;}
    public double getEndX() {return endX;}
    public double getEndY() {return endY;}
    public double getTraffic() {return traffic;}
    public int getCode() {return code;}
}
```

3.2. El modelo en código

A continuación presentamos la sección del código que realiza los cálculos del modelo, una vez que ha sido cargada la base de datos.

```
public void model() {
    double trafficMean;
    int trafficSize;
    Vector tau;

    trafficMean = 0.0;
    trafficSize = 0;

    /* Se calcula el promedio de los flujos en A_0 */
    for(int n = 0; n < carto.size(); n++) {
        if (((street) carto.get(n)).getCode() == 1) {
            trafficMean = trafficMean + ((street) carto.get(n)).getTraffic();
            trafficSize++;
        }
    }
    trafficMean = trafficMean / trafficSize;

    /* Se calculan los valores de tau */
    tau = new Vector();
    for(int n = 0; n < carto.size(); n++) {
        street temp1 = (street) carto.get(n);
        if(temp1.getCode() == 1) {
            tau.add(new Double(1.0));
        }
        else {
            double sumRho;
            sumRho = 0.0;
            for(int m = 0; m < carto.size(); m++) {
                street temp2 = (street) carto.get(m);
                if(temp2.getCode() == 1) {
                    sumRho = sumRho + rho(temp2, temp1);
                }
            }
            tau.add(new Double(sumRho));
        }
    }
```

```
        /* Se calculan la extension */
        for(int n = 0; n < carto.size(); n++) {
            street temp1 = (street) carto.get(n);
            if(temp1.getCode() == 0) {
                double prediction;
                double sumPonderateRho;
                sumPonderateRho = 0.0;
                for(int m = 0; m < carto.size(); m++) {
                    street temp2 = (street) carto.get(m);
                    if(temp2.getCode() == 1) {
                        double t;
                        t = rho(temp2, temp1);
                        sumPonderateRho = sumPonderateRho +
                           t * (t * temp2.getTraffic() + (1 - t) * trafficMean);
                    }
                }
                prediction =
                  (1 / ((Double) tau.get(n)).doubleValue()) * sumPonderateRho;
                temp1.setTraffic(prediction);
                carto.setElementAt(temp1, n);
            }
        }
    }
```

3.3. Las funciones de correlación de flujo en código

Los siguientes métodos son los que constituyen las funciones de correlación de flujo que se usaron en el modelo.

Función de correlación de flujo en términos del nombre (sec. 2.3.1).

```
    public double rhoName(street a, street b) {
        if (a.getName().equals(b.getName())) {
            return 1.0;
        }
        else {
            return 0.0;
        }
```

```
}
```

Función de correlación de flujo en términos de la distancia (sec. 2.3.2).

```
public double distance(street a, street b) {
    return Math.abs(a.getStartX() - b.getStartX()) +
           Math.abs(a.getEndX() -b.getEndX()) +
           Math.abs(a.getStartY() - b.getStartY()) +
           Math.abs(a.getEndY() - b.getEndY());
}

public double rhoDistance(street a, street b) {
    return Math.exp(-theta1 * distance(a, b));
}
```

Función de correlación de flujo en términos de la distancia (sec. 2.3.3).

```
public double norm(street a) {
    return Math.sqrt(Math.pow(a.getEndX() - a.getStartX(), 2)
                   + Math.pow(a.getEndY() - a.getStartY(), 2));
}

public double rhoOrientation(street a, street b) {
    double cosAlpha =
        ((a.getEndX() - a.getStartX()) * (b.getEndX() - b.getStartX()) +
         (a.getEndY() - a.getStartY()) * (b.getEndY() - b.getStartY())) /
         (norm(a) * norm(b));
    return Math.pow((1 + cosAlpha) / 2, theta2);
}
```

Combinación de las funciones de correlación de flujo anteriores.

```
public double rho(street a, street b) {
    return pi1 * rhoName(a, b) +
           pi2 * rhoDistance(a, b) +
           pi3 * rhoOrientation(a, b);
}
```

3.4. El error en código

Para calcular los errores, se puede repetir el código del modelo que describimos anteriormente, sólo tenemos que cambiar el subconjunto A_0. El código se lee a continuación.

```
public Vector modelError() {
    double trafficMean;
    int trafficSize;
    Vector tau;
    Vector cartoError;
    Vector errors;

    cartoError = new Vector();
    errors = new Vector();

    for(int k = 0; k < cartoError.size(); k++) {
        street temp = (street) cartoError.get(k);
        temp.setCode(0); /* Se quita el k-esimo elemento de A_0 */
        cartoError.setElementAt(temp, k);

        trafficMean = 0.0;
        trafficSize = 0;

        for(int n = 0; n < cartoError.size(); n++) {
            if (((street) cartoError.get(n)).getCode() == 1) {
                trafficMean =
                    trafficMean + ((street) cartoError.get(n)).getTraffic();
                trafficSize++;
            }
        }
        trafficMean = trafficMean / trafficSize;

        tau = new Vector();
        for(int n = 0; n < cartoError.size(); n++) {
            street temp1 = (street) cartoError.get(n);
            if(temp1.getCode() == 0) {
                double sumRho;
                sumRho = 0.0;
                for(int m = 0; m < cartoError.size(); m++) {
                    street temp2 = (street) cartoError.get(m);
                    if(temp2.getCode() == 1) {
```

```
                        sumRho = sumRho + rho(temp2, temp1);
                    }
                }
                tau.add(new Double(sumRho));
            }
            else {
                tau.add(new Double(1.0));
            }
        }

        for(int n = 0; n < cartoError.size(); n++) {
            street temp1 = (street) cartoError.get(n);
            if(temp1.getCode() == 0) {
                double prediction;
                double sumPonderateRho;
                sumPonderateRho = 0.0;
                for(int m = 0; m < cartoError.size(); m++) {
                    street temp2 = (street) cartoError.get(m);
                    if(temp2.getCode() == 1) {
                        double t;
                        t = rho(temp2, temp1);
                        sumPonderateRho = sumPonderateRho +
                         t * (t * temp2.getTraffic() + (1 - t) * trafficMean);
                    }
                }
                prediction =
                  (1 / ((Double) tau.get(n)).doubleValue()) * sumPonderateRho;
                errors.add(new Double(temp1.getTraffic() - prediction));
                temp1.setCode(1); /* Se reestablece el k-esimo elemento de A_0 */
                cartoError.setElementAt(temp1, n);
            }
        }
    }
    return errors;
}
```

3.5. Lectura de datos

A continuación presentamos el método que lee la cartografía de la base de datos y la carga en la memoria.

```
private void readCarto() {
    BufferedReader input;
    String data;
    street temp;
    String way;
    double trafficF, trafficB;
    double xi, yi, xt, yt;
    int n, i, j;
    boolean moreRecords = true;

    carto = new Vector();

    n = 0;

    try {
        input = new BufferedReader(new FileReader(cartoTextField.getText()));
        while (moreRecords) {
            try {
                data = new String(input.readLine());
                i = 0;
                j = 0;

                temp = new street();

                i = data.indexOf( (int) '\t', j);
                j = ++i;
                i = data.indexOf( (int) '\t', j);
                temp.setName(data.substring(j, i));
                j = ++i;
                i = data.indexOf( (int) '\t', j);
                j = ++i;
                i = data.indexOf( (int) '\t', j);
                way = data.substring(j, i);
                j = ++i;
                i = data.indexOf( (int) '\t', j);
                xi = Double.parseDouble(data.substring(j, i));
```

```
j = ++i;
i = data.indexOf( (int) '\t', j);
yi = Double.parseDouble(data.substring(j, i));
j = ++i;
i = data.indexOf( (int) '\t', j);
xt = Double.parseDouble(data.substring(j, i));
j = ++i;
i = data.indexOf( (int) '\t', j);
yt = Double.parseDouble(data.substring(j, i));
j = ++i;
i = data.indexOf( (int) '\t', j);
trafficF = Double.parseDouble(data.substring(j, i));
j = ++i;
trafficB = Double.parseDouble(data.substring(j));

if (way.equals("a")) {
    temp.setStartX(xi);
    temp.setStartY(yi);
    temp.setEndX(xt);
    temp.setEndY(yt);
    temp.setTraffic(trafficF);
    if (trafficF > 0) {
        temp.setCode(1);
    }
    else {
        temp.setCode(0);
    }
    carto.addElement(temp);
}
else if (way.equals("b")) {
    temp.setStartX(xt);
    temp.setStartY(yt);
    temp.setEndX(xi);
    temp.setEndY(yi);
    temp.setTraffic(trafficB);
    if (trafficB > 0) {
        temp.setCode(1);
    }
    else {
        temp.setCode(0);
    }
    carto.addElement(temp);
```

```
            }
            else if (way.equals("c")) {
                temp.setStartX(xi);
                temp.setStartY(yi);
                temp.setEndX(xt);
                temp.setEndY(yt);
                temp.setTraffic(trafficF);
                if (trafficF > 0) {
                    temp.setCode(1);
                }
                else {
                    temp.setCode(0);
                }
                carto.addElement(temp);

                temp.setStartX(xt);
                temp.setStartY(yt);
                temp.setEndX(xi);
                temp.setEndY(yi);
                temp.setTraffic(trafficB);
                if (trafficB > 0) {
                    temp.setCode(1);
                }
                else {
                    temp.setCode(0);
                }
                carto.addElement(temp);
            }
        }
        catch (java.io.IOException e) {
            moreRecords = false;
        }
        catch (java.lang.NullPointerException e) {
            moreRecords = false;
        }
    }
    try {
        input.close();
    }
    catch (java.io.IOException e) {
        System.err.println("Can't close file..." + e.toString());
        System.exit(1);
```

```
            }
        }
        catch (java.io.IOException e) {
            System.err.println("Can't open file... " + e.toString());
            System.exit(1);
        }
    }
```

3.6. Dibujo del mapa

El siguiente es el código que pinta el mapa.

```
public class map extends java.awt.Canvas {
    private java.util.Vector carto;

    /** Crea una nueva instancia de map */
    public map() {
    }

    public void paint(java.awt.Graphics g) {
        g.setColor(java.awt.Color.white);
        g.fillRect(0, 0, this.getWidth(), this.getHeight());
        try {
            double Xmin = ((street) carto.get(0)).getStartX();
            double Xmax = Xmin;
            double Ymin = ((street) carto.get(0)).getStartY();
            double Ymax = Ymin;
            for(int n = 0; n < carto.size(); n++) {
                if (((street) carto.get(n)).getStartX() > Xmax) {
                    Xmax = ((street) carto.get(n)).getStartX();
                }
                if (((street) carto.get(n)).getEndX() > Xmax) {
                    Xmax = ((street) carto.get(n)).getEndX();
                }
                if (((street) carto.get(n)).getStartY() > Ymax) {
                    Ymax = ((street) carto.get(n)).getStartY();
                }
                if (((street) carto.get(n)).getEndY() > Ymax) {
```

```
            Ymax = ((street) carto.get(n)).getEndY();
        }
        if (((street) carto.get(n)).getStartX() < Xmin) {
            Xmin = ((street) carto.get(n)).getStartX();
        }
        if (((street) carto.get(n)).getEndX() < Xmin) {
            Xmin = ((street) carto.get(n)).getEndX();
        }
        if (((street) carto.get(n)).getStartY() < Ymin) {
            Ymin = ((street) carto.get(n)).getStartY();
        }
        if (((street) carto.get(n)).getEndY() < Ymin) {
            Ymin = ((street) carto.get(n)).getEndY();
        }
    }
    double adjustX = (Xmax - Xmin) / 50;
    double adjustY = (Ymax - Ymin) / 50;
    Xmax = Xmax + adjustX + (int) (Xmax -Xmin) / 5;
    Xmin = Xmin - adjustX - (int) (Xmax -Xmin) / 5;
    Ymax = Ymax + adjustY + (int) (Ymax -Ymin) / 5;
    Ymin = Ymin - adjustY - (int) (Ymax -Ymin) / 5;
    int XMax = this.getWidth() - 1;
    int YMax = this.getHeight() - 1;
    for(int n = 0; n < carto.size(); n++) {
        int xi = (int) ((((street) carto.get(n)).getStartX() - Xmin)
           * XMax / (Xmax - Xmin));
        int yi = (int) ((((street) carto.get(n)).getStartY() - Ymin)
           * YMax / (Ymax - Ymin));
        int xt = (int) ((((street) carto.get(n)).getEndX() - Xmin)
           * XMax / (Xmax - Xmin));
        int yt = (int) ((((street) carto.get(n)).getEndY() - Ymin)
           * YMax / (Ymax - Ymin));

        g.setColor(java.awt.Color.black);
        g.drawLine(xi, YMax - yi, xt, YMax - yt);

        if (((street) carto.get(n)).getTraffic() > 40) {
            if (((street) carto.get(n)).getTraffic() < 80) {
                g.setColor(new java.awt.Color (510 -
                  (int) ((street) carto.get(n)).getTraffic() *
                  510 / 80 , 255, 0));
            }
```

```
                    else {
                        g.setColor(java.awt.Color.green);
                    }
                }
                else {
                    if (((street) carto.get(n)).getTraffic() > 0) {
                        g.setColor(new java.awt.Color (255,
                          (int) ((street) carto.get(n)).getTraffic() *
                          510 / 80, 0));
                    }
                    else {
                        g.setColor(java.awt.Color.red);
                    }
                }

                if (((street) carto.get(n)).getTraffic() > 0) {
                    g.fillOval((new Double((xi + xt) / 2 + 3 * (yt - yi) /
                      Math.sqrt(Math.pow(xi - xt, 2) +
                      Math.pow(yi - yt, 2)))).intValue(),
                      YMax - (new Double((yi + yt) / 2 - 3 * (xt - xi) /
                      Math.sqrt(Math.pow(xi - xt, 2) +
                      Math.pow(yi - yt, 2)))).intValue(), 4, 4);
                }
            }
        }
        catch (NullPointerException e) {
            ;
        }
    }

    void setCarto(java.util.Vector c) {
        carto = new java.util.Vector();
        for (int n = 0; n < c.size(); n++) {
            carto.addElement(c.get(n));
        }
    }

}
```

3.7. Gráfica de los errores

El siguiente es el código que pinta la gráfica de los errores locales, globales y la velocidad promedio.

```
public class error extends java.awt.Canvas {
    private java.util.Vector localError;
    private double globalError;
    private int trafficMean;

    /** Crea una nueva instancia de error */
    public error() {
    }

    public void paint(java.awt.Graphics g) {

        g.setColor(java.awt.Color.white);
        g.fillRect(0, 0, this.getWidth(), this.getHeight());

        // Dibuja los ejes para la grafica del error
        int Xmax, Ymax;
        Xmax = this.getWidth() - 1;
        Ymax = this.getHeight() - 1;
        g.setColor(java.awt.Color.black);
        g.drawRect(0, 0, (new Integer(Xmax / 2)).intValue(), Ymax);
        for (int k = 0; k < 2; k++) {
            g.setColor(java.awt.Color.gray);
            g.drawLine(0, (int) Ymax / 2, (new Integer(Xmax / 2)).intValue(),
               Ymax / 2);
            g.drawLine(0, (int) Ymax / 2 + 25, (new Integer(Xmax / 2)).intValue(),
               Ymax / 2 + 25);
            g.drawLine(0, (int) Ymax / 2 - 25, (new Integer(Xmax / 2)).intValue(),
               Ymax / 2 - 25);
            g.drawLine(0, (int) Ymax / 2 + 50, (new Integer(Xmax / 2)).intValue(),
               Ymax / 2 + 50);
            g.drawLine(0, (int) Ymax / 2 - 50, (new Integer(Xmax / 2)).intValue(),
               Ymax / 2 - 50);
        }
        g.setFont(new java.awt.Font("Dialog", 0, 10));
        g.drawString("0", (new Integer(Xmax / 2)).intValue() + 5, Ymax / 2 + 4);
        g.drawString("+25", (new Integer(Xmax / 2)).intValue() + 5, Ymax / 2 - 22);
        g.drawString("-25", (new Integer(Xmax / 2)).intValue() + 5, Ymax / 2 + 27);
```

```
g.drawString("+50", (new Integer(Xmax / 2)).intValue() + 5, Ymax / 2 - 47);
g.drawString("-50", (new Integer(Xmax / 2)).intValue() + 5, Ymax / 2 + 56);
g.setColor(java.awt.Color.black);
g.setFont(new java.awt.Font("Dialgo", 1, 18));
g.drawString(new Integer((new Double(globalError).intValue())).toString(),
   (new Integer(23 * Xmax / 32)).intValue(),
   (new Integer(Ymax / 2)).intValue() + 8);

// Dibuja el medidor de trafico
for (int k = 0; k < 512; k = k + 4) {
    if (k < 256) {
        g.setColor(new java.awt.Color (255, k, 0));
    }
    else if (k > 255) {
        g.setColor(new java.awt.Color (511 - k, 255, 0));
    }
    g.fillRect(Xmax - 20, 10 + k * (Ymax - 20) / 512, 10, 1);
}

g.setColor(java.awt.Color.black);

if(trafficMean < 80) {
    g.fillOval(Xmax - 30, 6 + trafficMean * (Ymax - 20) / 80, 6, 6);
}
else {
    g.fillOval(Xmax - 30, Ymax - 20, 6, 6);
}

try {
    for (int k = 0; k < localError.size(); k++) {
        if ((int) Xmax / (2 * localError.size()) > 0) {
            if (((Double) localError.get(k)).intValue() > 0) {
                g.fillRect((int) k * Xmax / (2 * localError.size()),
                   (int) Ymax / 2, (int) Xmax / (2 * localError.size()),
                   ((Double) localError.get(k)).intValue());
            }
            else {
                g.fillRect((int) k * Xmax / (2 * localError.size()),
                   (int) Ymax / 2 + ((Double) localError.get(k)).intValue(),
                   (int) Xmax / (2 * localError.size()),
                   -((Double) localError.get(k)).intValue());
            }
```

```
                }
                else {
                    g.fillRect((int) k * Xmax / (2 * localError.size()),
                       (int) Ymax / 2, 1, ((Double) localError.get(k)).intValue());
                }
            }
        }
        catch (NullPointerException e) {
        }
    }


    public void setErrors(java.util.Vector e) {
        globalError = 0.0;
        localError = new java.util.Vector();
        for (int n = 0; n < e.size(); n++) {
            localError.addElement(e.get(n));
            globalError = globalError + Math.abs(((Double) e.get(n)).doubleValue());
        }
        if(e.size() > 0) {
            globalError = globalError / localError.size();
        }
    }

    public void setTrafficMean(int tm) {
        trafficMean = tm;
    }
}
```

3.8. La GUI

El siguiente es el código de la GUI.

```
    private void initComponents() {
        pi1TextField = new javax.swing.JTextField();
        pi2TextField = new javax.swing.JTextField();
        pi3TextField = new javax.swing.JTextField();
        theta1TextField = new javax.swing.JTextField();
        theta2TextField = new javax.swing.JTextField();
```

```
pi1Label = new javax.swing.JLabel();
pi2Label = new javax.swing.JLabel();
pi3Label = new javax.swing.JLabel();
theta1Label = new javax.swing.JLabel();
theta2Label = new javax.swing.JLabel();
cartoTextField = new javax.swing.JTextField();
mapPanel = new javax.swing.JPanel();
cartoLabel = new javax.swing.JLabel();
setButton = new javax.swing.JButton();
modelButton = new javax.swing.JButton();
errorPanel = new javax.swing.JPanel();

getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

setTitle("TRAFFIC CONSOLE");
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        exitForm(evt);
    }
});

pi1TextField.setText(".33");
getContentPane().add(pi1TextField,
new org.netbeans.lib.awtextra.AbsoluteConstraints(150, 230, 70, 20));

pi2TextField.setText(".33");
getContentPane().add(pi2TextField,
new org.netbeans.lib.awtextra.AbsoluteConstraints(150, 250, 70, -1));

pi3TextField.setText(".33");
getContentPane().add(pi3TextField,
new org.netbeans.lib.awtextra.AbsoluteConstraints(150, 270, 70, -1));

theta1TextField.setText("1");
getContentPane().add(theta1TextField,
new org.netbeans.lib.awtextra.AbsoluteConstraints(150, 310, 70, -1));

theta2TextField.setText("1");
getContentPane().add(theta2TextField,
new org.netbeans.lib.awtextra.AbsoluteConstraints(150, 330, 70, -1));

pi1Label.setFont(new java.awt.Font("Arial", 1, 12));
```

```
pi1Label.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
pi1Label.setText("Name %");
getContentPane().add(pi1Label,
new org.netbeans.lib.awtextra.AbsoluteConstraints(50, 230, 90, -1));

pi2Label.setFont(new java.awt.Font("Arial", 1, 12));
pi2Label.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
pi2Label.setText("Distance %");
getContentPane().add(pi2Label,
new org.netbeans.lib.awtextra.AbsoluteConstraints(50, 250, 90, -1));

pi3Label.setFont(new java.awt.Font("Arial", 1, 12));
pi3Label.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
pi3Label.setText("Orientation %");
getContentPane().add(pi3Label,
new org.netbeans.lib.awtextra.AbsoluteConstraints(50, 270, 90, -1));

theta1Label.setFont(new java.awt.Font("Arial", 1, 12));
theta1Label.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
theta1Label.setText("Distance decay");
getContentPane().add(theta1Label,
new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 310, 120, -1));

theta2Label.setFont(new java.awt.Font("Arial", 1, 12));
theta2Label.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
theta2Label.setText("Orientation decay");
getContentPane().add(theta2Label,
new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 330, 120, -1));

cartoTextField.setText("BenitoJuarez.carto");
getContentPane().add(cartoTextField,
new org.netbeans.lib.awtextra.AbsoluteConstraints(100, 150, 120, -1));

mapPanel.setLayout(new java.awt.GridLayout(1, 1));

mapPanel.setBackground(new java.awt.Color(255, 255, 255));
mapPanel.setBorder(
new javax.swing.border.BevelBorder(javax.swing.border.BevelBorder.RAISED));
getContentPane().add(mapPanel,
new org.netbeans.lib.awtextra.AbsoluteConstraints(310, 20, 540, 510));

cartoLabel.setText("Cartography");
```

```
        getContentPane().add(cartoLabel,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 130, -1, -1));

        setButton.setText("Set");
        setButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                setButtonMouseClicked(evt);
            }
        });

        getContentPane().add(setButton,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(100, 20, 120, 30));

        modelButton.setText("Model on");
        modelButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                modelButtonMouseClicked(evt);
            }
        });

        getContentPane().add(modelButton,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(100, 70, 120, 30));

        errorPanel.setBackground(new java.awt.Color(255, 255, 255));
        errorPanel.setBorder(
        new javax.swing.border.BevelBorder(javax.swing.border.BevelBorder.RAISED));
        getContentPane().add(errorPanel,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 380, 280, 150));

        pack();
    }
```

Bibliografía

[1] M. Blank. Dynamics of traffic jams: order and chaos. Mosc. Math. J. **1** (2001), no. 1, 1-26.

[2] P. Brémaud. Markov chains. Gibbs fields, Monte Carlo simulation and queues. TAM. Springer Verlag. New York 1999.

[3] P. Bose, P. Morin, I. Stojmenović y J. Urrutia. Routing with guaranteed delivery in *ad hoc* wireless networks. Wireless Networks **0**. (2001).

[4] Cómo programar en JAVA. Deitel y Deitel. Pearson Educación. 1998.

[5] U. Fastenrath. Floating car data on a larger scale. Gesellschaft für Verkehrsdaten mbH. `www.ddg.de/pdf-dat/ddgfcd.pdf`

[6] N. Gartner, C.J. Messer y A.K. Rathi. Traffic flow theory. Transportation Research Board Special Report **165**. `http://www.tfhrc.gov/its/tft/tft.htm`

[7] R. Gómez. Flow extensions in networks. En preparación.

[8] V.B. Kwella y H. Lehman. Floating car data analysis of urban road networks. Lectures Notes for Computer Sciences. Springer, Berlin-Heidelberg.

[9] V.B. Kwella y H. Lehman. Traffic flow analysis in urban nets on the basis of floating car-data. Preprint.

[10] V.B. Kwella, H. Lehman y D. Skrobotz. BERTRAM - individuelle Verkehrsanalyse für mehr Mobilität. Der GMD - SPIEGEL. 1/2 2000

[11] R.T. Rockafellar. Network flows and monotropic optimization. *Athena Scientific.* Belmont, Massachusetts. 1998.

[12] Volúmenes de Tránsito en el Distrito Federal. Agosto a Noviembre 1996. Resumen Ejecutivo. Secretaria de Transporte y Vialidad.

[13] E. Verhoef. An integrated dynamic model of road traffic congestion based on simple car-following theory. `html://www.econ.vu.nl/vakgroep/re/members/everhoef/et.html`